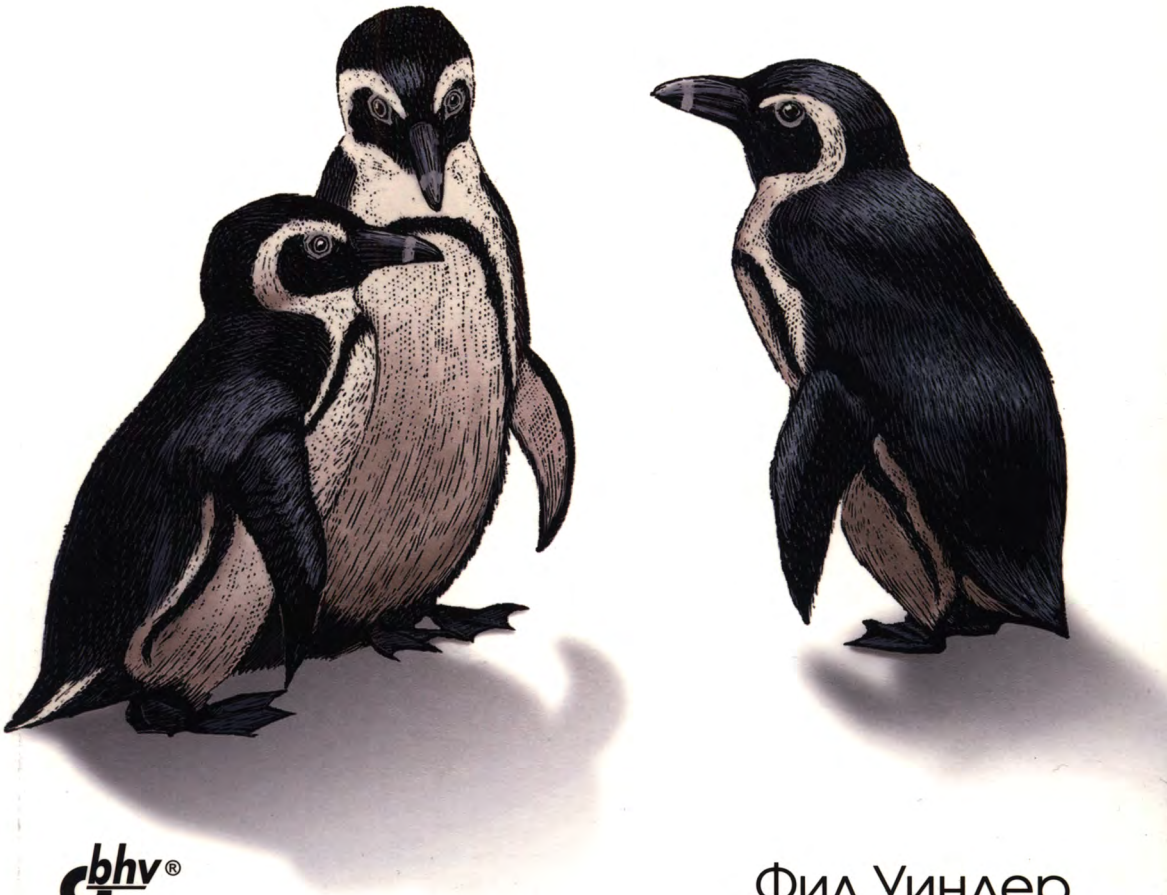
O'REILLY®

rescuer

# Обучение с подкреплением для реальных задач

## Инженерный подход



bhv®

Фил Уиндер

# Reinforcement Learning

## *Industrial Applications of Intelligent Agents*

*Phil Winder, Ph.D.*

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY

**Фил Уиндер**

# Обучение с подкреплением для реальных задач

## Инженерный подход

Санкт-Петербург
«БХВ-Петербург»
2023

УДК 004.43
ББК 32.973.26-018.1
У37

**Уиндер Ф.**

У37 Обучение с подкреплением для реальных задач: Пер. с англ. — СПб.: БХВ-Петербург, 2023. — 400 с.: ил.

ISBN 978-5-9775-6885-2

Книга посвящена промышленно-ориентированному применению обучения с подкреплением (Reinforcement Learning, RL). Объяснено, как обучать промышленные и научные системы решению любых пошаговых задач методом проб и ошибок — без подготовки узкоспециализированных учебных множеств данных и без риска переобучить или переусложнить алгоритм. Рассмотрены марковские процессы принятия решений, глубокие Q-сети, градиенты политик и их вычисление, методы устранения энтропии и многое другое. Данная книга — первая на русском языке, где теоретический базис RL и алгоритмы даны в прикладном, отраслевом ключе.

*Для аналитиков данных*
*и специалистов по искусственному интеллекту*

УДК 004.43
ББК 32.973.26-018.1

**Группа подготовки издания:**

| | |
|---|---|
| Руководитель проекта | *Олег Сивченко* |
| Зав. редакцией | *Людмила Гауль* |
| Перевод с английского | *Екатерины Черских* |
| Редактор | *Анна Кузьмина* |
| Компьютерная верстка | *Ольги Сергиенко* |
| Оформление обложки | *Зои Канторович* |

© 2022 BHV
Authorized Russian translation of the English edition of ***Reinforcement Learning*** ISBN 9781098114831
© 2021 Winder Research and Development Ltd
This translation is published and sold by permission of O'Reilly Media, Inc , which owns or controls all rights to publish and sell the same

Авторизованный перевод с английского языка на русский издания ***Reinforcement Learning*** ISBN 9781098114831
© 2021 Winder Research and Development Ltd
Перевод опубликован и продается с разрешения компании-правообладателя O'Reilly Media, Inc

Подписано в печать 29.07 22
Формат 70×100$^{1}/_{16}$ Печать офсетная. Усл. печ. л. 32,25
Тираж 1300 экз. Заказ № 5077
"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20

Отпечатано с готового оригинал-макета
ООО "Принт-М", 142300, М.О., г. Чехов, ул. Полиграфистов, д. 1

ISBN 978-1-098-11483-1 (англ.)
ISBN 978-5-9775-6885-2 (рус.)

© Winder Research and Development Ltd., 2021
© Перевод на русский язык, оформление
ООО "БХВ-Петербург", ООО "БХВ", 2023

# Оглавление

**Отзывы** ..... **15**

**Об авторе** ..... **19**

**Предисловие** ..... **21**
Цель ..... 21
Кому следует прочитать эту книгу? ..... 22
Руководящие принципы и стиль ..... 22
Предварительная подготовка ..... 24
Объем и план ..... 24
Дополнительные материалы ..... 25
Условные обозначения, используемые в этой книге ..... 26
Аббревиатуры ..... 26
Математические обозначения ..... 27

**Глава 1. Для чего нужно обучение с подкреплением?** ..... **29**
Почему сейчас? ..... 30
Машинное обучение ..... 31
Обучение с подкреплением ..... 32
Когда следует использовать обучение с подкреплением? ..... 33
Варианты применения обучения с подкреплением ..... 35
Таксономия подходов обучения с подкреплением ..... 37
Без модели или на основе модели ..... 37
Как агенты используют и обновляют свою стратегию ..... 38
Дискретные или непрерывные действия ..... 39
Методы оптимизации ..... 39
Оценка и улучшение политики ..... 40
Фундаментальные концепции обучения с подкреплением ..... 41
Первый RL-алгоритм ..... 41
Оценка ценности ..... 42
Ошибка предсказания ..... 43
Правило обновления веса ..... 43
RL — это то же самое, что ML? ..... 44
Награда и отклик ..... 45
Отложенные награды ..... 46
Ретроспектива ..... 46
Обучение с подкреплением как дисциплина ..... 47
Резюме ..... 49
Дополнительные материалы для чтения ..... 49
Использованные источники ..... 50

**Глава 2. Марковские процессы принятия решений, динамическое программирование и методы Монте-Карло ..... 53**
Алгоритм многорукого бандита ..... 53
Разработка наград ..... 53
Оценка стратегии: функция ценности ..... 54
Совершенствование политики: выбор лучшего действия ..... 57
Моделирование среды ..... 58
Запуск эксперимента ..... 59
Улучшение ε-жадного алгоритма ..... 61
Марковские процессы принятия решений ..... 62
Контроль запасов ..... 64
Таблица переходов ..... 65
Граф переходов ..... 66
Матрица переходов ..... 66
Симуляция управления запасами ..... 68
Политики и функции ценности ..... 70
Дисконтированные вознаграждения ..... 70
Прогнозирование вознаграждений с помощью функции ценности состояния ..... 71
Моделирование с использованием функции ценности состояния ..... 73
Прогнозирование вознаграждений с помощью функции ценности действия ..... 75
Оптимальные политики ..... 76
Генерирование политики Монте-Карло ..... 78
Итерация по ценности с динамическим программированием ..... 80
Реализация итерации по ценности ..... 82
Результаты итерации по ценнности ..... 84
Резюме ..... 85
Дополнительные материалы для чтения ..... 86
Использованные источники ..... 86

**Глава 3. Обучение с учетом временных различий, Q-обучение и *n*-шаговые алгоритмы ..... 87**
Обучение с учетом временных различий: формулировка подхода ..... 88
Q-обучение ..... 90
SARSA ..... 92
Q-обучение против SARSA ..... 93
Пример использования: автоматическое масштабирование контейнеров приложений для снижения затрат ..... 96
Отраслевой пример: торги рекламы в режиме реального времени ..... 98
Определение марковского процесса принятия решения ..... 98
Результаты торгов в режиме реального времени ..... 99
Дальнейшие улучшения ..... 101
Расширения для Q-обучения ..... 102
Двойное Q-обучение ..... 102
Отложенное Q-обучение ..... 103
Сравнение стандартного, двойного и отложенного Q-обучения ..... 103
Обучение с подкреплением на основе противодействия ..... 104
*n*-Шаговые алгоритмы ..... 105
*n*-Шаговые алгоритмы в распределенных средах ..... 108
Трассировки соответствия ..... 109

Расширения для трассировки соответствия ..... 112
Алгоритм обучения $Q(\lambda)$ Уоткинса ..... 112
Нечеткие стирания в алгоритме обучения Q(λ) Уоткинса ..... 113
Быстрое Q-обучение ..... 113
Накопление или замена трассировок соответствия ..... 113
Резюме ..... 114
Дополнительные материалы для чтения ..... 114
Использованные источники ..... 114

**Глава 4. Глубокие Q-сети ..... 117**
Архитектуры глубокого обучения ..... 118
Основные положения ..... 118
Архитектуры нейронных сетей ..... 119
Фреймворки глубокого обучения ..... 120
Глубокое обучение с подкреплением ..... 121
Глубокое Q-обучение ..... 122
Воспроизведение опыта ..... 122
Клоны Q-сети ..... 123
Архитектура нейронной сети ..... 123
Внедрение глубокой Q-сети ..... 124
Пример: глубокая Q-сеть в среде CartPole ..... 125
Зачем обучаться онлайн? ..... 127
Что лучше? Глубока Q-сеть против Q-обучения ..... 128
Практический пример: сокращение энергопотребления в зданиях ..... 128
Радужная DQN ..... 130
Распределительное RL ..... 130
Воспроизведение приоритетного опыта ..... 132
Зашумленные сети ..... 133
Дуэльные сети ..... 133
Пример: радужная глубокая Q-сеть в Atari Games ..... 134
Результаты ..... 134
Обсуждение ..... 136
Другие улучшения глубокой Q-сети ..... 138
Улучшение исследования ..... 138
Повышение вознаграждения ..... 139
Обучение на основе автономных данных ..... 140
Резюме ..... 142
Дополнительные материалы для чтения ..... 143
Использованные источники ..... 143

**Глава 5. Методы градиента политики ..... 145**
Преимущества прямого изучения политики ..... 145
Как рассчитать градиент политики ..... 146
Теорема о градиенте политики ..... 147
Функции политики ..... 149
Линейные политики ..... 150
Логистическая политика ..... 150
Политика softmax ..... 151
Произвольные политики ..... 152

Основные реализации ..... 152
Метод Монте-Карло (алгоритм REINFORCE) ..... 153
Пример: алгоритм REINFORCE в среде CartPole ..... 153
Алгоритм REINFORCE с базовыми показателями ..... 154
Пример: алгоритм REINFORCE с базовыми показателями в среде CartPole ..... 156
Уменьшение градиентной дисперсии ..... 158
*n*-Шаговый и улучшенный алгоритмы "актор — критик" ..... 159
Пример: *n*-шаговый алгоритм "актор — критик" в среде CartPole ..... 161
Темпы затухания ценностного обучения по сравнению с темпами ослабления политики ..... 163
Трассировки соответствия алгоритма "актор — критик" ..... 164
Пример: трассировка соответствия требованиям алгоритма "актор — критик" в среде CartPole ..... 165
Сравнение основных алгоритмов градиента политики ..... 166
Отраслевой пример: автоматическая продажа товаров клиентам ..... 166
Рабочее окружение: корзина заказов, написанная при помощи библиотеки Gym ..... 167
Ожидания ..... 168
Результаты из среды "Корзина покупок" ..... 169
Резюме ..... 172
Дополнительные материалы для чтения ..... 173
Использованные источники ..... 173

**Глава 6. Другие методы ..... 175**
Алгоритмы, действующие вне политик ..... 175
Выборка по значимости ..... 176
Поведенческие и целевые политики ..... 178
Q-обучение, действующее вне политики ..... 178
Градиентное обучение с учетом временных различий ..... 179
Жадный GQ-алгоритм ..... 180
Алгоритм "актор — критик" вне политики ..... 181
Детерминированные градиенты политики ..... 182
Обычные детерминированные градиенты политики ..... 182
Глубокие детерминированные градиенты политики ..... 184
Вывод DDPG ..... 184
Внедрение DSP ..... 185
Дважды отложенный DPG ..... 188
Отложенные обновления политики ..... 188
Ограниченное двойное Q-обучение ..... 189
Сглаживание целевой политики ..... 189
Реализация TD3 ..... 190
Практический пример: рекомендации на основе отзывов ..... 192
Улучшения DPG ..... 193
Методы доверительной области ..... 194
Дивергенция Кульбака — Лейблера ..... 196
Эксперименты по дивергенции Кульбака — Лейблера ..... 196
Естественные градиенты политики и оптимизация политики доверительной области ..... 197
Проксимальная оптимизация политики ..... 200
Усеченная цель PPO ..... 201
Ценностная функция PPO и цели разведки ..... 203

Пример: использование сервоприводов для Real-Life Reacher ..... 205
Описание эксперимента ..... 205
Реализация алгоритма RL ..... 206
Повышение сложности алгоритма ..... 208
Настройка гиперпараметров в моделировании ..... 209
Результирующие политики ..... 210
Другие алгоритмы градиента политики ..... 212
Алгоритм Retrace (λ) ..... 212
Алгоритм ACER ..... 212
Алгоритм ACKTR ..... 213
Эмпатические методы ..... 214
Расширения для алгоритмов градиента политики ..... 214
Квантильная регрессия в алгоритмах градиента политики ..... 215
Резюме ..... 215
Какой алгоритм следует использовать? ..... 215
Замечание об асинхронных методах ..... 216
Дополнительные материалы для чтения ..... 216
Использованные источники ..... 217

**Глава 7. Изучение всех возможных политик с помощью энтропийных методов ..... 221**
Что такое энтропия? ..... 221
Максимальная энтропия обучения с подкреплением ..... 222
Мягкий "актор — критик" ..... 223
Детали реализации SAC и дискретные пространства действий ..... 224
Автоматическая регулировка температуры ..... 224
Практический пример: автоматическое управление трафиком для сокращения очередей ..... 225
Расширения методов максимальной энтропии ..... 226
Другие меры энтропии (и ансамбли) ..... 226
Оптимистичное исследование с использованием верхней границы двойного Q-обучения ..... 227
Играем с воспроизведением опыта ..... 227
Мягкий градиент политики ..... 227
Мягкое Q-обучение (и производные) ..... 228
Обучение согласованности пути ..... 228
Сравнение производительности: SAC против PPO ..... 228
Как энтропия способствует исследованиям? ..... 230
Как температурный параметр влияет на исследование? ..... 233
Отраслевой пример: обучение вождению автомобиля с дистанционным управлением ..... 235
Описание задачи ..... 235
Минимизация времени обучения ..... 236
Выразительные действия ..... 238
Поиск гиперпараметров ..... 239
Финальная политика ..... 240
Дальнейшие улучшения ..... 240
Резюме ..... 241
Эквивалентность градиентов политики и мягкого Q-обучения ..... 242
Что это означает для будущего? ..... 242
Что это значит сейчас? ..... 242
Использованные источники ..... 243

**Глава 8. Улучшение процесса обучения агента ..... 245**
Переосмысление марковских процессов принятия решений ..... 246
Частично наблюдаемый марковский процесс принятия решений ..... 246
Предсказание доверительного состояния ..... 247
Практический пример: POMDP в автономных транспортных средствах ..... 248
Контекстные MDP ..... 249
MDP с изменяющимися действиями ..... 249
Регуляризованные MDP ..... 250
Иерархическое обучение с подкреплением ..... 250
Наивный HRL ..... 251
Высокоуровневые и низкоуровневые иерархии с внутренними наградами ..... 252
Навыки обучения и неконтролируемое RL ..... 254
Использование навыков в HRL ..... 255
Выводы HRL ..... 255
Мультиагентное обучение с подкреплением ..... 256
Фреймворки MARL ..... 257
Централизованное или децентрализованное ..... 259
Алгоритмы с одним агентом ..... 260
Практический пример: использование децентрализованного обучения с одним агентом в беспилотном летательном аппарате ..... 261
Централизованное обучение, децентрализованное выполнение ..... 262
Децентрализованное обучение ..... 263
Другие комбинации ..... 264
Проблемы MARL ..... 265
Выводы о MARL ..... 266
Экспертное руководство ..... 267
Клонирование поведения ..... 267
Имитационное RL ..... 267
Обратное RL ..... 268
Обучение по учебной программе ..... 270
Другие парадигмы ..... 271
Метаобучение ..... 271
Трансферное обучение ..... 272
Резюме ..... 273
Дополнительные материалы для чтения ..... 274
Использованные источники ..... 275

**Глава 9. Практическое обучение с подкреплением ..... 279**
Жизненный цикл проекта RL ..... 279
Определение жизненного цикла ..... 281
Жизненный цикл науки о данных ..... 281
Жизненный цикл обучения с подкреплением ..... 282
Определение проблемы: что такое проект RL? ..... 284
Проблемы с RL являются последовательными ..... 284
Проблемы RL имеют стратегический характер ..... 285
Низкоуровневые индикаторы RL ..... 286
Сущность ..... 286
Среда ..... 286
Состояние ..... 287

Действие ..... 287
Количественная оценка успеха или неудачи ..... 287
Типы обучения ..... 288
Онлайн-обучение ..... 288
Автономное или пакетное обучение ..... 288
Параллельное обучение ..... 290
Обучение без сброса ..... 291
Проектирование и доработка RL ..... 292
Процесс ..... 293
Инженерия среды ..... 293
Реализация ..... 294
Моделирование ..... 294
Взаимодействие с реальной жизнью ..... 295
Инжиниринг состояния или обучение представлениям ..... 296
Перспективные модели обучения ..... 297
Ограничения ..... 297
Преобразование (уменьшение размерности, автоэнкодеры и модели мира) ..... 298
Разработка политики ..... 299
Дискретные состояния ..... 300
Непрерывные состояния ..... 301
Преобразование в дискретные состояния ..... 303
Пространства смешанных состояний ..... 304
Сопоставление политик с пространствами действий ..... 305
Бинарные действия ..... 305
Непрерывные действия ..... 306
Гибридные пространства действий ..... 306
Когда выполнять действия ..... 307
Обширные пространства действий ..... 307
Исследование ..... 308
Является ли внутренняя мотивация исследованием? ..... 309
Количество посещений (выборка) ..... 310
Прирост информации (сюрприз) ..... 310
Прогноз состояния (любопытство или саморефлексия) ..... 311
Любопытные задачки ..... 311
Случайные вложения (сети случайной дистилляции) ..... 312
Расстояние до новизны (эпизодическое любопытство) ..... 313
Выводы по разведке ..... 313
Разработка вознаграждений ..... 314
Рекомендации по разработке вознаграждений ..... 315
Формирование вознаграждения ..... 316
Общие награды ..... 317
Выводы о вознаграждении ..... 318
Резюме ..... 318
Дополнительные материалы для чтения ..... 319
Использованные источники ..... 320

**Глава 10. Этапы в обучении с подкреплением ..... 325**
Реализация ..... 325
Фреймворки ..... 326
Фреймворки RL ..... 326
Другие фреймворки ..... 328

Масштабирование RL ..... 329
Распределенное обучение (Gorila) ..... 330
Обучение на одной машине (A3C, PAAC) ..... 331
Распределенное воспроизведение (Ape-X) ..... 333
Синхронное распределение (DD-PPO) ..... 333
Повышение эффективности использования (IMPALA, SEED) ..... 334
Масштабирование сделанных выводов ..... 336
Оценка ..... 337
Показатели эффективности политики ..... 338
Статистические сравнения политик ..... 340
Показатели производительности алгоритма ..... 343
Измерения производительности для конкретных задач ..... 343
Объяснимость ..... 344
Выводы оценки ..... 345
Развертывание ..... 346
Цели ..... 346
Цели на разных этапах развития ..... 346
Лучшие практики ..... 347
Иерархия потребностей ..... 348
Архитектура ..... 349
Вспомогательные инструменты ..... 351
Разработка против покупки ..... 352
Мониторинг ..... 352
Регистрация и отслеживание ..... 353
Непрерывная интеграция и непрерывная доставка ..... 353
Отслеживание экспериментов ..... 354
Настройка гиперпараметров ..... 355
Развертывание нескольких агентов ..... 355
Развертывание политик ..... 356
Безопасность, защита и этика ..... 357
Безопасное RL ..... 357
Защитное RL ..... 359
Этическое RL ..... 361
Резюме ..... 363
Дополнительные материалы для чтения ..... 364
Использованные источники ..... 365

**Глава 11. Выводы и будущее ..... 369**
Советы и рекомендации ..... 369
Формулирование задачи ..... 369
Ваши данные ..... 370
Тренировка ..... 371
Оценка ..... 372
Развертывание ..... 373
Отладка ..... 373
Алгоритм не может решить проблемы среды! ..... 375
Мониторинг для отладки ..... 376
Будущее обучения с подкреплением ..... 377
Рыночные возможности RL ..... 377
Будущее RL и направления исследований ..... 379

Исследования в промышленности ..... 379
Исследования в науке ..... 381
Этические стандарты ..... 383
Заключительные замечания ..... 384
Дальнейшие шаги ..... 384
Теперь ваша очередь ..... 385
Дополнительные материалы для чтения ..... 385
Использованные источники ..... 386

**Приложение 1. Градиент логистической политики для двух действий ..... 389**

**Приложение 2. Градиент политики softmax ..... 393**

**Предметный указатель ..... 395**

# Отзывы

"Обучение с подкреплением — одна из самых захватывающих областей машинного обучения и, к сожалению, также одна из самых сложных. Обучение с подкреплением отлично справляется с задачей выявления бэкграунда, ландшафта и возможностей использования этой новаторской техники способами, которые значительно улучшат возможности специалистов по обработке данных для своего бизнеса".

*Дэвид Арончик,*
*соучредитель Kubeflow*

"Книга доктора Фила Уиндера об обучении с подкреплением — это глоток свежего воздуха. Он превратил невероятно динамичную тему в простую для понимания книгу, которая фокусируется на идеях и понимании читателем. Эта книга интересна тем, насколько уместно обучение с подкреплением для обучения в неопределенной среде".

*Основатель Your Chief Scientist, инструктор и автор книги*
*"Вдумчивое машинное обучение"*

"Книга, необходимая для всех, кто хочет применить методы обучения с подкреплением к реальным задачам. Она ведет читателя от первых принципов к современному состоянию с множеством практических примеров и подробных объяснений".

*Дэвид Фостер,*
*партнер Applied Data Science Partners*
*и автор книги "Генеративное глубокое обучение"*

"Отличная книга Фила Уиндера. Проверенный природой метод обучения через действия наконец нашел свое место в стандартном наборе инструментов разработчика программного обеспечения. Обучение с подкреплением — это маховик искусственного интеллекта, и эта книга направлена на то, чтобы привнести эту перспективу в приложения в промышленности и бизнесе".

*Дэнни Ланге,*
*старший вице-президент по искусственному интеллекту, Unity*

*Для Эммы, Евы и Коры*

# Об авторе

**Доктор Фил Уиндер** (Dr. Phil Winder) — многопрофильный инженер-программист, специалист по обработке данных и генеральный директор Winder Research[1], консалтинговой компании в области облачных технологий. Он помогает стартапам и предприятиям улучшать свои процессы, основанные на данных, платформы и продукты. Фил специализируется на внедрении машинного обучения для облачных вычислений в производственной среде и был одним из первых сторонников движения MLOps.

Он восхитил тысячи инженеров своими учебными курсами по data science в государственных и частных организациях, а также на платформе онлайн-обучения O'Reilly. Курсы Фила посвящены использованию науки о данных в промышленности и охватывают широкий спектр актуальных, но практических тем, от очистки данных до глубокого обучения с подкреплением. Он регулярно выступает с докладами и активно участвует в сообществе специалистов по анализу данных.

Фил имеет докторскую степень и степень магистра. Он получил степень в электронной инженерии в Университете Халла; живет в Йоркшире (Великобритания) со своим пивоваренным оборудованием и семьей.

[1] См. **https://winderresearch.com/?.utm_source=oreilly&utm_medium.=book&utm_campaign=rl**.

# Предисловие

Обучение с подкреплением (reinforcement learning, RL) — это парадигма машинного обучения (machine learning, ML), которая способна оптимизировать последовательные решения. RL интересно тем, что имитирует то, как мы, люди, учимся. Мы инстинктивно способны изучать стратегии, которые помогают нам справляться со сложными задачами, такими как езда на велосипеде или сдача экзамена по математике. RL пытается скопировать этот процесс, взаимодействуя с окружающей средой для изучения стратегий.

В последнее время компании применяют алгоритмы машинного обучения для принятия единоразовых решений. Они обучаются на данных, чтобы принять лучшее на текущий момент решение. Однако часто правильное в настоящий момент решение может оказаться не лучшим решением в долгосрочной перспективе. Да, полная ванна мороженого осчастливит вас в краткосрочной перспективе, но на следующей неделе вам придется пропадать в тренажерном зале. Точно так же кликбейтные рекомендации могут давать самую высокую кликабельность, но в долгосрочной перспективе такие статьи воспринимаются как мошенничество и наносят ущерб долгосрочному вовлечению или удержанию читателя.

RL интересно тем, что позволяет изучить долгосрочные стратегии и применить их к сложным промышленным задачам. Как компании, так и специалисты-практики могут преследовать цели, которые напрямую связаны с бизнесом, такие как извлечение прибыли, наращивание количества пользователей и их удержание, а не технические показатели оценки, такие как точность или F-мера. Проще говоря, решение многих проблем зависит от последовательного принятия решений. ML не предназначено для решения этих проблем, RL — предназначено.

## Цель

Я написал эту книгу, потому что прочитал о стольких удивительных примерах использования RL для решения, казалось бы, невыполнимых задач. Правда, все эти примеры взяты из академических исследовательских работ, а книги, которые я впоследствии прочитал, были либо ориентированы на академические круги, либо представляли собой расхваленные автором листинги. Мало кто из авторов писал с промышленной точки зрения или объяснял, как использовать RL в производственных условиях. Я знал, насколько мощной может быть эта технология, поэтому решил написать книгу об использовании RL в промышленности.

Когда я только приступил к написанию, я хотел сосредоточиться на эксплуатационных аспектах, но быстро понял, что вряд ли кто-нибудь в отрасли слышал о RL, не говоря уже о том, чтобы использовать обечение с подкреплением в производстве. Кроме того, в ходе исследования моей аудитории я обнаружил, что многие инженеры и специалисты по обработке данных никогда даже не видели многих основополагающих алгоритмов. Таким образом, эта книга превратилась частично в фундаментальное объяснение, а частично — в практические советы по реализации. Я надеюсь, что эта книга вдохновит и подтолкнет к использованию RL в промышленной сфере.

Считаю, что это первая книга, в которой обсуждаются проблемы практического применения RL, и, безусловно, единственная книга, которая объединила алгоритмические и операционные разработки в целостную картину процесса разработки RL.

## Кому следует прочитать эту книгу?

Цель этой книги — продвигать использование RL в производственных системах. Если вы (сейчас или в будущем) создаете продукты в области RL, будь то исследования, разработки или прикладные вещи, то эта книга для вас. Это также означает, что я написал книгу, скорее, для практиков, чем для людей из академических кругов.

## Руководящие принципы и стиль

Я выбрал несколько руководящих принципов, которые считал важными для такой книги, основываясь на моем собственном опыте работы с другими книгами.

Во-первых, я полностью избегаю листингов. Я считаю, что в большинстве случаев книги не подходят для полотен кода (книги по разработке программного обеспечения являются очевидным исключением). Это противоречит общепринятому мнению, но лично мне надоело пролистывать страницы и страницы кода. Я покупаю книги, чтобы узнать мнение автора, то, как авторы объясняют концепции, идеи. Другая причина не печатать код заключается в том, что многие реализации, особенно в последующих главах, действительно довольно сложны, с большим количеством деталей оптимизации в реализации, которые отвлекают от основных идей, которыми я хочу поделиться. В любом случае вы обычно используете библиотечную реализацию. Кроме того, есть алгоритмы, которые еще не реализованы, потому что они слишком новы или слишком сложны для включения в стандартные библиотеки. Исходя из этих и других причин, предупреждаю, что это не типичная книга в жанре "покажи мне код".

Но не волнуйтесь, это не значит, что кода вообще нет. Есть, но он находится в сопутствующем репозитории вместе с множеством других практических примеров, практических руководств, обзоров, сборников статей и многих других материалов (*см. разд. "Дополнительные материалы" далее в предисловии*).

И это значит, что есть больше возможностей для понимания, объяснений и иногда нескольких неудачных шуток. Вы отстранитесь от чтения книги, оценив объем и плотность содержания, широту охвата и тот факт, что вам не приходилось пролистывать целые страницы кода.

Второй принцип, которого я придерживался, касался математики. RL — это в высшей степени математическая тема, потому что обычно намного проще объяснить алгоритм с помощью нескольких строк метаматематических выражений, чем двадцатью строками кода. Но я полностью осознаю, что математика иногда может казаться чужеродным языком. Как и любой другой язык программирования, математика имеет собственный синтаксис, предполагаемые знания и встроенные функции, которые вы должны знать, прежде чем сможете полностью оценить их.

Поэтому на протяжении всей этой книги я не уклоняюсь от математики, особенно при объяснении алгоритмов, фундаментальных для RL, потому что они являются важны сами по себе. Однако я стараюсь ограничить математику там, где могу, и давать длинные объяснения там, где не могу. Обычно я стараюсь следовать обозначениям, предоставленным Томасом и Окалом[1], — марковской нотацией процесса принятия решений, версией 1. Но я часто злоупотребляю обозначениями, чтобы сделать все еще проще.

Третий принцип, который может отличаться от других технических книг, в которых больше внимания уделяется передовым методам и инженерному искусству, связан с тем фактом, что разработка RL проводилась на основе исследований, а не практики. Так что эта книга полна ссылок на исследовательские работы. Я пытаюсь сопоставить и обобщить все эти исследования, чтобы дать вам общее представление о современном состоянии дел. Я также пытаюсь сбалансировать глубину изложения.

Как учителю, это действительно трудно сделать, потому что вы уже можете быть экспертом или вы можете быть полным новичком, который только что научился программировать. Я не могу угодить всем, но могу стремиться к золотой середине. В среднем, я надеюсь, вы почувствуете, что существует хороший баланс: вы получаете достаточное количество информации, чтобы чувствовать себя уверенно, но при этом изучаете материал с достаточным упрощением, чтобы не перегружаться. Если вы хотите углубиться в конкретные темы, обратитесь к исследовательским работам, справочным материалам и другим учебным книгам. Если вы чувствуете себя подавленным, не спешите. Я предоставил множество ссылок на дополнительные ресурсы, которые помогут вам на вашем пути.

Четвертый принцип заключается в том, что я всегда пытаюсь указать на подводные камни или сущности, которые могут пойти не так. Некоторые люди, с которыми я разговаривал, считают, что это означает, будто RL не готово или я не верю в него; оно готово, и я в это верю. Но жизненно важно понимать неизвестности и трудности, чтобы вы не переусердствовали и выделяли достаточно времени на выполнение работы. Это определенно не "нормальная" разработка программного обеспече-

---

[1] Thomas P. S., Okal D. A Notation for markov decision processes // ArXiv:1512.09075. — 2016. — September. — URL: **https://oreil.ly/VT7np**.

ния. Так что везде, где вы видите "проблемы" или объяснения "как улучшить", это существенная и важная информация. Неудача — лучший учитель.

## Предварительная подготовка

Все это означает, что RL — довольно сложная тема, еще до того, как вы к ней приступите. Чтобы читать эту книгу было максимально интересно, вам нужно немного познакомиться с наукой о данных (data science) и машинным обучением, и вам потребуются небольшие математические знания.

Но не волнуйтесь, если этого у вас нет. Вы всегда сможете наверстать позже. Я привожу много источников и ссылок для дальнейшего чтения и объясняю вспомогательные концепции там, где это имеет смысл. Обещаю, что вы все равно получите огромное количество знаний.

## Объем и план

Книга охватывает весь ваш путь по внедрению продуктов RL в производство. Во-первых, вам нужно изучить базовую структуру, на которой построено RL. Затем вы перейдете к простым алгоритмам, использующим эту парадигму. Тогда вы сможете узнать о все более совершенных алгоритмах, способных на большие подвиги. Затем вам нужно подумать о том, как применить эти знания к вашей отраслевой задаче. И наконец, вам необходимо разработать надежную систему, чтобы сделать ее жизнеспособной в эксплуатации.

Это путь изложенного в книге, и я рекомендую вам читать ее последвательно, от начала до конца. Последующие главы основываются на идеях первых глав, поэтому вы можете что-то пропустить, если не прочтете их. Однако не стесняйтесь переходить к конкретным главам или разделам, которые вас интересуют. При необходимости я возвращаюсь к предыдущим разделам.

Вот общее содержание, чтобы подогреть интерес.

- *Глава 1 "Для чего нужно обучение с подкреплением?".* Книга начинается с аккуратного введения в историю и основы RL, вдохновленные другими научными дисциплинами. Оно закладывает азы и дает обзор различных типов алгоритмов в RL.
- *Глава 2 "Марковские процессы принятия решений, динамическое программирование и методы Монте-Карло".* Более сложный материал начинается с главы, в которой определяются фундаментальные концепции RL, включая марковские процессы принятия решений, динамическое программирование и методы Монте-Карло.
- *Глава 3 "Обучение с учетом временных различий, Q-обучение и n-шаговые алгоритмы".* В этой главе вы перейдете к так называемым методам оценки, которые призваны количественно охарактеризовать ценность пребывания в определенном состоянии, базовый алгоритм, который доминирует во всех современных системах RL.

- *Глава 4 "Глубокие Q-сети"*. Бóльшая часть недавнего ажиотажа была вызвана сочетанием методов оценки и глубокого обучения. Вы подробно изучите это сочетание, и я обещаю, что вы будете удивлены производительностью этих алгоритмов.
- *Глава 5 "Методы градиента политики"*. Теперь вы узнаете о второй по популярности форме алгоритмов RL — методах градиента политики — которые призваны натолкнуть вас на параметризованную стратегию к повышению производительности. Основное преимущество состоит в том, что они могут справляться с непрерывными действиями.
- *Глава 6 "Другие методы"*. У базовых алгоритмов градиента политики имеется ряд проблем, но в этой главе рассматриваются и исправляются многие недостатки, от которых они страдают. И для повышения эффективности вводится перспективное обучение вне политики.
- *Глава 7 "Изучение всех возможных политик с помощью энтропийных методов"*. Методы энтропии показали высокую надежность и позволяют вырабатывать стратегии для сложных действий, таких как вождение автомобиля или управление транспортным потоком.
- *Глава 8 "Улучшение процесса обучения агента"*. Отступив от основных алгоритмов RL, в этой главе я расскажу, как вспомогательные компоненты могут помочь в решении сложных проблем. Здесь я сосредоточусь на различных парадигмах RL и альтернативных способах формулирования марковского процесса принятия решений.
- *Глава 9 "Практическое обучение с подкреплением"*. Это первая из двух глав, посвященных созданию производственных RL-систем. В данной главе вы познакомитесь с процессом разработки и реализации промышленных алгоритмов RL. В ней описывается процесс, проектные решения и практические аспекты реализации.
- *Глава 10 "Этапы в обучении с подкреплением"*. Если вам нужен совет о том, как использовать продукты RL в производственной среде, эта глава для вас. Здесь я углубляюсь в архитектурный проект, который вам следует рассмотреть, чтобы сделать ваше решение масштабируемым и более надежным, а затем подробно описываю ключевые аспекты, на которые вам нужно обратить внимание.
- *Глава 11 "Выводы и будущее"*. Последняя глава — это не просто резюме вышеизложенного. Она содержит множество практических советов и приемов, которые вы найдете полезными во время вашего путешествия по RL, в ней также представлены предложения для будущих исследований.

# Дополнительные материалы

Я создал веб-сайт **https://rl-book.com**, чтобы систематизировать на нем все дополнительные материалы, сопровождающие эту книгу. Здесь вы найдете сопроводительный код, подробные статьи и рабочие таблицы, сравнения и обзоры технологии RL, базы данных текущих тематических исследований RL и многое другое.

*См. разд. "Руководящие принципы и стиль" ранее в предисловии*, чтобы узнать, почему в этой книге не напечатан код.

Причина создания целого веб-сайта, а не просто репозитория кода, заключалась в том, что я считаю, что RL — это больше чем просто код. Это меняющий парадигму способ мышления о том, как решения могут иметь долгосрочные последствия. Это новый набор технологий, и для него нужна совершенно другая архитектура. По всем этим и другим причинам дополнительная информация не помещается в репозиторий. Она не подходит для печати, потому что может быстро меняться или становится просто неэффективной. Итак, я создал эту экосистему и уверен, что вы найдете ее ценной. Обязательно просмотрите ее, и если чего-то не хватает, дайте мне знать.

# Условные обозначения, используемые в этой книге

В этой книге используются следующие типографские условные обозначения:

- *курсив* указывает на новые термины и иногда используется для выделения;
- **полужирный шрифт** — URL-адреса, адреса электронной почты;
- рубленый шрифт используется для обозначения определенных классов или сред RL, обозначения элементов программ, таких как переменные или имена функций, базы данных, типы данных, переменные среды, инструкции и ключевые слова.

Данный элемент обозначает подсказку или совет.

Данный элемент обозначает общее замечание.

Данный элемент обозначает предупреждение или предостережение.

## Аббревиатуры

В обучении с подкреплением много аббревиатур, особенно когда речь идет о реализации алгоритмов.

## Математические обозначения

В целом я предпочитаю использовать марковскую нотацию процесса принятия решений Томаса и Окала, версию 1. Однако я попытался еще больше упростить ее, убрав такие формальности, как разграничение по времени, и расширив использование апострофа для обозначения текущего и следующего моментов. Во всей математической строгости эти концепции изложены в академических учебниках и статьях.

Как правило, фигурные буквы обозначают множество, а строчные буквы — элемент множества. Апостроф обозначает следующий временной шаг. Прописные буквы представляют функцию или константу.

Я отказываюсь от формальности выборки определенного состояния из случайной переменной и вместо этого использую конкретную реализацию переменной, например $s$, чтобы облегчить читаемость уравнений. В литературе вы обычно встретите заглавные буквы, представляющие стохастические переменные.

Некоторые алгоритмы насыщены индексами, и это означает, что вам нужно буферизовать данные и обращаться к определенным точкам в получившемся буфере. Когда я получаю эти алгоритмы, мне приходится прибегать к использованию нижних индексов; например $a_t$ будет означать действие в какой-то момент времени или позиции $t$.

Если вы не привыкли читать уравнения, делайте это медленно. Сначала посмотрите, чтобы понять, что представлено каждым символом, а затем определите, что в этом уравнении делается. Как и в случае с любым навыком, чем больше раз вы будете выполнять его, тем проще станет для вас процесс выполнения. Знак "точка равно" $\doteq$ можно читать как "определяется как".

Изучая алгоритмы, прочтите их процедурно. Где возможно, я использую текст, но во многих случаях уравнение оказывается более кратким. Символ $\leftarrow$ в алгоритмах читается как "обновить"; это предпочтительнее, потому что технически знак равенства означает математическое равенство, например == в вашем программном обеспечении. Большинство языков программирования злоупотребляют этой нотацией и используют символ равенства для обозначения как определения, так и обновления.

Я решил представить алгоритмы в академическом псевдокоде, а не в стиле, принятом в программной инженерии. Я долго думал об этом, но в итоге есть три основные причины для этого решения. Во-первых, так алгоритмы представлены во всей академической литературе. Я хотел, чтобы эта книга стала мостиком между промышленным и академическим сообществом, и я думаю, что наличие еще одного представления усилит этот разрыв. Во-вторых, эти алгоритмы более лаконичны в академической форме, такова математика. Если бы вам пришлось преобразовать математику в программный псевдокод, это привело бы к огромному количеству циклов for и временных переменных. Последняя причина в том, что ошибиться было бы слишком легко. Несмотря на то что я упростил математику, псевдокод представляет собой реальную реализацию. Преобразование реализаций, представленных в академических статьях, в программный псевдокод привело бы к слишком большому количеству ошибок.

ГЛАВА 1

# Для чего нужно обучение с подкреплением?

Как люди учатся? Этот обманчиво простой вопрос сбивал с толку мыслителей на протяжении тысячелетий. Греческий философ Платон и его ученик Аристотель задались вопросом: находятся ли истина и знание внутри нас (рационализм) или они пережиты (эмпиризм)? Даже сегодня, 2500 лет спустя, люди все еще пытаются ответить на этот вечный вопрос.

Если бы люди уже всё знали, им не нужно было бы больше приобретать жизненный опыт. Люди могли бы проводить остаток своего земного времени, улучшая жизнь, принимая правильные решения и размышляя над такими важными вопросами, как "где мои ключи?" и "я запер входную дверь?". Но как люди вообще получают эти знания? Вы можете научить знанию. А более высокий уровень среднего образования ведет к лучшему обществу. Но всему нельзя научить. И на уроках, и в жизни ученик должен *переживать*.

Маленькие дети вдохновляют в этом отношении. Им нужно испробовать ряд ситуаций и результатов. В долгосрочной перспективе они начинают искать полезный опыт и избегать пагубного (хочется надеяться). Они активно принимают решения и оценивают результаты. Но жизнь ребенка загадочна, и награды часто вводят в заблуждение. Немедленная награда за то, что ребенок залезет в шкаф и съест печенье, велика, но наказание будет суровее.

Обучение с *подкреплением* объединяет две задачи. Первая — это исследование новых ситуаций. Вторая — использование этого опыта для принятия более качественных решений. Со временем так формируется план достижения цели. Например, ребенок учится ходить, вставая, наклоняясь вперед и падая в объятия любящего родителя. Но это только после многих часов хождения, держась за руки, шатания и падений. В конце концов, мышцы ног ребенка начинают работать слаженно, используя многоступенчатую стратегию, которая сообщает, что и когда нужно делать. Вы не можете вложить в голову ребенку решение всех жизненных ситуаций, которое когда-либо ему понадобится, поэтому вместо этого жизнь предоставляет ребенку основу, на которой можно учиться.

В этой книге показано, как реализовать процесс подкрепления для компьютера. Но зачем это делать? Это позволяет машине учиться самостоятельно. Перефразируя любимое телешоу моей жены, скажу: вы даете машине возможность искать новые впечатления, смело идти туда, куда раньше не ступала никакая машина.

В этой главе представлено введение в обучение с подкреплением (я откладываю формальное определение обучения с подкреплением до *главы 2*). Во-первых,

я опишу, зачем это искусство нужно инженерам и почему именно сейчас. К концу главы вы узнаете, в каких отраслях можно использовать обучение с подкреплением, и разработаете свою первую математическую модель. Также будет дан обзор тех типов алгоритмов, с которыми вы встретитесь позже в этой книге.

В этой книге я употребляю слово "инженер", чтобы абстрактно говорить обо всех, кто использует свои навыки для разработки решения проблемы. Я имею в виду инженеров-программистов, инженеров по обработке данных, специалистов по данным, исследователей и т. д.

## Почему сейчас?

Две причины обусловили необходимость и способность выполнять обучение с подкреплением: доступ к большим объемам данных и возросшая скорость обработки данных.

Вплоть до 2000 г. человеческие знания хранились на аналоговых устройствах, таких как книги, газеты и магнитные ленты. Если бы вы сжали эти знания, то в 1993 г. вам потребовалось бы 15,8 эксабайт пространства (один эксабайт равен одному миллиарду гигабайт) [1]. В 2018 г. этот показатель увеличился до 33 зеттабайт (1 Збайт = 1 000 000 000 000 Гбайт). Поставщикам облачных услуг даже приходится прибегать к использованию жестких дисков размером с контейнер для загрузки больших объемов данных [2].

Вам также потребуются необходимые вычислительные мощности для анализа всех этих данных. В качестве демонстрации давайте рассмотрим случай одной из самых ранних реализаций обучения с подкреплением.

В 1947 г. Дитрих Принц (Dietrich Prinz) работал на компанию Ferranti в Манчестере (Великобритания). Там он помог спроектировать и сконструировать первую производственную версию манчестерского компьютера под названием Ferranti Mark 1 [3]. Он научился программировать Mark 1 под руководством Алана Тьюринга (Alan Turing) и Сисели Попплуэлл (Cicely Popplewell). Под влиянием статьи Тьюринга на эту тему в 1952 г. Принц выпустил шахматную программу, которая могла решать единственный набор задач под названием "matein-2". Это шахматные композиции, в которых игрок выбирает два хода, приводящих к мату в шахматах. Алгоритм Принца тщательно перебирал все возможные позиции и вырабатывал решение в среднем за 15–20 минут. Это реализация алгоритма Монте-Карло, описанного в *главе 2*. Принц стал рассматривать шахматное программирование как "ключ к методам, которые можно использовать для решения структурных или логистических задач в других областях с помощью электронных компьютеров". Он был прав [4].

Одновременное наращивание объема данных и вычислительной мощности аппаратного обеспечения привело к тому, что примерно в 2010 г. стало возможным и необходимым учить машины обучаться.

# Машинное обучение

Полное описание машинного обучения выходит за рамки этой книги. Но на машинном обучении основано обучение с подкреплением. Прочтите как можно больше о машинном обучении, особенно о книгах, которые я рекомендую в *разд. "Дополнительные материалы для чтения " в конце этой главы.*

Повсеместное распространение данных и доступность дешевых высокопроизводительных вычислений позволили исследователям пересмотреть алгоритмы 1950-х годов. Они выбрали название *"машинное обучение"* (machine learning, ML), но такое название не вполне удачно, потому что ML одновременно считается и дисциплиной, и набором методов. Я считаю машинное обучение детищем *науки о данных* (data science), которая представляет собой всеобъемлющую научную область, изучающую данные, генерируемые явлениями. Мне не нравится термин *"искусственный интеллект"* (ИИ — artificial intelligence, AI) по той же причине; достаточно сложно определить, что такое интеллект, не говоря уже о том, как он воплощается.

ML начинается с большого количества информации в виде данных, полученных в ходе *наблюдений*. Наблюдение представляет собой набор атрибутов в единой точке, которые описывают сущность. Например, в избирательном опросе одно наблюдение представляет собой предполагаемый голос одного человека. Для задачи формулирования рекомендаций наблюдением может быть щелчок по определенному продукту. Инженеры используют ML-алгоритмы для интерпретации этой информации и принятия решений.

При *обучении с учителем метки* представляют ответ на проблему для конкретного наблюдения. Здесь алгоритм пытается использовать информацию, чтобы угадать правильный результат. *Обучение без учителя* работает без меток, и вы принимаете решения на основе характеристик данных. Я всегда рекомендую своим клиентам из Winder Research стремиться к контролируемому обучению — например, путем оплаты или проведения экспериментов для поиска меток, — потому что, если у вас нет основополагающей истины, вам будет сложно количественно оценить эффективность.

Процесс поиска алгоритма решения задачи называется *моделированием*. Инженеры проектируют модели для упрощения и представления основных явлений. Они используют модель, чтобы делать обоснованные предположения о новых наблюдениях. Например, модель может сказать вам, что новый клиент предоставил ложную информацию в своем приложении, или может преобразовать вашу речь в текст.

Учитывая эти описания, попробуйте научить ребенка кататься на велосипеде. Как лучше всего это сделать? Согласно парадигме ML вы должны разметить множество наблюдений. Вы можете посоветовать своему ребенку посмотреть видео с профессиональными велосипедистами. Как только он просмотрит достаточное количество видеороликов, вы, игнорируя любые его протесты о том, что ему было скучно, можете проверить его способности в соответствии с некоторыми произвольными техническими критериями успеха. Думаете, это сработает? Нет.

Несмотря на то что ML принципиально подходит для многих прикладных задач, некоторые проблемы не поддаются машинному обучению. Лучшее решение, продолжая предыдущий пример, — позволить своему ребенку попробовать самостоятельно прокатиться. Некоторые его попытки ничем не увенчаются. В других случаях у него что-то получится. Каждое решение будет сказываться на его представлении о задаче. После достаточного количества попыток и определенных наставлений он изучит стратегии, позволяющие максимизировать собственное определение успеха. Вот в чем обучение с подкреплением превосходит обучение с учителем.

# Обучение с подкреплением

*Обучение с подкреплением* (reinforcement learning, RL) поясняет, как принимать наилучшие решения последовательно, в определенном контексте, чтобы максимизировать реальный показатель успеха. Лицо, принимающее решения, узнает об этом методом проб и ошибок. Ему не говорят, какие именно решения принимать, вместо этого он должен учиться самостоятельно, методом проб и ошибок. На рис. 1.1 представлены четыре компонента RL, в *главе 2* мы углубимся в подробности.

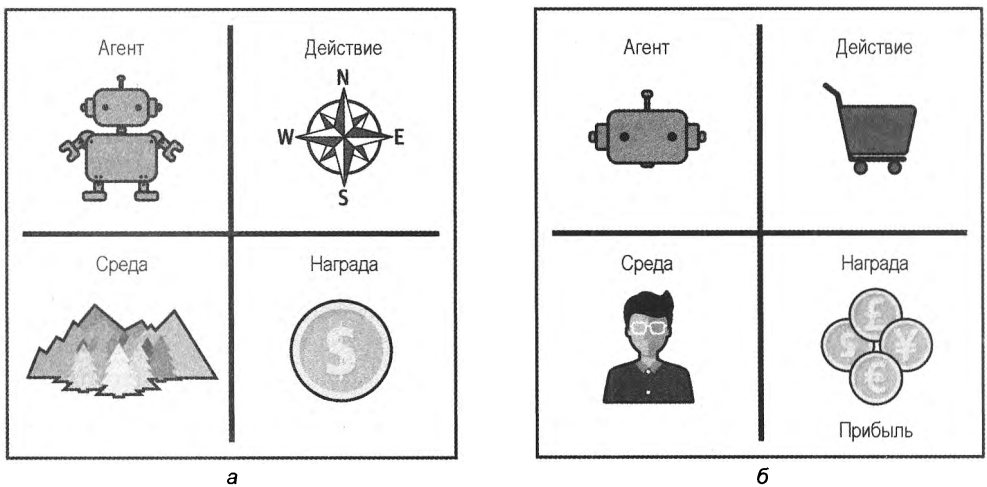



**Рис. 1.1.** Набросок четырех компонентов, необходимых для RL: *агента*, который совершает *действия* в *окружающей среде* для наибольшего *вознаграждения*. Пример (*а*) демонстрирует робота, который намеревается пройти через лабиринт, чтобы получить монету. Пример (*б*) показывает приложение для электронной коммерции, которое автоматически добавляет товары в корзины пользователей, чтобы максимизировать прибыль

Каждое решение — это *действие*. Например, когда вы едете на велосипеде, действиями являются рулевое управление, кручение педалей и торможение. Если вы пытаетесь автоматически добавлять товары в корзину, то такими действиями являются решения о добавлении определенных товаров.

Контекст, хотя он может отражать любую реальную ситуацию, часто ограничен, что не позволяет сделать проблему разрешимой. Практики RL позволяют подготовить своеобразный интерфейс взаимодействия с *окружающей средой*. Это может быть симуляция, реальная жизнь или их комбинация. Окружающая среда принимает действия и отвечает на них результатом и новым набором наблюдений.

*Агент* — это субъект, который принимает решения. Это может быть ваш ребенок, какая-нибудь программа или, например, робот.

*Вознаграждение* кодирует вызов. Этот механизм обратной связи сообщает агенту, какие действия привели к успеху (или неудаче).

Сигнал вознаграждения обычно числовой, но нужен только для подкрепления поведения; например, стратегии генетического обучения могут удалять неэффективных агентов и не предоставлять никакого вознаграждения.

Вот еще пример: вы можете вознаградить робота за достижение цели или агента за добавление нужного продукта в корзину. Все просто, правда? Но что делать, если роботу требуется три дня, чтобы выйти из простого лабиринта, потому что он проводит бóльшую часть времени, нарезая круги? А если агент начнет добавлять все подряд товары в корзину?

Такие процессы происходят и в мире животных. Они должны максимально увеличить свои шансы на выживание, чтобы передать свои гены потомству. Например, как и большинству травоядных, лосям нужно много есть, чтобы выжить. Но в 2011 г. в окрестностях Гётеборга (Швеция) нашли лося, застрявшего в ветвях дерева после того, как он наелся ферментированных яблок [5]. Система "вознаграждения" лося, которая вызывает голод, дала сбой, потому что цель ее слишком лишена конкретики. Нельзя есть все подряд, чтобы максимизировать свои шансы на выживание. Все гораздо сложнее.

Эти примеры подводят нас к главной проблеме в RL, которая известна с тех пор, как Ада Лавлейс (Ada Lovelace) впервые написала алгоритм для получения чисел Бернулли. Как сказать машине, что она должна делать? Агенты RL часто остаются крайними, потому что они оптимизируются не для того, что на самом деле нужно. И пока я рекомендую вам максимально не усложнять награду. Многие задачи предполагают естественную награду. В *главе 9* эта проблема обсуждается более подробно.

Итак, четыре компонента образуют *марковский процесс принятия решений* (Markov decision process, MDP). MDP используют для того, чтобы сформулировать задачи, даже не связанные с инженерией. В *главе 2* эти идеи представлены более подробно.

## Когда следует использовать обучение с подкреплением?

Некоторые примеры RL, которые вы найдете в Интернете, выглядят вымученными. Их авторы берут пример ML и пытаются применить к нему RL, несмотря на отсутствие четкого агента или действия. Посмотрите, например, несколько примеров с попытками включить RL в прогнозирование фондового рынка. Существует воз-

можность использования автоматизированного агента для совершения сделок, но во многих примерах это не главное; основное внимание по-прежнему уделяется прогнозной модели. Это неуместно, и такие примеры лучше оставить для ML.

RL работает лучше всего, когда решения принимаются последовательно, а действия связаны с исследованием окружающей среды. Возьмите робототехнику, это классическая область применения RL. Цель робота — научиться выполнять неизвестные задачи. Вы не должны указывать роботу, как добиться успеха, потому что это либо слишком сложно (допустим, вы просите робота построить дом), либо вы можете быть предвзяты в силу собственного опыта (вы не робот), поэтому вы не знаете, как поставить себя на место робота. Если вместо этого вы позволите роботу провести исследование, он сможет найти оптимальное решение. Этот случай хорошо подходит для RL.

Всегда выбирайте самое простое решение, которое удовлетворительно решает вашу прямую задачу.

Основное преимущество RL заключается в том, что такое обучение оптимизируется для получения долгосрочных многоэтапных вознаграждений. Второстепенное преимущество состоит в том, что очень легко включить в процесс метрики, используемые бизнесом. Например, рекламные решения обычно оптимизированы в целях обеспечения наилучшей кликабельности для отдельной рекламы. Это неоптимально, потому что зрители часто видят несколько рекламных объявлений, а цель не щелчок (клик), а нечто большее, например удержание потребителя, регистрация или покупка. Комбинация показываемых рекламных объявлений (в определенном порядке и с конкретным содержанием) может быть автоматически оптимизирована RL с помощью простой в использовании цели, соответствующей потребностям бизнеса.

Вы можете отказаться от некоторых из четырех компонентов, представленных в предыдущем разделе, чтобы упростить разработку. Если в вашей модели нет естественного сигнала, свидетельствующего о вознаграждении, например, "робот достиг цели", то можно создать искусственное вознаграждение. Также часто создают симуляцию окружающей среды. Вы можете квантовать или обрывать действия. Но это все компромиссы. Симуляция никогда не заменит реальный жизненный опыт.

В RL активно ищется оптимальная модель. Вам не нужно создавать случайную выборку и подстраивать ее в автономном режиме. Быстрое онлайн-обучение может творить чудеса, когда важно в самые сжатые сроки добиться максимальной производительности. Например, в A/B-тестах, ориентированных на прибыль, когда нужно решить, какой маркетинговый текст использовать, не хочется тратить время на подготовку случайной выборки, если какой-то вариант недостаточно эффективен. RL делает это бесплатно. О том, как A/B-тестирование соотносится с RL, вы можете узнать в *главе 2*.

Таким образом, RL лучше всего подходит для прикладных задач, которые требуют последовательных, сложных решений и имеют долгосрочную цель (в контексте единственного решения). ML может помочь вам в качестве вспомогательного инструмента, но RL лучше всего подходит для сред с прямой обратной связью. Утверждаю это, поскольку я разговаривал с некоторыми практиками, которые использовали RL для замены групп специалистов по обработке данных, настраивающих производительность решений машинного обучения.

## Варианты применения обучения с подкреплением

В этой книге я привожу целый спектр примеров по двум причинам. Во-первых, я хочу проиллюстрировать теоретические аспекты, например, как работают алгоритмы. Эти примеры просты и абстрактны. Лично я считаю, что просмотр примеров помогает мне учиться. Я также рекомендую вам воспроизвести примеры, это поможет вам в обучении. Во-вторых, я хочу показать, как использовать RL в промышленности.

В СМИ, как правило, наибольшее внимание уделяется примерам, демонстрирующим, как агенты побеждают людей в играх. Журналистам нравятся броские истории о том, как люди сдают свои позиции. А ученые продолжают обращаться к играм из-за сложной моделируемой среды. Но я решил не говорить ни о DeepMind AlphaGo Zero, ни о версии агента, победившего чемпиона мира по го, ни об OpenAI Five, победившей чемпионов мира по Dota 2, а вместо этого сосредоточиться на приложениях и примерах из самых разных промышленных отраслей. Я не говорю, что игровые примеры — пустая трата времени. Игровые компании могут использовать RL для многих практических целей, например для помощи в тестировании или оптимизации внутриигровых вариантов "AI" для максимизации дохода. Мне хочется помочь вам абстрагироваться от хайпа и показать разнообразные области, где применимо RL. Для того чтобы продемонстрировать, что именно возможно уже сейчас, я представляю широкий выбор экспериментов, которые лично мне кажутся интересными.

- Область робототехники имеет множество приложений RL, включая улучшение движения и производственного процесса, игру в бильбоке и переворачивание блинов [6]. Автономные транспортные средства также являются темой активных исследований [7].
- Вы можете использовать RL для улучшения облачных вычислений. В одной статье рассказано, как оптимизируются приложения с учетом задержки [8], в другой обсуждается соотношение "энергоэффективность/использование" [9]. Охлаждение центра обработки данных, охлаждение процессора и сетевая маршрутизация — все это варианты применения RL, используемые сегодня [10–12].
- Финансовая отрасль применяет RL для совершения сделок и распределения портфеля [13, 14]. Также существует значительный интерес к оптимизации ценообразования в режиме реального времени [15].
- Количество энергии, потребляемой при коммунальном обслуживании (через отопление, воду, свет и т. д.), может быть значительно уменьшено с помощью

RL [16]. А электрические сети могут использовать RL для решения ситуаций, когда спрос неоднороден; дома являются одновременно производителями и потребителями [17].

- RL улучшает управление светофорами и активное управление полосами движения [18, 19]. Умные города также остаются в выигрыше [20].
- Недавние статьи предлагают множество вариантов применения RL в здравоохранении, особенно в областях дозирования и составления схем лечения [21, 22]. RL можно использовать для разработки более совершенных протезов и протезных контроллеров [23].
- Система образования и электронное обучение могут выиграть благодаря прицельно подобранным учебным программам на основе RL [24].

Ни один бизнес-сектор не остался незатронутым: игры, технологии, транспорт, финансы, наука и окружающая среда, промышленность, производство и государственные службы — все они ссылались на приложения RL.

Я не хочу терять вас в бесконечном списке, поэтому вместо этого я отсылаю вас на соответствующий веб-сайт[1], где у меня есть полный каталог приложений RL.

Любая технология опасна в шаловливых руках. И, помня о популистских аргументах против AI, можно интерпретировать RL как опасное явление. Прошу вас, как инженер, как человек, подумать о том, что вы строите. Прикиньте, как это повлияет на других людей? Какие есть риски? Это противоречит вашей морали? Будьте ответственны за свою работу перед собой. Если вы не можете этого сделать, вам, вероятно, не следует этим заниматься. Далее приведены еще три задокументированных гнусных приложения. У каждого свои этические границы. Где ваша граница? Какие приложения вам подходят?

- Pwnagotchi — это устройство на базе RL, которое активно сканирует, анализирует и взламывает Wi-Fi-сети с WPA/WPA2-защитой путем дешифрования рукопожатий [25].
- Исследователи показали, что можно обучить агентов обходить статические модели вредоносных программ в антивирусных сканерах [26].
- Военное ведомство США разрабатывает модели боевых действий, чтобы продемонстрировать, как автономные роботы могут помочь на поле боя [27].

Я более подробно обсуждаю вопросы безопасности и этики в *главе 10*.

[1] См. **https://rl-book.com/applications/?utm_source=oreilly&utm_medium=book&utm_campaign=rl**.

# Таксономия подходов обучения с подкреплением

В ходе разработки RL сформировалось несколько тем. Вы можете использовать их для группировки алгоритмов по признаку сходства. В этой книге подробно описаны многие из этих алгоритмов, но сейчас я приведу их беглый обзор.

## Без модели или на основе модели

Первое важное решение, которое вы должны принять, — определить, есть ли у вас точная модель окружающей среды. *Алгоритмы на основе моделей* используют точные сведения об окружающей среде, в которой они работают, для улучшения обучения. Например, настольные игры часто ограничивают количество ходов, которые вы можете сделать, и вы можете использовать эти знания, чтобы, во-первых, ограничить алгоритм так, чтобы он не выполнял недопустимые действия, и во-вторых, улучшить производительность за счет прогнозирования во времени (например, если я двинусь сюда, а противник двинется туда, я могу выиграть). В таких играх, как го и покер, можно использовать алгоритмы, позволяющие опередить людей благодаря фиксированным правилам игры. Вы и ваш противник можете сделать ограниченный набор ходов. Это лимитирует количество стратегий, которые алгоритмы должны искать. Как и в экспертных системах, решения на основе моделей обучаются эффективно, потому что они не тратят время на поиск неправильных путей [28, 29].

Теоретически *безмодельные алгоритмы* применимы к любой задаче. Они изучают стратегии через взаимодействие, усваивая при этом любые правила окружающей среды.

Однако это еще не все. Некоторые алгоритмы могут изучать модели окружающей среды одновременно с изучением оптимальных стратегий. Несколько новых алгоритмов также могут использовать потенциальные, но неизвестные действия других агентов (или других игроков). Другими словами, эти агенты способны научиться противодействовать стратегиям другого агента.

Алгоритмы, подобные этим, имеют тенденцию стирать границу между подходами, основанными на моделях и свободными от моделей, потому что в конечном счете вам понадобится модель окружающей среды. Разница в том, сможете ли вы статистически определить это, сможете ли вы изучить или сможете ли вы перенять модель из стратегии.

В этой книге я делаю упор на безмодельные алгоритмы, потому что они применимы к любой промышленной задаче. Но в ситуации, когда ваша среда имеет строгие статические правила, подумайте о разработке индивидуального RL-алгоритма на основе модели, который сможет воспользоваться этим преимуществом.

## Как агенты используют и обновляют свою стратегию

Цель любого агента — изучить стратегию, максимизирующую вознаграждение. Я использую слово *"стратегия"*, потому что это слово легче понять, но правильный термин *"политика"*. В *главе 2* политики рассматриваются более подробно.

То, как и когда алгоритм обновляет стратегию, является определяющим фактором между большинством безмодельных алгоритмов RL. Есть две ключевые формы стратегии, которые определяют производительность и функциональность агента, но их очень легко спутать.

Во-первых, разница между обновлениями стратегии онлайн и офлайн. Онлайн-агенты улучшают свои стратегии, используя только данные, которые они только что наблюдали, а затем немедленно избавляются от них. Они не хранят и не используют повторно старые данные. Всем агентам RL необходимо в некоторой степени обновить свою стратегию, когда они сталкиваются с новым опытом, но большинство современных алгоритмов написаны так, что сохранение и повторное использование прошлого опыта в них целесообразно.

Автономные агенты могут учиться на офлайн-множествах данных или старых журнальных файлах (log-файлах). Это может быть весьма кстати, потому что иногда сложно или дорого взаимодействовать с реальным миром. Однако, как правило, RL наиболее полезно, когда агенты обучаются онлайн, поэтому большинство алгоритмов нацелены на сочетание онлайнового и офлайнового обучения.

Второе, порой тонкое отличие зависит от того, как агенты выбирают действие, определяемое их стратегией. Агенты *политики* учатся предсказывать награду за пребывание в определенных состояниях после выбора действий в соответствии с текущей стратегией. Агенты *вне политики* учатся предсказывать награду после выбора *любого* действия.

Я понимаю, что эту тонкость трудно понять, поэтому позвольте мне продемонстрировать небольшой пример. Представьте, что вы младенец и собираетесь попробовать новую еду. Эволюция любезно снабдила ваш язык вкусовыми рецепторами, которые доставляют вам удовольствие, когда на язык попадает что-то сладкое, поэтому вы любите материнское молоко. Ребенок, придерживающийся такой политики, будет пытаться изучить новую политику, используя текущую в качестве отправной точки. Скорее всего, дети будут склонны пробовать другие сладости, похожие на молоко. Ребенок, придерживающийся политики, будет сладкоежкой. Ребенок вне политики, однако, по-прежнему использует текущую политику в качестве отправной точки, но ему разрешается исследовать другие, возможно, случайные варианты, пока ему дают молоко. Ребенок вне политики все еще любит сладкое молоко, но может также обнаружить, что ему нравятся другие приятные вкусы.

На данный момент различие может показаться небольшим, но это раннее открытие позволило удивительно продвинуться в использовании RL, что мы и наблюдаем сегодня. Большинство современных алгоритмов вне политики и призваны поощрять или улучшать исследования. Они позволяют использовать механизм планирования для управления агентом и, как правило, лучше работают над задачами с отложенным вознаграждением. Однако алгоритмы политик, как правило, обучаются

быстрее, потому что они могут мгновенно использовать новые стратегии. Современные алгоритмы пытаются найти баланс между этими качествами, чтобы достичь золотой середины.

## Дискретные или непрерывные действия

Действия в среде могут быть самыми разнообразными: вы можете изменять величину крутящего момента, приложенного к устройству управления двигателем; решать, добавлять ли банан в корзину для покупок, или покупать акции на миллионы долларов в сделке.

Некоторые действия, по сути, бинарны: дорожный знак "стоп/движение разрешено" имеет ровно два класса. В других случаях у вас могут быть категории, которые вы можете закодировать в виде бинарных действий. Например, вы можете разделить управление дроссельной заслонкой транспортного средства на три бинарных действия: полное закрытие, средняя мощность и полная мощность.

Но часто действия требуют большей ловкости. Когда вы ведете машину, вы поворачиваете рулевое колесо на бесконечное количество углов. Если бы вы делали это дискретно, то это обернулось бы головной болью. Хотели бы вы, чтобы водитель автобуса поворачивал руль с шагом 90 градусов?

Промежуток или продолжительность действия также могут быть важны. В игре Super Mario Bros. чем дольше вы удерживаете кнопку прыжка, тем выше прыгает Марио. Можно включить время в число условий задачи и получить непрерывное действие, которое, например, представляет интервал времени, в течение которого вы удерживаете кнопку. Либо можете сделать эту величину дискретной и убедиться, что вы неоднократно опрашиваете агента, чтобы узнать, должен ли он продолжать выполнять действие. Если вы уверены, что продолжительность действия не связана с его выполнением, то ее можно обозначить через отдельную переменную.

Алгоритмы RL должны обрабатывать как бинарные, так и непрерывно изменяемые действия. Но многие алгоритмы ограничиваются одним вариантом.

## Методы оптимизации

Примерно в возрасте 14–16 лет вы, вероятно, научились решать линейные уравнения вручную, имея исходные данные и много бумаги. Но примеры, которые вы решали, скорее всего, были очень простыми и с одной неизвестной переменной. В реальном мире вы часто будете работать с сотнями, тысячами или даже миллионами независимых переменных. На этом этапе невозможно использовать те же методы, которым вы научились в школе, из-за вычислительной сложности реальных примеров.

Как правило, вы будете строить модели (которые могут содержать или не содержать линейные уравнения) и обучать их с помощью метода оптимизации. У RL такая же проблема: нужно создать агента, который сможет предложить решение для поставленной цели. Как именно это происходит — еще одна фундаментальная тема в RL.

Один из способов — попробовать как можно больше действий и записать результаты. В дальнейшем вы можете направлять агента, следуя стратегии, которая привела к наилучшему результату. Это так называемые *ценностные* алгоритмы, и я вскоре познакомлю вас с ними.

Другой способ — поддерживать модель и настраивать параметры модели, чтобы стремиться к действиям, которые привели к наилучшему результату. Это так называемые алгоритмы *на основе политик*. Вы можете прочитать о них подробнее в *главе 5*.

Для того чтобы это было проще понять, представьте себе двумерную сетку с обрывом на юге. Ваша задача — сконструировать робота, который будет многократно проверять каждый квадрат и узнавать, что падение со скалы связано с определенными затратами. Если бы вы использовали алгоритм, основанный на ценностях, и преобразовали стратегию в слова, он бы сказал: "Не ступайте с обрыва". Алгоритм, основанный на политике, сказал бы: "Отойдите от обрыва". Тонкое, но важное отличие.

Алгоритмы, основанные на ценностных значениях и применении политик, в настоящее время наиболее изучены и, следовательно, наиболее популярны. Но алгоритмы *на основе имитации*, в которых вы оптимизируете агента для имитации действий эксперта, могут хорошо работать, когда вы пытаетесь привлечь человека к руководству процессом. Любые другие алгоритмы, которые не подходят ни к одному из этих классов, могут породить новые методологии в будущем.

## Оценка и улучшение политики

Другой способ интерпретации того, как алгоритм улучшает свою стратегию, — рассматривать его с точки зрения *оценки политики* и *улучшения политики* (рис. 1.2).

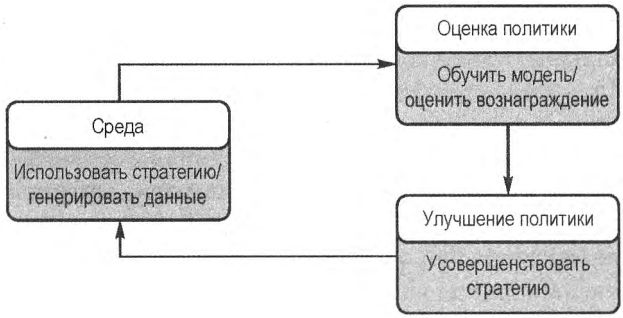



**Рис. 1.2.** Интерпретация того, как алгоритмы обновляют свою стратегию

Во-первых, агент следует стратегии (политике) для принятия решений, и эта политика генерирует новые данные, описывающие состояние среды.

На основе этих новых данных агент пытается предсказать вознаграждение исходя из текущего состояния среды; он оценивает текущую политику.

Затем агент использует этот прогноз, чтобы решить, что делать дальше. В целом он пытается изменить стратегию, чтобы улучшить политику. Он может предложить перейти в состояние с более высокой прогнозируемой наградой или запросить дополнительные исследования. В любом случае действие возвращается к окружающей среде, и так начинается следующая итерация.

Подавляющее большинство алгоритмов следуют этому шаблону. Это настолько фундаментальная структура, что если мне когда-нибудь представится возможность переписать эту книгу, я бы подумал о том, чтобы представить ее содержание таким образом.

# Фундаментальные концепции обучения с подкреплением

Идея обучения методом проб и ошибок как фундаментальная основа всех алгоритмов RL возникла в ранних работах по психологии обучения животных. Знаменитый русский физиолог Иван Петрович Павлов впервые сообщил в 1927 г., что можно запустить пищеварительную систему животного, используя стимулы, не имеющие отношения к процессу приема пищи. В одном известном эксперименте он измерил количество слюны, выделяемой собаками, когда им давали пищу. В то же время он ввел в эксперимент звуковой сигнал. После нескольких повторений у собаки возникало слюноотделение в качестве реакции только на звук [30].

Звук не является естественным предвестником пищи и не помогает при приеме пищи. Связь между врожденными рефлексами, такими как моргание глаз или образование слюны, и новыми стимулами теперь называется классическим, или павловским, обусловливанием.

## Первый RL-алгоритм

В 1972 г. Роберт Рескорла (Robert Rescorla) и Аллан Вагнер (Allan Wagner) обнаружили еще один интересный феномен, который невозможно объяснить с помощью павловского обусловливания. Сначала они подули струей воздуха в глаз кролика, и он моргнул. Затем они научили кролика ассоциировать внешний раздражитель, звук, с дуновением воздуха. Кролик моргнул, когда услышал звук, даже при отсутствии дуновения. Затем они переучили кролика моргать при воздействии как звука, так и света. Опять же, когда кролик услышал звук и увидел свет без дуновения воздуха, он моргнул. Но затем, когда исследователи только мигнули светом, кролик не моргнул [31].

У кролика сложилась иерархия ожиданий; звук и свет равны мерцанию. Когда кролик не соблюдал базовое ожидание (звук), это *блокировало* все последующие условия. Возможно, вы сами испытали это ощущение. Время от времени вы узнаете что-то настолько невероятное, настолько фундаментальное, что вам может казаться, что любые убеждения, следующие из узнанного, не выдерживают критики. Ваша базовая обусловленность была нарушена, а ожидания высшего порядка за-

блокированы. Результатом этой работы стала модель Рескорла — Вагнера. Их исследование никогда ранее не было представлено в таком ракурсе, но здесь оно описывается с применением метода, называемого *оценкой ценности*.

## Оценка ценности

Представьте, что вы пытаетесь смоделировать опыт с кроликами в постановке Рескорла — Вагнера. Цель состоит в том, чтобы предсказать, когда кролик моргнет. Вы можете создать модель этого эксперимента, описав входные данные и ожидаемый результат. Входные данные представляют действия под вашим контролем, а выход — предсказание того, моргает ли кролик.

Вы можете представить входные данные в виде вектора $\mathbf{s} = (s_0, s_1, ..., s_{n-1})$, где $s_i = 1$, если $i$-й стимул присутствует в испытании, и $s_i = 0$ — в противном случае. Это бинарные действия.

Допустим, например, что функция $s_0$ представляет звук, а $s_1$ — свет. Тогда вектор $\mathbf{s} = [0, 1]$ представляет ситуацию, когда звук отсутствует, а свет есть.

Вы можете записать предполагаемый результат в виде $V$, выразив таким образом прогноз того, моргнет кролик или нет. Затем состояния отображаются в правильное предсказание с помощью функции $V(\mathbf{s})$.

Я твердо решил максимально упростить математику, чтобы улучшить читаемость и понимание книги. Это означает, что книга теряет математическую формальность, такую как обозначение оценки с помощью оператора hat, для улучшения читаемости. См. академические статьи, где соблюдается полная математическая строгость.

Теперь самое сложное: определение функции отображения. Одним из распространенных решений является умножение входных данных на *параметры*, находящиеся под вашим контролем. Вы можете изменить эти параметры, чтобы получить результат, зависящий от *входов*. Эти параметры называются *весами*, а модель, которую вы только что построили, является *линейной*.

Веса определяются другим вектором **w**, который имеет ту же форму, что и признаки. Например, данные могут показать, что свет не заставлял кролика моргать; он моргнул только тогда, когда услышал звук. В результате получается модель с весом, равным 1, для параметра звука, и весом, равным 0, для параметра света.

Формально функция представляет собой сумму входных данных, умноженных на веса. Эта операция называется *скалярным произведением*. Результат, который показан в уравнении 1.1, является предсказанием того, моргает ли кролик.

**Уравнение 1.1. Оценка ценности**

$$V(\mathbf{s}, \mathbf{w}) \doteq w_0 s_0 + w_1 s_1 + \ldots + w_n s_n = \mathbf{w} \cdot \mathbf{s} = \mathbf{w}^T \mathbf{s}.$$

Но в целом, как вы определяете значения весов?

## Ошибка предсказания

Вы можете использовать метод оптимизации, чтобы найти оптимальные параметры. Самый распространенный метод — количественно оценить, насколько неверна ваша оценка по сравнению с реальной (правильным ответом). Затем вы можете попробовать множество разных весов, чтобы минимизировать ошибку.

Ошибка $\delta$ в вашем прогнозе — это разница между фактическим результатом эксперимента $E$ и прогнозом (оценкой ценности). Прогноз основан на текущем состоянии окружающей среды (наблюдение за кроликом) $\mathbf{s}$ и текущих весах. Все переменные меняются со временем и часто обозначаются индексом $t$, но я игнорирую это, чтобы уменьшить количество обозначений. Вы можете увидеть это в уравнении 1.2.

**Уравнение 1.2. Ошибка предсказания**

$$\delta \doteq E - V(\mathbf{s}, \mathbf{w}).$$

Одной из трактовок уравнения 1.2 является утверждение, что $\delta$ характеризует степень непредсказуемости. Если вы сделаете прогноз, что кролик обязательно моргнет, а этого не произойдет, то разница между тем, что вы предсказали, и тем, что произошло, будет велика; вы будете удивлены результатом.

Учитывая количественное определение удивления, как следует изменить веса? Один из способов — изменить веса пропорционально ошибке прогноза. Например, рассмотрим ситуацию, когда ожидалось, что звук (индекс 0) и свет (индекс 1) имеют одинаковую важность. Набор весов $\mathbf{w} = [1, 1]$.

На этот раз эксперимент состоит в том, чтобы использовать только свет, а звук не использовать. Таким образом, наблюдаемое состояние эксперимента $\mathbf{x}(\mathbf{s}) = [0, 1]$. Для того чтобы сделать прогноз $V(\mathbf{s}, \mathbf{w})$, вычислите скалярное произведение на только что приведенных векторах. Результат равен 1 (моргает).

Когда вы запустили симуляцию, кролик не моргнул. Фактическое значение было 0. Значение $\delta$ в уравнении 1.2 равно 1.

Знание предыдущего состояния и ошибки прогноза помогает изменить веса. Умножая их, получаем $\delta\mathbf{x}(\mathbf{s}) = [0, 1]$. Добавление этого к текущим весам дает $\mathbf{w} = [1, 0]$.

Обратите внимание, как новые веса правильно предсказали бы фактический результат 0 (не моргает): $\mathbf{w}^T \mathbf{x}(\mathbf{s}) = [1, 0]^T [0, 1] = 0$.

## Правило обновления веса

Если у вас есть какой-либо опыт работы с машинным обучением, вы знаете, что не следует сразу пытаться перейти к правильному, *оптимальному результату*. Одна из причин заключается в том, что экспериментальный результат может быть зашумленным. Всегда постепенно приближайтесь к правильному результату и повторяйте эксперимент несколько раз. Затем *по закону больших чисел* результат сой-

дется к лучшему среднему ответу, который также является оптимальным. Одно предостережение заключается в том, что это верно лишь в том случае, если лежащая в основе математика доказывает, что конвергенция неизбежна; во многих алгоритмах, например с нелинейными аппроксиматорами, данный факт не гарантируется.

Это формализовано как *правило обновления весов*. В уравнении 1.3 вы можете управлять скоростью обновления весов с помощью гиперпараметра $\alpha$, который должен находиться в диапазоне от 0 до 1. Уравнение 1.3 *обновляет* веса последовательно, т. е. веса на следующем временно́м шаге выводятся из весов текущего временно́го шага. Опять же, я игнорирую нижние индексы, чтобы упростить обозначения.

**Уравнение 1.3. Правило обновления весов**

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha\delta\mathbf{x}(\mathbf{s}).$$

## RL — это то же самое, что ML?

Математические выражения, представленные в уравнениях 1.1 и 1.3, описывают психологическую модель обучения животных. Вы можете легко применить тот же подход к программным агентам. Но настоящая причина для такого углубления заключалась в том, чтобы дать математическое упражнение для разминки перед остальной частью книги. Здесь я ввел математические символы, которые используются в последующих главах (в исходной статье использовались символы и терминология, неприменяемые в психологии).

Если вы ненадолго абстрагируетесь от математики, идея обновления весов для получения лучшего соответствия может показаться знакомой. Вы правы, если узнали в этом регрессию из ML. Цель состоит в том, чтобы предсказать числовой результат (общее вознаграждение) на основе заданного набора входных данных (наблюдений). Практики называют это *проблемой предсказания*. Для того чтобы принимать правильные решения, вам необходимо уметь предугадывать, какие действия будут оптимальными.

Главное отличие ML от RL в том, что вы даете агентам свободу выбора. Это *контрольная* часть задачи, и на первый взгляд кажется, что это простое дополнение к задаче *предсказания* машинного обучения. Однако не стоит недооценивать эту проблему. Вы можете подумать, что лучшая политика — выбрать действие, для которого прогнозируется наивысший известный результат. Нет, так не делается, потому что могут быть другие состояния, которые приносят еще большее вознаграждение.

Контроль обеспечивает автономию; супермашины могут учиться на своих ошибках. Этот навык открывает ряд новых возможностей. Многие задачи настолько сложны — например, как научить робота фотографировать определенный объект, — что инженерам приходится прибегать к ограничениям, правилам и разбивать задачу на подзадачи. Точно так же я работаю с клиентами, которым приходит-

ся ежедневно или ежечасно переобучать свои модели для вовлечения новых данных. Хватит! Откажитесь от контроля и используйте RL.

Алгоритмы RL пытаются сбалансировать эти две проблемы — исследование и эксплуатацию — разными способами. Одно общее различие заключается в том, как алгоритмы изменяют скорость, с которой ведется исследование. Некоторые алгоритмы исследуют задачу с фиксированной скоростью, другие устанавливают скорость, пропорциональную предсказанному значению. Некоторые даже пытаются определить вероятность получения максимальной награды для заданного состояния. Это проблема обсуждается на протяжении всей книги.

## Награда и отклик

Исследователи ML черпали вдохновение в нейробиологии. Наиболее часто приводимым примером, иллюстрирующим это, является создание *искусственного нейрона*, на основе которого строятся нейронные сети сетей и *глубокое обучение*. Искусственный нейрон — это модель единичного нейрона, входящего в состав мозга, во многом подобно тому, как атомная модель предполагает наличие элементарной единицы материи. На парадигму RL, в частности, повлияла коммуникативная модель, действующая в мозге.

Нейромедиатор дофамин вырабатывается специальными клетками мозга. Он участвует в основных мозговых процессах, включая мотивацию и обучение, и следовательно, в принятии решений. Он также может иметь негативные аспекты, такие как зависимость и ряд заболеваний. Хотя о дофамине еще многое неизвестно, имеются данные, что он играет важную роль в обработке вознаграждений.

Традиционные теории о присутствии дофамина основывались на усиливающемся и приятном эффекте химического вещества. Но исследования начала 1990-х годов выявили ряд поразительных фактов. Исследователи смогли количественно измерить активность дофамина у обезьян. Их результаты показали, что существует фоновый уровень постоянного выброса дофамина. Исследователи обучили каждую обезьяну выполнению традиционной условной задачи (подобной той, которую мы рассмотрели в *разд. "Оценка ценности" ранее в данной главе*): обезьяны ожидали еду через несколько минут после включения света. Когда тренировка началась, наблюдался значительный всплеск уровня дофамина над фоновым уровнем при получении награды. Со временем исследователи заметили, что высвобождение дофамина у обезьян сдвигается в сторону появления света. В конце концов, каждая обезьяна переживала выброс дофамина всякий раз, когда видела свет [32]. Если вы простите мне небольшое отступление к реальному человеческому опыту, я могу засвидетельствовать, что мои дети больше взволнованы перспективой получить мороженое, чем на самом деле есть мороженое.

Еще более увлекательное дальнейшее исследование использовало ту же тренировку с условиями и продемонстрировало такой же всплеск дофамина, когда обезьяна предсказывала вознаграждение из-за условного стимула. Однако когда исследователи показывали стимул, но не награждали обезьяну в ожидаемое время, наблюдалось значительное снижение фонового уровня дофамина. Когда обезьяна не полу-

чала награды, наблюдался отрицательный эффект дофамина (по сравнению с исходным уровнем) [33].

Как и в *разд. "Ошибка предсказания" ранее в данной главе*, эти события можно хорошо смоделировать с помощью корректирующего процесса. Дофаминовые нейроны сами по себе не сигнализируют о награде. Дофамин — это сигнал, представляющий ошибки предсказания вознаграждения. Другими словами, дофамин — это $\delta$ мозга.

## Отложенные награды

Вознаграждение имитирует идею мотивации агентов. Что побуждает вас вставать по утрам? Вы мгновенно получаете награду? Возможно, нет. Вот почему отсроченное удовлетворение является таким важным и трудным жизненным навыком, который нужно освоить. Награды за то, что вы встали с постели утром, съели здоровый завтрак, усердно потрудились и были вежливыми с людьми, для всех нас разные.

Проблему отложенного вознаграждения также сложно решить в RL. Часто агенту нужно долго ждать, чтобы обнаружить награду. Здесь могут помочь алгоритмические приемы, но, по сути, изменение сигнала вознаграждения так, чтобы он обеспечивал более частые обновления, помогает агенту найти решение. Такое изменение вознаграждения называется *формированием вознаграждения*, но мне нравится думать об этом как об *инжиниринге наград*, сродни разработке функций в ML.

С этой проблемой связано присвоение награды. Например, какие решения привели вас к ситуации, в которой вы находитесь сейчас? Как вы можете это знать? Раннее решение могло повлиять на успех или неудачу. Но, учитывая все возможные состояния и действия в сложной системе, решить эту проблему сложно. В целом можно сказать только, что решения являются оптимальными в среднем с учетом набора допущений и ограничений.

## Ретроспектива

Идиома *"доверяй своей интуиции"* означает, что вы должны доверять своей инстинктивной реакции, осваиваясь в новой ситуации. Эти эвристические штампы подтверждены опытным путем и, как правило, приводят к хорошим результатам. Критическое мышление — процесс, при котором вы логически и методично прорабатываете проблему, — требует гораздо больше энергии и концентрации.

Эти два процесса в мозге, часто называемые "1-й и 2-й моделями системы", работают как механизм энергосбережения. Зачем тратить драгоценные умственные ресурсы на повседневные задачи? Затруднительно осознанно думать о том, как именно ходить. Но страдающим болезнью Альцгеймера приходится сталкиваться с этой ужасно изнурительной ситуацией ежедневно [34].

Исследователи предполагают, что неврологические различия в головном мозге объясняют эти системы [35]. Задача "инстинктивной" части мозга — быстро принимать решения. Задача другой — проверить эти действия и при необходимости принять корректирующие меры.

Эта структура привела к развитию семейства алгоритмов *"актор — критик"* (actor — critic). *Актор* (исполнитель, actor) несет ответственность за принятие важных решений, чтобы получить максимальное вознаграждение. *Критик* (critic) готов спланировать будущее и поправить актора, когда тот получает неправильный ответ. Это открытие было жизненно важным для многих продвинутых алгоритмов RL.

## Обучение с подкреплением как дисциплина

Парадигма RL развивалась как две независимые дисциплины примерно до 1980-х годов. Психология изучала поведение животных. Инженеры-механики и электронщики разработали теорию для описания оптимального управления системами.

Термин *"оптимальное управление"* возник в 1950-х годах с целью описания того, как настроить систему для достижения поставленной цели. Усилия достигли кульминации в 1957 г., когда Ричард Беллман (Richard Bellman) разработал *марковский процесс принятия решений* (Markov decision process, MDP) — набор требований к математически управляемой среде и *динамическому программированию* (метод решения MDP) [36].

Согласно одному источнику, исследования поведения животных восходят к XIX веку, когда проводились эксперименты, включающие "экспериментальное нащупывание" [37]. Эдвард Торндайк (Edward Thorndike) запирал кошек в "коробках с головоломками" и фиксировал, сколько времени требовалось животным, чтобы сбежать. Он обнаружил, что время побега уменьшилось посредством повторения опытов и закрепления результатов с 5 минут до 6 секунд. Результатом этой работы стал "закон эффекта", более известный как обучение методом проб и ошибок.

Термин *"подкрепление"* впервые появился в переводах из рукописей Павлова об условных рефлексах в 1927 г. Но RL было популяризировано дедушкой вычислительной техники Аланом Тьюрингом, когда в 1948 г. он изложил свои самые ранние мысли об искусственном интеллекте, как "организовать" физический набор электронных схем, которые он назвал "машинами", чтобы сделать что-то практическое:

> "Это можно сделать, просто позволив машине произвольно перемещаться в последовательности ситуаций и применяя болевые стимулы, когда сделан неправильный выбор, и стимулы удовольствия, когда сделан правильный. Лучше всего применять болевые стимулы, когда сделан неуместный выбор. Это сделано для того, чтобы не попасть в кольцо неуместных ситуаций. Теперь машина „готова к работе“" [39].
>
> *Алан Тьюринг (1948)*

Я нахожу удивительным, насколько работы Тьюринга актуальны и сегодня. В его время исследователи создавали роботов для решения повседневных задач. Один особенно гениальный исследователь по имени Уильям Грей Уолтер (William Grey Walter) построил "механическую черепаху" в 1951 г. В 1953 г. он представил "черепаху" под названием CORA (conditioned reflex analogue — аналог условного рефлекса), которая была способна "учиться" в окружающей среде. Робот содержал

схемы, которые могли имитировать эксперименты с павловским обусловливанием. Уже тогда публика была очарована такими машинами, которые могли "учиться":

> "В Англии полицейский свисток состоит из двух звуков, которые звучат вместе и издают особенно неприятный звук. Поэтому я пытался научить [CORA], что одна нота означает препятствие, а другая — пищу. Я попытался создать этот дифференциальный рефлекс с помощью двух настроенных контуров, один из которых был связан с реакцией аппетита, а другой — с реакцией избегания. Все было устроено так, что одна сторона свистка использовалась до того, как машина коснулась объекта, чтобы она научилась избегать его; в то время как другая сторона свистка использовалась до того, как она должна была увидеть свет. Эффект от подачи обеих нот почти всегда был катастрофическим; машина сразу шла в темноту в правой части комнаты и пять минут зависала там в каком-то угрюмом настроении. Она стала невосприимчивой к стимуляции и бегала по кругу" [40].
>
> *Уильям Грей Уолтер (1956)*

К концу 1950-х годов интересы исследователей сместились от обучения методом проб и ошибок к обучению с учителем. Фактически сначала люди использовали эти два термина как синонимы. Пионеры нейронных сетей, включая Фрэнка Розенблатта (Frank Rosenblatt), Бернарда Уидроу (Bernard Widrow) и Теда Хоффа (Ted Hoff), использовали в своих статьях термины *"вознаграждение"* и *"наказание"*. Это вызвало путаницу, потому что понятия "обучение с учителем", "обучение без учителя" и "обучение с подкреплением" использовались для обозначения одной и той же идеи.

В то же время завышенные ожидания возможностей искусственного интеллекта в 1950–1960-х годах вызвали всеобщее недовольство медленным прогрессом. В 1973 г. в отчете "Lighthill report" о состоянии исследований искусственного интеллекта в Великобритании критиковалась полная неспособность достичь "грандиозных целей" [41]. На основании этого отчета бóльшая часть государственного финансирования исследований искусственного интеллекта была урезана сначала в Великобритании, а затем и в остальном мире. 1970-е годы вошли в историю как "зима искусственного интеллекта", и прогресс застопорился.

Возрождение RL в 1980-х годах, как было признано, произошло благодаря Гарри Клопфу (Harry Klopf), который в 1970-х годах заявлял, что знания и навыки использования ключевых методов обучения теряются. Он подчеркивал, что контроль окружающей среды для достижения желаемого результата является ключом к созданию интеллектуальных систем. Ричард Саттон (Richard Sutton) и Эндрю Барто (Andrew Barto) работали в 1980–1990-х годах над продвижением идей Клопфа по объединению областей психологии и теории контроля (*см. разд. "Фундаментальные концепции обучения с подкреплением" ранее в данной главе*) посредством *обучения на основе временны́х различий* (temporal-difference, TD). В 1989 г. Крис Уоткинс (Chris Watkins) интегрировал все предыдущие направления исследований RL и создал *Q-обучение*.

На этом я заканчиваю свой очень краткий обзор истории RL, потому что результаты следующих 30 лет исследований составляют содержание этой книги. Вы можете

подробнее узнать об истории RL из ссылок в *разд. "Дополнительные материалы для чтения" далее в этой главе*.

## Резюме

Компания по исследованию рынка Gartner предполагает, что в США развитие искусственного интеллекта приносит предприятиям триллионы долларов в год [42]. RL играет большую роль на этом рынке, поскольку многие из сегодняшних бизнес-задач имеют стратегический характер. От торгового зала до руководства компании существует множество неоптимизированных многоэтапных решений, таких как добавление товаров в корзины или определение стратегий выхода на рынок. В одиночку ML не оптимально, т. к. оно недостаточно дальновидно для таких задач. Но бурный рост объемов данных, вычислительных мощностей и улучшенное моделирование дают возможность существовать программно-управляемым агентам ML и RL, которые могут изучать оптимальные стратегии, превосходящие придуманные людьми.

Биологические процессы продолжают стимулировать внедрение RL. Ранние психологические эксперименты подчеркивали важность исследования и вознаграждения. Исследователи предложили различные способы эмуляции обучения с подкреплением, что породило несколько общих тем. Как инженер, вы должны решить, какая из этих тем подходит для вашей задачи: например, как вы оптимизируете свой алгоритм RL, какую форму принимают действия, требуется ли агенту формальный механизм планирования и следует ли обновлять политику в режиме онлайн? Вы приобретете больше опыта с помощью этой книги, и в последних главах я покажу вам, как применять полученный опыт. Тщательно определив действия и вознаграждения, вы можете разработать агентов, которые будут работать в среде для решения ряда промышленных задач. Но сначала вам нужно узнать об алгоритмах, которые позволяют агентам изучать оптимальные политики. Таким образом, основная цель этой книги — научить вас тому, как агенты работают и как их применять.

## Дополнительные материалы для чтения

- Введение в RL.
  - Ресурсов много, но у Сергея Левина[2] один из лучших.
- История RL.
  - Саттон Р. С., Барто Э. Дж. Обучение с подкреплением: введение. — MIT Press, 2018 (Sutton R. S., Barto A. G. Reinforcement learning: an introduction. — MIT Press, 2018).
  - Исторический обзор RL с 1996 г., но актуален и сегодня [43].

[2] См. **https://oreil.ly/wgxnk**.

# Использованные источники

[1] Reinsel D., Gantz J., Rydning J. The digitization of the world from edge to core. — IDC, 2018. — URL: **https://oreil.ly/96vhZ**.

[2] AWS Snowmobile is a service that allows you to use a shipping container to snailmail your data to its datacenters. — URL: **https://oreil.ly/O5-K9**.

[3] Papers of Dr Dietrich G. Prinz-Archives Hub. — URL: **https://oreil.ly/XCG9g**.

[4] Copeland B. Jack. The essential turing. — Clarendon Press, 2004.

[5] Drunk Swedish Elk found in Apple Tree Near Gothenburg // BBC News. — 2011. — 8 September. — URL: **https://oreil.ly/zr3Da**.

[6] Kormushev P. et al. Reinforcement learning in robotics: applications and real-world challenges // Robotics. — 2013. — Vol. 2, № 3. — P. 122–48. — URL: **https://oreil.ly/juf15**.

[7] Huang W., Braghin F., Wang Z. Learning to drive via apprenticeship learning and deep reinforcement learning // ArXiv:2001.03864. — 2020. — January. — URL: **https://oreil.ly/pqtHi**.

[8] Dutreilh X. et al. 2011. Using reinforcement learning for autonomic resource allocation in clouds: towards a fully automated workflow // ICAS 2011, The Seventh International Conference on Autonomic and Autonomous Systems. — 2011. — P. 67–74.

[9] Liu N. et al. 2017. A Hierarchical framework of cloud resource allocation and power management using deep reinforcement learning // ArXiv:1703.04221. — 2017. — August. — URL: **https://oreil.ly/N2wL7**.

[10] DeepMind AI reduces Google Data Centre cooling bill by 40% // Deep-Mind. — Accessed 3 July 2019. — URL: **https://oreil.ly/rjAae**.

[11] Das A. et al. Reinforcement learning-based inter- and intra-application thermal optimization for lifetime improvement of multicore systems // Proceedings of the 51st Annual Design Automation Conference. — 2014. — P. 170:1–170:6. — DAC'14. New York, NY, USA: ACM.

[12] Littman M., Boyan J. A Distributed reinforcement learning scheme for network routing // Proceedings of the International Workshop on Applications of Neural Networks to Telecommunications. — 2013. — 17 June. — URL: **https://oreil.ly/543Qz**.

[13] Wang H. Large scale continuous-time mean-variance portfolio allocation via reinforcement learning // ArXiv:1907.11718. — 2019. — August. — URL: **https://oreil.ly/5J5qV**.

[14] Wang J. et al. AlphaStock: a buying-winners-and-selling-losers investment strategy using interpretable deep reinforcement attention networks // Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining — KDD'19. — 2019. — P. 1900–1908. — URL: **https://oreil.ly/c0HA2**.

[15] Maestre R. et al. Reinforcement learning for fair dynamic pricing // In Intelligent Systems and Applications, edited by Kohei Arai, Supriya Kapoor, and Rahul Bhatia. Advances in Intelligent Systems and Computing. — Springer International Publishing, 2019. — P. 120–135.

[16] Mason K., Grijalva S. A review of reinforcement learning for autonomous building energy management // ArXiv:1903.05196. — 2019. — March. — URL: **https://oreil.ly/lSfBf**.

[17] Rolnick D. et al. Tackling climate change with machine learning // ArXiv:1906.05433. — 2019. — June. — URL: **https://oreil.ly/eUDYX**.

[18] LA P., Bhatnagar S. Reinforcement learning with function approximation for traffic signal control // IEEE Transactions on Intelligent Transportation Systems. — 2011. — Vol. 12, № 2. — P. 412–421. — URL: **https://oreil.ly/1tBej**.

[19] Rezaee K., Abdulhai B., Abdelgawad H. Application of reinforcement learning with continuous state space to ramp metering in real-world conditions // In 2012 15th International IEEE Conference on Intelligent Transportation Systems. — 2012. — P. 1590–1595. — URL: **https://oreil.ly/G60Wu**.

[20] Mohammadi M., Al-Fuqaha A., Guizani M., Oh J. Semisupervised deep reinforcement learning in support of IoT and smart city services // IEEE Internet of Things Journal. — 208. — Vol. 5, № 2. — P. 624–635. — URL: **https://oreil.ly/orzkT**.

[21] Shah P. Reinforcement learning with action-derived rewards for chemotherapy and clinical trial dosing regimen selection // MIT Media Lab. — Accessed 4 July 2019. — URL: **https://oreil.ly/p3sBr**.

[22] Liu Y. et al. Deep reinforcement learning for dynamic treatment regimes on medical registry data // Healthcare Informatics: The Business Magazine for Information and Communication Systems (August). — 2017. — P. 380–85. — URL: **https://oreil.ly/BftR5**.

[23] Mohammedalamen M. et al. Transfer learning for prosthetics using imitation learning // ArXiv:1901.04772. — 2019. — January. — URL: **https://oreil.ly/hTpsA**.

[24] Chi M. et al. Empirically evaluating the application of reinforcement learning to the induction of effective and adaptive pedagogical strategies // User Modeling and User-Adapted Interaction. — 2011. — Vol. 21, № 1. — P. 137–80. — URL: **https://oreil.ly/XyKdy**.

[25] Pwnagotchi: Deep reinforcement learning for Wifi Pwning! — URL: **https://oreil.ly/olVVt**.

[26] Anderson H. S. et al. Learning to evade static PE machine learning malware models via reinforcement learning // ArXiv:1801.08917. — 2018. — January. — URL: **https://oreil.ly/X2n4W**.

[27] Freedberg S. J., Jr. AI & robots crush foes in army wargame // Breaking Defense. — 2019. — December. — URL: **https://oreil.ly/4UA9Z**.

[28] Silver D. et al. Mastering the game of go without human knowledge // Nature. — 2017. — Vol. 550, № 7676. — P. 354–359. — URL: **https://oreil.ly/LJROD**.

[29] Zha D. et al. RLCard: a toolkit for reinforcement learning in card games // ArXiv:1910.04376. — 2019. — October. — URL: **https://oreil.ly/YbLGv**.

[30] Pavlov I. P. Conditioned reflexes: an investigation of the physiological activity of the cerebral cortex. — Oxford, England: Oxford Univ. Press, 1927.

[31] Rescorla R. A., Wagner A. R. A Theory of pavlovian conditioning: variations in the effectiveness of reinforcement and nonreinforcement // Classical Conditioning II: Current Research and Theory. — New York, NY: Appleton-Century-Crofts, 1972. — P. 64–99.

[32] Schultz W. et al. Reward-related signals carried by dopamine neurons // Models of Information Processing in the Basal Ganglia. — Computational Neuroscience. Cambridge, MA: The MIT Press, 1995. — P. 233–248.

[33] Schultz W., Dayan P., Montague P. R. A Neural substrate of prediction and reward // Science. — 1997. — Vol. 275, № 5306. — P. 1593–1599. — URL: **https://oreil.ly/6KztR**.

[34] Kahneman D. Thinking, fast and slow. — Penguin UK, 2012.

[35] Takahashi Y., Schoenbaum G., Niv Y. Silencing the critics: understanding the effects of cocaine sensitization on dorsolateral and ventral striatum in the context of an actor/critic model // Frontiers in Neuroscience 2. — 2008. — URL: **https://oreil.ly/fXxaf**.

[36] Bellman R. A Markovian decision process // Journal of Mathematics and Mechanics. — 1957. — Vol. 6, № 5. — P. 679–684.

[37] Newman E. B. Experimental psychology // Psychological Bulletin. — 1954. — Vol. 51, № 6. — P. 591–593. — URL: **https://oreil.ly/qC3Ne**.

[38] Thorndike E. L. Animal intelligence: an experimental study of the associative processes in animals // The Psychological Review: Monograph Supplements. — 1898. — Vol. 2, № 4. — P. i–109. — URL: **https://oreil.ly/VLNEN**.

[39] Turing A. M. Intelligent machinery // B. Jack. Copeland. — The Essential Turing. — Clarendon Press, 2004. — P. 428.

[40] Grey Walter W. Presentation: Dr. Grey Walter // J.M. Tanner and B. Inhelder (eds.). Discussions on Child Development. — London: Tavistock Publications, 1956. — Vol. 2. — P. 21–74.

[41] Lighthill J. Artificial intelligence: a paper symposium. — London: Science Research Council, 1973.

[42] Gartner Says AI Augmentation Will Create $2.9 Trillion of Business Value in 2021 // Gartner : [site]. — Accessed 17 August 2020. — URL: **https://oreil.ly/n6kP7**.

[43] Kaelbling L. P., Littman M. L., Moore A. W. Reinforcement learning: a survey // ArXiv:Cs/9605103. — 1996. — April. — URL: **https://oreil.ly/xmhsX**.

ГЛАВА 2

# Марковские процессы принятия решений, динамическое программирование и методы Монте-Карло

Основу обучения с подкреплением составляют три раздела. Самый важный — *марковский процесс принятия решений* (Markov decision process, MDP), фреймворк, который поможет описать вашу задачу. *Динамическое программирование* (dynamic programming, DP) и *методы Монте-Карло* составляют основу всех алгоритмов, нацеленных на реализацию MDP. Однако сначала давайте обсудим приложение, о котором вы, вероятно, слышали, но не предполагали, что оно относится к обучению с подкреплением.

## Алгоритм многорукого бандита

Представьте себе, что вы работаете в компании, занимающейся электронной торговлей. Вы — инженер и должны разрабатывать новые приложения и обслуживать уже существующие. Например, вас могут попросить усовершенствовать процесс оформления покупок или переместить данные в новую библиотеку.

Как же удостовериться, что вносимые вами изменения принесут желаемый эффект? С одной стороны, можно следить за ключевыми показателями эффективности (key performance indicators, KPI). Вам нужно, чтобы новые элементы заставляли эти показатели расти. Если задача связана с обслуживанием, необходимо, чтобы все оставалось без изменений.

Для примера возьмем классическую задачу с подбором оптимального цвета для кнопки. Какой лучше: красный или зеленый? Как количественно оценить разницу? Для того чтобы подойти к этой задаче с использованием обучения с подкреплением, следует определить три ее ключевых элемента: награду, действия и среду.

Подробные практические советы по разработке RL-решений представлены в *главе 9*.

### Разработка наград

Для того чтобы можно было количественно оценить производительность, результат действия должен поддаваться измерению. В RL для этого и нужны награды. Награ-

ды подтверждают наличие исхода — хорошего или плохого. Как правило, хороший исход кодируется положительными значениями (но не всегда).

Какой сигнал вознаграждения лучше выбрать в примере с кнопкой для сайта? Можно сказать, что клики — это и есть позитивный исход. И уж точно продажи. Зачастую доступ к метрикам будет ограничен, и вам придется искать компромисс. Например, вместо того чтобы в качестве KPI использовать продажи, можно вычислить среднюю продажу для каждого, кто нажимает на кнопку, и присвоить это значение каждому из кликов. Таким образом, не нужно будет переходить на другую систему оценок.

Возьмем еще одну связанную с предыдущей задачу: требуется проверить, что новое ПО не нанесет значительного ущерба. Здесь может сработать такое настолько простое, как клики плюс награда. Если метрики отображают примерно одинаковое количество кликов — радуйтесь! Ничего не сломалось, день прошел не зря.

Так или иначе, награда должна быть измерима количественно и в идеале максимально простой. Если вы будете использовать слишком сложный сигнал вознаграждения, тогда разрабатываемый алгоритм может переобучиться и стать способным использовать недостатки в сигнале вознаграждения вместо решения предполагаемой проблемы. Например, награды типа "дистанция — цель" используются довольно часто и дают неплохие результаты. Однако агент, ищущий выход из лабиринта при помощи таких наград, скорее всего, застрянет в тупике, потому что близость к цели (но не сама цель!) гарантирует *почти* оптимальную награду.

Далее обсудим, как можно использовать награду для того, чтобы вычислить наилучший исход.

## Оценка стратегии: функция ценности

В среде агент берет на себя некоторое действие $a$, которое является членом математического множества $\mathcal{A}$. Другими словами, агент мог бы выбрать любое действие из множества $\mathcal{A}$, но в данном случае он выбрал $a$.

Записываем это как $a \in \mathcal{A}$.

Среда выдала агенту некоторую награду $r$, которая происходит из большой выборки потенциальных наград, определяемых множеством $\mathcal{R}$. Это перевело среду в новое состояние $s \in \mathcal{S}$.

Все эти переменные могут быть стохастическими, т. е. если ваш агент запросил то же самое действие в точно такой же ситуации, среда может ответить немного другой наградой или состоянием. В литературе вы можете увидеть обозначение этих терминов строчными буквами, но в этой книге в целях упрощения я буду использовать только прописные, хотя, возможно, это и не совсем верно.

В предыдущем примере с кнопкой на веб-сайте действие заключалось в том, чтобы определить, какого цвета должна быть кнопка — красного или зеленого. Математическое выражение будет следующим: $\mathcal{A} \doteq \{a_{\text{красная}}, a_{\text{зеленая}}\}$. Каждый раз, когда на ваш сайт заходит пользователь, ваш агент решает, какое действие выбрать, что определяется текущей стратегией. Затем пользователь нажимает или не нажимает

на кнопку, а награда отправляется агенту. Проще всего представить, что награда выдается за один клик и не выдается при отсутствии клика. Однако выдача награды может основываться также и на продажах или любом другом деловом показателе.

Вернемся к первоначальному вопросу: что же лучше? Количественно оценить производительность можно, вычислив среднюю награду $r^{avg}$ для определенного количества покупателей $N$ для каждого действия $a$, как показано в уравнении 2.1.

Помните, что представить одну кнопку и ожидать одного клика — это не последовательное задание, есть только одно-единственное решение. Это ключевая разница между бандитами и RL. Последовательные решения я продемонстрирую чуть позже

**Уравнение 2.1. Вычисление средней награды**

$$r^{avg}(a) \doteq \frac{1}{N(a)} \sum_{i=1}^{N(a)} r_i(a) = \frac{r_1 + r_2 + \ldots + r_{N(a)}}{N(a)}.$$

Уравнение 2.1 объясняет следующее: для того чтобы вычислить среднюю награду для каждого действия, нужно суммировать наблюдаемые награды и разделить их на количество клиентов.

Это работает, но неэффективно с точки зрения объема занимаемой памяти. Вдобавок вам придется ждать, пока вашу кнопку нажмет большое количество пользователей, чтобы вы смогли выполнить такое вычисление.

Вместо этого вы можете уравнение 2.1 преобразовать в *онлайн-алгоритм*: параметры алгоритма будут обновляться в режиме реального времени, и не придется ждать окончания всех экспериментов. Уравнение 2.2 определяет этот подход, потому что он важен для многих последующих алгоритмов RL.

**Уравнение 2.2. Онлайн-алгоритм вычисления функции ценности**

$$r_N^{avg} \leftarrow \frac{1}{N} \sum_{i=1}^{N} r_i \qquad (1)$$

$$\leftarrow \frac{1}{N} \left( r_N + \sum_{i=1}^{N-1} r_i \right) \qquad (2)$$

$$\leftarrow \frac{1}{N} \left( r_N + (N-1) \frac{1}{N-1} \sum_{i=1}^{N-1} r_i \right) \qquad (3)$$

$$\leftarrow \frac{1}{N} \left( r_N + (N-1) r_{N-1}^{avg} \right) \qquad (4)$$

$$\leftarrow \frac{1}{N} \left( r_N + N r_{N-1}^{avg} - r_{N-1}^{avg} \right) \qquad (5)$$

$$\leftarrow r_{N-1}^{avg} + \frac{1}{N} \left( r_N - r_{N-1}^{avg} \right) \qquad (6)$$

Это одна из таких математических манипуляций, которые очевидны лишь в перспективе. Цель — конвертировать статичное среднее вычисление в онлайн-версию при помощи математических хитростей. Если вы недостаточно сосредоточены, эти хитрости легко упустить.

Я начал с шага 1, где вновь использовал уравнение 2.1, однако на этот раз я убрал символ, указывающий на то, что награда является функцией действия. Это будет сохраняться для каждого действия. Также я использовал оператор присваивания значения $\leftarrow$, указывающий на то, что переменная обновляется на месте. Равенства тут нет.

В шаге 2 я начал извлекать из суммы самую последнюю награду так, чтобы оператор суммирования остановился на предпоследнем клиенте $N-1$.

В шаге 3 я применил математическую хитрость, умножив сумму на $1=(N-1)/(N-1)$. Этот шаг потребовал всей моей дальновидности для того, чтобы увидеть, что $\frac{1}{N-1}\sum_{i=1}^{N-1} r_i$ — то же самое, что и шаг 1, что делает его эквивалентным $r_{N-1}^{avg}$. Так, в шаге 4 я могу заменить расширение из предыдущего шага на $r_{N-1}^{avg}$

Между шагами 4 и 5 я произвожу некоторые вычисления, раскрывая $N-1$, и вы можете увидеть $1/N$ и $N$, которые сокращают друг друга.

В шаге 6, после умножения на $1/N$, результат выглядит как экспоненциально взвешенная скользящая средняя (известная также как экспоненциальное сглаживание) и является широко используемой формой онлайн-алгоритмов, вычисляющей среднее значение.

Большинство алгоритмов RL работают именно таким образом. Вы можете отбросить индексы в уравнении 2.2 и получить обычный экспоненциально взвешенный скользящий алгоритм, как показано в уравнении 2.3.

**Уравнение 2.3. Общая форма экспоненциально взвешенного скользящего среднего**

$$r \leftarrow r + a(r - r').$$

Возможно, вам это покажется знакомым, ведь это то же уравнение 1.3, разве что здесь я не использовал матричную математику. Вычисление ошибок и обновление предсказаний — популярная тема для специалистов по машинному обучению и RL.

Теперь вы знаете, как вычислить награду и обновить результат онлайн. Наш следующий шаг — на основании этих средних наград решить, какие действия предпринимать.

## Совершенствование политики: выбор лучшего действия

Предоставление агенту полного контроля над выбором, который он делает, отличает RL от ML. Вам нужно решить, как именно агент принимает эти решения. Позвольте мне начать с простого решения, а затем расширить его.

Используя тот же пример кнопки веб-сайта, предположим, что вы уже решили, что агент получает фиксированное вознаграждение за каждое нажатие кнопки.

Вы могли бы показывать новую версию веб-сайта в 50% случаев. Позже вы можете суммировать вознаграждение (количество кликов), чтобы определить, какая кнопка вам нужна. Эту идею реализует тестирование A/B. Но у данного решения есть серьезный недостаток: 50% клиентов видят неоптимальную кнопку. Ваша компания потеряет продажи.

При другом подходе вы могли бы для начала показывать новую версию в 50% случаев, но со временем сместить разделение в сторону предпочтительного варианта. Эта идея лежит в основе алгоритмов бандита.

Представленная стратегия подчеркивает важную проблему в RL. Вам нужно несколько раз просмотреть все награды, прежде чем вы поймете, что лучше. Но нужно также, чтобы агент использовал самые известные действия, чтобы вы могли максимизировать вознаграждение. Теперь это компромисс между *разведкой* и *эксплуатацией.*

Обобщенная версия этого алгоритма отводит некоторое время и на разведку, и на эксплуатацию. При каждом действии существует некая вероятность того, что действие будет случайным выбором, заданным параметром $\varepsilon$. Это называется $\varepsilon$-жадным алгоритмом, который показан в алгоритме 2.1. Смысл $r^{avg}$ и $N$ тот же, что и раньше.

**Алгоритм 2.1. $\varepsilon$-жадный алгоритм бандита**

1: **входные данные (input)**: вероятность разведки $0 \le \varepsilon \le 1$.

2: инициализация: $r^{avg}(a) \leftarrow 0$, $N(a) \leftarrow 0$, для каждого $a \in \mathcal{A}$.

3: **цикл (loop)** вечный:

4: $a \leftarrow \begin{cases} \arg\max_{a_s \in \mathcal{A}(s)} r^{avg}(a_s) & \text{с вероятностью } 1-\varepsilon, \text{ разрыв связей случайным образом;} \\ \text{случайное } a & \text{с вероятностью } \varepsilon. \end{cases}$

5: представить действие $a$ в среде и получить вознаграждение $r$.

6: $N(a) \leftarrow N(a) + 1$.

7: $r^{avg}(a) \leftarrow r^{avg}(a) + \frac{1}{N(a)}\left[r - r^{avg}(a)\right]$.

Шаг 1 гласит, что вам нужно передать значение $\varepsilon$ между нулем и единицей. Более высокие значения $\varepsilon$ позволят выполнить разведку. Более низкие значения — эксплуатировать. Шаг 2 создает массивы для хранения результатов. Шаг 3 говорит вам, что цикл будет длиться вечно.

Алгоритм становится интересным на шаге 4. С вероятностью $1-\varepsilon$ агент должен выбрать действие, которое дает текущее наивысшее среднее вознаграждение. Аналогично агент должен выбрать случайное действие с вероятностью $\varepsilon$.

Шаг 5 запускает выбранное действие в среде и возвращает вознаграждение, которое я прокомментирую вкратце. Например, если алгоритм выбрал красную кнопку, ваш пользовательский интерфейс должен представить ее и сообщить вам, нажал ли ее пользователь. Это взаимодействие с окружающей средой.

Шаг 6 увеличивает счетчик для этого действия для последующего использования. И, наконец, шаг 7 обновляет функцию среднего вознаграждения в онлайн-режиме (см. уравнение 2.2) на основе мгновенного вознаграждения, действия и количества раз, когда агент выбирал это действие. Со временем эта оценка стоимости станет более точной, если предположить, что вознаграждение не изменится.

Алгоритм 2.1 — это алгоритм бандита, названный в честь однооруких игровых автоматов. Со временем этот алгоритм позволяет автоматически выбирать "руку" на игровом автомате-бандите, которая обеспечивает наивысшую награду. Если вы хотите выполнить автоматическое тестирование бандита, процесс сравнения двух или более конкурирующих решений, алгоритм 2.1 позволяет автоматически выбирать версию, которая приносит наибольшее вознаграждение.

## Моделирование среды

Когда вы работаете над промышленным проектом, лучшая среда — это реальная жизнь. Однако ваша среда может быть слишком дорогой, опасной или сложной для разработки. Вместо этого вы можете создать симуляцию, которая будет упрощенной версией предметной области (например, трехмерный игровой движок) или смоделирована на основе собранных данных.

Представьте, что условные посетители веб-сайта нажимают кнопку не каждый раз, а лишь в некоторые моменты времени. Кроме того, они предпочитают красную кнопку зеленой кнопке. В этом примере я предполагаю, что колоссальные 40% пользователей нажимают красную кнопку, но только 5% пользователей нажимают зеленую. Но эти значения могут быть получены из предыдущих экспериментов или опубликованной модели. Вы можете использовать эту информацию для создания простого симулятора для проверки вашего алгоритма RL, как показано в алгоритме 2.2.

**Алгоритм 2.2. Симулятор веб-сайта**

**function**: ENVIRONMENT(a, p(a))

**входные данные (input)**: действие $a$ и вероятность $p(a)$.

**выходные данные (output)**: $r \leftarrow 1$ с вероятностью $p(a)$, иначе 0.

Я знаю, что это просто, но я подумал, что было бы полезно пояснить, что моделирование не обязательно должно быть сложным, чтобы стать полезным. Оно принимает (input) действие и вероятность действия и выводит (output) вознаграждение в размере 1 с вероятностью $p(a)$. Здесь я предоставляю вознаграждение в размере 1, но вы можете использовать значение, которое имеет больше смысла в вашем контексте, например общую стоимость продажи (в этом случае вам также потребуется смоделировать значения заказов).

## Запуск эксперимента

Теперь вы можете запустить свой первый эксперимент. На рис. 2.1 проиллюстрирован рабочий процесс типичного алгоритма бандита, реализованного на веб-сайте. Это изображение может отличаться от того, что вы ожидаете, потому что я специально включаю следующие компоненты RL: агента, действие и вознаграждение. Сначала пользователь переходит по вашей ссылке, и агент решает, какую кнопку показывать. Затем пользователь просматривает страницу и может нажать на кнопку. Результат возвращается агенту, чтобы он мог узнать, какое действие выбрать в будущем.

Весь код доступен для онлайн-использования на специально созданном веб-сайте (дополнительную информацию *см. в разд. "Дополнительные материалы" в предисловии*)

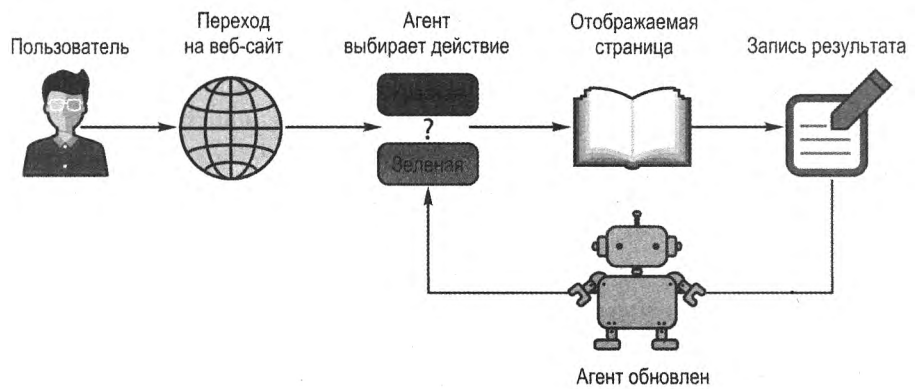



**Рис. 2.1.** Процесс тестирования работы бандитского алгоритма

Я реализовал алгоритм 2.2, используя два распределения Бернулли, которые представляют две кнопки с вероятностью выбора 40% для красной кнопки и 5% для зеленой. Затем я реализовал алгоритм 2.1, чтобы узнать, какая кнопка была выбрана этими моделируемыми пользователями автоматически.

На рис. 2.2 сравниваются три различных значения гиперпараметра $\varepsilon$, которые я передаю в алгоритм 2.1. Каждое моделирование выполнялось в течение 250 шагов, т. е. алгоритм имел 250 возможностей изучить основные цветовые предпочтения

клиентов (моделируемый клиент предпочитает красную кнопку) и извлечь максимальную ценность. Каждая строка отражает процент времени, в течение которого агент выбирал оптимальную кнопку для показа посетителю, в среднем более 1000 повторов, чтобы уменьшить шум из-за случайной выборки.

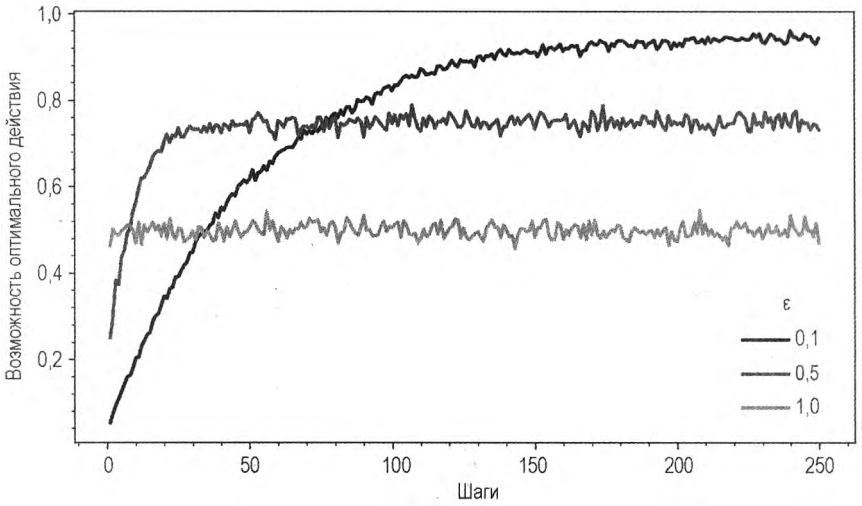



**Рис. 2.2.** Эффект изменения гиперпараметра ε, доля времени, в течение которого нужно, чтобы агент исследовал. Высокие значения ε приводят к большему исследованию, но меньшему использованию оптимального действия, и наоборот

Глядя на начало эксперимента, вы можете видеть, что для каждого запуска требовалось некоторое время, чтобы выучить оптимальную кнопку. Скорость этого обучения зависит от следующих факторов. Когда ε велико, агент бóльшую часть времени выбирает случайные действия, и у агента есть много возможностей наткнуться на лучшее действие. Когда ε мало, у агента меньше возможностей для изучения, и вам придется долго ждать. Другими словами, ε влияет на скорость обучения.

Если смотреть ближе к концу эксперимента, то можно увидеть, что различные значения ε влияют на конечный оптимальный процент вознаграждения. Я говорю "конечный", потому что каждая кривая является асимптотической, что означает стремление к некоторому конечному значению. Представьте, что вы являетесь алгоритмом с числом ε, равным 1. Это означает, что вы всегда выбираете случайное действие (показать случайную кнопку). В этом случае максимальная награда, которую вы когда-либо могли получить, составляет 50%. Если вы начнете уменьшать ε, то в какой-то момент сможете выбрать оптимальное действие. Асимптота оптимального действия равна $1/n+(1-\varepsilon)/n=(2-\varepsilon)/n$, где $n$ — количество действий. Когда $\varepsilon=1$, агент всегда выбирает случайное действие, и результатом является оптимальное действие 0,5. Когда $\varepsilon=0{,}1$, результатом является оптимальное действие 0,95. ε влияет на конечную скорость, с которой алгоритм выбирает оптимальное действие.

## Улучшение ε-жадного алгоритма

Все алгоритмы принципиально ограничены тем, насколько эффективно они исследуют и как быстро начинают использовать усвоенное. Лучшие алгоритмы способны выполнять и то и другое одновременно, но определение того, как именно исследовать, в значительной степени зависит от проблемы. Например, лабиринт и оптимальный график медицинского обслуживания потребовали бы различных стратегий исследования.

В предыдущем разделе было показано, что гиперпараметр ε контролирует этот компромисс. Если он (параметр) слишком высокий, тогда агент ищет активно, что приводит к выбору неправильных действий. Слишком низкий, и агент мало ищет и тратит много времени на поиск новых оптимальных действий.

Еще парочка методов исследования приведена в *разд. "Дополнительные материалы для чтения" в конце данной главы.*

Обсуждая этот вопрос, также важно обращать внимание на саму симуляцию. Если разница между вознаграждениями за два действия невелика, агенту приходится часто пробовать два результата. Согласно закону больших чисел уверенность агента в вознаграждении уменьшается с увеличением количества наблюдений.

Часто бывает лучше выбрать действие, основанное на текущих оценках распределения вознаграждений. Другими словами, вместо того чтобы возвращать одно действие, агент возвращает вероятности каждого действия, взвешенные по ожидаемым вознаграждениям каждого состояния. Это называется *функцией softmax* и обеспечивает естественную функцию исследования, определяемую текущими оценками вознаграждения.

Если конечная цель состоит в том, чтобы получить как можно больше вознаграждения, то нет смысла продолжать исследования. Вы могли бы удалить ε -жадное действие. Но чаще всего значение ε со временем уменьшается. Это называется *отжигом*. Слово "отжиг" — металлургический термин. Он означает медленное нагревание и охлаждение металлов, усиление и снятие нагрузки.

Третье популярное усовершенствование вращается вокруг идеи о том, что нет смысла исследовать случайным образом, особенно в простых симуляциях. Агент узнает больше, исследуя состояния, которые он раньше не видел. Эти алгоритмы добавляют бонус к каждому действию для состояний с неадекватной выборкой и называются *методами с верхним доверительным интервалом* (upper-confidence-bound, UCB). Алгоритмы UCB полезны, потому что у них нет гиперпараметров, таких как ε (или скорость, с которой ε уменьшается, для отжига).

На рис. 2.3 сравниваются эти три метода. Я выбрал значение ε, чтобы приблизить конечную производительность к величинам, характерным для других методов. Вы можете видеть, что стратегии отжига softmax и UCB изучают оптимальное дейст-

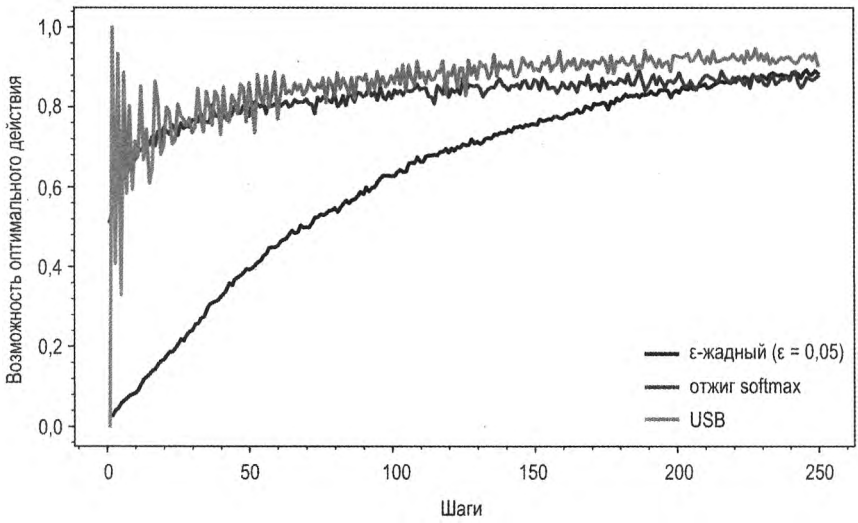



**Рис. 2.3.** Сравнение трех распространенных стратегий исследования: ε-жадность, отжиг softmax и UCB

вие гораздо быстрее, чем метод ε-жадности. UCB также выигрывает от отсутствия настраиваемых параметров.

Увеличение скорости обучения алгоритмов является общей целью новых алгоритмов RL.

# Марковские процессы принятия решений

*Марковский процесс принятия решений* (Markov decision process, MDP) является математическим формализмом многих идей, представленных в *главе 1. Агент* (agent) влияет на наблюдаемое поведение стохастической системы (*среды* — environment), выбирая действия. Цель состоит в том, чтобы выбрать набор действий, которые позволяют системе вести себя оптимальным образом, как это определено некоторым критерием успеха (*вознаграждением* — reward).

Агент никогда не сможет в полной мере оценить окружающую среду. Он получает ограниченный набор *наблюдений*, которые представляют текущее *состояние* системы. Эти два слова означают именно то, что они есть. *Состояние* (state) — это условие, в котором находится окружающая среда в определенное время. *Наблюдение* (observation) — скрытый взгляд на это состояние. Во многих средах состояние можно наблюдать напрямую.

Цифровые агенты выполняют действия в дискретное время. Это рождает идею о *шаге* во времени. Обратите внимание, что это не препятствует использованию непрерывных действий, таких как угол поворота рулевого колеса. Противополож-

ность цифровому агенту — аналоговый агент, очень похожий на систему с механическим демпфированием — технически возможен, но я никогда его не видел. Может быть, кто-то попробует создать такой хобби-проект?

Таким образом, агент и окружающая среда взаимодействуют в последовательности дискретных временны́х шагов $t$. На каждом временно́м шаге агент получает представление о состоянии $s$. Основываясь на состоянии окружающей среды, агент может предложить действие $a$ и через один временной шаг получить значение, представляющее вознаграждение $r$, и новый набор состояний $s'$. Здесь апостроф у буквы $s$ используется для обозначения следующего состояния. Вы часто будете видеть индексы с $t$ и $t+1$, которые означают одно и то же. На рис. 2.4 показаны эти элементы.

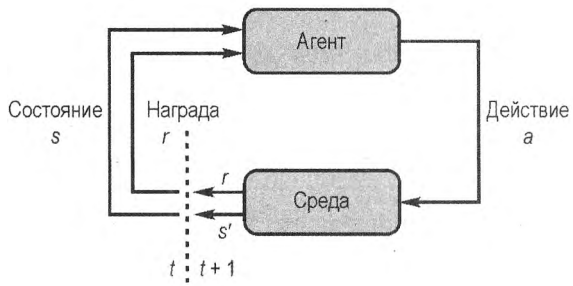



**Рис. 2.4.** Представление интерфейса и цикла обратной связи между средой и агентом в MDP. Агент принимает текущее состояние и вознаграждение от окружающей среды и на основе этой информации выполняет действие. Окружающая среда выполняет действие и создает новое вознаграждение и состояние для следующего временно́го шага

Помните, что состояния и вознаграждения — случайные величины, а не определенные значения, потому что большинство сред являются стохастическими. Например, ваш агент может выбрать поворот рулевого колеса в самоуправляемом автомобиле, но нет никакой гарантии, что автомобиль действительно повернет; это может быть на льду. Для этого используется концепция модели перехода. В некоторых книгах это также называется *динамикой* MDP. Модель перехода, показанная в уравнении 2.4, представляет собой вероятность перехода в новое состояние с вознаграждением за предыдущее состояние и выполненное действие. Обозначения были упрощены, чтобы подчеркнуть причинно-следственный эффект действия.

**Уравнение 2.4. Модель перехода MDP**

$$p(s', r \mid s, a).$$

Уравнение 2.4 является ключом к MDP. В марковском процессе принятия решений модель перехода описывает динамику среды. Следующее состояние и награда зависят только от предыдущего состояния и действия. И единственное различие между MDP и традиционной марковской цепью, которая также называется *марковским процессом*, заключается в добавлении параметров действия и вознаграждения. Действие — это *решение* (decision), которое выражено буквой "D" в MDP.

Мне нравится думать о MDP как об интерфейсе. Это контракт между агентом и средой. Агент может только наблюдать за состояниями и предлагать действия. Взамен среда даст вам новое состояние и некоторую ценность, которая отражает то, насколько хорошо вы справляетесь. Это может показаться жестким, но это обеспечивает замечательную гибкость с обеих сторон интерфейса. Среда может быть настолько сложной, насколько это необходимо. Это может быть симуляция нажатия пользователем кнопки, игра Atari или даже реальная жизнь. Агент также может быть настолько сложным, насколько это необходимо. Это может быть решение показать зеленую или красную кнопку, выбор двигаться влево или вправо в игре или принятие решения о том, стоит ли переезжать в сельскую местность.

Я использую апостроф, чтобы обозначить шаг вперед во времени. Но это время не обязательно должно иметь фиксированный интервал. Такая условность служит лишь для того, чтобы отличать действия друг от друга. Другими словами, как только вы чувствуете необходимость принять решение, именно тогда вы делаете шаг. Думайте о действиях как об исправлениях курса, а о времени — как о записи метки времени.

## Контроль запасов

Определение проблемы в терминах RL является общей проблемой для людей, которые имеют опыт работы в области науки о данных. Стоит набраться большего опыта на этапе определения проблемы, прежде чем переходить к другим способам решения MDP.

Контроль запасов — хороший пример для начала, потому что он одновременно полезен и прост для понимания. Этот пример имеет прямые аналогии с ценообразованием, оптимальной остановкой (продажей актива), обслуживанием и многим другим. Я начну с очень простого примера, а затем расширю его до более сложной версии.

Представьте, что у вас есть небольшой магазин. Он такой маленький, что у вас есть только один товар. Каждый день клиенты покупают ваш товар, поэтому вам нужно пополнять запасы. Если у вас закончатся запасы, вы не сможете продать свой продукт и заработать; вы не хотите, чтобы запасы заканчивались. Вы арендуете свой магазин, и это обходится вам в определенную сумму денег за квадратный метр. Вы не хотите хранить миллион товаров, потому что вам понадобится помещение большего размера. Я мог бы определить множество проблем высокого уровня, таких как оптимальный размер заказов или минимизация расходов на доставку, но для начала я определяю проблему как решение наилучшей точки для пополнения запасов.

Пусть $s$ представляет количество товаров на складе в определенный день. Таково состояние окружающей среды. В попытке сделать все как можно проще, я предполагаю три возможных состояния: нет запасов, один товар на складе или два товара на складе, $\mathcal{S} \doteq \{0, 1, 2\}$ (фигурный шрифт здесь означает, что он представляет множество).

Учитывая текущее состояние, агент может выполнить действие. Проще говоря, предположим, что агент может выполнить одно из двух действий. Либо пополнить

запасы — действие, которое заказывает 1 новый продукт, либо ничего не делать (None): $\mathscr{A} \doteq \{\text{пополнить запасы, None}\}$. Если текущее состояние было 1 и агент выполняет действие, то следующее состояние будет 2. Также предположим, что вы не можете пополнять запасы, когда все хранилища заняты.

Следующее, чего не хватает в уравнении 2.4, — это вероятности перехода. Какова вероятность того, что агент перейдет из одного состояния в другое? Поскольку вы моделируете окружающую среду, вы должны выбрать эти вероятности, но они могут быть получены из наблюдений за вашим магазином. Предположим, что вероятность одной продажи (sale) в течение дня $p(\text{sale})$ равна 0,7. В будущем вы могли бы предсказать количество проданных товаров. Но сейчас я снова использую распределение Бернулли. Это означает, что в каждом состоянии вероятность продажи составляет 70%, а вероятность отсутствия продажи — 30%.

Это чисто гипотетический пример. Конечно, для пополнения запасов в реальной жизни требуется время Упрощения являются необходимой частью проектирования, особенно на ранних стадиях разработки. Как только у вас будет рабочий прототип, вы сможете поработать над устранением некоторых из этих упрощений.

Последнее, что вам нужно, — это награда $r$. Вашему магазину нужно зарабатывать деньги, чтобы вы могли вознаграждать себя каждый раз, когда продаете товар. Но продажи возможны только в том случае, если товар в данный момент имеется на складе или если вы пополняете запасы вовремя. Это означает, что вознаграждение зависит от текущего состояния и продаж; отсутствие продаж означает отсутствие вознаграждения. Вы можете увидеть это определение в математической форме в уравнении 2.5.

**Уравнение 2.5. Определение награды в задаче контроля запасов**

$$r \doteq \begin{cases} 1, \text{ если } s > 0 \text{ и имеются продажи (sale);} \\ 1, \text{ если } a = \text{пополнить запасы и имеются продажи (sale);} \\ 0, \text{ в других случаях.} \end{cases}$$

## Таблица переходов

Следующая задача — определить все возможные переходы состояний в системе, и вы можете сделать это тремя разными способами. Первый способ — представить все возможные комбинации в виде таблицы, известной как *таблица переходов* (табл. 2.1).

В табл. 2.1 есть строка для каждой возможной комбинации текущего состояния $s$, действия $a$, следующего состояния $s'$ и вероятности перехода в состояние, заданное предыдущим состоянием и действием, $p(s', r \mid s, a)$; вознаграждение за этот переход находится в конечном столбце $r$. Вы можете использовать эту таблицу для отслеживания *траектории* агента.

***Таблица 2.1.*** *Таблица переходов MDP*

| $s$ | $a$ | $s'$ | $p(s', r \mid s, a)$ | $r$ |
|---|---|---|---|---|
| 0 | None | 0 | 1 – $p$(sale) | 0 |
| 0 | None | 0 | $p$(sale) | 0 |
| 0 | Пополнить запасы | 1 | 1 – $p$(sale) | 0 |
| 0 | Пополнить запасы | 0 | $p$(sale) | 1 |
| 1 | None | 1 | 1 – $p$(sale) | 0 |
| 1 | None | 0 | $p$(sale) | 1 |
| 1 | Пополнить запасы | 2 | 1 – $p$(sale) | 0 |
| 1 | Пополнить запасы | 1 | $p$(sale) | 1 |
| 2 | None | 2 | 1 – $p$(sale) | 0 |
| 2 | None | 1 | $p$(sale) | 1 |
| 2 | Пополнить запасы | 2 | $p$(sale) | 1 |

Состояние 2 имеет только три возможных перехода, потому что я отказался от окончательного пополнения запасов без возможности продажи. Я считаю, что пополнение запасов, когда они полны, невозможно.

## Граф переходов

Другой способ представления этой информации — через ориентированный граф, известный как *граф переходов* (рис. 2.5).

Граф имеет два типа узлов (называемых *вершинами*): один для состояния, представленного кругом с именем этого состояния внутри, и другой для действия, представлен закрашенным кругом. Они соединяются ориентированными дугами (называемыми *ребрами*), помеченными вероятностью перехода по этим дугам после действия (*вероятность перехода*) и вознаграждением, полученным после перехода по дуге.

## Матрица переходов

Окончательный способ представления вероятностей перехода — это матрица для каждого действия, известная как *матрица переходов*. Вы можете построить матрицу таким же образом, как в табл. 2.1.

$$p(s',\ a=\text{None}) = \begin{bmatrix} 1 & 0 & 0 \\ p(\text{sale}) & 1-p(\text{sale}) & 0 \\ 0 & p(\text{sale}) & 1-p(\text{sale}) \end{bmatrix},$$

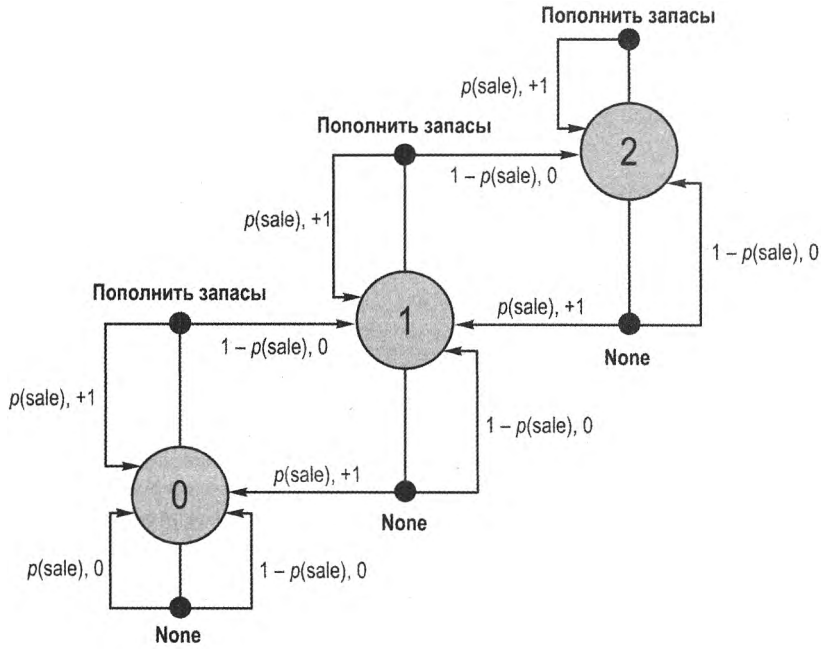



**Рис. 2.5.** Граф, представляющий вероятности перехода простой задачи пополнения запасов

$$p(s',\ a = \text{Пополнить запас}) = \begin{bmatrix} p(\text{sale}) & 1-p(\text{sale}) & 0 \\ 0 & p(\text{sale}) & 1-p(\text{sale}) \\ 0 & 0 & 1 \end{bmatrix}.$$

Каждая ячейка представляет вероятность перехода из текущего состояния (0, 1 или 2 в строках) в новое состояние (0, 1 или 2 в столбцах). Обратите внимание, что сумма каждой строки должна равняться 1. Например, если агент решает не пополнять запасы, действие $a$ — None (ничего не делать), вы используете верхнюю матрицу перехода. Если в магазине есть один товар на складе, состояние равно 1 (вторая строка), вы можете видеть, что существует вероятность $p$(sale) перехода в состояние 0, вы продали товар, и 1 – $p$(sale) пребывания в том же состоянии после отсутствия продажи. Также обратите внимание, что вам еще нужна отдельная функция или матрица для полученных вознаграждений.

Я предпочитаю граф переходов, потому что он позволяет вам четко проследить путь, пройденный агентом. Но вы можете построить граф только до небольшого числа состояний и действий. Вычислительная мощность — единственное, что ограничивает матрицу переходов, но ее трудно интерпретировать на глаз.

## Симуляция управления запасами

В предыдущих разделах были представлены все компоненты, необходимые для создания симуляции (среды). Но как насчет агента? Когда вам следует пополнить запасы?

Вы уже установили ограничение, что не можете пополнить запасы, если склад заполнен. Это одно из жестких правил. Но что лучше всего предпринять после этого? Если вы посмотрите исключительно на награды, которые представляют собой цель, то обнаружите, что получите положительное вознаграждение только за совершение продаж. Отрицательного вознаграждения за размещение заказов не существует. Другими словами, согласно матрице вознаграждений хранение запасов ничего не стоит, и вы можете бесплатно пополнить запасы. Учитывая такую структуру вознаграждения, можно сделать предположение, что лучшая стратегия — продолжать заказывать как можно больше.

Несмотря на то что стратегия в этом случае очевидна, вы всегда должны проверять свою гипотезу, ведь у нас научный метод. Всегда полезно сравнить свою теорию с другими базовыми показателями.

На рис. 2.6 сравниваются три различные стратегии, или политики, пополнения запасов. Политика указывает агенту, какие действия следует предпринять, и в целом лучшие из них максимизируют общее вознаграждение. Политика заключается в том, чтобы пополнять запасы либо как можно больше, либо только тогда, когда уровень запасов падает до нуля, либо случайным образом. Линии на графике представляют собой совокупные вознаграждения за каждую из политик. Доля произведенных продаж представлена в скобках в условных единицах.

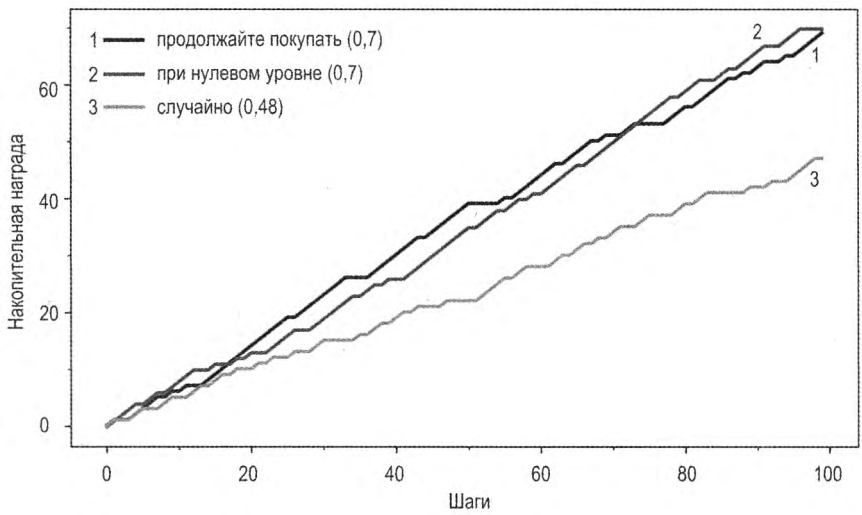



**Рис. 2.6.** Сравнение различных политик агентов для простой задачи пополнения запасов. Доля продаж, совершенных в день, указана в скобках.
0,7 — это максимум по определению проблемы

В этом случае ясно, что пополнение запасов случайным образом не является хорошей идеей, потому что довольно часто люди хотели купить товар, когда запасов не было. Это связано с тем, что вероятность продажи (0,7) была больше, чем вероятность пополнения запасов (0,5). Политика постоянного пополнения запасов позволяла не отставать от спроса, потому что вы заказывали новую партию товара на каждом временном шаге. Таким образом, уровень запасов (состояние) колебался между 2 и 1. При применении политики "повторный заказ при полном исчерпании" запасы балансировали на уровне 0–1, поскольку пополнение производилось, лишь когда запасов не оставалось совсем. По сути, между этими двумя показателями нет разницы в производительности, но обратите внимание, что вы можете считать политику "продолжайте покупать" расточительной, потому что вы используете больше места, чем вам нужно, или безопасной на случай, если возникнет спешка. Это зависит от вашей перспективы и далеко идущих целей.

Это интересный пример того, как одни и те же данные могут означать различные вещи для разных людей. Ваш начальник может возненавидеть остатки или ему может понравиться заначка. Ни одна из позиций не является неправильной.

Вы только что сформировали свой первый набор политик. И я еще не обсуждал, как научиться оптимальной политике — на это уйдет остальная часть книги. Но это важный результат, который необходимо учитывать; оптимальная политика зависит от всех компонентов MDP.

Изменение условий для включения большего количества запасов или предоставление людям возможности покупать более одного товара за раз влияет на выбор наилучшей стратегии пополнения запасов. Изменение функции вознаграждения путем введения штрафа за заказ, например плата за доставку, также изменяет лучшую стратегию. В этих случаях, возможно, имеет смысл подождать, пока у вас запас не обратится в ноль, а затем заказать две полные единицы товара. Функция вознаграждения определяет, какую проблему вы хотите решить, а не как ее достичь.

Стандартные MDP требуют, чтобы состояние было полностью наблюдаемым. Если агент не может наблюдать истинное состояние системы, то как он может выбрать оптимальное действие? Существует расширение модели MDP, называемое *частично наблюдаемыми MDP* (partially observable MDPs, POMDPs). Здесь вы можете ввести дополнительное сопоставление между наблюдениями и фактическим состоянием. Другими словами, агенту нужно будет научиться сопоставлять действия с наблюдениями, сопоставляемыми с состояниями, которые он не может наблюдать, — двойное сопоставление. Как вы можете себе представить, это сложно решить из-за большего количества неизвестных. Во многих приложениях инженеры возвращают систему к стандартному MDP, вводя дополнительную информацию с предыдущих временных шагов в текущий временной шаг. Знание прошлого может помочь агенту понять, как действия изменили воспринимаемые состояния в течение более длительного периода времени.

Я также хочу подчеркнуть, что здесь важно наблюдать за состоянием, а не за окружающей средой. Агенты не заботятся о том, что происходит за пределами интер-

фейса MDP. Вы можете изменять симуляцию столько, сколько захотите, до тех пор, пока не измените интерфейс. В примере с запасами изменение покупательского поведения клиента повлияет только на оптимальную политику. Это не меняет того, как агент учится.

Имейте в виду эти моменты, когда будете читать остальную часть этой книги.

# Политики и функции ценности

Политика похожа на стратегию. Например, вы не можете быть полностью уверены в том, что собирается делать другая футбольная команда (действия другой команды стохастичны), но вы можете аппроксимировать их действия и сказать своим игрокам, чтобы они действовали соответственно. Политика — это сопоставление состояний с потенциальными действиями. Именно то, как вы выполняете это сопоставление, является основным отличием между большим количеством алгоритмов RL. Но зачем проводить это сопоставление в первую очередь? И как вы можете оценить различные стратегии?

## Дисконтированные вознаграждения

Во многих задачах, включая примеры с пополнением запасов и кнопками веб-сайта, вознаграждение на каждом этапе легко понять. На каждом временном шаге агент может получать или не получать вознаграждение. Пройдя несколько этапов (в примере с запасами), агент может получить несколько вознаграждений. В целом задача состоит в том, чтобы максимизировать общее вознаграждение, которое вы ожидаете получить.

*Доходность* $G$ — это общая награда с текущего шага до последнего временного шага (или бесконечности). Первый знак "$\doteq$" можно интерпретировать как "я определяю $G$ как". Обратите внимание, что агент получает вознаграждение после перехода к следующему шагу. Большинство практических реализаций возвращают состояние и вознаграждение одновременно, поэтому они устанавливают начальное вознаграждение равным нулю.

Уравнение 2.6 суммирует все будущие вознаграждения вплоть до последнего временного шага $T$. Это конечное состояние называется *терминальным состоянием* и возникает только в *эпизодических* задачах. Эпизод представляет собой один полный проход через все этапы, которые заканчиваются в естественной точке завершения. Это может произойти, когда ваш робот упадет со скалы, или если ваш персонаж в игре умрет. Но, конечно, могут быть задачи, которые продолжаются вечно, например задача с пополнением запасов, если вы позволите мне оптимистично оценивать рынок сбыта. В этом случае конечное состояние $T = \infty$ и ожидаемая доходность также может быть бесконечной.

**Уравнение 2.6. Доходность**

$$G \doteq r + r' + ... + r_T .$$

Для того чтобы смягчить взрывную силу бесконечности, в уравнение 2.7 добавлен коэффициент дисконтирования, который экспоненциально уменьшает будущие вознаграждения.

**Уравнение 2.7. Дисконтированная доходность**

$$G \doteq r + \gamma r' + \gamma^2 r'' + \ldots = \sum_{k=0}^{T} \gamma^k r^{(k)} .$$

В уравнении 2.7 $T$ представляет конечное состояние и может быть бесконечностью. $\gamma$ называется *коэффициентом* или *рейтингом дисконтирования*. Он контролирует, как скоро грядущие вознаграждения будут игнорироваться, и должен иметь значение от 0 до 1 включительно. Другими словами, если $\gamma = 0$, то принимается во внимание только нынешнее вознаграждение. Если $\gamma = 1$, то мы получим уравнение 2.6. Как правило, требуется учитывать будущие награды, чтобы иметь возможность решать проблемы, когда награда остается в отдаленной перспективе. Но точные значения будут отличаться от ожидаемых в зависимости от рассматриваемой проблемы. В большинстве нетривиальных примеров коэффициент дисконтирования устанавливается в диапазоне от 0,9 до 0,99.

Поэтому вам предстоит создать агента, который может генерировать наибольшую *ожидаемую отдачу*. Один из вопросов, который у вас может возникнуть, заключается в том, как рассчитать ожидаемую прибыль, учитывая, что она зависит от будущих вознаграждений. Ответ заключается в повторении и изучении того, какие состояния и действия соответствуют лучшим наградам. Другими словами, вы должны повторять и повторять, чтобы найти наилучшую политику.

## Прогнозирование вознаграждений с помощью функции ценности состояния

В примере с пополнением запасов из *разд. "Симуляция управления запасами" ранее в данной главе* я произвольно выбрал три стратегии пополнения запасов. Однако мне хочется оптимизировать выбор действий, чтобы автоматически максимизировать ожидаемую отдачу. Это цель политики; она привязывает состояния к действиям. Формально это вероятность выбора действия при заданном состоянии $p(a|s)$, и обозначается $\pi$.

Политику можно улучшить, изменив ее градиент в отношении ожидаемой отдачи. Я вернусь к этому в *главе 5*.

Вы можете рассчитать доход от доходности $G$, ожидаемый от вашей текущей политики $\pi$, начиная с текущего состояния $s$. Это называется *функцией ценности состояния* (state-value function) для политики и показано в уравнении 2.8.

**Уравнение 2.8. Функция ценности состояния**

$$V_{\pi}(s) \doteq \mathbb{E}_{\pi}[G \mid s] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{T} \gamma^{k} r^{(k)} \mid s\right].$$

На первый взгляд это уравнение выглядит сложным, поэтому позвольте мне всё разъяснить. В уравнении три части, разделенные знаками равенства. Слева — $V$; это символ, используемый для обозначения функции ценности.

Для тех, кто не очень разбирается в математике, считаю, будет полезным представлять математические функции как программный код.

Вторая часть, определение функции, гласит, что функция ценности — это ожидаемое значение доходности, заданное текущим состоянием, в соответствии с политикой $\pi$. Символ $\mathbb{E}$ является оператором ожидания (подробнее об этом чуть позже). Таким образом, вы получаете указание рассчитать ожидаемое значение параметра в скобках. И индекс $\pi$ говорит о том, что агент должен следовать политике, чтобы рассчитать это ожидание.

Третья часть уравнения — справа — представляет собой расширенную запись члена в скобках. Из уравнения 2.7 вам известно, как определяется $G$. И вы знаете, что нужно начать с текущего состояния. На данный момент математическое выражение в скобках в общих чертах говорит: "суммируйте все награды с этого момента".

Уравнение 2.8 имеет два важных момента. Первый — это оператор ожидания. Большинство алгоритмов RL предполагают, что вознаграждения характеризуются нормальным распределением, и математическое ожидание может быть рассчитано как среднее значение выборки. Другими словами, результатом оператора ожидания является среднее дисконтированное вознаграждение от данного состояния. Вознаграждения в некоторых приложениях могут распределяться не по нормальному закону. Например, что бы вы выбрали: одну большую награду или много маленьких наград? В обеих ситуациях среднее значение одинаково, но распределение вознаграждений — нет. Нюансы вознаграждения могут быть скрыты оператором ожидания. На самом деле, в последнее время многие исследования были сосредоточены на попытках оценить и использовать вместо этого распределение вознаграждений (*см. разд. "Распределительное RL" главы 4*).

Вторая важная часть — это политика. Вы можете рассчитать ожидаемую доходность только в том случае, если у вас уже есть политика. Но это уравнение предназначено для того, чтобы помочь вам количественно оценить эффективность политики. Это классическая проблема с курицей и яйцом, которую RL решает, начиная случайным образом процесс и повторяя его, чтобы найти улучшение.

## Моделирование с использованием функции ценности состояния

Я считаю, что мое понимание уравнения возрастает с помощью простой симуляции. Я разработал программу, которая имитирует позиции на пяти квадратах (состояниях). Агент начинает с крайней левой позиции. На каждом временном шаге агент может двигаться влево или вправо. Моделирование заканчивается, если агент достигает целевого состояния квадрата 5 с правой стороны или если агент перемещается влево по квадрату 0 и падает с воображаемого обрыва. Для простоты политика является случайной. Так, она случайным образом выбирает движение либо влево, либо вправо на каждом временном шаге.

Вы должны постоянно учитывать различные условия. *Переобучение* — это процесс случайного построения модели или алгоритма, который работает с очень конкретными данными или обстоятельствами. Легко улучшить алгоритм до такой степени, чтобы он соответствовал вашей симуляции, и обнаружить, что, например, он не работает в реальной жизни. Промышленные алгоритмы RL должны быть надежными, и учет различных условий — один из способов помочь *обобщению* вашего алгоритма. Кроме того, использование различных сред является важным способом получения опыта. Поэтому я намеренно продолжаю менять среды на протяжении всех рассматриваемых примеров

Цель уравнения 2.8 состоит в том, чтобы рассчитать ожидаемое вознаграждение для каждого состояния. Вы можете сделать это, записывая положение агента в течение каждой *эпохи*, что представляет собой непрерывный цикл среды до тех пор, пока агент не достигнет конечного состояния. В конце концов, вы получите вознаграждение и сможете вернуться и посмотреть на все состояния, которые посетили по пути. Затем вы добавите вознаграждение к совокупной сумме и сохраните счетчик для расчета средней доходности, наблюдаемой после посещения всех этих состояний.

Всякий раз, когда вы проводите эксперимент, разумно представлять себе некоторые теоретические ожидания или предположения — определение гипотезы является основой научного метода.

Предположим, что вы находитесь на квадрате 1 с левой стороны. Представьте, как добраться до каждого конечного состояния. Вы можете передвинуться на один квадрат влево и упасть с обрыва. Или можете переместиться на четыре квадрата вправо и дойти до конца. Ваша первая политика является случайной; она случайным образом равномерно выбирает новое направление. Она случайным образом выбирает перемещение на один квадрат влево чаще, чем на четыре квадрата вправо. В среднем только 1 из 5 попыток позволяет достигнуть правой стороны; остальные приводят к падению с обрыва. Если вы установите вознаграждение равным 5 (чтобы оно соответствовало количеству квадратов) и не будете наказывать за падение с обрыва, то ожидаемая награда каждого состояния должна совпадать с положением квадрата.

В этом примере ожидаемый доход от квадрата 1 должен быть равен 1. Ожидаемое вознаграждение в конечном состоянии должно составлять 5, потому что нет необходимости двигаться. В промежуточных состояниях, поскольку агент движется случайным образом, все еще существует вероятность того, что он может случайным образом вернуться к обрыву. Таким образом, даже на четвертом квадрате, в одном шаге от награды, агент в среднем 1 раз из 5 будет скатываться с обрыва.

Если всё сказанное не очень понятно, представьте себе два квадрата, где одно движение вправо определяет состояние цели. Теперь увеличьте количество квадратов.

Обратите внимание, что в этом примере вы устанавливаете коэффициент дисконтирования равным 1. В более сложных средах вы должны установить это значение меньше 1, потому что ранние эпизодические перемещения, как правило, менее важны, чем те, которые близки к завершающему состоянию. Но точное значение зависит от вознаграждения и от того, является ли проблема постоянной или нет.

Когда я запускаю это моделирование, для стабилизации требуется большое количество итераций. После первой итерации агент сразу же прыгнул влево и погиб. Таким образом, ожидаемый возврат на итерации 0 равен [0,00; NaN; NaN; NaN], где позиции в этом массиве представляют квадраты с 1-го по 4-й. Квадрат 5 не записывается, потому что в терминальном (конечном) состоянии не требуется никаких действий. После десяти итераций агент пару раз достигал целевого состояния: [0,77; 3,00; 5,00; 5,00]. После тысячи итераций ожидаемая доходность начинает приближаться к предыдущей оценке: [1,04; 2,09; 2,93; 3,83]. После десяти тысяч случайных экспериментов результат очень близок: [1,00; 1,97; 2,92; 3,92]. Чем больше итераций я выполняю, тем ближе моделирование будет к истинному ожидаемому значению. Результат этого моделирования можно увидеть на рис. 2.7.

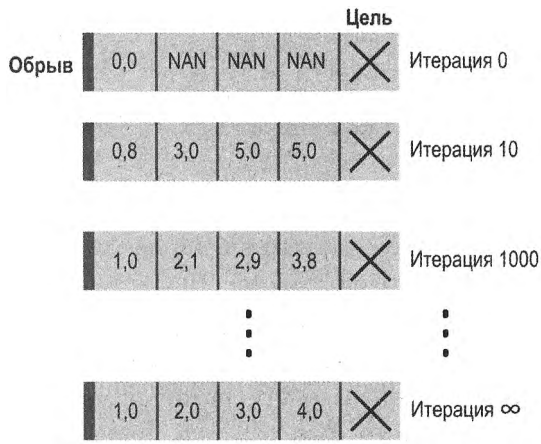



**Рис. 2.7.** Результат симулирования вычисления функции ценности состояния в простой среде. Каждый квадрат представляет состояние. Значение внутри каждого квадрата демонстрирует ожидаемую доходность, начиная с этого состояния и следуя политике случайности. После повторных итераций ожидаемая отдача от каждого состояния приближается к теоретическому ожиданию

Еще один интересный эксперимент, который стоит попробовать, — это изменение исходного положения. После десяти тысяч итераций результаты остаются теми же (в диапазоне случайных изменений). Вы можете попробовать добавить дисконтирование или наказать за падение с обрыва, чтобы посмотреть, что произойдет. Я призываю вас разработать собственный код, чтобы понять это.

## Прогнозирование вознаграждений с помощью функции ценности действия

В предыдущем примере показано, как предсказать ожидаемую доходность от данного состояния, которая представляет собой среднее вознаграждение, наблюдаемое после предыдущих посещений этого состояния. Но не все действия созданы равными. Что будет, если одно действие приблизит вас к вознаграждающим состояниям? Многие алгоритмы используют эту информацию, чтобы помочь решить, какое действие лучше выбрать. Это требует ожидаемой доходности от любого действия в любом состоянии и показано в уравнении 2.9.

**Уравнение 2.9. Функция ценности действия**

$$Q_\pi(s,a) \doteq \mathbb{E}_\pi[G \mid s,a] = \mathbb{E}_\pi\left[\sum_{k=0}^{T} \gamma^k r^{(k)} \mid s,a\right].$$

Уравнение 2.9 является *функцией ценности действия* (action-value function) для политики $\pi$. Видно, что это практически то же самое, что и уравнение 2.8, за исключением того факта, что у функции есть два параметра: состояние и действие. Представьте, что это реализовано в виде двумерной таблицы поиска для хранения вознаграждений за состояния и их действия. Для краткости эту функцию часто называют *Q-функцией*.

На рис. 2.8 показан результат симулирования из предыдущего раздела, но вместо этого здесь используется функция ценности действия. Уравнение 2.9 добавляет сохранение ожидаемой отдачи как для состояния, так и для действия. Одним из результатов является то, что перемещение влево от квадрата 0 всегда будет возвращать вознаграждение в размере нуля. Перемещение вправо от квадрата 4 всегда приведет к вознаграждению в размере 5,0.

Еще один интересный результат заключается в том, что ожидаемое значение для всех правильных действий выше, чем для соответствующих левых действий. Это происходит потому, что чем дальше агент движется вправо, тем больше шансов фактически достичь целевого состояния. Наконец, позвольте мне задать вам вопросы. Учитывая эти состояния, действия и ожидаемую отдачу, можете ли вы придумать политику или стратегию, оптимальную для данной среды? Можете ли вы представить себя агентом, принимающим решение на каждом квадрате? Как вы выбираете, в каком направлении двигаться?

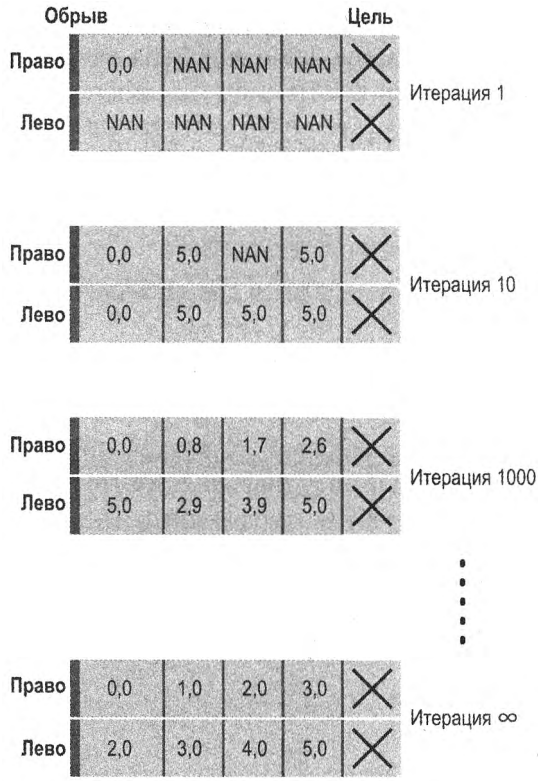



**Рис. 2.8.** Результат симулирования вычисления функции ценности действия в простой среде. Суть такая же, как на рис. 2.7. Разница здесь в том, что я записываю результат, действия и состояния

## Оптимальные политики

Оптимальная политика — это политика, при соблюдении которой генерируется максимально возможная ожидаемая доходность. Это означает, что существует по крайней мере одна политика, обеспечивающая наибольшую доходность, которую я обозначу $\pi_*$. Эквивалентными функциями ценности состояния и ценности действия в плане оптимальности являются $V_*(s)$ и $Q_*(s,a)$ соответственно.

Функции ценности состояния и ценности действия взаимосвязаны. Они обе представляли одну и ту же информацию, но на разных уровнях детализации. *Функция ценности состояния* — это ожидаемое вознаграждение, усредненное по всем действиям, что обеспечивает представление с низким разрешением о том, какие состояния высоко ценятся. *Функция ценности действия* обеспечивает представление с высоким разрешением о том, какие действия и состояния являются наилучшими.

Представьте, что у вас есть хорошие значения ожидаемых значений, например, потому что вы постоянно перемещаетесь по среде в течение длительного времени. Тогда оптимальная политика будет предлагать действия, которые приведут агента

к более высоким ожидаемым результатам. Другими словами, оптимальная политика — это та, которая выбирает действие с максимальной доходностью.

Большинство проблем RL сформулированы таким образом, чтобы максимизировать доходность, но это не является строго необходимым Однако я рекомендую придерживаться максимизации, чтобы избежать путаницы.

Теперь, когда у вас есть оптимальная политика, представьте, что вы снова вычисляете функции ценности состояния и ценности действия, следуя этой политике. Оба значения будут точно одинаковыми. Функция ценности состояния равна выбору наилучшего действия в функции ценности действия. Уравнение 2.10 представляет эту идею в математической форме и называется *уравнением оптимальности Беллмана.*

Официальное уравнение оптимальности Беллмана является более формальным, объединяющим вероятности перехода и вероятность того, что политика выберет действие. Для получения дополнительной информации *см. разд. "Дополнительные материалы для чтения" в конце данной главы.*

**Уравнение 2.10. Функция оптимальной ценности**

$$V_*(s) \doteq \arg\max_{a_s \in \mathcal{A}(s)} Q_{\pi_*}(s, a_s) =$$

$$= \arg\max_{a_s \in \mathcal{A}(s)} \mathbb{E}_{\pi_*}[G \mid s, a_s].$$

Уравнение 2.10 переиспользует определение функции ценности действия $Q$ из уравнения 2.9, выраженное через доходность $G$. В нем говорится, что выбор действия, которое максимизирует функцию ценности действия на каждом шаге, совпадает с функцией оптимальной ценности. Позвольте мне повторить это еще раз, потому что это важно. Вы получаете наибольшее общее вознаграждение, если неоднократно выбираете действие с наибольшей ожидаемой доходностью.

Поначалу это может показаться бессмысленным, потому что вы можете представить, что в других местах могут быть бóльшие награды. Но помните, что уравнения имеют полное представление обо всех будущих вознаграждениях (посмотрите на доходность $G$ в уравнениях). Таким образом, функция ценности действия (или функция ценности состояния) уже учитывает тот факт, что агент собирается посетить это вознаграждающее состояние в своих вычислениях.

Таким образом, отвечая на вопросы, заданные в конце предыдущего раздела, следует сказать, что лучше всего подходит та политика, которая неоднократно выбирает действие с наибольшим вознаграждением. В этом примере действие с наивысшей наградой всегда было на правой стороне, поэтому оптимальная политика заключается в том, чтобы каждый раз выбирать "движение вправо".

# Генерирование политики Монте-Карло

В примере из *разд. "Моделирование с использованием функции ценности состояния" ранее в данной главе* произвольно выбраны действия и записано вознаграждение, но вам так делать необязательно. Вместо этого вы можете выполнить исчерпывающий поиск всех возможных перестановок ходов и сохранить траекторию. Тогда оптимальной политикой является траектория, по которой достигается наивысшее вознаграждение. Но эта стратегия в большинстве случаев непрактична. Даже для простого примера на рис. 2.7 вы можете представить стратегию, которая многократно выбирает вправо, затем влево и никогда не заканчивается. Вы могли бы помочь агенту, ограничив исчерпывающий поиск, но в целом вам следует использовать что-то другое.

Формально проблема кроется в высокой размерности задачи. Основным алгоритмом, который с радостью решает задачи с многомерной выборкой, является *метод Монте-Карло* (Monte Carlo, MC). Представьте себе среду черного ящика, показанную на рис. 2.9, о функционировании которой у вас нет никаких сведений. Вы управляете действиями, представленными среде, поэтому знаете распределение входных данных. Но вы понятия не имеете о распределении выходных данных, потому что среда вам не подконтрольна. Характеристика распределения выходных данных помогает вам выбрать действие, которое обеспечивает наибольшую доходность (в среднем).

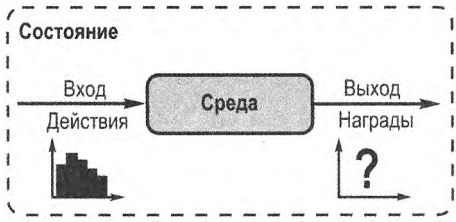



**Рис. 2.9.** Описание черного ящика техники Монте-Карло. Входы известны, выходы — нет. Случайным образом стимулируя входы, вы можете наблюдать и характеризовать выход

Методы Монте-Карло произвольно отбирают входные данные, чтобы вы могли наблюдать за тем, что происходит на выходе. После адекватных действий по выборке вы можете охарактеризовать результат и, следовательно, определить вознаграждение. Такой подход широко используется для обозначения любого метода получения значения, включающего существенную случайную составляющую. Но в RL методы Монте-Карло означают алгоритмы, которые усредняют вознаграждение, возвращаемое по всем траекториям. Вы можете узнать больше о методах MC *в разд. "Дополнительные материалы для чтения" в конце данной главы*.

Теперь я могу превратить представленную идею в алгоритм, который сначала генерирует всю траекторию, а затем обновляет функцию ценности действия в соответствии с наблюдаемым вознаграждением. Со временем политика будет стремиться к оптимальной. Алгоритм 2.3 называется *алгоритмом Монте-Карло на основе по-*

*литики*, потому что он, во-первых, выбирает полные траектории и, во-вторых, оценивает ценность политики и использует ее для одновременного выбора действий.

**Алгоритм 2.3. Алгоритм Монте-Карло на основе политики**

1: **input**: функция политики $\pi(a \mid s, Q_\pi(s,a))$.

2: Инициализировать $Q(s,a) \leftarrow 0$ для всех $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$,

$Returns(s,a) \leftarrow [\,]$ для всех $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$.

3: **loop**:

4: Сгенерировать полную траекторию эпизода, следуя политике $\pi$.

5: Инициализировать $G \leftarrow 0$.

6: **loop** для каждого шага в эпизоде, $t \doteq T-1,\ T-2,\ ...,\ t_0$:

7: $G \leftarrow \gamma G + r$

8: **if** $(s,a)$ not in $(s_0,a_0),(s_1,a_1),...,(s_{t-1},a_{t-1})$:

9: добавить $G$ к $Returns(s,a) \leftarrow [\,]$

10: $Q(s,a) \leftarrow average(Returns(s,a))$

Алгоритм 2.3 начинается на шаге 1 с передачи реализации политики. Этот алгоритм обычно представляют со встроенной политикой, но я считаю, что проще игнорировать детали политики и вместо этого вводить ее в качестве зависимости. Политика $\pi(a \mid s, Q_\pi(s,a))$ возвращает действие для данного состояния, используя функцию ценности действия, заполненную в этом алгоритме. Шаг 2 инициализирует функцию ценности действия и создает буфер для всех взаимодействий со средой.

Шаг 4 создает траекторию для полного эпизода, используя текущую политику. В своем коде я создал политику, которая выбирает действие с наибольшей ожидаемой доходностью, как в уравнении 2.10, с помощью $\varepsilon$-жадного поиска.

Шаг 6 повторяет каждый шаг в эпизоде в обратном порядке. Это важное различие; агент выбирает действия, основанные на *будущих* вознаграждениях. Итак, шаг 6 начинается в конце и повторяется в обратном порядке, суммируя награды по ходу. В итоге он окажется в первом состоянии, и тогда у вас будет один прогноз ожидаемой доходности от этого состояния.

Шаг 7 добавляет дисконтирование вознаграждения к ожидаемой доходности из уравнения 2.7. Если текущая пара "состояние — действие" не была посещена ранее в траектории [шаг 8], то шаг 9 добавляет пару в буфер. Эта реализация *первого визита* оценивает $V_\pi(s)$ как среднее значение доходности от первого визита в состояние. Реализации с *каждым посещением* оценивают $V_\pi(s)$ как среднюю доходность от каждого посещения состояния. Обе реализации приводят к одной и той же

оценке, но исследователи более подробно изучили алгоритм первого посещения и обнаружили, что он чуть менее ошибочен (значения действий при каждом посещении более зашумлены — *см. "Дополнительные материалы для чтения" в конце этой главы*, чтобы узнать больше).

Наконец, на шаге 10 обновляется текущее предсказание "состояние — действие", что обеспечивает политику ожидаемой доходности для любой пары "состояние — действие". Эта оценка представляет собой среднюю дисконтированную доходность, наблюдаемую по всем траекториям. Использование среднего значения позволяет предположить, что ожидаемая доходность распределяется нормально; возможно, это не так, но данное предположение хорошо работает на практике.

Результаты алгоритма 2.3 такие же, как на рис. 2.8, если вы установите параметр ε-жадности только для исследования, а коэффициент дисконтирования равен 1.

## Итерация по ценности с динамическим программированием

В предыдущих разделах я передавал вознаграждение назад по траектории, когда достигал конца эпизода. Но есть и другой способ. Представьте себе алгоритм, который сканирует на шаг вперед. Учитывая состояние, агент может заглянуть на шаг вперед, чтобы увидеть следующую ожидаемую доходность. В средах, где терминальное состояние (Term) обеспечивает вознаграждение, как показано на рис. 2.10,

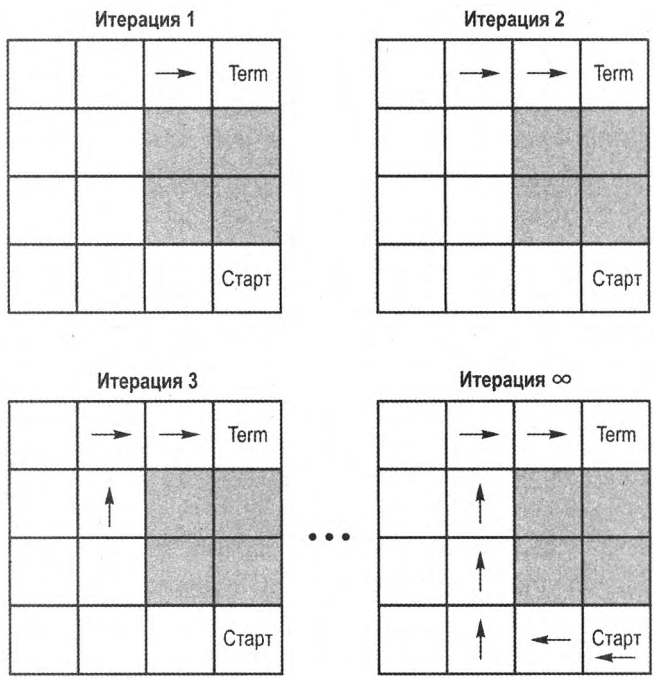



**Рис. 2.10.** Политика, генерируемая при просмотре одного состояния вперед на разных итерациях алгоритма

на итерации 1 агент будет смотреть на одно состояние вперед и видеть терминальное состояние, поэтому он знает, какое действие предпринять. На итерации 2 агент видит состояние до терминального и вновь знает, какое действие следует предпринять. Это продолжается до тех пор, пока политика не стабилизируется.

В литературе часто на основе этого процесса создаются две абстракции: оценка политики и совершенствование политики. *Оценка политики* обеспечивает оценку ожидаемого значения. *Совершенствование политики* обновляет политику, чтобы использовать новые знания, полученные в результате вычисления функции ценности, как новый многообещающий путь для изучения.

Объединение оценки и улучшения называется *обобщенной итерацией по стратегии* (generalized policy iteration, GPI). Реализации различаются в зависимости от того, на каком уровне вы чередуете улучшение и оценку. Например, вы можете подождать до конца эпизода, чтобы выполнить улучшение с помощью методов Монте-Карло, как в *разд. "Генерирование политики Монте-Карло" ранее в данной главе.*

В качестве альтернативы вы можете обновлять свои значения после каждого шага. Динамические методы программирования смотрят на шаг вперед и повторяют все действия. Сравните это с методами Монте-Карло, которые выбирают одно действие, но должны имитировать полный эпизод.

На рис. 2.11 проиллюстрирована разница с помощью так называемой *схемы резервного копирования* для функции ценности состояния. Светлые круги представляют

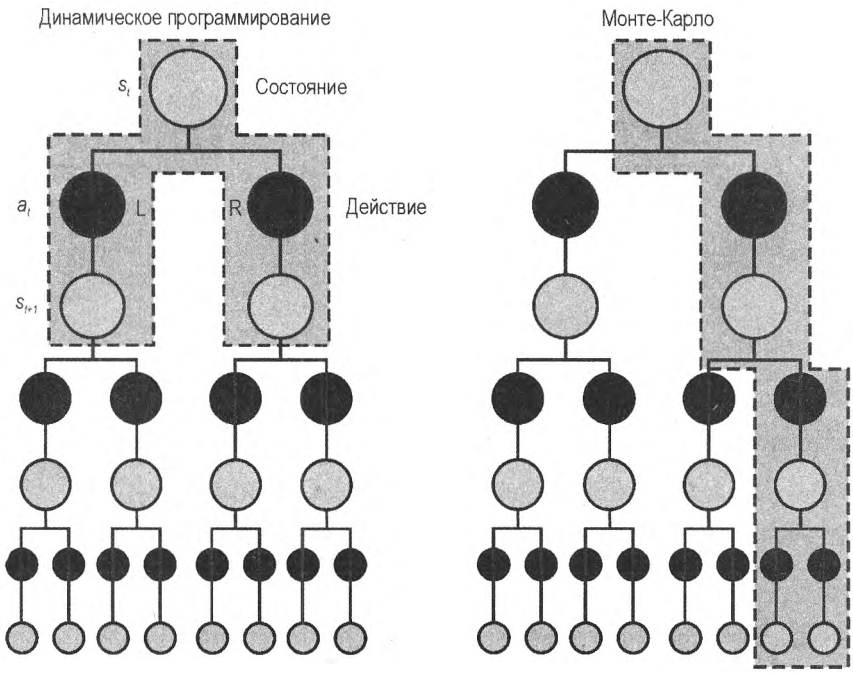



**Рис. 2.11.** Схема резервного копирования, показывающая разницу между методами динамического программирования и обновления политики Монте-Карло (для функции ценности состояния)

состояния, а темные круги — действия. В динамическом программировании в данном состоянии выполняется поиск по всем возможным действиям. Это позволяет обновлять политику после одного шага. Метод Монте-Карло сначала создает полный эпизод, следуя политике, а затем обновляет политику.

## Реализация итерации по ценности

Метод, называемый *итерацией по ценности* (value iteration), реализует динамическое программирование, но есть проблема. Итерация значения зависит от полного знания вероятностей перехода; вам нужна модель вашей среды, ведь агенту нужно знать, какие действия ему разрешено пробовать. Например, в двумерной сетке вам нужно сообщить агенту, что он не может выйти за пределы границ, установив эти вероятности равными нулю.

Если вы контролируете дизайн среды и проблема достаточно проста, можете заранее указать вероятности перехода, как в примерах из *разд. "Контроль запасов"* или *"Прогнозирование вознаграждений с помощью функции ценности состояния" ранее в этой главе*. В *главе 3* показана разработка общих алгоритмов для использования, когда у вас отсутствует полная модель среды. А пока предположим, что вы это делаете так, как показано в алгоритме 2.4.

**Алгоритм 2.4. Итерация по ценности для создания оптимальной политики**

1: **input**: вероятности перехода $p(s', r \mid s, a)$, порог остановки $\theta > 0$.

2: Инициализировать $V(s)$ для всех $s \in \mathcal{S}, \nabla \leftarrow 0$.

3: **do**:

4: **loop** для каждого $s \in \mathcal{S}$:

5: $v \leftarrow V(s)$

6: $V(s) \leftarrow \max_a \sum_{s', r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$

7: $\nabla \leftarrow \max\left(\nabla, \left|v - V(s)\right|\right)$

8: **while** $\nabla > \theta$

9: **output**: оптимальная политика $\pi(s) \doteq \arg\max_{a_s \in \mathcal{A}(s)} \sum_{s', r} p(s', r \mid s, a_s)\left[r + \gamma V(s')\right]$.

На шаге 1 алгоритма 2.4 необходимо указать вероятности перехода и порог остановки. В среде игрушечной сетки, например, вероятности перехода равны для всех действий, за исключением тех, которые выходят за пределы сетки или блокируются стеной, у которых вероятность перехода равна нулю. Порог остановки определяет, как долго нужно уточнять оценку значения.

Шаг 2 инициализирует буфер значений состояния для всех состояний и переменную для поддержания текущего количества ошибок. На шаге 3 алгоритм входит

в цикл do-while, а шаг 4 перебирает все состояния в среде. На шаге 5 сохраняется локальная копия значения состояния для текущего состояния.

На шаге 6 происходит действие. Начиная с левой стороны, алгоритм находит наибольшую ожидаемую награду после попытки всех действий. Происходит суммирование по всем потенциальным следующим состояниям и наградам за действие (потому что, даже если вы выберете действие слева, можете в итоге пойти дальше). Тогда вероятность представляет собой шансы оказаться в следующем состоянии с наградой с учетом текущего состояния и действия. И наконец, ценность этой серии событий — это награда за действие плюс ценность следующего состояния.

Я рекомендую вам перечитать алгоритм и описание еще раз, потому что шаг 6 позволит сократить запись, выразив многое в одной строке. Теперь, когда вы его перечитали, позвольте мне привести пример ручной работы, который я визуализировал на рис. 2.12.

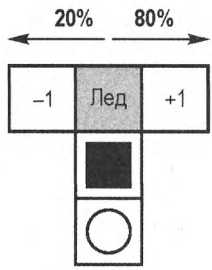



**Рис. 2.12.** Стилизованный пример, демонстрирующий обновление итерации по ценности. Квадрат представляет ваше текущее положение. Ценности представляют собой награды. Ожидаемое значение для текущего состояния — 0,6

Представьте, что вы стоите на сетке, представленной квадратом, по которой можете двигаться только вперед или назад, а все значения состояний равны нулю. Если вы сделаете шаг назад, движение обязательно, но вы получите награду 0. Если вы сделаете шаг вперед, вы поскользнетесь на льду и с вероятностью 20% упадете влево со значением –1 и вероятностью 80% — вправо со значением 1.

Шаг 6 алгоритма 2.4 направлен на поиск наивысшего значения для любого действия. Эта среда имеет два действия: вверх и вниз. Вероятность перехода вниз при действии вниз равна 1. Вознаграждение равно 0, и поскольку все значения состояний равны нулю, общая сумма действия вниз составляет $1 \times 0 = 0$. Для действия вверх два результата имеют две награды. Эта сумма становится равной $0,2 \times (-1) + 0,8 \times 1 = 0,6$. Следовательно, максимальное значение по всем действиям для текущего состояния составляет 0,6.

Предыдущий пример был для одного состояния, но этот расчет зависит от других состояний. Вот почему алгоритм 2.4 должен перебирать все состояния, чтобы заполнить пробелы. После первого прохода всех состояний вполне вероятно, что вам потребуется обновить их все снова, потому что значения изменились. Шаг 7 записывает максимальную ошибку, чтобы позволить шагу 8 решить, продолжать ли

итерацию. Если вам интересно, вы можете найти доказательство сходимости в ссылках в *разд. "Дополнительные материалы для чтения" в конце данной главы.*

Наконец, на шаге 9 алгоритм выводит детерминированную политику, которая выбирает действие с наибольшим ожидаемым доходом.

## Результаты итерации по ценнности

Результаты реализации алгоритма 2.4 показаны на рис. 2.13. Коэффициент дисконтирования 0,9 использовался для поиска оптимальной политики для простой проблемы обрыва, с которой столкнулись ранее.

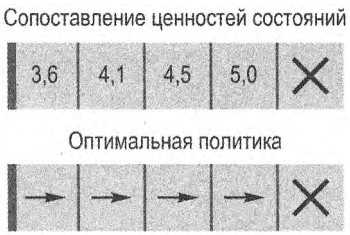



**Рис 2.13.** Результат обучения агента с использованием итерации по ценности с коэффициентом дисконтирования 0,9. На этом рисунке показаны ценности состояний и оптимальная политика

Как и в версии метода Монте-Карло, карту ценности состояния легко интерпретировать. Чем ближе агент к цели, тем больше ожидаемая награда. Небольшая разница в том, что ценность не зависит от случайного достижения целевого состояния. Это дисконтированная версия вознаграждения, если вы следуете политике. Однако результирующая политика остается той же. И я мог бы использовать ту же технику значения политики в алгоритме Монте-Карло.

Если бы я вставил алгоритм 2.4 (алгоритм улучшения политики) в цикл взаимодействия агента (оценка политики), вы могли бы улучшить политику на каждом временном шаге. Сравните это с техниками Монте-Карло, которые ждут до конца эпизода. Динамическое программирование может улучшить политику намного быстрее, чем метод Монте-Карло, что очень важно при решении проблем с длинными (или бесконечными) эпизодами.

Но четыре предостережения не позволяют использовать динамическое программирование во всех средах, кроме самых простых. Первое, на что следует обратить внимание, — это использование дисконтирования. Если бы вы этого не сделали, все состояния имели бы значение, равное награде. Вы должны использовать дисконтирование, чтобы направлять агента и подавлять длинные траектории. Это общая тема для всех RL: предпочитайте более короткие траектории.

Во-вторых, алгоритм 2.4 завершается, когда агент обновит все состояния. Это нормально для рассмотренных до сих пор игрушечных примеров, потому что все они имеют несколько состояний. Но как только количество состояний или действий

увеличивается, использование динамического программирования становится вычислительно невыполнимым.

Третий момент, связанный с предыдущим, — это эффективность выборки алгоритма. При динамическом программировании необходимо посетить все состояния и действия. Для этого требуется, по крайней мере, столько же выборок из среды, сколько состояний, умноженных на действия. Это может быть проблемой для сред, которые вы не можете моделировать. Другие алгоритмы RL на основе моделей могут быть более эффективными, поскольку они могут интерполировать политику с меньшим количеством симуляций. Методы Монте-Карло также более эффективны в отношении выборки, поскольку им, вероятно, не придется посещать каждое отдельное состояние, чтобы найти область состояний, дающих наилучшую отдачу.

Наконец, и, вероятно, самая большая проблема из всех, — это то, что вам нужны вероятности перехода. Вам либо нужна модель динамики окружающей среды, либо вы можете изучить модель с помощью методов выборки. Если выполнить действие несколько раз, можно оценить вероятность приземления в любых состояниях. Вы можете руководствоваться этим в следующей главе.

# Резюме

Эта глава началась с демонстрации того, что тестирование многорукого бандита, которое вы, возможно, использовали раньше, является формой RL. Однако оно ограничено, потому что у большинства промышленных приложений гораздо больше состояний и отложенных вознаграждений.

Марковские процессы принятия решений (Markov decision processes, MDP) — это каркас, на котором строятся все задачи. Вы можете переосмыслить многие проблемы, чтобы они соответствовали парадигме MDP. Ключевыми элементами MDP являются среда, которая обеспечивает наблюдение за ее состоянием, вознаграждение, обозначающее успех или неудачу, и действия, которые изменяют внутреннее состояние среды.

Вычисляя ожидаемое значение следующих состояний, алгоритмы могут выбрать действие, которое максимизирует общую награду. Вы видели это при моделировании управления запасами. Я ограничил задачу, чтобы упростить ее, но вы можете найти примеры более сложных сценариев в литературе.

Оптимальная политика — это такая политика, которой агент может следовать, чтобы получить высокое вознаграждение. Найти оптимальную политику сложно, и алгоритмы, представленные в этой книге, пытаются достичь этого разными способами.

Наконец, вы открыли два способа поиска в состояниях среды для значения доходности: Монте-Карло и динамическое программирование. Методы Монте-Карло хорошо подходят для небольших пространств состояний и когда успеваемость не важна. Но динамическое программирование более эффективно в том смысле, что оно может сразу изучить предполагаемую политику, в отличие от метода Монте-Карло, который должен ждать до конца эпизода.

В следующей главе динамическое программирование рассматривается подробно и представлен один из наиболее распространенных алгоритмов RL, используемых сегодня в промышленности.

## Дополнительные материалы для чтения

- Статистика.
  - Downey A. Think Stats: Exploratory Data Analysis. — O'Reilly Media, 2014.
- A/B-тестирование и бандитские алгоритмы.
  - White J. Bandit Algorithms for Website Optimization. — O'Reilly Media, 2012;
  - моя реализация библиотеки моделирования бандитов на Python[1];
  - подробное руководство по методам отбора для бандитов [1].
- Марковские процессы принятия решений.
  - Puterman M. L. Markov Decision Processes: Discrete Stochastic Dynamic Programming. — John Wiley & Sons, 2014.
- Уравнения Беллмана, динамическое программирование и метод Монте-Карло:
  - Sutton R. S., Barto A. G. Reinforcement Learning: AnIntroduction. — MIT Press, 2018.
- Методы Монте-Карло в других областях.
  - Hilpisch Y. Python for Finance: Mastering Data-Driven Finance. — O'Reilly Media, 2018.

## Использованные источники

[1] Russo D., Van Roy B., Kazerouni A., Osband I., Wen Z. Tutorial on Thompson sampling // ArXiv:1707.02038. — 2020. — July. — URL: **https://oreil.ly/N947v**.

[1] См. **https://gitlab.com/winderresearch/rl/BanditsBook/**.

ГЛАВА 3

# Обучение с учетом временных различий, Q-обучение и *n*-шаговые алгоритмы

В *главе 2* представлены две ключевые концепции для анализа марковских процессов принятия решений (Markov decision processes, MDP). Методы Монте-Карло (Monte Carlo, MC) применяют в качестве инструмента на следующем этапе оценки ценностной функции, обрабатывая выборку значений. Их можно использовать без явного знания вероятностей переходов, поскольку они могут эффективно осуществлять выборку состояний в больших объемах. Однако они должны выполняться в течение всего эпизода обучения, прежде чем агент сможет обновить политику.

И наоборот, методы динамического программирования (dynamic programming, DP) *загружаются* путем обновления политики после одного временно́го шага. Но DP-алгоритмы должны иметь полное представление о вероятностях перехода, посетить все возможные состояния и выполнить необходимые действия, прежде чем смогут найти оптимальную политику.

В самых разных дисциплинах термин *"самозагрузка"* (бутстрэппинг, bootstrapping) используется для обозначения того, что сущность может "поднять себя". Предприятия начинают с того, что собирают наличные деньги без привлечения кредитов. В электронных транзисторных схемах самозагрузка используется для повышения входного сопротивления или рабочего напряжения. В статистике и RL бутстрэппинг — это метод выборки, который использует индивидуальные наблюдения для оценки статистики совокупности данных.

*Обучение с учетом временны́х различий* (син.: обучение с учетом временно́й разницы, обучение временно́й разницы — temporal-difference, TD) представляет собой комбинацию этих двух подходов. Обучение происходит непосредственно с опорой на опыт путем выборки, а также бутстрэппинга (bootstrap). Такой подход представляет собой прорыв в возможностях и позволяет агентам изучать оптимальные стратегии в любой среде. До этого обучение было настолько медленным, что делало задачи неразрешимыми, либо для решения задачи требовалась полная модель окружающей среды. Не стоит недооценивать эти методы; с помощью инструментов, представленных в этой главе, вы можете проектировать сложные автоматизированные системы. В этой главе рассказывается о TD-обучении и его использовании для определения различных реализаций.

# Обучение с учетом временных различий: формулировка подхода

Напомним, что политики зависят от прогнозирования функции ценности состояния (или ценности действия). Как только агент предсказывает ожидаемую прибыль, ему нужно выбрать действие, которое максимизирует награду.

Помните, что "ожидаемое" значение — это среднее арифметическое (среднее значение) большого количества наблюдений этого значения. На данный момент вы можете вычислить среднее значение всех наград, вручаемых при посещении этого состояния в прошлом. А онлайн-версия среднего — это экспоненциально убывающая скользящая среднего. Далее в книге мы будем использовать модели для прогнозирования ожидаемой доходности.

Уравнение 2.8 определяет функцию ценности состояния как ожидаемый результат, начиная с данного состояния, и в уравнении 2.2 вы видели, как преобразовать математическую запись в онлайн-алгоритм. Уравнение 3.1 представляет собой онлайн-реализацию оценки по методу Монте-Карло функции ценности состояния. Ключевой частью является экспоненциально взвешенная оценка скользящего среднего дохода $G$. Со временем вы обновляете оценку ценности состояния, чтобы приблизиться к истинному ожидаемому доходу, а $\alpha$ управляет скоростью, с которой вы обновляете эту оценку.

**Уравнение 3.1. Онлайн-реализация функции ценности состояния по методу Монте-Карло**

$$V_\pi(s) \doteq \mathbb{E}_\pi[G \mid s]$$
$$\leftarrow \mathbb{E}_\pi\left[V_\pi(s) + \alpha\left(G - V_\pi(s)\right) \mid s\right].$$

Напомним, что $G$ — это общий доход от всей траектории. Таким образом, проблема уравнения 3.1 заключается в том, что вам нужно дождаться окончания эпизода обучения, прежде чем вы получите значение $G$. Так что, слово "онлайн" здесь означает то, что алгоритм обновляет один шаг за раз, но вам все равно нужно ждать конца эпизода.

Оценка ценности из *разд. "Итерация по ценности с динамическим программированием" главы 2* несколько иначе интерпретирует функцию ценности состояния. Вместо того чтобы ожидать окончания эпизода и получения результата, вы можете итеративно обновлять текущую оценку ценности состояния, используя дисконтированную оценку со следующего шага. В уравнении 3.2 представлена онлайн-версия динамического программирования. Термин "онлайн" указывает на то, что все время алгоритм просчитывает ситуацию на шаг вперед. Обратите внимание, что это уравнение начинается с того же определения функции ценности состояния, что и версия для метода Монте-Карло.

**Уравнение 3.2. Функция ценности состояния динамического программирования для онлайн-режима**

$$V_\pi(s) \doteq \mathbb{E}_\pi[G \mid s] \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right].$$

Вы уже знаете, что существует проблема неизвестности вероятностей перехода. Но обратите внимание на последнюю строку уравнения 3.2: версия для динамического программирования использует интересное определение для $G$ — $r + \gamma V(s')$. Это означает, что доходность равна текущей награде плюс прогноз следующего состояния со скидкой. Другими словами, доход — это текущее вознаграждение плюс оценка всех будущих вознаграждений. Можно ли использовать это определение $G$ в алгоритме Монте-Карло, чтобы не ждать конца эпизода? Да! Рассмотрим уравнение 3.3.

**Уравнение 3.3. Функция ценности состояния для обучения с учетом временны́х различий**

$$V_\pi(s) \doteq \mathbb{E}_\pi\left[V_\pi(s) + \alpha\left(r + \gamma V(s') - V_\pi(s)\right) \mid s\right].$$

Уравнение 3.3 — это TD-оценка ожидаемой доходности. Она сочетает преимущества бутстрэппинга с эффективностью выборки скользящего среднего по методу Монте-Карло. Обратите внимание, что вы можете видоизменить это уравнение, чтобы оценить функцию ценности действия.

Наконец я могу преобразовать это выражение в онлайновый алгоритм, который предполагает, что экспоненциально взвешенное среднее является подходящей оценкой для оператора ожидания, и приводит к онлайн-оценке ценности состояния для TD-обучения (уравнение 3.4).

**Уравнение 3.4. Интерактивная функция определения состояния для обучения с учетом временны́х различий**

$$V_\pi(s) \leftarrow V_\pi(s) + \alpha\left(r + \gamma V(s') - V_\pi(s)\right).$$

Уделите минуту, чтобы осмыслить уравнения; попробуйте интерпретировать уравнения своими словами. Можете изучить оптимальную политику в соответствии с произвольным определением вознаграждения, мысленно абстрагируясь от окружающей среды и итеративно обновляя функцию ценности состояния в соответствии с уравнением 3.3. Обновления функции ценности состояния и, следовательно, политики, происходят сразу; таким образом, каждое ваше следующее действие должно быть лучше предыдущего.

## Q-обучение

В 1989 г. реализация TD-обучения под названием *Q-обучение* позволила математикам упростить доказательство сходимости. Считается, что именно с этого алгоритма начался ажиотаж вокруг RL. Можно переписать уравнение 3.3, чтобы получить функцию ценности действия (см. уравнение 2.9). Уравнение 3.5 реализует простую политику путем преднамеренного выбора наилучшего действия для данного состояния (по аналогии с уравнением 2.10).

**Уравнение 3.5. Правило онлайн-обновления Q-обучения для оценки ожидаемой прибыли**

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(r + \gamma \arg\max_{a_s \in \mathscr{A}(s)} Q(s', a_s) - Q(s,a)\right).$$

Напомним, что $Q(s,a)$ — функция ценности действия из уравнения 2.9, которая сообщает вам ожидаемый результат каждого действия. Уравнение 3.5 реализовано в качестве правила онлайн-обновления, и это означает, что вам следует выполнить итерацию для уточнения оценки $Q(s,a)$ путем перезаписи предыдущей оценки. Все, что находится справа от $\alpha$, можно рассматривать как *дельту*, показывающую, насколько неверна оценка по сравнению с тем, что произошло на самом деле. Пошаговый прогноз TD-обучения, реализованный в дельте, позволяет алгоритму прогнозировать ситуацию на один шаг вперед, а агенту — выяснить, какое действие лучше всего.

Это самое важное улучшение. Предыдущие попытки оценивали ожидаемую доходность каждого состояния, используя только информацию об этом состоянии. Уравнение 3.5 позволяет агенту увидеть, какие будущие траектории являются оптимальными. Это называется *развертыванием* (rollout). Агент развертывает состояния и действия в траектории (как максимум до конца эпизода, например, для методов Монте-Карло). Агент может создавать политики, которые не только оптимальны локально, но и будут оптимальны в будущем. Агент может создавать политики, которые могут предсказывать будущее.

Эта реализация функции ценности действия является особенно примечательной за счет одного простого дополнения. Аргумент arg max из уравнения 3.5 сообщает алгоритму, что требуется просмотреть все возможные действия для этого состояния и выбрать то, которое дает наилучший ожидаемый доход.

Такое активное направление агента называется Q-обучением и реализовано в алгоритме 3.1.

**Алгоритм 3.1. Q-обучение (TD вне политики)**

1: **input**: политика, которая использует функцию ценности действия $\pi(a \mid s, Q(s,a))$.

2: Инициализировать $Q(s,a) \leftarrow 0$ для всех $s \in \mathscr{S}, a \in \mathscr{A}(s)$.

3: **loop**: для каждого эпизода

4: Инициализировать окружающую среду, чтобы обеспечить состояние $s$.

5: **do**:

6: Выбрать действие $a$ из состояния $s$, используя политику $\pi$ (неоднозначности разрешить случайным образом).

7: Выполнить действие $a$, а затем наблюдать за наградой $r$ и следующим состоянием $s'$.

8: $$Q(s,a) \leftarrow Q(s,a) + \alpha\left(r + \gamma \arg\max_{a_s \in \mathcal{A}(s)} Q(s', a_s) - Q(s,a)\right).$$

9: $s \leftarrow s'$.

10: **while** состояние $s$ не станет конечным.

Алгоритм 3.1 демонстрирует реализацию Q-обучения. Бóльшая часть этого алгоритма должна показаться вам знакомой, поскольку он тесно связан с алгоритмами 2.3 и 2.4. Как и раньше, шаг 1 начинается с передачи политики, которая используется для определения вероятности того, какое действие следует предпринять (например, которое может реализовать $\varepsilon$-жадное исследование из алгоритма 2.1). Шаг 2 инициализирует массив для хранения всех оценок ценности действия. Это можно эффективно реализовать с помощью словаря, в который автоматически записывается значение, заданное по умолчанию.

Шаг 3 повторяется по всем эпизодам, а шаг 5 — для каждого шага в эпизоде. Шаг 4 инициализирует начальное состояние. Среды обычно предоставляют функцию для "сброса" агента до состояния по умолчанию.

Шаг 6 использует функцию внешней политики, чтобы решить, какое действие предпринять. Выбранное действие является наиболее вероятным. Я постарался представить это более явно, чем обычно дается литературе, поскольку хочу подчеркнуть, что никогда не бывает единственного оптимального действия. Вместо этого ряд действий имеет некоторую вероятность оказаться оптимальным вариантом. Q-обучение выбирает действие, которому присуща максимальная вероятность, которая часто получается из наибольшего значения $Q(s,a)$. И вы можете реализовать $\varepsilon$-жадность, задав для всех действий одинаковую вероятность, например в течение некоторого времени.

Шаг 7 использует выбранное действие в окружающей среде и наблюдает за наградой и следующим состоянием. Шаг 8 обновляет текущую оценку ценности действия для данного состояния и данного действия на основе нового поведения окружающей среды. Помните, что ожидаемый доход рассчитывается как сумма текущего вознаграждения и ожидаемого значения следующего состояния. Таким образом, дельта — это данный результат минус ожидаемое значение текущего состояния. Шаг 8 также экспоненциально уменьшает дельту с помощью параметра $\alpha$, который помогает усреднить зашумленные обновления.

Шаг 9 устанавливает следующее состояние, и наконец, шаг 10 повторяет этот процесс до тех пор, пока среда не сообщит, что состояние является конечным. При достаточном количестве эпизодов (см. шаг 3) оценка ценности действия приближается к правильному ожидаемому доходу.

Посмотрите на конец алгоритма 3.1 на шаге 8. Обратите внимание на погрешность TD в правой части. $\arg\max_{a_s \in \mathcal{A}(s)} Q(s', a_s)$ просматривает все возможные действия для следующего состояния и выбирает то, которое дает наибольший ожидаемый доход. Это не имеет ничего общего с действием, выбранным на шаге 6. Вместо этого агент планирует маршрут, используя будущие действия.

Это первый практический пример агента, действующего *вне политики.* Алгоритм не соответствует политике, поскольку обновление влияет только на текущую политику. Он не использует обновление для управления агентом. Агент по-прежнему должен выводить действие из текущей политики. Это тонкое, но важное улучшение, которое было использовано совсем недавно. Далее в этой книге вы увидите и другие примеры алгоритмов, действующих вне политики.

Еще одно замечание — использование массива на шаге 2. Он сохраняет оценку ожидаемого значения для каждой пары "состояние — действие" в таблице, и поэтому данный метод часто называют *табличным методом.* Это различие важно, т. к. оно означает, что вы не можете использовать алгоритм 3.1, когда у вас есть непрерывные состояния или действия, которые могут привести к образованию массива бесконечной длины. Вы увидите методы, которые могут изначально обрабатывать непрерывные состояния, в *главе 4*, или действия, в *главе 5*. Вы сможете узнать, как преобразовать непрерывные значения в дискретные в *главе 9*.

## SARSA

Подход SARSA[1] был разработан вскоре после Q-обучения, чтобы предоставить более общее решение для TD-обучения. Основное отличие SARSA от Q-обучения — отсутствие $\arg\max$ в дельте. Вместо этого он рассчитывает ожидаемую доходность путем усреднения по всем прогонам (в режиме онлайн).

Из уравнения 3.6 видно, что название этой конкретной реализации TD-обучения происходит от требований состояния, действия и вознаграждения для вычисления функции ценности действия. Алгоритм SARSA приведен в алгоритме 3.2.

**Уравнение 3.6. Онлайн-оценка SARSA функции ценности действия**

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(r + \gamma Q(s',a') - Q(s,a)\right).$$

И алгоритм 3.2, и алгоритм 3.1 имеют гиперпараметры. Первый — $\gamma$; это коэффициент, который дисконтирует ожидаемую доходность, введенную в уравнении 2.7.

---

[1] State Action Reward State Action означает действие, которое символизирует кортеж ($s$, $a$, $r$, $s'$, $a'$). Агент SARSA взаимодействует со средой и обновляет политику на основе предпринятых действий. — *Прим. ред.*

Второй — это поправка на скорость обучения ценности действия $\alpha$, которая должна находиться в диапазоне от 0 до 1. Высокие значения обучаются быстрее, но усредняются хуже.

**Алгоритм 3.2. SARSA (TD-обучение вне политики)**

1: **input**: политика, которая использует функцию ценности действия $\pi(a \mid s, Q(s,a))$

2: Инициализировать $Q(s,a) \leftarrow 0$ для всех $s \in \mathcal{S}, a \in \mathcal{A}(s)$.

3: **loop**: для каждого эпизода

4: Инициализировать окружающую среду, чтобы обеспечить состояние $s$.

5: **do**:

6: Выбрать действие $a$ из состояния $s$, используя политику $\pi$ (неоднозначности разрешить случайным образом).

7: Выполнить действие $a$, а затем наблюдать за наградой $r$ и следующим состоянием $s'$.

8: Выбрать действие $a'$ из состояния $s'$, используя политику $\pi$ (неоднозначности разрешить случайным образом).

9: $Q(s,a) \leftarrow Q(s,a) + \alpha\left(r + \gamma Q(s',a') - Q(s,a)\right)$.

10: $s \leftarrow s', a \leftarrow a'$.

11: **while** состояние $s$ не станет конечным.

Алгоритм 3.2 почти такой же, как алгоритм 3.1, поэтому я не буду повторять объяснение полностью. Одно отличие состоит в том, что алгоритм выбирает следующее действие $a'$ на шаге 8 до того, как обновит функцию ценности действия на шаге 9. Действие $a'$ обновляет политику и направляет агента на следующий шаг, именно поэтому данный алгоритм называется *алгоритмом на основе политики*.

Различия между алгоритмами, следующими политике, и алгоритмами вне политики увеличиваются при применении аппроксимации функций, которую я использую в *главе 4*. Но в самом ли деле эти тонкости имеют какое-либо значение? Позвольте мне протестировать эти алгоритмы, чтобы показать вам, что происходит.

## Q-обучение против SARSA

Одной из классических сред в литературе по RL является Gridworld — двумерная сетка из квадратов. Вы можете размещать на квадрате различные препятствия. В первом примере я размещаю глубокую яму вдоль одного края сетки. Это имитация обрыва. Цель состоит в том, чтобы научить агента перемещаться по обрыву от начальной точки до целевого состояния.

Я разрабатываю награды, чтобы среди всех возможных путей к цели найти кратчайший. Каждый шаг имеет штраф –1, падение со скалы дает штраф –100, а состояние цели имеет награду 0. Обрыв и состояние цели являются конечными состояниями (здесь заканчивается эпизод). Визуализация этой среды показана на рис. 3.1.

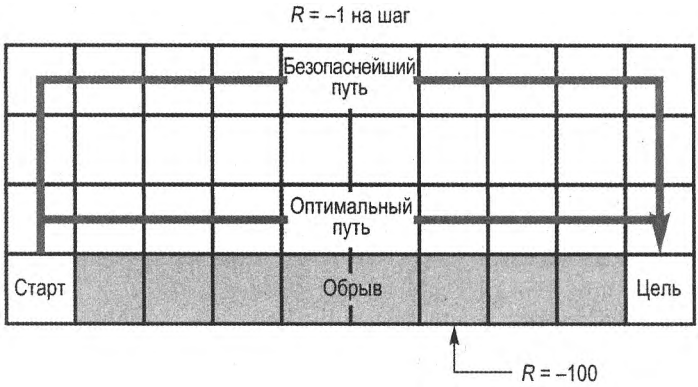



**Рис. 3.1.** Изображение сеточной среды с обрывом вдоль одного края [1]

Чтобы продемонстрировать различия, я реализовал Q-обучение и SARSA на соответствующем веб-сайте. Там вы найдете обновленный список текущих RL-фреймворков. И я подробнее расскажу о практических аспектах реализации RL в *главе 9*.

Если я обучаю этих агентов в среде, показанной на рис. 3.1, я могу записывать награды за эпизоды для каждого временно́го шага. Из-за случайного исследования агента мне нужно повторять эксперимент много раз, чтобы усреднить шум. Результаты продемонстрированы на рис. 3.2, где графики показывают среднюю сумму вознаграждений за 100 параллельных экспериментов.

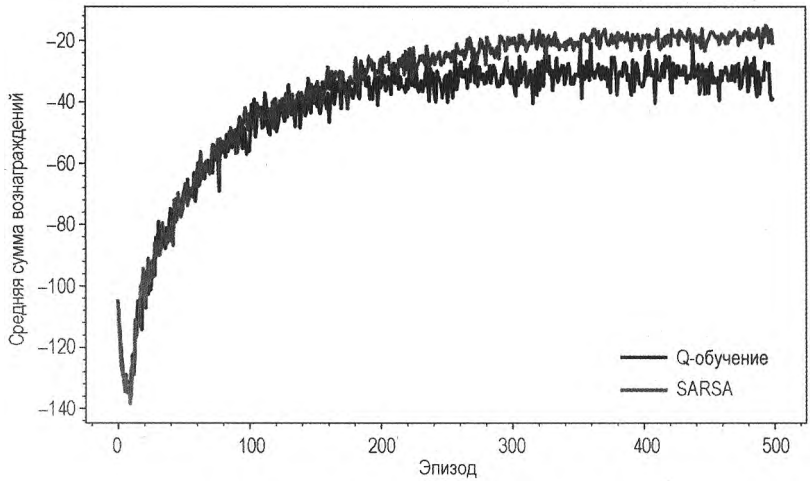



**Рис. 3.2.** Сравнение Q-обучения и SARSA для простой задачи с сеткой. Агенты были обучены в среде, показанной на рис. 3.1. Я использовал $\gamma \doteq 1,0$, $\varepsilon \doteq 0,1$ и $\alpha \doteq 0,5$. Награды за каждый эпизод были зафиксированы и усреднены по 100 испытаниям

Напомним, что эпизод — это один полный эксперимент (от начальной точки до конечного узла), который в данном случае завершается падением с обрыва или достижением цели. По оси абсцисс на рис 3 2 отложены номера эпизодов 500 серий означают, что агент успевает 500 раз погибнуть

На рис. 3.2 показаны две интересные области. Первая — это провал в начале обучения. Прохождение первого эпизода было лучше пятого. Другими словами, случайный агент работает лучше, чем агент, обученный пяти эпизодам. Причина этого кроется в структуре вознаграждения. Я назначил штраф –100 за падение с обрыва. Используя полностью случайного агента со стартом, слишком близким к обрыву, будьте готовы к тому, что агент сразу же упадет. Это даст около –100 очков награды в начале. Однако после нескольких испытаний агент узнал, что падать с обрыва больно, поэтому он пытается уклониться. Уход приводит к штрафу –1 за шаг, а поскольку обрыв довольно длинный и агент еще не имеет большого опыта в достижении цели, он тратит шаги впустую, а затем все равно срывается — такая тяжелая у него жизнь. Сумма отрицательного вознаграждения (точнее штрафа) из-за потраченных впустую шагов и возможного полета с обрыва дает меньшее вознаграждение, чем при запуске агента (примерно –125).

Мы получили интересный результат. Иногда нужно принять более низкое вознаграждение в краткосрочной перспективе, чтобы найти лучшее вознаграждение в долгосрочной. Это расширение проблемы эксплуатации и разведки.

Вторая любопытная особенность заключается в том, что результат SARSA, как правило, дает большее вознаграждение и имеет меньший шум, чем Q-обучение. Причина этого лучше всего может быть описана в результирующей политике.

На рис. 3.3 показан пример политик для Q-обучения и SARSA. Обратите внимание, что конкретная политика может меняться от испытания к испытанию из-за случайного исследования. Агент SARSA предпочитает "безопасный" путь подальше от обрыва. Это потому, что иногда $\varepsilon$ -жадное действие случайно сбивает агента с обрыва. Напомним, что SARSA рассчитывает *среднюю* ожидаемую доходность. Следовательно, в среднем ожидаемая отдача лучше вдали от обрыва, потому что шансы получить три случайных толчка в сторону обрыва невелики.

Алгоритм Q-обучения, напротив, не предсказывает ожидаемую доходность. Он предсказывает наилучшую возможную отдачу от этого состояния. А лучшая — та, при которой нет случайных толчков прямо к краю обрыва. Рискованная, но, по определению вознаграждения, оптимальная политика.

Всё сказанное также объясняет разницу в дисперсии. Поскольку Q-обучение метафорически находится на острие ножа, оно иногда вызывает падение. В этом причина огромных, но случайных потерь. Политика SARSA в том, чтобы находиться далеко от края обрыва, и следовательно, она приводит к редким случаям падания, поэтому вознаграждения более стабильны.

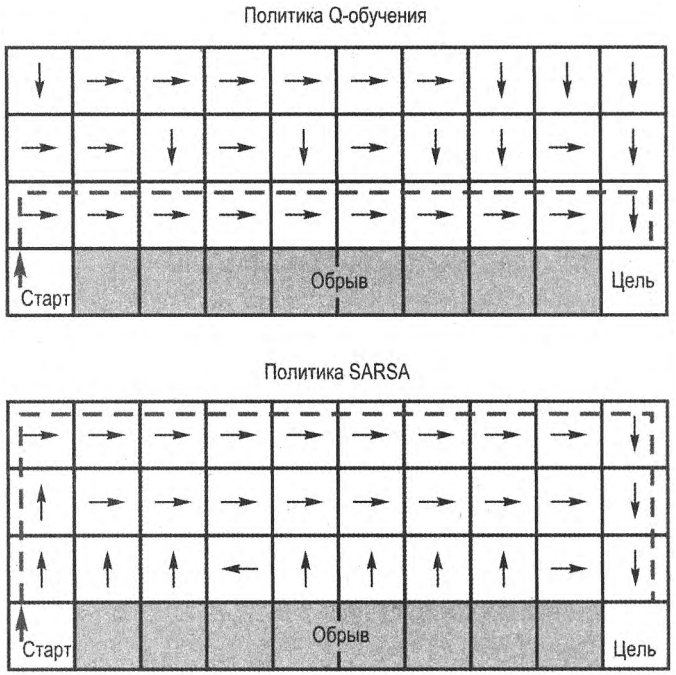



**Рис. 3.3.** Политики, разработанные агентами Q-обучения и SARSA. Q-обучение, как правило, выбирает оптимальный маршрут, SARSA — безопасный путь

## Пример использования: автоматическое масштабирование контейнеров приложений для снижения затрат

Программные *контейнеры* меняют способ развертывания и оркестрации распределенными приложениями в облаке. Их основные преимущества — инкапсуляция всех зависимостей времени выполнения и атомарность развертываемого модуля. Это означает, что приложения можно динамически масштабировать как по горизонтали, дублируя их, так и по вертикали, наращивая выделенные ресурсы. Традиционные стратегии масштабирования базируются на фиксированных эвристических методах на основе пороговых значений, таких как использование центрального процессора (ЦП) или памяти, и, вероятно, будут неоптимальными. Использование RL для обеспечения политики масштабирования может снизить затраты и повысить производительность.

Росси, Нарделли и Карделлини предложили эксперимент с использованием Q-обучения и подхода, основанного на моделях, для обеспечения оптимальных политик масштабирования как по вертикали, так и по горизонтали [2]. Затем они реализовали эти алгоритмы в реальной среде Docker Swarm.

Росси и соавт. начали с использования универсального контейнера, работабщего по принципу черного ящика, который выполняет произвольную задачу. Агент взаимодействует с приложением в рамках марковского процесса принятия решений.

Состояния определены как количество контейнеров, загрузка ЦП и выделение ЦП каждому контейнеру. Авторы эксперимента квантуют непрерывные переменные в фиксированное количество ячеек.

Действия определяются как изменение горизонтального или вертикального масштаба приложения. Они могут добавить или удалить реплику, добавить или удалить выделенные процессоры или и то и другое. Авторы проводят два эксперимента: первый упрощает пространство действий, допуская только одно вертикальное или горизонтальное изменение, а второй позволяет и то и другое.

Вознаграждение — довольно сложная функция, помогающая сбалансировать количество изменений развертывания, задержку и использование. Эти метрики имеют произвольные веса, позволяющие пользователю указать, что именно важно для его приложения.

Хотя Росси и соавт. изначально использовали симулятор для разработки собственного подхода, они, к счастью, протестировали свой алгоритм в реальной среде с помощью Docker Swarm. Они внедрили контроллер для мониторинга, анализа, планирования и выполнения изменений в приложении. Затем смоделировали запросы пользователей, выстраивая их в соответствии с паттерном, характерным для изменяющегося спроса, чтобы нагружать приложение и обучать алгоритмы для получения максимального вознаграждения.

Этот эксперимент дает несколько интересных результатов. Во-первых, авторы обнаружили, что алгоритмы быстрее обучались в ситуации с меньшим пространством действия в эксперименте, когда агенту приходилось выбирать масштабирование лишь по вертикали или горизонтали, а не то и другое одновременно. Это предсказуемый результат; меньшие пространства состояний и действий означают, что агент должен провести меньше изысканий и может быстрее найти оптимальную политику. Во время простого эксперимента подходы с Q-обучением и на основе моделей работали одинаково. С более сложным пространством действий подход, основанный на модели, учился быстрее и результаты были намного лучше.

Второй интересный результат заключается в том, что использование модели, специально предназначенной для решения этой задачи, улучшает обучение и конечные результаты. Такой подход, основанный на моделях, включает вероятности перехода в аппроксимацию значения, как вы это делали в *разд. "Контроль запасов" главы 2*, поэтому алгоритм заранее знает, каков будет эффект. Однако предложенная авторами модель неявно смещает результаты в сторону этой модели. По результатам видно, в каких случаях подход, основанный на модели, располагает к масштабированию по вертикали больше, чем к масштабированию по горизонтали. В большинстве современных оркестраторов гораздо проще масштабировать по горизонтали, поскольку вертикальное масштабирование будет только во вред. Мораль в том, что подходы, основанные на моделях, вводят в систему априорные знания и значительно ускоряют обучение, но вы должны быть уверены, что модель отражает истинную динамику системы и нет неявной предвзятости в отношении определенной политики.

Конечный результат в любом случае — значительное улучшение по сравнению со статическими политиками на основе пороговых значений. Оба алгоритма улучшили

коэффициент использования (что должно привести к снижению затрат) при сохранении аналогичных уровней задержки.

Вы могли бы улучшить эту работу несколькими способами. Мне бы хотелось, чтобы этот пример был реализован в Kubernetes, и я также хотел бы видеть вознаграждения, основанные на реальной стоимости используемых ресурсов, а не на какой-то мере использования. Я бы также улучшил представление состояния и действия: аппроксимация функции устранит необходимость дискретизации состояния, а непрерывное действие позволит выбрать оптимальные параметры вертикального масштабирования. У меня также возникло бы искушение разделить проблему, потому что масштабирование по вертикали и горизонтали имеет очень разные затраты; по крайней мере, в Kubernetes гораздо проще масштабировать по горизонтали.

# Отраслевой пример: торги рекламы в режиме реального времени

В отличие от спонсорской рекламы, где рекламодатели устанавливают фиксированные ставки, при торгах в режиме реального времени (real-time bidding, RTB) вы можете установить ставку для каждого отдельного показа. Когда пользователь посещает веб-сайт, поддерживающий рекламу, запускается аукцион, на котором рекламодатели делают ставки за показ. Рекламодатели должны подавать заявки в течение определенного периода времени, обычно на это отводится 100 мс.

Рекламная платформа предоставляет рекламодателю контекстную и поведенческую информацию для оценки. Рекламодатель использует автоматический алгоритм, чтобы решить, какую ставку следует сделать в зависимости от контекста. В долгосрочной перспективе продукты платформы должны нравиться посетителям, чтобы можно было поддерживать поток доходов от рекламы. Однако рекламодатели хотят максимизировать некоторый ключевой показатель эффективности (key performance indicator, KPI), например количество показов или рейтинг кликов (click-through rate, CTR), с наименьшими затратами.

RTB воплощает четкое действие (ставку), состояние (информацию, предоставляемую платформой) и агента (алгоритм торгов). Обе платформы и рекламодатели могут использовать RL для оптимизации определения вознаграждения. Как вы понимаете, необработанные данные о показах на уровне ставок очень важны. Но в 2013 г. китайская маркетинговая компания iPinYou выпустила набор данных в надежде, что исследователи смогут улучшить современные алгоритмы торгов [3].

## Определение марковского процесса принятия решения

Я покажу, как простые RL-алгоритмы, такие как Q-обучение и SARSA, могут предоставить решение проблемы торгов в режиме реального времени. Основное ограничение этих алгоритмов состоит в том, что им требуется простое пространство состояний. Большие или непрерывные состояния не подходят, потому что агентам нужно много выборок, чтобы сформировать разумную оценку ожидаемого значе-

ния (согласно закону больших чисел). Непрерывные состояния вызывают дополнительные проблемы из-за дискретной выборки. Для табличных алгоритмов вы должны упростить и ограничить состояние, чтобы сделать его управляемым.

Предлагаю создать алгоритм торгов в режиме реального времени, который максимизирует количество показов при заданном бюджете. Я разделяю данные на партии, и у каждой партии есть бюджет. Состояние состоит из средней ставки, которую агент может сделать с учетом количества оставшихся аукционов. Например, если бюджет равен 100, а размер партии равен 10, то на первом аукционе состояние будет $100/10 = 10$. Цель этой функции — сообщить агенту, что я хочу распределить ставки по всей партии. Я не хочу, чтобы агент потратил весь бюджет сразу. Агент получит вознаграждение в размере 1, если он выиграет аукцион, и 0 — в противном случае. Для того чтобы еще больше уменьшить количество состояний, я округляю и обрезаю вещественное число до целого. Позже вы увидите примеры агентов, которые могут адаптировать функцию к состоянию, чтобы она могла захватывать гораздо более сложные данные.

Всё вышесказанное дает нам важный урок. Формулировка MDP произвольна. Вы можете представить себе множество разных частей информации, которые нужно включить в состояние или создать другую награду. Вы должны разработать MDP в соответствии с задачей, которую пытаетесь решить, в рамках ограничений алгоритма. Специалисты по промышленным данным редко меняют детали реализации алгоритма машинного обучения; они уже готовы к применению. Специалисты, применяющие RL на практике, тратят бóльшую часть времени на улучшение представления состояния (обычно называемое конструированием признаков) и уточнение определения вознаграждения. Промышленное машинное обучение устроено похоже; занимаясь им, специалисты по обработке данных тратят бóльшую часть своего времени на улучшение данных или переформулирование задач.

## Результаты торгов в режиме реального времени

Я создал две новые среды. Первая — это статическая среда RTB для тестирования. Каждый эпизод — это партия из 100 аукционов, и я устанавливаю бюджет в 10 000. Если агент предложит цену бóльшую или равную 100, он выиграет аукцион. Следовательно, оптимальной политикой для этой среды является ставка 100 на каждом аукционе, чтобы можно было получить вознаграждение в размере 100. Агент может увеличить или уменьшить текущую ставку на 50, 10 или 0%. Результаты представлены на рис. 3.4.

Картина не очень реалистична, но демонстрирует, что реализации алгоритма работают должным образом. Несмотря на изучение оптимальной политики, награда не достигает 100 из-за высокого значения $\varepsilon$. Иными словами, получая эти награды, агент по-прежнему совершает случайные (часто неправильные) действия. Существует также тонкая разница между SARSA и Q-обучением, потому что SARSA предпочитает более безопасный путь с небольшим завышением цены, и такая стратегия означает, что бюджет расходуется быстрее, чем при Q-обучении; то же самое, что вы уже видели в *разд. "Q-обучение против SARSA" ранее в данной главе*.

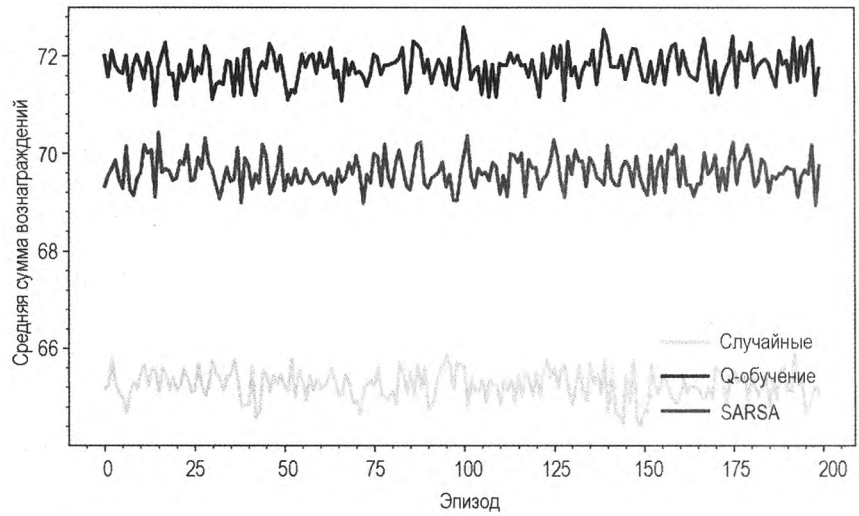



**Рис 3.4.** Среднее вознаграждение, полученное при применении SARSA, Q-обучения и случайных агентов в простой, статичной среде торгов в режиме реального времени

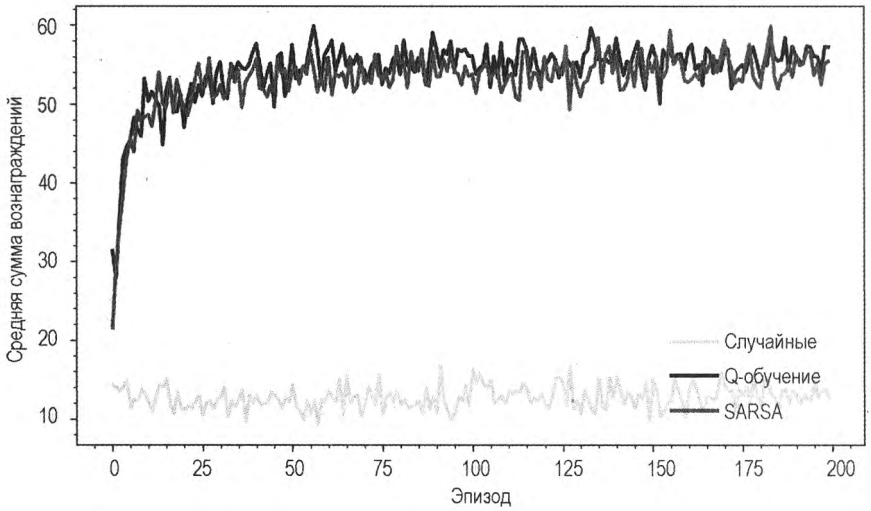



**Рис. 3.5.** Среднее вознаграждение, полученное при применении SARSA, Q-обучения и случайных агентов в среде, основанной на реальных данных. Агент получает большее вознаграждение по мере прохождения эпизодов. Причина в том, что агент узнал, путем случайного исследования, что увеличение ставки приводит к большему вознаграждению. Если позволить агенту продолжить обучение, это, вероятно, еще больше увеличит вознаграждение

Вторая среда использует реальные данные о торгах. Если вы будете следовать инструкциям из репозитория make-ipinyou-data[2], у вас будут каталоги, представляющие рекламодателей. Внутри эти файлы содержат прогнозируемый рейтинг кликов, информацию о том, действительно ли пользователь щелкнул на объявлении, и величину выигравшей ставки. Размер пакета и действия агента такие же, как и раньше. Основное отличие состоит в том, что стоимость выигрыша в аукционе теперь переменная. Задача, как и прежде, — обучить агента получению наибольшего количества показов при заданном бюджете. Я также усложняю ситуацию, устанавливая слишком низкую начальную ставку агента. Агенту придется научиться увеличивать свою ставку. Результаты приведены на рис. 3.5.

## Дальнейшие улучшения

Предыдущий пример ограничивал задачу торгов в режиме реального времени, потому что мне нужно было упростить пространство состояний. Табличные методы, такие как Q-обучение и SARSA, не справляются с большим пространством состояний; агенту пришлось бы выбрать огромное, а возможно, и бесконечное количество состояний. Позже в книге вы увидите методы, которые создают модель пространства состояний, а не таблицу поиска.

Еще одна проблема — выбор вознаграждения. Я решил использовать счетчик показов, потому что это приводило к обучению на каждом временно́м шаге и, следовательно, ускоряло обучение. Вместо этого может быть полезным использовать клики пользователя в качестве сигнала к выдаче вознаграждения, поскольку рекламодатели часто заинтересованы лишь в том, действительно ли пользователь щелкает по их рекламе.

Я также ограничил количество данных, используемых при обучении, чтобы оно могло завершиться за считанные секунды; в противном случае на работу уйдут часы. В промышленных условиях вы должны использовать как можно больше данных, чтобы помочь агенту обобщить их для новых примеров.

Первоначальная установка ставки неуместна, потому что я заставляю агента учиться на крайностях. Я сделал это, чтобы показать, что агент может научиться справляться с новыми ситуациями, например когда новый, финансово обеспеченный рекламодатель пытается захватить рынок, перебивая вашу цену. Более реалистичный сценарий может заключаться не в сбрасывании начальной ставки, а в переносе ее из предыдущего эпизода.

Для повышения отказоустойчивости и стабильности агента всегда применяйте ограничения. Например, можете использовать отсечение ошибок, чтобы исключить возможность слишком сильного смещения оценок ожидаемой доходности за один раз (*см. разд. "Усеченная цель PPO" главы 6*, где приведен пример алгоритма, который делает это).

Мне известна работа, в которой показан еще один шаг вперед в пакетной обработке — объединение набора аукционов в один обобщенный аукцион. Агент устанав-

[2] См. **https://oreil.ly/uZ-xk**.

ливает уровень ставки в начале и сохраняет его неизменным на протяжении всей партии. Преимущество состоит в том, что такой подход позволяет снизить базовую изменчивость данных за счет усреднения.

Данные также характеризуются временны́ми различиями. В 3 часа ночи проводится меньше аукционов и активно меньше рекламодателей по сравнению с 3 часами дня. Эти два периода времени требуют принципиально разных агентов. Включение времени работы пользователя в состояние агента в качестве параметра позволяет агенту изучать зависящие от времени политики.

Действия, использованные в этой среде, были дискретными изменениями ставки. Это уменьшает размер таблицы поиска действий состояния. Предпочтительно разрешить агенту устанавливать значение изменения. Можно даже позволит агенту устанавливать ставку напрямую, используя алгоритмы, которые могут логически выводить непрерывно изменяемое действие.

Что ж, это был длинный список улучшений, и я уверен, что вы можете придумать гораздо больше. Однако обратите внимание, что выбор алгоритма RL часто является одним из самых простых шагов. Этот пример демонстрирует, что в промышленности вы потратите большую часть своего времени на формулирование соответствующего марковского процесса принятия решений или улучшение функций, участвующих в формировании состояния.

# Расширения для Q-обучения

Исследователи могут улучшить TD-алгоритмы различными способами. В этом разделе исследуются некоторые из наиболее важных улучшений, конкретно относящихся к SARSA и Q-обучению. Кое-какие улучшения разумнее оставить до следующих глав.

## Двойное Q-обучение

Уравнение 3.5 обновляет $Q(s,a)$, используя понятие наилучшего действия, которое в данном случае является действием, приносящим наибольшее ожидаемое вознаграждение. Проблема в том, что текущий максимум может быть статистической аномалией. Рассмотрим случай, когда агент только начал обучение. В первый раз, когда агент получит положительное вознаграждение, он обновит оценку ценности действия. В последующих эпизодах будет неоднократно выбираться один и тот же набор действий. Это может создать цикл, в котором агент продолжает принимать неоптимальные решения из-за первоначального плохого обновления.

Одним из способов решения этой проблемы является использование двух функций ценности действия, другими словами, двух таблиц поиска [4]. Агент использует одну из них для обновления другой, и наоборот. Мы получаем непредвзятую оценку $Q(s,a)$, потому что она не возвращается сама в себя. Обычно выбор комбинации для обновления, например обновления $Q_1$ с оценкой ценности действия $Q_2$, является случайным.

## Отложенное Q-обучение

Пожалуй, в лучшей вычислительной теории всех времен, *вероятно приближённо корректном* (probably approximately correct, PAC) *обучении*, утверждается, что для каждой заданной гипотезы вам необходимо определенное количество выборок для решения задачи в пределах допустимого ограничения ошибки. В 2006 г. математики разработали алгоритм на базе Q-обучения, основанный на этом принципе, который получил название *отложенного Q-обучения* [5].

Вместо того чтобы обновлять функцию ценности действия каждый раз, когда агент посещает пару "состояние — действие", он буферизует вознаграждения. После того как агент посетил "состояние — действие" определенное количество раз, он однократно обновляет основную функцию ценности действия.

Идея, опять же, в силу закона больших чисел, заключается в том, что одна оценка ожидаемой доходности может быть зашумленной. Следовательно, не нужно обновлять основную таблицу ценности действия потенциально ошибочными значениями, потому что агент использует их для управления будущими агентами. Рекомендуется немного подождать и выполнить обновление только тогда, когда появится репрезентативная выборка.

Обратной стороной этого подхода является то, что он дает хорошие результаты, но с некоторым запаздыванием. Правда, как только вы на него перейдете, обучение будет происходить очень быстро, потому что агент будет избавлен от излишней зашумленности.

В каком-то смысле это просто способ продлить период "разведки". Вы получите аналогичный эффект, если превратите алгоритм $\varepsilon$ в $\varepsilon$-жадный или используете UCB (*см. разд. "Улучшение $\varepsilon$-жадного алгоритма" главы 2*). Аналогичная реализация, называемая направленным отложенным Q-обучением, также включает бонус за исследование для пар "действие — состояние", которые были отобраны не лучшим образом [6].

## Сравнение стандартного, двойного и отложенного Q-обучения

На рис. 3.6 сравнивается производительность стандартного, двойного и отложенного Q-обучения в среде-сетке. Реализация двойного Q-обучения приводит к лучшей политике, чем стандартное Q-обучение, что выражается в более высоком среднем результате (он будет улучшаться, если вы уделите обучению больше времени; сравните с рис. 3.2). Очевидные разрывы отложенного Q-обучения вызывают беспокойство в отношении надежности, но конечный результат очень впечатляет. Это связано с тем, что агрегирование многих обновлений обеспечивает более надежную оценку функции ценности действия. Область с горизонтальной линией вверху возникает из-за того, что отложенное Q-обучение намеренно прекращает обучение после определенного количества обновлений.

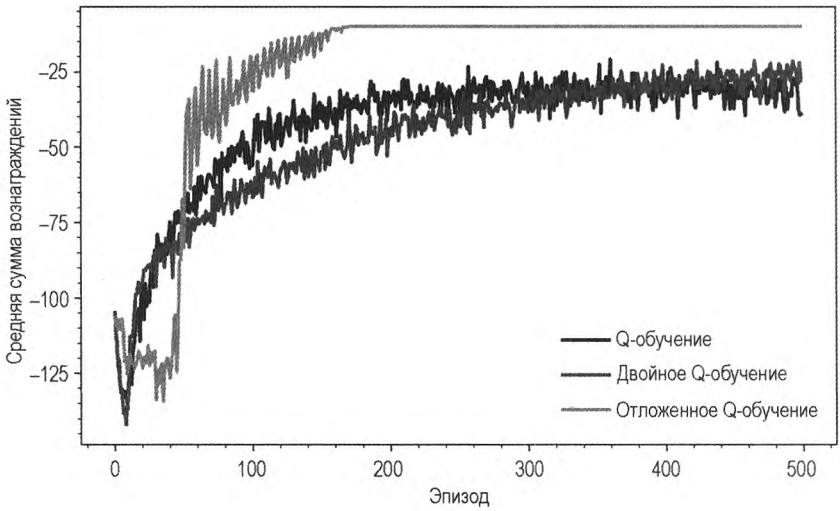



**Рис. 3.6.** Сравнение усредненных вознаграждений за стандартное, двойное и отложенное Q-обучение в среде-сетке

## Обучение с подкреплением на основе противодействия

В качестве интересного дополнения Тижуш попытался улучшить эффективность обучения, намеренно выбирая вариант, противоположный тому, что выбирает политика [7]. Известная как обучение с подкреплением на основе противодействия (opposition-based RL), реализация, вдохновленная Q-обучением, намеренно выбирает и обновляет противоположное действие, а также действие, выбранное в соответствии с текущей политикой.

Суть этой идеи в том, что противоположное действие может выявить революционный или противоречащий фактам ход, действие, которое политика никогда бы не рекомендовала.

Если существует естественная противоположность, то выбор ее, вероятно, приведет к более быстрому обучению на ранних этапах, потому что это предотвращает худший сценарий. Однако если она не существует, вы также можете выполнять выборку случайным образом, что равносильно обычному и случайному обновлению. Если это приемлемо, я могу схитрить и пробовать все действия.

Основная проблема заключается в том, что во многих средах невозможно тестировать несколько действий одновременно. Например, представьте, что вы пытались использовать этот алгоритм на роботе. Он должен будет выполнить противоположное (или случайное) действие, вернуться назад, а затем выполнить действие политики. Действие возврата может быть неточным и приводить к ошибкам. Это также не соответствует непрерывным доменам. Лучшим подходом является использование нескольких агентов или отдельных критических политик для создания противодействий.

# *n*-Шаговые алгоритмы

В этой главе рассматривается одношаговый прогноз. Но зачем останавливаться на достигнутом? Почему бы не сделать двухшаговый прогноз? Или больше? Именно в этом суть *n*-шаговых алгоритмов. Они обеспечивают обобщение развертывания оценки ожидаемого значения на любое количество шагов. Это полезно, поскольку устраняет разрыв между динамическим программированием и методом Монте-Карло.

Для некоторых прикладных вариантов имеет смысл заглянуть на несколько шагов вперед, прежде чем сделать выбор. В примере с сеткой, если агент видит, что текущая траектория ведет его к падению с обрыва, агент может предпринять действия по уклонению сейчас, пока не стало слишком поздно. В некотором смысле агент смотрит в будущее.

Суть идеи состоит в том, чтобы расширить одну из реализаций обучения с учетом временны́х различий. Например, уравнение 3.3 используется для перебора любого количества будущих состояний. Напомним, что функция прогнозирования пришла из динамического программирования (см. уравнение 3.3), которое объединило текущее вознаграждение с предсказанием следующего состояния. Я могу расширить это, чтобы посмотреть на следующие две награды и ожидаемое возвращение состояния после, как показано в уравнении 3.7.

**Уравнение 3.7. Двухшаговая ожидаемая доходность**

$$G_{t\,t+2} \doteq r + \gamma r' + \gamma^2 Q(s'', a'').$$

Я добавил дополнительное обозначение $t+2$, чтобы показать, что уравнение 3.7 просчитывает ситуацию на два шага вперед. Уравнение 3.3 эквивалентно $G_{t\,t+1}$. Суть в том, что в уравнении 3.7 используется не только следующая награда, но и награда после нее. Точно так же оно использует не следующую оценку ценности действия, а оценку, сделанную постфактум. Общая форма приведена в уравнении 3.8.

**Уравнение 3.8. *n*-Шаговая ожидаемая доходность**

$$G_{t\,t+n} \doteq r + \gamma r' + \ldots + \gamma^{n-1} r^{(n-1)} + \gamma^n Q\left(s^{(n)}, a^{(n)}\right).$$

Вы можете дополнить различные правила обновления TD-обучения (например, см. уравнение 3.6), чтобы использовать новое обновление динамического программирования. В уравнении 3.9 представлено правило для обновления функции ценности действия.

**Уравнение 3.9. *n*-Шаговое правило обновления TD-обучения для оценки ожидаемого дохода с использованием функции ценности действия**

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(G_{t\,t+n} - Q(s,a)\right).$$

Уравнение 3.9 показывает $n$-шаговое обновление функции ценности действия. Вы можете переписать его, чтобы представить функцию ценности состояния или сделать его Q-обучением, добавив $\arg\max_{a_t \in \mathcal{A}(s)}$ внутрь $G_{t:t+n}$. В некоторых источниках приводятся дополнительные математические обозначения, в соответствии с которыми новое значение оценки ценности действия $Q$ зависит от предыдущей оценки. Например, вы должны обновить $Q$, прежде чем использовать $Q'$. Я опустил их, чтобы упростить уравнение.

Но эта идея проблематична. Награды доступны *после* перехода агента в состояние. Таким образом, агент должен следовать по траектории, буферизовать награды, а затем вернуться и обновить исходное состояние, бывшее $t$ шагов назад.

Когда мои дочери спят и видят сны, можно сказать, что они заново переживают прошедший день. Они восстанавливают свой опыт, обрабатывают его, опираясь на прошлый опыт, и извлекают уроки из этого опыта. Агенты RL могут делать то же самое. Некоторые алгоритмы используют *буфер воспроизведения опыта*, чтобы позволить агенту анализировать свои предыдущие действия и учиться на них. Это хранилище траектории текущего эпизода. Агент может оглянуться назад, чтобы вычислить уравнение 3.8.

$n$-Шаговый алгоритм 3.3 функционально такой же, как алгоритм 3.2, но на вид сложнее, потому что ему нужно выполнять итерацию в обратном направлении по буферу воспроизведения, и он может сделать это только после того, как в наличии будет достаточно выборок. Например, в двухэтапном алгоритме SARSA вам нужно дождаться третьего шага, прежде чем вы сможете вернуться и обновить первый, т. к. вам необходимо отследить вторую награду.

**Алгоритм 3.3. SARSA $n$-шаговый**

**input**: политика, которая использует функцию ценности действия $\pi(a \mid s, Q(s,a))$, размер шага $0 < \alpha < 1$, положительное целое $n$.

Инициализировать $Q(s,a) \leftarrow 0$ для всех $s \in \mathcal{S}, a \in \mathcal{A}(s)$.

**loop** для каждого эпизода:

    Инициализировать $T \leftarrow \infty, t \leftarrow 0$, буферы воспроизведения для $S$ и $A$ и начальное состояние $s$.

    Выбрать действие $a$ из состояния $s$, используя политику $\pi$ (неоднозначности разрешить случайным образом).

    **do**:

        **if** $t < T$:

            Выполнить действие $a$, а затем наблюдать за наградой $r$ и следующим состоянием $s'$.

            **if** состояние $s'$ является конечным:

                $T \leftarrow t+1$

11: **else**:

12: Выбрать действие $a'$ из состояния $s'$, используя политику $\pi$ (неоднозначности разрешить случайным образом).

13: $\tau \leftarrow t - n + 1$

14: **if** $\tau \geq 0$:

15: $G \leftarrow \sum_{i \leftarrow \tau+1}^{\min(\tau+n, T)} \gamma^{i-\tau-1} r_i$

16: **if** $\tau + n < T$:

17: $G \leftarrow G + \gamma^n Q(s_{\tau+n}, a_{\tau+n})$

18: $Q(s_\tau, a_\tau) \leftarrow Q(s_\tau, a_\tau) + \alpha[G - Q(s_\tau, a_\tau)]$

19: $t \leftarrow t + 1,\ s \leftarrow s',\ a \leftarrow a'$

20: **while** $\tau \neq T - 1$

Алгоритм 3.3 выглядит пугающе, но обещаю, что дополнительный псевдокод будет повторяться по буферам воспроизведения. Идея в точности такая же, как у SARSA, за исключением того, что она рассчитана на несколько временны́х шагов, а это означает, что вы должны буферизовать опыт. Смысл апострофа тоже немного иной, потому что вам нужно индексировать предыдущие действия, состояния и награды. Думаю, что этот алгоритм подчеркивает, насколько сложна реализация некоторых подобных алгоритмов, — и это еще простой алгоритм.

Шаг 1 начинается как обычно, здесь от вас требуется ввести политику и параметры, которые управляют количеством шагов в *n*-шаге и размером шага ценности действия. Шаг 2 инициализирует таблицу ценности действия. Шаг 3 выполняет итерацию по каждому циклу, а шаг 4 инициализирует: переменную *T*, представляющую номер шага в конце эпизода *t*, который представляет номер текущего шага; буферы воспроизведения и *s* — состояние эпизода по умолчанию. Шаг 5 выбирает первое действие на основе состояния, заданного по умолчанию и текущей политики, а шаг 6 является началом основного цикла.

По большей части всё происходит так же, как и раньше. Шаг 7 отличается проверкой, закончился ли уже эпизод. В случае успешности проверки больше ничего предпринимать не нужно. А до тех пор шаг 8 выполняет действие в среде.

Шаг 9 проверяет, привело ли это действие к концу эпизода. В случае успеха проверки на шаге 10 устанавливается переменная *T* для обозначения шага, на котором эпизод подошел к концу. В противном случае шаг 12 выбирает следующее действие.

Шаг 13 обновляет переменную, указывающую на пару "состояние — действие", которая была $n$ шагов назад. Первоначально это будет до $t = 0$, поэтому на шаге 14 выполняется проверка для предотвращения ошибок выхода индекса за пределы.

Если $\tau \geq 0$, то алгоритм 3.3 начинает обновлять функции ценности действия с ожидаемой отдачей.

На шаге 15 вычисляется вознаграждение за $n$-шаговый ожидаемый доход, определенный в уравнении 3.7. Шаг 16 проверяет, дошел ли алгоритм до момента, означающего конец эпизода плюс $n$. Если не дошел, добавьте прогноз следующего состояния к ожидаемому доходу, как определено в правой части уравнения 3.7.

Шаг 18 такой же, как и раньше; он обновляет функцию ценности действия, используя оценку ожидаемого значения за $n$ шагов. Наконец, шаг 19 увеличивает счетчик шагов. Алгоритм возвращается к основному циклу на шаге 20 после обновления всех шагов, повторяющемуся до завершения.

## $n$-Шаговые алгоритмы в распределенных средах

Я реализовал $n$-шаговый агент SARSA в распределенной среде и установил количество шагов равными 2 и 4. Все остальные настройки такие же, как и в стандартной реализации SARSA. Разница между стандартным и $n$-шаговым агентом SARSA показана на рис. 3.7.

Из рис. 3.7 очевидно, что $n$-шаговые алгоритмы способны изучать оптимальные политики быстрее, чем их одношаговое аналоги. Это интуитивно понятно. Если вы способны заглядывать в "будущее" (в кавычках, потому что на самом деле вы откладываете обновления), то можете узнать оптимальные траектории раньше. Оптимальное значение $n$ зависит от конкретной задачи и должно рассматриваться как гиперпараметр, требующий настройки. Главный недостаток этих методов — чрезмерная вычислительная нагрузка и нагрузка на память.

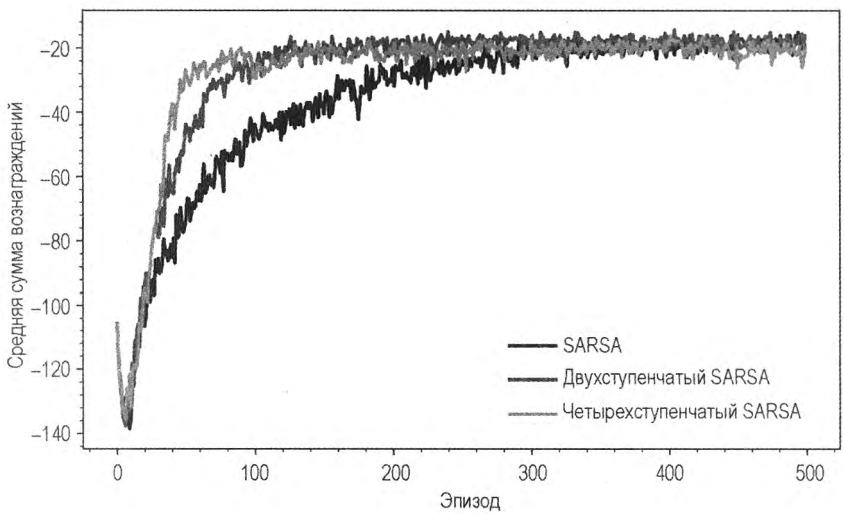



**Рис. 3.7.** Сравнение усредненных вознаграждений стандартных и $n$-ступенчатых агентов SARSA в среде-сетке. Обратите внимание, как агенты, использующие несколько шагов, быстрее усваивают оптимальные траектории

Хотя этот раздел посвящен политике SARSA, вы, конечно, также можете создать реализации вне политики и не-SARSA реализации *n*-шагового TD-обучения. Подробности можете прочитать в книгах, перечисленных в *разд. "Дополнительные материалы для чтения" в конце этой главы*.

## Трассировки соответствия

В *разд. "n-Шаговый алгоритм" ранее в данной главе* был разработан метод, обеспечивающий преимущества как самозагрузки, так и перспективного планирования. Расширением этой идеи является усреднение по различным значениям *n*, например по 2- и 4-шаговому SARSA. Можете довести эту идею до крайности и усреднить по всем значениям *n*.

Конечно, повторение различных *n*-шаговым обновлений (уравнение 3.9) значительно увеличивает вычислительную сложность. Кроме того, по мере увеличения *n* агенту придется ждать все дольше и дольше, прежде чем он сможет начать обновление первого временно́го шага. В крайнем случае, это будет так же плохо, как в ситуации ожидания конца эпизода, прежде чем агент сможет внести какие-либо обновления (например, в случае методов Монте-Карло).

Проблема с алгоритмами, которые мы видели до сих пор, заключается в том, что они пытаются заглянуть в будущее, что невозможно, поэтому они откладывают обновления и делают вид, будто с нетерпением ждут. Одно из решений — смотреть назад во времени, а не вперед. Если вы захотите подняться на гору в кратчайшие сроки, то можете попробовать несколько маршрутов, а затем обдумать свои действия, чтобы найти лучшую траекторию.

Другими словами, агенту требуется новый механизм для пометки пары "состояние — действие" как отработанной, чтобы он мог вернуться и уточнить это состояние с помощью будущих обновлений. Вы можете использовать *индикаторы*, чтобы отметить местоположение чего-то интересного. Радиоактивные индикаторы помогают диагностировать болезни и подбирать лекарства. Геофизики вводят их в породу при гидроразрыве пласта, чтобы получить данные о размере и местоположении трещин. Вы можете использовать виртуальный трассировщик, чтобы напомнить агенту о том, какие оценки пары "действие — состояние" нуждаются в обновлении в будущем.

Затем вам надо обновить функцию "состояние — действие", указав среднее значение всех возвращаемых значений за *n* шагов. Одноэтапное обновление — это обновление TD-ошибок (см. уравнение 3.6). Вы можете аппроксимировать среднее значение, используя экспоненциально взвешенное скользящее среднее, которое является онлайн-эквивалентом среднего (как показано в *разд. "Оценка стратегии: функция ценности" главы 2*). Заинтересованные читатели могут найти доказательство того, что это эквивалентно усреднению многих значений доходности за *n* шагов и что оно сходится, в *разд. "Дополнительные материалы для чтения" в конце этой главы*.

Сочетание одношагового обновления со скользящим средним и дополнением идеей трассировщика обеспечивает онлайн-вариант SARSA с нулевой задержкой (без

прямого $n$-шагового прогнозирования), но со всеми преимуществами перспективного планирования и самозагрузки. Поскольку вы уже видели математические выкладки раньше, позвольте мне погрузиться в алгоритм 3.4.

**Алгоритм 3.4. SARSA(λ)**

1: **input**: политика, которая использует функцию ценности действия $\pi(a \mid s, Q(s,a))$, размер шага $0 < \alpha < 1$, скорость затухания индикатора $0 \le \lambda \le 1$.

2: Инициализировать $Q(s,a) \leftarrow 0$ для всех $s \in \mathcal{S}, a \in \mathcal{A}(s)$.

3: **loop** для каждого эпизода:

4: Инициализировать $z(s,a) \leftarrow 0$ для всех $s \in \mathcal{S}, a \in \mathcal{A}(s)$, а также состояние по умолчанию $s$.

5: Выбрать действие $a$ из состояния $s$, используя политику $\pi$ (неоднозначности разрешить случайным образом).

6: **do**:

7: Выполнить действие $a$, а затем наблюдать за наградой $r$ и следующим состоянием $s'$.

8: Выбрать действие $a'$ из состояния $s'$, используя политику $\pi$ (неоднозначности разрешить случайным образом).

9: $\delta \leftarrow r + \gamma Q(s',a') - Q(s,a)$.

10: $z(s,a) \leftarrow z(s,a) + 1$.

11: **for** каждого $s \in \mathcal{S}$:

12: **for** каждого $a \in \mathcal{A}(s)$:

13: $Q(s,a) \leftarrow Q(s,a) + \alpha\delta z(s,a)$.

14: $z(s,a) \leftarrow \gamma\lambda z(s,a)$.

15: $s \leftarrow s', a \leftarrow a'$.

16: **while** состояние $s$ не станет конечным.

Бóльшая часть кода в алгоритме 3.4 дублирует алгоритм 3.2, поэтому я не буду повторяться. Различия начинаются на шаге 9, где алгоритм 3.4 вычисляет одношаговую TD-ошибку, чтобы использовать ее позже. Шаг 10 реализует средство трассировки, увеличивая на единицу значение той ячейки в таблице, которая представляет текущую пару "состояние — действие". Таким образом, алгоритм "запоминает", какие состояния и действия были затронуты им во время движения по траектории.

В шагах 11 и 12 перебираются все состояния и действия, нужные для обновления функции ценности действия на шаге 13. Здесь алгоритм дополнительно взвешивает

TD-ошибку по текущему значению индикатора, которое экспоненциально уменьшается на шаге 14. Другими словами, если агент затронул пару "состояние — действие" давным-давно, обновление будет незначительным. Если же это произошло на последнем временно́м шаге, будет выполнено большое обновление.

Подумайте об этом. Если получили свое образование давным-давно, насколько это влияет на вашу сегодняшнюю профессиональную деятельность? Возможно, влияние сказывается, но не сильно. Вы можете контролировать степень влияния с помощью параметра $\lambda$.

На рис. 3.8 показаны результаты этой реализации в сравнении со стандартным SARSA. Обратите внимание, что параметр $\lambda$ оказывает аналогичный эффект, как и при изменении $n$ в $n$-шаговом SARSA. Более высокие значения имеют тенденцию к режиму MC, при котором увеличивается количество усредненных шагов. Более низкие значения тяготеют к чистому TD, что эквивалентно исходной реализации SARSA.

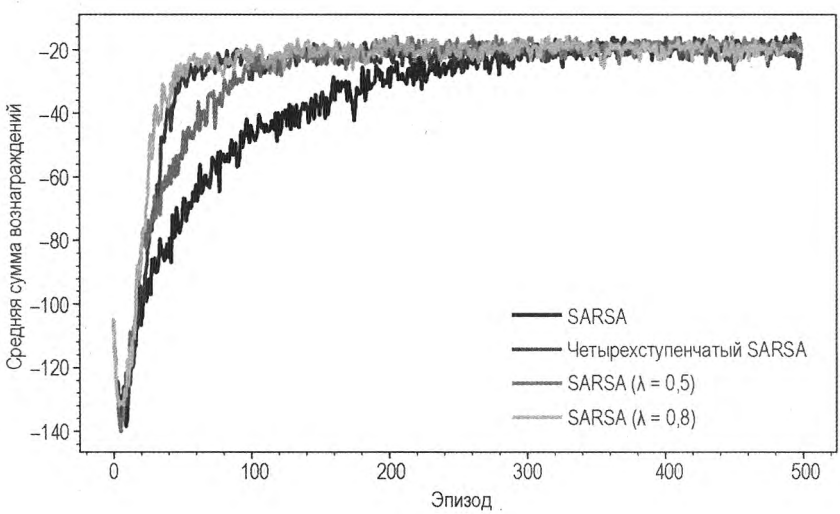



**Рис. 3.8.** Сравнение усредненных вознаграждений агентов SARSA и SARSA($\lambda$) в среде-сетке. Все настройки такие же, как на рис. 3.7

Если вам интересно, термин *"соответствие"* основан на идее, что трассировщик решает, подходит ли конкретная пара "состояние — действие" в функции ценности действия для обновления. Если индикатор для пары имеет ненулевое значение, то он получит некоторую долю вознаграждения.

*Трассировки соответствия* предоставляют собой эффективный с вычислительной точки зрения управляемый гиперпараметр, который плавно изменяет режим обучения между методами обучения с учетом временны́х различий (TD) и методами Монте-Карло (MC). Но возникает проблема: где разместить этот параметр? Исследователи не нашли теоретической основы для выбора $\lambda$, но опыт дал общие правила, сформированные эмпирическим путем. В задачах с большим количеством шагов на эпизод имеет смысл обновлять не только текущий шаг, но и все шаги, кото-

рые привели к этому моменту. Подобно бортикам дорожек для боулинга, которые направляют мяч к кеглям, обновление оценок ценности действия на основе предыдущей траектории помогает направлять движение будущих агентов. В общем случае для нетривиальных задач стоит использовать $\lambda > 0$.

Однако высокие значения $\lambda$ означают, что ранние пары "состояние — действие" получают обновления, даже если маловероятно, что они способствовали текущему результату. Рассмотрим пример сетки с $\lambda \doteq 1$, что делает его эквивалентным примеру с агентом на основе метода Монте-Карло. Первый шаг всегда будет получать обновления, потому что это единственное направление, в котором может двигаться агент. Это означает, что независимо от того, где окажется агент, все действия в первом состоянии будут сходиться примерно к одному и тому же значению, т. к. все они зависят от любого будущего шага. Это мешает агенту изучить какую-либо жизнеспособную политику. В нетривиальных задачах следует держать $\lambda < 1$.

Единственным недостатком является дополнительная вычислительная сложность. Для трассировки требуется еще один буфер. Вам также необходимо перебрать все состояния и действия на каждом шаге, чтобы увидеть, есть ли для них подходящие обновления. Вы можете упростить эту задачу, используя гораздо меньший список допустимости, поскольку большинство пар "состояние — действие" будут иметь значение, близкое к нулю. В целом преимущества перевешивают незначительную вычислительную нагрузку.

# Расширения для трассировки соответствия

Рискуя углубиться в эту тему, исследователи разработали множество настроек основного алгоритма. Некоторые получили большую известность, чем другие, и, конечно же, решение о том, какие улучшения важны, открыто для обсуждения. Вы можете взять любую из предыдущих настроек и применить их здесь. Например, вы можете создать эквивалент Q-обучения и применить все настройки, описанные в *разд. "Расширения для Q-обучения" данной главы*. Но позвольте мне представить подборку наиболее известных расширений, которые конкретно относятся к трассировке соответствия.

## Алгоритм обучения Q(λ) Уоткинса

В большей части литературы Q(λ) реализуется как не связанный с политикой алгоритм с выборкой по значимости. До сих пор в этой книге я рассматривал только стандартные неисправленные версии, не учитывающие политики. Это обсуждается позже в *разд. "Выборка по значимости" главы 6*.

Одно из простых эмпирических улучшений стандартного $Q(\lambda)$-обучения, которое реализует arg max при обновлении TD, заключается в том, чтобы сбрасывать трассировку соответствия, когда агент обнаруживает первое же нежелательное действие. Это оправданно, потому что нет желания наказывать агента за предыдущие шаги как за неправильные, когда эти шаги были частью исследования новых со-

стояний. Обратите внимание, что это не имеет смысла в настройке SARSA, потому что цель SARSA — изучить все состояния и усреднить их. Если бы вы реализовали это, он работал бы немного хуже, чем стандартный SARSA(λ), потому что вы ограничиваете возможности обновления.

## Нечеткие стирания в алгоритме обучения Q(λ) Уоткинса

Подумайте, что бы произошло, если бы вы стерли все свои воспоминания прямо перед тем, как попробовать что-то новое. Как вы думаете, приведет ли это к оптимальному обучению? Возможно, нет. Стирать в памяти все следы всякий раз, когда агент выполняет исследовательский ход, как в случае $Q(\lambda)$-обучения Уоткинса, — это крайность.

Вместо этого при нечетком обучении алгоритм пытается решить, в самом ли деле уместно стирание [8]. В рамках этой недавней разработки предлагается использовать правила принятия решений, чтобы установить, следует ли стирать трассировку соответствия для пары "состояние — действие". Такой подход похож на решение, взятое "с потолка", поэтому я ожидаю, что исследователи скоро найдут новые, более простые альтернативы. Несмотря на жесткий подход, вроде бы это ускоряет обучение.

## Быстрое Q-обучение

Быстрое Q-обучение похоже на трассировку соответствия требованиям, но реализовано несколько иначе [9]. Данный подход обновляет функцию ценности действия, экспоненциально уменьшая сумму предыдущего и текущего обновлений TD. Результатом является усреднение по одношаговым обновлениям TD. Но здесь не предусмотрено обновление ранее посещенных состояний будущими наградами. Другими словами, способ обеспечивает улучшения по сравнению со стандартным Q-обучением, но не позволит агенту обучаться так быстро, как это делают методы, которые могут адаптивно загружать все предыдущие пары "состояние — действие".

## Накопление или замена трассировок соответствия

Исследования показывают, что метод накопления трассировок соответствия вместо замены может иметь некоторое влияние на эффективность обучения [10]. Накопительный трассировщик похож на метод, реализованный в алгоритме 3.4. Индикатор для пары "состояние — действие" увеличивается на некоторое значение. Если агент должен неоднократно посещать состояние, например если пара "состояние — действие" находилась на оптимальной траектории, то трассировщик становится очень большим. Возникает петля обратной связи, которая подталкивает будущие сценарии обучения в сторону одного и того же состояния. В каком-то смысле это хорошо, т. к. система научится очень быстро, и в то же время плохо, потому что подавляется исследовательская активность.

Рассмотрим ситуацию, когда среда не является марковским процессом принятия решения (MDP), например частично наблюдаемым MDP, где состояние не полно-

стью наблюдаемо, когда ваш агент не может заглядывать за угол, или полу-MDP, в котором переходы являются стохастическими. Я вернусь к ним в *главе 8*. В таких ситуациях слепое следование одному и тому же пути может быть неоптимальным, поскольку траектория могла бы быть шумной. Вместо этого имеет смысл доверять исследованию и заменять траекторию, а не накапливать ее. С учетом сказанного, экспериментальные результаты показывают, что разница в производительности невелика и, вероятно, не будет иметь значения для большинства приложений. На практике настройка гиперпараметров оказывается значительно более действенным подходом.

# Резюме

В этой главе обобщены наиболее важные исследования, проведенные на протяжении 1990–2000-х годов. В частности, $Q$-обучение — это алгоритм, который сегодня широко используется в промышленности. Он зарекомендовал себя как надежный, простой в реализации и быстрый в освоении.

Трассировки соответствия работают хорошо, в большинстве случаев лучше, чем Q-обучение, и будут широко использоваться в дальнейшем. Недавние исследования показывают, что люди также применяют трассировку соответствия критериям для решения задачи присваивания коэффициентов доверия [11].

Помните, что вы можете комбинировать любое количество этих или предыдущих настроек для создания нового алгоритма. Например, есть Q-обучение с двойной задержкой, которое, как вы можете себе представить, является комбинацией двойного и отложенного Q-обучения [12]. Вы можете добавить трассировки соответствия критериям и нечеткие стирания, чтобы сгенерировать более эффективный алгоритм.

Обратной стороной алгоритмов, описанных в этой главе, является то, что в них оценки ценности действия или ценности состояния хранятся в таблице. Это означает, что они работают только в дискретных пространствах состояний. Вы можете обойти эту проблему, используя разработку функций (*см. разд. "Разработка политики" главы 9*), но в следующей главе будет исследовано, как обрабатывать непрерывные пространства состояний с помощью аппроксимации функций.

# Дополнительные материалы для чтения

- $n$-Шаговые реализации TD-обучения и доказательства соответствия требованиям.
  - Sutton R. S., Barto A. G. Reinforcement learning: an introduction. — MIT Press, 2018.

# Использованные источники

[1] Пример, адаптированный из книги Sutton R. S., Barto A. G. Reinforcement learning: an introduction. — MIT Press, 2018.

[2] Rossi F, Nardelli M., Cardellini V. Horizontal and vertical scaling of container-based applications using reinforcement learning // 2019 IEEE 12th International Conference on Cloud Computing (CLOUD). — 2019. — P. 329–38. — URL: **https://oreil.ly/tGvuN**.

[3] Zhang W. et al. Real-Time Bidding benchmarking with IPinYou dataset // ArXiv:1407.7073. — 2014. — July. — URL: **https://oreil.ly/l9O_v**.

[4] Hasselt H. V. Double Q-learning. — 2010. — URL: **https://oreil.ly/akRzg**.

[5] Strehl A. L. et al. PAC model-free reinforcement learning // ICML'06: Proceedings of the 23rd International Conference on Machine Learning. — 2006. — P. 881–88. — URL: **https://oreil.ly/Z0mxa**.

[6] Min-hwan O., Iyengar G. Directed exploration in PAC model-free reinforcement learning // ArXiv:1808.10552. — 2018. — August. — URL: **https://oreil.ly/zM1E7**.

[7] Tizhoosh H. Opposition-based reinforcement learning // JACIII 10 (January). — 2006. — P. 578–85.

[8] Shokri M. et al. Adaptive fuzzy Watkins: a new adaptive approach for eligibility traces in reinforcement learning // International Journal of Fuzzy Systems. — 2019. — № 21. — P. 1443–1454. — URL: **https://oreil.ly/8nW3P**.

[9] Ghavamzadeh M. et al. Speedy Q-learning // In Advances in Neural Information Processing Systems 24. — 2011. — URL: **https://oreil.ly/oyIAo**.

[10] Leng J. et al. Experimental analysis of eligibility traces strategies in temporal-difference learning // IJKESDP. — 2008. — Vol. 1, № 1. — P. 26–39. — URL: **https://oreil.ly/OOEF0**.

[11] Lehmann M., Xu H. Liakoni V. et al. One-shot learning and behavioral eligibility traces in sequential decision making // ArXiv:1707.04192. — 2017. — July. — URL: **https://oreil.ly/syD_E**.

[12] Abed-alguni B. H., Ottom M. A. Double delayed Q-learning // International Journal of Artificial Intelligence. — 2019. — Vol. 16, № 2. — P. 41–59.

# ГЛАВА 4

# Глубокие Q-сети

Алгоритмы табличного обучения с подкреплением, такие как Q-обучение или SARSA, выдают оценки ожидаемых значений состояния или пары "состояние — действие" в таблице поиска (также известной как Q-таблица или Q-значения). Вы видели, что этот подход хорошо работает для небольших наборов дискретных состояний. Когда же количество состояний возрастает, размер таблицы увеличивается в геометрической прогрессии. Пространство состояний становится бесконечно большим и в случае непрерывных переменных.

Действия имеют аналогичную проблему. В версии Q-обучения или SARSA, ориентированной на действие, количество действий увеличивает размер таблицы. Если действия продолжаются, таблица становится бесконечной. Однако во многих приложениях желательны именно непрерывные действия. В примере размещения рекламы в *разд. "Отраслевой пример: торги рекламы в режиме реального времени" главы 2* лучше, если агент может предложить значение ставки напрямую, а не полагаться на заранее определенный набор дискретных вариантов.

Наиболее распространенное решение таких проблем — замена таблицы аппроксимационной функцией. Вместо того чтобы пытаться сохранить карту того, как состояния и действия изменяют ожидаемый результат, можете создать функцию, которая его аппроксимирует. На каждом временно́м шаге агент смотрит на текущую пару "состояние — действие" и прогнозирует ожидаемое значение. Это задача регрессии.

Вы можете выбрать один из множества алгоритмов регрессии, чтобы решить эту задачу. Правда, для пар "состояние — действие" наблюдается тенденция быть очень нелинейными и часто прерывистыми. Например, представьте себе пространство состояний окружающей среды обрыва. Ожидаемые значения должны сходиться так, чтобы состояния, близкие к цели, были высокими, а состояния, расположенные дальше от цели, уменьшались пропорционально схеме дисконтирования и вознаграждения. Но рядом с обрывом будет большой разрыв. Если агент использует линейную модель для аппроксимации функции ценности состояния, то большое отрицательное вознаграждение за обрыв действует как выброс по сравнению с остальной частью пространства состояний и искажает модель.

Однако в прикладной сфере часто отдают предпочтение простым моделям, поскольку они хорошо понятны, надежны и легко интерпретируются. Если хотите использовать простую модель, вы должны обеспечить репрезентативность вашего пространства состояний. Для этого вы можете случайным образом выбрать свою

среду и сохранить ее состояния. Затем можете выполнить регрессию в автономном режиме, чтобы оценить, насколько хорошо ваша модель соответствует данным.

Наиболее часто используемая модель — это *глубокая искусственная нейронная сеть* (deep artificial neural network). Сеть использует состояние окружающей среды в качестве входных данных, а выходными данными является ожидаемое значение каждого действия. Аппроксимация способна обрабатывать огромные и сложные пространства состояний. Я откладываю обсуждение непрерывных действий до конца книги.

# Архитектуры глубокого обучения

Для создания искусственной нейронной сети (artificial neural network, ANN) можете сформировать стек линейных аппроксиматоров и изменять каждый с помощью нелинейной функции активации. Затем обучите сеть нахождению оптимальных весов с помощью регрессии. В итоге получится модель, которая может аппроксимировать любую произвольную функцию.

Это тот результат, который обещает глубокое обучение (deep learning, DL), выступающее прозвищем для "глубоких" стеков ANN. Алан Тьюринг мог бы описать их как одну из своих "универсальных вычислительных машин". Изменяя структуру и гиперпараметры ANN, агенты могут моделировать произвольное пространство состояний.

Я кратко рассмотрю DL, чтобы определить контекст для глубокого RL. Но это не книга по глубокому обучению или машинному обучению; если вы хотите узнать больше, обратитесь к ссылкам в *разд. "Дополнительные материалы для чтения" в конце этой главы*.

## Основные положения

Нейронные сети состоят из множества взаимосвязанных функций, называемых *нейронами*, которые действуют согласованно для решения задач классификации или регрессии. Слой — это набор нейронов с входами и выходами. Вы можете накладывать слои друг на друга, чтобы ANN могла изучать абстракции данных. Слой, связанный с данными, является *входным слоем*. Слой, производящий выходные данные, является *выходным слоем*. Все промежуточные слои являются *скрытыми*.

Значение каждой связи, если связь находится между входами, другими нейронами или выходами, умножается на *вес*. Нейрон суммирует (скалярные) произведения входных данных и весов. Затем вы тренируете веса для оптимизации функции ошибок, которая представляет собой разницу между выходом ANN и истинным значением. Представленные таким образом ANN полностью линейны. Если вы выполняли классификацию, тогда простая ANN является линейным классификатором. То же является верным и для регрессии.

Выход каждого нейрона проходит через нелинейную *функцию активации*. Выбор функции является гиперпараметром, но все функции активации по определению

являются нелинейными или разрывными. Это позволяет ANN аппроксимировать нелинейные функции или выполнять нелинейную классификацию.

ANN "учатся", изменяя значения весов. При каждом обновлении программа обучения корректирует веса пропорционально градиенту функции ошибок. Ключевая идея состоит в том, что вы обновляете веса итеративно, многократно стимулируя черный ящик и уточняя его оценку. При наличии достаточного количества данных и времени ANN будет предсказывать точные значения или классы.

## Архитектуры нейронных сетей

Глубокие ANN бывают самых разных форм и размеров. Все они, как правило, основаны на идее объединения нейронов. Термин *"глубокие"* — это отсылка к большому количеству скрытых слоев, необходимых для моделирования абстрактных концепций. Но конкретная архитектура, вообще, зависит от предметной области.

*Многослойные персептроны* (multilayer perceptrons, MLP) — самая простая и традиционная архитектура глубокого обучения. Несколько уровней нейронных сетей *полностью связаны* и имеют *прямую связь*; выходной сигнал каждого нейрона в более высоком слое направляется каждому нейрону в слое, находящемся ниже. Количество нейронов во входном слое такое же, как размер данных. Размер выходного слоя устанавливается равным количеству классов и часто обеспечивает распределение вероятностей по классам, передавая нейроны через функцию *softmax*.

*Глубокие сети доверия* (deep belief networks, DBN) похожи на MLP, за исключением того, что связи между двумя (или более) верхними уровнями неориентированы; информация может передаваться от второго уровня к первому. Ненаправленные слои — это ограниченные машины Больцмана (restricted Boltzmann machines, RBM). RBM позволяют моделировать данные, а слои MLP — проводить классификацию на основе модели. Это полезно, потому что модель способна извлекать скрытую информацию — такую, которую нельзя наблюдать напрямую. Однако обучение RBM становится сложнее по мере увеличения размера сети.

*Автоэнкодеры* имеют архитектуру, которая сужается к середине, как песочные часы. Цель состоит в том, чтобы как можно лучше воспроизвести некоторые входные данные, учитывая ограничение сужения. Например, если в середине было два нейрона и соответствующее воспроизведение было приемлемым, то вы могли бы использовать выходы двух нейронов для представления ваших данных вместо необработанных данных. Другими словами, это форма сжатия. Многие архитектуры включают автоэнкодеры как форму автоматического извлечения признаков.

*Сверточные нейронные сети* (convolutional neural network, CNN) хорошо работают в областях, где отдельные наблюдения локально коррелированы. Например, если один пиксел изображения черный, то окружающие пикселы также могут быть черными. Это означает, что CNN хорошо подходят для изображений, естественного языка и временны́х рядов. Основная предпосылка заключается в том, что CNN предварительно обрабатывает данные с помощью набора фильтров, называемых *свертками*. После прохождения нескольких слоев фильтров результат передается

в MLP. В процессе обучения оптимизируются фильтры и MLP для решаемой задачи. Основное преимущество заключается в том, что фильтры позволяют архитектуре в целом зависеть от времени, поворота и перекоса, если эти примеры существуют в тренировочных данных.

*Рекуррентные нейронные сети* (recurrent neural network, RNN) — это класс архитектур, которые возвращают результат одного временно́го шага на вход следующего шага. Учитывая данные, которые коррелированы во времени, такие как текст или временные ряды, RNN могут "запоминать" прошлое и использовать эту информацию для принятия решений. Известно, что RNN сложно обучить из-за петли обратной связи. Долгая краткосрочная память (long short-term memory, LSTM) и закрытые рекуррентные блоки (gated recurrent units, GRU) улучшают RNN за счет включения *шлюзов* для сброса предыдущей "истории" и отключения деструктивного контура обратной связи, как выпускной клапан в двигателе с турбонаддувом.

*Сети эхо-состояний* (echo state networks, ESN) используют случайно инициализированный "пул" RNN для преобразования входных данных в большее количество измерений. Основным преимуществом ESN является то, что вам не нужно обучать RNN; вы тренируете преобразование в обратном порядке — из многомерного пространства к своей проблеме, возможно, используя что-то столь же простое, как логистический классификатор. Так снимаются все проблемы, связанные с обучением нейронных сетей вообще и рекуррентных в частности.

## Фреймворки глубокого обучения

Я всегда рекомендую написать собственную простую библиотеку нейронной сети, чтобы облегчить понимание, но для промышленных приложений вы должны использовать библиотеки Intel, Mathworks, Wolfram и другие подобные. Но я остановлюсь на открытом исходном коде.

Библиотеки глубокого обучения с открытым исходным кодом имеют две формы: реализацию и оболочку. Библиотеки реализации отвечают за определение нейронной сети (часто как *ориентированный ациклический граф* — directed acyclic graph, DAG), оптимизацию выполнения и абстрагирование вычислений. Библиотеки оболочки представляют еще один уровень абстракции, который позволяет определять нейронную сеть в концепциях высокого уровня.

После многих лет жесткой конкуренции самые популярные и активно поддерживаемые фреймворки реализации глубокого обучения — это TensorFlow, PyTorch и MXNet. TensorFlow достаточно гибок, чтобы обрабатывать все архитектуры и варианты использования нейронных сетей, но он низкоуровневый, упрямый и сложный. У него высокий барьер для входа, но он хорошо поддерживается и работает везде, где только возможно. PyTorch имеет более низкий барьер для входа и замечательную функцию, которая позволяет изменять граф вычислений в любой момент во время обучения. API достаточно высокого уровня, что упрощает его использование. MXNet является частью Apache Foundation, в отличие от двух предыдущих, которые поддерживаются компаниями Google и Facebook соответственно. MXNet имеет привязки к разным языкам и масштабируется лучше, чем любая другая библиотека.

Keras, вероятно, самая известная библиотека-оболочка. Вы можете написать высокоуровневый API, где каждая строка кода представляет слой в вашей сети, который затем можно развернуть в TensorFlow, Theano или CNTK (последние два сейчас не функционируют). Отдельная библиотека также позволяет использовать MXNet. PyTorch имеет собственный набор библиотек-оболочек, причем Ignite и Skorch являются наиболее популярными универсальными высокоуровневыми API-интерфейсами. Gluon — это оболочка для MXNet. Наконец, ONNX — это другой тип оболочки, который призван стать стандартизированным форматом для моделей нейронных сетей. Он способен преобразовывать обученные модели из одной структуры в другую для прогнозирования. Например, вы можете тренироваться в PyTorch, а работать в MXNet.

Если бы вы вынудили меня дать рекомендацию, я бы предложил начать с оболочки Keras или Gluon, а затем перейти на PyTorch (потому что он тоже имеет высокоуровневый API). Но у TensorFlow сильные промышленные возможности, так что это тоже хороший выбор. MXNet становится все более популярной по соображениям производительности. Таким образом, подойдет любая библиотека. ONNX может стать стандартом для моделей в будущем по этой же причине.

## Глубокое обучение с подкреплением

Как глубокое обучение вписывается в RL? Помните, в начале данной главы я упоминал о моделях для работы со сложными пространствами состояний (*сложными* в том смысле, что они непрерывны или имеют большое количество измерений).

Простые модели, такие как линейные, могут работать в простых средах, в которых отображение ожидаемого значения изменяется линейно. Но многие среды существенно сложнее. Например, рассмотрим вновь среду размещения ставок на рекламу из *разд. "Отраслевой пример: торги рекламы в режиме реального времени" главы 3*. Если бы нас интересовала скорость расходов во времени, как в примере, то линейная модель хорошо бы соответствовала этой задаче. Но если ваша реклама подходит для определенного возрастного диапазона и вы добавите нелинейную функцию, например информацию о возрасте человека, просматривающего рекламу, тогда линейная модель будет плохо подходить.

Использование глубокого обучения позволяет агентам принимать любую форму информации и делегировать моделирование ANN. А если у вас есть состояние, основанное на изображении? Не проблема для модели, использующей CNN. Данные временно́го ряда? Какая-нибудь рекуррентная нейронная сеть. Многомерные мультисенсорные данные? Многослойный персептрон. Глубокое обучение эффективно превращается в инструмент, используемый для преобразования необработанных наблюдений в действия. RL не заменяет машинное обучение, а дополняет его.

Использовать глубокое обучение в RL сложнее, чем в ML. В управляемом машинном обучении вы улучшаете модель, оптимизируя меру производительности по сравнению с реальной ситуацией. В RL агенту часто приходится долго ждать, чтобы вообще получить какую-либо обратную связь. Глубокое обучение печально известно тем, что для тренировки хорошей модели требуется много данных, но в сочетании с тем фактом, что агенту приходится случайно наткнуться на награду, это

может занять целую вечность. По этой и многим другим причинам исследователи разработали набор уловок, чтобы помочь модулям с глубоким обучением учиться в разумные сроки.

# Глубокое Q-обучение

Еще до 2013 г. исследователи поняли, что им нужна аппроксимация функций для решения задач с большими или сложными пространствами состояний. Они доказали, что использование линейного приближения сохраняет те же гарантии сходимости, что и табличные методы. Но они предупредили, что использование аппроксиматоров нелинейных функций может на самом деле расходиться, а не сходиться [1]. Таким образом, бóльшая часть исследований была сосредоточена на улучшении методов линейной аппроксимации.

В конце концов, запретный плод, которым является глубокое обучение, оказался слишком заманчивым для исследователей, и они разработали *глубокую Q-сеть* (deep Q-network, DQN) [2]. Они поняли, что главной проблемой является движущаяся цель. Представьте, что вы пытаетесь прихлопнуть муху. Движение мухи настолько случайное, а реакция настолько быстрая, что только каратист может надеяться поймать ее в воздухе. Если вы подождете, пока муха не перестанет метаться, у вас будет гораздо больше шансов предсказать, где она будет находиться через 500 миллисекунд, и следовательно, у вас больше шансов попасть в нее. У агента такая же проблема. Если данные не являются стационарными, им может быть очень трудно сойтись. В некоторых случаях они могут действительно расходиться, потому что, как в деструктивной петле обратной связи, следующее действие может усугубить ситуацию.

В целом оптимизаторы моделей, такие как стохастический градиентный спуск, предполагают, что ваши данные являются независимыми и одинаково распределенными (independent and identically distributed, IID) случайными величинами. Когда вы завершите выборку вовремя, данные, вероятно, будут коррелированы. Например, определение марковского процесса принятия решений гласит, что следующее наблюдение зависит от предыдущего состояния и предпринятых действий. Это нарушает предположение IID, и модели могут не сойтись.

## Воспроизведение опыта

Первоначальным решением этой задачи было использование *воспроизведения опыта*. Это буфер наблюдений, действий, вознаграждений и последующих наблюдений, который можно применять для обучения модели глубокого обучения. Он позволяет использовать старые данные для тренировки вашей модели, делая обучение более эффективным для выборки, во многом как в *разд. "Трассировки соответствия" главы 3*. Выбирая случайное подмножество при тренировке, вы нарушаете корреляцию между последовательными наблюдениями.

Размер буфера воспроизведения опыта важен. Чжан и Саттон заметили, что неудачно заданный размер буфера может негативно влиять на производительность

[3]. В худшем случае агенты могут "катастрофически забыть" опыт, когда наблюдения выпадают из конца буфера FIFO (first in, first out — первым пришел, первым ушел), который может привести к большим изменениям в политике. В идеале распределение обучающих данных должно соответствовать распределению, наблюдаемому агентом [4].

## Клоны Q-сети

В обновлении исходного алгоритма DQN те же исследователи предложили клонировать Q-сеть. Это приводит к двум нейронным сетям: одна из них — *онлайн-сеть*, которая производит действия, а другая — *целевая сеть*, которая продолжает обучение. После некоторого количества итераций сеть прогнозирования копирует текущие веса из целевой сети. Исследователи обнаружили, что это сделало алгоритм более стабильным и менее склонным к колебаниям или отклонениям. Причина та же, что и раньше. Данные, полученные агентом, остаются неизменными в течение ограниченного времени. Хитрость заключается в том, чтобы установить этот период довольно малым, чтобы вы могли быстро улучшить модель, но и достаточно длинным, чтобы данные оставались неизменными [5]. Вы можете заметить, что эта идея уже была представлена в *разд. "Двойное Q-обучение" главы 3*; две оценки более стабильны, чем одна.

## Архитектура нейронной сети

За счет использования случайной выборки воспроизведения опыта, а также целевой Q-сети DQN достаточно стабильна для того, чтобы хорошо справляться со сложными задачами. Основным фактором является выбор нейронной сети, которая моделирует и преобразует наблюдения в действия. В оригинальной статье DQN игры Atari воспроизводились с использованием видеокадров, поэтому использование CNN — очевидный выбор. Но это не мешает вам предпочесть другие архитектуры нейронных сетей. Выберите архитектуру, которая хорошо работает в вашей области.

Еще одна интересная особенность DQN — выбор для прогнозирования функции ценности состояния. Напомним, стандартное Q-обучение (или SARSA) использует функцию ценности действия; Q-значения содержат как состояние, так и действия в качестве параметров. Результатом является оценка ожидаемого значения для состояния и отдельного действия. Вы можете тренировать одну-единственную нейронную сеть для каждого действия. В таком случае у нейронной сети может быть несколько *"голов"*[1], которые предсказывают значения всех действий одновременно. При обучении вы можете установить желаемое действие на 1 и нежелательные действия на 0, что является быстрой кодировкой действий.

---

[1] Представьте себе многоголового советника, каждая голова которого дает вам рекомендацию, как поступить в конкретной ситуации. У каждой головы свой взгляд на вещи в зависимости от предпочтений, только ей ведомых. Вы, выслушав советы каждой головы, поступаете так, как считаете нужным. Здесь всё точно также. — *Прим. ред.*

## Внедрение глубокой Q-сети

Уравнение, представляющее правило обновления глубокой Q-сети, похоже на пример из *разд. "Q-обучение" главы 3*. Основное различие состоит в том, что Q-значение аппроксимируется функцией, и эта функция имеет набор параметров. Например, чтобы выбрать оптимальное действие, выберите то, которое имеет наибольшее ожидаемое значение, как в уравнении 4.1.

**Уравнение 4.1. Выбор действия с DQN**

$$a \leftarrow \underset{a_s \in \mathcal{A}(s)}{\arg\max} Q(s, a_s; \theta).$$

В уравнении 4.1 $Q$ представляет функцию, используемую для прогнозирования значений действия из нейронной сети, а $\theta$ — параметры функции. Все остальные символы такие же, как в уравнении 3.5.

Для того чтобы обучить нейронную сеть, вам необходимо предоставить функцию потерь. Помните, что цель состоит в том, чтобы предсказать функцию ценности действия. Таким образом, естественный выбор функции потерь — это возведенная в квадрат разность между фактической функцией ценности действия и прогнозом, как показано в уравнении 4.2. Обычно эта разность реализуется как *среднеквадратичная ошибка* (mean squared error, MSE), но возможна любая функция потерь.

**Уравнение 4.2. Функция потерь DQN**

$$L(\theta) = \mathbb{E}_{\pi}\left[\left(y - Q(s, a; \theta)\right)^2\right].$$

Оптимизаторы нейронных сетей для обновления оценок их параметров применяют *градиентный спуск*. Они используют знание градиента функции потерь, чтобы сдвинуть параметры в направлении, которое минимизирует функцию потерь. Базовая структура глубокого обучения выполняет расчет градиента за вас. Но вы можете самостоятельно вывести, дифференцируя уравнение 4.2 по $\theta$.

Реализация DQN во многом зависит от Q-обучения по алгоритму 3.1. Различия заключаются в том, что действие доставляется непосредственно из нейронной сети, опыт нуждается в буферизации, и иногда вам нужно передавать или обучать параметры целевой нейронной сети. Дьявол кроется в деталях. Именно то, как вы реализуете воспроизведение опыта, какую архитектуру и гиперпараметры нейронной сети выбираете, а также в какой момент проводите обучение, может иметь значение в выборе между сверхчеловеческой производительностью и алгоритмическим взрывом.

В оставшейся части этой главы я решил использовать среду Coach[2] от Intel Nervana Systems. Мне она нравится по трем причинам: я нахожу абстракции интуитивно понятными, она реализует множество алгоритмов и поддерживает множество сред.

[2] См. **https://oreil.ly/HLkqv**.

## Пример: глубокая Q-сеть в среде CartPole

Для того чтобы получить некоторый опыт работы с DQN, рекомендую вам использовать простую "игрушечную" среду, например среду CartPole из OpenAI Gym[3]. Среда имитирует балансировку шеста на тележке.

Агент может подтолкнуть тележку влево или вправо; это действия. Он представляет состояние с положением по оси $x$, скоростью тележки, скоростью наконечника шеста и углом шеста (0°— прямо вверх). Агент получает вознаграждение в размере 1 за каждый сделанный шаг. Эпизод заканчивается, когда угол наклона шеста превышает ±12°, положение тележки больше ±2,4 (край дисплея) или длина эпизода превышает 200 шагов. Для того чтобы решить эту проблему, вам понадобится средняя награда, не менее 195 за 100 последовательных испытаний.

У Coach есть концепция предустановок[4], которые представляют собой настройки алгоритмов. В предустановке CartPole_DQN[5] есть решение для реализации среды CartPole с DQN. Я взял эти предустановки и внес несколько изменений, чтобы соблюсти следующие условия:

- сеть копирует целевые веса в онлайн-веса каждые 100 шагов среды;
- коэффициент дисконтирования установлен на 0,99;
- максимальный размер памяти — 40 000 опытов;
- используется постоянный ε-жадный параметр, равный 0,05 (для согласованности графиков);
- нейронная сеть использует потери на основе среднеквадратичной ошибки, а не потери Хьюбера по умолчанию;
- отсутствие "разогрева" среды для предварительного заполнения памяти (для получения результата с самого начала).

Из всех этих настроек наибольшее значение имеет использование потерь на основе среднеквадратичной ошибки (см. уравнение 4.2). Потери Хьюбера ограничивают потерю как линейную ошибку, превышающую пороговое значение, и равную нулю — в противном случае. В более сложных средах абсолютные потери, такие как потеря Хьюбера, помогают снизить выбросы. При использовании среднеквадратичной ошибки выбросы возводят в квадрат, и огромная ошибка подавляет все остальные данные. Но в этом случае среда проста и нет шума, поэтому среднеквадратичная ошибка из-за возведения в квадрат помогает нейронной сети учиться быстрее, потому что ошибки действительно велики лишь с самого начала. По этой причине потери Хьюбера, как правило, являются функцией потерь по умолчанию, используемой в RL.

---

[3] См. **https://oreil.ly/YGpj0**.

[4] См. **https://oreil.ly/wDeC3**.

[5] См. **https://oreil.ly/KTjRB**.

В общем, я рекомендую вам использовать абсолютные потери в шумных средах и квадратичные потери для более простых и бесшумных сред.

Для того чтобы обеспечить основу для алгоритма DQN, я реализовал в Coach два новых агента. Случайный агент выбирает случайное действие на каждом шаге. Агент Q-обучения реализует Q-обучение, как описано в *разд. "Q-обучение" главы 3*. Напомним, что табличное Q-обучение не может обрабатывать непрерывные состояния. Я смог решить проблему среды CartPole с помощью Q-обучения, умножив все состояния на 10, округлив до ближайшего целого числа и приведя к целочисленному типу данных integer. Я также использую только угол и угловую скорость падения (третий и четвертый элементы в массиве наблюдения соответственно). Это приводит примерно к 150 состояниям в справочной таблице значений $Q$.

На рис. 4.1 показаны награды за эпизоды для трех агентов. И DQN, и Q-обучение способны найти оптимальную политику. Случайная политика способна обеспечить среднее вознаграждение примерно в 25 шагов. Это связано с физикой окружающей среды. Столько времени требуется, чтобы шест упал. Агент Q-обучения изначально работает лучше, чем DQN. Это связано с тем, что DQN требуется определенный объем данных, прежде чем она сможет обучить разумную модель Q-значений. Точный объем требуемых данных зависит от сложности глубокой нейронной сети и размера пространства состояний. Агент Q-обучения иногда плохо работает из-за ε-жадного случайного действия. В отличие от этого DQN может делать обобщения до состояний, которых раньше не наблюдала, поэтому производительность более стабильна.

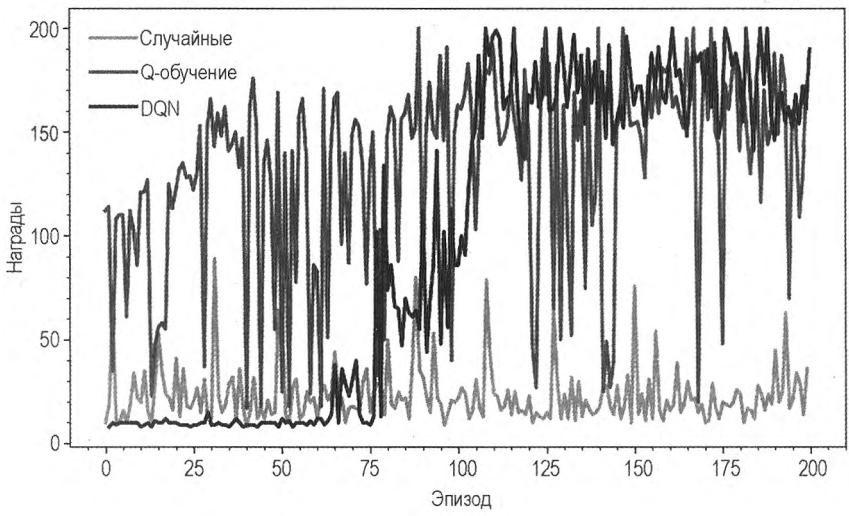



**Рис. 4.1.** Сюжет эпизода вознаграждений для случайных агентов, агентов Q-обучения и DQN. Результаты зашумлены, потому что не усреднены

На рис. 4.2–4.4 показан пример эпизода из случайных агентов, агентов Q-обучения и DQN-агентов соответственно. Примеры Q-обучения и DQN рассчитаны на 200 шагов, что является максимумом для эпизода.

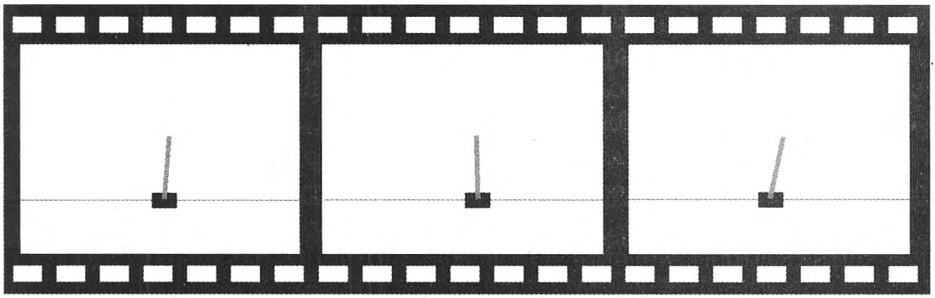


**Рис. 4.2.** Пример случайного агента в среде CartPole

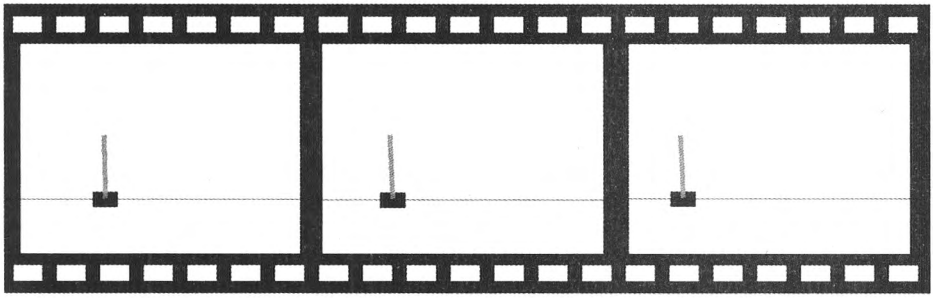


**Рис. 4.3.** Пример агента Q-обучения в среде CartPole

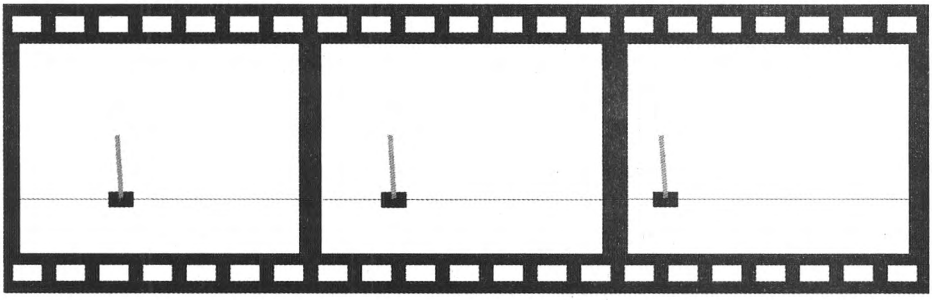


**Рис. 4.4.** Пример DQN-агента в среде CartPole

## Зачем обучаться онлайн?

Вы можете спросить: зачем вообще нужно тренироваться онлайн? Вы можете сгенерировать и сохранить 200 эпизодов данных из среды и обучить модель в автономном режиме под контролем. Для простых сред этот подход может сработать, хотя и неэффективно. Для того чтобы сгенерировать данные, агенту по-прежнему необходимо следовать политике. Вы можете выбрать случайную политику, но это

вряд ли приведет к созданию длинных эпизодов. Рассмотрим среду CartPole. Как вы думаете, сколько попыток потребуется случайной политике для достижения 200 временных шагов? Много. В большинстве приложений агенту необходимо обучаться онлайн, чтобы он мог наблюдать за более поздними состояниями.

Вторая причина заключается в том, что агент должен производить *контрфактические* действия (действия, которые могут быть плохими), чтобы подтвердить, что у него действительно лучшая политика. Это суть исследования. Действия порождают новые невидимые траектории. Одна из них будет оптимальной. Если у вас нет последовательной среды или вам не нужно заниматься поиском оптимальных политик, используйте стандартное машинное обучение. Подробнее об этом я расскажу в *разд. "Обучение на основе автономных данных" далее в этой главе.*

### Что лучше? Глубока Q-сеть против Q-обучения

Результаты показывают, что стандартное Q-обучение ничем не хуже DQN. И я не применял никаких улучшений к Q-обучению, описанных в *разд. "Расширения для Q-обучения" главы 3*. Зачем использовать DQN, учитывая сложность добавления нейронной сети?

Для того чтобы Q-обучение заработало, мне пришлось резко сократить объем информации. Я удалил половину функций и квантовал оставшиеся, чтобы оставить около 150 дискретных значений. Для того чтобы найти эти настройки, мне пришлось прибегнуть к конструированию признаков, которое включало в себя множество других комбинаций функций и уровней квантования — усилия, которые потребовали бы целой книги. Это оправданно, если вы хотите запускать подобные модели в производственной среде, потому что простота облегчит объяснение решений, а модели будут более надежными, более эффективными, более стабильными и т. д.

В DQN я ничего не менял. Нейронная сеть смогла взять необработанную информацию, выяснить, какие функции были важны, и сообщить мне, какое действие было оптимальным. Модель может включать непрерывные значения, что делает результат более детальным по сравнению с табличными методами. Если ваша область применения сложная, тогда разработка функций становится затруднительной и нейронные сети могут быть эффективным решением. Но имейте в виду, что применяются все обычные предостережения для нейронных сетей: им требуется много данных, они склонны к переобучению, чувствительны к начальным условиям и гиперпараметрам, не являются устойчивыми или надежными и открыты для враждебных атак (*см. разд. "Безопасное RL" главы 10*).

## Практический пример: сокращение энергопотребления в зданиях

Около 40% энергии Европейского союза расходуется на электроэнергию и обогрев зданий, что составляет около 25% выбросов парниковых газов [6]. Существует ряд технологий, которые могут помочь снизить потребности зданий в энергии, но многие из них являются агрессивными и дорогостоящими. Однако точная настройка

управления отоплением, вентиляцией и кондиционированием воздуха (heating, ventilation, and air conditioning, HVAC) применима к зданиям любого возраста. Марантос и соавт. предлагают использовать RL внутри интеллектуального термостата для повышения комфорта и эффективности систем отопления зданий [7].

Они начали с определения марковского процесса принятия решений. Состояние представлено комбинацией наружных погодных датчиков — температуры и количества солнечного света, внутренних датчиков — влажности и температуры, датчиков энергии для количественной оценки использования и индикации температурного комфорта.

Агент может выбрать одно из трех действий: поддерживать температуру, повышать температуру на один градус или понижать температуру на один градус.

Вознаграждение оценивается количественно путем сочетания теплового комфорта и использования энергии, которое масштабируется с помощью скользящего среднего и стандартного отклонения, чтобы исключить необходимость использования произвольных параметров масштабирования. Поскольку это в основном постоянная проблема, Марантос и соавт. вводят состояние терминала, чтобы предотвратить выход агента за пределы: если агент пытается выйти за пределы предопределенного рабочего окна, он наказывается.

Марантос и соавт. решили использовать алгоритм нейронной аппроксимации Q-итерации (neural fitted Q-iteration, NFQ), который был предшественником глубокой Q-сети. В основе последней те же идеи, что и в NFQ, за исключением того, что он также включает воспроизведение опыта и целевую сеть. Они используют многослойный персептрон с прямой связью в качестве функции представления состояния для прогнозирования бинарных действий.

Авторы оценивают свою реализацию с помощью моделирования, которое используется во множестве аналогичных работ. Моделирование соответствует реальному зданию, расположенному на острове Крит в Греции, с использованием общедоступной информации о погоде. По сравнению со стандартным пороговым термостатом агент может снизить среднее потребление энергии до 59% в зависимости от выбранного уровня комфорта.

Одна интересная идея, которую вы можете не заметить, — это то, что Марантос и соавт. реализовали данный алгоритм на базе Raspberry Pi Zero[6] и продемонстрировали, что даже с нейронной сетью у них более чем достаточно вычислительных ресурсов для обучения модели.

Если бы я помогал улучшить этот проект, первым делом я бы подумал о переходе на DQN или одну из ее производных для повышения алгоритмической производительности. NFQ явно слабее DQN из-за воспроизведения опыта и целевой сети. Это изменение приведет к повышению эффективности выборки и позволит использовать стандартные отраслевые методы. Мне нравится использование встроенного устройства для выполнения вычислений, и я бы поддержал это, рассмотрев воз-

[6] Одноплатный компьютер размером с банковскую карту, изначально разработанный как бюджетная система для обучения информатике. — *Прим. ред.*

можность размещения нескольких агентов по всему зданию. В этой многоагентной настройке агенты могут сотрудничать, чтобы оптимизировать температурный комфорт для различных областей, например чтобы было прохладнее в спальнях или полное отключение на ночь в других помещениях.

В принципе, это относится не только к отоплению. Агент может легко оптимизировать освещение, потребление горячей воды и электричества. Все из перечисленного документально подтверждено тематическими исследованиями. Вы можете найти дополнительную информацию на специальном веб-сайте[7].

# Радужная DQN

Так как DQN является производной от Q-обучения, вы можете улучшить его, используя расширения, представленные в *разд. "Расширения для Q-обучения" главы 3*. В 2017 г. исследователи создали радужную DQN, используя шесть расширений, каждое из которых решает фундаментальные проблемы с Q-обучением и DQN. Исследователи также провели эксперименты, чтобы выяснить, какие из них улучшили производительность больше всего [8].

Вы уже видели первые два улучшения: двойное Q-обучение (*см. разд. "Двойное Q-обучение" главы 3*) для уменьшения систематической ошибки максимизации и *n*-ступенчатой отдачи (*см. разд. "n-Шаговые алгоритмы" главы 3*), чтобы накапливать вознаграждение за несколько будущих шагов.

## Распределительное RL

Наиболее фундаментальным отклонением от стандартной DQN является повторное введение вероятности в агентов. Все агенты Q-обучения выводятся из уравнения Беллмана (см. уравнение 2.10), которое определяет оптимальную траекторию как действие, которое максимизирует ожидаемое вознаграждение. Ключевое слово здесь *"ожидаемое"*. Q-обучение реализует это как экспоненциально убывающее среднее. Всякий раз, когда вы используете сводную статистику, вы подразумеваете предположения о базовом распределении. Обычно, если вы выбираете использование среднего значения в качестве сводной статистики, вы делаете неявное предположение, что данные распределены по нормальному закону. Правильно ли это?

Например, представьте, что вы балансируете у обрыва в среде Cliffworld. Если вы упадете с обрыва, получите большой отрицательный штраф. Но есть также гораздо большее (но почти нулевое) вознаграждение за прогулку по краю обрыва. В *разд. "Q-обучение против SARSA" главы 3* показано, что алгоритм SARSA следовал среднему "безопасному" пути вдали от обрыва. Алгоритм Q-обучения пошел по рискованному пути рядом с обрывом, потому что меньшее количество шагов к цели привело к немного большей ожидаемой награде. Все состояния в среде Cliffworld приводят к двум результатам. Оба исхода имеют собственное распределение, пред-

[7] См. **https://rl-book.com/applications/?utm_source=oreilly&utm_medium=book&utm_campaign=rl**.

ставляющее небольшой шанс того, что некоторое исследование может привести агента к альтернативному исходу. Ожидаемая доходность обычно не распределяется. Как правило, вознаграждения являются мультимодальными.

Теперь представьте себе среду, похожую на Cliffworld, где разница между успехом и неудачей не так велика. Представьте, что награда за успех равна +1, а штраф за неудачу равен –1. Тогда ожидаемая доходность по крайней мере для некоторых состояний будет близка к нулю. В этих состояниях среднее ожидаемое значение приводит к действиям, которые не имеют значения. Это может привести к тому, что агенты будут случайным образом блуждать по бессмысленным состояниям (сравните это с алгоритмами градиентного спуска, которые выходят на плато, когда функции потерь становятся слишком монотонными), как показано на рис. 4.5. В целом распределение вознаграждений намного сложнее.

**Рис. 4.5.** Изображение мультимодальных наград для состояния: *а* — простая сетка с обрывом с одной стороны и полезной целью — с другой; *б* — распределение вознаграждения имеет два противоположных вознаграждения, а среднее значение, которое совпадает с функцией ценности для этого состояния, равно нулю

На практике это приводит к проблемам сходимости. В худшем случае агент никогда не сойдется. Когда это происходит, исследователи часто описывают агента как "болтающегося" (испытывающего колебания).

Здесь агент сходится, но никогда не стабилизируется. Он может колебаться, как на рис. 4.6, или быть более сложным. Учтите, что это не единственная причина колебания.

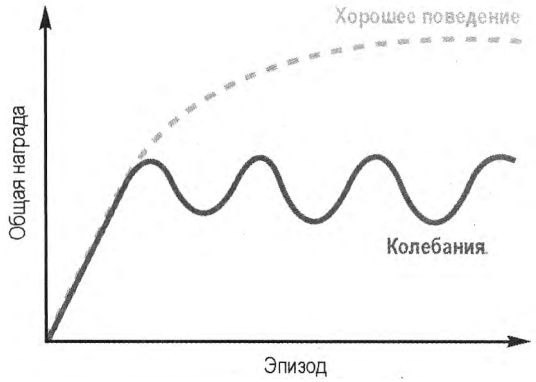



**Рис. 4.6.** Изображение болтанки с использованием сюжета наград во время тренировки

Изучение распределения вознаграждений может привести к более стабильному обучению [9]. Беллмар и соавт. переформулировали уравнение Беллмана, чтобы учесть случайные вознаграждения, которые вы можете увидеть в уравнении 4.3, где $\gamma$ — коэффициент дисконтирования, а $R$ — стохастическое вознаграждение, которое зависит от состояния и действия.

**Уравнение 4.3. Распределительное уравнение Беллмана**

$$Z(s,a) \sim R(s,a) + \gamma Z(S', A').$$

Здесь я временно нарушаю правила этой книги и использую заглавные буквы для обозначения стохастических переменных. $Z$ представляет собой распределение вознаграждений. Математическое ожидание $Z$ — это значение $Q$. Как и значение $Q$, $Z$ следует обновлять рекурсивно в соответствии с полученным вознаграждением. Ключевым отличием является акцент на том, что ожидаемая доходность $Z$ является случайной величиной, распределенной по всем будущим состояниям $S$ и будущим действиям $A$. Другими словами, политика $Z^{\pi}$ представляет собой сопоставление пар "состояние — действие" с распределением ожидаемой доходности. Это представление сохраняет мультимодальность, и Беллмар и др. предполагают, что это приводит к более стабильному обучению. Рекомендую вам посмотреть видео[8] с примером распределения наград, полученных во время игры Space Invaders.

Первое воплощение этой идеи — алгоритм под названием C51. Он задает два ключевых вопроса. Как вы представляете раздачу? Как создать правило обновления для уравнения 4.3? Выбор распределения или функции, применяемой для моделирования доходности, важен. C51 использует дискретное параметрическое распределение, другими словами, гистограмму доходности. "51" в C51 относится к количеству интервалов гистограммы, выбранных в реализации исследователей, и предполагает, что производительность пропорциональна количеству интервалов, хотя и с убывающей отдачей.

Для получения оценки распределения Беллмар и соавт. построили сеть с 51 выходным "бункером". Когда среда производит вознаграждение, которое попадает в одну из корзин, агент может обучить нейронную сеть предсказывать эту корзину. Теперь это проблема контролируемой классификации, и агент может быть обучен с использованием кросс-энтропийной потери. Буква "C" в C51 указывает на то, что агент предсказывает классы или категории.

## Воспроизведение приоритетного опыта

Следующий набор улучшений касается нейронной сети. В DQN переходы равномерно отбираются из буфера воспроизведения (*см. разд. "Глубокое Q-обучение" ранее в этой главе*). Пакет может содержать переходы, которые агент видел много раз. Лучше включать переходы из областей, в которых есть чему поучиться. *Прио-*

[8] См. **https://oreil.ly/vTVHj**.

*ритетное воспроизведение опыта* формирует выборку буфера воспроизведения с вероятностью, пропорциональной абсолютной ошибке обновления с учетом временны́х различий (см. уравнение 3.6) [10]. Шаул и соавт. изменили эту идею, чтобы ввести $n$-шаговую абсолютную ошибку вместо одношаговой.

## Зашумленные сети

В сложных средах ε-жадному алгоритму может быть затруднительно получить успешный результат. Одна из идей, называемая *зашумленными сетями*, состоит в том, чтобы расширить ε-жадный поиск путем добавления шума в нейронную сеть [11]. Напомним, что нейронная сеть — это модель предполагаемых действий для данного состояния, а именно — политика. Применение шума непосредственно к нейронной сети обеспечивает случайный поиск в контексте текущей политики. Модель научится игнорировать шум (потому что он случайный), когда в ней будет достаточно переходов для конкретного состояния. Однако новые состояния попрежнему будут шумными, что позволит проводить исследование с учетом контекста и автоматический отжиг (*см. разд. "Улучшение ε-жадного алгоритма" главы 2*).

## Дуэльные сети

В алгоритмах Q-обучения оценки ожидаемого значения (Q-значения) очень похожи. Часто нет необходимости даже рассчитывать ожидаемое значение, потому что оба варианта могут привести к одному и тому же результату. Например, в среде CartPole первое действие не будет иметь большого значения, если оно научится спасать шест до того, как тот упадет. Следовательно, имеет смысл изучить функцию ценности состояния, в отличие от функции ценности действия, которую я использовал в *разд. "Прогнозирование вознаграждений с помощью функции ценности действия" главы 2*. Тогда каждое наблюдение может обновить всё состояние, что должно привести к более быстрому обучению. Однако агенту еще нужно выбрать действие.

Представьте на секунду, что вы можете точно предсказать функцию ценности состояния. Напомним, что это ожидание (среднее значение) по всем действиям для этого состояния. Таким образом, я могу представить отдельные действия относительно этого среднего значения. Исследователи называют это *функцией преимущества*, и она определяется в уравнении 4.4. Для того чтобы восстановить функцию ценности действия, требуемую Q-обучением, вам необходимо оценить функцию преимущества и функцию ценности состояния.

**Уравнение 4.4. Функция преимущества**

$$A^{\pi}(s,a) \doteq Q^{\pi}(s,a) - V^{\pi}(s).$$

*Дуэльные сети* достигают этого путем создания единой нейронной сети с двумя "головами", которые являются выходными данными для прогнозирования. Один руководитель отвечает за оценку функции ценности состояния, а другой — за

функцию преимущества. Нейронная сеть рекомбинирует эти две "головы", чтобы сгенерировать требуемую функцию ценности действия [12]. Новым является специальный агрегатор на выходе двух "голов", который позволяет обучать всю нейронную сеть как единое целое. Преимущество состоит в том, что функцию ценности состояния быстрее и легче изучить, а это приводит к более быстрой сходимости. Кроме того, эффект вычитания базовой линии в оценке преимущества улучшает стабильность. Это связано с тем, что предвзятость максимизации Q-обучения может вызвать большие изменения в индивидуальных оценках функции ценности действия.

# Пример: радужная глубокая Q-сеть в Atari Games

В научных работах по DQN популяризируется использование вознаграждений в играх Atari в качестве меры производительности агентов. В этом разделе я воссоздал некоторые из этих результатов, обучая агента играть в две игры Atari: Pong и Ms Pac-Man. Я снова использую фреймворк Coach и его предустановки для Atari.

Среда Atari является частью OpenAI Gym[9] и предоставляет необработанные кадры Atari 2600. Каждый кадр представляет собой изображение размером 210 × 160 пикселов с 128-битной цветовой RGB-палитрой. Для того чтобы уменьшить вычислительную сложность, вызванную большим количеством входных данных, метод предварительной обработки преобразует массив 210 × 160 × 3 путем прореживания, обрезки и преобразования его в оттенки серого. Это уменьшает количество входных размеров до 84 × 84. Агент фиксирует награды на уровне ±1, чтобы объединить доход игр Atari.

В играх Atari много движущихся объектов, например пуль или призраков. Агент должен знать траекторию этих сущностей, чтобы реагировать должным образом, уклоняться от пули или убегать от призрака. Более продвинутые агенты включают память, зависящую от времени, часто с использованием повторяющихся нейронных сетей. Глубокая Q-сеть и ее производные этого не делают, поэтому агенты обычно складывают несколько кадров в одно наблюдение, выравнивая изображение и объединяя их в один большой массив. Эта реализация объединяет четыре кадра, и агент повторно использует одно и то же действие для следующих четырех кадров.

## Результаты

Я решил провести два эксперимента из-за количества времени, которое требовалось на обучение агента. Я использовал настройки гиперпараметров по умолчанию в предустановках библиотеки Coach. К ним относятся ε-жадное исследование отжига для 250 000 кадров и низкая скорость обучения искусственной нейронной сети. Я обучил агентов на платформе Google Cloud Platform, используя виртуальные ма-

[9] См. **https://oreil.ly/DAe39**.

шины n1-standard-4, подключенные к одному графическому процессору Nvidia Tesla P4 в Нидерландах.

Время обучения и приблизительная стоимость описаны в табл. 4.1. На обучение агента DQN на 1 млн кадров в среднем уходит 4,8 часа и 5,80 доллара. Агенту радужной DQN требуется 7,6 часа и 8,50 доллара для достижения аналогичного результата. Обратите внимание, что эти значения являются индикаторами. На них сильно влияют гиперпараметры, которые у этих двух агентов разные. Исследования показывают, что радужная DQN намного превосходит DQN по производительности во всех играх Atari.

***Таблица 4.1.*** *Время и стоимость обучения Atari-играм Pong и Ms Pac-Man до сравнимых уровней с использованием агентов DQN и радужной DQN во фреймворке Coach*

| Агент | Игра | Количество кадров, млн штук | Время, ч | Стоимость, долл. |
|---|---|---|---|---|
| DQN | Pong | 2 | 10 | 12 |
| DQN | Ms Pac-Man | 3 | 14 | 17 |
| Радужная DQN | Pong | 2,5 | 19 | 22 |
| Радужная DQN | Ms Pac-Man | 3 | 23 | 26 |

Эти результаты подчеркивают экспоненциальное увеличение сложности, что приводит к увеличению времени обучения. Несмотря на доказательство того, что радужная DQN приводит к улучшению политики и занимает меньше времени, важны контекст и обстоятельства. Вы должны разработать решение, соответствующее вашим конкретным требованиям. Например, улучшение производительности, полученное с помощью радужной DQN, может не оправдать почти двукратное увеличение вычислительных затрат. Однако обучение базовой ANN забирает основную часть денежных вложений. Если вы сможете упростить, настроить или ускорить архитектуру ANN, цены резко упадут.

На рис. 4.7 показаны награды, полученные при обучении игре Pong. Агенту радужной DQN требуется дополнительно 0,5 млн кадров, чтобы получить результат, аналогичный DQN. Это может быть связано с разными настройками гиперпараметров. Скорость обучения $\alpha$ установлена на $10^{-4}$ для DQN. Для радужной DQN та же скорость составляет чуть меньше двух третей от этого значения и равна $6{,}25 \cdot 10^{-5}$. DQN использует $\varepsilon$-жадный алгоритм, а радужная DQN — исследование параметрического шума. Я уверен, что с подобными гиперпараметрами радужная DQN будет работать не хуже, чем DQN в этом примере.

На рис. 4.8 показаны награды в игре Ms. Pac-Man. Я сгладил награду с помощью скользящего среднего 100 кадров. Пришлось сделать это, поскольку награда намного шумнее из-за случайных стартовых позиций призраков. Например, одиночная игра может иметь низкую награду, потому что в этом случае призраки начинали двигаться из опасных положений. В других играх награда может быть высокой,

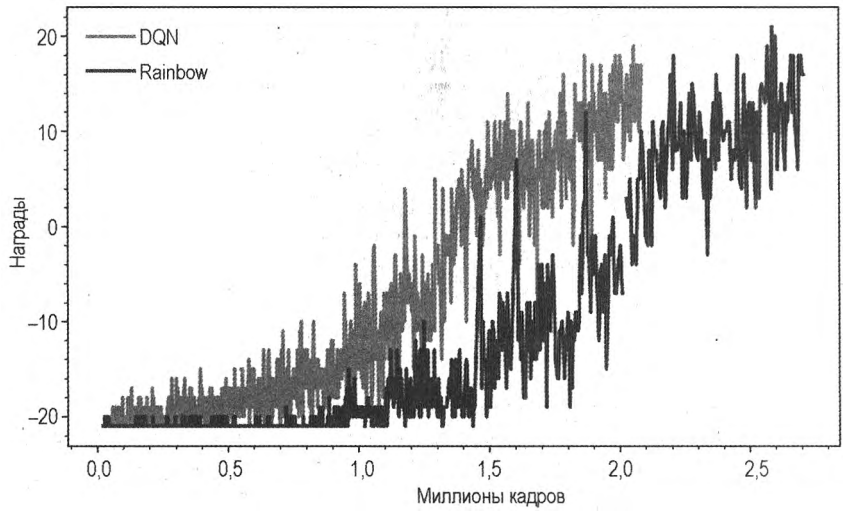



**Рис. 4.7.** График наград для агентов DQN и радужной DQN в Atari-игре Pong

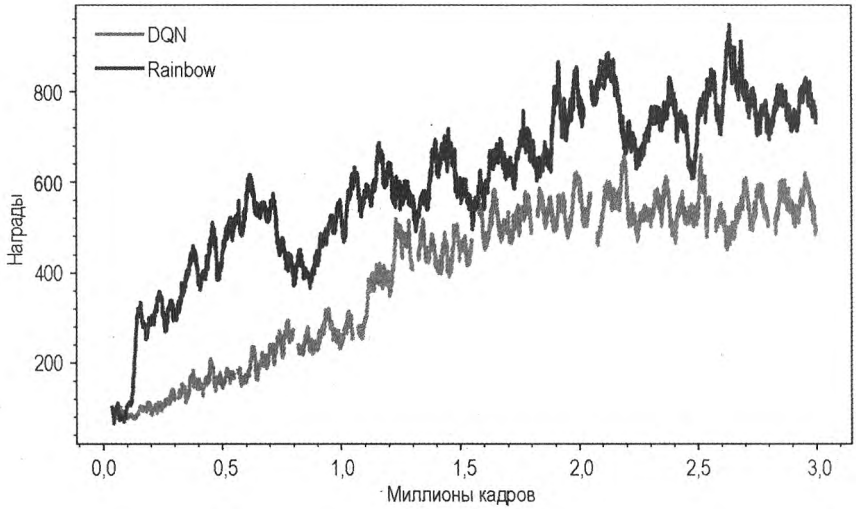



**Рис. 4.8.** График наград для агентов DQN и радужной DQN в Atari-игре Ms. Pac-Man. Награды были сглажены с использованием скользящего среднего за 100 кадров

потому что призраки начинают с безопасных позиций. Рекомендую учитывать распределение вознаграждений при работе в стохастической среде.

## Обсуждение

На рис. 4.9 и 4.10 показан полный эпизод из игры Pong для агентов DQN и радужной DQN соответственно. Поскольку это отдельные эпизоды и гиперпараметры, используемые для обучения каждого агента, и они различаются, не торопитесь делать какие-либо выводы из результата. Но вы можете проанализировать политику с позиции человека.

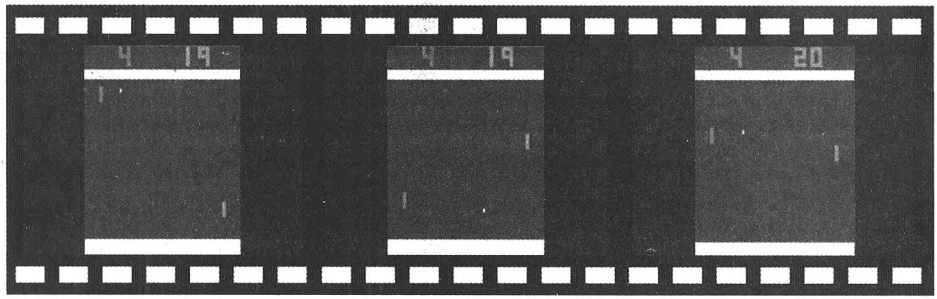


**Рис. 4.9.** Пример обученного агента DQN, играющего в Atari-игру Pong

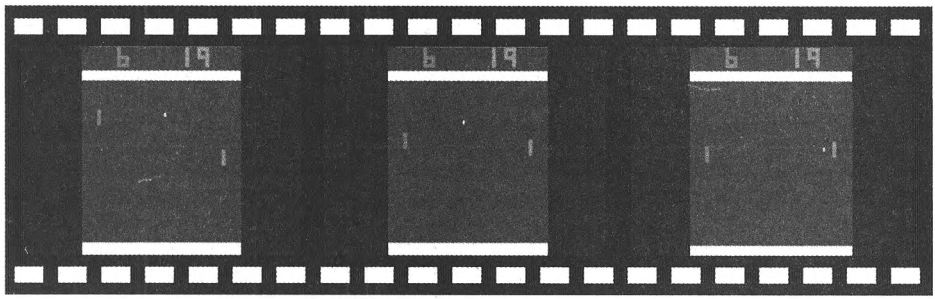


**Рис. 4.10.** Пример обученного агента радужной DQN, играющего в Atari-игру Pong

Когда я впервые показал эти видео своей жене, она сначала воскликнула: "Работает верно!" Но после того, как я объяснил, что это серьезное дело, она заметила, что движение ракетки начиналось, когда мяча не было рядом. Это происходило потому, что действия агента не влияют на результат; Q-значения почти одинаковы и приводят к случайному перемещению. Агент быстро реагирует, когда мяч приближается к ракетке, потому что он узнал, что проиграет, если не окажется рядом с мячом.

У двух агентов есть тонкие различия. DQN имеет тенденцию отбивать мяч об угол ракетки, тогда как радужная DQN стремится просто отбивать мяч. Подозреваю, что эти действия — артефакты досрочного прекращения обучения. Если бы я провел эти эксперименты для 20 млн кадров, предполагаю, что методы будут иметь тенденцию к сближению, но это дорого, поэтому я оставляю убедится в этом на откуп читателю.

На рис. 4.11 и 4.12 показаны полные эпизоды Ms. Pac-Man для агентов DQN и радужной DQN соответственно. Оба агента проделывают работу хорошо, избегая призраков, собирая гранулы и используя телепорты, которые переносят персонажа на другую сторону экрана. Но в целом радужная DQN работает лучше, чем DQN. Агент также научился искать энергетические гранулы, чтобы съесть призраков. Радужная DQN почти завершает первый уровень, и, учитывая, что на обучение требуется больше времени, я уверен, что сможет завершить. Ни один из агентов не научился есть фрукты для получения дополнительных очков или активно выслеживать призраков после того, как съел гранулы силы.

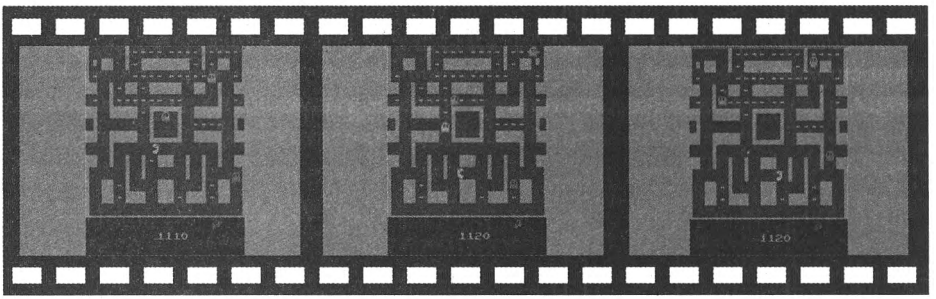



**Рис. 4.11.** Пример обученного агента DQN, играющего в Atari-игру Ms. Pac-Man

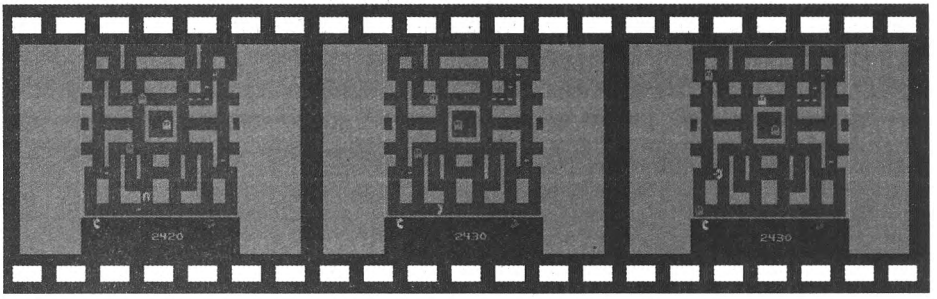



**Рис. 4.12.** Пример обученного агента радужной DQN, играющего в Atari-игру Ms. Pac-Man

# Другие улучшения глубокой Q-сети

В *разд. "Радужная DQN" ранее в данной главе* было введено шесть улучшений в DQN. Каждый алгоритм устраняет одно препятствие и обобщает три темы исследования.

1. Нейронные сети должны учиться быстрее и давать более информативные результаты.
2. Алгоритм Q-обучения не должен быть нестабильным или предвзятым.
3. RL должен эффективно искать оптимальную политику.

Вы можете видеть, что все они относятся к трем разным уровням абстракции. В радужной DQN, например, приоритетны попытки воспроизведения опыта для улучшения обучения нейронной сети, дуальное двойное Q-обучение помогает предотвратить предвзятость максимизации в Q-обучении, а распределенный RL помогает получать более реалистичные вознаграждения в RL. В этом разделе я расскажу о других недавних улучшениях, касающихся DQN.

## Улучшение исследования

Стратегия ε-жадного исследования широко распространена и предназначена для случайной выборки небольших пространств с конечным состоянием. Но методы возбуждения, при которых вы беспорядочно возмущаете агента, непрактичны

в сложных средах; они требуют экспоненциально увеличивающихся объемов данных. Один из способов описания этой проблемы — вообразить поверхность, представленную Q-значениями. Если агент Q-обучения неоднократно выбирает неоптимальную пару состояний и действий, последующие эпизоды также будут выбирать эту пару, как будто агент застревает в локальных максимумах. Одной из альтернатив является рассмотрение других методов отбора проб.

В *разд. "Улучшение ε-жадного алгоритма" главы 2* приведена выборка с верхним доверительным интервалом (upper-confidence-bound, UCB), которая смещает действия в сторону состояний с недостаточной выборкой. Это предотвращает передискретизацию исходных оптимальных траекторий. Выборка Томпсона, часто используемая в A/B-тестировании (*см. разд. "Алгоритм многорукого бандита" главы 2*), поддерживает распределения, которые представляют ожидаемые значения. Для Q-обучения это означает, что каждое Q-значение является распределением, а не ожиданием. Здесь реализована та же идея, что и в распределенном RL (*см. разд. "Распределенное RL" ранее в этой главе*). Бутстрэпированная (самозагружаемая) DQN (bootstrapped DQN, BDQN) использует эту идею и обучает нейронную сеть с несколькими "головами", чтобы моделировать распределение Q-значений совокупности оценок ценности действия [13]. Это было заменено распределенным RL, которое эффективно предсказывает фактическое распределение, а не грубое приближение. Но Осбанд и соавт. дальновидно продолжили повторно использовать случайно выбранную политику, чтобы стимулировать исследования в будущем. Это привело к идее "глубокого исследования". В сложных условиях ε-жадный алгоритм движется, как муха, часто возвращаясь в состояния, которые наблюдал ранее. Имеет смысл углубиться в поиск новых, ненаблюдаемых состояний. Здесь тоже можно увидеть сходство с зашумленными сетями.

С байесовской точки зрения, моделирование Q-значений как распределений, возможно, является правильным способом, но часто требует больших вычислительных затрат. Если вы рассматриваете каждую "голову" нейронной сети как независимую оценку одной и той же политики, вы можете объединить их в ансамбль и получить аналогичную производительность. Напомним, ансамбль — это комбинация алгоритмов, которая генерирует несколько гипотез. Комбинация, как правило, более надежна и менее подвержена переобучению. Следование независимым политикам более эффективным способом, например с помощью выборки с верхним доверительным интервалом, является еще одним способом достижения "глубокого исследования" [14, 15]. Еще одно важное преимущество: вы можете следовать политике, предложенной одной "головой", чтобы получить выгоду от поиска глубокой неизведанной области в пространстве состояний, а затем поделиться этим знанием с другими "головами" [16]. Более поздние исследования показали, что хотя совместное использование ускоряет обучение, но конечная производительность повышается при большем разнообразии "голов" [17].

## Повышение вознаграждения

В предыдущем разделе использовались отдельные оценки политики в качестве прокси для оценки распределения вознаграждения. C51 (*см. разд. "Распреде-*

*лительное RL" ранее в этой главе*) предсказывает гистограмму распределения. Ни один из них не эффективен так, как прямая аппроксимация распределения, например выбор границ гистограммы и количества интервалов в C51.

Более элегантным решением является оценка квантилей целевого распределения. После выбора количества квантилей, которое определяет разрешение оценки и количество "голов" в нейронной сети, функция потерь минимизирует ошибку между прогнозируемыми и фактическими квантилями. Основное различие между самонастраиваемыми DQN, C51 и ансамблем DQN — это функция потерь. В распределительном RL с квантильной регрессией Дабни и соавт. заменяют стандартную функцию потерь, используемую в RL (потери Хьюбера), квантильным эквивалентом. Это приводит к более высокой производительности во всех стандартных играх Atari, но, что важно, дает вам оценку распределения вознаграждений с высоким разрешением [18].

Недавние исследования показали, что распределение вознаграждений за обучение полезно, но не по той причине, которую вы ожидаете. Распределенное обучение должно предоставлять больше информации, что должно привести к более совершенным политикам. Но исследователи обнаружили, что оно оказывает регуляризирующее действие на нейронную сеть. Другими словами, методы распределения могут помочь стабилизировать обучение не потому, что распределенное обучение более эффективно в отношении данных, а потому, что оно ограничивает громоздкую нейронную сеть [19].

## Обучение на основе автономных данных

Многим промышленным приложениям RL необходимо учиться на уже собранных данных. *Пакетное RL* решает задачу изучения политики из фиксированного набора данных без дальнейшего взаимодействия со средой. Это обычное дело в ситуациях, когда взаимодействие является дорогостоящим, рискованным или требует много времени. Использование библиотеки данных также может ускорить разработку. Эта проблема тесно связана с обучением на примере, которое также называется имитационным обучением и обсуждается в *разд. "Имитационное RL" главы 8*.

Большинство алгоритмов RL генерируют наблюдения посредством взаимодействия с окружающей средой. Затем наблюдения сохраняются в буферах для последующего повторного использования. Могут ли эти алгоритмы работать без взаимодействия и учиться полностью в автономном режиме?

В одном эксперименте исследователи обучили двух агентов: один учился на взаимодействии, а другой использовал только буфер первого. Удивительно, но агент, обученный с помощью буфера, работал значительно хуже при обучении с тем же алгоритмом на одном и том же наборе данных.

Причина этого, исходящая из теории статистического обучения, заключается в том, что тренировочные данные не соответствуют данным тестирования. Данные, генерируемые агентом, зависят от его политики в конкретный момент. Собранные данные не обязательно репрезентативны для всего пространства "состояние — действие". Первый агент способен хорошо работать, потому что способен самокорректи-

роваться посредством взаимодействия. Агент, обученный буфером, не может этого делать и требует действительно репрезентативных данных.

Приоритетное воспроизведение опыта, двойное Q-обучение и дуэльные Q-сети — все это помогает смягчить указанную проблему, в первую очередь делая данные более репрезентативными, но они не решают основную проблему. *Глубокое Q-обучение с пакетными ограничениями* (batch-constrained deep Q-learning, BCQ) обеспечивает более простое решение. Реализация довольно сложна, но ключевым дополнением является другой способ предоставления опыта. Вместо того чтобы передавать необработанные наблюдения агенту, обученному с помощью буфера, исследователи обучают другую нейронную сеть генерировать предполагаемые действия с использованием *условного вариационного автоэнкодера*. Это тип автоэнкодера, который позволяет генерировать наблюдения из определенных классов. Это приводит к ограничению политики только за счет генерирования действий, которые приводят к состояниям в буфере. Он также включает в себя возможность настройки модели для генерации случайных действий путем добавления шума к действиям, если это необходимо. Это помогает сделать результирующую политику более устойчивой за счет заполнения пробелов, оставшихся после выборки.

Последнее изменение состоит в том, что он обучает две сети, как в двойном Q-обучении, для получения двух независимых оценок. Затем он обрезает эти оценки для генерации обновлений Q-значения с низкой дисперсией, взвешенных с помощью гиперпараметра.

На рис. 4.13 сравниваются результаты двух экспериментов в среде CartPole с использованием Coach. Сначала я генерирую буфер переходов с помощью стандартного агента DQN. Затем этот буфер передается второму агенту, который никогда не проводит исследований. Так имитируется ситуация, когда у вас есть алгоритм, работающий в производственной среде, и вы хотите использовать зарегистрированные данные. Во время тренировки второго агента вы не сможете проводить исследования, потому что не можете выполнить действие.

В первом эксперименте я обучил агента DQN работе с данными буфера (серая линия). Обратите внимание, насколько скудна награда на графике (см. рис. 4.13, *а*). Причина этого в том, что агент нестабилен, что демонстрируется расходящимися Q-значениями на графике (см. рис. 4.13, *б*). Агент будет пытаться следовать политике для состояний, которых нет в буфере. Шаг максимизации в Q-обучении приведет к чрезмерно оптимистичной оценке Q-значения. Поскольку агент не может взаимодействовать с окружающей средой, он не может получить свидетельства, позволяющие предположить обратное. Агент продолжит следить за плохим значением $Q$, а значение продолжит увеличиваться. Напомним, что в онлайн-среде агент может выбрать это состояние, чтобы проверить ожидаемый результат. Это ограничивает Q-значения.

Во втором эксперименте я обучил агента BCQ работе с данными буфера (черная линия). Обратите внимание на сходимость вознаграждения на графике (см. рис. 4.13, *а*) и стабильные значения $Q$ (см. рис. 4.13, *б*). Это в первую очередь связано с генеративной моделью, которая обеспечивает реалистичные оценки до-

ходности посредством моделирования окружающей среды (моделирование моделирования), но также, в меньшей степени, из-за выборки с низкой дисперсией независимых оценок $Q$-значения.

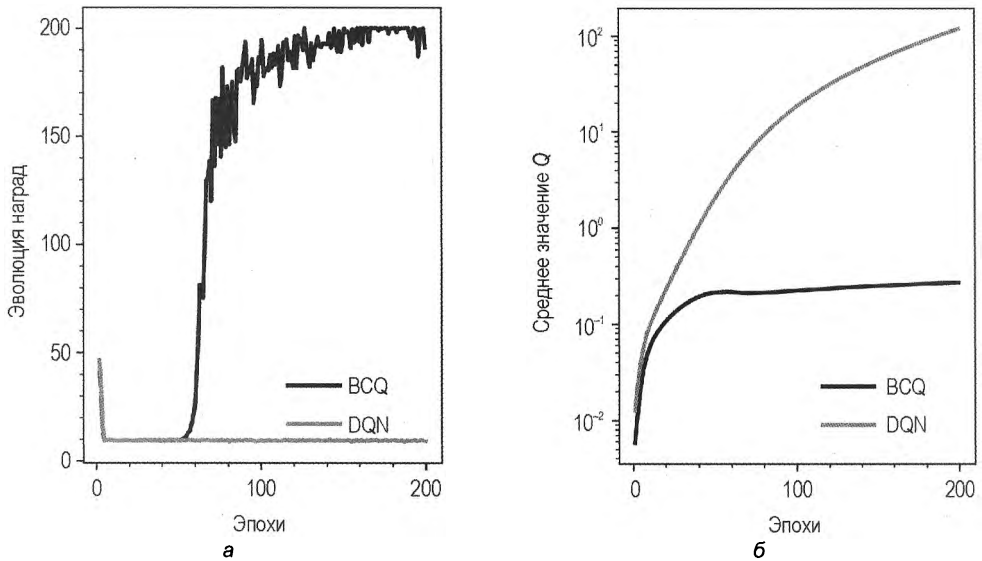



**Рис. 4.13.** Результаты эксперимента по обучению агентов BCQ и DQN без взаимодействия с окружающей средой: *а* — вознаграждение при оценке внутри окружающей среды; *б* — средние значения $Q$ (обратите внимание на логарифмическую ось $y$)

Обратите внимание, что использование автоэнкодера добавляет сложности. В более простых агентах, таких как линейное Q-обучение, вы можете использовать более простые генеративные модели, такие как *k ближайших соседей* [21].

# Резюме

В этом кратком обзоре вы можете увидеть, как использование глубоких нейронных сетей в качестве аппроксиматоров функций произвело революцию и вдохнуло новую жизнь в Q-обучение, алгоритм, опубликованный в 1991 г. Я подумал, что было бы полезно включить очень краткий обзор основных архитектур нейронных сетей, но, пожалуйста, обратитесь к книге по этой теме в *разд. "Дополнительные материалы для чтения" далее в этой главе*.

Глубокие Q-сети были прорывной работой, но нейронные сети использовались в RL в течение длительного времени [22]. Учитывая гибкость нейронных сетей, вы можете найти столько же улучшений в DQN, сколько и научных работ по глубокому обучению. Ключевой вывод заключается в том, что, хотя аппроксиматоры нелинейных функций неуправляемы и могут не сходиться, они обладают невероятной способностью аппроксимировать любую функцию. Это открывает двери для приложений, которые ранее считались слишком сложными.

Эта сила была продемонстрирована в играх Atari, но каждый день вы можете видеть все больше примеров промышленного использования. Эта глава знаменует собой конец вашего путешествия по основным табличным методам. В следующей главе я переформулирую задачу поиска оптимальной политики.

## Дополнительные материалы для чтения

- Машинное обучение.
  - Provost F., Fawcett T. Data Science for Business: What You Need to Know about Data Mining and Data-Analytic Thinking. — O'Reilly Media, 2013. — URL: **https://oreil.ly/gcJ09**.
  - Raschka S. Python Machine Learning. — Packt Publishing Ltd., 2015.
- Глубокое обучение.
  - Goodfellow I., Bengio Y., Courville A. Deep Learning. — MIT Press, 2016.
  - Chollet F. Deep Learning with Python. — Manning Publications Company, 2017.
- Генеративные модели (автокодеры и т. п.).
  - Foster D. Generative Deep Learning: Teaching Machines to Paint, Write, Compose, and Play. — O'Reilly Media, 2019.

## Использованные источники

[1] Tsitsiklis J. N., Van Roy B. An Analysis of temporal-difference learning with function approximation // IEEE Transactions on Automatic Control. — 1997. — Vol. 42, № 5. — P. 674–690. — URL: **https://oreil.ly/aK17s**.

[2] Mnih V., Kavukcuoglu K., Silver D. et al. Playing Atari with deep reinforcement learning // ArXiv:1312.5602. — 2013. — December. — URL: **https://oreil.ly/HOqeN**.

[3] Zhang S., Sutton R. S. A Deeper look at experience replay // ArXiv:1712.01275. — 2018. — April. — URL: **https://oreil.ly/qnRKD**.

[4] Isele D., Cosgun A. Selective experience replay for lifelong learning // ArXiv:1802.10269. — 2018. — February. — URL: **https://oreil.ly/Wsk1d**.

[5] Mnih V. et al. Human-level control through deep reinforcement learning // Nature. — 2018. — № 518. — P. 529–533. — URL: **https://oreil.ly/AU1td**.

[6] Shedding light on energy on the EU: how are emissions of greenhouse gases by the EU evolving? // Shedding Light on Energy on the EU. — URL: **https://oreil.ly/yLgeU** (date of access: 12.09.2020).

[7] Marantos C., Lamprakos C. P., Tsoutsouras V., Siozios K., Soudris D. Towards plug&play smart thermostats inspired by reinforcement learning // Proceedings of the Workshop on INTelligent Embedded Systems Architectures and Applications. — 2018. — INTESA 2018. New York, NY, USA: Association for Computing Machinery. — P. 39–44. — URL: **https://oreil.ly/ozLFF**.

[8] Hessel M., Modayil J., Van Hasselt H. et al. Rainbow: combining improvements in deep reinforcement learning // ArXiv: 1710.02298. — 2017. — October. — URL: **https://oreil.ly/wKDL4**.

[9] Bellemare M. G., Dabney W., Munos R. A Distributional perspective on reinforcement learning // ArXiv:1707.06887. — 2017. — July. — URL: **https://oreil.ly/ODa3y**.

[10] Schaul T., Quan J., Antonoglou I., Silver D. Prioritized experience replay // ArXiv:1511.05952. — 2015. — November. — URL: **https://oreil.ly/0NPIN**.

[11] Fortunato M., Azar M. G., Piot B. et al. Noisy networks for exploration // ArXiv:1706.10295. — 2017. — June. — URL: **https://oreil.ly/TNcdO**.

[12] Wang Z., Schaul T., Hessel M. et al. Dueling network architectures for deep reinforcement learning // ArXiv:1511.06581. — 2015. — November. — URL: **https://oreil.ly/w4xRr**.

[13] Osband I., Blundell C., Pritzel A., Van Roy B. Deep exploration via bootstrapped DQN // ArXiv:1602.04621. — 2016. — July. — URL: **https://oreil.ly/RmkiV**.

[14] Lu X., Van Roy B. Ensemble Sampling // ArXiv: 1705.07347. — 2017. — November. — URL: **https://oreil.ly/7Bo99**.

[15] Chen R. Y., Sidor S., Abbeel P., Schulman J. UCB exploration via Q-ensembles // ArXiv:1706.01502. — 2017. — November. — URL: **https://oreil.ly/c2e4v**.

[16] Menon R. R., Ravindran B. Shared learning: enhancing reinforcement in Q-ensembles // ArXiv:1709.04909. — 2017. — September. — URL: **https://oreil.ly/dYjc7**.

[17] Jain S., Liu G., Mueller J., Gifford D. Maximizing overall diversity for improved uncertainty estimates in deep ensembles // ArXiv: 1906.07380. — 2019. — June. — URL: **https://oreil.ly/Ounz8**.

[18] Dabney W., Rowland M., Bellemare M. G., Munos R. Distributional reinforcement learning with quantile regression // ArXiv:1710.10044. — 2017. — October. — URL: **https://oreil.ly/pLAuN**.

[19] Lyle C., Castro P. S., Bellemare M. G. A Comparative analysis of expected and distributional reinforcement learning // ArXiv: 1901.11084. — 2019. — February. — URL: **https://oreil.ly/jY6-f**.

[20] Fujimoto S., Meger D., Precup D. Off-policy deep reinforcement learning without exploration // ArXiv:1812.02900. — 2019. — August. — URL: **https://oreil.ly/6EdcI**.

[21] Shah D., Xie Q. Q-learning with nearest neighbors // ArXiv:1802.03900. — 2018. — October. — URL: **https://oreil.ly/1VJH5**.

[22] Gaskett C., Wettergreen D., Zelinsky A. Q-Learning in continuous state and action spaces // Advanced Topics in Artifcial Intelligence / ed. by Norman Foo. Lecture Notes in Computer Science. — Berlin, Heidelberg: Springer, 1999. — P. 417–428. — URL: **https://oreil.ly/jMjWm**.

ГЛАВА 5

# Методы градиента политики

В *главе 2* были представлены методы оценки, которые позволяют агентам узнавать ожидаемую доходность при посещении каждой пары "состояние — действие". Затем агент может выбрать наиболее ценное действие на каждом временном шаге, чтобы максимизировать вознаграждение. Вы видели три разных способа оценки ожидаемой доходности: Монте-Карло, динамическое программирование и методы обучения с учетом временны́х различий, — но все они пытаются количественно оценить ценность каждого состояния.

Поразмыслите еще раз над проблемой. Почему вы хотите узнать ожидаемые значения? Они позволяют перейти к оптимальной политике. Q-обучение находит оптимальную политику, многократно максимизируя следующее ожидаемое значение. Но это косвенный путь к политике. Можно ли напрямую найти оптимальную политику? Ответ положительный. Методы на основе политик позволяют агенту выбирать действия, не обращаясь к функции ценности, и напрямую изучать оптимальную политику.

## Преимущества прямого изучения политики

В *разд. "Оптимальные политики" главы 2* политика определялась как вероятность действия при заданном состоянии. Q-обучение реализовало эту политику, направив действия на выработку оптимальной политики. Это упрощение подходит для детерминированных задач. Возьмем, к примеру, карточную игру блек-джек (также называемую 21 или очко). Вы играете, решая, взять ли другую карту, чтобы увеличить общую сумму чисел на картах, или отказаться, *придерживаясь* того, что у вас есть. При счете 22 или более вы проигрываете, обанкротившись. Другой игрок делает то же самое, а побеждает тот, кто набрал наибольшее количество очков.

Детерминированная политика — остановиться, когда набирается 21 очко, иначе вы каждый раз будете проигрывать. Для этой задачи подходят методы значений. Но что делать, если вы уже набрали 17 очков, а ваш противник нервничает? В реальной игре вы бы сделали прогноз, основанный на наблюдении за окружающими, который часто бывает стохастическим в реальных задачах. Детерминированная политика будет выбирать одно и то же действие каждый раз, независимо от вероятности того, что человек блефует. Но стохастическая политика может случайным образом выбирать между двумя вариантами, используя наблюдение как свидетельство того или другого.

Использование стохастической политики дает еще одно преимущество; она четко формирует дилемму исследования. Агент может выбирать действия в соответствии с произвольными вероятностями, устраняя необходимость в стратегиях принудительного исследования, таких как ε-жадная стратегия. А для задач, которые включают аппроксимацию функции (любой алгоритм, прогнозирующий Q-значения), наилучшей приближенной политикой может быть стохастическая.

Еще одним преимуществом является то, что предпочтения действий стохастической политики плавно меняются с течением времени. Методы ценности действия могут метаться между двумя почти одинаково вероятными действиями, создавая нестабильные переходы. Это происходит даже тогда, когда оценки ценности действия относительно стабильны; несущественных изменений Q-значений достаточно, чтобы кардинально изменить политику. Напротив, стохастические политики предоставляют возможность вероятностных действий, которые могут привести к более надежным политикам.

Но ключевое отличие состоит в том, что эту политику легче аппроксимировать. Промышленные проблемы, как правило, охватывают целый спектр задач. Иногда проще аппроксимировать функцию ценности действия, а иногда — напрямую получить политику. Там, где политику легче аппроксимировать, методы, основанные на политике, усваиваются быстрее.

## Как рассчитать градиент политики

Цель этого раздела — создать метод выработки оптимальной политики. В общем случае вы не можете использовать аналитический метод; политики потенциально нелинейны, а взаимодействие с окружающей средой делает задачу нестационарной (иными словами, распределение данных изменяется). В *главе 4* была представлена идея использования функции для аппроксимации оценок ценности действия. Это возможно, т. к. агент мог сохранять предыдущие взаимодействия со средой для последующего использования. Вы можете использовать аналогичную процедуру, чтобы найти оптимальную политику, но сначала вам нужно определить, что вы пытаетесь решить.

Напомним, что цель обучения с подкреплением (reinforcement learning, RL) — создать оптимальную политику $\pi_*$, которая максимизирует ожидаемую отдачу $G$.

Если бы у вас была полная наблюдаемость всех возможных траекторий, вы могли бы вывести оптимальную политику напрямую. Но в целом количество перестановок слишком велико (либо бесконечно для непрерывных состояний или действий). Вместо этого вы можете создать параметризованную модель политики и возиться с параметрами, пока не найдете подходящую политику.

Например, разумной долгосрочной стратегией покупки индексов может быть удержание, когда рынок растет, и покупка, когда рынок падает. Таким образом, вы выигрываете, когда рынок растет, и покупаете дешевые активы, когда рынок падает. Если предположить, что индекс продолжит расти в долгосрочной перспективе, мы будем иметь дело с безопасной стратегией с низкой волатильностью. Моделью

такой политики может быть линейный коэффициент, описывающий текущую тенденцию индекса. Вы можете найти оптимальную политику, протестировав множество различных значений линейного параметра. Но как определить производительность?

В области оптимизации производительность — это функция $J(\theta)$, параметризованная параметрами модели $\theta$. Уравнение 5.1 в данном случае является целевой функцией, которая представляет собой ожидание общего вознаграждения при следовании политике $\pi$.

**Уравнение 5.1. Целевая функция политики**

$$J(\theta) \doteq \mathbb{E}_{\pi}[G \mid s, a].$$

Предположим, что вы можете вычислить целевую функцию, тогда задача состоит в том, чтобы подтолкнуть параметры политики для максимизации доходности. Одним из популярных методов достижения этого является использование градиента целевой функции относительно параметров $\nabla_{\theta} J(\theta)$ для непосредственного улучшения. Это называется *упрощенным градиентным подъемом* (vanilla gradient ascent), как показано в уравнении 5.2 (обратите внимание, что градиентный спуск такой же, как градиентный подъем, но с измененным знаком для движения вниз, а не вверх).

**Уравнение 5.2. Упрощенный градиентный подъем**

$$\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta).$$

# Теорема о градиенте политики

Повторю идею предыдущего раздела: вы пытаетесь вычислить градиент общего вознаграждения по отношению к параметрам политики. Результат вычисления подскажет вам, как изменить параметры, чтобы увеличить вознаграждение.

Правда, всё зависит от расчета градиента целевой функции (см. уравнение 5.1) и не так просто, как может показаться, потому что производная требует немного сложной математики. Главная же проблема заключается в том, что градиентный подъем предполагает стационарное распределение данных. Агент на основе политики будет обновлять ее одновременно с генерацией траекторий. Это все равно что пытаться передвинуть ковер, стоя на нем. Позже я займусь нестационарной проблемой, а пока позвольте мне продолжить вывод градиента.

Думаю, важно, чтобы вы видели, как выводится градиент, поэтому я усердно поработал, чтобы максимально упростить вывод. Лучше всего прочитать этот вывод строка за строкой, обращаясь к объяснению ниже. Вы можете найти формальный вывод в ресурсах, приведенных в *разд. "Дополнительные материалы для чтения" в конце этой главы*. Я удалил ссылки на $\theta$; во всех случаях политика $\pi_{\theta}$ определя-

ется ее параметрами $\theta$. Также обратите внимание, что этот вывод предназначен для случая без учета дисконтирования. Результат вывода показан в уравнении 5.3.

**Уравнение 5.3. Градиент политики**

$$\nabla_\theta J(\theta) \doteq \nabla \mathbb{E}_\pi [G \mid s, a] = \tag{1}$$

$$= \nabla \sum_{a \in \mathcal{A}} \left[ Q(s,a)\pi(s,a) \right] \propto \tag{2}$$

$$\propto \sum_{a \in \mathcal{A}} \left[ Q(s,a)\nabla\pi(s,a) \right] \propto \tag{3}$$

$$\propto \sum_{a \in \mathcal{A}} \left[ Q(s,a)\pi(s,a)\frac{\nabla\pi(s,a)}{\pi(s,a)} \right] \propto \tag{4}$$

$$\propto \sum_{a \in \mathcal{A}} \left[ Q(s,a)\pi(s,a)\nabla \ln \pi(s,a) \right] \propto \tag{5}$$

$$\propto \mathbb{E}_\pi \left[ G\nabla \ln \pi(s,a) \right]. \tag{6}$$

Задача состоит в том, чтобы найти градиент целевой функции (уравнение 5.1), который я описал на шаге 1.

На шаге 2 я заменил ожидаемый доход реализацией. Можете выбрать любой, но если вы выберете функцию ценности действия, это немного упростит математические выкладки. Ожидаемый доход равен значению каждого действия $Q$, умноженному на вероятность посещения этого действия (как определено политикой $\pi$).

Уравнение 5.1 содержит большой объем математических преобразований между шагами 2 и 3. Формально это приводит к тому, что градиент зависит от распределения состояний. Другими словами, градиент пропорционален вероятности нахождения в определенном состоянии; если невозможно остаться в состоянии, то это не влияет на градиент. Следовательно, градиент пропорционален только изменениям в политике. Я решил скрыть эти шаги и распределение состояний, потому что в итоге оно отменяется из-за возврата ожидания. Если вас интересует формальное доказательство, см. источники в *разд. "Дополнительные материалы для чтения" далее в этой главе.*

Градиент берется с учетом параметров политики, поэтому я могу переместить функцию ценности действия $Q$ за пределы вычисления градиента на шаге 3.

Шаг 4 использует дерзкий математический трюк, который не столь очевиден. Я умножил выражение на $\pi(a \mid s)/\pi(a \mid s)$, что равно единице, потому это не меняет уравнения.

В шаге 5 используется обычное математическое тождество $\nabla \ln x = \nabla x / x$ для замены дроби натуральным логарифмом. Это распространенный вычислительный трюк для предотвращения умножения малых вероятностей, приводящих градиенты к нулю (это называется проблемой исчезающих градиентов).

Наконец, на шаге 6 напомним, что сумма значений действий, умноженная на вероятности действий, — это именно то, с чего я начал на шаге 2. Поэтому вместо того, чтобы суммировать все действия, я могу обязать агента следовать политике и возвращать ожидание этого конкретного действия с течением времени.

И вот — вывод алгоритма градиента политики. Подводя итог, можно сказать, что градиент доходности по отношению к политике равен ожидаемому значению доходности, умноженному на градиент политики. Другими словами, ожидаемая доходность увеличивает градиент политики. Это имеет смысл, т. к. предпочтительнее больше изменять параметры политики, когда ожидаемое значение выше.

Ключевой вывод состоит в том, что доход $G$ можно оценить множеством способов. Это единственное различие между многими известными алгоритмами градиента политики. Я вернусь к этому в *разд. "Сравнение основных алгоритмов градиента политики" далее в этой главе.*

Шаг 6 в уравнении 5.3 — это величина, которая может быть выбрана на каждом временно́м шаге, ожидание которого равно градиенту. Вы можете подключить результат математического вывода к алгоритму градиентного подъема (см. уравнение 5.2), чтобы создать правило обновления для агента, как в уравнении 5.4.

**Уравнение 5.4. Правило обновления градиента политики**

$$\theta \leftarrow \theta + \alpha G \nabla \ln \pi(a \mid s, \theta).$$

# Функции политики

Вернемся к уравнению 5.4. В алгоритме градиента политики отсутствуют две ключевые детали. Первая — это определение $G$, которое я обсуждаю в *разд. "Основные реализации" далее в этой главе* .

Вторая — это градиент политики. Мне нужно вычислить $\nabla \ln \pi(a \mid s, \theta)$, и для этого нужно указать мою функцию политики.

Должен признаться, я чувствую отчаяние из-за необходимости разрабатывать еще одну модель, но выбор важен. Политика определяет, насколько хорошо агент может планировать все возможные траектории в вашей среде. Он должен быть достаточно сложным, чтобы справляться со сложными траекториями, и в то же время довольно простым для обучения. Попытки предпринимались, чтобы автоматизировать этот процесс [1].

Главное преимущество и одна из основных причин популярности методов градиента политики заключается в том, что вы можете выбрать любую функцию, если она выводит распределение вероятностей и дифференцируется по $\theta$. Мир — ваша устрица, если модели — ваш жемчуг.

## Линейные политики

Если вы используете библиотеку, которая выполняет автоматическое дифференцирование, например TensorFlow или PyTorch, через обратное распространение, то вам даже не нужно думать о производной. Но я убежден, что размышлять над простыми задачами полезно — это помогает развить интуицию.

### Логистическая политика

Вам нужна функция, которая предсказывает действие с учетом состояния: $\pi_\theta(a \mid s)$. Напомним, что у политики есть два требования: она должна выводить вероятности и быть дифференцируемой. Представьте на секунду, что вы решаете проблему, имеющую два отдельных действия, например среду CartPole, где вам нужно решить, перемещать ли тележку влево или вправо (*см. разд. "Пример: глубокая Q-сеть в среде CartPole" главы 3* для описания окружающей среды). Проблема становится задачей предсказания одного из двух классов с учетом некоторых входных данных; должны выводиться вероятности. Естественным выбором для этого типа задач является *логистическая функция*, обычно используемая при классификации, поскольку она проста, линейна, легко дифференцируется и естественным образом выводит оценки вероятности (в соответствии с логистическим распределением, которое имеет большие хвосты по сравнению с нормальным распределением). Логистическая функция, сосредоточенная вокруг нуля с максимальным значением 1, представлена в уравнении 5.5 (также известном как *сигмоидальная функция*).

**Уравнение 5.5. Логистическая функция**

$$f(x) \doteq \frac{1}{1+e^{-x}}.$$

Среда CartPole имеет четыре состояния, поэтому $\theta$ — это вектор с четырьмя весами. Для того чтобы использовать эту функцию политики, введите состояние и текущие оценки весов, и она выведет вероятность выбора действия. Вначале веса распределяются случайным образом, а во время обучения их подталкивают к выбору лучших действий. Эти шаги покажутся вам знакомыми, если вы привыкли к контролируемому машинному обучению.

В общем, вы можете использовать несколько логистических функций для моделирования вероятности многоклассовых задач (это эквивалентно функции softmax, которую я использую в *разд. "Политика softmax" далее в этой главе*). Но поскольку эта среда имеет только два действия, а результат уравнения 5.5 охватывает диапазон от нуля до единицы, вы можете задать вероятность одного класса равной $f(\theta, s)$, а вероятность другого — равной $1 - f(\theta, s)$. Это *фиктивное кодирование*, которое используется при очистке данных. Политика, прогнозирующая действие $a$, заданные состояния $s$ на основе логистической функции, приведена в уравнении 5.6.

**Уравнение 5.6. Логистическая политика**

$$\boldsymbol{\pi}_\theta(a \mid s) \doteq \begin{bmatrix} \pi_\theta(a=0 \mid s) \\ \pi_\theta(a=1 \mid s) \end{bmatrix} = \begin{bmatrix} \dfrac{1}{1+e^{-\theta_0^T s}} \\ 1 - \dfrac{1}{1+e^{-\theta_1^T s}} \end{bmatrix}.$$

Уравнение 5.4 обеспечивает правило улучшения весов, но для этого вам понадобится градиент уравнения 5.6. Результат показан в уравнении 5.7, но полный расчет приведен в *приложении 1*.

**Уравнение 5.7. Градиент логистической политики**

$$\nabla \ln \boldsymbol{\pi}(a \mid s, \boldsymbol{\theta}) = \begin{bmatrix} s - s\pi\left(\boldsymbol{\theta}_0^T s\right) \\ -s\pi\left(\boldsymbol{\theta}_1^T s\right) \end{bmatrix}.$$

У градиента есть интуитивное объяснение. Уравнение 5.7 — это разница между наблюдаемым состоянием и состоянием, ожидаемым политикой.

## Политика softmax

Функцию softmax можно воспринимать как многоклассовую версию логистической функции. Это означает, что вы можете использовать эту версию функции для представления вероятности выбора из нескольких действий. Политика softmax определена в уравнении 5.8.

**Уравнение 5.8. Политика softmax**

$$\pi_\theta(a \mid s) \doteq \frac{e^{\theta^T s}}{\sum_a e^{\theta_a^T s}}.$$

Другими словами, уравнение 5.8 представляет собой логистическую функцию, нормализованную по всем возможным действиям, чтобы можно было гарантировать равенство единице всех действий в сумме. Соответствующий градиент приведен в уравнении 5.9, а полный вывод представлен в *приложении 2*.

**Уравнение 5.9. Градиент политики softmax**

$$\nabla \ln \pi(a \mid s, \boldsymbol{\theta}) = s - \sum_a s\pi\left(\boldsymbol{\theta}^T s\right).$$

Как и *разд. "Логистическая политика" ранее в этой главе*, градиент в уравнении 5.9 выражается как разница между наблюдаемыми и ожидаемыми векторами признаков.

## Произвольные политики

Вычисление производных политик вручную позволило мне показать вам, как именно я могу улучшить политику с помощью итераций. Но в целом этот процесс утомительный и запутанный. Вместо этого вы можете вычислить производную тремя способами: методом конечных разностей, автоматическим дифференцированием и символьным дифференцированием.

Конечная разность $\nabla f(x) = (f(x+h) - f(x)/2h)$ вычисляет разницу в политике за небольшой шаг в пространстве параметров. Это просто, но при этом возможны числовые ошибки из-за размера шага $h$.

Инструменты автоматического дифференцирования, такие как JAX[1] или методы обратного распространения в Tensor Flow/PyTorch/MXNet/другие, пытаются отслеживать использование параметра в уравнении для построения ориентированного ациклического графа (directed acyclic graph, DAG). Библиотеки DAG сопоставляют каждый шаг с соответствующей производной и объединяют их с помощью правила цепочки.

Символьное дифференцирование с использованием таких инструментов, как SymPy[2], — это фактически то же самое, что и автоматическое дифференцирование, за исключением того, что оно обычно работает в одном направлении (вперед). Инструменты символьного дифференцирования также уделяют больше внимания рефакторингу, который может приводить к появлению более простых для понимания производных, в отличие от автоматических методов, которые способны создавать цепочки цепочек и т. д.

Другими словами, если вы можете использовать библиотеку, рекомендую вам не проводить дифференцирование вручную. Используйте библиотеку, которая подходит для вашего случая.

# Основные реализации

В *разд. "Функции политики" ранее в этой главе* было показано, как обеспечить функцию градиента, требуемую уравнением 5.4. Оставшаяся деталь — использование $G$. Напомним, что это общая награда от точки до конца эпизода, если она эпизодическая, или навсегда, в любом другом случае (*см. разд. "Дисконтированные вознаграждения" главы 2*).

В *главах 2–4* показаны три различных способа расчета ожидаемой доходности: от методов Монте-Карло, методов полной траектории до методов аппроксимации и многошаговых проекций. Эти детали реализации являются единственной разницей между несколькими базовыми реализациями градиента политики, включая PREINFORCE и алгоритмы "актор — критик".

---

[1] См. **https://oreil.ly/lPjjA**.

[2] См. **https://www.sympy.org/**.

## Метод Монте-Карло (алгоритм REINFORCE)

Первый алгоритм использует подход Монте-Карло. Сначала он собирает траекторию, основанную на текущей политике, для всего эпизода. Затем обновляет политику с помощью уравнения 5.4.

REINFORCE приведен в алгоритме 5.1. На шаге 6 $G$ ссылается на дисконтированное вознаграждение с момента $t$ до конца эпизода, как в алгоритме 2.3. На шаге 7 я использую значение $G$ для взвешивания дифференциала политики. Дифференциал вычисляет направление, в котором вам нужно подтолкнуть вероятность выбора действия, определяемого $\theta$. Коллективный вес $\alpha\gamma^t G$ определяет, насколько вам нужно его подтолкнуть. На шаге 7 я также снова ввожу дисконтирование, которое извлекается из производной, поскольку оно не зависит от $\theta$.

Проблема здесь в том, что REINFORCE будет продолжать продвигать действие пропорционально рассчитанной прибыли. Даже если агент сделал правильный выбор, REINFORCE продолжит поднимать вероятность выбора этого действия еще выше. В долгосрочной перспективе он может перевесить относительную важность действия по сравнению с другими действиями, что повлияет на эффективность обучения. Например, все действия могут иметь равный ожидаемый доход, но REINFORCE всегда будет увеличивать вероятность последнего действия.

**Алгоритм 5.1. Алгоритм REINFORCE**

1: **input**: дифференцируемая параметризованная политика $\pi(a \mid s, \theta)$, размер шага $\alpha > 0$.

2: Инициализировать $\theta$ произвольно.

3: **loop** для каждого эпизода:

4: Генерировать траекторию, которая следует политике $\pi(\cdot \mid \cdot, \theta)$.

5: **for** каждого шага в эпизоде $t \leftarrow 0, 1, ..., T$:

6: $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} r^{(k)}$.

7: $\theta \leftarrow \theta + \alpha\gamma^t G \nabla\pi(a \mid s, \theta)$.

8: **endfor**

## Пример: алгоритм REINFORCE в среде CartPole

Красота использования градиентов политики демонстрируется низкой размерностью политики. Среда CartPole имеет четыре функции. Линейная политика имеет четыре параметра, которые определяют важность каждой функции. Результатом является вероятностное представление наилучшего курса действий. Если угол наклона шеста близок к углу падения, тогда вероятность выбора действия, предусматривающего исправление этой ситуации, очень высока. Награды, полученные

в процессе обучения, показаны на рис. 5.1. Для того чтобы можно было получить хорошие награды, политика должна смотреть далеко в будущее, поэтому я установил $\gamma \doteq 0{,}99$. Я выбрал $\alpha \doteq 0{,}01$, т. к. при более высоких значениях обучение было нестабильным.

Сравните этот результат с табличной реализацией в *разд. "Пример: глубокая Q-сеть в среде CartPole" главы 4*. Там мне пришлось использовать нейронную сеть для аппроксимации пространства состояний. Здесь у меня линейное отображение, использующее всего четыре параметра. Элегантность и простота использования просто невероятны.

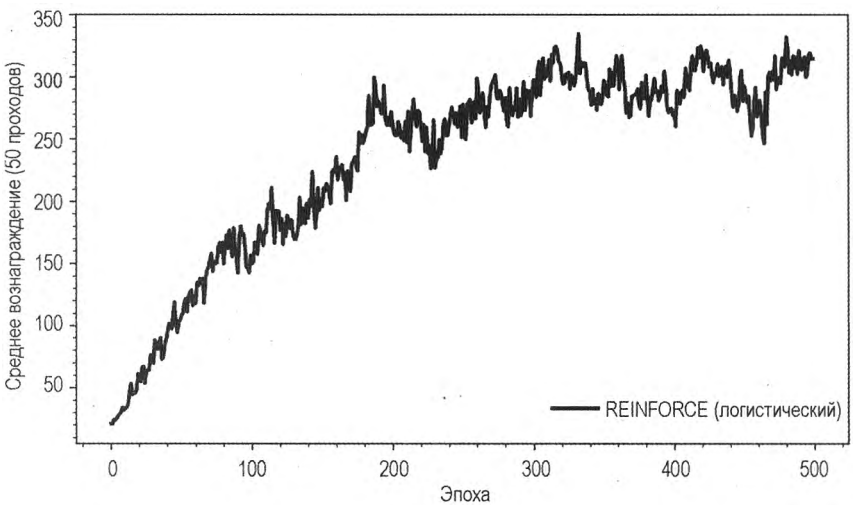



**Рис. 5.1.** Среднее вознаграждение при использовании алгоритма градиента политики на основе REINFORCE в среде CartPole

## Алгоритм REINFORCE с базовыми показателями

Более распространенная реализация включает базовый ожидаемый доход как функцию преимущества, показанный в *разд. "Дуэльные сети" главы 4* (см. уравнение 4.4). Это помогает улучшить нестабильность градиентов из-за больших изменений среднего вознаграждения.

Представьте сценарий (например, CartPole), который вознаграждает на каждом временно́м шаге. Такой подход обеспечивает высокую ожидаемую доходность на ранних временных этапах траектории. Большие изменения значения создают большие обновления градиентов (как определено уравнением 5.4). Но агент не должен заниматься махинациями средней прибыли; цель — найти оптимальные действия. Вместо этого агент должен узнать, как каждое действие увеличивает ожидаемую доходность по сравнению со средним значением.

Алгоритмы преимуществ реализуют это, поддерживая прогноз текущей средней доходности для данного состояния, который совпадает с функцией ценности со-

стояния (см. уравнение 2.8). И, как и раньше, вы можете аппроксимировать функцию ценности состояния разными способами: например, табличными методами или аппроксимацией функций. Поскольку это параметризованный алгоритм Монте-Карло, естественно обновлять параметризованную оценку ценности состояния аналогичным образом, вычисляя ошибку между прогнозируемыми и фактическими значениями и подталкивая параметры в соответствии с правилом обновления по полным траекториям эпизодов.

Конечный результат — повышение стабильности обучения. Вычитание базовых показателей означает, что каждое обновление градиента является менее экстремальным, что приводит к меньшей дисперсии градиентов. В долгосрочной перспективе вы должны заметить, что вознаграждения, полученные во время обучения, более стабильны. Но это в значительной степени зависит от сложности пространства состояний, возможности предсказать функцию ценности состояния и сложности действий. В примере CartPole обе функции очень просты, поэтому не надейтесь получить небольшую разницу по сравнению со стандартным алгоритмом REINFORCE.

REINFORCE с базовыми показателями приведен в алгоритме 5.2. Основное отличие заключается в добавлении шагов 7 и 8. Здесь я тренирую вторую линейную функцию, чтобы аппроксимировать функцию ценности состояния. Шаг 7 вычитает приближение из вознаграждения, наблюдаемого на этой траектории. Применение политики к весовым коэффициентам на шаге 9 способствует изменению вероятности выбора действия в соответствии с разницей. Если разница больше нуля (вознаграждение выше базового уровня), вероятность увеличивается. Если разница меньше нуля, вероятность уменьшается.

**Алгоритм 5.2. Алгоритм REINFORCE с базовыми показателями**

1: **input**: дифференцируемая параметризованная политика $\pi(a \mid s, \theta)$, дифференцируемая параметризованная функция ценности состояния $V(s, w)$, размер шагов $0 < \alpha_w < 1$ и $0 < \alpha_\theta < 1$.

2: Инициализировать $\theta$ и $w$ произвольно.

3: **loop** для каждого эпизода:

4: Генерировать траекторию, которая следует политике $\pi(\cdot \mid \cdot, \theta)$.

5: **for** каждого шага в эпизоде $t \leftarrow 0, 1, ..., T$:

6: $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} r^{(k)}$.

7: $\delta \leftarrow G - V(s, w)$.

8: $w \leftarrow w + \alpha_w \delta \nabla V(s, \alpha_w)$.

9: $\theta \leftarrow \theta + \alpha_\theta \gamma^t \delta \nabla \pi(a \mid s, \theta)$.

10: **endfor**

## Пример: алгоритм REINFORCE с базовыми показателями в среде CartPole

Я внес несколько изменений в эксперимент, описанный в *разд. "Пример: алгоритм REINFORCE в среде CartPole" ранее в данной главе*, за исключением реализации алгоритма 5.2. Основным архитектурным решением является расчет оценки "состояние — значение" $V(s,w)$. В своих экспериментах я изо всех сил пытался найти простую функцию, которая соответствовала бы данным. Это произошло потому, что лежащее в основе состояние является удивительно сложным.

Для того чтобы эксперимент был максимально простым, я решил использовать скользящее среднее ранее наблюдаемых вознаграждений; другими словами, онлайн-реализацию среднего вознаграждения. Далее я переобучил агента и сохранил состояние и дисконтированные вознаграждения. Затем я выполнил анализ главных компонент (principal component analysis, PCA) состояния, чтобы уменьшить количество измерений с четырех до одного с целью упростить пространство функций и обеспечить возможность построения графиков.

Вы можете узнать больше о снижении размерности и анализе главных компонент в любой книге по машинному обучению; некоторые рекомендации приведены в *разд. "Дополнительные материалы для чтения" в конце этой главы.*

На рис. 5.2 показан график зависимости первой главной компоненты состояния от дисконтированного вознаграждения. Награды образуют треугольную форму вокруг нулевого значения состояния. Прогнозируемое значение будет выше, если вы сохраните значения состояния близко к нулю. Другими словами, если значения состояния отличны от нуля, то это, как правило, приводит к более низкому общему

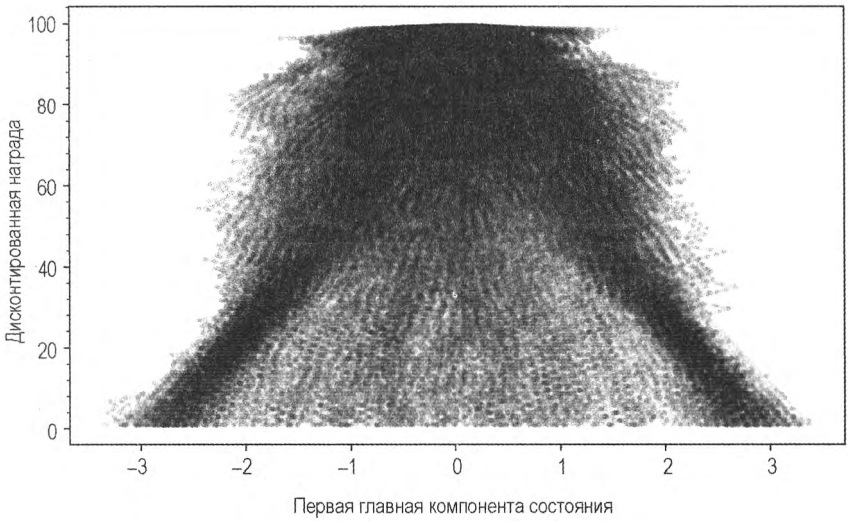



**Рис. 5.2.** Зависимость первой главной компоненты состояния CartPole от дисконтированного вознаграждения

вознаграждению, потому что шест падает. Мне также кажется забавным, что сюжет очень похож на *всевидящее око* на оборотной стороне долларовой банкноты.

Вы можете заметить, что простое среднее или линейное приближение плохо предсказывает значение. Кроме того, я показал первую главную компоненту на построенном графике. Данные, вероятно, также осложнены во втором, третьем и четвертом измерениях.

Я пока буду придерживаться своего выбора использования среднего и рассмотрю более сложную функцию аппроксимации в *разд. "Пример: n-шаговый алгоритм „актор — критик" в среде CartPole" далее в этой главе*. Но в целом важно учитывать этот компромисс. Вы стремитесь как можно лучше предсказать функцию ценности состояния, но также хотите, чтобы аппроксимация функции была надежной (в том смысле, что невидимое состояние не должно приводить к нелепому результату).

Результаты моего эксперимента можно увидеть на рис. 5.3. Мне пришлось проводить эксперимент в течение 50 различных проходов, чтобы получить результаты, которые были статистически значимыми при одних и тех же настройках. Обратите внимание, что вознаграждения обоих алгоритмов изначально почти совпадают, но позже расходятся. Это указывает на то, что базовый алгоритм немного стабильнее. В алгоритме REINFORCE обновления с большим градиентом случайно возвращают агента к худшей производительности. Базовый агент способен стабильно продолжать повышать производительность.

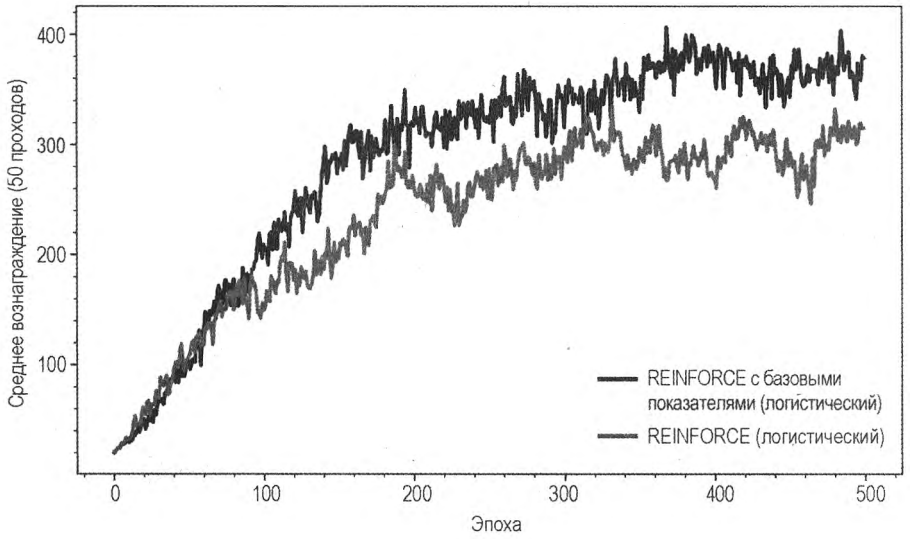



**Рис. 5.3.** Среднее вознаграждение при использовании стандартного алгоритма REINFORCE и алгоритма REINFORCE с базовыми показателями политики в среде CartPole

## Уменьшение градиентной дисперсии

В *разд. "Алгоритм REINFORCE с базовыми показателями" ранее в данной главе* я сказал, что метод базовых показателей уменьшает дисперсию градиентов. В таких алгоритмах, как REINFORCE или других базовых методах Монте-Карло (*см. разд. "Генерирование политики Монте-Карло" главы 2*), цель состоит в том, чтобы изменить политику и максимизировать вознаграждение. Методы градиента политики достигают этого, подталкивая политику в определенном направлении и на определенную величину. Они рассчитывают направление, используя производную политики. Но величина для REINFORCE — это текущая ожидаемая доходность.

Проблема заключается в том, что ожидаемая доходность последующих выборок может значительно отличаться. Например, представьте, что политика достаточно эффективна в среде CartPole. Шест может случайно упасть в начале эпизода. Это вызовет большую разницу в ожидаемом значении и большое обновление градиента. Несмотря на то что политика была хорошей, агент форсировал большое обновление.

Другой пример — среда Pong, описанная в *разд. "Пример: радужная глубокая Q-сеть в Atari Games" главы 4*. В ранних эпизодах штраф близок к –21. Все действия приводят к отрицательной награде. Поскольку все параметры плохие, обновление градиента подавляет все действия. Лучшее решение — посмотреть на изменение вознаграждения. Агент должен использовать относительную разницу действий по сравнению со средним значением.

Я зафиксировал обновления градиента от одного запуска стандартного алгоритма REINFORCE и алгоритма REINFORCE с базовыми показателями. Стандартное отклонение градиентов представлено на рис. 5.4. Обратите внимание, что после

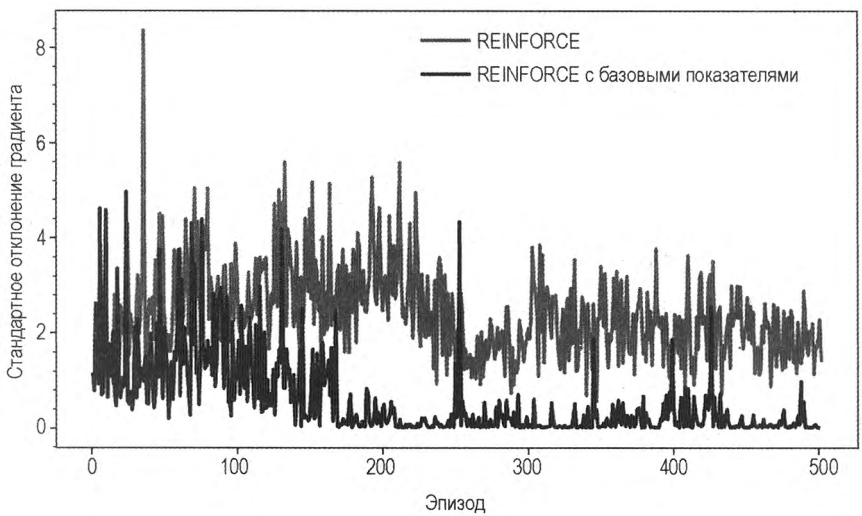



**Рис. 5.4.** Стандартное отклонение градиента при использовании стандартного алгоритма REINFORCE и алгоритма REINFORCE с базовыми показателями политики в среде CartPole

периода обучения стандартное отклонение алгоритма с базовыми показателями резко падает, тогда как стандартный алгоритм REINFORCE продолжает создавать большие градиенты, которые изменяют политику, даже после получения приличного вознаграждения. Когда в литературе говорится об "уменьшении дисперсии", подразумевается именно это: методы повышения стабильности градиентов без влияния на производительность обучения.

## *n*-Шаговый и улучшенный алгоритмы "актор — критик"

Следующее улучшение ведет нас по пути, выбранному с помощью с табличных методов. Агенты Монте-Карло не загружаются, а значит, им нужно дождаться окончания эпизода, прежде чем они смогут выполнить какое-либо обучение. Как и в *главе 3*, решение состоит в том, чтобы проработать методы обучения с учетом временны́х различий вплоть до *n*-шаговых методов. Однако *n*-шаговые алгоритмы на основе градиента политики имеют интересную интерпретацию, которую вы раньше не видели.

Прежде чем я перейду к сути, позвольте мне описать *n*-шаговую реализацию, т. к. есть одно существенное изменение по сравнению с алгоритмом 5.2. На шаге 5 алгоритма 5.3 выполняется обновление для каждого временно́го шага. Это больше не алгоритм Монте-Карло. Начальная загрузка (*см. разд. "n-Шаговые алгоритмы" главы 3*) улучшает скорость обучения, а шаг 8 реализует бутстрэппинг, чтобы обеспечить одношаговые обновления $G_{t:t+1} \doteq r + \gamma V(s', w)$, как в уравнении 3.3.

**Алгоритм 5.3. *n*-Шаговый алгоритм "актор — критик"**

1: **input**: дифференцируемая параметризованная политика $\pi(a \mid s, \theta)$, дифференцируемая параметризованная функция ценности состояния $V(s, w)$, размер шагов $\alpha_w$ и $\alpha_\theta > 0$.

2: Инициализировать $\theta$ и $w$.

3: **loop** для каждого эпизода:

4: Инициализировать $I \leftarrow 1$, $t \leftarrow 0$ и $s$.

5: **while** $s$ не является терминальным состоянием:

6: Выбрать действие $a$ из состояния $s$, используя политику $\pi$ (неоднозначности разрешить случайным образом).

7: Выполнить действие $a$, а затем наблюдать за следующим состоянием $s'$ и наградой $r$.

8: $\delta \leftarrow G_{t:t+n} - V(s, \theta)$.

9: $w \leftarrow w + \alpha_w \delta \nabla V(s, w)$.

10: $\theta \leftarrow \theta + \alpha_\theta I \delta \nabla \ln \pi(a \mid s, \theta)$.

11: $I \leftarrow \gamma I$.

12: $s \leftarrow s'$.

Алгоритм 5.3 является улучшенным алгоритмом 5.2 и включает возврат временно́й разницы. Значение этого алгоритма больше, чем кажется. Вы получили алгоритм, способный изучать как политику, так и ее функцию ценности состояния на каждом отдельном шаге, в отличие от изучения базовых показателей по всем траекториям (алгоритм базовых показателей, вдохновленный методом Монте-Карло).

Это различие важно. Политика изучает и описывает наилучшие действия, которые следует предпринять. На языке RL это называется *а́ктором* (actor). Прогнозирование базового уровня из $n$ шагов критикует последствия действия и используется для обновления политики. Это *критик* (critic). Вместе они составляют основу алгоритмов "актор — критик", часто их называют алгоритмом *"актор — критик" с преимуществом* (advantage actor-critic, A2C). Обратите внимание, что алгоритмы Монте-Карло не считаются алгоритмами "актор — критик", поскольку они не обновляются на каждом шаге.

Я представил алгоритм таким образом, потому что вы знакомы с разработкой из *главы 3*. Но другой способ сформулировать проблему — рассматривать критика как меру преимущества выбора действия по сравнению с базовыми показателями. Это точное определение функции преимущества из уравнения 4.4.

В контексте алгоритмов градиента политики единственное отличие состоит в том, что агент по определению следует текущей оптимальной политике, поэтому вам не нужно вычислять значение каждого действия. Вам необходимо только значение для состояний, которые собирается посетить агент. Приближение представлено в уравнении 5.10 [2].

**Уравнение 5.10. Аппроксимация функции преимущества для алгоритмов градиента политики**

$$
\begin{aligned}
A^{\pi}(s,a) \doteq Q(s,a) - V(s) = \\
= r + Q(s',a') - V(s) \approx \quad & \text{(ошибка временных различий)} \\
\approx r + V_{\pi}(s') - V(s) \quad & \text{(когда агент следует оптимальной политике } \pi).
\end{aligned}
$$

Можно напрямую оценить функцию преимущества или функцию ценности действия, но в большинстве реализаций используется версия ценности состояния с целью уменьшения количества моделей, которые необходимо обучить.

Другой частый вопрос — необходимость $I$ на шаге 11 алгоритма 5.3. При включении дисконтирования (*см. разд. "Дисконтированные вознаграждения" главы 2*) в теорему о градиенте политики (см. уравнение 5.3) $\gamma$ является константой, не участвующей в вычислении градиента. Но дисконтирование зависит от номера шага, и в онлайн-алгоритме, подобном этому, имеет смысл реализовать его таким образом, хотя вы можете просто заменить его на $\gamma^t$, если хотите.

## Пример: *n*-шаговый алгоритм "актор — критик" в среде CartPole

Цель этого примера — продемонстрировать различия между методом Монте-Карло (*см. разд. "Алгоритм REINFORCE с базовыми показателями" ранее в этой главе*) и реализациями градиента политик с учетом временны́х различий, созданными с помощью самозагрузки.

Напомним, что для начальной загрузки требуется функция, которая оценивает ожидаемый возврат данного состояния. Если бы я использовал скользящее среднее вознаграждения, как сделал в *разд. "Пример: алгоритм REINFORCE с базовыми показателями в среде CartPole" ранее в этой главе*, тогда оценки для $V(s,w)$ и $V(s',w)$ были бы одинаковыми. Это дает $\delta = r + (1-\gamma)V \approx r$. Использование предыдущего базового показателя приводит к обновлениям, которые имеют вес 1 в среде CartPole. Другими словами, у критика нет мнения. И вы должны разработать функцию, которая более способна предсказывать значение для данного состояния. Эффективность обучения алгоритма "актор — критик" зависит от точного приближения значения.

На рис. 5.2 вы видели, что вознаграждение — это сложная функция состояния. Простая линейная комбинация состояния и некоторых весов не способна моделировать эту сложность. Сейчас есть соблазн сразу перейти к глубокому обучению, но я все же рекомендую сначала попробовать линейные подходы. Они более жесткие, их легче обучать, но сложнее переобучить. В среде CartPole я могу конструировать функции для состояния, чтобы получить функции, которые можно изучить с помощью линейной комбинации весов.

После некоторых экспериментов я решил использовать кодирование ячеек для преобразования двух последних состояний (угол наклона шеста и угловая скорость) в сетку из 10 ячеек на состояние с 5 перекрытиями (вы можете узнать больше о методах разработки общих функций в *разд. "Преобразование в дискретные состояния" главы 9*). В результате получается 605 функций. Я использовал 605 весов, чтобы узнать линейное приближение этих функций. Настройки такие же, как и раньше, за исключением нового параметра затухания для весов $\alpha_w \doteq 0{,}1$. Обычно рекомендуется, чтобы оценка стоимости имела скорость обучения, превышающую или равную скорости обучения политике. Другими словами, оценка стоимости должна учиться по крайней мере так же быстро, как и политика.

На рис. 5.5 показаны результаты преднамеренной установки скорости обучения политики $\alpha_\theta$ в соответствии с конечной эффективностью обучения предыдущих алгоритмов.

На рис. 5.6 показаны результаты после ускорения затухания политики (как для алгоритмов критического анализа, так и для базового алгоритма) до $\alpha_\theta \doteq 0{,}1$. Все остальные настройки остаются прежними. Увеличение скорости затухания означает, что обновлений, внесенных в политику, больше. Политика меняется быстрее. Образно говоря, пересев с четырехколесного велосипеда на двухколесный, вы можете увидеть, что *n*-шаговый алгоритм "актор — критик" может не отставать. Он

быстро увеличивает свое вознаграждение в течение первых 50 эпизодов и устанавливает вознаграждение в размере 400 (что зависит от точности моделирования политики и функции ценности, а не от скорости обучения). Однако базовый алгоритм не может обучаться быстрее. Обновления по методу Монте-Карло значительно снижают производительность обучения. Они действуют как тормоз; изменение скорости обучения политики не оказывает никакого эффекта (и может повлиять на долгосрочную стабильность).

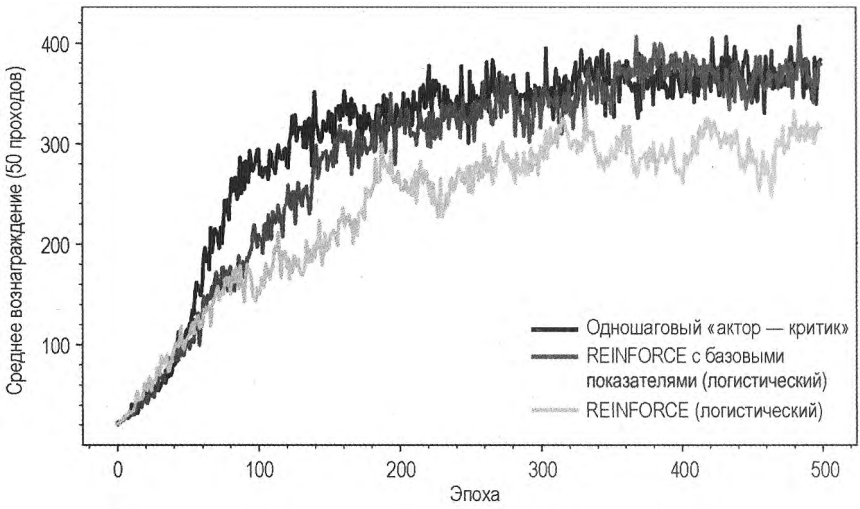



**Рис. 5.5.** Среднее вознаграждение с использованием одношагового алгоритма градиента политики "актор — критик" в среде CartPole

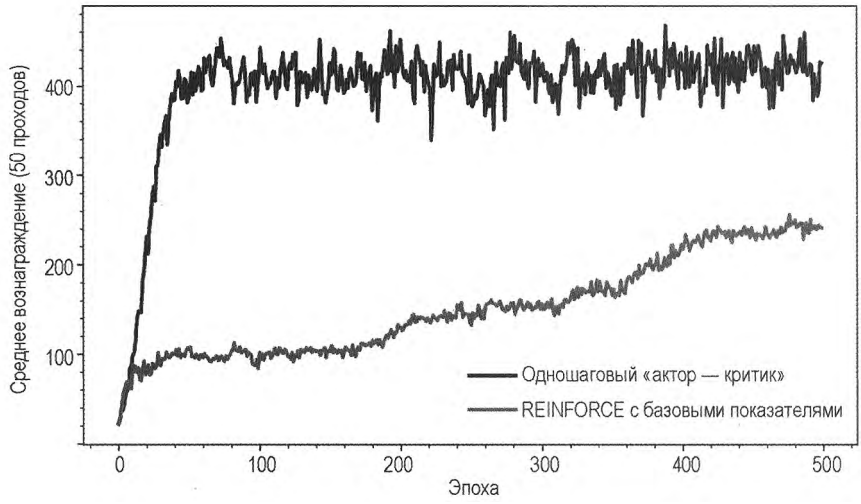



**Рис. 5.6.** Среднее вознаграждение для среды CartPole после изменения скорости затухания политики ($\alpha_\theta$) с 0,01 до 0,1 как для алгоритмов "актор — критик", так и для алгоритмов с базовыми показателями

## Темпы затухания ценностного обучения по сравнению с темпами ослабления политики

Параметры скорости ухудшения, обозначенной $\alpha$, регулируют размер скачка при обновлении. Поскольку мы имеем дело с алгоритмом градиентного спуска, следует выбрать достаточно большую скорость затухания для быстрого обучения, но достаточно маленькую, чтобы приблизиться к наилучшей возможной политике. Правда, *n*-шаговый алгоритм градиента политики использует две скорости спада: одну для модели, которая оценивает функцию ценности состояния, а другую — для политики.

Политика использует оценку ценности состояния в своем обновлении, которое вводит зависимость. Установка оценки ценности состояния $\alpha$ на медленное обновление ограничит политику. Это связано с тем, что ошибка временно́го различия в последовательных шагах будет близка к нулю и приведет к небольшому изменению политики. Установка $\alpha$ на быстрое обновление приведет к чрезмерным изменениям ошибки вре́менной разницы и большим изменениям в политике. После изменения политики истинное значение состояния также изменится, и для оценки ценности состояния потребуется еще одно большое обновление. Конечным результатом является нестабильность, и тогда агент вряд ли сможет сойтись.

Результаты, показанные на рис. 5.7, получены в ходе эксперимента в среде CartPole, где я зафиксировал скорость ослабления политики $\alpha_\theta \doteq 0{,}1$ и изменил скорость спада оценки ценности состояния $\alpha_w \doteq \{0{,}01;\ 0{,}1;\ 0{,}5\}$. Результаты показывают, что оптимальная скорость спада составляет приблизительно 0,1, хотя значение 0,01 может повысить производительность в долгосрочной перспективе, поскольку меньший размер шага приближает нас к глобальным оптимумам. Значение 0,5 слишком велико; это вызывает нестабильные обновления и препятствует сходимости.

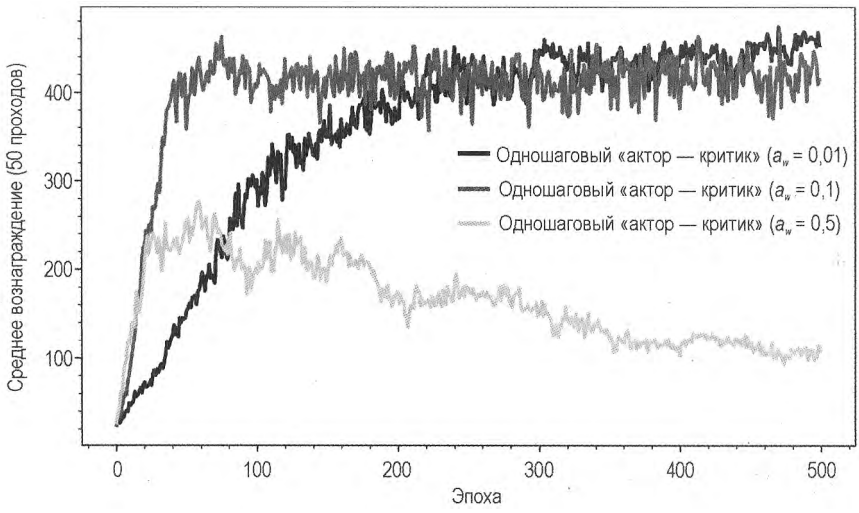



**Рис. 5.7.** Среднее вознаграждение для среды CartPole при фиксировании скорости ослабления политики и изменении скорости спада оценки ценности состояния

Существует популярная математическая теорема об отсутствии бесплатного обеда (no free lunch, NFL)[3], которую я чрезмерно обобщаю как доказательство того, что ни один алгоритм, реализация или гиперпараметры не подходят для всех приложений. Точно так же оптимальные скорости ослабления полностью зависят от проблемы и алгоритма. Проектировать и настраивать агентов нужно для каждой новой ситуации [3].

## Трассировки соответствия алгоритма "актор — критик"

Согласно изменениям, описанным в *разд. "Трассировки соответствия" главы 3*, в алгоритме 5.4 приведен алгоритм "актор — критик", основанный на отслеживании соответствия. Как и раньше, в нем есть гиперпараметр, позволяющий контролировать уровень бутстрапирования. Оптимальное значение гиперпараметра трассировочного затухания зависит от конкретной задачи. Большие значения обновляют веса, которые были в далеком прошлом. Маленькие значения обновляют только недавние веса.

Основное различие между алгоритмами 5.4 и 5.3 заключается во включении трассировщиков, которые следят за тем, какие состояния необходимо обновить на шагах 9 и 10. Это в конечном счете подталкивает весовые коэффициенты оценки значения $w$ и веса политики $\theta$ в направлении, в котором политика уже двигалась в прошлом.

**Алгоритм 5.4. Алгоритм "актор — критик" с квалификационными признаками**

1: **input**: дифференцируемая параметризованная политика $\pi(a|s,\theta)$, дифференцируемая параметризованная функция ценности состояния $V(s,w)$, размер шагов $0<\alpha_w<1$ и $0<\alpha_\theta<1$, скорости трассировочного затухания $0\le\lambda_\theta,\ \lambda_w\le 1$.

2: Инициализировать $\theta$ и $w$ произвольно.

3: **loop** для каждого эпизода:

4: Инициализировать $t\leftarrow 0$, $I\leftarrow 1$ и $s$,
$z_w\leftarrow 0$ (та же размерность, что и у $w$),
$z_\theta\leftarrow 0$ (та же размерность, что и у $\theta$).

5: **while** $s$ не является терминальным состоянием:

6: Выбрать действие $a$ из состояния $s$, используя политику $\pi$ (неоднозначности разрешить случайным образом).

7: Выполнить действие $a$, а затем наблюдать за следующим состоянием $s'$ и наградой $r$.

---

[3] Фольклорная теорема об отсутствии бесплатного обеда подразумевает, что если алгоритм хорошо работает с определенным классом задач, то это обязательно компенсируется снижением производительности на множестве всех оставшихся задач. — *Прим ред*

8: $\delta \leftarrow r + \gamma V(s', w) - V(s, \theta)$.

9: $z_w \leftarrow \gamma \lambda_w z_w + \nabla V(s, w)$.

10: $z_\theta \leftarrow \gamma \lambda_\theta z_\theta + I \nabla \ln \pi(a \mid s, \theta)$.

11: $w \leftarrow w + \alpha_w \delta z_w$.

12: $\theta \leftarrow \theta + \alpha_\theta I \delta z_\theta$.

13: $I \leftarrow \gamma I$.

14: $s \leftarrow s'$.

## Пример: трассировка соответствия требованиям алгоритма "актор — критик" в среде CartPole

Я внес незначительные изменения в пример из *разд. "Пример: n-шаговый алгоритм „актор — критик" в среде CartPole" данной главы*, чтобы включить в него спад трассировочного соответствия. В этом эксперименте используется $\alpha_\theta = \alpha_w = \lambda_\theta = \lambda_w \doteq 0{,}1$, результаты можно увидеть на рис. 5.8.

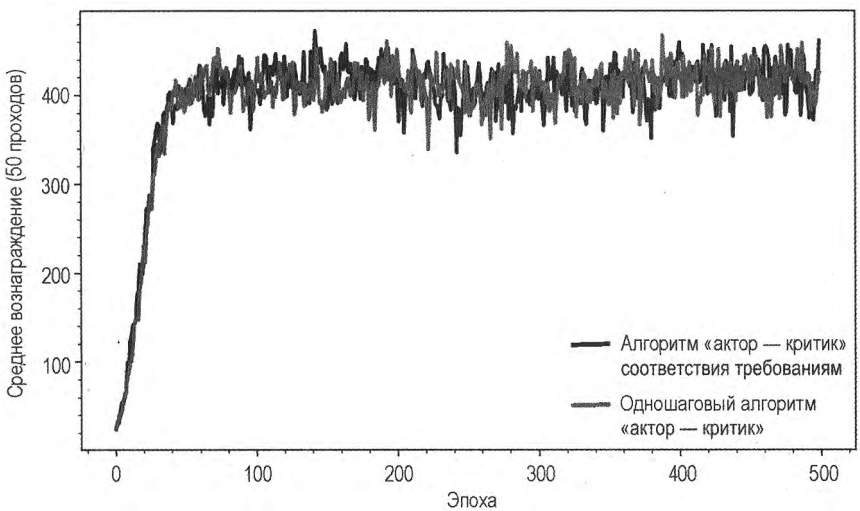



**Рис. 5.8.** Среднее вознаграждение для среды CartPole за одношаговый алгоритм "актор — критик" и алгоритм отслеживания соответствия требованиям

Производительность практически не изменилась. А значит, что успешный агент не требует предвидения в среде CartPole. Достаточно наблюдать за потенциальным падением на следующем этапе.

Изменение значения $\lambda$ имеет небольшой эффект, за исключением значений 0, где нет обновления параметров временно́го различия, и 1, где градиенты резко увеличиваются. В моих экспериментах все промежуточные значения привели к аналогичным результатам.

В средах, где вознаграждения нечасты или требуется значительный объем планирования, многоступенчатые оценки с учетом временно́го различия превосходны.

## Сравнение основных алгоритмов градиента политики

В этой главе представлены четыре различные реализации алгоритма градиента политики. Все используют градиентный спуск для итеративного улучшения политики, но у каждой есть собственные реализации оценки значения. Более точная оценка уменьшает количество излишне больших обновлений; это сужает разброс градиентов.

Различные варианты представлены в уравнении 5.11. Градиент политики равен ожиданию градиента параметров, умноженному на доходность. Единственное различие между этими алгоритмами — это способ количественной оценки доходности. Я считаю эту высокоуровневую интерпретацию алгоритмов градиента политики глубокомысленной, поскольку она проясняет тонкие различия между разными реализациями алгоритмов.

**Уравнение 5.11. Эквивалентные реализации алгоритма градиента политики**

$$\nabla_\theta J(\theta) \doteq \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \ln \pi_\theta(a|s) G\right] \quad \text{(REINFORCE)}$$

$$\doteq \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \ln \pi_\theta(a|s)\left(G - V_w(s)\right)\right] \quad \text{(REINFORCE с базовыми показателями)}$$

$$\doteq \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \ln \pi_\theta(a|s) Q_w(s,a)\right] \quad \text{(алгоритм "актор — критик")}$$

$$\doteq \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \ln \pi_\theta(a|s) A_w(s,a)\right] \quad \text{(алгоритм A2C)}$$

$$\doteq \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \ln \pi_\theta(a|s)\delta\right] \quad \text{(алгоритм "актор — критик" с учетом временны́х различий)}$$

$$\doteq \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \ln \pi_\theta(a|s)\delta(\lambda)\right] \quad \text{(приемлемый алгоритм "актор — критик")}$$

# Отраслевой пример: автоматическая продажа товаров клиентам

Не так давно можно было наблюдать за тенденцией распространения бизнес-моделей, которые предусматривали автоматическую доставку товаров клиентам и с условием возврата тех экземпляров, которые покупателям не интересны. Stitch Fix — один из самых известных бизнесов, основанных на *модели подписки*. После получения индивидуальных сведений о типе телосложения и личных предпочтениях клиенту Stitch Fix отправляется пакет новой одежды. Покупатель может вернуть любую одежду, которая ему не нравится, и оплатить ту, что подойдет (плюс накладные расходы за обслуживание).

Эта идея может показаться новинкой, но компании уже давно применяют модели подписки "без обязательств". Например, Hewlett Packard с радостью доставит чер-

нила для вашего принтера по заданной цене. Вы можете подписаться на определенные продукты на Amazon. Hello Fresh доставит вам еду. Разница между моделями без обязательств и подпиской заключается в том, что если клиент использует неправильный план (например, получает слишком много чернил), он, скорее всего, откажется от товара. В интересах HP поставлять чернила с приемлемой частотой, чтобы максимизировать клиенту пожизненную ценность товара.

Хочу подчеркнуть, насколько впечатляющей является эта задача. Здесь агент без вмешательства человека может автоматически отправлять вам товары и итеративно узнавать ваши предпочтения. Вы получите товары, когда они вам понадобятся, без предварительного заказа. Более того, агент знает, какие предметы вам могут понравиться, даже если вы никогда не думали о том, чтобы покупать их самостоятельно. С точки зрения рынка это яркая демонстрация того, почему обучение с подкреплением может быть таким прибыльным. Продавцы могут продать вам больше товаров, и вы будете искренне рады тому, что они вам предлагают!

## Рабочее окружение: корзина заказов, написанная при помощи библиотеки Gym

В этом примере реализована модель подписки с использованием общедоступных данных. Я создал среду[4], которая использует данные из Instacart для моделирования покупательского поведения.

Набор данных Instacart с открытым исходным кодом[5] содержит три миллиона заказов своей одноименной торговой платформы. Я выполнил значительную очистку данных, в результате чего получил набор данных, содержащий сведения об одном дне недели и часе дня, с быстрым кодированием. Оптимизированный набор данных также включает количество дней с момента последнего заказа, нормализованное до 1. Действия представляют собой список из 50 тыс. однократно упомянутых продуктов в двоичной кодировке. Для того чтобы заказать продукт, вы помечаете столбец продукта значением 1. Я предпочитаю простое вознаграждение. Вы получите вознаграждение в размере +1 за каждый продукт, который соответствует тому, что фактически заказал клиент. Штраф, равный –1, выписывается за каждый неправильно заказанный товар. Цель состоит в том, чтобы заказать то, что хочет клиент, и не более того. Каждый заказ — это шаг, а все заказы для одного покупателя — это эпоха.

Это упрощение того, что происходит в реальном мире. Фактическое вознаграждение вряд ли будет таким простым, как ±1. Предложение нового продукта покупателю может стимулировать к большему количеству покупок в будущем, тогда как ошибка может стоить дорого, если продукты скоропортящиеся и не могут быть перепроданы. Я ограничен возможностями, предлагаемыми в наборе данных. Другие функции, вероятно, будут более информативными. Например, в Stitch Fix есть

[4] См. **https://gitlab.com/winderresearch/gym-shopping-cart**.

[5] См. **https://oreil.ly/-g1ix**.

библиотека ваших предпочтений. Вы должны построить процесс, чтобы собирать предпочтения пользователей о том, какая еда им нравится.

Однако самая большая проблема заключается в том, что при моделировании используются статические данные. В идеале агент должен исследовать, какие именно продукты хочет приобрести покупатель. Заказчик может даже не знать об этом. Но имея фиксированные данные, вы оптимизируете процесс поиска статического решения. Другими словами, вы можете использовать стандартные методы машинного обучения. В основе RL лежит практическое исследование, а в этом примере его нет. RL — это больше чем просто предсказание.

Одним из возможных решений этой проблемы является моделирование покупок, совершаемых клиентами. Например, вы можете построить генеративно-состязательную сеть, чтобы моделировать клиентские покупки в динамичной среде. Затем можете перенести изученную политику в реальную среду (*см. разд. "Дополнительные материалы для чтения" в конце этой главы*). Для простоты я использую статические данные.

## Ожидания

Состояние и действия в этой среде являются двоичными значениями, поэтому вы можете использовать табличную технику, такую как Q-обучение. Но чтобы продолжить тему этой главы, я воспользуюсь алгоритмом градиента политики "актор — критик" (*см. разд. "Трассировки соответствия алгоритма „актор — критик"" ранее в данной главе*). Агент должен генерировать столько бинарных действий, сколько имеется продуктов. Я использую линейную политику с логистической функцией для каждого действия. Обратите внимание, что функция softmax не подходит, потому что она нормализует все действия, чтобы общее значение равнялось 1. Я также моделирую функцию ценности как линейную функцию. И то и другое делается для простоты. Вы можете использовать более сложные модели.

Позвольте мне ненадолго остановиться, чтобы выразить некоторые ожидания. Склонность к покупке продукта зависит от профиля клиента. Вы можете себе представить, что, если бы у клиента были дети, он с большей вероятностью купил бы детские товары. Если они веганы, то вряд ли они закажут мясо. Если они молоды, они, вероятно, будут покупать иные продукты, нежели пожилые клиенты. Все это скрытые черты. Их нельзя напрямую наблюдать по предпринимаемым ими действиям. По этой причине маловероятно, что линейная модель будет хорошо работать в реальных испытаниях. Линейная модель позволяет узнать, как часто клиенты покупают товары, не обращая внимания на то, почему они их покупают.

Упрощайте проблему, уменьшая количество скрытых функций. Создавайте явные особенности.

Таким образом, в реальном приложении дивиденды будут выплачиваться, если вы инвестируете в построение более сложной модели, которая представляет клиентов,

а не только продукты. Имея достаточно данных, вам также следует обратить внимание на глубокое обучение, чтобы представить эти высокоуровневые функции.

Другая проблема в том, что линейная модель не имеет памяти. Когда я покупаю стиральный порошок оптом в одном магазине, я редко покупаю его дополнительно в другом магазине. Исключение составляют случаи, когда дети решают вывозится в грязи. Но по линейным весам видно, что я купил много стирального порошка и, вероятно, алгоритм должен предложить мне докупить еще. Для того чтобы знать о закупках клиентами товаров по разным ценам, вам нужно кое-что вспомнить. Один из самых простых способов — включить в состояние предыдущие заказы. Вот почему, например, DQN так хорошо работает в среде Atari. Исследователи обнаружили, что четырех кадров пикселов было достаточно, чтобы обеспечить прокси для памяти. Более продвинутые решения включают использование нейронных сетей с отслеживанием состояния, таких как нейроны рекуррентной и долгой краткосрочной памяти (*см. разд. "Архитектуры нейронных сетей" главы 4*).

## Результаты из среды "Корзина покупок"

Опять же, для простоты я придерживаюсь линейной реализации. Для того чтобы уменьшить масштаб проблемы — 50 тыс. действий — я разрешаю агенту покупать только самые популярные продукты.

Первые результаты приведены на рис. 5.9. В этом первоначальном эксперименте я ограничил набор данных одним клиентом. Я решил оставить 15 самых популярных продуктов для этого клиента и обучил агента в течении 50 эпох. Вы также можете видеть деятельность агента, который случайным образом заказывает продукты. В этом примере у покупателя было 358 покупок популярных товаров и более 71 заказа. Следовательно, максимальная возможная награда — 358.

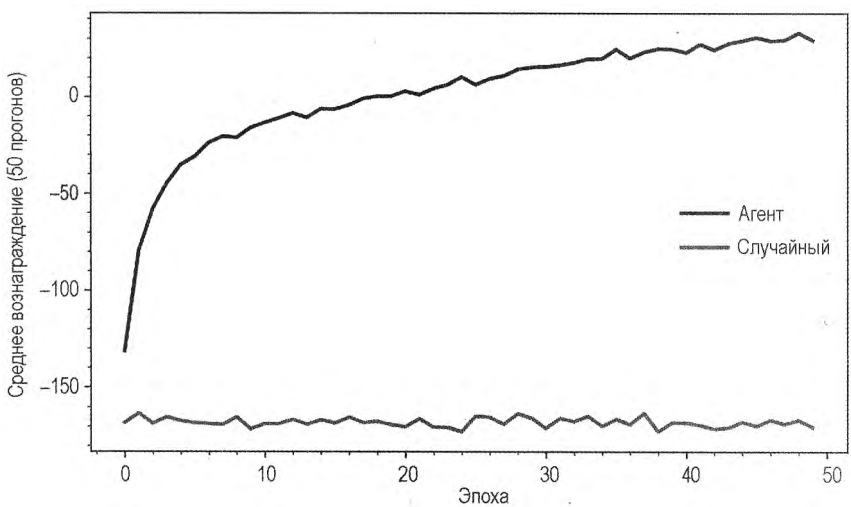



**Рис. 5.9.** Среднее вознаграждение для среды "Корзина покупок" для одного клиента (повторяется 50 раз) и для 15 самых популярных товаров

Самая высокая награда для агента составила 57, что указывает на то, что агент покупал продукты, которые действительно были нужны клиенту. Но это намного меньше, чем максимально возможное вознаграждение — 358 товаров. Случайный агент постоянно работает плохо, потому что он неоднократно заказывает продукты, которые не нужны клиенту.

На рис. 5.10 я повторяю тот же эксперимент, но на этот раз с 50 случайными покупателями, у которых разное количество заказов. Другими словами, агент не может переобучить одного клиента. Он должен пытаться обобщить данные для всех клиентов. На этот раз агент получает максимальное вознаграждение –1. Не особо впечатляет по сравнению с предыдущим экспериментом. Это говорит о том, что агент способен работать практически безубыточно с любым случайно заданным клиентом. Но вы должны помнить, насколько прост агент. У агента есть только несколько линейных параметров, и он в ограниченной степени способен уловить сложность процесса покупки. Я считаю это замечательным.

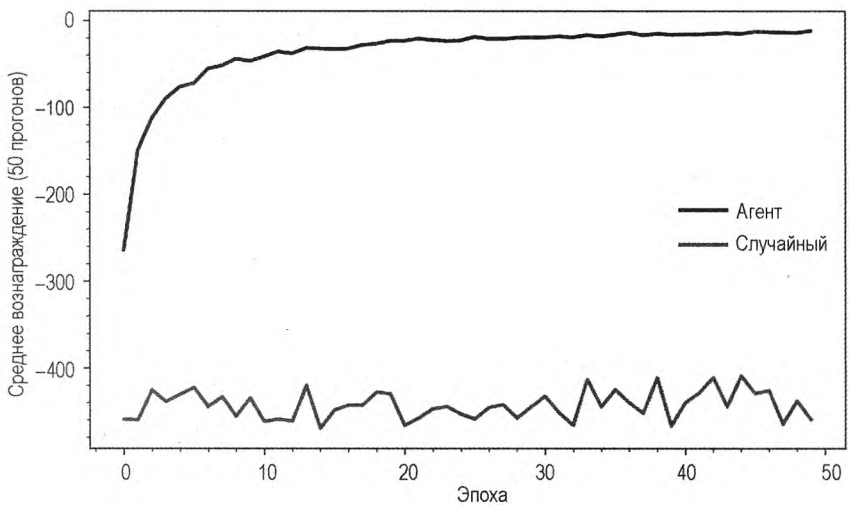



**Рис. 5.10.** Среднее вознаграждение для среды "Корзина покупок" для 50 случайных клиентов (50 эпох) и для 15 самых популярных продуктов. Наивысшая награда была –1

На рис. 5.11 показана производительность одного и того же агента по случайным клиентам при изменении количества продуктов, из которых агент может выбирать. Другими словами, я меняю количество действий. Когда количество товаров увеличивается, агент получает меньше вознаграждения. Однако оказывается, что производительность асимптотическая. Производительность может сходиться с увеличением времени обучения.

Наконец, я хочу продемонстрировать то, что называется *передающим обучением* (transfer learning). Цель состоит в том, чтобы обучить модель решать одну задачу, а затем применить ту же модель к другой, но связанной задаче. Эта парадигма очень хорошо работает с RL, потому что вы потенциально можете учиться на автономных данных или моделях и выполнять точную настройку, используя онлайн-данные.

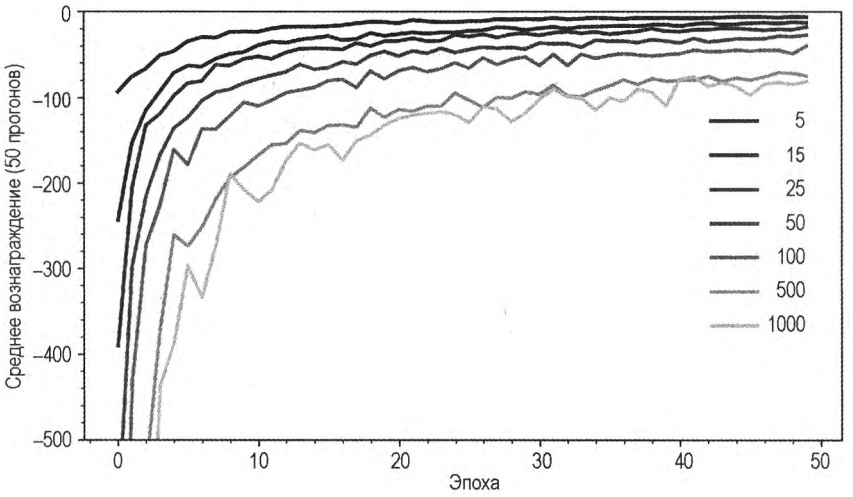



**Рис. 5.11.** Среднее вознаграждение для среды "Корзина покупок" для 50 случайных клиентов (50 эпох) при изменении количества самых популярных продуктов

Например, я могу использовать модель корзины покупок, обученную на случайных покупателях (см. рис 5.10), а затем передать изученную модель новому покупателю. На рис. 5.12 показаны результаты такого эксперимента.

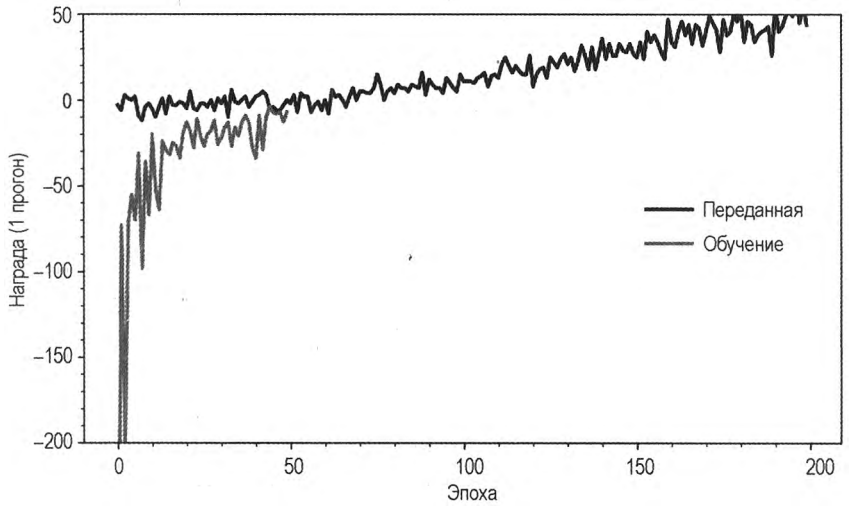



**Рис. 5.12.** Среднее вознаграждение для среды "Корзина покупок" для одного клиента после первоначального обучения на случайных данных клиента и передачи модели (15 продуктов, 1 запуск)

Чтобы перенести модель, я скопировал весовые коэффициенты политики и оценки, полученные при обучении случайных клиентов, и вставил их в другого агента только с одним клиентом. Кроме того, эти результаты относятся к одному запуску. Если бы я повторил это 50 раз, вы смогли бы увидеть более гладкие результаты.

Если вы сравните этот результат с результатом, показанным на рис. 5.9, то увидите, что начального повышения производительности нет, т. к. предварительно обученные веса заведомо очень хороши.

Другое наблюдение заключается в том, что одному агенту клиента требуется много времени (примерно 200 эпох), чтобы получить производительность, аналогичную показанной на рис. 5.9. Это связано с тем, что агент изучил политику, которая в среднем работает лучше всего для всех клиентов. Когда я передаю политику новому агенту для данных одного клиента, ему необходимо перенастроить эти веса. Но средняя политика, похоже, снизила вероятность изучения состояний, которые наиболее важны для этого отдельного клиента.

Другими словами, кажется, что на рис. 5.9 агент переобучен для этого конкретного клиента. Обычно переобучение не сулит ничего хорошего, поскольку модель не распространяется на других клиентов. Но в этом примере дело именно в этом. Я хочу, чтобы агент соответствовал предпочтениям отдельного клиента.

Здесь есть компромисс. Вы хотите начать с разумной политики, чтобы клиент быстро получал хорошие рекомендации. Но вам не нужно ждать 200 заказов, прежде чем они окажутся на территории положительного вознаграждения. Одно из решений этой проблемы — рассматривать количество эпох до обучения как гиперпараметр и настраивать его для получения удовлетворительного результата.

# Резюме

До этой главы я сосредоточивался на методах, основанных на ценностях, методах, которые количественно определяют ценность нахождения в состоянии. Политика выводится как максимизация ожидаемой доходности в каждом состоянии. Но другой способ — напрямую параметризировать политику и изменить веса параметризации, чтобы создать политику, которая максимизирует ожидаемую доходность. Вы можете использовать методы градиента политики, чтобы найти обновления, необходимые для оптимизации политики.

У методов градиента политики есть ряд преимуществ. Они могут научиться выводить вероятности, что устраняет необходимость в явном алгоритме исследования, таком как ε-жадный алгоритм. Они могут выводить непрерывные действия. Иногда проще смоделировать политику, чем моделировать пару "состояние — действие".

С другой стороны, методы градиента политики более чувствительны к выбору модели. С помощью методов значения довольно легко изменить сложность модели, чтобы спрогнозировать функцию ожидаемой ценности, тогда как модель политики необходимо выбирать осторожно.

Материал этой главы излагался по тому же принципу, что и раньше: от методов Монте-Карло до трассировок соответствия. Несмотря на то что методы, основанные на политике, оптимизируются для немного другого решения, оценки ожидаемого значения по-прежнему невероятно важны, как вы видели в алгоритмах "актор — критик".

Я привел упрощенный практический пример, цель которого в том, чтобы автоматически заказывать продукты для клиентов. Вы можете назвать большое количество решений, которые можно улучшить, например работу рекомендательных движков, активно исследуемых в сфере RL. Я знаю несколько организаций, которые используют RL для подбора рекомендаций.

## Дополнительные материалы для чтения

- Градиентный спуск.
  - Ruder S. An Overview of Gradient Descent Optimization Algorithms // ArXiv:1609.04747. — 2017. — June. — URL: **https://oreil.ly/iAYEs**.
- Вывод градиента политики.
  - Краткое математическое введение в градиенты политики: Kämmerer M. M. On Policy Gradients // ArXiv:1911.04817. — 2019. — November. — URL: **https://oreil.ly/1EAUY**.
  - Sutton R. S., Barto A. G. Reinforcement Learning: An Introduction. — MIT Press, 2018.
- Построение генеративных моделей.
  - Foster D. Generative Deep Learning: Teaching Machines to Paint, Write, Compose, and Play. — O'Reilly, 2019.

## Использованные источники

[1] Houthooft R., Chen R. Y., Isola P. et al. Evolved policy gradients // ArXiv:1802.04821. — 2018. — April. — URL: **https://oreil.ly/IQPmY**.

[2] Bhatnagar S., Ghavamzadeh M., Lee M., Sutton R. S. Incremental natural actor-critic algorithms // Advances in Neural Information Processing Systems 20 / ed. by J. C. Platt, D. Koller, Y. Singer, S. T. Roweis. — Curran Associates, Inc., 2008. — P. 105–112. — URL: **https://oreil.ly/9UYvm**.

[3] Wolpert D. H., Macready W. G. No Free lunch theorems for optimization // IEEE Transactions on Evolutionary Computation. — 1997. — Vol. 1, № 1. — P. 67–82.

ГЛАВА 6

# Другие методы

Алгоритмы градиента политики (policy gradient, PG) широко используются в задачах обучения с подкреплением с непрерывными пространствами действий. В *главе 5* изложена основная идея: представить политику параметрическим распределением вероятностей $\pi_\theta \doteq \mathbb{P}[a \mid s; \theta]$, затем скорректировать параметры политики $\theta$ в направлении большего совокупного вознаграждения. Это широко известный метод, потому что альтернатива, такая как жадная максимизация функции ценности действия, выполняемая в Q-обучении, становится проблематичной, если действия являются непрерывными, поскольку это повлечет за собой глобальную максимизацию бесконечного числа действий. Сдвиг параметров политики в сторону более высокой ожидаемой доходности может быть более простым и выгодным с точки зрения вычислений.

До недавнего времени практические онлайн-алгоритмы градиента политики с гарантиями сходимости были ограничены настройкой *политики*, в которой агент учится на основе выполняемой политики, политики *поведения*. В настройках *вне политики* агент изучает политики, отличные от политики поведения. Эта идея подразумевает новые исследовательские возможности и приложение, например обучение из автономного журнала, изучение оптимальной политики при выполнении исследовательской политики, обучение на демонстрации и параллельное изучение нескольких задач в единой среде.

В этой главе исследуются современные действующие вне политик алгоритмы PG, начиная с подробного рассмотрения некоторых проблем, возникающих при попытке обучения вне политики.

## Алгоритмы, действующие вне политик

Алгоритмы PG, представленные в *главе 5*, соответствовали политике; вы генерируете новые наблюдения, следуя текущей политике — той же, которую хотите улучшить. Вы уже видели, что такой подход, являющийся петлей обратной связи, может привести к чрезмерно оптимистичным ожидаемым доходам (*см. разд. "Двойное Q-обучение" главы 3*). Алгоритмы вне политики разделяют исследование и использование с помощью двух политик: целевой политики и политики поведения.

Все алгоритмы RL, представленные до этого, выполняли оптимизацию типа наименьших квадратов, когда оценки приближенных значений сравниваются с наблю-

даемым результатом; если вы наблюдаете неожидаемый результат, то соответственно обновляете оценку. В настройках вне политики целевая политика никогда не имеет возможности проверить свою оценку. Тогда как же можно выполнить оптимизацию?

## Выборка по значимости

Прежде чем продолжить, хочу, чтобы вы поняли, что называется выборкой по значимости, которая является результатом некоторых математических уловок, позволяющих оценить интересующую величину без непосредственной выборки.

Я уже несколько раз упоминал, что математическое ожидание случайной величины $X$ — это интеграл отдельного значения $x$, умноженный на вероятность получения этого значения $f(x)$ для всех возможных значений $x$. В непрерывном распределении вероятностей это означает $\mathbb{E}_f[x] \doteq \int_x xf(x)$. В дискретном случае это $\mathbb{E}_f[x] \doteq \sum_x xf(x)$. Это называется *средним по совокупности.*

Но что, если вы не знаете вероятностей? Что ж, вы все равно могли бы оценить среднее значение, если бы у вас была возможность взять образец из этой среды. Например, рассмотрим кубик. Вы не знаете вероятность выпадения какой-либо конкретной грани, но знаете, что существует шесть значений. Вы можете оценить ожидаемое значение, поворачивая матрицу и вычисляя *выборочное среднее*. Другими словами, вы можете оценить математическое ожидание, используя формулу $\mathbb{E}_f[x] \approx \frac{1}{n}\sum_{i=1}^{n} x_i$.

Теперь представьте, что у вас есть второй экземпляр случайной величины, например второй кубик, который привел к тем же значениям, и вы знаете вероятность этих значений, но вы не можете сделать выборку из этого экземпляра. Не очень правдоподобно, но попрошу проявить снисходительность к моим суждениям, т. к. в этом суть проблемы. Допустим, вы знали, сколько весит второй кубик, $g(x)$, но не могли его бросить. Может быть, он принадлежит печально известному другу, который, бросая кубик, получает слишком много шестерок. В этом случае вы можете сформулировать ожидание таким же образом: $\mathbb{E}_g(x) \doteq \sum_x xg(x)$. Это показано на шагах 1 и 2 уравнения 6.1.

**Уравнение 6.1. Выборка по значимости**

$$\mathbb{E}_f[x] \doteq \sum_x xf(x) \qquad (1)$$

$$\mathbb{E}_g[x] \doteq \sum_x xg(x) = \qquad (2)$$

$$= \sum_x \frac{xg(x)}{f(x)} f(x) = \qquad (3)$$

$$= \mathbb{E}_f \left[ \frac{xg(x)}{f(x)} \right] \approx \qquad (4)$$

$$\approx \frac{1}{n} \sum_{i=1}^{n} \frac{x_i g(x_i)}{f(x_i)}. \qquad (5)$$

Теперь посмотрите на шаг 3. И снова математическая уловка: я умножаю шаг 2 на $f(x)/f(x) = 1$. Теперь сумма рассчитывается относительно $f(x)$, а не $g(x)$, и это означает, что я могу заменить шаг 1, если вы представите, что $x$ теперь $xg(x)/f(x) = 1$, что приводит к шагу 4. На шаге 5 вы можете вычислить это ожидание, используя выборочное среднее. Если вы сконцентрируетесь, то, вероятно, сможете следовать логике, но я хочу, чтобы вы понимали, что это означает на высоком уровне. Это значит, что вы можете рассчитать математическое ожидание второй функции вероятности, используя опыт первой. Это важный элемент обучения вне политики.

Для того чтобы продемонстрировать идею, лежащую в основе уравнения 6.1, я смоделировал бросание двух игральных кубиков: одного правильного, другого — шулерского. Я знаю вероятностное распределение игральных кубиков, но могу бросить только правильный кубик. Я могу записать числа, выпавшие в результате бросания правильного кубика, и построить гистограмму выпадений. Я также могу измерить выборочное среднее после каждого броска. Результаты представлены на рис. 6.1, *а*. При достаточном количестве бросков гистограмма становится более

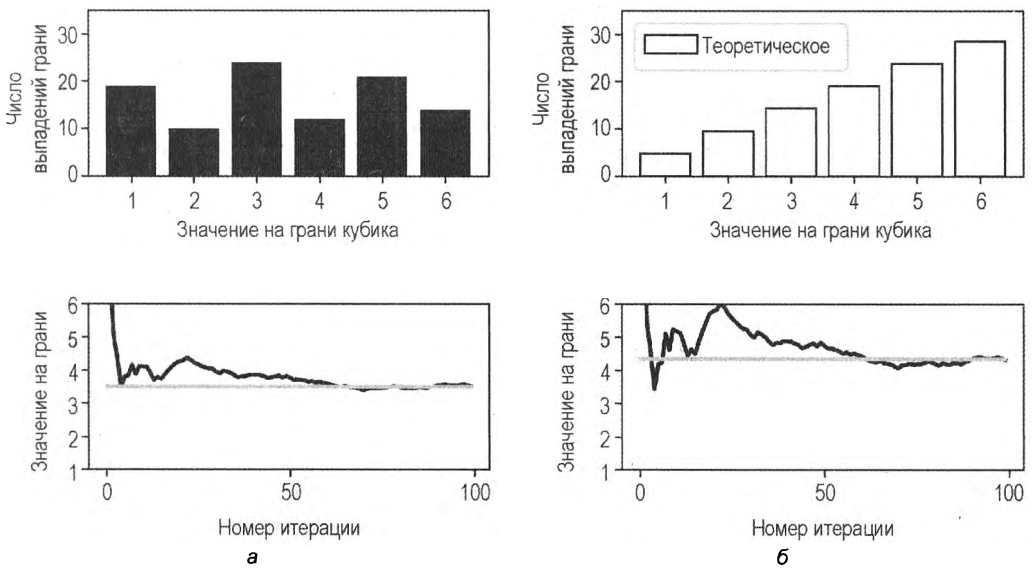



**Рис. 6.1.** Использование выборки по значимости для расчета ожидаемого выпавшего значения на грани кубика, который вы не можете бросить: *а* — правильный кубик; *б* — шулерская кость

однородной, а среднее значение выборки приближается к среднему значению по генеральной совокупности, равному 3,5.

На рис. 6.1, *б* вы можете увидеть (решающую) известную теоретическую гистограмму бросков шулерского кубика. Это теоретическое предположение, потому что я не бросал кубик. И вот в чем волшебство: используя уравнение 6.1, я могу оценить выборочное среднее, и вы можете увидеть, что оно стремится к 4,3, правильному выборочному среднему значению для шулерского кубика. Позвольте мне повторить это еще раз, чтобы всё стало достаточно ясно. Вы не можете просто вычислить среднее значение второй переменной, потому что не можете использовать ее для выборки. Выборка по значимости позволяет оценить ожидаемое значение второй переменной, не делая выборку для нее.

## Поведенческие и целевые политики

Если у вас есть две политики, одну из которых вы можете реализовать, а другую — нет, вы все равно можете получить ожидаемое значение для обеих. Ситуация равносильна задаче научиться водить машину, наблюдая за тем, как кто-то водит машину. Этот расчет открывает дверь для настоящих алгоритмов, действующих вне политики, когда одна политика побуждает агента к исследованию, а другая или другие могут делать что-то еще, например разрабатывать более совершенную политику.

В RL термин "обучение вне политики" относится к изучению оптимальной политики, называемой *целевой политикой*, с использованием данных, генерируемых *поведенческой политикой*. Освобождение поведенческой политики от бремени поиска оптимальной политики дает возможность более гибкого исследования. Я развиваю эту идею на протяжении всей главы.

Целевая политика снабжена информацией, например цыплятам дают пищу; им все равно, откуда она берется. Данные могут поступать от других агентов, таких как, например, предыдущие политики или люди. Или в средах, где сложно либо дорого получить новые данные, но легко хранить старые, например в робототехнике, данные могут быть из автономного хранилища.

Еще одно интересное направление исследований, к которому я вернусь позже, заключается в том, что вы можете изучить несколько целевых политик из одного потока данных, генерируемых одной поведенческой политикой. Это позволяет изучить оптимальные политики для различных подцелей (*см. разд. "Иерархическое обучение с подкреплением" главы 8*).

## Q-обучение, действующее вне политики

Одним из ключей к обучению вне политики является использование приближений функции ценности действия и приближений политики. Но традиционные методы вне политики, такие как Q-обучение, не могут сразу воспользоваться преимуществами линейных приближений, потому что они нестабильны; приближения расходятся до бесконечности при любом положительном размере шага. Но в 1999 г.

Ричард Саттон и его команда впервые показали, что стабильность возможна, если вы используете правильную целевую функцию и градиентный спуск [1]. Эти методы легли в основу усовершенствованных алгоритмов вне политики, которые вы увидите позже.

## Градиентное обучение с учетом временных различий

Во многих задачах число состояний слишком велико, чтобы их можно было приблизительно оценить, например в случае существования непрерывных состояний. Приближение линейной функции отображает состояния в векторы признаков с помощью функции $\varphi(s)$. Другими словами, эта функция принимает состояния и выводит абстрактный набор значений с другим числом измерений.

Затем вы можете аппроксимировать функцию ценности состояния, используя линейные параметры $\theta$, как $V\pi(s) \approx \theta^T \varphi(s)$ (более подробную информацию о разработке политики *см. в разд. "Разработка политики" главы 9*). Учитывая это определение, вы можете вывести новое уравнение для TD-ошибки, подобное уравнению 6.2.

**Уравнение 6.2. Параметризованная линейная TD-ошибка**

$$\delta \doteq r + \gamma\theta^T \varphi(s') - \theta^T \varphi(s).$$

Это то же уравнение, что и уравнение 3.3, в котором линейное предсказание является явным. Обратите внимание, что в литературе $\varphi(s)$ часто сокращается до $\varphi$. Требуются некоторые шаги, необходимые для изменения этого уравнения для решения $\theta$, но решение удивительно простое: используйте ошибку TD в качестве прокси для определения того, насколько неверны текущие параметры, найдите градиент, измените параметры в сторону более низкой ошибки и повторяйте до тех пор, пока все не сойдется.

Если вас интересует вывод, то обратитесь к приведенной статье [2]. Полученный алгоритм выглядит как алгоритм 6.1.

**Алгоритм 6.1. Алгоритм GTD(0)[1]**

1: **input**: политика $\pi(a|Q(s,a;\theta))$, которая использует параметризованную функцию ценности действия; параметры скрости обучения $0<\alpha<1$ и $0<\beta<1$.

2: Инициализировать $\theta$ и $u$ произвольно.

3: **loop** для каждого эпизода

4: Инициализировать $s$.

[1] GDT — gradient-temporal-difference (градиентно-временно́е различие).

5: **while** $s$ не является терминальным состоянием:

6: Выбрать действие $a$ из состояния $s$, используя политику $\pi$ (неоднозначности разрешить случайным образом).

7: Выполнить действие $a$, а затем наблюдать за наградой $r$ и следующим состоянием $s'$.

8: $\delta \leftarrow r + \gamma\theta^T\boldsymbol{\varphi}(s') - \theta^T\boldsymbol{\varphi}(s)$.

9: $u \leftarrow u + \beta(\delta\boldsymbol{\varphi}(s) - u)$.

10: $\theta \leftarrow \theta + \alpha(\boldsymbol{\varphi}(s) - \gamma\boldsymbol{\varphi}(s'))\boldsymbol{\varphi}^T(s)u$.

11: $s \leftarrow s'$.

Общая схема аналогична стандартному Q-обучению. Шаги 9 и 10 вычисляют градиент ошибки TD и результирующее правило обновления. Если бы у вас хватило упорства самостоятельно разобраться в математике, это было бы замечательно. $u$ — промежуточная рассчитываемая величина для обновления параметров $\theta$.

Следует отметить, что этот алгоритм не использует выборку по значимости, которая теоретически обеспечивает более низкие обновления дисперсии. В статье [3] и в последующем обновлении (GTD2) Саттон и соавт. вводят другой весовой коэффициент для коррекции смещений в обновлениях градиента.

## Жадный GQ-алгоритм

Жадный GQ-алгоритм, который вводит вторую линейную модель для целевой политики, в отличие от единственного поведения и целевой политики в GTD. Преимущество подгонки второй модели для целевой политики состоит в том, что эта политика дает больше свободы. Целевая политика больше не ограничивается потенциально неподходящей моделью поведения. Теоретически это обеспечивает потенциально более быстрое обучение. Реализация такая же, как и в алгоритме 6.1, поэтому я обновляю лишь последние этапы, как показано в алгоритме 6.2. Здесь $\mathbf{w}$ — веса целевой политики.

**Алгоритм 6.2. Жадный GQ-алгоритм**

...

9: $\mathbf{w} \leftarrow \mathbf{w} + \beta\left[\boldsymbol{\varphi}(s') - \boldsymbol{\varphi}^T(s)\mathbf{w}\right]\boldsymbol{\varphi}(s)$.

10: $\theta \leftarrow \theta + \alpha\left[\delta\boldsymbol{\varphi}(s) - \gamma\left(\mathbf{w}^T\boldsymbol{\varphi}(s)\right)\boldsymbol{\varphi}(s')\right]$.

11: $s \leftarrow s'$.

Вы также можете расширить этот и другие производные от TD-алгоритмов с помощью трассировок соответствия [5].

## Алгоритм "актор — критик" вне политики

*Алгоритмы градиентно-временных различий* (gradient-temporal-difference, GTD) имеют линейную сложность и сходятся даже при использовании аппроксимации функций. Но они страдают тремя важными ограничениями. Во-первых, целевая политика является детерминированной, что становится проблемой при использовании мультимодальных или стохастических ценностных функций. Во-вторых, использование жадной стратегии становится проблематичным для больших или непрерывных пространств действий. И в-третьих, незначительное изменение в оценке ценности действия может вызвать большие изменения в политике. Вы можете решить эти проблемы, используя комбинацию методов, описанных ранее. Но Дегрис и соавт. скомбинировали GTD с градиентами политик для решения этих проблем за один раз [6].

Алгоритм REINFORCE и упрощенные алгоритмы "актор — критик" из *главы 5* используют политику. Алгоритм "актор — критик" вне политики (off-policy actor-critic, off-PAC) был одним из первых алгоритмов, доказавших, что можно использовать параметризованного критика. Если вы помните, вам нужно вычислить градиент политики, чтобы использовать алгоритмы градиента политики. Добавление параметризованных (или табличных) критиков увеличивает сложность производной. Дегрис и соавт. доказали, что, если вы аппроксимируете градиент политики с помощью оптимизатора на основе градиентного спуска и проигнорируете градиент критика, политика в итоге все равно достигнет оптимума.

В результате получается алгоритм, практически идентичный алгоритму 5.3. Он представлен в уравнении 6.3, где основным добавлением является выборка по значимости $\pi_\theta(a|s)/\beta_\theta(a|s)$. $\beta$ представляет политику поведения, $\pi$ — цель. Думайте о выборке по значимости как о коэффициенте преобразования. Агент выполняет выборку из политики поведения, поэтому ему необходимо преобразовать этот опыт, чтобы использовать его в целевой политике. Я также включил специальные обозначения, чтобы показать, что состояния распределяются в соответствии с траекториями, генерируемыми политикой поведения ($s \sim \rho_\beta$, где $\rho$ — распределение состояний), а действия распределяются в зависимости от выбора, сделанного поведенческой политикой ($\alpha \sim \beta$).

**Уравнение 6.3. Оценка алгоритма "актор — критик" вне политики**

$$\nabla_\theta J_\beta(\pi_\theta) = \mathbb{E}_{s\sim\rho_\beta, a\sim\beta}\left[\frac{\pi_\theta(a|s)}{\beta_\theta(a|s)}\nabla_\theta \ln \pi_\theta(a|s) Q_\pi(a|s)\right].$$

Таким образом, уравнение 6.3 вычисляет градиент целевой политики, взвешенный по оценке ценности действия и распределяемый политикой поведения. Этот градиент сообщает вам, насколько и в каком направлении обновлять целевую политику. В этой работе все выглядит более сложным, поскольку используются трассировки соответствия и учитывается градиент параметризованной линейной функции для обеих политик.

Дегрис и соавт. сделали небольшое, но важное теоретическое замечание о значении частоты обновления для каждой политики. Они предлагают для актора частоту обновления делать меньше, а отжиг устремлять к нулю быстрее, нежели для критика. Другими словами, важно, чтобы поведенческая политика была более стабильной, чем целевая политика, поскольку она создает траектории. На механическом быке сложно ездить, потому что он движется в случайных направлениях. Ездить на лошади проще, потому что у нее предсказуемые траектории. Как инженер, вы должны максимально упростить для цели аппроксимацию функции ценности действия.

# Детерминированные градиенты политики

В этом разделе я вернусь назад, чтобы обсудить еще одно направление исследований, которое вращается вокруг идеи о том, что политика не обязательно должна быть стохастической. Одна из основных проблем всех PG-алгоритмов заключается в том, что агент должен выполнять выборку как в пространстве состояний, так и в пространстве действий. Только тогда вы можете быть уверены, что агент нашел оптимальную политику. Вместо этого возможно ли разработать детерминированную политику — одно конкретное действие для каждого состояния? Если это удастся сделать, то можно резко сократить количество выборок, необходимых для оценки такой политики. Если агенту повезет, за одну попытку можно будет найти оптимальную политику. Теоретически это должно привести к более быстрому обучению.

## Обычные детерминированные градиенты политики

Алгоритмы градиента политики широко используются в задачах с пространствами непрерывных действий, представляя политику как распределение вероятностей, которое случайным образом выбирает действие в состоянии. Распределение параметризовано таким образом, что агент может улучшить результат политики с помощью стохастического градиентного спуска.

Алгоритмы градиента, действующие вне политики, такие как off-PAC (*см. разд. "Алгоритм „актор — критик" вне политики" ранее в этой главе*), являются многообещающими, потому что они позволяют актору сосредоточиться на исследовании, а критик может обучаться в автономном режиме. Но актор, использующий политику поведения, все равно должен максимизировать все действия, чтобы выбрать лучшую траекторию, а это может стать проблематичным с вычислительной точки зрения. Кроме того, агент должен выполнить выборку распределения действий в достаточном объеме, чтобы произвести оценку градиентного спуска.

Вместо этого Сильвер и соавт. предложили сместить акцент в поведенческой политике в сторону максимальной оценки ценности действия, а не напрямую максимизировать ее. По сути, они использовали тот же прием, что и PG-алгоритмы, чтобы подтолкнуть целевую политику к более высокому ожидаемому результату, но на этот раз они применяют этот подход к поведенческой политике. Таким образом, оптимальное действие становится простым вычислением с использованием теку-

щей политики поведения. При такой же параметризации процесс детерминирован. Этот прием исследователи назвали алгоритмом *детерминированного градиента политики* (deterministic policy gradients, DPG) [7].

Исследователи доказывают, что DPG — это ограниченная версия стандартного стохастического алгоритма PG, поэтому все обычные инструменты и методы, такие как аппроксимация функций, естественные градиенты и архитектуры "актор — критик", работают точно так же.

Однако DPG не дает вероятностей действия и, следовательно, не имеет естественного механизма исследования. Представьте себе агента на основе DPG, который только что добился успеха. Последующие запуски будут производить ту же траекторию, потому что политика поведения генерирует одни и те же действия снова и снова.

В результате получился знакомый набор алгоритмов, но расчет градиента немного изменился. Для обычных алгоритмов PG, таких как REINFORCE или "актор — критик", цель состоит в том, чтобы найти градиент целевой функции. В случае DPG цель включает детерминированную функцию, а не стохастическую политику.

В уравнении 6.4 представлены градиенты целевых функций для упрощенных и детерминированных алгоритмов "актор — критик" вне политики. Я использовал ту же терминологию, что и в работе о DPG, поэтому здесь $\mu$ представляет собой детерминированную политику. Одно из основных отличий заключается в том, что вам не нужно использовать выборку по значимости для корректировки предвзятости, вносимой поведенческой политикой, поскольку нет распределения действий. Еще одно отличие следует из детерминизма: вам больше не нужно рассчитывать оценку ценности действия глобально по всем действиям.

**Уравнение 6.4. Градиенты целевых функций для упрощенных и детерминированных алгоритмов "актор — критик" вне политики (для сравнения)**

$$\nabla_\theta J_\beta(\pi_\theta) = \mathbb{E}_{s\sim\rho_\beta,\, a\sim\beta}\left[\frac{\pi_\theta(a|s)}{\beta_\theta(a|s)}\nabla_\theta \ln \pi_\theta(a|s) Q_\pi(a|s)\right]$$

алгоритм "актор — критик" вне политики

$$\nabla_\theta J_\beta(\mu_\theta) = \mathbb{E}_{s\sim\rho_\beta}\left[\nabla_\theta Q_\mu(s, \mu_\theta(s))\right] = \mathbb{E}_{s\sim\rho_\beta}\left[\nabla_\theta \mu_\theta(s) \nabla_a Q_\mu(s, a)\right]$$

алгоритм "актор — критик" вне политики (детерминированный)

В оставшейся части работы Сильвера и соавт. обсуждаются обычные проблемы расхождения Q-обучения с использованием параметризованных функций и несовершенного градиента функции ценности действия. В итоге Сильвер и соавт. получают алгоритм GTD в критике и алгоритм PG в акторе, и все это выглядит удивительно похоже на алгоритм off-PAC без выборки по значимости и трассировки соответствия.

Производные по-прежнему зависят от параметризации актора и критика. При желании вы можете выполнить аналогичную процедуру, описанную в *главе 5*, чтобы

вычислить их вручную. Но большинство приложений, которые я вижу, сразу переходят к использованию глубокого обучения отчасти потому, что они достаточно сложны для моделирования нелинейностей, но главным образом потому, что эти фреймворки будут выполнять автоматическое дифференцирование с помощью обратного распространения.

Таким образом, вам не нужно реализовывать каждое действие отдельно, чтобы разработать хорошую политику, как это было сделано с помощью Q-обучения в *главе 3*. И, вероятно, это ускорит обучение. Вы по-прежнему можете использовать градиенты политик для решения проблем с большим пространством действий. Но, конечно, некоторые проблемы имеют очень сложные функции ценности действия, которые выигрывают от стохастической политики. В общем, даже если стохастические политики учатся медленнее, они могут повысить производительность. Все зависит от окружающей среды.

## Глубокие детерминированные градиенты политики

Логическим продолжением алгоритмов DPG, действующих вне политики, является внедрение глубокого обучения (deep learning, DL). Как и ожидалось, *глубокие детерминированные градиенты политики* (deep deterministic policy gradients, DDPG) следуют пути, заданному глубокой Q-сетью.

Однако я не хочу преуменьшать важность этого алгоритма. DDPG был и остается одним из самых важных алгоритмов в RL. Его популярность связана с тем, что он может обрабатывать как сложные многомерные пространства состояний, так и многомерные пространства непрерывного действия.

Кроме того, нейронные сети, которые представляют актора и критика в DDPG, являются полезной абстракцией. Рассмотренные алгоритмы усложены математическими выкладками, потому что исследователи добавили в алгоритмы вычисление градиента. DDPG переносит оптимизацию актора/критика в структуру глбокого обучения. Это значительно упрощает архитектуру, алгоритм и конечную реализацию. Гибкость обусловлена возможностью изменять DL-модели для лучшего соответствия марковским процессам принятия решений. Хотя стоит отметить, что для решения большинства задач вам не понадобятся компоненты RL.

Конечно же, использование DL создает новые проблемы. Вам нужно быть не только экспертом в области RL, но и экспертом в области DL. Здесь также проявляются все распространенные подводные камни DL. Сложные модели DL добавляют значительную вычислительную нагрузку, и многие приложения тратят на них больше времени и денег, чем на базовый алгоритм RL или определение марковского процесса принятия решений. Несмотря на эти проблемы, я все же рекомендую DDPG с учетом архитектурных и эксплуатационных преимуществ.

### Вывод DDPG

В 2015 г. Лилликрап и соавт. представили DDPG с использованием того же вывода, что и DPG. Исследователи обучали модель предсказывать действие напрямую, без

необходимости максимизировать функцию ценности действия. Также они с самого начала работали с версией DPG вне политики [8].

Новинка заключается в использовании DL как для актора, так и для критика, а также в том, как авторы решали проблемы, вызванные нелинейными аппроксиматорами. Во-первых, решалась проблема сходимости. Как и DQN, исследователи ввели буфер воспроизведения для декорреляции наблюдений за состоянием. Они также использовали копию весов сети акторов и критиков (*см. разд. "Двойное Q-обучение" главы 3*) и медленно обновляли их для повышения стабильности.

Вторым дополнением была пакетная нормализация, которая нормализует каждую функцию с использованием скользящего среднего и дисперсии. Если этого не сделать, вы должны найти иной способ убедиться, что входные объекты правильно масштабированы.

Третье и последнее дополнение — это рекомендация использовать шум при прогнозировании действия, чтобы стимулировать исследовательскую деятельность. Постулируется, что вероятностное распределение шума зависит от задачи, но на практике обычно хорошо работает и простой гауссовский шум. Кроме того, добавление шума к параметрам нейронной сети — более действенный шаг, чем добавление шума к действиям (также *см. разд. "Зашумленные сети" главы 4*).

## Внедрение DSP

Все фреймворки RL имеют реализацию DDPG. И, как вы уже догадались, все они разные. Не ждите одинакового результата при разных реализациях. Различия варьируются от инициализации нейронной сети до гиперпараметров, заданных по умолчанию.

На рис. 6.2 представлена упрощенная архитектура. Как обычно, среда передает агенту наблюдение за состоянием. Актор принимает решение о следующем действии в соответствии со своей детерминированной политикой. Кортеж из состояния, действия, награды и следующего состояния $(s, a, r, s')$ также хранится в буфере воспроизведения. Поскольку DDPG не соответствует политике, он выбирает кортежи из буфера воспроизведения, чтобы измерить ошибку и обновить веса критика.

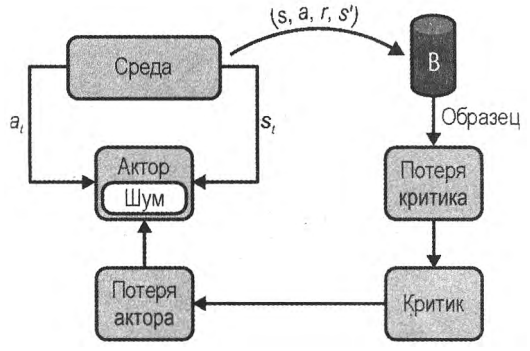



**Рис. 6.2.** Упрощенное описание алгоритма DDPG

Наконец, актор использует оценки критика, чтобы измерить свою ошибку и обновить свои веса. Алгоритм 6.3 представляет собой полную версию и немного отличается от академической реализации алгоритма, определенного в исходной статье. В этой версии используются мои обозначения, поэтому она лучше соответствует типичной реализации.

Алгоритм 6.3 обеспечивает полную реализацию DDPG.

**Алгоритм 6.3. Алгоритм DDPG**

1: **input**: актор нейронной сети $\mu_{\theta_\mu}(s)$ с весами $\theta_\mu$,
критик нейронной сети $Q_{\theta_Q}(s,a)$ с весами $\theta_Q$,
буфер воспроизведения $B$,
размер тренировочного мини-пакета $N$,
скорость обновления сети $\tau$,
коэффициент дисконтирования $\gamma$,
функция шума для исследования $\mathcal{N}$.

2: Инициализировать $\theta_\mu$, $\theta_Q$, $B$, а также целевые веса $\theta'_\mu \leftarrow \theta_\mu$, $\theta'_Q \leftarrow \theta_Q$.

3: **loop**: для каждого эпизода

4: Инициализировать $s$.

5: **while** $s$ не является терминальным состоянием:

6: $a \leftarrow \mu_{\theta_\mu}(s) + \mathcal{N}$.

7: Выполнить действие $a$, а затем наблюдать за наградой $r$ и следующим состоянием $s'$.

8: Сохранить кортеж $(s, a, r, s')$ в буфере воспроизведения $B$.

9: Выбрать случайно мини-пакет из $N$ кортежей $(s_i, a_i, r_i, s'_i)$ из буфера $B$.

10: $y_i \leftarrow r_i + \gamma Q_{\theta'_Q}\left(s'_i, \mu_{\theta'_\mu}(s'_i)\right)$ (обратите внимание на целевые веса и данные буфера).

11: Обновить критика, минимизируя $\frac{1}{N}\sum_i \left(y_i - Q_{\theta_Q}(s_i, a_i)\right)^2$.

12: Обновить актора, минимизируя $\frac{1}{N}\sum_i \left(\nabla_{\theta_Q} Q_{\theta_Q}\left(s_i, \mu_{\theta_\mu}(s_i)\right)\right)^2$.

13: $\theta'_Q \leftarrow \tau\theta_Q + (1-\tau)\theta'_Q$.

14: $\theta'_\mu \leftarrow \tau\theta_\mu + (1-\tau)\theta'_\mu$.

15: $s \leftarrow s'$.

Для того чтобы использовать алгоритм 6.3 в приложениях, вам сначала нужно определить нейронную архитектуру, которая соответствует вашей задаче. Понадобятся две сети: одна для актора (политика), а другая для критика (аппроксимация функции ценности действия). Например, для необработанных пикселов подойдет сверточная нейронная сеть в критике. Для информации о клиентах годится многоуровневая сеть прямой связи. Что касается политики, помните, что DDPG является детерминированным алгоритмом, поэтому для каждого состояния нужно прогнозировать одно действие. Следовательно, количество выходов зависит от вашего пространства действий. Также очень важны гиперпараметры сети. Емкости сети должно хватить, чтобы приблизиться к сложности вашей предметной области, но не допускайте переобучения. Скорость обучения напрямую влияет на стабильность обучения. Если вы попытаетесь обучить вашу сеть слишком быстро, можете обнаружить, что агент не сходится. Остальные входные данные на шаге 1 алгоритма 6.3 состоят из гиперпараметров, управляющих обучением, и функции шума, которая позволяет проводить исследования.

На шагах 2 и 3 инициализируются веса сети и буфер воспроизведения. Буфер воспроизведения должен хранить новые кортежи и удалять кортежи с историческими данными в случае своего заполнения. Буфер также должен иметь возможность случайной выборки для обучения сетей. Вы можете инициализировать веса сети поразному, и они влияют как на стабильность, так и на скорость обучения [10]. Шаг 3 инициализирует копии сетей критиков и акторов, или *целевых сетей* (так их называют в литературе), для повышения стабильности. В остальной части алгоритма копии обозначаются символом '. Это сделано для экономии места, но их легко упустить. Если вы создаете собственную реализацию DDPG, еще раз проверьте, что вы используете правильную сеть. Реализации часто называют исходные сети *локальными*.

Шаг 6 использует детерминированную политику для прогнозирования действия. Именно здесь оригинальный алгоритм вносит в решение шум. Но вы можете рассмотреть добавление шума разными способами (например, *см. разд. "Зашумленные сети" главы 4*).

На шаге 8 текущий кортеж сохраняется в буфере воспроизведения, а на шаге 9 производится выборка из пакета, готового для обучения. Многие реализации откладывают первое обучение сетей, чтобы получить более репрезентативную выборку.

Шаг 10 означает начало процесса обучения сети. Сначала он предсказывает значение действия для следующего состояния, используя целевые сети и текущее вознаграждение. Затем шаг 11 обучает критика предсказывать это значение с использованием локальных весов. Если вы объедините шаги 10 и 11, то увидите, что это ошибка временной разницы.

Шаг 12 обучает сеть акторов с использованием локальных весов, минимизируя градиент функции ценности действия при использовании политики актора. Это уравнение 6.4 до применения правила цепочки для перемещения градиента политики из функции ценности действия. Реализации используют эту версию, потому что можно получить градиент оценки ценности действия из вашей DL-среды, вызвав функ-

цию градиента. В противном случае вам пришлось бы вычислять градиент функции ценности действия и градиент политики.

Наконец, шаги 13 и 14 копируют целевые веса в веса локальные. Они делают это путем экспоненциального взвешивания обновления; не стремитесь обновлять веса слишком быстро, иначе агент никогда не сойдется.

## Дважды отложенный DPG

*Дважды отложенный глубокий детерминированный градиент политики* (twin delayed deep deterministic policy gradients, TD3), как следует из названия, является улучшением DDPG. Как и в случае с радужным алгоритмом, Фуджимото и соавт. доказывают, что три новые детали реализации значительно улучшают производительность и надежность DDPG [11].

### Отложенные обновления политики

Первое и наиболее интересное предположение состоит в том, что архитектуры "актор — критик" страдают от неявной обратной связи. Например, оценки ценности будут плохими, если траектории, порождающие политику, неоптимальны. Но политика не может улучшиться, если оценки ценности неточны. В своей классической работе Фуджимото и соавт. предложили сценарий "Что появилось раньше: курица или яйцо?", суть которого — дождаться стабильных приближений, прежде чем переобучать сети. Статистическая теория утверждает, что вам нужна репрезентативная выборка, прежде чем вы сможете надежно оценить количество. То же самое и здесь; задерживая обновления, вы улучшаете производительность (*см. разд. "Отложенное Q-обучение" главы 3*).

Если вы реже переобучаете свои сети, производительность повышается. Тренироваться меньше, а не больше в долгосрочной перспективе — лучшее решение. Поначалу это кажется нелогичным, но у меня есть аналогия. Когда я работаю над сложными проектами, бывают случаи, когда что-то фундаментально влияет на работу, например финансирование. В этих ситуациях я понял, что спонтанные реакции вряд ли будут оптимальными. Они эмоционально обусловлены и основаны на неопределенной, зашумленной информации. Вместо этого я просто слушаю и собираю информацию, откладывая любые решения на потом. Я обнаружил, что, подождав несколько дней, я могу получить более четкое представление о ситуации. Отсроченная информация имеет меньшую дисперсию, поэтому мои прогнозы действий становятся более точными.

То же самое происходит и в архитектурах "актор — критик". Неявная обратная связь увеличивает вариативность обновлений. Когда сеть изменяется, результирующее изменение действий и траекторий является неопределенным. Вам нужно немного подождать, чтобы увидеть, каков будет эффект. Одно простое решение — отложить обновления. Переобучайте нейронные сети глубокого обучения через определенное время.

## Ограниченное двойное Q-обучение

Второе улучшение — двойное Q-обучение (*см. разд. "Двойное Q-обучение" главы 3*). Если бы вы сопоставили оценку ценности действия с истинной ценностью, вы бы увидели, что жадные методы имеют тенденцию к чрезмерной переоценке. Двойное Q-обучение решило эту проблему, введением двух оценок ценности действия. Когда вы переобучаете эти оценки, вы используете другие веса. Но в глубоких архитектурах это работает не так хорошо, потому что оценки не являются независимыми. Они используют один и тот же буфер воспроизведения и связаны с политикой генерации. Одно простое решение — выбрать самую низкую оценку ценности действия. Это предотвращает переоценку из-за активной недооценки. Несмотря на жесткий подход, исследователи обнаружили, что на практике он хорошо работает.

Ограниченное двойное Q-обучение (clipped double Q-learning, CDQ), например, значительно улучшает производительность в среде Ant (рус. муравей), что говорит о том, что оценки ценности действия, вероятно, будут сильно завышены. Предполагаю, что причина этого в том, что награда, которая обычно представляет собой скорость муравья, не зависит исключительно от этого единственного действия. У муравья шесть ног, поэтому, если одна из них производит плохое действие, пять других могут компенсировать это и потенциально привести к хорошей награде. Это приведет к ошибочному назначению положительной награды за такое плохое действие. CDQ смягчает эту проблему, сообщая о минимально возможном значении для любой пары "состояние — действие", но вы также можете улучшить производительность с помощью награды, предназначенной для изоляции каждой ноги (или сустава). Подозреваю, что дальнейшее повышение производительности в среде Ant или в любой среде ассоциировано со связанными действиями, если вы можете сделать назначение вознаграждения за конкретное действие более независимым.

## Сглаживание целевой политики

Последним и наименее значимым улучшением является добавление шума к действию при прогнозировании оценки ценности действия. Поскольку политика детерминирована, очень легко повторно выбрать одно и то же состояние и действие. Это приводит к пиковым значениям ценности действия. Одни исследователи предложили сглаживать пик, добавляя шум к действию, прежде чем предсказывать значение. Другие рекомендуют напрямую сглаживать оценку ценности действия, добавляя шум к весам сети.

Фактические данные показывают, что применение сглаживания целевой политики (target policy smoothing, TPS) в качестве дополнительного алгоритма само по себе не приводит к значительному улучшению результатов. А вот удаление его из алгоритма TD3 действительно имеет большое значение. Предполагаю, что причина этого в улучшении исследования. Если есть пики в оценке ценности действия, то потребуется много посещений соседних пар "состояние — действие", чтобы сгладить этот пик, что маловероятно с учетом большого пространства действий. Сглаживание упрощает изменение решения политики.

## Реализация TD3

Реализация TD3 представлена в алгоритме 6.4. Она похожа на реализацию DDPG, за исключением дополнительной логики для обработки двойного Q-обучения и отложенных обновлений политики. Обозначения соответствуют остальной части книги, и поэтому алгоритм выглядит несколько иначе, чем в исходной статье. Вы можете видеть, что терминология начинает усложняться из-за различных копий данных и параметров. Очень легко допустить ошибку реализации, так что будьте осторожны.

**Алгоритм 6.4. Алгоритм TD3**

1: **input**: актор нейронной сети $\mu_{\theta_\mu}(s)$ с весами $\theta_\mu$,
два критика нейронной сети $Q_{\theta_1}(s,a), Q_{\theta_2}(s,a)$ с весами $\theta_1$, $\theta_2$,
буфер воспроизведения $B$,
размер тренировочного мини-пакета $N$,
скорость обновления сети $\tau$,
коэффициент дисконтирования $\gamma$,
порог отсечения шума $c$,
функция шума для исследования $\mathcal{ON}$.

2: Инициализировать $\theta_\mu$, $\theta_1$, $\theta_2$, а также целевые веса $\theta'_\mu \leftarrow \theta_\mu$, $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2$.

3: **loop**: для каждого эпизода

4: Инициализировать $s$.

5: **while** $s$ не является терминальным состоянием:

6: $a \leftarrow \mu_{\theta_\mu}(s) + \mathcal{ON}$.

7: Выполнить действие $a$, а затем наблюдать за наградой $r$ и следующим состоянием $s'$.

8: Сохранить кортеж $(s, a, r, s')$ в буфере воспроизведения $B$.

9: Выбрать случайно мини-пакет из $N$ кортежей $(s_i, a_i, r_i, s'_i)$ из буфера $B$.

10: $\tilde{a}_i \leftarrow \mu_{\theta'_\mu}(s'_i) + \text{обрезка}(\mathcal{ON}, -c, c)$ (сглаживание целевой политики).

11: $y_i \leftarrow r_i + \gamma \min_{j=1,2} Q_{\theta'_j}(s'_i, \tilde{a}_i)$ (ограниченное двойное Q-обучение).

12: Обновить обоих критиков, $\theta_{j \leftarrow 1,2}$, минимизируя $\frac{1}{N}\sum_i \left(y_i - Q_{\theta_j}(s_i, a_i)\right)^2$.

13: **if** $t$ mod $d$ **then**: (отложенное обновление политики)

14: Обновить актора, минимизируя $\frac{1}{N}\sum_i \left(\nabla_{\theta_\mu} Q_{\theta_1}(s_i, \mu_{\theta_\mu}(s_i))\right)^2$.

15: Обновить обоих критиков, $\theta_{j\leftarrow 1,2}: \theta'_j \leftarrow \tau\theta_j + (1-\tau)\theta'_j$.

16: $\theta'_\mu \leftarrow \tau\theta_\mu + (1-\tau)\theta'_\mu$.

17: **end if**

18: $s \leftarrow s'$.

На шаге 1 теперь нужно пройти через две сети критиков, обозначенные параметрами $\theta_1$ и $\theta_2$. Большинство реализаций ожидают, что вы пройдете через две независимые сети DL, обычно имеющие одинаковую архитектуру. Но пусть это не помешает вам пробовать разные архитектуры или, возможно, многоголовую архитектуру.

Исходный алгоритм отсекает шум действия, чтобы сохранить результат, близкий к исходному. Цель состоит в том, чтобы предотвратить выбор агентом недопустимых действий. В большинстве реализаций это значение устанавливается в соответствии с пространством действий среды. Я предварительно предполагаю, что поскольку вы контролируете функцию шума, отсечение не требуется. Выберите функцию шума, которая соответствует вашему пространству действий и целям сглаживания, затем повторно используйте стандартную детерминированную функцию действия из DDPG (шаг 6).

Обратите внимание, что на шаге 11 предсказанное значение является минимальным значением от обоих критиков из целевых сетей. Другими словами, если вы выберете самую низкую оценку ценности действия, то это, скорее всего, будет правильным из-за предвзятости переоценки. Шаг 12 обновляет обоих критиков, используя прогноз дисконтированного вознаграждения. Это помогает сдерживать сеть, которая делает переоценку.

Внутренний цикл после шага 13 выполняется каждые $d$ итераций, что задерживает обновление политики. Строка 14 интересна тем, что для обновления политики используется первый критик, а не критик с минимальным значением. Я подозреваю, что причина здесь в том, что поскольку сеть с завышенной оценкой ограничена шагом 12, не имеет значения, выберете ли вы первую или вторую сеть. Любая сеть может быть преоценена. Но вы можете получить незначительный прирост производительности, если решите, какую именно сеть использовать более разумно.

Все остальные строки такие же, как и в алгоритме DDPG.

Всегда обращайтесь к оригинальной работе и добавляйте достаточно кода для контроля, чтобы осуществлять проверку своей реализации. Я считаю, что развертывание циклов и вычисления вручную — лучший способ убедиться, что код работает так, как я ожидал.

## Практический пример: рекомендации на основе отзывов

В эпоху информационной перегрузки пользователи все больше полагаются на алгоритмы рекомендаций для предложения релевантного контента или продуктов. Совсем недавно такие сервисы, как YouTube и TikTok, начали использовать интерактивные рекомендательные системы, которые постоянно советуют контент отдельным пользователям на основе отзывов других. Современные методы без RL пытаются построить модели, которые сопоставляют пользователей с элементами, применяют локально коррелированные шаблоны с помощью сверточных фильтров или используют модели последовательного прогнозирования на основе внимания. Последовательные модели работают лучше всего, потому что они адаптируются к повторяющейся обратной связи. Но эти методы обучаются в автономном режиме, и их необходимо часто переобучать, чтобы предоставлять актуальные рекомендации. В сочетании со сложностью моделей глубокого обучения это приводит к большим затратам на обучение.

RL хорошо подходит для этого типа задач, потому что обратная связь может быть учтена непосредственно через определение вознаграждения. Ванг и соавт. продемонстрировали пример, в котором они делают вещи более интересными, предлагая использовать текст в качестве механизма обратной связи, а не обычное голосование за или против. Они предполагают, что в реальном мире разреженное голосование может серьезно ухудшить производительность. Вместо этого они предлагают, чтобы текстовая информация, такая как отзывы и описания предметов, содержала больше информации и была менее чувствительной к разреженности [12].

Ванг и соавт. сформулировали проблему, используя марковский процесс принятия решений, где состояние — это взаимодействие между пользователем и рекомендательной системой, действие — это количество рекомендуемых элементов, а вознаграждение определяется обратной связью пользователя.

Основная проблема, как отмечается в других работах, — это высокая размерность, обусловленная огромным количеством пользователей и элементов. Ванг и соавт. выбрали путь использования встраивания слова для сопоставления текста элемента и отзыва пользователя с вектором признаков, а затем применили метод кластеризации, чтобы сократить вектор признаков до управляемого числа групп.

Исследовтели выбрали DDPG в качестве алгоритма выбора, чтобы иметь возможность изучать политики в многомерном непрерывном пространстве действий. Им нужна была способность предсказывать непрерывные действия для работы с постоянным пространством для встраивания пользователей. Затем они объединили набор элементов с действием для создания упорядоченного списка элементов, готового для представления пользователю. Со временем политика узнает, какие действия (т. е. встраивание пользовательских элементов) предпочтительны с учетом данного пользователя, а также элементов-кандидатов в рамках рассматриваемого состояния. По мере добавления новых отзывов пользователей или новых элементов политика может изменяться в соответствии с требованиями пользователей или предлагать новые рекомендуемые элементы.

К сожалению, Ванг и соавт. проверили это только в моделировании. Они использовали набор данных продуктов Amazon, имеющих отзывы. Была использована оценка отзыва для обозначения положительного или отрицательного отзыва и вознаграждения в зависимости от того, был ли рассмотренный товар пользователя в топе результатов. Их метод дал результаты, превосходящие все другие современные реализации машинного обучения.

Мне нравится идея включения текстовой информации как способа уменьшения разреженности проблемы. Но я бы хотел, чтобы исследователи проверили свой метод в реальных условиях. Конечно, не у всех есть доступ к YouTube или Amazon для проведения тестирования. Но только тогда мы сможем увидеть, насколько хорошо всё работает. Предполагаю, что предложенный метод должен работать как минимум так же хорошо, как любой хорошо настроенный рекомендательный движок машинного обучения, и, вероятно, превзойти его за счет того, что он не требует переобучения. И помните, что все это происходит без какой-либо специальной настройки или навороченных нейронных сетей. Политика представляла собой упрощенный многослойный персептрон. Я также уверен, что она требует улучшений. Другие исследователи сосредоточились на решении проблемы размерности, и им не нужно выполнять кластеризацию для выработки рекомендаций [13].

Вам также может показаться интересной настраиваемая платформа моделирования для рекомендательных систем RecSim[2] [14].

## Улучшения DPG

Вы можете применить многие фундаментальные улучшения, описанные в предыдущих главах, например различные стратегии исследования или способы представления ожидаемой награды. Все это может положительно повлиять на производительность. Например, алгоритм распределенного детерминированного градиента политики от Барт-Марона и соавт., которые не только ввели параллелизм (*см. разд. "Масштабирование RL" главы 10*), но и добавили распределительное RL (*см. разд. "Распределительное RL" главы 4*) и *n*-шаговые обновления (*см. разд. "n-Шаговые алгоритмы" главы 3*) [15]. Цай и соавт. на примере детерминированных градиентов целостной политики доказали, что вы можете использовать детерминированные алгоритмы для сред с бесконечным горизонтом и добавлять *n*-шаговые обновления, как вишенку на торт [16].

Но есть одна фундаментальная проблема, которая не решена, по крайней мере, в этом контексте. Несмотря на все преимущества, проблемы у алгоритмов вне политики такие же, как и у самолетов с треугольным крылом, подобных Eurofighter Typhoon: они по своей природе нестабильны. Без искусственного управления они распадутся и рухнут. Eurofighter достигает стабилизации с помощью сложной системы управления полетом. Алгоритмы, действующие без учета политики, добиваются этого, снижая скорость обновления параметров в нейронных сетях до точки, при которой обновления активно пропускаются. Конечно, это замедляет обучение. Но есть ли способ лучше?

---

[2] См. **https://oreil.ly/Dg_xB**.

# Методы доверительной области

PG-алгоритмы учатся делать лучший выбор, следуя градиенту политики в сторону более высоких прогнозируемых значений, взвешенных с помощью некоторой меры ожидаемой доходности. Все следующие методы используют функцию преимущества $A(s, a)$, но вы можете с уверенностью выбрать любую интерпретацию PG (*см. разд. "Сравнение основных алгоритмов градиента политики" главы 5*).

### Делегирование вычислений градиента библиотеке

Алгоритмы Q-обучения и градиентной политики могут решить значительный процент задач. Но основная цель разработки более продвинутых алгоритмов RL заключается в том, чтобы агенты могли решать более сложные задачи, такие как создание политик для робота, занимающегося сбором и перемещением объектов.

Задача, требующая более сложного RL-алгоритма, также подразумевает, что вам нужны более сложные аппроксиматоры. Их становится труднее кодировать, управлять ими и вычислять градиенты (в случае градиентов политики), и в какой-то момент, который является чрезвычайно нечетким, это становится слишком трудоемким. Очень заманчиво доверить все это фреймворку глубокого обучения. Да, у нейронных сетей есть свои проблемы, но эти фреймворки невероятно упрощают создание, управление и обновление очень сложных аппроксиматоров.

Поэтому в сложных PG-алгоритмах специалисты по RL склонны игнорировать оператор градиента ($\nabla$) и вместо него представлять функцию потерь (математические выражения внутри $\nabla$). Вычисление градиента передается на аутсорсинг алгоритму дифференцирования в обратном режиме, предоставляемому TensorFlow, PyTorch и т. п.

Уравнение 6.5 представляет обновление A2C (advantage actor-critic — алгоритм "актор — критик" с преимуществом) в форме функции потерь, где $A_\pi(s, a) \doteq \doteq Q_\pi(s, a) - V_\pi(s)$. Функция преимущества имеет интуитивно понятную интерпретацию. Действия с ценностью, большей нуля, лучше среднего. В среднем они обеспечат хорошую производительность. Действия с ценностью меньше нуля в среднем приводят к снижению производительности.

**Уравнение 6.5. Градиент политики алгоритма A2C**

$$L^{PG}(\theta) \doteq \mathbb{E}_{\pi_0}\left[\ln \pi_\theta(a|s) A_w(s, a)\right].$$

Использование функции преимущества в сочетании с градиентом политики заставляет политику производить более эффективные действия. Сравните это с использованием функции ценности действия. Для многих задач вполне вероятно, что все действия имеют положительную ценность; и если вы будете блуждать наугад, то в конце концов наткнетесь на награду. Так что, если вы использовали функции преимущества, как в случае с упрощенными актором и критиком, тогда алгоритм

будет двигать политику ко всем возможным результирующим случаям, но с разной скоростью. Это может привести к бесконечному увеличению оценок стоимости и создать нестабильность, когда значение близко к оптимальному, поэтому вам необходимо использовать дисконтирование. Функция преимущества обеспечивает естественное демпфирование. Если политика когда-либо зайдет слишком далеко, например если она совершит серьезную ошибку, функция преимущества вернет ее в нужное русло. Неправильный ход в политике проявится в плохих результатах, которые дадут отрицательное преимущество и, следовательно, отрицательный градиент, что приведет к повышению эффективности политики.

Но вы должны быть осторожны, чтобы политика не обновлялась слишком быстро. Функция преимущества — это приближение (контролируемая модель), поэтому вы должны относиться к результату с недоверием. Всё зашумлено, и вы не хотите испортить свою политику одним плохим шагом. Возникает вопрос: как определить размер шага?

Упрощенный преимущественный "актор — критик" продвигает траектории, которые приносят положительные преимущества. Он пытается максимизировать будущую прибыль, что достигается с помощью функции оптимизации, которая перемещает градиент политики вверх, что само по себе зависит от преимуществ. Этот градиент является локальной линейной оценкой сложной нелинейной функции. Вы не можете переместиться прямо с того места, где вы были, туда, куда хотите, потому что велика вероятность, что вы окажетесь неизвестно где.

Это все равно как если бы вы отправлялись в поход и задавали направление, взяв вначале одно показание компаса. Вы окажетесь в нескольких милях от пункта назначения, потому что не сможете определить азимут с достаточной точностью. Для того чтобы решить эту проблему, вы снимаете новые показания каждый километр или около того. Вы вводите размер шага и итеративно проводите измерения. В этой аналогии размер шага — это евклидово расстояние, гипотенуза треугольника, измеряемая в метрах. Стохастический градиентный подъем/спуск использует то же евклидово расстояние, но единицы гораздо сложнее интерпретировать. Функция стоимости выражается в единицах параметров политики, которые сами по себе сложным образом взаимодействуют с состояниями для создания вероятностей действий.

Связанная с этим проблема заключается в том, что данные, которые вы используете для обучения своих моделей, нестационарны. Например, рассмотрим обычную ситуацию, когда у вас мало вознаграждений (ваша политика не предусматривает вознаграждение в течение длительного времени). Вероятно, вы не захотите сильно изменять политику во время исследования, поэтому размер шага должен быть небольшим. Но когда вы все же найдете награду, то захотите значительно обновить политику, чтобы воспользоваться этими новыми знаниями. Но как сильно? Если вы совершите ошибку и слишком сильно обновите ее, политика может стать настолько плохой, что снова будет невозможно найти хорошую награду.

Эти проблемы затрудняют выбор единого размера шага, который работает во всех ситуациях. Практики обычно рассматривают шаг как гиперпараметр и задают диапазон значений. Другими словами, это грубая сила.

Методы доверительной области стремятся решить эту проблему, пытаясь указать область в пространстве параметров, шагам градиента которой можно доверять. Вы можете использовать эту информацию для динамического изменения размера шага, чтобы параметры менялись быстро, когда градиенты являются надежными, и медленно, когда это не так. Затем вы можете найти способы количественной оценки этого доверия, первый из которых основан на использовании расхождения Кульбака — Лейблера для измерения разницы между двумя политиками.

## Дивергенция Кульбака — Лейблера

Дивергенция Кульбака — Лейблера (Kullback — Leibler, KL) часто описывается как мера "неожиданности" при сравнении двух распределений. Когда распределения одинаковы, расхождение KL равно нулю. Когда распределения резко отличаются, расхождение KL велико.

Его также можно использовать для вычисления дополнительного количества битов, необходимых для описания нового распределения с учетом другого. Если распределения одинаковы, вам не требуются дополнительные биты для различения нового распределения. Если распределения разные, понадобится много новых битов.

Например, если у вас есть особая монета, которая падает только орлом вверх, вам понадобятся нулевые биты для представления состояний монеты. Есть только одно состояние — оно всегда будет орлом, поэтому для кодирования этого состояния вам нужны нулевые биты, только нулевое значение. Если вы решите сравнить "счастливую" монету с обычной, у вас будут два состояния, поэтому вам понадобится один полный бит для представления этих состояний. В этом гипотетическом примере расхождение KL между этими двумя распределениями равно 1 (гипотетически, потому что, когда одно состояние приближается к нулю, деление в алгоритме расхождения KL даст NAN).

Уравнение 6.6 представляет расчет расхождения. Символ || представляет собой оператор расхождения, который предписывает рассчитать расхождение между тем и этим. Вы можете использовать любую основу для логарифма, но большинство людей используют основание 2 ( $\log_2$ ), которое представляет количество битов.

**Уравнение 6.6. Дивергенция Кульбака — Лейблера**

$$D_{KL}\left(P \| Q\right) \doteq \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right).$$

### Эксперименты по дивергенции Кульбака — Лейблера

Рассмотрим следующие два эксперимента, в которых я подбрасываю монеты и кости. Дивергенция KL — это мера того, насколько различаются распределения, поэтому бросание двух нормальных монет должно привести к нулевой дивергенции. Затем я моделирую подбрасывание фальшивой монеты, всегда падающей решкой вверх.

Во втором эксперименте я моделирую подбрасывание трех шестигранных игральных кубиков. Первый — стандартный кубик. Второй — неплохой кубик (поврежденный), но его подбрасывание дает вдвое больше очков, чем бросание стандартного кубика. Другими словами, вероятности похожи, но распределения не совпадают. Третий кубик (шулерский) дает стандартные значения, но вбрасывает больше шестерок, чем вы ожидаете.

На рис. 6.3 показаны гистограммы этих экспериментов, где в скобках указано расхождение KL. Как и следовало ожидать, расхождение KL между двумя нормальными монетами близко к нулю. Однако при сравнении с предвзятой монетой расхождение отлично от нуля.

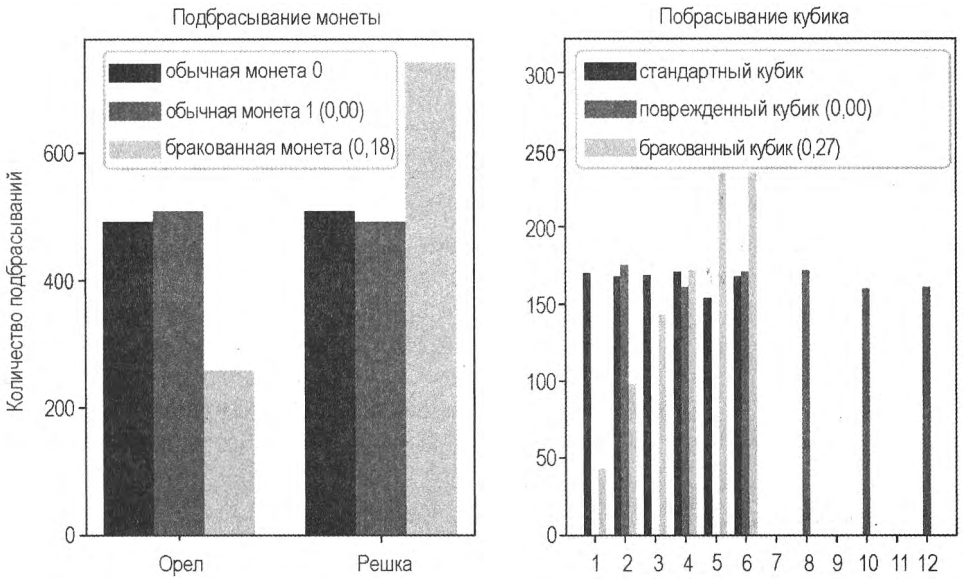



**Рис. 6.3.** Вычисление расхождения KL для моделирования подбрасывания монет и кубиков

Сравнение нормальных монет и фальшивой монеты дает один и тот же ненулевой результат. Но обратите внимание на результат сравнения нормального кубика и шулерского кубика, который дает значения, выходящие за пределы допустимого диапазона. Дивергенция KL близка к нулю потому, что там, где она перекрывается, она имеет такое же равномерное распределение. Это связано с тем, что дивергенция KL не измеряет разброса между распределениями; если вас это интересует, используйте вместо этого общую дивергенцию дисперсии.

## Естественные градиенты политики и оптимизация политики доверительной области

Естественные градиенты политики (natural policy gradients, NPG) ввели идею о том, что вы можете количественно определять размер шага не с точки зрения простран-

ства параметров политики, а с точки зрения метрики, основанной на расстоянии между текущей политикой и политикой после обновления с помощью шага градиента [17].

На рис. 6.4 продемонстрирована разница между упрощенными и естественными градиентами политики. Контуры представляют значение определенного состояния, а стрелки показывают направление градиента, определяемое типом вычисления градиента. Этот рисунок демонстрирует, что градиенты меняют направление, когда вы вычисляете дифференциал по отношению к другому объекту.

Вы можете оценить расстояние между двумя политиками, сравнивая распределения сгенерированных траекторий, используя одну из многих статистических мер для сравнения двух распределений. Но алгоритм оптимизации политики доверительной зоны (trust region policy optimization, TRPO) использует расхождение KL [18]. Напомним, что алгоритмы оптимизации измеряют размеры шага евклидовым расстоянием в пространстве параметров; шаги имеют единицы измерения $\delta\theta$. TRPO предлагает вам сдержать или ограничить этот размер шага в соответствии с изменениями в политике. Вы можете указать алгоритму оптимизации, что не хотите, чтобы распределение вероятностей сильно изменилось. Помните, что это возможно, потому что политики — это распределения вероятностей действия.

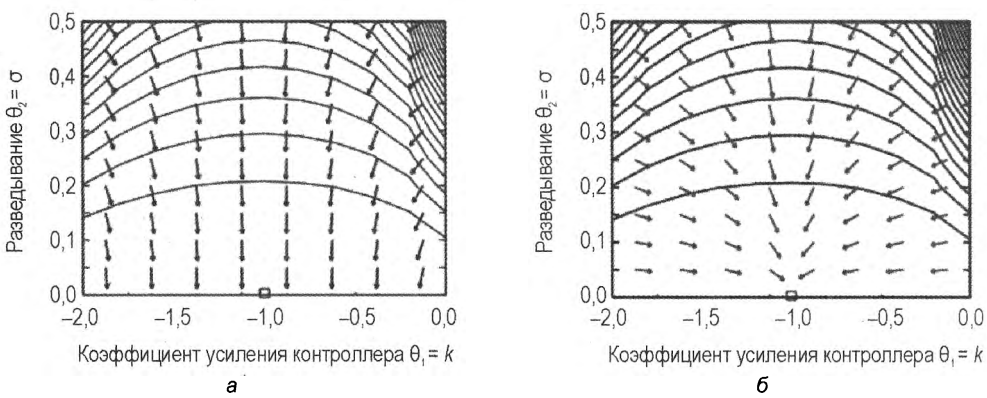



**Рис. 6.4.** Пример упрощенного (*а*) и естественного (*б*) градиентов политики, созданных в простой двумерной задаче. Изображения Peters et al. (2005), размещенные здесь с разрешения авторов [19]

Уравнение 6.7 представляет собой небольшую переформулировку градиента политики A2C. Если вы посмотрите на вывод теоремы о градиенте политики в уравнении 5.3, то увидите, что в конце я ввел логарифм для упрощения математических выкладок. Технически этот градиент вычисляется с использованием траекторий, взятых из старой политики. Поэтому, если вы отмените последний шаг (правило цепочки), получите то же уравнение, но использование старых параметров политики будет более явным.

**Уравнение 6.7. Переформулированная функция потерь "актор — критик"**

$$\nabla J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla \ln \pi_\theta(a \mid s) A_w(s,a)\right] =$$
$$= \mathbb{E}_{\pi_\theta}\left[\frac{\nabla \pi_\theta(a \mid s)}{\pi_{\theta_{old}}(a \mid s)} A_w(s,a)\right].$$

Затем, поскольку вы можете делегировать вычисление градиента, можете переписать уравнение 6.7 с точки зрения цели, которая заключается в максимизации ожидаемых преимуществ путем изменения параметров $\theta$ политики. Результат приведен в уравнении 6.8.

**Уравнение 6.8. Представление A2C с точки зрения цели максимизации**

$$\underset{\theta}{\text{максимизировать}}\ L^{PG}(\theta) = \mathbb{E}_{\pi_\theta}\left[\frac{\nabla \pi_\theta(a \mid s)}{\pi_{\theta_{old}}(a \mid s)} A_w(s,a)\right].$$

Альтернативная интерпретация уравнения 6.8 состоит в том, что это выборка преимуществ по значимости (а не градиент, как в *разд. "Алгоритм „актор — критик" вне политики" ранее в данной главе*). Ожидаемое преимущество равно преимуществу траекторий, выбранных старой политикой, скорректированных путем выборки значимости, чтобы дать результаты, которые были бы получены, если бы вы использовали новую политику.

Уравнение 6.9 показывает функцию потерь для алгоритма NPG, которая, как вы можете видеть, такая же, как уравнение 6.8, но с ограничением (математический инструмент, который ограничивает значение уравнения над ним). NPG использует ограничение, основанное на расхождении KL между новой и старой политиками. Это ограничение препятствует тому, чтобы расхождение было больше $\delta$.

**Уравнение 6.9. Функция потерь оптимизации градиентов естественной политики**

$$\underset{\theta}{\text{максимизировать}}\ L^{TRPO}(\theta) = \mathbb{E}_{\pi_\theta}\left[\frac{\dot{\nabla} \pi_\theta(a \mid s)}{\pi_{\theta_{old}}(a \mid s)} A_w(s,a)\right]$$
$$\text{с ограничением}\ \ \mathbb{E}_{\pi_\theta}\left[\mathrm{KL}\left[\pi_{\theta_{old}}(\cdot \mid s) \,\|\, \pi_\theta(\cdot \mid s)\right]\right] \le \delta.$$

Это все прекрасно, но главная проблема все еще в том, что вы не можете использовать стохастический градиентный спуск для оптимизации нелинейных функций потерь с ограничениями. Вместо этого вы должны использовать сопряженный градиентный спуск, который похож на стандартный градиентный спуск, но ограничение вы можете включить.

TRPO добавляет несколько усовершенствований, таких как дополнительная линейная проекция, чтобы дважды проверить, действительно ли шаг улучшает цель.

Однако основная проблема заключается в том, что реализация содержит сложные вычисления для нахождения сопряженных градиентов, что затрудняет понимание алгоритма и делает его затратным с точки зрения вычислений. Есть и другие проблемы, такие как тот факт, что вы не можете использовать нейронную сеть с многоголовой структурой, например, когда вы прогнозируете вероятности действий и оценку ценности состояния, потому что расхождение KL ничего не говорит вам о том, как вы должны обновить прогноз ценности состояния. Таким образом, на практике он не работает для задач, в которых важны хорошие оценки сложного пространства ценности состояния, таких как игры Atari.

Эти проблемы мешают популяризации TRPO. Вы можете узнать больше о происхождении NPG и TRPO в ресурсах, предложенных в *разд. "Дополнительные материалы для чтения" в конце этой главы.* К счастью, есть гораздо более простая альтернатива.

## Проксимальная оптимизация политики

Одной из основных проблем с NPG и TRPO является использование сопряженного градиентного спуска. Напомним, что цель состоит в том, чтобы предотвратить большие изменения в политике или, другими словами, предотвратить большие изменения в распределении вероятности действий. NPG и TRPO используют ограничение на оптимизацию для достижения этого, но другой способ — штрафовать большие шаги.

Уравнение 6.8 показывает, как выполнить максимизацию; оно пытается повысить оценки наибольших преимуществ, поэтому вы можете добавить отрицательный член внутри функции потерь, который произвольно уменьшает значение преимущества всякий раз, когда политика меняется слишком сильно. Уравнение 6.10 показывает это в действии, где ограничение переходит в отрицание, а дополнительный член $\beta$ управляет пропорцией штрафа, применяемого к преимуществам.

Основным достоинством является возможность использовать стандартные алгоритмы оптимизации градиентного подъема, что значительно упрощает реализацию. Однако сложно выбрать значение $\beta$, которое будет стабильно хорошо работать. Насколько большой штраф вам следует применить? И должен ли он меняться в зависимости от того, что делает агент? Это те же проблемы, что были представлены в начале данного раздела, и которые делают реализацию не лучше, чем выбор размера шага для A2C.

**Уравнение 6.10. Штраф за большие изменения в политике**

$$\max_{\theta} \mathbb{E}_{\pi_\theta}\left[\frac{\nabla\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} A_w(s,a) - \beta \mathrm{KL}\left[\pi_{\theta_{old}}(\cdot|s) \| \pi_\theta(\cdot|s)\right]\right].$$

Но не отчаивайтесь; уравнение 6.10 выражает важную идею: вы можете добавлять произвольные функции, чтобы влиять на размер и направление шага в алгоритме оптимизации. Это открывает ящик Пандоры, где вы можете представить себе до-

бавление таких вещей, как (более простые) функции для предотвращения больших размеров шагов, функции для увеличения размеров шагов, когда вам нужно больше исследований, и даже изменения направления, чтобы попытаться исследовать различные области в пространстве политики. Проксимальная оптимизация политики (proximal policy optimization, PPO) — одна из таких реализаций, которая добавляет более простые штрафы, интегрированную функцию, позволяющую использовать многоголовые сети, и дополнение на основе энтропии для улучшения исследования [20].

## Усеченная цель PPO

Шульман и соавт. предположили, что хотя TRPO технически более правильный, нет необходимости изобретать что-то необычное. Вы можете просто использовать коэффициент выборки по значимости как оценку размера изменения в политике. Значения, не близкие к 1, указывают на то, что агент хочет внести большие изменения в политику. Простое решение — обрезать эти значения, что равносильно ограничению на изменение политики на небольшую величину. Это приведет к новой функции ограничения потерь, показанной в уравнении 6.11.

**Уравнение 6.11. Отсечение крупных изменений в политике в соответствии с коэффициентом выборки по значимости**

$$L^{CLIP}(\theta) \doteq \mathbb{E}\left[\min\left(r(\theta)A(s,a)\right),\ \text{clip}\left(r(\theta),1-\varepsilon,1+\varepsilon\right)\hat{a}(s,a)\right].$$

$r(\theta) \doteq \pi_0(a|s)/\pi_{\theta_{\text{old}}}(a|s)$ — это коэффициент выборки по значимости. Внутри оператора ожидания есть функция min. Правая часть min содержит коэффициент выборки по значимости, который обрезается всякий раз, когда значение отклоняется более чем на $\varepsilon$ от 1. Левая часть функции min — это нормальная необрезанная цель. Причина использования min заключается в обеспечении наихудших возможных потерь для этих параметров, что называется нижней границей или *пессимистической границей*.

Сначала это звучит нелогично. В подобных ситуациях я считаю полезным представить, что происходит, когда вы тестируете функцию с разными входными данными. Помните, что функция преимущества может иметь положительные и отрицательные значения. На рис. 6.5 показаны усеченные потери при положительном (слева) и отрицательном (справа) преимуществах.

Положительное преимущество (слева) указывает на то, что политика оказывает на результат лучший, чем ожидалось, эффект. Положительные значения потерь подтолкнут параметры политики к этим действиям. По мере увеличения коэффициента выборки по значимости $[r(\theta)]$ правая часть функции min будет обрезаться в точке $1 + \varepsilon$ и производить наименьшие потери. Это приводит к ограничению максимального значения потерь и, следовательно, максимального изменения параметров. Не стоит менять параметры слишком быстро при улучшении, потому что слишком большой прыжок может привести к снижению производительности. В большинстве

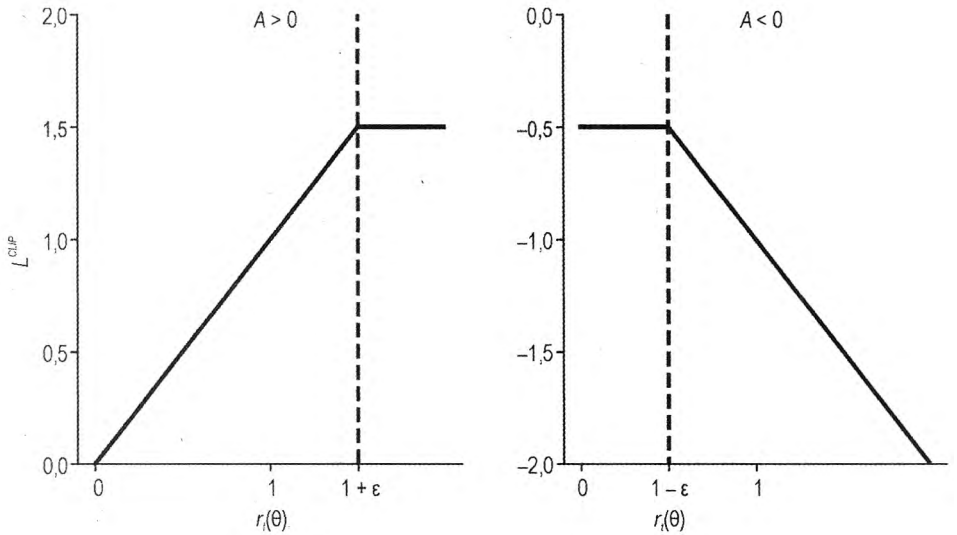



**Рис. 6.5.** График потери ограниченного PPO, когда преимущества положительные (*слева*) или отрицательные (*справа*)

случаев соотношение будет около 1. Величина ε дает немного простора для улучшения.

Отрицательное преимущество предполагает, что политика оказывает на результат худшее, чем ожидалось, влияние. Отрицательные значения потерь отодвигают параметры политики от этих действий. При высоких соотношениях преобладает левая часть функции min. Большие отрицательные преимущества подтолкнут агента к более многообещающим действиям. При низких соотношениях правая часть функции min снова доминирует и ограничивается $1-\varepsilon$. Другими словами, если распределение имеет низкое соотношение (т. е. существует значительная разница между старой и новой политиками), то параметры политики будут продолжать изменяться с той же скоростью.

Но почему? Зачем вообще это делать? Почему бы просто не использовать чистое преимущество? Шульман и соавт. намеревались установить и улучшить нижнюю границу производительности, что является наихудшим сценарием. Если вы можете разработать функцию потерь, которая гарантирует улучшение (они это сделали), тогда вам не нужно так сильно заботиться о размере шага.

На рис. 6.6 показана эта проблема с использованием обозначений TRPO. θ — параметры текущей политики. В идеале хочется наблюдать кривую, обозначенную $\eta(\theta)$, которая является истинной производительностью с использованием параметров $\theta$. Но вы не можете этого сделать, потому что для оценки этой сложной нелинейной кривой на каждом этапе потребуется слишком много времени. Более быстрый метод — вычислить градиент в точке, которая отображается линией $L(\theta)$. Это градиент потерь A2C. Если вы сделаете большой шаг в этом градиенте, можете получить параметры, которые превышают или занижают максимум. Кривая, обозначенная $L(\theta)-\beta KL$, представляет собой нижнюю границу TRPO. Это градиент A2C

без расхождения KL. Вы можете видеть, что независимо от того, какой размер шага вы выбрали, вы все равно окажетесь где-то в пределах $\eta(\theta)$); даже если агент делает действительно плохой шаг, он вынужден оказаться где-то в пределах этой нижней границы, что весьма неплохо.

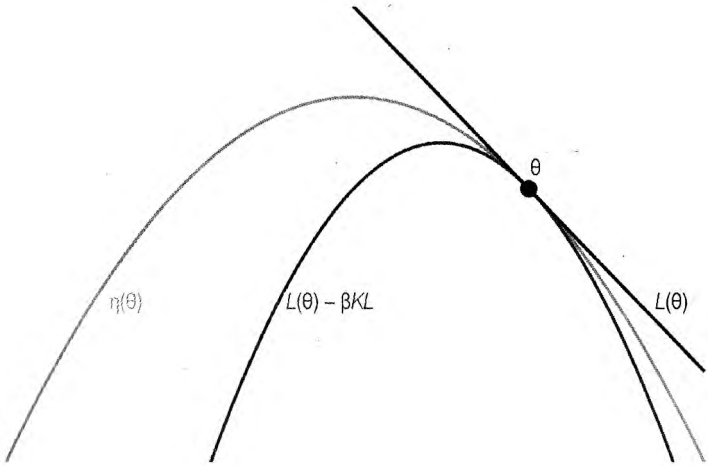



**Рис. 6.6.** Изображение нижней границы TRPO с градиентами

PPO достигает аналогичной нижней границы, игнорируя выборку по значимости, когда политика улучшается (делает большие шаги), но включает ее и, следовательно, подчеркивает потери, когда политика ухудшается (обеспечивает нижнюю границу). Предпочитаю думать об этом как о нарастании давления; PPO требует улучшения преимуществ.

## Ценностная функция PPO и цели разведки

Большинство методов вычисления преимуществ используют приближение для оценки функции ценности состояния. На этом этапе вы, вероятно, смирились с использованием нейронной сети, поэтому имеет смысл использовать одну и ту же сеть для прогнозирования действий политики и функции ценности состояния. В этом сценарии PPO включает член, связанный с потерями в конечной цели в квадрате, чтобы позволить структурам глубокого обучения изучать функцию ценности состояния одновременно с изучением политики: $L^{VF}(\theta) \doteq \left(V_\theta(s) - V^{\text{цель}}\right)^2$.

PPO также включает в конечную цель термин, который вычисляет энтропию политики $\mathcal{H}[\pi_\theta](s)$ [21]. Это улучшает исследование, препятствуя преждевременному переходу к субоптимальным детерминированным политикам. Другими словами, он активно препятствует детерминированной политике и заставляет агента выполнять разведку. Вы узнаете больше об этой идее в *главе 7*.

В уравнении 6.12 приведена окончательная целевая функция PPO. $C_1$ и $C_2$ — гиперпараметры, которые управляют составляющими функции потерь, связанной с потерей ценности состояния и потерей энтропии соответственно.

**Уравнение 6.12. Конечная целевая функция для PPO**

$$L^{CLIP+VF+S}(\theta) \doteq \mathbb{E}\left[ L^{CLIP}(\theta) + C_1 L^{VF}(\theta) + C_2 \mathscr{H}[\pi_\theta](s) \right].$$

Алгоритм 6.5 показывает обманчиво простой псевдокод из исходной статьи Шульмана и соавт. [22]. Он не учитывает детали реализации, такие как приближение политики и функции ценности, хотя предполагает реализацию на основе глубокого обучения для обеспечения автоматического дифференцирования. Обратите внимание, что в алгоритме есть неявное отложенное обновление; он выполняет итерацию для $T$ временны́х шагов перед повторным обучением весов. Но он не касается каких-либо других расширений (например, детерминизма, буферов воспроизведения, отдельных обучающих сетей, распределения действий/вознаграждений и т. д.). Любое из перечисленного может улучшить производительность для вашей конкретной задачи.

Что интересно, так это использование нескольких агентов на шаге 2. Я не много говорил об асинхронных методах, но это простой способ улучшить скорость обучения (в режиме реального времени). Несколько агентов могут наблюдать за разными ситуациями и передавать их в центральную модель. Я обсуждаю эту идею более подробно в *разд. "Замечание об асинхронных методах" далее в этой главе.*

Шульман и соавт. использовали значения от 0,1 до 0,3 для гиперпараметра отсечения дивергенции $\varepsilon$. Гиперпараметр $C_1$ зависит от того, используете ли вы одну и ту же нейронную сеть для прогнозирования политики и функции ценности состояния. Им даже не нужно было использовать бонус энтропии в некоторых своих экспериментах, и они установили $C_2 \doteq 0$. Правильного ответа нет, поэтому используйте поиск по гиперпараметрам, чтобы найти оптимальные значения для вашей задачи.

**Алгоритм 6.5. Асинхронный алгоритм PPO**

1: **for** iteration $\doteq$ 0, 1, ...:

2: **for** agent $\doteq$ 0, 1, ..., $N-1$:

3: Запустить политику $\pi_{\theta_{old}}$ в среду за $T$ временны́х шагов.

4: Вычислить оценки преимущества $A_0, ..., A_{T-1}$.

5: Оптимизировать $L$ с учетом параметра $\theta$ для $K$ эпох и мини-пакета размером $V \leq NT$.

6: $\theta_{old} \leftarrow \theta$.

# Пример: использование сервоприводов для Real-Life Reacher

Вы, наверное, заметили, что в большинстве примеров RL используются симуляции. Это делается в основном из практических соображений; намного проще, дешевле и быстрее тренироваться в смоделированной среде. Но вы можете легко применить RL к проблемам, имеющим непосредственное отношение к реальной жизни. Одним из наиболее распространенных примеров RL в прикладной сфере является роботизированное управление, т. к. RL особенно хорошо подходит для разработки эффективных политик для сложных задач.

В академических кругах попытались стандартизировать платформы робототехники для улучшения воспроизводимости. Платформа ROBEL[3] — один из ярких примеров трех простых стандартизированных роботов, которые способны демонстрировать ряд сложных движений [23]. Но даже несмотря на то, что они "относительно недороги", стоимость D'Claw в 3500 долларов является довольной высокой, хотя ее вполне можно оправдать.

Две проблемы: отсутствие реальных примеров RL и стоимость — явились причинами проведения следующего эксперимента. Я хотел воссоздать типичный эксперимент RL, но в реальной жизни, который вернул бы меня к классическим задачам CartPole и Reacher. К тому же высокая стоимость D'Claw в первую очередь связана с дорогими серводвигателями. Для этого эксперимента мне нужен только один серводвигатель и драйвер, которые можно купить менее чем за 20 долларов. Полную информацию о настройке оборудования можно найти на соответствующем вебсайте (*см. разд. "Дополнительные материалы" в предисловии*).

## Описание эксперимента

Этот эксперимент состоит из смоделированного и реального серводвигателей, которых я веду к заданной цели. Серводвигатели похожи на стандартные двигатели постоянного тока, за исключением того, что они имеют дополнительный редуктор и элементы управления для обеспечения точного управления. Сервоприводы, которые я использовал, также имели обратную связь по положению, чего нет в более дешевых моделях. На рис. 6.7 показаны движения серводвигателя во время обучения.

Состояние представлено текущим положением сервопривода и фиксированной целью, которая изменяется в каждом эпизоде. Есть одно действие: положение сервопривода. Состояния и действия нормализованы, чтобы попасть в диапазон от –1 до 1. Наградой является отрицательное евклидово расстояние до цели — классический показатель расстояния до цели, который используется во многих примерах робототехники. Эпизод заканчивается, когда сервопривод перемещается в пределах 5% целевой позиции. Поэтому цель состоит в том, чтобы обучить агента вывести

[3] См. **https://oreil.ly/Kki8B**.

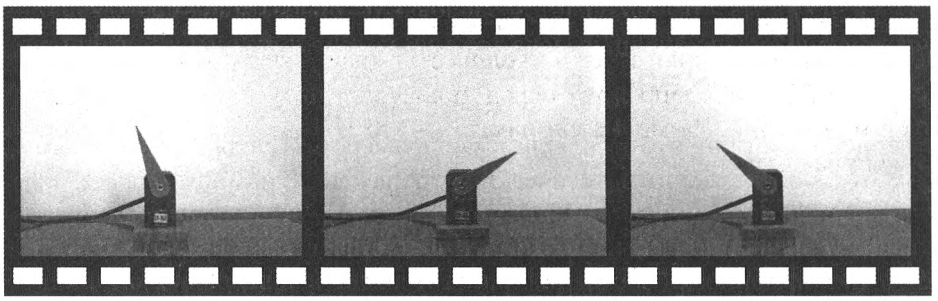


**Рис. 6.7.** Видеокадры агента PPO, изучающего движение серводвигателя

политику, которая перемещается к точному положению цели за наименьшее количество ходов.

Эта постановка задачи отличается от CartPole или Reacher. Но идея научить робота двигаться для достижения желаемой цели — обычное требование. Единственный сервопривод значительно упрощает решение проблемы, и вы можете вручную кодировать политику. Эта простота облегчает понимание, а более сложные варианты являются лишь продолжением той же идеи.

## Реализация алгоритма RL

Всегда пытайтесь решить следующую простейшую задачу. RL особенно сложно отлаживать, потому что существует так много циклов обратной связи, гиперпараметров и хитросплетений среды, которые коренным образом меняют эффективность обучения. Лучшее смягчение последствий — делать маленькие шаги и убеждаться, что более простые варианты работают, прежде чем переходить к чему-то более сложному.

Моя первая попытка решить поставленную задачу заключалась в том, чтобы научиться двигаться к цели, которая была полностью зафиксирована для всех эпизодов. В этом примере я хотел использовать PPO. Я выбрал значение –1, т. к. произвольные инициализации часто случайно приводят к политике, которая на каждом шаге сваливается в 0. Это означает, что агент должен изучить невероятно простую политику: всегда предсказывать действие с –1. Легко, правда?

Невероятно, но не сбылось. Если бы я был менее настойчив, забросил бы свою инженерную деятельность и ушел на пенсию. В общем, считаю, что большинство проблем связано с ошибкой, существующей между мозгом и клавиатурой. Итак, я начал отладку с построения функции предсказываемой ценности, политики и фактических действий, которые прогнозировались для данного наблюдения, и обнаружил, что через некоторое время функция ценности предсказывала правильные значения, а политика просто зависала. Она вообще не двигалась. Оказалось, что скорость обучения по умолчанию (stable-baselines, PPO2, $2{,}5 \cdot 10^{-4}$) слишком мала для такой простой задачи. Дело не в том, что агент не учился, он просто учился слишком медленно.

После значительного увеличения скорости обучения (0,05), чтобы соответствовать простоте задачи, алгоритм решил задачу всего за несколько обновлений.

Следующая проблема возникла, когда я начал возиться с количеством шагов на обновление и размером мини-пакета. Количество шагов на обновление показывает, сколько наблюдений отбирается перед повторным обучением аппроксиматоров политики и значений. Размер мини-пакета — это количество случайных выборок, переданных на обучение.

Помните, что PPO, согласно бумажной реализации, не имеет никакого буфера воспроизведения. Количество шагов на обновление представляет собой размер мини-пакета, используемого для обучения. Если вы установите слишком низкое значение, то образцы будут коррелированы и политика никогда не сойдется.

Хочется установить параметр количества шагов больше, чем размер мини-пакета (чтобы выборки были менее коррелированными) и в идеале захватить несколько эпизодов, чтобы снова декоррелировать. Другой способ подумать об этом заключается в том, что каждый эпизод, снятый во время обновления, подобен нескольким независимым работникам. Более независимые работники позволяют агенту усреднять независимые наблюдения и, следовательно, нарушать корреляцию с уменьшением прибыли.

Размер мини-пакета влияет на точность градиентов аппроксимации политики и ценности, поэтому большее значение обеспечивает более точный градиент для этой выборки. В машинном обучении это считается положительным моментом, но в RL может вызвать всевозможные проблемы со стабильностью. Приблизительные значения политики и ценности действительны только для этого этапа разведки. Например, в сложной задаче с большими пространствами состояний маловероятно, что первая выборка будет репрезентативной для всей среды. Если вы используете большой размер мини-пакетов, это усилит текущее поведение политики, а не подтолкнет политику к лучшей политике. Вместо этого разумнее использовать меньший размер мини-пакетов, чтобы политика не соответствовала траектории. Однако оптимальный размер мини-партии зависит от задачи и алгоритма [24].

Для рассматриваемой задачи я установил очень маленький размер мини-пакета, 2 или 4 в большинстве экспериментов, чтобы можно было часто переучиваться и быстро решать задачу, и количество шагов до 32 или 64. Агент легко мог решить задачу среды за такое количество шагов, поэтому весьма вероятно, что в одном обновлении присутствовало несколько эпизодов.

Наконец, поскольку получившуюся политику было так легко изучить, я резко уменьшил размеры политик и функций ценностей сетей. Это ограничивало их обоих и предотвращало переоснащение. Реализация по умолчанию многослойного персептрона с прямой связью в стабильных базовых состояниях не имеет отсева, поэтому я также заставил сеть изучить скрытое представление данных с помощью подобного автокодировщику разветвления.

На рис. 6.8 показано поведение окончательной политики, которую я создал, запросив прогнозируемое действие для всех наблюдений за положением для фиксированной цели –1. После всего лишь 20 шагов двигателя PPO изучил политику, которая производит действие –1 для всех наблюдений.

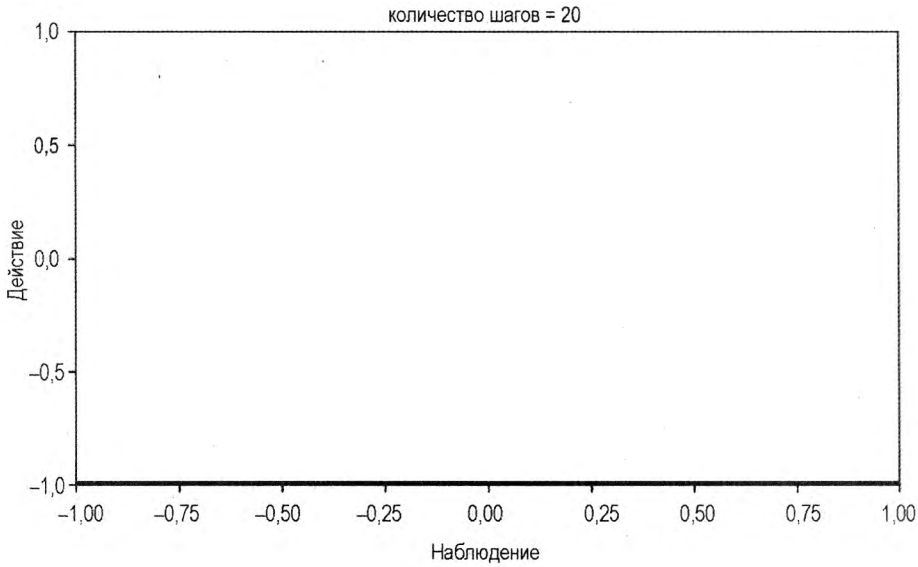



**Рис. 6.8.** Финальная политика эксперимента с фиксированной целью. Ось $x$ — это текущее наблюдение за положением. Ось $y$ — результат действия политики. На участке действия $y = -1$ есть прямая линия, которая правильно соответствует цели –1

## Повышение сложности алгоритма

Теперь, когда я убедился, что оборудование работает и агент может изучить жизнеспособную политику, пришло время сделать следующий шаг и заняться немного более сложной задачей.

Теперь я расширяю проблему, генерируя случайные цели для каждого эпизода. Это сложнее, потому что агенту нужно будет наблюдать множество разных эпизодов и, следовательно, несколько целей, чтобы узнать, что ему нужно перейти в ожидаемую позицию. Агент не будет знать, что есть разные цели, пока не столкнется с ними.

Агенту также необходимо обобщать. Цели выбираются равномерно и находятся в диапазоне от –1 до 1. Это непрерывная переменная; агент никогда не будет соблюдать каждую цель.

Однако цель все же проста. Агент должен научиться переходить на позицию, предложенную целью. Это взаимно однозначное сопоставление. Но это представляет собой небольшую проблему для агентов RL, потому что они этого не знают. Агенты достаточно оснащены для отображения очень сложной функции. Так же как и раньше, мне нужно будет ограничить это исследование рассматриваемой задачей.

С технической точки зрения это интересный эксперимент, поскольку он подчеркивает взаимодействие между динамикой среды и гиперпараметрами агента. Несмотря на то что эту функцию легко освоить (прямое сопоставление), исследование и взаимодействие агента вносят значительный шум в процесс. Вы должны быть осторожны и предпринять шаги, чтобы усреднить этот шум. В частности, если скорость

обучения слишком высока, то агент имеет тенденцию постоянно обновлять свои политики и ценности сетей, что приводит к крайне нестабильной политике.

Количество шагов тоже важно. Раньше было легко завершить эпизод, т. к. всегда была одна цель. Но теперь цель может быть где угодно, поэтому эпизод может занять намного больше времени в начале обучения. Это в сочетании с высокой скоростью обучения может привести к быстрому изменению выборок.

Я мог бы решить эту проблему, ограничив агента использованием более простой модели, например линейной. Другой подход — оптимизировать гиперпараметры. Я выбрал последний вариант, чтобы продолжать использовать стандартный подход многослойного персептрона.

Я обнаружил, что небольшое количество мини-пакетов с высокой частотой обновления обучаются быстрее всего за счет долгосрочной стабильности обучения. Если вы тренируете свою сеть для выполнения более важной задачи, вам следует снизить скорость обучения, чтобы сделать обучение более стабильным и последовательным.

## Настройка гиперпараметров в моделировании

Правда, я не занимался всей этой настройкой с сервоприводами напрямую. Когда проблемы начинают усложняться, становится все труднее выполнять поиск по гиперпараметрам. В первую очередь это связано с количеством времени, которое требуется для выполнения шага в реальной среде, но также существуют опасения по поводу безопасности и износа. В этих экспериментах мне удалось успешно повредить один сервопривод всего за 5 дней разработки; позиционная обратная связь не сработала. На рис. 6.9 представлено изображение мертвого сервопривода, которое является напоминанием о том, что в реальной жизни чрезмерные движения вызывают износ.

По этой причине имеет смысл создать симулятор для первоначальной настройки. Вероятно, работу модели будет намного быстрее и дешевле оценить, чем работу

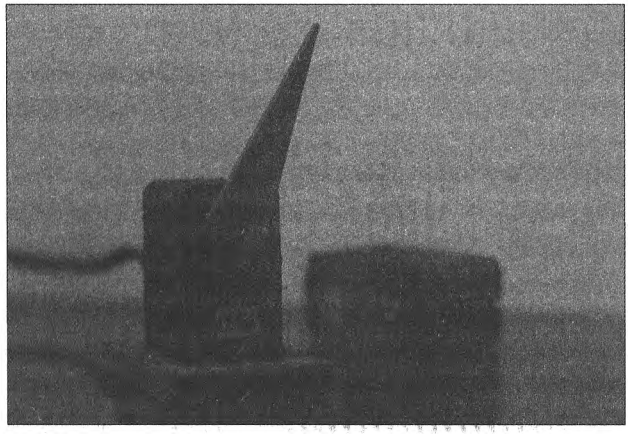


**Рис. 6.9.** Последнее место упокоения перегруженного сервопривода

реальной вещи. Итак, я добавил базовую симуляцию сервопривода и немного отрегулировал гиперпараметры.

Вот список советов по настройке.

- Скорость обучения обратно пропорциональна необходимому времени обучения. Меньшие, более медленные темпы обучения требуют больше времени для обучения.
- Мини-пакеты меньшего размера лучше подходят для обобщения.
- Старайтесь иметь по несколько эпизодов в каждом обновлении, чтобы нарушить корреляцию выборки.
- Независимо настраивайте сети оценки стоимости и политики.
- Не обновляйтесь слишком быстро, иначе обучение будет колебаться.
- Существует компромисс между скоростью обучения и его стабильностью. Если вы хотите провести более стабильное обучение, выполняйте его медленнее.

Но, честно говоря, не имеет значения, какие гиперпараметры вы для этого используете. Это может быть решено как угодно. Все, что мы здесь делаем, — это пытаемся контролировать стабильность обучения и повышать скорость. Даже в этом случае для получения достаточно стабильной политики все еще требуется около 1000 шагов. Главный фактор — скорость обучения. Вы можете уменьшить ее и получить более быстрое обучение, но политика склонна к колебаниям.

## Результирующие политики

На рис. 6.10 показана обученная политика на четырех различных этапах обучения в смоделированной среде: 250, 500, 750 и 1000 шагов среды, которые соответствуют смоделированным движениям сервопривода. Вы можете создать подобные графики, попросив политику предсказать действие (значения, указанные на шкалах справа от графиков) для данной цели (ось $x$) и положения (ось $y$) наблюдения. Идеальная политика должна выглядеть как гладкие вертикальные полосы; гладкие, т. к. это непрерывное действие, и вертикальные, потому что мы хотим, чтобы действие соответствовало запрошенной цели; наблюдение положения не имеет значения.

В начале обучения все прогнозы равны нулю. Со временем агент учится двигаться выше, когда цель выше, и ниже, когда цель ниже. В результирующей политике есть тонкая диагональ, потому что модель имеет достаточно свободы, чтобы сделать этот выбор, и различные случайные выборки "сговорились", чтобы превратить эту политику в идеальную. При наличии дополнительного времени на обучение и с учетом усреднения это решение в итоге было бы идеальным.

На рис. 6.11 показан очень похожий результат при обучении агента на реальном сервоприводе. Диагональ более выражена, но это может быть связано с выбранными случайными выборками. Но и действия не столь ярко выражены. Черные пикселы не такие черные, а белые — не такие белые. Как будто в реальной жизни больше неопределенности в правильном действии, что я объясняю тем фактом, что запрошенное движение не идеально. Если я запрошу позицию 0,5, она фактически пере-

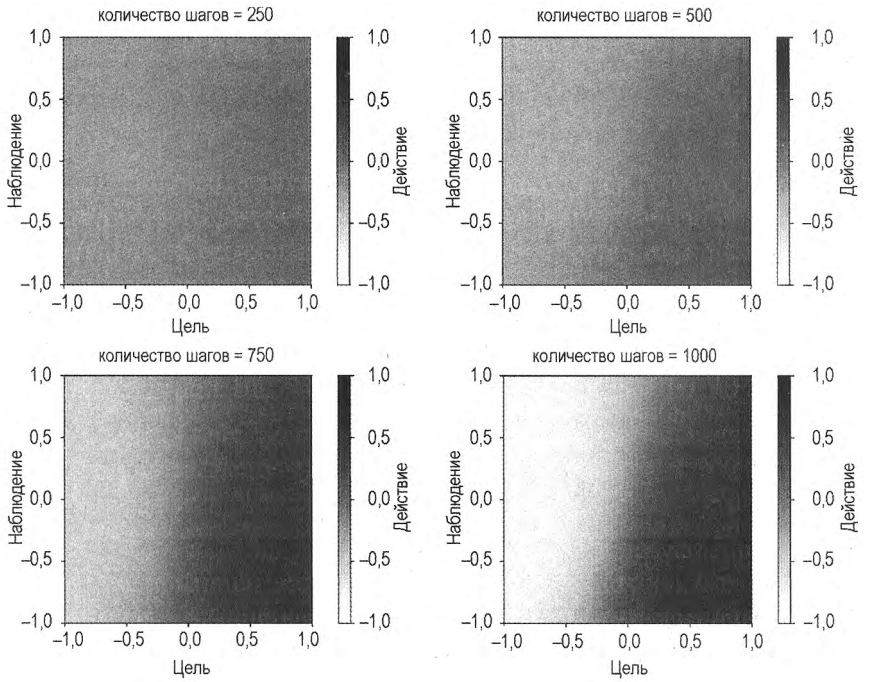



**Рис. 6.10.** Обученная на разном количестве шагов политика для моделируемой среды. На этом изображении показана политика, предсказывающая действие (затенение) для данной цели (ось $x$) и наблюдаемого положения (ось $y$)

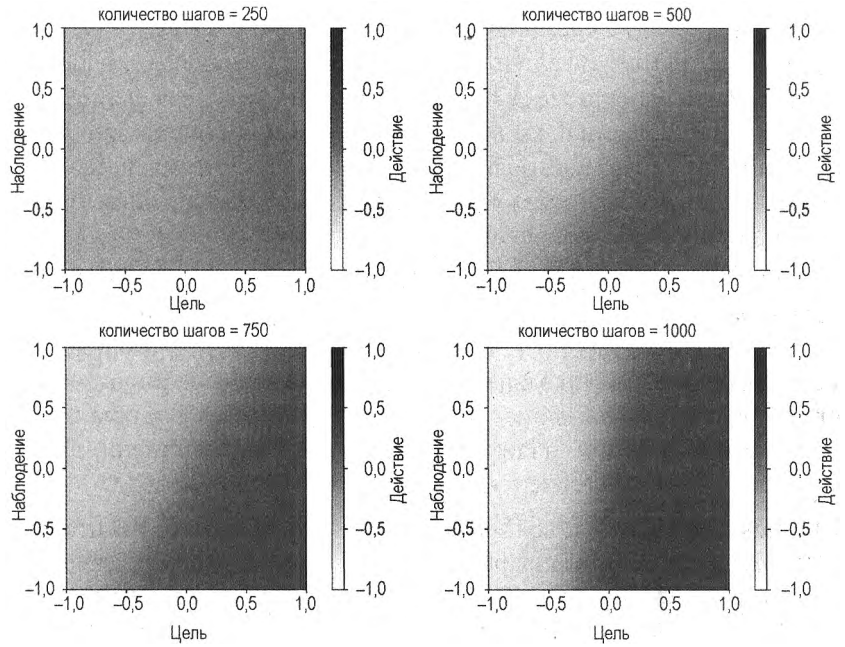



**Рис. 6.11.** Обученная политика, использующая настоящий серводвигатель, на разном количестве шагов среды. Оси такие же, как на рис. 6.10

местится, например, на 0,52. Я считаю, что эта позиционная ошибка проявляется как слегка неуверенная политика.

Как и раньше, если у вас будет больше времени на обучение и усреднение, это приведет к совершенно жизнеспособной и полезной политике.

# Другие алгоритмы градиента политики

Градиенты политики, особенно алгоритмы PG, не связанные с политикой, являются областью активных исследований благодаря неотъемлемой привлекательности прямого изучения политики на собственном опыте. Но в некоторых средах, особенно в средах со сложными пространствами состояний, производительность отстает от методов оценки. В этом разделе я представляю введение в два основных направления исследований: иные способы стабилизации обновлений вне политики, которые вы уже видели в форме TRPO и PPO, и выразительные методы, которые пытаются использовать градиент ошибки Беллмана, а не преимущества.

## Алгоритм Retrace (λ)

Алгоритм Retrace (λ) — повторная трассировка — это ранний PG-алгоритм вне политики, который использует взвешенную и отсеченную выборку по значимости для того, чтобы помочь контролировать ту часть вне политики, которая включается в обновление [25].

Мунос и соавт. усекают коэффициент выборки по значимости до максимального значения 1, что аналогично пессимистической границе в TRPO/PPO, и ограничивают улучшения в политике. В лучшем случае разница минимальна, поэтому обновление — стандартный алгоритм A2C. В худшем случае обновлением можно пренебречь, т. к. эти две политики очень разные. Причина отсечения — предотвратить взрывной рост дисперсии (повторяющиеся высокие значения приводят к резким изменениям в политике). Недостаток кроется в добавлении предвзятости; результат более стабилен, но менее доступен для исследования.

Доходность также взвешивается по $\lambda$, прослеживается соответствие требованиям из *разд. "Трассировки соответствия" главы 3*. Это обновляет политику на основе предыдущих посещений, что обеспечивает гиперпараметр для управления дилеммой смещения и дисперсии (bias-variance trade-off).

## Алгоритм ACER

ACER (actor-critic with experience replay — "актор — критик" с опытом воспроизведения) расширяет алгоритм Retrace (λ), включая комбинацию воспроизведения опыта, оптимизацию политики доверительной области и дуэльную сетевую архитектуру [26].

Для начала, Ванг и соавт. переформулировали алгоритм Retrace (λ), чтобы компенсировать предвзятость, вызванную усечением выборки по значимости. Другими

словами, повторная трассировка предотвращает большие изменения в политике, что делает ее более стабильной (более предвзятой), но менее доступной для исследования. Компенсирующий коэффициент частично устраняет эту потерю исследования пропорционально величине усечения в алгоритме Retrace ($\lambda$).

Авторы также включили более эффективную реализацию TRPO, которая поддерживает среднюю политику, предсказываемую нейронной сетью. Если вы кодируете дивергенцию KL как штраф, а не как ограничение, можете использовать фреймворки глубокого обучения с автоматическим дифференцированием.

Наконец, исследователи повысили эффективность выборки за счет воспроизведения опыта (*см. разд. "Воспроизведение опыта" главы 4*). Для пространств непрерывного действия они включили дуэльную архитектуру (*см. разд. "Дуэльные сети" главы 4*), чтобы уменьшить расхождения. Это приводит к двум отдельным алгоритмам: одному для дискретных действий и одному для непрерывных.

Результатом является алгоритм, который приближается к производительности глубокой Q-сети с приоритетным воспроизведением опыта (*см. разд. "Воспроизведение приоритетного опыта" главы 4*), но может также работать с непрерывными пространствами действий. Однако все исправления приводят к довольно сложному алгоритму и, следовательно, сложной архитектуре нейронной сети.

## Алгоритм ACKTR

В *разд. "Естественные градиенты политики и оптимизация политики доверительной области" ранее в этой главе* вы увидели, что использование естественного градиента политики и ограничения дивергенцией KL выгодно, поскольку градиент не зависит от параметризации аппроксимирующей функции, а ограничение KL улучшает стабильность. Однако определение градиента требует дорогостоящих вычислений для нахождения информационной матрицы Фишера. Линейная проекция TRPO — еще один источник неэффективности.

В 2017 г. Ву и соавт. предложили использовать аппроксимации для ускорения вычислений [27]. Они применяют приближение с факторизацией Кронекера (Kronecker-factored approximation, K-FAC), которое является приближением информационной матрицы Фишера, для улучшения вычислительной производительности, и скользящее среднее этого приближения вместо линейной проекции, чтобы уменьшить количество взаимодействий агента с окружающей средой.

Кроме того, поскольку вы можете использовать K-FAC со сверточными нейронными сетями (например, PPO), это означает, что алгоритм ACKTR (actor-critic using kronecker-factored trust regions — "актор — критик", использующий доверительные области по фактору Кронекера) может обрабатывать дискретные визуальные пространства состояний, такие как тесты Atari.

С 2017 г. независимые обзоры показали, что эффективность ACKTR по сравнению с TRPO сильно зависит от среды. Например, в крайне нестабильной среде MuJoCo Swimmer-v1 TRPO превосходит большинство алгоритмов. Но в среде HalfCheetah-v1 TRPO работает значительно хуже [28]. В своей статье Хендерсон и соавт. предста-

вили результаты Swimmer-v1, которые показывают, что алгоритмы A2C, TRPO и ACKTR работают одинаково (*см. разд. "Какой алгоритм следует использовать?" далее в этой главе*). Как ни странно, практики предпочитают ACKTR, а не TRPO, потому что идея и конечная производительность аналогичны, но эффективность вычислений значительно повышается.

## Эмпатические методы

В методах градиентна с учетом временны́х различий (*см. разд. "Градиентное обучение с учетом временных различий" ранее в этой главе*) была представлена идея использования функции для аппроксимации функции ценности состояния и прямого вычисления градиента прогнозируемой ошибки Беллмана, чтобы избежать расхождения аппроксимации. Однако эмпатическим методам нужны два набора параметров и две скорости обучения, что затрудняет их практическое использование. Для того чтобы решить эту проблему, Саттон и соавт. предложили выразительные методы временных различий [29, 30].

В наиболее общей форме эмпатическое обучение с учетом временны́х различий использует функцию для количественной оценки интереса к конкретным состояниям в соответствии с текущей политикой. Это представлено вместе с выборкой по значимости для компенсации действий агентов, не связанных с политикой. Однако выбор функции оказался трудным, и проблемы с производительностью, связанные с проекцией, препятствуют успешному прорыву.

Одно особенно интересное выразительное предложение, называемое обобщенным критиком субъектов вне политики (generalized off-policy actorcritic, Geoff-PAC), предполагает использование градиента противоречивой цели [32].

*Контрфактическое (сомнительное) обоснование* — это процесс опроса данных, которые не наблюдаются; например, спрашивается: какое вознаграждение за новую политику будут давать данные, собранные в прошлом? Не вдаваясь в детали, скажу, что идея состоит в том, что вы можете узнать приближение к эмпатической функции (конечно, с помощью нейронной сети), предсказав *коэффициент плотности*, что похоже на непрерывную версию выборки по значимости. Результаты многообещающие, но методы не получили широкого распространения.

# Расширения для алгоритмов градиента политики

Читая книгу, вы уже познакомились с рядом усовершенствований алгоритмов, которые в принципе можно применить к любому алгоритму. Каждые несколько лет исследователи публикуют научные работы, в которых эти усовершенствования объединяются для получения новейших результатов в тестовых средах и привлекают внимание средств массовой информации. Но практику, как правило, более интересны сопутствующие исследования, поскольку они показывают, какие настройки могут лучше всего подходить для вашей задачи. Для того чтобы продемон-

стрировать этот подход, далее приведен пример того, как вы можете добавить дистрибутивный RL в алгоритмы PG.

### Квантильная регрессия в алгоритмах градиента политики

Рихтер и соавт. демонстрируют, что можно включить распределение RL в алгоритмы градиента политики во многом таким же образом, как было описано в *разд. "Повышение вознаграждения" главы 4* для ценностных методов [33]. Они используют нейронную сеть для прогнозирования квантилей каждого действия. Как и раньше, это полезно, потому что отображение действий в награды часто бывает стохастическим, и многие среды имеют мультимодальные распределения. Ключевым отличием этой работы является адаптация к алгоритмам градиента политики. Авторы начинают со стандартной функции потерь A2C (см. уравнение 6.8) и функцию преимущества заменяют ее оценками квантилей. Затем они делают серию приближений, чтобы вычесть квантильную оценку, что приводит к версии, которая итеративно обновляет оценки квантилей в соответствии с температурным гиперпараметром. Они внедрили версию этого алгоритма, основанную на политике, которая превзошла TRPO в некоторых тестах (авторы не сравнивали ее с PPO), и предположили, что можно использовать этот алгоритм аналогичным образом.

## Резюме

В этой довольно обширной главе вы познакомились с множеством современных алгоритмов градиента политики с упором на методы вне политики алгоритма "актор — критик". В *главе 5* была представлена идея абстрагирования оценки политики и улучшения политики в архитектурах "актор — критик". Целью этой главы было объяснить и продемонстрировать силу обучения вне политики.

Перспектива обучения вне политики велика, когда агент изучает множество политик, отличных от поведенческой. Изучение вне политики освобождает агента от тирании компромисса между разведыванием и добычей наград, позволяя поведенческой политике свободно исследовать. Поскольку оптимальная политика является отдельной, вы можете использовать автономные журналы данных или учиться на человеческом примере. Но главное преимущество состоит в том, что вы сколько угодно переобучаете свои аппроксимирующие функции, потому что это не требует затрат на подготовку среды. В результате получаются алгоритмы, которые гораздо эффективнее с точки зрения выборки, хотя и затратнее за счет вычислительной сложности, особенно если вы используете нейронные сети.

### Какой алгоритм следует использовать?

Ваша растерянность по поводу того, какой алгоритм лучше использовать, понятна. Этот вопрос также волнует и исследователей. Был проведен ряд исследований, в которых была предпринята попытка дать ответ, но, перефразируя одно резюме,

для исследователей очень соблазнительно представить случайные результаты, которые могут быть не совсем надежными [34]. Приведу в качестве конкретного примера результаты для ряда сред, представленные в оригинальной статье об ACKTR, которые другие пытались воспроизвести. Это, вероятно, связано с деталями реализации или удачным набором гиперпараметров, но действительно подчеркивает проблему. Однако я могу дать несколько общих рекомендаций.

- Выбирайте TD3, а не DDPG.
- Предпочитайте метод доверительной области в средах с нестабильной динамикой, в которых небольшое изменение политики может иметь катастрофические последствия.
- Предпочитайте детерминированные методы в средах со стабильной динамикой.
- Рассмотрите более простые алгоритмы и нейронные сети меньшего размера, если производительность или надежность важны, например PPO или TD3.

## Замечание об асинхронных методах

Еще одна важная особенность алгоритмов вне политики заключается в том, что они освобождают вас от использования единственного критика. Я еще не упомянул об этом (*см. разд. "Масштабирование RL" главы 10*), но поскольку критики учатся на основе буфера воспроизведения, нет причин отказываться от множества повторяющихся критиков, обучающихся отдельным правилам. В этих *асинхронных методах* вам нужно в какой-то момент снова объединить знания в политику поведения, это может значительно улучшить производительность алгоритмов в режиме реального времени. Я еще не обсуждал эти вопросы, т. к. считаю, что это особенности конкретной реализации и многое зависит от задачи, которую нужно изучить быстрее, чтобы оправдать дополнительные расходы. Что касается промышленных проблем, я бы сначала посоветовал вам рассмотреть возможность упрощения задачи или предметной области, чтобы устранить необходимость в таких методах, поскольку они значительно увеличивают операционную сложность и стоимость.

# Дополнительные материалы для чтения

- Градиенты естественной политики и TRPO (ссылки на статьи):
  - краткие, но приятные лекции от Джона Шульмана[4];
  - презентация о CMU 10703[5];
  - лекция Сергея Левина[6];
  - пояснительное видео от Ксандера Стинбрюгге (Xander Steenbrugge)[7].

[4] См. **https://oreil.ly/eOhYM**.

[5] См. **https://oreil.ly/-Ys4X**

[6] См. **https://oreil.ly/604zI**.

- Численная оптимизация:
  - Nocedal J., Wright S. Numerical optimization. — 2nd ed. — Springer Series in Operations Research and Financial Engineering. — New York: Springer-Verlag, 2006.

# Использованные источники

[1] Sutton R. S., Maei H. R., Szepesvári C. A convergent $O(n)$ temporal-difference algorithm for off-policy learning with linear function approximation // Advances in Neural Information Processing Systems 21 / ed. by D. Koller, D. Schuurmans, Y. Bengio, L. Bottou. — Curran Associates, Inc., 2009. — P. 1609–1616. — URL: **https://oreil.ly/iq9Pg**.

[2] Sutton R. S., Maei H. R., Precup D. et al. Fast gradient-descent methods for temporal-difference learning with linear function approximation // Proceedings of the 26th Annual International Conference on Machine Learning. ICML'09. — Montreal, Quebec, Canada: Association for Computing Machinery, 2009. — P. 993–1000. — URL: **https://oreil.ly/aprqz**.

[3] Sutton R. S., Maei H. R., Precup D. et al. Fast gradient-descent methods for temporal-difference learning with linear function approximation // Proceedings of the 26th Annual International Conference on Machine Learning. ICML'09. — Montreal, Quebec, Canada: Association for Computing Machinery, 2009. — P. 993–1000. — URL: **https://oreil.ly/qpk-g**.

[4] Maei H. R., Szepesvári C., Bhatnagar S., Sutton R. S. Toward off-policy learning control with function approximation // Proceedings of the 27th International Conference on International Conference on Machine Learning. ICML'10. — Haifa, Israel: Omnipress, 2010. — P. 719–726.

[5] White A., Sutton R. S. GQ(λ) quick reference and implementation guide // ArXiv:1705.03967. — 2017. — May. — URL: **https://oreil.ly/pj7Wc**.

[6] Degris T., White M., Sutton R. S. Off-Policy actor-critic // ArXiv:1205.4839. — 2013. — June. — URL: **https://oreil.ly/zcR5K**.

[7] Silver D., Lever G., Heess N. et al. Deterministic policy gradient algorithms // International Conference on Machine Learning. — 2014. — P. 387–395. — URL: **https://oreil.ly/PU2hT**.

[8] Lillicrap T. P., Hunt J. J., Pritzel A. et al. Continuous control with deep reinforcement learning // ArXiv:1509.02971. — 2019. — July. — URL: **https://oreil.ly/D_Zjm**.

[9] Plappert M., Houthooft R., Dhariwal P. et al. Parameter space noise for exploration // ArXiv:1706.01905. — 2018. — January. — URL: **https://oreil.ly/klLak**.

[10] Salimans T., Kingma D. P. Weight normalization: a simple reparameterization to accelerate training of deep neural networks // Advances in Neural Information Processing Systems 29 / ed. by D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, R. Garnett. — Curran Associates, Inc., 2016. — P. 901–909. — **URL: https://oreil.ly/LFWvR**.

[11] Fujimoto S., Van Hoof H., Meger D. Addressing function approximation error in actor-critic methods // ArXiv:1802.09477. — 2018. — October. — URL: **https://oreil.ly/r0qaN**.

[12] Wang C., Guo Z., Li J., Pan P., Li G. A text-based deep reinforcement learning framework for interactive recommendation // ArXiv:2004.06651. — 2020. — July. — URL: **https://oreil.ly/32vIS**.

[7] См **https://oreil.ly/AzF9z**.

[13] Ie E., Jain V., Wang J. et al. Reinforcement learning for slate-based recommender systems: A tractable decomposition and practical methodology // ArXiv:1905.12767. — 2019. — May. — URL: **https://oreil.ly/lPePY**.

[14] Ie E., Hsu C.-wei, Mladenov M. et al. RecSim: A configurable simulation platform for recommender systems // ArXiv:1909.04847. — 2019. — September. — URL: **https://oreil.ly/ZnQs6**.

[15] Barth-Maron G., Hoffman M. W., Budden D. et al. Distributed distributional deterministic policy gradients // ArXiv:1804.08617. — 2018. — April. — URL: **https://oreil.ly/Uz0S5**.

[16] Cai Q., Pan L., Tang P. Deterministic Value policy gradients // ArXiv:1909.03939. — 2019. — November. — URL: **https://oreil.ly/AXfzI**.

[17] Kakade S. M. A Natural policy gradient // Advances in Neural Information Processing Systems 14 / ed. by T. G. Dietterich, S. Becker, Z. Ghahramani. — MIT Press, 2002. — P. 1531–1538. — URL: **https://oreil.ly/yQkbV**.

[18] Schulman J., Levine S., Moritz P., Jordan M. I., Abbeel P. Trust region policy optimization // ArXiv:1502.05477. — 2017. — April. — URL: **https://oreil.ly/d4qxR**.

[19] Peters J., Vijayakumar S., Schaal S. Natural actor-critic // Machine Learning: ECML 2005 / ed. by J. Gama, R. Camacho, P. B. Brazdil et al. — Lecture Notes in Computer Science. — Berlin, Heidelberg: Springer, 2005. — P. 280–291. — URL: **https://oreil.ly/Ks5mV**.

[20] Schulman J., Wolski F., Dhariwal P., Radford A., Klimov O. Proximal policy optimization algorithms // ArXiv:1707.06347. — 2017. — August. — URL: **https://oreil.ly/HDdBG**.

[21] Mnih V., Badia A. P., Mirza M. et al. Asynchronous methods for deep reinforcement learning // ArXiv:1602.01783. — 2016. — June. — URL: **https://oreil.ly/x84W3**.

[22] Schulman J., Wolski F., Dhariwal P., Radford A., Klimov O. Proximal policy optimization algorithms // ArXiv:1707.06347. — 2017. — August. — URL: **https://oreil.ly/HDdBG**.

[23] Ahn M., Zhu H., Hartikainen K. et al. ROBEL: Robotics benchmarks for learning with low-cost robots // ArXiv:1909.11639. — 2019. — December. — URL: **https://oreil.ly/yJsP2**.

[24] Song X., Du Y., Jackson J. An Empirical study on hyperparameters and their interdependence for RL generalization // ArXiv: 1906.00431. — 2019. — June. — URL: **https://oreil.ly/LDusn**.

[25] Munos R., Stepleton T., Harutyunyan A., Bellemare M. G. Safe and efficient off-policy reinforcement learning // ArXiv:1606.02647. — 2016. — November. — URL: **https://oreil.ly/29RAJ**.

[26] Wang Z., Bapst V., Heess N. et al. Sample efficient actor-critic with experience replay // ArXiv:1611.01224. — 2017. — July. — URL: **https://oreil.ly/4dMmv**.

[27] Wu Y., Mansimov E., Liao S., Grosse R., Ba J. Scalable trust-region method for deep reinforcement learning using kroneckerfactored approximation // ArXiv:1708.05144. — 2017. — August. — URL: **http://arxiv.org/abs/1708.05144**.

[28] Henderson P., Islam R., Bachman P. et al. Deep reinforcement learning that matters // ArXiv: 1709.06560. — 2019. — January. — URL: **https://oreil.ly/zgyQ4**.

[29] Sutton R. S., Mahmood A. R., White M. An emphatic approach to the problem of off-policy temporal-difference learning // ArXiv:1503.04269. — 2015. — April. — URL: **https://oreil.ly/95_Z3**.

[30] Zhang S., Boehmer W., Whiteson S. Generalized off-policy actor-critic // ArXiv:1903.11329. — 2019. — October. — URL: **https://oreil.ly/tHEt8**.

[31] Gelada C., Bellemare M. G. Off-Policy Deep reinforcement learning by bootstrapping the covariate shift // ArXiv:1901.09455. — 2019. — January. — URL: **https://oreil.ly/1TKVs**.

[32] Zhang S., Boehmer W., Whiteson S Generalized off-policy actor-critic // ArXiv:1903.11329. — 2019. — October. — URL: **https://oreil.ly/NCo2a**.

[33] Richter O., Wattenhofer R. Learning policies through quantile regression // ArXiv:1906.11941. — 2019. — September. — URL: **https://oreil.ly/6I-C9**.

[34] Henderson P., Islam R., Bachman P. et al. Deep reinforcement learning that matters // ArXiv: 1709.06560. — 2019. — January. — URL: **https://oreil.ly/F9Xvc**.

ГЛАВА 7

# Изучение всех возможных политик с помощью энтропийных методов

Глубокое обучение с подкреплением (deep reinforcement learning, DRL) является стандартным инструментом благодаря способности обрабатывать и аппроксимировать сложные наблюдения, которые приводят к сложному поведению. Однако многие методы глубокого RL оптимизируются для детерминированной политики; если бы у вас была полная наблюдаемость, существовала бы только одна лучшая политика. Но часто бывает желательно изучить стохастическую политику или вероятностное поведение для повышения устойчивости и работы со стохастическими средами.

## Что такое энтропия?

*Энтропия Шеннона* (далее — *энтропия*) — это мера количества информации, содержащейся в стохастической переменной, где информация рассчитывается как количество битов, необходимых для кодирования всех возможных состояний. Это определение иллюстрируется уравнением 7.1, где $X \doteq \{x_0, x_1, ..., x_{n-1}\}$ — стохастическая переменная; $\mathscr{H}$ — энтропия; $I$ — информационное содержание; $b$ — основание логарифма (используют обычно логарифмы: двоичный для $b \doteq 2$, десятичный на $b \doteq 10$ и натуральный для $b \doteq e$). Двоичная система (биты) — самое распространенное основание.

**Уравнение 7.1. Информационное содержание случайной величины**

$$\mathscr{H}(X) \doteq \mathbb{E}\left[I(X)\right] = -\sum_{x \in X} p(x) \log_b p(x).$$

Например, упавшая монета имеет два состояния, если предположить, что она не приземляется на ребро. Эти два состояния могут быть закодированы нулем и единицей, поэтому количество информации, содержащейся в монете, измеренное энтропией в битах, равно единице (1 бит). У кубика есть шесть возможных состояний, поэтому вам понадобится три бита для описания всех этих состояний.

Вероятностным решением оптимального управления является стохастическая политика. Для того чтобы получить точное представление о распределении "действие — вероятность", вы должны выбрать достаточное количество состояний и действий. Можете измерить количество посещенных состояний и действий, как в UCB (*см. разд. "Улучшение ε-жадного алгоритма" главы 2*). Но это не связано напрямую

с политикой; UCB — это стратегия разведки, а не часть политики. Вместо этого вы можете использовать прокси-меру того, насколько хорошо распределена политика, например энтропию, и включить ее в качестве штрафного члена в целевую функцию.

# Максимальная энтропия обучения с подкреплением

Максимизация энтропии политики заставляет агента посещать все состояния и действия. Вместо того стремления найти детерминированное поведение, которое максимизирует вознаграждение, политика может изучить *все* модели поведения. Другими словами, вместо того, чтобы изучать лучший способ выполнения задачи, терм энтропии заставляет агента изучать *все* способы решения задачи. Это приводит к политикам, которые могут интуитивно исследовать и обеспечивать более устойчивое поведение перед лицом враждебных атак, например забрасывания робота кирпичами. Посмотрите видео в блоге PPO[1], чтобы повеселиться. Подробнее об этом я расскажу в *главе 10*.

В уравнении 7.2 представлена целевая функция для максимальной энтропии RL, где $\alpha$ — гиперпараметр (называемый *температурой* и связанный с термодинамическим определением энтропии), который определяет относительную важность члена энтропии по сравнению с вознаграждением и контролирует стохастичность оптимальной политики.

**Уравнение 7.2. Целевая функция максимальной энтропии**

$$\pi^* \doteq \arg\max_{\pi} \sum_{s \in \mathcal{S},\, a \in \mathcal{A}} \mathbb{E}\left[ r + \alpha \mathcal{H}\left(\pi(a \mid s)\right) \right].$$

Уравнение 7.2 гласит, что оптимальная политика определяется как политика, которая максимизирует доход, регулируемый энтропией политики. Это похоже на стандартное определение оптимальной политики, за исключением того, что оно дополняется бонусом для продвижения недетерминированных политик, что прямо противоположно детерминированным градиентам политики.

Насколько хорошо это работает, зависит от задачи. Если у вас есть дискретная задача, имеющая только одну оптимальную траекторию, и вы уверены, что агент не отклонится от этой траектории, то детерминированная политика — хороший выбор. Но в случае непрерывных задач, когда вероятно, что агент отклонится от оптимальной политики, лучшим вариантом может быть рассматриваемый подход.

Гиперпараметр $\alpha$ позволяет контролировать, какая часть бонуса включена. Значения, близкие к нулю, становятся детерминированными, а большие значения способствуют недетерминированным действиям и, следовательно, большему исследованию.

---

[1] См. **https://oreil.ly/ckDQo**.

# Мягкий "актор — критик"

Мягкий алгоритм "актор — критик" (soft actor-critic, SAC) представляет собой аппроксимацию градиента вне политики максимальной целевой функцией энтропии [1]. Термин "мягкий" используется для выделения недетерминированных элементов в алгоритме; стандартное уравнение Беллмана без члена энтропии можно считать "жестким", поэтому Q-обучение и DDPG являются соответствующими "жесткими" алгоритмами.

Хаарноя и соавт. начинают свое исследование с определения критика как мягкой аппроксимации функции ценности действия, представленной в более ранней статье [2]. Напомним, что цель состоит в том, чтобы создать политику, максимизирующую энтропию, а это означает, что агент должен ценить состояния с высокой энтропией больше, чем другие. Продолжая развивать идею уравнения 7.2, уравнение 7.3 показывает, что ценность состояния — это ценность действия плюс бонус за энтропию этого нового состояния. Я упростил обозначения, чтобы они соответствовали остальной части этой книги. Уравнение 7.3 предполагает, что вы следуете политике, но я скрыл эту информацию, чтобы упростить понимание математики. $V$ и $QQ$ представляют функции "состояния — значение" и ценности действия, соответственно.

**Уравнение 7.3. Функция ценности состояния максимальной энтропии**

$$V(s) = \mathbb{E}\left[Q(s,a) + \alpha\mathscr{H}\left(\pi(a|s)\right)\right] =$$
$$= \mathbb{E}\left[Q(s,a) - \alpha\log\left(\pi(a|s)\right)\right].$$

Уравнение 7.4, используя уравнение 7.3, показывает соответствующую функцию ценности действия из динамического программирования.

**Уравнение 7.4. Функция максимальной энтропии ценности действия**

$$Q(s,a) \doteq \mathbb{E}\left[r + \gamma\mathbb{E}\left[V(s')\right]\right].$$

Для обработки сложных пространств состояний вы можете обучить функцию аппроксимировать уравнение 7.4, для чего вам потребуется целевая функция (уравнение 7.5).

**Уравнение 7.5. Целевая функция максимальной энтропии ценности действия**

$$J_Q(\theta) = \mathbb{E}\left[\frac{1}{2}\left(Q_\theta(s,a) - Q(s,a)\right)^2\right] =$$
$$= \mathbb{E}\left[\frac{1}{2}\left(Q_\theta(s,a) - \left(r + \gamma\mathbb{E}\left[V_\theta(s')\right]\right)\right)^2\right].$$

Вы можете получить градиент уравнение 7.4 (вскоре вы увидите это), но реализации SAC обычно используют фреймворки с автоматическим дифференцированием, включенные в библиотеку глубокого обучения. Вам нужна политика для актора и аналогичная целевая функция. Можно вывести мягкий градиент политики аналитически [3]. Но Хаарноя и соавт. предпочли вместо этого минимизировать расхождение Кульбака — Лейблера (Kullback — Leibler, KL), (*см. разд. "Дивергенция Кульбака — Лейблера" главы 6*). Причина этого заключалась в том, что Лю и Ван разработали подход, называемый *вариационным градиентным спуском Штейна*, который представляет собой аналитическое решение градиента дивергенции KL, вывод которого выходит за рамки книги [4]. Хаарноя и соавт. применили этот метод к мягкой политике, что привело к целевой политике, представленной в уравнении 7.6, которое можно решить с помощью автодифференциации.

**Уравнение 7.6. Целевая функция политики SAC**

$$J_{\pi}(\varphi) = \mathbb{E}\Big[\alpha \log\big(\pi_{\varphi}(a|s) - Q_{\theta}(s,a)\big)\Big].$$

## Детали реализации SAC и дискретные пространства действий

В выводе есть детали, которые влияют на реализацию. Во-первых, производительность очень чувствительна к выбору температуры. Вам необходимо выполнить сканирование гиперпараметров, чтобы установить оптимальную температуру для каждой задачи. Во-вторых, гиперпараметр чувствителен к температуре, что также означает, что он чувствителен к шкале вознаграждений. Например, большие награды будут преобладать над небольшими значениями температуры. Измените размер награды, чтобы уменьшить ее. Эти две проблемы исправлены в следующем алгоритме.

Хаарноя и соавт. предположили гауссовские действия, что делает задачу решаемой (так называемый *трюк с репараметризацией*) и позволяет ей работать с непрерывными пространствами действий [5]. SAC не работает с дискретными пространствами действий без предварительной настройки.

Эта возможность добавлена в алгоритм SAC, предложенный Христодулу, который переформулирует проблему максимальной энтропии с дискретными действиями — различия минимальны [6]. Вы можете спросить: зачем использовать алгоритм градиента политики вместо чего-то вроде DQN? Основное преимущество SAC с дискретными пространствами действий в том, что он может конкурировать с ценностными методами с точки зрения эффективности обучения, но с нулевой настройкой гиперпараметров. Эта идея воплощена в измененной SAC-реализации с помощью алгоритма, который автоматически регулирует температуру.

## Автоматическая регулировка температуры

В своей статье Хаарноя и соавт. представили способ автоматической регулировки температурного параметра алгоритма SAC, задав условия работы алгоритма в виде

ограничения, а не вводя штраф. Они ограничили текущую политику, чтобы иметь более высокую энтропию, чем ожидаемая минимальная энтропия решения. Затем они обнаружили, что могут итеративно обновлять модель температурного параметра, чтобы минимизировать разницу между энтропией текущей политики и ожидаемой энтропией, как показано в уравнении 7.7, где $\alpha$ — температурный параметр, а $\bar{\mathcal{H}} \doteq -\dim(\mathcal{A})$ — ожидаемая энтропия, что совпадает с отрицательным числом действий [7].

**Уравнение 7.7. Целевая функция температурного параметра алгоритма SAC**

$$J_\alpha = \mathbb{E}\left[-\alpha \log\left(\pi(a \mid s)\right) - \alpha\bar{\mathcal{H}}\right].$$

Хаарноя и соавт. добавили эту строку в исходный алгоритм SAC, чтобы превратить его в алгоритм градиента вне политики без гиперпараметров. Это заманчивая функция, поскольку она сглаживает человеческий фактор и ускоряет разработку, обеспечивая высокую производительность. Конечно, вам еще предстоит настройка: структура нейронной сети, конструирование состояний, класс действий и т. д. Но это шаг в правильном направлении.

## Практический пример: автоматическое управление трафиком для сокращения очередей

Нет ничего более универсального, чем синхронный стон пассажиров при приближении к пробке. Узкие места дороги, создаваемые зонами ремонтов и авариями, являются одним из наиболее важных факторов непериодических перегрузок и вторичных аварий. "Умные" автомагистрали призваны улучшить среднюю скорость в потоке с высокой плотностью движения, предлагая автомобилистам возможность перестроиться в открытые полосы движения. Последние разработки в области автоматизированного вождения, такие как адаптивный круиз-контроль, который позволяет согласовывать вашу скорость с движущимися впереди автомобилями, позволяют транспортным средствам двигаться с высокой скоростью, сохраняя небольшую дистанцию. Но при этом не используется знание о транспортных средствах на других полосах движения или сзади. В будущем автомобили смогут получать доступ к большему объему локальной информации, что потенциально позволит транспортным средствам взаимодействовать друг с другом. Если транспортные средства могут взаимодействовать друг с другом, вы можете использовать RL для выработки глобальной оптимальной политики для увеличения скорости и плотности, чтобы уменьшить заторы и количество вторичных аварий.

Рен, Се и Цзян разработали RL-подход к решению этой проблемы. Сначала они разбивают дорогу на зоны, одна из которых является зоной слияния, где автомобили вынуждены перестроиться в открытую полосу движения. Предыдущая зона позволяет автомобилям ускоряться или замедляться, чтобы они могли выровнять скорость с автомобилями, движущимися по открытой полосе [8].

Состояние марковского процесса принятия решений (MDP) — это матрица скорости и ускорения в этой зоне, а также локальная информация об управляемом транс-

портном средстве. Эта матрица вводится в довольно сложную нейронную сеть, которая включает сверточные, полносвязные и повторяющиеся элементы.

Действие заключается в управлении скоростью отдельного транспортного средства, чтобы поместить его в безопасный промежуток. Авторы подхода перебирают все транспортные средства, находящиеся в настоящее время в зоне контроля.

Награда сложная, но направлена на содействие успешным слияниям (без сбоев и с безопасным расстоянием) и увеличение скорости.

Исследователи выбрали SAC в качестве алгоритма из-за непрерывных действий, его эффективности выборки вне политики и его стабильности во время разведки.

Они смоделировали эту формулировку в инструменте моделирования дорожного движения под названием VISSIM[2] и сравнили реализацию RL с распространенными стратегиями слияния на основе дорожных знаков, которые сообщают транспортным средствам, что они должны выехать на открытую полосу раньше или позже. Решение RL значительно улучшает мобильность по сравнению с базовыми алгоритмами, на порядок уменьшая задержки. Такой подход также намного безопаснее, т. к. имеет стабильные низкие значения плотности по всей очереди, в то время как базовые алгоритмы способствуют возникновению опасной обратной ударной волны.

Несмотря на впечатляющие результаты, есть еще много возможностей для улучшения. Во-первых, это симуляция. Было бы здорово получить некоторые реальные результаты, хотя по опыту я знаю, что невероятно сложно экспериментировать в потенциально опасных средах, подобных этой. Также хотелось бы видеть более реалистичные наблюдения. Например, положение автомобиля и местная информация могут быть получены с видеокамер. Сомнительно, что транспортные средства или оракул в совершенстве знают всё о машинах вокруг себя. Наконец, меня беспокоит сложность модели политики для довольно простого пространства состояний. Я не считаю, что требуется такая сложность, а государственное управление на дорогах следует делегировать. Уверен, что сама политика по выбору правильной скорости для преодоления промежутка на самом деле довольно проста для освоения.

# Расширения методов максимальной энтропии

SAC предоставляет простой и производительный алгоритм вне политики прямо из коробки, но, как и для других алгоритмов, исследователи продолжают предлагать улучшения в поисках максимальной производительности. Как обычно, эти модификации не гарантируют лучшей производительности для вашей конкретной задачи; вы должны их протестировать.

## Другие меры энтропии (и ансамбли)

Энтропия Шеннона — только один из способов измерения энтропии. Существуют и другие формы, изменяющие способ, которым SAC наказывает политику, предпо-

[2] См. **https://oreil.ly/48EpL**.

читая или игнорируя бесперспективные действия. Чен и Пэн опробовали энтропию Цаллиса и Реньи и обнаружили, что производительность базовых алгоритмов действительно улучшена [9]. Они также продемонстрировали использование ансамбля политик, при котором несколько политик обучаются одновременно. При каждом новом эпизоде агент использует случайную политику из этого набора для выбора действий. Данный подход исходит из бутстрэппированной глубокой Q-сети с целью улучшения исследования (*см. разд. "Улучшение исследования" главы 4*).

## Оптимистичное исследование с использованием верхней границы двойного Q-обучения

Подобно тому, как двойное Q-обучение формулирует нижнюю границу оценки ценности действия (*см. разд. "Двойное Q-обучение" главы 3*), Чосек и соавт. предлагают использовать для формирования политики поведения не пессимистическую границу в качестве целевой политики (с членом энтропии, который делает ее похожей на стандартный SAC), а *верхнюю границу*, стоимость действия, которое переоценивает вознаграждение [10]. Они предполагают, что это поможет предотвратить *пессимистическое недооценивание*, вызванное опорой на нижнюю границу. Эмпирически результаты в стандартных средах MuJoCo лучше, но незначительно. Возможно, вам стоит рассмотреть этот подход, если вы работаете в среде, где агентам сложно изучить все возможности.

## Играем с воспроизведением опыта

Как описано в *разд. "Воспроизведение приоритетного опыта" главы 4*, опыт, на котором вы хотите учиться, влияет на обучение. Ванг и Росс демонстрируют, что добавление приоритетного воспроизведения опыта может улучшить алгоритм SAC в некоторых средах, а передискретизация более свежего опыта стабильно улучшает результаты во всех средах [11]. Это простое дополнение, и его стоит попробовать, но остерегайтесь катастрофического забывания (*см. разд. "Воспроизведение опыта" главы 4*), в результате чего агент забывает предыдущий опыт. Большой буфер воспроизведения смягчает этот эффект.

## Мягкий градиент политики

Вместо того чтобы наказывать мерой энтропии, почему бы не получить основанный на энтропии мягкий градиент политики? Ши, Сонг и Ву представили метод мягкого градиента политики, который, как сообщается, является более принципиальным и стабильным, чем SAC [12].

Ши и соавт. начали с определения максимальной энтропии RL и дифференцировали политику, чтобы получить формулу вычисления градиента, которая на удивление проста (уравнение 7.8). Использование этой формулы более эффективно, чем фрейморков автоматического дифференцирования, поскольку вы можете одновременно проводить разведку и работать непосредственно с градиентами; поэтому авторы подхода решили обрезать градиенты напрямую, а не применять оптимизацию доверительной области, чтобы предотвратить катастрофические обновления. Всё

это приводит к более простой реализации и лучшей стабильности. Я не удивлюсь, если в будущем такой подход станет интерпретацией максимальной энтропии "по умолчанию".

**Уравнение 7.8. Глубокий мягкий градиент политики**

$$\nabla J_{\pi}(\varphi) = \mathbb{E}\left[\left(Q_{\theta}(s,a) - \log \pi(a \mid s) - 1\right)\nabla_{\varphi} \log \pi(a \mid s)\right].$$

## Мягкое Q-обучение (и производные)

Вы также можете применить максимальную энтропию RL к Q-обучению. Хаарноя и соавт. впервые показали, что вы можете получить основанные на энтропии оценки ценности и действий [13]. Другие исследователи применили ту же идею для вычисления функции преимущества и обнаружили, что она хорошо работает в сложных, потенциально непрерывных средах с дискретными пространствами действий [14]. Я могу представить, что интеграция этих оценок в другие алгоритмы, такие как PPO, может дать положительные результаты.

## Обучение согласованности пути

*Обучение согласованности пути* (path consistency learning, PCL) от Нахума и соавт. связано с SAC, потому что он заточен на то, что энтропия стимулирует исследование и порождает те же функции мягкого состояния и "действия — стоимости", что и мягкое Q-обучение. Однако PCL исследует использование развертываний в стиле *n*-шаговых алгоритмов и приводит к *n*-ступенчатому алгоритму вне политики, который не нуждается в каких-либо исправлениях (например, выборка по значимости) [15]. В другом исследовании реализовано KL-расхождение — метод доверия (trust-PCL), который превосходит TRPO [16]. Таким образом, даже с помощью методов глубокого обучения и выборки вне политики вы все равно можете внести улучшения в базовые алгоритмы, учитывающие временны́е различия. Несмотря на это, SAC, как правило, значительно превосходит методы PCL.

# Сравнение производительности: SAC против PPO

В комнате есть слон, и он беснуется по всему RL. Какой алгоритм лучше для задач непрерывного контроля: PPO или SAC? Принято считать, что SAC и его производные работают лучше, чем PPO. Но не все так однозначно, как хотелось бы. Как я упоминал ранее, что исследователи измеряют производительность по-разному, и детали реализации сильно влияют на производительность [17]. Основная проблема этого раздела заключалась в том, что исследователи в большинстве случаев не публикуют исходные значения для кривых обучения. И часто даже не публикуют таблицы данных, демонстрируя лишь графики. Это делает невероятно трудным сравнение результатов между статьями.

В табл. 7.1 приведена выборка результатов, полученных в ходе работ и реализаций. Все результаты были получены на 1 млн шагов, за исключением отмеченных символом "†" с 1,5 млн шагов. Результаты, отмеченные символом "*", оценивалась по опубликованным изображениям.

***Таблица 7.1.*** *Результаты тестов производительности непрерывного контроля для SAC и PPO*

| Environment | SAC[1] | SAC[2] | SAC†[3] | SACauto*[4] | PPO[5] | PPO†[6] | SACauto[7] | PPO[7] |
|---|---|---|---|---|---|---|---|---|
| MuJoCo Walker | 3475 | 3941 | 3868 | 4800 | 3425 | 3292 | | |
| MuJoCo Hopper | 2101 | 3020 | 2922 | 3200 | 2316 | 2513 | | |
| MuJoCo Half-cheetah | 8896 | 6095 | 10 994 | 11 000 | 1669 | | | |
| MuJoCo Humanoid | 4831 | | 5723 | 5000 | 4900 | 806 | | |
| MuJoCo Ant | 3250 | 2989 | 3856 | 4100 | | | | |
| MuJoCo Swimmer | | | 42 | | 111 | | | |
| AntBulletEnv-v0 | | | | | | | 3485 | 2170 |
| BipedalWalker-v2 | | | | | | | 307 | 266 |
| BipedalWalkerHardcore-v2 | | | | | | | 101 | 166 |
| HalfCheetahBulletEnv-v0 | | | | | | | 3331 | 3195 |
| HopperBulletEnv-v0 | | | | | | | 2438 | 1945 |
| HumanoidBulletEnv-v0 | | | | | | | 2048 | 1286 |
| InvertedDoublePendulumBulletEnv-v0 | | | | | | | 9357 | 7703 |
| InvertedPendulum-SwingupBulletEnv-v0 | | | | | | | 892 | 867 |
| LunarLanderContinuous-v2 | | | | | | | 270 | 128 |
| MountainCarContinuous-v0 | | | | | | | 90 | 92 |
| Pendulum-v0 | | | | | | | –160 | –168 |
| ReacherBulletEnv-v0 | | | | | | | 18 | 18 |
| Walker2DBulletEnv-v0 | | | | | | | 2053 | 2080 |

[1] Haarnoja T , Zhou A , Abbeel P , Levine S Soft actor-critic off-policy maximum entropy deep reinforcement learning with a stochastic actor // ArXiv 1801.01290 — 2018 — August — URL **https://oreil.ly/JjjwB**

[2] Wang T , Bao X , Clavera I et al Benchmarking model-based reinforcement learning // ArXiv 1907 02057 — 2019 — July — URL **https://oreil.ly/9ZBoU**

[3] Wang C , Ross K Boosting Soft actor-critic emphasizing recent experience without forgetting the past // ArXiv·1906 04009. — 2019 — June. — URL· **https://oreil.ly/eigQt**

[4] Haarnoja T , Zhou A , Hartikainen K et al Soft actor-critic algorithms and applications // ArXiv·1812 05905. — 2019 — January. — URL· **https://oreil.ly/s3apP**

[5] Dhariwal P , Hesse C , Klimov O et al OpenAI Baselines GitHub repository — 2017 — URL· **https://oreil.ly/9642Z**

[6] Logan E et al Implementation matters in deep policy gradients a case study on PPO and TRPO // ArXiv 2005 12729 — 2020 — May — URL **https://oreil.ly/pOPdK**

[7] Raffin A (2018) 2020 Araffin/RI-Baselines-Zoo Python — 2018 — URL· **https://oreil.ly/H3Nc2**

Первое, что вы должны заметить, — это разброс результатов. Некоторые реализации и эксперименты SAC работают намного хуже, чем другие. Вероятно, худшая настройка гиперпараметров привела к снижению производительности. Вот почему так важен автоматический температурный вариант SAC. Он дает хорошие результаты без настройки. Если вы просмотрите все числа, то обнаружите, что SAC в основном превосходит PPO в средах MuJoCo. Глядя на результаты модуля PyBullet с открытым исходным кодом, можно также предположить, что автоматический температурный SAC в целом лучше, но есть много сред, в которых производительность аналогична.

Приведенные данные не отражают всей картины. Непонятно, будут ли расти ваши результаты, если вы оставите обучение своих моделей. Представленные данные демонстрируют среднюю производительность, которая не говорит, насколько стабильными были прогоны во время обучения. Как я уже рекомендовал несколько раз, всегда проверяйте алгоритмы и реализации самостоятельно для решения ваших конкретных задач. Ваши результаты могут не соответствовать опубликованным базовым показателям.

## Как энтропия способствует исследованиям?

Обучение с максимальным энтропийным подкреплением поощряет старание, но как? Уравнение 7.2 добавляет бонус, пропорциональный энтропии действий в состоянии, а это означает, что агент получит искусственное вознаграждение, превышающее фактическое вознаграждение. Размер дополнительного вознаграждения определяется значением температуры и энтропией действий.

Представьте себе ребенка в кондитерской. Вы можете измерить ожидание конфет с помощью энтропии; больше вариантов — больше удовольствия. Посреди магазина ребенок мог бы двигаться в сотнях разных направлений и получить новую конфету. Но у дверей, прямо на краю, есть ограниченное число вариантов на выбор. Энтропия предполагает, что у ребенка будет больше вариантов, если он пойдет к центру магазина.

Комбинируя награду с бонусом энтропии, вы продвигаете действия, которые не являются строго оптимальными. Это помогает обеспечить адекватную выборку всех состояний.

Для демонстрации я создал простую сетку с целью в правом верхнем углу и начальной точкой посередине. Агент может двигаться в любом направлении (включая диагонали), и есть скудная награда в размере 1, когда агент достигает цели. Эпизод заканчивается нулевым вознаграждением, если агенты не достигают цели за 100 шагов.

Я использовал ε-жадный агент Q-обучения в качестве основы. На рис. 7.1 показан результат запуска агента со значениями действий, отображенными цветных метках; более светлые цвета представляют высокие значения действия, более темные — низкие. Стрелками отмечена оптимальная траектория (без исследования) для политики. Вы можете видеть, что агент нашел путь к цели, но он неоптимален с точки

зрения количества шагов (помните, я не штрафую за шаги, и исследование зависит от случайной начальной точки), а агент не исследовал другие регионы области. Если вы измените начальную локацию агента, он не сможет найти путь к цели. Это ненадежная политика.

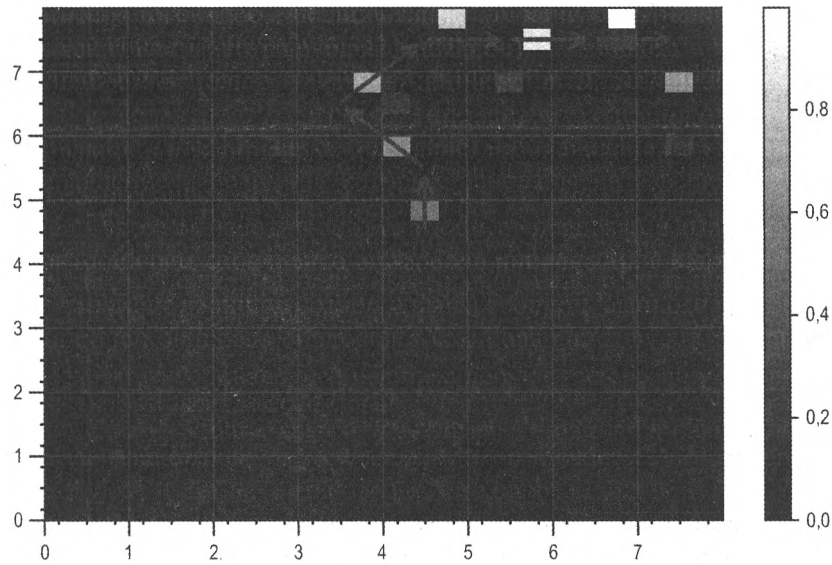



**Рис. 7.1.** Результаты использования ε-жадного агента Q-обучения в распределенной среде. Цветные метки (прямоугольники) отражают ценности действий значения действий, а сетка — состояние. Стрелки — оптимальная траектория согласно текущей политике

Существует одно простое усовершенствование, которое объясняет бóльшую часть производительности SAC, — это использование целевых значений в качестве распределения вместо того, чтобы полагаться на случайную выборку ε-жадного агента для исследования. Когда этот грубый вероятностный подход применяется к алгоритму Q-обучения с использованием функции softmax для преобразования ценности действий в вероятности, исследование значительно улучшается (рис. 7.2). Я также изобразил наиболее вероятное действие в виде стрелок для каждого состояния, направление и величина которых основаны на взвешенной комбинации значений действия. Обратите внимание, как все стрелки направления отмечают оптимальную траекторию, поэтому независимо от того, откуда агент стартует, он всегда будет достигать цели. Эта политика гораздо надежнее и исследовала больше пространства состояний, чем предыдущий агент.

Эффективность надежной политики должна постепенно ухудшаться при наличии неблагоприятных факторов. Политики, у которых есть неизведанные области, близкие к оптимальной траектории, не являются надежными, потому что малейшие отклонения могут привести к состояниям с неопределенной политикой.

Затем я реализовал мягкое Q-обучение (*см. разд. "Мягкое Q-обучение (и производные)" ранее в данной главе*) и установил параметр температуры на 0,05. Вы можете

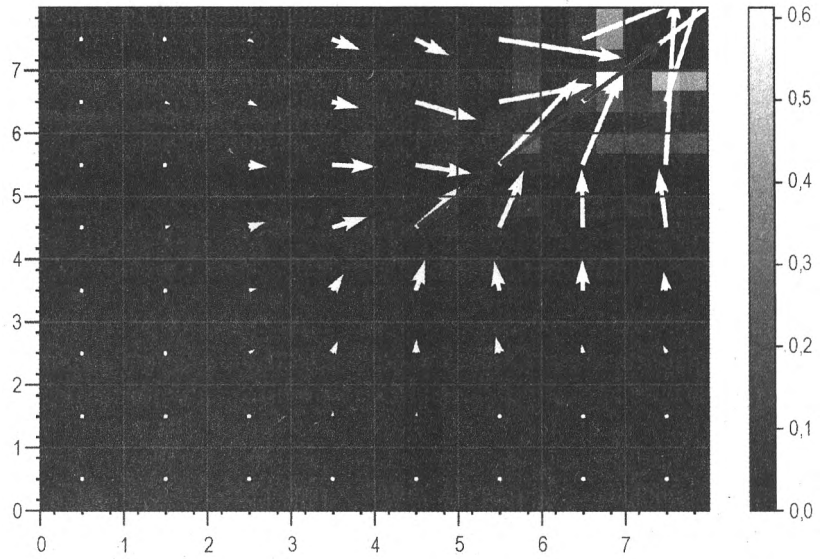



**Рис. 7.2.** Результаты использования грубого вероятностного агента Q-обучения на основе функции softmax в распределенной среде. Условия такие же, как на рис. 7.1, за исключением того, что положения стрелок отражают наиболее вероятное действие с направлением и величиной, взвешенными по ценности действия

увидеть результаты на рис. 7.3. Оптимальные стратегии ничем не примечательны, хотя в целом они указывают в правильном направлении. Но если вы присмотритесь, то увидите, что значения действий и стрелки направления меньше или отсутствуют на графике грубого Q-обучения. Мягкое Q-обучение посетило состояния, отличные от оптимальной траектории. Вы также можете видеть, что белые цвета, представляющие бóльшие значения действия, смещены к центру изображений для

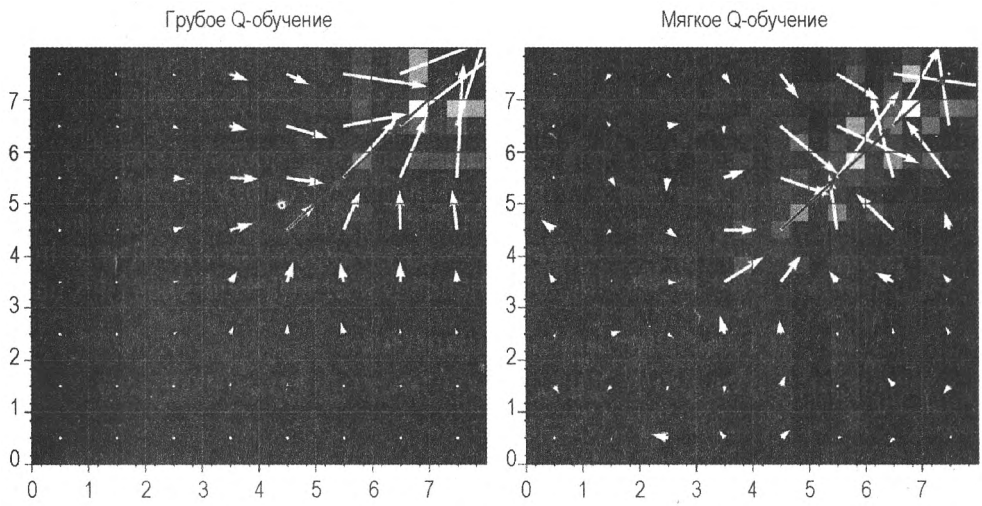



**Рис. 7.3.** Сравнение грубого вероятностного Q-обучения с мягким Q-обучением с той же настройкой, что и на рис. 7.2

мягкого Q-обучения. Причина объясняется на примере ребенка в кондитерской: энтропия способствует переходу к состояниям с наибольшим выбором.

Отсутствие исследования в агенте Q-обучения становится еще более заметным, если вы отслеживаете, сколько раз каждый агент посещает каждое действие, как я это сделал на рис. 7.4. Вы можете видеть, что агент мягкого Q-обучения более равномерно выбирает состояния. Вот почему энтропия помогает исследовать более сложные пространства состояний.

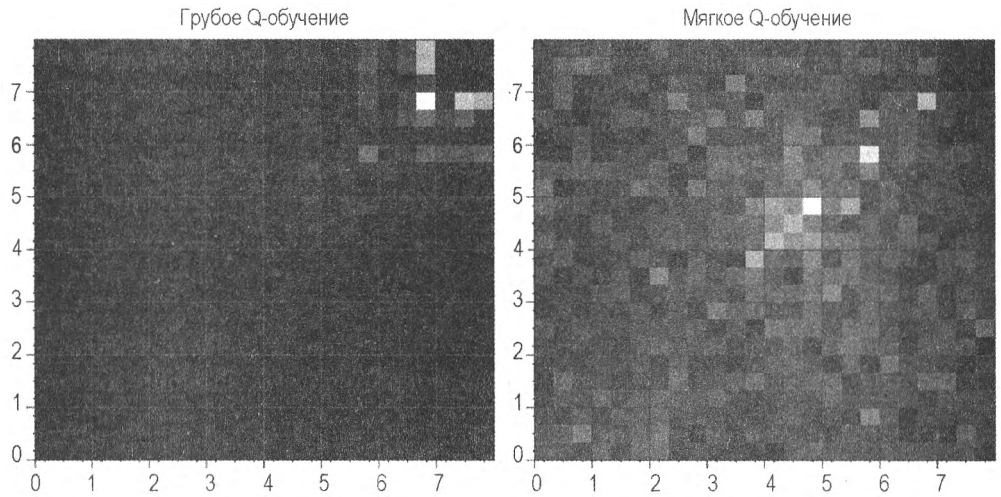



**Рис. 7.4.** Количество взаимодействий "действие — ценность" для обоих агентов. Цвета показывают, сколько раз агент посещал каждое действие

## Как температурный параметр влияет на исследование?

Мягкое Q-обучение (soft Q-learning, SQL), или алгоритм SAC, очень зависит от температурного параметра. Он контролирует объем исследования, и если вы слишком высоко оцените энтропию, агент получит более высокое вознаграждение за исследование, чем за достижение цели!

Есть два решения этой проблемы. Самый простой — *отжиг*. Снижение температурного параметра заставляет агента исследовать на ранней стадии, но позже становиться детерминированным. Второй подход заключается в автоматическом изучении параметра температуры, который используется в большинстве реализаций SAC.

В эксперименте на рис. 7.5 я увеличиваю значение температуры, чтобы способствовать дальнейшим исследованиям. Как вы можете видеть из величины ценности действия, член энтропии теперь настолько велик, что превышает вознаграждение (со значением 1). Итак, оптимальная политика — двигаться к состояниям с самой высокой энтропией, т. е. к состояниям посередине, как в примере с ребенком в кондитерской.

Для того чтобы справиться с этой проблемой, вы можете со временем снизить температуру, чтобы направить ребенка, я имею в виду агента, к цели. В эксперименте, показанном на рис. 7.6, я отжигаю (снижаю) температурный параметр от 0,5 до 0,005. Обратите внимание, что и в исходной статье SAC использовался отжиг.

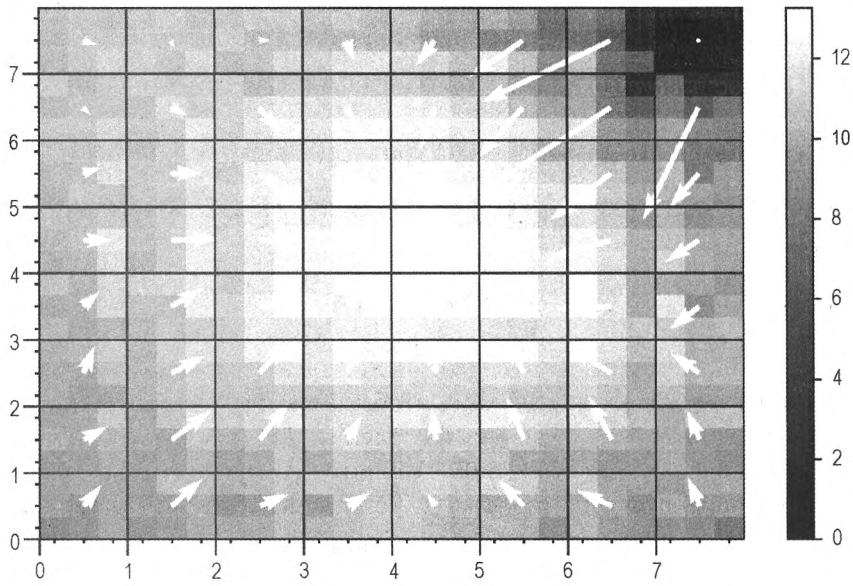



**Рис. 7.5.** Установка параметра температуры на 1 в агенте мягкого Q-обучения. Для того чтобы подчеркнуть эффект, я увеличил количество серий до 500. Все остальные настройки такие, как на рис. 7.3

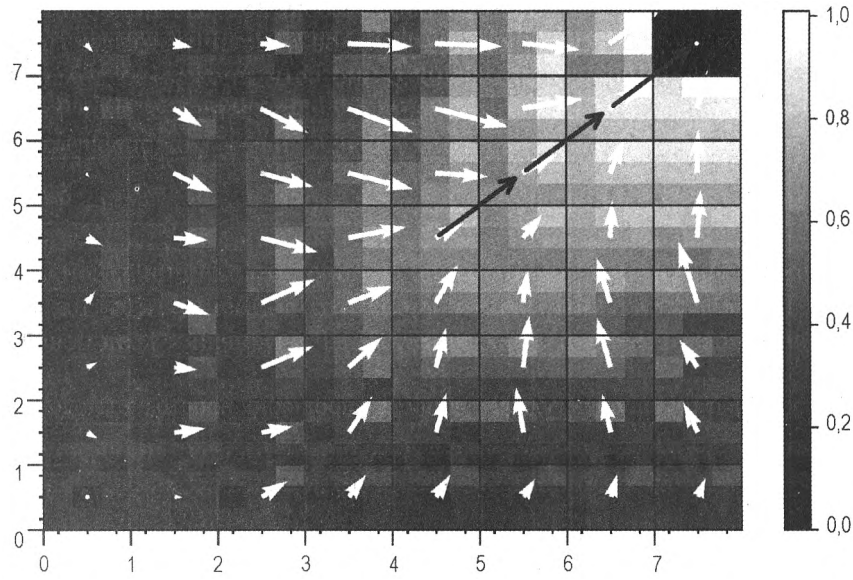



**Рис. 7.6.** Отжиг по температурному параметру от 0,5 до 0,005 в мягком агенте Q-обучения. Все остальные настройки соответствуют рис. 7.5

Этот эксперимент демонстрирует, как отжиг помогает агенту исследовать ранние эпизоды. Это чрезвычайно полезно в сложных средах, требующих большого количества исследований. Изменение температуры приводит к тому, что агент становится более похожим на Q-обучение и придерживается оптимальной политики.

Однако важны параметры скорости запуска, остановки и отжига. Вам нужно тщательно выбирать их в соответствии с вашей средой и сочетать надежность с оптимальностью.

# Отраслевой пример: обучение вождению автомобиля с дистанционным управлением

Проект Donkey Car[3] начал свою жизнь как радиоуправляемая машина с открытым исходным кодом на базе одноплатного компьютера Raspberry Pi с единственным датчиком — широкоугольной видеокамерой. Все управление осуществляется с помощью одного изображения, что создает значительные проблемы и возможности для RL. Название Donkey было выбрано, чтобы оправдать низкие ожидания. Но не позволяйте названию вводить вас в заблуждение; это простое оборудование воплощено в жизнь с помощью передового машинного обучения. Взгляните на некоторые впечатляющие результаты и узнайте больше на веб-сайте Donkey Car[4].

Таун Крамер, энтузиаст Donkey Car, разработал среду моделирования на основе фреймворка Unity [18]. Вы можете использовать его как платформу быстрой разработки для создания новых алгоритмов самоуправления. После того как меня вдохновил Антонин Раффин, один из основных разработчиков ряда библиотек Stable Baselines, я решил попробовать сам [19]. В этом примере я демонстрирую, что подход RL на основе SAC генерирует приемлемую политику.

## Описание задачи

Состояние представлено наблюдением с помощью видеокамеры. Видео имеет разрешение 160 × 120 с 8-битными цветовыми каналами RGB. Пространство действий обманчиво простое, позволяя учитывать величину давления на педаль газа и угол поворота рулевого колеса. Среда вознаграждает агента за скорость и нахождение в центре полосы движения на каждом временно́м шаге. Если автомобиль выходит слишком далеко за пределы своей полосы движения, он получает отрицательную награду (штрафуется). Цель состоит в том, чтобы добраться до конца дистанции как можно быстрее.

Есть несколько треков. Самый простой — это пустынная дорога с четко обозначенными и последовательными полосами. Самым сложным является симулятор городских гонок, где агенту предстоит преодолевать препятствия и трассу без четких границ.

---

[3] Ослиная повозка.

[4] См. **https://www.donkeycar.com/**.

Моделирование в Unity поставляется в виде *бинарного файла*[5]. Среда Gym подключается к моделированию через веб-сайт и поддерживает несколько подключений, что позволяет проводить соревновательные гонки с участием RL.

## Минимизация времени обучения

Всякий раз, когда вы работаете с данными, основанными на пиксельной информации, большую часть времени обучения занимает извлечение свойств. Ключ к быстрому развитию такого проекта — разделить задачу; чем на большее число частей вы сможете разбить ее, тем быстрее и легче можно будет улучшать отдельные компоненты. Да, можно узнать приемлемую политику непосредственно из пикселов, но это замедляет итерации разработки и требует больших денежных затрат. Ваш кошелек будет вам благодарен.

Одно из очевидных мест для старта — это отделить часть процесса, связанную с визуальной информацией. Агенту RL все равно, является ли состояние плоским изображением или набором высокоуровневых функций. Но на производительность влияет то, насколько информативны функции; более информативные функции приводят к более быстрому обучению и более надежным политикам.

Один из подходов компьютерного зрения может заключаться в попытке сегментировать изображение или определить границы дороги. Использование преобразования перспективы может компенсировать прямой вид на дорогу. Алгоритм сегментации или обученная модель сегментации способны помочь вам выбрать дорогу. Вы можете использовать преобразование Хафа[6] или Радона[7], чтобы найти линии на изображении, и использовать физические предположения, чтобы найти ближайшие полосы. Вы, наверняка, можете еще что-то придумать. Помните, что цель состоит в том, чтобы сделать данные наблюдений более информативными.

Другой (симбиотический) подход — обучить автоэнкодер (*см. разд. "Архитектуры нейронных сетей" главы 4*), чтобы ограничить количество функций теми, которые являются наиболее информативными. Этот вариант заключается в использовании сверточной нейронной сети, имеющей форму песочных часов для разложения и восстановления изображений из набора данных до тех пор, пока небольшое количество нейронов в середине не будет надежно кодировать всю необходимую информацию об изображении. Это автономный процесс, независимый от RL. Предварительно обученное низкоразмерное представление затем используется в качестве наблюдения для RL.

На рис. 7.7 и 7.8 показаны исходное наблюдение и реконструкция соответственно после обучения вариационного автоэнкодера для 1000 шагов обучения. Обратите внимание на расплывчатое сходство на этом этапе.

---

[5] См **https://oreil.ly/2B3Nl**.

[6] Преобразование Хафа (Hough transform) — вычислительный алгоритм, применяемый для параметрической идентификации геометрических элементов растрового изображения. — *Прим. ред.*

[7] Преобразование Радона — интегральное преобразование функции многих переменных, родственное преобразованию Фурье. — *Прим. ред.*

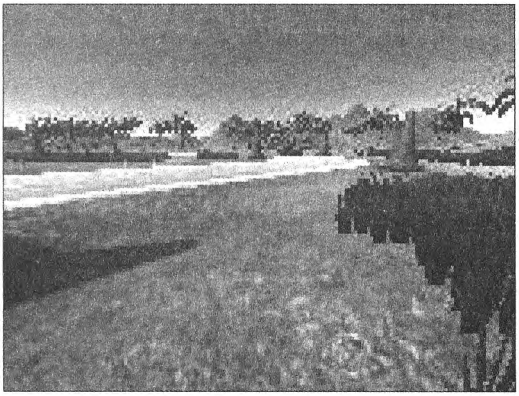


**Рис. 7.7.** Необработанное наблюдение за средой Donkey Car на 1000 тренировочных шагов

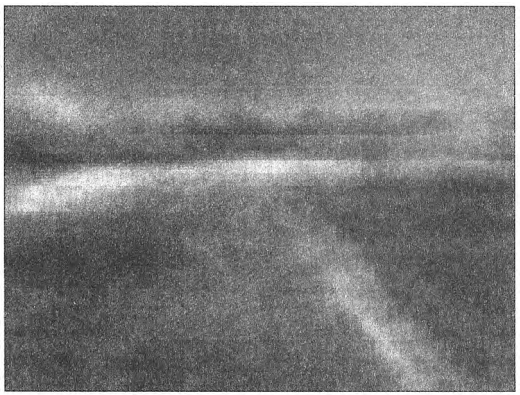


**Рис. 7.8.** Реконструкция среды Donkey Car на 1000 тренировочных шагов с помощью вариационного автоэнкодера

На рис. 7.9 и 7.10 показано другое наблюдение и реконструкция после 10 000 тренировочных шагов. Обратите внимание, сколько деталей теперь в реконструкции, и это указывает на то, что встраивание скрытой нейронной сети усвоило хорошие скрытые функции.

**Рис. 7.9.** Необработанное наблюдение за средой Donkey Car на 10 000 тренировочных шагов

**Рис. 7.10.** Реконструкция среды Donkey Car на 10 000 тренировочных шагов VAE

На рис. 7.11 сравниваются видеокадры наблюдения с камеры RGB на платформе Donkey Car с реконструкцией низкоразмерного наблюдения, которое является входом для алгоритма SAC.

Сокращение пространства состояний, входящего в алгоритм обучения с подкреплением, сократит время обучения RL до нескольких минут, а не дней. Подробнее об этом я рассказываю в *разд. "Преобразование (уменьшение размерности, автэнкодеры и модели мира)" главы 9.*

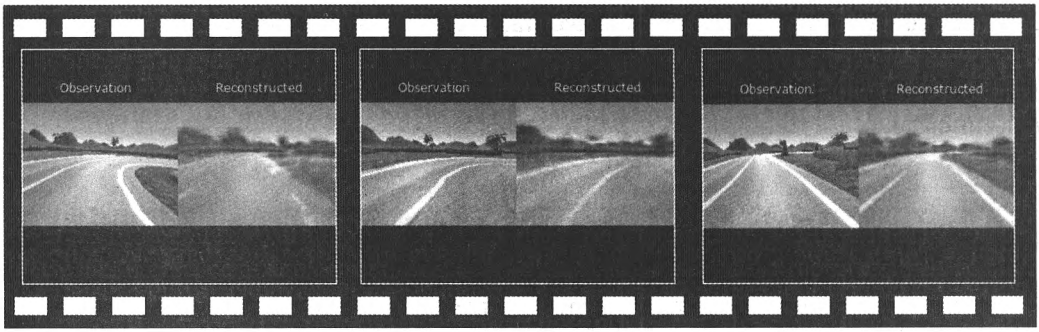



**Рис. 7.11.** Реконструкция симуляции Donkey Car глазами вариационного автоэнкодера. Картинка слева показывает кадр исходного видео из моделирования. Справа — восстановленный видеокадр из низкоразмерного состояния. Восстановленное видео было обрезано для удаления фрагмента неба, в котором нет необходимости

## Выразительные действия

В некоторых средах можно применять действия, которые резко изменяют среду. В случае Donkey Car большие значения для действия рулевого управления приводят к тому, что автомобиль разворачивает, заносит и иным образом отклоняет от оптимальной траектории. Но, конечно, автомобилю нужно поворачивать, а иногда и проходить крутые повороты.

Эти действия и последующие штрафы могут привести к тому, что обучение остановится, потому что машина будет постоянно застревать, и ей не удастся зафиксировать достаточно хороших наблюдений. В среде Donkey Car этот эффект усиливается, когда допускаются большие скачки в действиях. Например, когда при езде на велосипеде вы знаете, что технически возможно резко повернуть руль на скорости, но я сомневаюсь, что вы когда-либо пробовали это делать. Вы, вероятно, пробовали что-то гораздо менее энергичное, и у вас был неудачный опыт, поэтому вы никогда не будете пробовать ничего вроде поворота рулевого управления на 90°. На рис. 7.12 показан пример обучающего материала с исходным пространством действий.

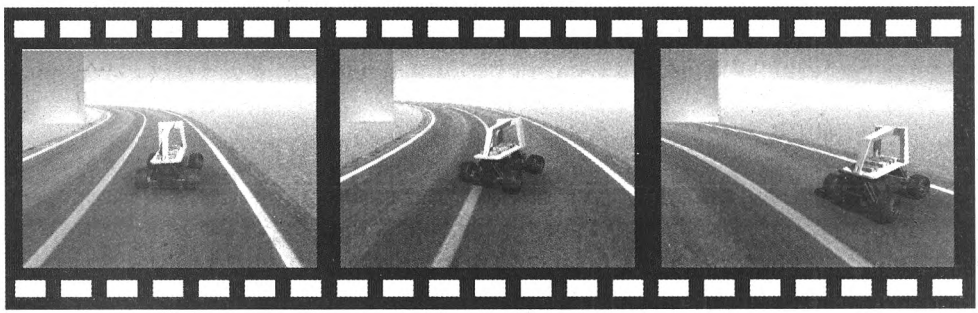


**Рис. 7.12.** Видеокадры обучения Donkey Car, когда пространство действия остается неизменным

Вы можете повлиять на обучение агента, включив эту первичную информацию в свою модель RL. Можете изменить сигнал вознаграждения, чтобы повысить (или понизить) определенные состояния. Или, как в этом случае, можете отфильтровать действие агента, чтобы заставить его вести себя более безопасным образом.

Если вы попытаетесь обучить агента, используя полный диапазон углов поворота (попробуйте!), то увидите, что через довольно большое количество времени он научится держаться подальше от краев, но делает это так упорно, что отправляет автомобиль в занос. Несмотря на то что всё выглядит круто, это не помогает закончить курс. Вам нужно сгладить рулевое управление.

Один из простейших способов ограничить поведение агента — изменить пространство действий. Как и в случае с машинным обучением, ограничение задачи упрощает ее решение. В обучении, показанном на рис. 7.13, я ограничил газ максимумом 0,15 (первоначально 5,0) и угол поворота до абсолютного максимума 0,5 (первоначально 1,0).

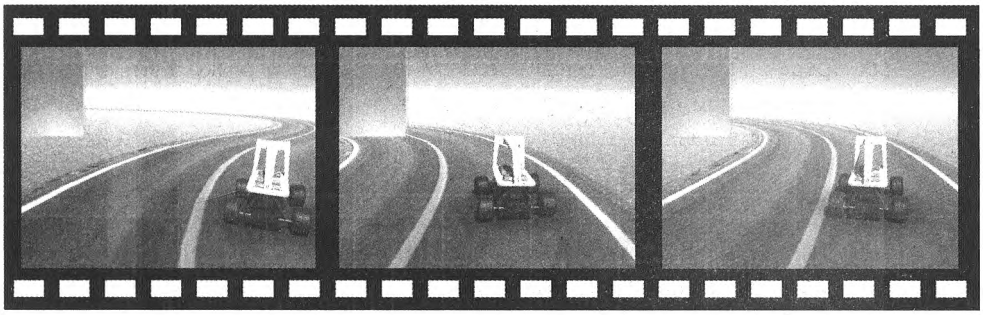


**Рис. 7.13.** Видео тренировки Donkey Car после упрощения пространства действий

Низкая скорость предотвращает занос автомобиля, а меньший угол поворота рулевого колеса помогает смягчить рывки. Обратите внимание, что эти параметры сильно влияют на эффективность обучения. Попробуйте сами их изменить.

## Поиск гиперпараметров

Максимальные настройки газа и рулевого управления — гиперпараметры задачи; они не настраиваются напрямую. Изменения в определении задачи, как правило, больше всего влияют на производительность, но есть множество других гиперпараметров из алгоритма SAC, таких как архитектура модели политики. Цель состоит в том, чтобы найти такую комбинацию параметров, которая приводит к лучшим наградам. Для этого вы должны выполнить поиск по гиперпараметрам, написав код, который выполняет итерацию по всем перестановкам и повторно запускает обучение. Подробности о настройке гиперпараметров *см. в разд. "Настройка гиперпараметров" главы 10.*

Я провел оптимизированный поиск по сетке, используя библиотеку optuna, чтобы настроить гиперпараметры алгоритма SAC, и просмотрел все результаты в TensorBoard. Я обнаружил, что могу увеличить скорость обучения на порядок.

Также помогла настройка коэффициента энтропии; по умолчанию реализация стабильных базовых показателей начинается со значения 1,0 и определяет оптимальное значение. Но в этом примере агент обучается небольшое количество шагов, поэтому у него нет времени на то, чтобы узнать оптимальный коэффициент. Начиная с 0,1, изучение оптимального коэффициента работало лучше. Но в целом меня не впечатлило, насколько гиперпараметры влияют на производительность (и это хорошо!). Изменение гиперпараметров вариационного автоэнкодера и изменение пространства действий повлияло на производительность гораздо больше.

## Финальная политика

После всей инженерной работы, всего кода и поиска гиперпараметров у меня теперь есть приемлемая политика, которая была обучена от 5000 до 50 000 шагов среды, в зависимости от самой среды и того, насколько идеальной должна быть политика. На моем старом Macbook Pro 2014 г. это заняло 10–30 минут. На рис. 7.14 и 7.15 показаны примеры политик для сред donkey-generated-track-v0 и donkey-generated-road-v0 соответственно.

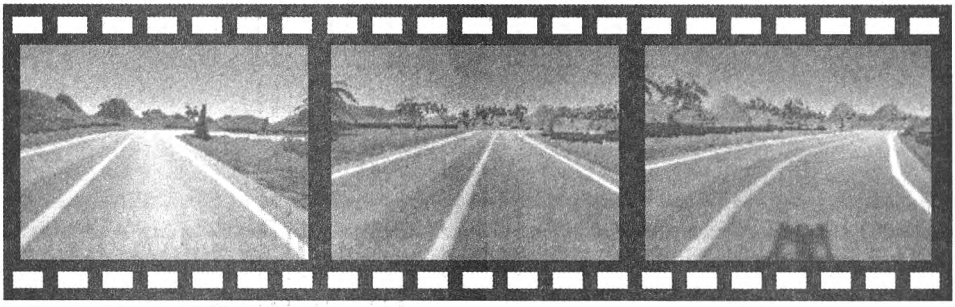


**Рис. 7.14.** Видеокадры финальной политики Donkey Car для среды donkey-generated-track-v0

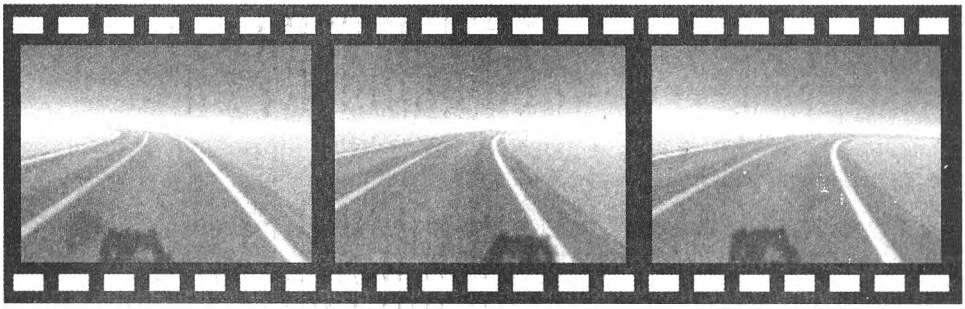


**Рис. 7.15.** Видеокадры с финальной политикой Donkey Car для среды donkey-generated-road-v0

## Дальнейшие улучшения

В эту политику можно внести множество улучшений. К тому времени, как вы прочтете эти слова, другие инженеры превзойдут мои небольшие достижения. Вы мо-

жете потратить свое время на улучшение автокодирования или попробовать воплотить любые другие идеи, предложенные в *разд. "Минимизация времени обучения" ранее в этой главе*. Вы можете изменить пространство действий, например увеличивая максимальное количество действий, пока это пространство будет выдерживать. Одна вещь, которую я хотел попробовать, — это отжиг ограничения пространства действия, при котором пространство действия начинается с малого и увеличивается до максимальных значений за время обучения. Вы также можете попробовать изменить награды; может быть, скорость не стоит так высоко ценить. Также имеет смысл пытаться наказывать дерганье.

Вы, вероятно, можете вообразить гораздо больше, но если вы работаете над промышленной проблемой, вам нужно будет проявить больше осторожности: сделайте маленькие шаги, убедитесь, что вы даете правильную оценку, удостоверьтесь, что то, что вы делаете, безопасно. Разделите задачу на части, ограничьте проблему и действуйте дальше. Подробнее об этом можно прочитать в *главе 9*.

# Резюме

RL стремится найти политику, которая максимизирует ожидаемую доходность от выбранных состояний, действий и вознаграждений. Это определение подразумевает максимизацию ожидаемой доходности независимо от последствий. Исследования показали, что ограничение обновлений политики приводит к более стабильному обучению и более гибким политикам.

В *главе 6* эта проблема решена путем ограничения обновлений, чтобы они не меняли кардинально политику. Но это все равно может привести к хрупкой, слишком разреженной политике, когда большинство действий имеют нулевую ценность, за исключением оптимальной траектории. В этой главе были представлены алгоритмы на основе энтропии, которые изменяют исходное определение возвращаемых значений марковского процесса принятия решений (MDP) для масштабирования или *регуляризации* доходов. Это привело к предположению, что другие определения энтропии могут работать лучше, потому что энтропия Шеннона недостаточно разрежена. Я бы сказал, что можно настроить параметр температуры, чтобы выработать политику, которая соответствует вашей проблеме. Но вполне вероятно, что другая мера энтропии будет работать лучше для вашей конкретной проблемы.

Логический вывод этого аргумента — использовать произвольную функцию. Такой подход был предложен Ли, Яном и Чжаном в *регуляризованных* MDP, предписывающих несколько теоретических правил, которым должна следовать функция, например, что она должна быть строго положительной, дифференцируемой, вогнутой и падать до нуля за пределами диапазона от нуля до единицы. Они предполагают, что другие экспоненциальные и тригонометрические функции являются интересными предложениями [20].

Независимо от реализации, увеличение функции ценности за счет бонуса за разведку — хорошее решение для производственных реализаций. Вы же хотите, чтобы

ваш агент был устойчивым к изменениям. Будьте осторожны, чтобы не сделать его приспособленным только к работе в вашей среде.

## Эквивалентность градиентов политики и мягкого Q-обучения

В завершение этой главы я хотел вернуться к сравнению методов оценки с методами политики. Проще говоря, методы Q-обучения и градиента политики пытаются укрепить поведение, которое приводит к лучшему вознаграждению. Q-обучение увеличивает ценность действия, тогда как градиенты политики увеличивают вероятность выбора действия.

Добавляя в смесь (или эквивалент) градиенты политики на основе энтропии, вы повышаете вероятность действия, пропорциональную некоторой функции ценности действия (см. уравнение 7.2). Или, более кратко, политика пропорциональна функции Q-значений. Следовательно, методы градиента политики решают ту же проблему, что и Q-обучение [21].

Сравнение подходов Q-обучения и градиента политики даже эмпирическим путем показывает, что производительность схожа (посмотрите на рис. 4 в работе ACER, где подходы градиента политики и Q-обучения почти одинаковы) [22]. Многие различия можно объяснить при рассмотрении внешних улучшений таких вещей, как эффективность выборки или параллелизм. Шульман и Аббель зашли так далеко, что нормализовали Q-обучение и алгоритмы, основанные на градиентах политики, до эквивалентности и доказали, что они дают одинаковые результаты [21].

## Что это означает для будущего?

Вывод, который вы можете сделать из всего сказанного, заключается в том, что со временем разница между методами на основе ценности действий и методами градиента политики будет небольшой. Эти два направления сольются (что они уже сделали в некоторой степени в парадигме "актор — критик"), поскольку и улучшение политики, и оценка ценности неразрывно связаны.

## Что это значит сейчас?

Однако пока ясно, что ни одна реализация не подходит для всех задач — классическая интерпретация теоремы *об отсутствии бесплатного обеда*. Вы должны рассмотреть несколько реализаций, дабы удостовериться, что вы оптимизировали производительность. Но убедитесь, что вы выбрали самое простое решение вашей проблемы, особенно если вы используете его в производственной среде. Совокупные промышленные знания и опыт работы с этими алгоритмами также помогают. Например, вы бы не выбрали какой-то эзотерический алгоритм вместо SAC по умолчанию, потому что люди, платформы и компании имеют больший опыт работы с SAC.

# Использованные источники

[1] Haarnoja T., Zhou A., Abbeel P., Levine S. Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor // ArXiv:1801.01290. — 2018. — August. — URL: **https://oreil.ly/7sUno**.

[2] Haarnoja T., Tang H., Abbeel P., Levine S. Reinforcement learning with deep energy-based policies // ArXiv:1702.08165. — 2017. — July. — URL: **https://oreil.ly/zBwzW**.

[3] Shi W., Song S., Wu C. Soft Policy gradient method for maximum entropy deep reinforcement learning // ArXiv:1909.03198. — 2019. — September. — URL: **https://oreil.ly/j-5Pg**.

[4] Liu Q., Wang D. Stein variational gradient descent: a general purpose Bayesian inference algorithm // ArXiv:1608.04471. — 2019. — September. — URL: **https://oreil.ly/_PHaF**.

[5] Haarnoja T., Zhou A., Abbeel P., Levine S. Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor // ArXiv:1801.01290. — 2018. — August. — URL: **https://oreil.ly/PE_dM**.

[6] Christodoulou P. Soft Actor-critic for discrete action settings // ArXiv:1910.07207. — 2019. — October. — URL: **https://oreil.ly/6sVHP**.

[7] Haarnoja T., Zhou A., Hartikainen K. et al. Soft Actor-critic algorithms and applications // ArXiv:1812.05905. — 2019. — January. — URL: **https://oreil.ly/8GIoC**.

[8] Ren T., Xie Y., Jiang L. Cooperative highway work zone merge control based on reinforcement learning in a connected and automated environment // ArXiv:2001.08581. — 2020. — January. — URL: **https://oreil.ly/p6kSZ**.

[9] Chen G., Peng Y. Off-policy actor-critic in an ensemble: achieving maximum general entropy and effective environment exploration in deep reinforcement learning // ArXiv:1902.05551. — 2019. — February. — URL: **https://oreil.ly/M7mW-**.

[10] Ciosek K., Vuong Q., Loftin R., Hofmann K. Better exploration with optimistic actor-critic // ArXiv:1910.12807. — 2019. — October. — URL: **https://oreil.ly/2m-c-**.

[11] Wang C., Ross K. Boosting Soft actor-critic: emphasizing recent experience without forgetting the past // ArXiv:1906.04009. — 2019. — June. — URL: **https://oreil.ly/1f1sl**.

[12] Shi W., Song S., Wu C. Soft Policy gradient method for maximum entropy deep reinforcement learning // ArXiv:1909.03198. — 2019. — September. — URL: **https://oreil.ly/AcQXD**.

[13] Haarnoja T., Tang H., Abbeel P., Levine S. Reinforcement learning with deep energy-based policies // ArXiv:1702.08165. — 2017. — July. — URL: **https://oreil.ly/8Kn1_**.

[14] Gao Y., Darrell T. End to end learning in autonomous driving systems // EECS at UC Berkeley. — 2020. — URL: **https://oreil.ly/gi7ZM**.

[15] Nachum O., Norouzi M., Xu K., Schuurmans D. Bridging the gap between value and policy based reinforcement learning // ArXiv: 1702.08892. — 2017. — November. — URL: **https://oreil.ly/2lhFk**.

[16] Nachum O., Norouzi M., Xu K., Schuurmans D. Trust-PCL: an off-policy trust region method for continuous control // ArXiv: 1707.01891. — 2018. — February. — URL: **https://oreil.ly/dplSx**.

[17] Engstrom L., Ilyas A., Santurkar S. et al. Implementation matters in deep policy gradients: a case study on PPO and TRPO // ArXiv:2005.12729 Cs, Stat. — 2020. — May. — URL: **https://oreil.ly/mznN4**.

[18] Kramer T. Tawnkramer/Gym-Donkeycar. Python. — 2018. — URL: **https://oreil.ly/64YPf**.

[19] Raffin A., Sokolkov R. Learning to drive smoothly in minutes. GitHub. — 2019. — URL: **https://oreil.ly/XvPVj**.

[20] Li X., Yang W., Zhang Z. A Regularized approach to sparse optimal policy in reinforcement learning // ArXiv:1903.00725. — 2019. — October. — URL: **https://oreil.ly/iXHx4**.

[21] Schulman J., Chen X., Abbeel P. Equivalence between policy gradients and soft Q-learning // ArXiv:1704.06440. — 2018. — October. — URL: **https://oreil.ly/ZjQ67**.

[22] Mnih V., Badia A. P., Mirza M. et al. Asynchronous Methods for Deep Reinforcement Learning // ArXiv:1602.01783. — 2016. — June. — URL: **https://oreil.ly/QeJ1M**.

# ГЛАВА 8

# Улучшение процесса обучения агента

Сложные производственные задачи часто можно разложить на *ориентированные ациклические графы* (directed acyclic graphs, DAG). Так можно повысить производительность разработки за счет разделения одного длинного проекта на множество более мелких, которые легче реализовать. Это помогает обучению с подкреплением (reinforcement learning, RL), потому что более мелкие компоненты часто легче и быстрее обучать, и они могут оказаться более надежными. В *разд. "Иерархическое обучение с подкреплением" далее в этой главе* демонстрируется один общий формализм, который заключается в получении иерархии политик, где политики низкого уровня отвечают за детальные "навыки", а политики высокого уровня — за долгосрочное планирование.

До сих пор в этой книге я рассматривал только проблемы с одним агентом. Для некоторых задач требуются группы агентов, или, по крайней мере, чтобы в одной среде могли работать несколько агентов. В *разд. "Мультиагентное обучение с подкреплением" далее в этой главе* показано, как агенты могут сотрудничать или конкурировать для решения задач с участием нескольких агентов с помощью глобальных или локальных вознаграждений.

Еще одна быстро развивающаяся область RL — это пересмотр того, как вы должны думать о наградах. Традиционно агент несет исключительную ответственность за использование сигнала вознаграждения для изучения политики. Но вы можете дополнить этот процесс, предоставив дополнительную, потенциально внешнюю информацию в виде экспертного руководства. В *разд. "Экспертное руководство" далее в этой главе* обсуждается, как в политику включить экспертные знания, которые в итоге помогают ускорить обучение за счет улучшения исследования. Можно даже узнать оптимальное вознаграждение, которое устраняет необходимость в инжиниринге наград.

Но, во-первых, RL неразрывно связано с марковскими процессами принятия решений (Markov decision processes, MDP) и уравнением Беллмана, и все алгоритмы RL полагаются на теоретические гарантии оптимальности при принятии последовательных решений (возможно, вы захотите перечитать *разд. "Марковские процессы принятия решений" главы 2*). Однако различные реальные сценарии нарушают основополагающие предположения MDP, а это часто приводит к провалу предположений, сделанных уравнением Беллмана, и проявляется в виде политик, которые не будут сходиться независимо от того, сколько вы провозились с настройкой гиперпараметров. В следующем разделе рассматриваются варианты базовых MDP для учета этих сценариев.

В этой главе кратко излагаются последние достижения. Думаю, что четыре наиболее важные темы — это то, как именно вы определяете MDP, иерархическое RL, мультиагентное RL и предоставление экспертных руководств Все это можно считать "продвинутым", и для глубокого обсуждения перечисленных тем потребовалась бы еще одна книга Однако я думаю, что полезно сделать обзор, чтобы вы знали о существовании новых направлений и могли копнуть глубже, если захотите. Как обычно, ссылки и рекомендации в *разд. "Дополнительные материалы для чтения" в конце данной главы* могут предоставить большое количество вспомогательной информации.

# Переосмысление марковских процессов принятия решений

Алгоритмы RL, использующие фреймворк MDP и вытекающие из этого гарантии уравнения Беллмана, предполагают, что базовая среда является марковской. Хотя может наблюдаться большая неопределенность в отношении последствий действий агента, восприятие состояния окружающей среды агентом должно быть идеальным. Агент использует это всемогущее знание, чтобы выбрать следующее действие в соответствии с текущей политикой $\pi(a|s)$.

Но что, если представление о состоянии не идеально? Что если, когда вы прикажете сервоприводу переместиться в определенное положение, а он переместится в положение, распределенное случайным образом? Или что, если существуют скрытые функции, показывающие, насколько вероятно, что пользователь нажмет на рекомендацию? В этих ситуациях состояние не полностью наблюдаемо и марковские предположения нарушаются.

## Частично наблюдаемый марковский процесс принятия решений

Одним из решений *частичной наблюдаемости* является разделение представления состояния. Частично наблюдаемый марковский процесс принятия решений (partially observable Markov decision process, POMDP) содержит дополнительные параметры в сравнении с MDP, чтобы включить конечный набор наблюдений, с которыми агент может столкнуться, — $\Omega$ и *функцию наблюдения* $O$, которая отображает состояние и действие в распределении вероятностей по действиям. В дальнейшем вы можете преобразовать задачу обратно в MDP, аппроксимируя обратную функцию наблюдения и используя ее для преобразования наблюдений в обратное состояние.

На рис. 8.1 показано, что POMDP является расширением MDP, в котором агент не может наблюдать текущее состояние среды непосредственно. Вместо этого он использует наблюдение и информацию о предыдущем действии и состоянии для создания прогноза текущего состояния $\hat{s}$. Прогнозирование состояния часто называют *доверительным состоянием* $b$, которое можно описать как "состояние, в котором, по моему мнению, находится окружающая среда".

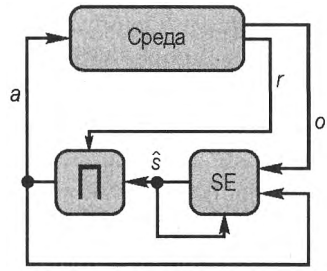



**Рис. 8.1.** Вы можете разложить POMDP на MDP (с состояниями, действиями, вознаграждениями и политикой) и оценщиком состояния (state estimator, SE), который предсказывает состояния по наблюдениям ($o$)

## Предсказание доверительного состояния

Итак, что такое доверительное состояние и как его предсказать? Рассмотрим пример, в котором вы пытались провести робота через простой лабиринт. Агент не знает схему лабиринта, но может быть достаточно умным, чтобы обнаружить существующий впереди Т-образный перекресток, и ему нужно сделать выбор, в каком направлении двигаться. Плохо натренированная политика будет соответствовать условиям обучения и приведет к тому, что каждый раз станет предлагать одно и то же решение.

Простое решение этой проблемы, как показано на рис. 8.1, заключается в том, что наилучшее действие может основываться на прошлом опыте. Я имею в виду не прошлый опыт в смысле обучения, не буфер воспроизведения, а прошлый опыт текущего эпизода и окружающей среды. Например, если вы знаете, что робот уже пробовал повернуть налево и оказался в тупике, то скрытое состояние лабиринта (лучший маршрут), вероятно, будет справа.

Многие сложные примеры используют эту простую идею, включая прошлые действия и наблюдения в представлении состояния. Например, тесты Atari всегда складывают четыре кадра наблюдений за пикселами, чтобы сохранить некоторую "историю" окружающей среды. Это позволяет агенту предсказывать траектории шариков для пинг-понга или движение врагов. И многие примеры робототехники включают предыдущие действия и состояния, чтобы постараться учесть неточности в позиционировании и сгладить движение.

Хотя эта идея помогает во многих случаях, в целом ее недостаточно. Для того чтобы действовать оптимально, агент должен учитывать, насколько он не уверен в текущей ситуации. Более сложным решением является моделирование оценки состояния с использованием аппроксиматора, способного изучать зависящие от времени функции.

Существует несколько вариантов решения этой проблемы. Вы можете попытаться отследить доверительное состояние; в простой версии может использоваться фильтр Калмана, хотя более поздние подходы пытаются построить нейронные сети для получения оптимального фильтра Байеса [1]. Вы можете включить аппроксимацию доверительной модели непосредственно в модель политики, используя предикторы с поддержкой памяти или доверительные трекеры [2, 3].

Но самый простой (в предыдущей работе было 24 автора!) и наиболее распространенный подход — использование рекуррентной нейронной сети (recurrent neural network, RNN) в конце функции представления состояния. В зависимости от задачи RNN не обязательно должны быть сложными, и они позволяют агенту принимать обоснованные решения на базе прошлых наблюдениях. Это позволяет вам обучать RNN, используя стандартные методы и инструменты.

В качестве концептуального примера рассмотрим эксперимент из *разд. "Отраслевой пример: обучение вождению автомобиля с дистанционным управлением" главы 7*, где я использовал вариационный автоэнкодер для представления состояния. Я мог бы добавить RNN к выходным данным представления состояния и обучить ее, предсказывая будущие наблюдения. В конце концов, автокодировщик сможет давать прогнозы того, что произойдет при определенных действиях, и, следовательно, решить проблему частичной наблюдаемости. Политика будет иметь полную наблюдаемость всех возможных будущих наблюдений, поэтому вы можете оптимизировать MDP, используя стандартные методы RL.

## Практический пример: POMDP в автономных транспортных средствах

Вы можете предположить, что предыдущий пример слишком надуман, чтобы его можно было применить к реальной жизни. Но удивительное количество реальных сценариев идеально вписывается в предыдущую модель. Одним из таких примеров являются автономные транспортные средства.

Распространенная автономная задача — решить, когда транспортное средство может совершать безопасное движение; например, пересечение перекрестка со знаками остановки или перекрестка с круговым движением — обычная дорожная ситуация. По прибытии агент должен довести свои действия до совершенства, чтобы избежать столкновения с другими транспортными средствами. Тем не менее, если транспортное средство ждет слишком долго, драгоценное время теряется, а эффективность перекрестка снижается.

Автомобиль имеет ограниченный обзор дороги. Встроенные датчики, такие как камера или модуль LIDAR, находятся на одной линии и не могут видеть препятствия. Это делает проблему лишь частично наблюдаемой, потому что агент не может узнать о важных объектах за пределами этого пространства, например, об ускоряющемся автомобиле, мчащемся к перекрестку.

Традиционные алгоритмы пытаются количественно оценить время до столкновения, но точная оценка времени и решение неожиданных ситуаций сложны и недостаточно надежны для автономной работы.

Моделируя такую задачу, как POMDP, вы можете выбрать оптимальную политику с помощью RL. Цяо и соавт. продемонстрировали (в имитационном моделировании), что агент с политикой, поддерживающей LSTM, способен узнать, когда следует выехать с четырехстороннего перекрестка, чтобы повернуть налево, направо или двигаться прямо [4]. Политика LSTM позволяет узнать, какие частично наблю-

даемые ситуации приводят к столкновению с помощью надлежащим образом разработанной награды, и избежать их. Вы можете самостоятельно найти результаты этого эксперимента на YouTube[1].

Цяо и соавт. добавили иерархическое RL к предыдущей модели, чтобы лучше моделировать высокоуровневые действия (или низкоуровневые подзадачи, в зависимости от того, как вы об этом думаете): начать движение, остановиться, следовать за впереди идущим автомобилем и т. д. [5]. Вы сможете прочитать больше об иерархическом RL в *разд. "Иерархическое обучение с подкреплением" далее в этой главе*.

## Контекстные MDP

Существует много проблем, в основе решения которых лежит MDP, но контекстная ситуация может быть разной. Например, представьте, что вы обучаете агента вводить лекарство пациенту с диабетом в зависимости от уровня сахара в его крови. Основа MDP одинакова для каждого пациента, уровень сахара — это наблюдение, а введение лекарственного препарата — это действие. Но на наблюдение влияют внешние переменные, такие как пол или возраст пациента.

Это похоже на ситуацию с POMDP, но есть существенная разница. Эти контекстные переменные не меняются при каждом измерении. Моделируя контекст как набор фиксированных параметров, вы ограничиваете агенту пространство поиска, что может привести к более надежным моделям и более быстрому обучению [7].

Это расширение имеет решающее значение, когда возникает проблема с небольшим количеством шагов в эпизоде (короткие цепи Маркова), но с огромным количеством независимых эпизодов. Именно в такой ситуации и оказываются рекомендательные алгоритмы, поэтому большая часть исследований контекстных MDP (contextual MDP, CMDP) основывается на рекомендациях или ранжировании приложений [8].

## MDP с изменяющимися действиями

Некоторые приложения имеют изменяющиеся наборы действий, и исследования идут по двум направлениям, приводя к аналогичным результатам. Первый набор исследований сосредоточен на ситуациях, когда действия не всегда доступны. Например, в приложениях для планирования пути, таких как маршрутизация транспортных средств или пакетов, пути могут быть закрыты из-за повреждения или проведения технического обслуживания, или в рекламе — ее выбор может меняться в зависимости от времени суток [9].

Предпосылкой является набор стохастических действий (stochastic action set, SAS), который содержит набор действий, доступных в определенное время и смоделированных как случайная величина. Общее решение этой проблемы — изучение политики, основанной только на действиях, наблюдаемых в этом конкретном состоянии.

---

[1] См **https://oreil.ly/ivbfj**.

Бутилье и соавт. показали, что в дальнейшем можно использовать стандартные варианты Q-обучения [10]. Чандак и соавт. расширили идею, включив методы градиента политики [11].

Второе, но более общее направление исследований сосредоточено на обучении на протяжении всей жизни, или непрерывном обучении. Здесь наборы действий меняются, потому что меняются проблемы и приоритеты. Например, вы можете захотеть добавить новую функцию в свою роботизированную руку или ввести новый набор объявлений, не переобучая свою модель заново и не теряя полезный опыт. Основная проблема здесь — проклятие размерности с постоянно растущими наборами потенциальных действий и нехваткой данных для выборки этих действий. Теоретически добавить дополнительные действия так же просто, как добавить их в MDP. Но создание архитектуры, способной справиться с такими изменениями, является сложной задачей.

Чандак и соавт. продемонстрировали, что это возможно, разделив политику на две части. Первая отвечает за принятие решений на основе состояния, но ее результат находится в метапространстве, а не в пространстве действий. Затем вторая часть сопоставляет это метапространство с пространством действий. Вместе они образуют целостную политику [12].

Преимущество заключается в том, что вы можете изменять параметры модели метадействий для учета новых действий, не касаясь части представления состояния. Затем вы можете использовать стандартное контролируемое обучение в буфере воспроизведения для обучения новой модели метадействий. Это ограничивает объем переобучения, требуемого в реальной среде, только теми частями модели, которые влияют на новое действие.

### Регуляризованные MDP

Стоит еще раз упомянуть, что класс алгоритмов, включающий регуляризаторы, также является переформулировкой MDP. Если вы помните (*см. разд. "Резюме" главы 7*), SAC и другие подобные алгоритмы пытаются найти оптимальную политику, которая максимизирует дисконтированную доходность, как и любой алгоритм, основанный на MDP, но они также включают терм регуляризации. Идея здесь в том, что вы можете использовать этот терм для обеспечения устойчивости подобно тому, как регуляризаторы влияют на линейную регрессию. Но добавление фундаментально изменяет равенство Беллмана, что означает влияние на лежащие в основе MDP. К счастью, Ли и соавт. доказали, что не имеет значения, какую регуляризацию вы применяете, алгоритм все равно будет сходиться, пока вы будете следовать некоторым правилам, определяющим регуляризующую функцию [13].

## Иерархическое обучение с подкреплением

Ряд задач используют низкоуровневое поведение, которое является атомарным и повторяющимся; они включают овладение "простым" навыком, таким как выбор объекта или рекомендация одного продукта. Задачи, предполагающие применение

многоуровневых рассуждений или сочетающие в себе несколько навыков, гораздо сложнее.

В инженерном деле разделение задач на более мелкие компоненты может упростить сложную проблему. *Иерархическое обучение с подкреплением* (hierarchical reinforcement learning, HRL) использует эту идею, разбивая задачу на подзадачи, где решение каждой подзадачи проще или лучше, чем решение задачи целиком. Это может привести к улучшению вычислений, повышению надежности, а иногда и к повышению производительности. Это также один из способов смягчить "проклятие размерности", сократив пространства поиска. Например, когда вы учитесь ходить, нет смысла искать пространство для такого действия, как ходьба на руках, если вы не гимнаст.

С помощью иерархии политик, в которой только низкоуровневые политики взаимодействуют с окружающей средой, вы можете обучить политики высокого уровня планированию или принятию стратегических решений в течение более длительного периода времени. Еще одна привлекательная идея заключается в инкапсуляции политик для эффективного выполнения одной задачи, которые вы можете повторно использовать в других задачах; библиотеках политик.

Однако существует целый ряд проблем, которые вы должны учитывать. Как определить и разработать хорошо инкапсулированные низкоуровневые политики? Как обучать политики более высокого уровня и как выстроить иерархию? И как обучать все это за разумное количество времени?

## Наивный HRL

Вам следует подумать о разработке иерархий вручную, потому что во многих задачах существует четкое разделение проблем. Например, обучение сложного робота движению явно отличается от обучения его приготовлению кофе, полезный разговор зависит от слов и высказываний, используемых в разговоре, а успешное завершение учебного курса — от его содержания [14, 15]. Оптимизация этих высокоуровневых целей с помощью низкоуровневых индивидуальных навыков приводит к гибкой, надежной и эффективной политике.

В более простых ситуациях может быть предпочтительнее проектировать абстракции и, следовательно, иерархию вручную. Затем вы можете разработать индивидуальную реализацию RL, которая использует преимущества этой разработанной вручную модели, ориентированной на конкретные проблемы. На рис. 8.2 показан пример разложения условной задачи приготовления кофе в иерархию моделей поведения.

Основное преимущество этого подхода состоит в том, что он, вероятно, будет надежнее и проще, чем более сложный общий подход. Это немного похоже на то, как попросить нейронную сеть предсказать цену автомобиля по фотографии; вы будете гораздо успешнее создавать подсети, способные классифицировать марку и модель, определять местонахождение номерных знаков для их автоматического распознавания и т. д. и вводить их в высокоуровневую модель, которая предсказывает цену.

С практической точки зрения это также поможет вам сосредоточиться на мелких подзадачах, а не на одной большой задаче, которая решает все проблемы.

Правда, такой подход может только запутать вас. Во многих задачах определение границ в иерархии само по себе является сложной задачей, и разграничительная линия, вероятно, будет неоптимальной. Например, при езде на велосипеде "педаль" — базовый навык или же это "толкать правой ногой"? Вам нужен способ автоматического поиска этих навыков.

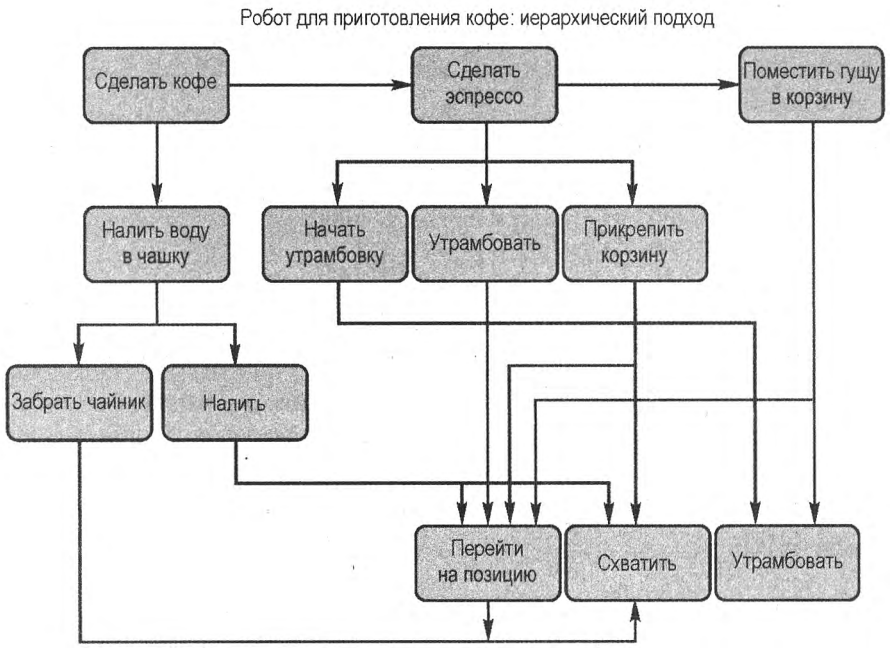



**Рис. 8.2.** Вы можете вручную разбить сложные задачи на иерархию атомарных моделей поведения и рассматривать каждую задачу отдельно. Это пример разложения условной задачи по приготовлению кофе на зависимые вспомогательные навыки. Следуйте стрелкам, чтобы найти "базовый" навык

## Высокоуровневые и низкоуровневые иерархии с внутренними наградами

Первым шагом на пути к автоматизированному решению HRL является разделение политики. Политика низкого уровня отвечает за взаимодействие с окружающей средой посредством состояний и действий, но не получает вознаграждения. Политика высокого уровня предполагает действие высокого уровня и поощряет политику низкого уровня наградой, присущей запрошенной цели. Когда среда приносит вознаграждение, ей дается политика высокого уровня. Эту идею, показанную на рис. 8.3, полностью объяснили Нахум и соавт. [16].

Например, политика высокого уровня сообщит политике низкого уровня, что нужно попытаться перейти в определенное состояние, и вознаградит политику низкого

уровня в зависимости от того, насколько близко она к этому состоянию приблизится. Политика высокого уровня выполняет выборку с более медленной скоростью, чем политика низкого уровня, чтобы обеспечить временну́ю абстракцию.

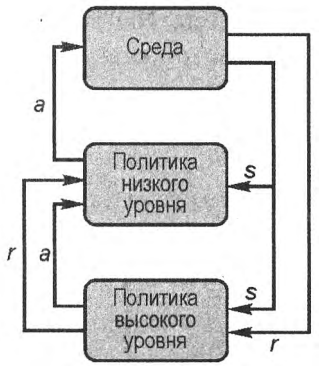



**Рис. 8.3.** Иерархическое RL с внутренними вознаграждениями

Вы можете параметризовать функцию низкоуровневого вознаграждения любым удобным для вас способом. Но исследователи предполагают, что вознаграждение за "расстояние — состояние" помогает ускорить обучение.

Следующая проблема заключается в том, что, хотя и можно применять алгоритмы на основе политики, такие как PPO, без дополнительной настройки, вы, вероятно, захотите использовать буфер воспроизведения для повышения эффективности выборки. Сложность в том, что буфер воспроизведения высокого уровня будет содержать наблюдения из предыдущих политик низкого уровня, как в самом начале, когда политика низкого уровня является случайной. Это означает, что буфер воспроизведения не является репрезентативным и приведет к проблемам конвергенции. Простым решением с предпочтением более свежих наблюдений может быть оценивание случайной выборки, что является формой воспроизведения опыта с определенными приоритетами.

Но Нахум и соавт. решили динамически *переназначить* изменение метки высокоуровневых действий/целей в соответствии с произошедшими действиями. Вспомните пример с кофе: у агента есть буфер воспроизведения с навыком "сварить кофе", но текущая обучаемая цель — "нажать кнопку". При их реализации выборка агента высокого уровня из этого буфера воспроизведения будет динамически изменять цель в соответствии с запрошенным навыком. В этом примере цель — "сварить кофе", которая должна соответствовать тому, что находится в буфере.

Может показаться, что вы фальсифицируете результаты постфактум, чтобы текущая политика высокого уровня выглядела лучше. Но вознаграждение от окружающей среды, как правило, нейтрализует некоторую предвзятость, добавленную переназначением. В худшем случае это похоже на дублирование наблюдений, которые уже находятся в буфере воспроизведения. В идеале вам нужен большой выбор некоррелированных, случайно распределенных переходов. Но если вы этого не сделаете, случайная выборка поможет сделать этот выбор более случайным.

Результаты показывают, что смоделированный робот со сложными ногами способен и учиться ходить, и изучать высокоуровневые задачи, такие как толкание блоков, поиск скрытых целей и сбор ресурсов.

## Навыки обучения и неконтролируемое RL

Основная проблема предыдущей интерпретации HRL заключается в том, что это все еще замкнутая система, которая оптимизируется для решения конкретной задачи, здесь нет никакого обобщения. Вы не смогли бы использовать обученную модель ни в какой другой ситуации. Она слишком хорошо подходит для текущей среды.

Но вы знаете, что, например, рука робота движется одинаково для каждой задачи. Команда "переместиться в положение $x$" зависит от робота и его конструкции, а не от окружающей среды. Все это время я описывал агента так, как будто он неявно связан с задачей, но на самом деле это независимая сущность. Итак, как вы можете обучить политике или *навыкам*, которые относятся к агенту, а не к среде?

Исследователи предлагают обучать агентов широкому спектру навыков, даже тех, которые не приносят положительных результатов. Политика более высокого уровня может затем выбрать, какие именно навыки использовать для решения проблемы. Обучение беспилотного летательного аппарата (БЛА) выполнению переворота может быть не лучшим решением для стабильной сети связи, но может оказаться полезным на концерте группы Metallica[2].

*"Разнообразие — это все, что вам нужно"* (diversity is all you need, DIAYN) — вот новая идея, которая способствует неконтролируемому обнаружению низкоуровневых политик. Айзенбах и соавт. предлагают, чтобы навыки были различимы, нацелены на разработку политики, ориентированной на разные состояния, а полученные навыки должны быть как можно более разнообразными. Для того чтобы поощрять случайность, вы можете максимизировать показатель энтропии, зависящий от со-

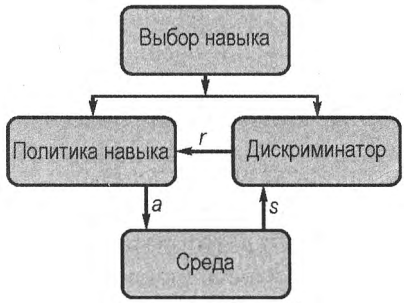



**Рис. 8.4.** Принципиальная схема системы обучения навыкам без учителя (DIAYN). Навык выбирается случайным образом, а политика воздействует на среду. Дискриминатор вычисляет энтропию политики, состояний и действий, которая используется в качестве вознаграждения. Со временем политика приобретает определенные навыки

[2] См. **https://oreil.ly/HZyeo**

стояний, действий или от того и другого. На рис. 8.4 показано упрощенное изображение этого процесса [17].

Результатом является набор разнообразных политик[3], которые могут быть полезны ли бесполезны для решения задачи. Но разнообразие действует как дополнительная ступенька, чтобы помочь в решении более сложных проблем. Мне особенно нравятся результаты в среде HalfCheetah, которые показывают бег, ползание и сидение[4].

## Использование навыков в HRL

Имея библиотеку навыков, как можно обучить алгоритм HRL для решения конкретной задачи? Я могу представить, что предварительно обученные навыки могут помочь почти всем алгоритмам RL, позволив им сосредоточиться на исследовании среды, а не на изучении того, как нужно исследовать. Это особенно верно в средах, где сам агент очень сложен.

*HRL с вознаграждением на основе преимуществ* (HRL with advantage-based rewards, HAAR) — один из алгоритмов, который использует предварительно обученные навыки для значительного улучшения начального разведывания и, следовательно, ускорения обучения. Ли и соавт. также представили идею использования преимуществ в качестве сигнала вознаграждения, а не евклидово расстояние для состояния, как в HIRO. Причина этого в том, что пространство состояний не обязательно отображается на реальное расстояние до цели. Например, если вы используете видеокамеру, расстояние в пикселах отличается от физического. Поэтому вместо этого они предлагают использовать оценку преимуществ из политики высокого уровня для переобучения и вознаграждения навыков низкого уровня. Таким образом, навыки, которые дают положительное преимущество, те, что хорошо справляются с задачей высокого уровня, будут продвигаться по сравнению с теми, которые этого не делают [18].

Еще одна интересная идея состоит в том, чтобы сократить время, в течение которого повторяется каждый навык, с целью еще больше стимулировать обширные исследования.

## Выводы HRL

Построение иерархии политик полезно, когда ваша задача достаточно сложна; лишь в этом случае можно извлечь выгоду из разделения. С инженерной точки зрения, если вы способны представить, что разделение вашей политики поможет упростить задачу или построить более надежные абстракции, тогда HRL может оказаться хорошим выбором. Трудно определить потенциал иерархической политики. Ищите ситуации, когда агентам приходится манипулировать сложными доменами. Робототехника — очевидный пример, но другие области столь же сложны,

[3] См. **https://oreil.ly/m68lG**.

[4] См. **https://oreil.ly/wVVYT**.

как агенты, работающие с текстом [19]. И, наоборот, во многих реальных задачах сложной является сама среда, а не агент, поэтому HRL может оказаться бесполезным.

В крайнем случае HRL можно воспринимать в качестве графа, когда связи между мини-субполитиками образуют большой граф. В недавних исследованиях предпринята попытка рассматривать состояние как граф и использовать топологические особенности со сверточными нейронными сетями на основе графов для решения задач, связанных с сеткой [20].

Даже если вы не работаете со сложными агентами, я думаю, что HRL показывает, что обучение без учителя играет важную роль в RL. И тем не менее, исследователи только начинают привлекать к нему внимание [21] Выделите время, чтобы подумать, может ли ваша задача извлечь выгоду из стандартной библиотеки навыков или предварительного обучения; это может значительно повысить скорость обучения.

Еще один интересный вопрос: какое влияние вам нужно на вашу политику? Навыки, генерируемые HRL, не определяются экспертом, поэтому они могут создавать последовательность действий, которая незнакома, неожиданна или даже нежелательна. Конечно, они зависят от того, как определено вознаграждение без учителя и с учителем, поэтому вы можете таким образом повлиять на навык. Но открытый вопрос, который, вероятно, связан с конкретной проблемой, заключается в том, следует ли вам вмешиваться в создание "воображаемых" политик или позволять агенту открывать собственные? Подозреваю, что в большинстве реальных задач это комбинация того и другого. Вряд ли вам хочется чрезмерно погружаться в процесс программирования, потому что в этом весь смысл RL — вы не можете вручную запрограммировать оптимальную политику для сложных задач. Но вам также необходимо ограничить задачу и действие агента, чтобы он не переусердствовал и не стал слишком ненадежным.

## Мультиагентное обучение с подкреплением

Пока что я рассматривал агентов-одиночек. Но многие задачи связаны более чем с одним игроком/агентом, и вполне вероятно, что вы сможете улучшить производительность, если систематически будете моделировать задачу с учетом нескольких агентов. Мультиагентное обучение с подкреплением (multi-agent reinforcement learning, MARL) решает проблему последовательного принятия решений несколькими автономными агентами, которые работают в общей среде. Каждый агент пытается оптимизировать долгосрочное вознаграждение, взаимодействуя с окружающей средой и другими агентами.

Приложения MARL делятся на три основные группы: кооперативные, конкурентные и сочетающие в себе оба этих качества. В условиях сотрудничества агенты пытаются сотрудничать, чтобы оптимизировать общую долгосрочную цель. Например, совместный MARL может моделировать и оптимизировать налоговую политику [22]. В условиях конкуренции глобальное вознаграждение часто сводится к нулю, поскольку агенты могут улучшить свою политику только за счет другого

агента. Указанная ситуация характерна для областей, которые включают в себя ценовые предложения, таких как торговля и реклама. Наконец, среда может содержать обе составляющие. Например, рекламный рынок хочет максимизировать как глобальный доход, так и долгосрочное участие пользователей, а это означает, что модель сотрудничества будет оптимальной; но рекламодатели хотят платить наименьшую сумму денег за наибольшее влияние, что добавляет конкуренцию в среду.

В литературе MARL обычно обсуждается в контексте игр, часто традиционных, таких как шахматы или покер, и все чаще компьютерных игр. Как и в стандартном RL, причиной использования является то, что его легко моделировать и проверять. Реальные или промышленные примеры встречаются реже, но в последнее время появляются все чаще. Например, приложения включают использование MARL для формального определения правительственной политики, снижение перегрузки TCP-пакетов низкого уровня и совместную работу нескольких роботов с участием человека [23–25].

## Фреймворки MARL

В литературе существует тенденция к моделированию задач MARL одним из двух способов, представленных на рис. 8.5, *б, в*.

Взглянув на рис. 8.5, *а*, сначала вспомните, что MDP моделируется как агент, который действует в среде, в свою очередь создающей новое состояние и вознаграждение. Если среда носит эпизодический характер, она также производит сигнал завершения.

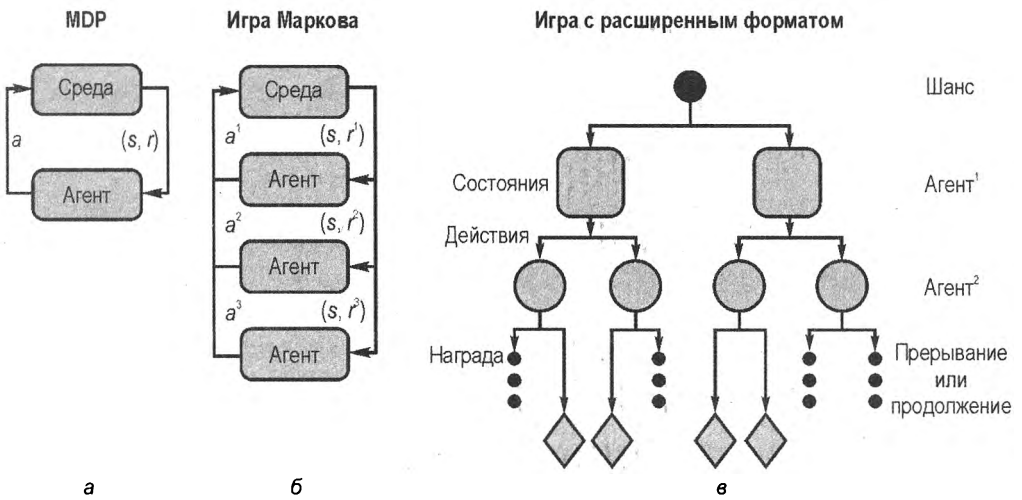



**Рис. 8.5.** Принципиальные схемы для различных фреймворков MARL: *а* — MDP, где агенты действуют в среде, которая производит состояния и вознаграждения; *б* — игра Маркова, в которой несколько агентов имеют независимые действия и вознаграждения; *в* — игра с расширенным форматом, в которой точки обозначают шанс, квадраты и круги обозначают двух агентов, а ромбы обозначают конечные состояния

На рис. 8.5, *б* показано, как первая структура MARL является расширением MDP, называемой игрой Маркова (Markov game, MG), в которой несколько агентов работают параллельно. Агенты совершают уникальные действия и получают уникальные награды, но состояние среды глобально. Это наиболее очевидное расширение MDP в контексте RL.

На рис. 8.5, *в* представлена вторая и более распространенная модель, игра в развернутой форме (extensive-form game, EFG), где шанс, состояние и действия представлены в виде дерева, что очень похоже на развертывание MDP. EFG по-прежнему изменяет состояние среды посредством действий, даже если это не показано явно. Например, если вы решите сделать ставку в игре в покер, ваш соперник будет вынужден сделать ставку или сбросить карты. Ваше действие влияет на состояние игры. Но вы можете видеть, что последовательные действия между агентами гораздо более явны. На рис. 8.5, *в* видно, что агент 2 не может сделать ход, пока агент 1 не завершит свой. Этот сценарий невозможен с MG. Обратите внимание, что EFG всегда рассматриваются с точки зрения одного из агентов.

Одновременные действия (например, MG) или несовершенная информация (например, POMDP) могут быть представлены *информационными наборами*, которые представляют собой группы узлов, принадлежащих одному агенту, который не знает, какой узел был достигнут. Обычно их обозначают пунктирной линией или большим кругом. Пример приведен на рис. 8.6.

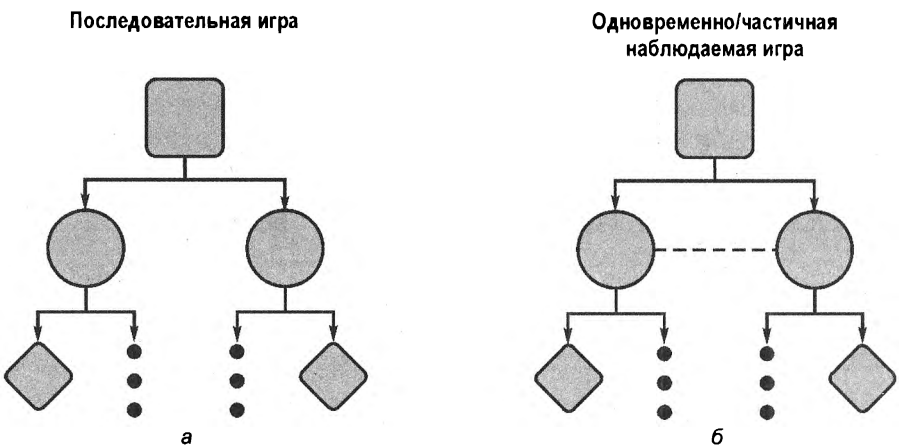



**Рис. 8.6.** Схемы игр в расширенной форме для последовательных (*а*) и одновременных (*б*) или несовершенных/частично наблюдаемых игр. Единственная разница в том, знают ли другие агенты, в каком состоянии они собираются остановиться

В качестве примера рассмотрим игру "Камень, ножницы, бумага". Это классическая игра, в которой два человека одновременно раскрывают одно из трех действий, чтобы получить награду. Для того чтобы отобразить это в EFG, вы можете попросить людей записать действия на листе бумаги, а затем одновременно раскрыть варианты. Агенты действовали в разное время, и второй агент не мог наблюдать за действиями первого агента. Это означает, что с точки зрения агента 2 игра находится в одном из трех возможных состояний, но он не знает, в каком именно.

EFG являются более общими и предпочитаются теоретиками. Но интуитивно мне по-прежнему нравится использовать MG в качестве ментальной модели, потому что подавляющее большинство промышленных ситуаций можно спроектировать для использования одновременных действий, и я считаю, что простота рекурсивной диаграммы менее обременительна, чем развернутое дерево. Многие исследователи соглашаются в этом и используют MG в своих приложениях.

## Централизованное или децентрализованное

Помимо различия между кооперацией и конкуренцией, наиболее важное решение, которое необходимо принять, заключается в том, выиграет ли ваша задача от централизованного или децентрализованного обучения. На рис. 8.7 показаны четыре сценария с разной степенью независимости агентов.

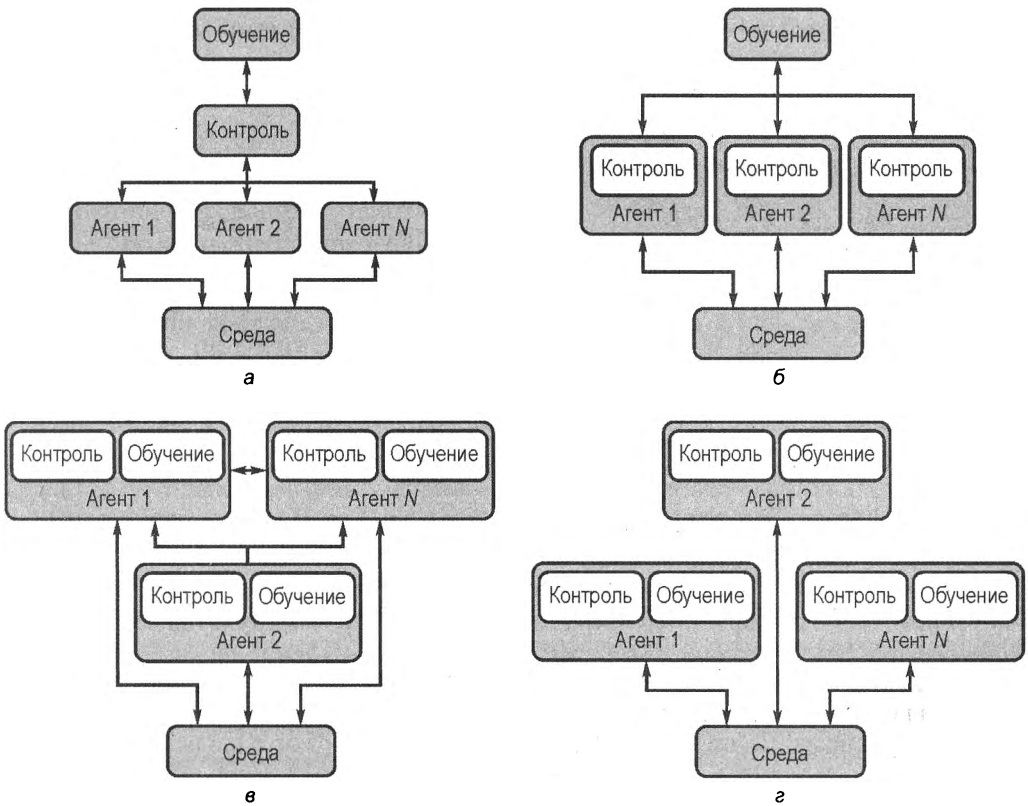



**Рис. 8.7.** Различные формы контроля и обучения в MARL: *а* — в полностью централизованных настройках агенты не могут контролировать выполнение или обучение; *б* — при централизованном обучении агенты имеют полный контроль над выполнением, но политики распределяются центральным контроллером; *в* — в децентрализованной среде, если возможно общение, агенты могут обмениваться опытом и политиками; *г* — в полностью децентрализованной модели агенты независимы

При централизованном обучении или, как правило, централизованном обучении и распределенном исполнении, опыт передается одному учащемуся, а политики возвращаются для локального исполнения. Основное преимущество этого метода заключается в том, что вы можете создавать политики на основе опыта нескольких участников. Например, если бы ребенок мог учиться централизованно, на опыте других, он выиграл бы от большого количества опытов и их разнообразия. Точно так же использование опыта разрозненного набора агентов помогает им гораздо быстрее искать пространство "состояний — действий". По сути, независимый опыт помогает исследовать, а централизованное обучение гарантирует, что у всех агентов будут одинаково приемлемые политики.

В децентрализованной модели (часто называемой DEC-MDP) и выполнение, и обучение происходят локально. Это более масштабируемый вариант, поскольку не имеет значения, сколько агентов вы вводите. Например, представьте, если бы все "управляли" автономным транспортным средством. Единый централизованный процесс обучения вряд ли будет оптимальным для всех условий вождения. Локальное обучение позволяет агентам адаптироваться к их местному восприятию окружающей среды. Например, правила движения по левой стороне дороги, вероятно, будут несколько отличаться от правил движения по правой стороне.

Другая проблема в том, что у агентов могут быть разные возможности. Например, опыт велосипедиста так же важен, как опыт автомобилиста, но объединить эти опыты централизованно сложно. У роботов могут быть разные конфигурации. Децентрализованное обучение позволяет агентам научиться использовать действия, имеющиеся в их распоряжении.

Еще одним преимуществом является то, что децентрализованная модель потенциально позволяет использовать стандартные алгоритмы RL для одного агента, поскольку вы можете награждать каждого агента независимо.

## Алгоритмы с одним агентом

Первый и наиболее очевидный путь к решению проблем MARL — предположить, что каждый агент независим, и применить любой из подходов RL с одним агентом, обсуждаемых в этой книге. К сожалению, традиционные алгоритмы RL, такие как Q-обучение или градиент политики, плохо подходят для мультиагентных сред. Основная проблема заключается в том, что во время обучения другие агенты взаимодействуют и изменяют состояние окружающей среды. Это делает среду нестационарной с точки зрения любого отдельного агента. Когда среда постоянно меняется, это мешает агенту создавать стабильную политику. Это также означает, что обычные надстройки RL, такие как воспроизведение базового опыта, не помогают. Однако это не означает, что они полностью перестают работать. Агент может изучить стабильные политики в простых средах, но они обычно неоптимальны и, как правило, не сходятся, когда среды усложняются.

Технически возможно использовать алгоритмы градиента политики на основе моделей, потому что модель дает вам идеальное описание динамики среды, но для многих проблем модель недоступна.

## Практический пример: использование децентрализованного обучения с одним агентом в беспилотном летательном аппарате

Использование беспилотных летательных аппаратов (БпЛА), часто называемых *дронами*, становится все более распространенным явлением в приложениях, где геопространственный охват важен, но труднодостижим. БпЛА могут летать в неизвестную среду для наблюдения за лесными пожарами, выполнять поисково-спасательные операции и создавать сети связи [26–28]. Их популярность обусловлена низкой стоимостью эксплуатации, набором датчиков и гибкостью их применения.

Нецелесообразно пытаться вручную кодировать контроллеры для выполнения задач из-за отсутствия точной модели. RL представляет собой естественное решение без моделей, способное осуществлять навигацию в сложных условиях. Существует множество примеров использования RL для изучения примитивных задач БпЛА низкого уровня, таких как полет, перемещение и уклонение от препятствий. Но беспилотные летательные аппараты великолепны, когда используются совместно для решения задачи.

Цуй, Лю и Налланатан используют смоделированный парк БпЛА для обеспечения беспроводной инфраструктуры в регионах с ограниченным покрытием. БпЛА особенно полезны в этой ситуации, потому что обычно легко установить прямую видимость. Исследователи используют MARL для решения сложной задачи оптимизации, которая максимизирует производительность сети.

Их работа предполагала независимость от агентов и использование простого решения с одним агентом. Они использовали Q-обучение, чтобы изучить оптимальный выбор пользователя для подключения, подканала и уровня мощности. Обратите внимание, что позиции в этой работе фиксированы. Они разрабатывают функцию вознаграждения, основанную на показателях качества, и решают в симуляции трех БпЛА. Обратите внимание, что нет поправки на нестационарность среды, и агенты должны соревноваться за вознаграждение.

Так что есть много возможностей для улучшения. Более эффективный децентрализованный метод мог бы объяснить нестабильность окружающей среды. Или, поскольку у них есть сеть, централизованный метод облегчил бы разработку. А более сложный агент мог бы обрабатывать гораздо более сложное пространство состояний, например пространственное позиционирование, что помогло бы масштабировать решение. С такими исправлениями это возможно запустить в реальной жизни. Проблема, как сегодня отмечают исследователи, заключается в том, что они способствуют только успеху, а не неудаче. Вполне вероятно, что Цуй, Лю и Налланатан пробовали некоторые из этих идей, но у них не хватило времени.

Несмотря на это, я настроен оптимистично и могу представить себе более стабильный набор алгоритмов, способных изучать более сложные модели поведения (например, оптимальное позиционирование нескольких агентов).

## Централизованное обучение, децентрализованное выполнение

Не так давно благодаря объединению лучших подходов в RL с выборочной эффективностью централизованного обучения были достигнуты определенные успехи. Один из таких алгоритмов, получивший известность, сочетает в себе DDPG (*разд. "Глубокие детерминированные градиенты политики" главы 6*) с централизованным обучением и подходом к выполнению, называемым *мультиагентным DDPG* (multi-agent DDPG, MADDPG) [29].

DDPG (или TD3/PPO/другие) выступают тем стержнем, который позволяет агентам изучать политики без модели среды. Алгоритмы градиента политики обеспечивают теоретическую основу для комбинирования градиентов нескольких агентов, в отличие от методов оценки, в которых глобальная функция ценности действия не является репрезентативной для локальных ситуаций. Использование централизованного подхода к обучению во время тренировки повышает эффективность выборки и ускоряет обучение. Во время оценивания агенты используют независимую политику и локальные наблюдения.

Основная мотивация рассматриваемого подхода заключается в том, что если вы знаете действия всех агентов, можно прогнозировать или отслеживать изменения в среде. Как и в квантовой механике, если вы знаете состояние и действия всех атомов во вселенной, можете считать, что жизнь неподвижна, и предсказывать будущее (или прошлое).

Для того чтобы рассчитать ценность политики для отдельного агента, вам необходимо знать результат политики агента и ценность, прогнозируемую в среде. Однако окружающая среда зависит от всех агентов, поэтому вам необходимо предоставить информацию о действиях всех остальных агентов. Оглядываясь назад на различные реализации градиента политики (см. уравнение 5.11), становится ясно, что парадигма "актор — критик" естественным образом учитывает действия агента, поэтому все, что вам нужно сделать, — это расширить его, чтобы учесть действия *всех* агентов, что и показано с точки зрения одного агента в уравнении 8.1, где $N$ — другие агенты в окружающей среде.

**Уравнение 8.1. Мультиагентный актор — критик**

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \ln \pi_\theta(a \mid s) Q_w(s, a)\right] \quad \text{актор — критик}$$

$$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \ln \pi_\theta(a \mid s) Q_w(s, a_1, a_2, ..., a_N)\right] \quad \text{мультиагентный актор — критик}$$

Таким образом, у нас может быть местный независимый актор, но централизованный критик. Интересно, что, поскольку для каждого агента есть актор и критик, агенты могут иметь разные политики и вознаграждения, а значит, это работает как в условиях сотрудничества, так и в конкурентной среде. В качестве домашнего задания попытайтесь применить различные улучшения к алгоритмам градиента по-

литики, чтобы в итоге получить мультиагентные версии стандартных алгоритмов RL "актора — критика".

Большинству алгоритмов "актор — критик" нужна политика для генерации следующего действия для обучения функции ценности действия. Рассмотрите в качестве примера любой алгоритм "актор — критик". Обратите внимание на одно простое решение — попросить каждого агента передать политику или все следующие действия (в зависимости от алгоритма) обратно централизованному критику. Конечно, это вызывает проблемы с синхронизацией и задержкой и поэтому снижает эффективность, но зато довольно просто.

Подход, использованный MADDPG, заключался в *изучении* политик других агентов, в то время как они сами учатся. Это очевидный источник нестационарности. Для того чтобы ограничить влияние нестационарности, MADDPG замедляет скорость обучения, обновляя ограниченное подмножество случайных политик. Не идеально, но работает. См. соответствующую статью для получения более подробной информации.

## Децентрализованное обучение

Если вы не можете использовать централизованную модель, вам нужны способы стабилизации децентрализованного обучения. Гистерезис с временно́й разницей — это метод, который использует две разные скорости обучения в зависимости от размера ошибки временного различия [30]. Если ошибка временного различия ниже порога, обычно установленного на ноль, это означает, что агент получил меньшее вознаграждение, чем ожидалось, применяется более медленная (меньшая) частота обновления. Предполагается, что агент получил меньшее вознаграждение, чем ожидалось, из-за того, что агент занимался исследованием, — вы не хотите, чтобы ваш ребенок копировал вас, если вы делаете что-то плохое. Если ошибка положительная, агент использует нормальную скорость обучения. По сути, это делает алгоритм более оптимистичным, что само по себе может привести к проблемам, но процесс часто можно контролировать с помощью гиперпараметров скорости обучения. Суть, однако, в том, что такой подход помогает бороться с нестационарностью среды; агент не отменяет потенциально хорошую политику только из-за нескольких неудачных обновлений, вызванных изменением среды.

Вторая проблема децентрализованного обучения — частичная наблюдаемость. Классическим и самым простым решением является включение рекуррентного слоя для записи истории, хотя существуют и другие подходы. Ожидаю, что не за горами более формальные подходы.

Последняя проблема — невозможность использовать стандартные буферы воспроизведения. Проблема в том, что опыт является локальным и нестационарным при независимой выборке, несмотря на то, что агенты обучаются одновременно. Маловероятно, что локальный вид в этот момент времени будет представлять среду в будущем. Связанная с этим проблема заключается в том, что случайная выборка из разных частей буфера воспроизведения может иметь совершенно различные со-

стояния среды, что в худшем случае может привести к тому, что у двух агентов будут абсолютно одинаковые локальные условия, а это что приведет к совершенно разным политикам. Есть много потенциальных способов улучшить ситуацию, но Омидшафии и соавт. реализовали буфер воспроизведения, синхронизированный по всем агентам. Одновременное воспроизведение опыта предрасполагает к локальным корреляциям, что обычно является вредным и разрешается случайной выборкой. Однако в этой ситуации такой подход помогает, т. к. заставляет всех агентов сосредоточиться на текущем воспроизведении, которое должно иметь одинаковые условия среды. Например, если у двух агентов были похожие локальные условия, то выборка из буфера в одной и той же точке должна дать аналогичный опыт и, следовательно, привести к аналогичным политикам. В данном конкретном случае опыт представляет собой траектории для обучения рекуррентных нейронных сетей. Эта комбинация DQN, гистерезиса и буфера синхронизированного воспроизведения называется Dec-HDRQN [31].

Обратите внимание, что это лишь один из многих потенциальных способов решения проблемы нестационарности во время обновлений различных локальных оптимумов, воспроизведения опыта и частичной наблюдаемости. Вы можете заменить любой из предыдущих компонентов другими реализациями, хотя у них будут свои особенности.

## Другие комбинации

Централизованную и децентрализованную парадигмы можно скомбинировать множеством различных способов, чтобы удовлетворить потребности приложения. Например, вы можете соединить централизованного критика с централизованной политикой, которая используется агентами для удаленного выполнения. Или вы можете применить агентов с децентрализованным критиком и политиками, которые используются их ближайшими соседями. Было исследовано множество перестановок, и, вероятно, найдется алгоритм, подходящий для вашего случая. Подробное описание данной области выходит за рамки этой книги, но в *разд. "Дополнительные материалы для чтения" в конце данной главы* есть несколько замечательных источников для изучения.

Для меня MARL подчеркивает важность разработки задачи. Огромное количество перестановок в сочетании с количеством алгоритмов RL и возможностей нейронной сети означает, что с вычислительной точки зрения сложно автоматизировать или выполнить крупномасштабное сканирование гиперпараметров. Вы должны выбрать небольшое подмножество алгоритмов и архитектур, чтобы сделать цикл обратной связи достаточно небольшим и произвести что-либо в разумные сроки. Это означает, что жизненно важно иметь правильное и надежное определение задачи; в противном случае, если вы внесете небольшое изменение в нее, это может иметь огромное влияние на реализацию. В худшем случае вам, возможно, придется бросить свою разработку и свои исследования и начать всё заново.

### Почему трудно заставить RL работать в реальной жизни

Представьте, что у вас возникла проблема, когда вы решили, что централизованное обучение возможно. Вы потратили много недель на исследования, создание симуляторов и тестирование множества алгоритмов. Ваши первоначальные тесты прошли хорошо, и вы сообщили своим заказчикам, что готовы перейти к следующему этапу. Итак, вы начинаете тестирование в реальной среде на гораздо большем количестве агентов. Вы, счастливые, уезжаете на день и хорошо спите.

Утром вы включаете свой ноутбук, чтобы увидеть, какие новые политики были обнаружены. Но вы понимаете, что, во-первых, все стратегии не лучше, чем случайные, и во-вторых, прошло всего четыре шага обучения. После отрицания, гнева, "быстрых решений" и депрессии вы понимаете, что бóльшую часть времени агенты не могли связаться с сервером, и ваш код ждет, пока каждый агент не проверит, прежде чем запускать этап обучения. Вы быстро понимаете, что в реальной жизни у агентов нет надежной связи, и даже если бы она была, у вас в конечном счете окажется слишком много агентов, что замедлит работу всей системы. Вам нужна децентрализованная модель.

Единственный вариант — сообщить о неудаче заинтересованным сторонам и вернуться к началу. Я бы не назвал это пустой тратой времени, потому что этот опыт позволил вам получить много передаваемых знаний и навыков, но, судя по моему опыту, заинтересованные стороны, как правило, не в восторге от ситуации, когда движение по дорожной карте продукта откатывается назад.

Этот образец сбоя, неправильное определение проблемы, также распространен в науке о данных. Но в RL, как правило, существует гораздо более высокий барьер для экспериментирования из-за взаимодействия агента и среды, поэтому количество потраченного впустую времени может быть намного больше.

Один из способов смягчения последствий, обсуждаемых в этой книге, заключается в том, чтобы не торопиться при определении проблемы и последующем моделировании. Ваша модель должна быть достаточно простой, чтобы ее можно было обдумать и найти для нее решение, но она также должна отражать реальную проблему. Это похоже на программное обеспечение. лучший код прост и очевиден, но, кажется, справляется с большинством ситуаций Однако не думайте, что простое означает легкое; простота возникает в результате усилий, проб и множества ошибок.

## Проблемы MARL

Несмотря на значительный прогресс и обескураживающее количество исследований, MARL сложен. Это не значит, что невозможно его воплощение, и я не хочу отговаривать вас идти по этому пути; исследования показывают, что решения вполне выполнимы. Однако я хочу, чтобы вы знали о проблемах, которых следует остерегаться.

- *Разнообразие проблем.*

  Вам необходимо точно смоделировать свою среду, принимая решения о кооперации или конкуренции, совместном использовании политик или отказе их

взаимодействия, последовательном или одновременном, и вам также необходимо рассмотреть все обычные вопросы марковских процессов принятия решений, такие как ценность или политика, полностью наблюдаемые или частичные процессы. Когда у вас будет надежная модель, вам нужно будет покопаться, чтобы найти подходящий алгоритм, который работает в рамках вашей модели. MARL — весьма сложное решение проблемы, существует гораздо больше потенциально реализуемых решений.

- *Многомерные и мультимодальные цели.*

  Цель одного агента может значительно отличаться от цели другого агента из-за локальных условий среды, что может затруднить совместное использование политик.

- *Масштабируемость.*

  Во многих задачах важно синхронизировать информацию, будь то внешняя, например текущее состояние агента, или внутренняя, например текущая политика, что приводит к проблемам с эффективностью связи в более крупных развертываниях.

- *Нестабильность.*

  Поскольку агенты активно изменяют среду, с точки зрения другого агента окружающая среда нестационарна. Это нарушает или опровергает некоторые из основных предположений, предоставляемых MDP, и приводит к нестабильности.

- *Оптимальность.*

  Трудно точно определить оптимальность для всех примеров, кроме простейших, что означает отсутствие гарантий сходимости для нетривиальных сред.

Эти проблемы уникальны для MARL, но они превосходят все стандартные проблемы в RL, такие как частичная наблюдаемость и эффективность выборки. Вполне вероятно, что со временем эти препоны станут менее серьезными, поскольку исследователи найдут различные способы решения проблем RL.

## Выводы о MARL

MARL — это захватывающее и быстро развивающееся расширение RL. Я ожидаю, что значительные улучшения не за горами. Например, в июне 2020 г. на arXiv появилось более 50 новых исследовательских работ MARL. Но даже сейчас можно использовать MARL для решения множества коммерчески жизнеспособных вариантов использования, и некоторые из них вы можете увидеть на веб-сайте к оригиналу этой книги[5].

В этом кратком обзоре MARL я проигнорировал бóльшую часть литературы, связанной с играми. Это все очень важно с исследовательской и теоретической точек зрения, но выходит за рамки данной книги.

---

[5] См. **https://rl-book.com/applications**.

# Экспертное руководство

Во многих примерах или ситуациях, представленных в этой книге, я говорил об обучении агента политике с нуля. Но люди обладают знаниями в предметной области, которые могут направлять, поощрять и улучшать обучение агентов. Например, возраст, в котором дети идут в школу, зависит от страны. В США и Великобритании дети начинают обучение в возрасте четырех или пяти лет, но во многих странах, таких как Сербия, Сингапур, Южная Африка и Швеция, дети не начинают занятия до семи лет. Аргумент в пользу отложенного начала обучения заключается в том, что детям дается больше времени для случайного изучения, для точной настройки эмоциональных и моторных навыков низкого уровня, прежде чем они будут вынуждены изучать произвольные абстракции высокого уровня. Контраргумент состоит в том, что дети тратят время на случайные исследования и лучше тонко направлять их, используя учебную программу. То же самое и в RL; экспертное руководство, вероятно, предоставит ценную информацию, но агент также должен проводить исследование самостоятельно. Здесь нет единого мнения.

Однако с уверенностью утверждаю, что задачи могут выиграть от некоторой экспертной помощи. В этом разделе рассказывается о том, как включить этот опыт в вашу политику.

## Клонирование поведения

Традиционный подход к внедрению экспертных политик заключается в использовании контролируемого обучения, чтобы максимизировать вероятность выбора агентом действий, которые совпадают с действиями эксперта для обнаруженных состояний. Это называется *клонированием поведения*, но не пользуется популярностью, потому что не соответствует демонстрациям экспертов. Если бы агент оказался в состоянии, которое не было продемонстрировано, он не знал бы, что делать. Такая ситуация из-за многомерности сложных проблем весьма вероятна, поэтому важно использовать алгоритмы, которые продолжают учиться на своих действиях, даже если они не демонстрируются [32].

## Имитационное RL

В 2017 г. Хестер и соавт. предположили, что во многих реальных жизненных ситуациях точное моделирование недоступно или невозможно; единственный вариант — обучить агента на месте. Но есть внешние источники, которые уже используют приемлемые политики, такие как рассмотренный ранее контроллер или руководство человека. Разработка решений для передачи знаний от этих экспертов агентам позволяет найти ряд решений задач, в которых демонстрации являются обычным явлением, а точные симуляторы — нет [33].

Во-первых, эксперт (человек или машина) создает траектории в окружающей среде. Затем контролируемая потеря включается в обновление Беллмана, которое измеряет разницу между действиями эксперта и агента. Со временем агент учится генерировать траектории, подобные эксперту.

Проблема с этим методом заключается в том, что не хочется, чтобы агент имитировал эксперта, поскольку политика эксперта, скорее всего, будет неоптимальной. *Глубокое Q-обучение на основе демонстраций* (deep Q-learning from demonstrations, DQfD) объясняет это попыткой предотвратить переоснащение траектории эксперта с помощью ряда регуляризаций. *Генеративное состязательное имитационное обучение* (generative adversarial imitation learning, GAIL) предлагает аналогичный подход, который противодействует распространению действий эксперта. Но он не дает ответа или не имеет интуитивно понятной настройки, которая определяет, *сколько* рекомендаций экспертных советов использовать [34].

*Имитационное мягкое Q-обучение* (soft Q imitation learning, SQIL) предоставляет одну идею и в то же время упрощает DQfD и GAIL, изменяя способ использования стандартных алгоритмов воспроизведения опыта. Редди и соавт. обнаружили, что, во-первых, тщательное управление траекториями, которые находятся в буфере воспроизведения, во-вторых, реконструкция сигнала вознаграждения для этих траекторий и, в-третьих, использование функции потерь, которая позволяет исследовать, но остается близкой к экспертным траекториям, позволяют агенту учиться на опыте, даже когда нет внешнего сигнала вознаграждения. Полученный алгоритм в целом похож на клонирование поведения, за исключением того, что он допускает отклонения в состояния, которые не были продемонстрированы. Он наказывает эти состояния посредством регуляризации и подталкивает агента обратно к экспертному поведению. Вы даже можете контролировать количество рекомендаций экспертов для применения, взвешивая регуляризацию. Так что я могу представить себе версию с отжигом, которая изначально использует демонстрации экспертов и со временем стремится к чистому RL. Но главное преимущество — простота, и ее можно адаптировать к любому алгоритму вне политики, дискретному или непрерывному.

## Обратное RL

Проблема с имитацией RL заключается в том, что нет гарантии, что эксперт демонстрирует оптимальную политику. Например, представьте, что вы тренируете свою машину для "хорошего вождения", и, чтобы помочь своему алгоритму, вы записали свое вождение и представили это в качестве демонстрации эксперта. Вы действительно опытный водитель? Даже если это так, должен ли агент учиться на вашем субъективном опыте вождения?

Другой подход — игнорировать траектории и концентрироваться на цели. Однако получение вознаграждения за "хорошее вождение" трудно выразить количественно. Вы можете предположить, что хотите добраться до места назначения в кратчайшие сроки и использовать общее затраченное время в качестве награды, но при этом игнорируется безопасная возможность добраться до места назначения. Вы можете интуитивно установить все основные проблемы, но как тогда их взвесить? В общем, сложно спроектировать вознаграждения для сложных доменов, поэтому почему бы не изучить вознаграждение?

*Обратное RL* (inverse RL, IRL) пытается предсказать, как будет выглядеть вознаграждение на основе демонстраций из окружающей среды [36]. Общая цель состо-

ит в том, чтобы найти произвольную функцию вознаграждения, основанную на продемонстрированных траекториях, которые объясняют поведение эксперта. Проектирование функций вознаграждения — серьезная проблема в RL, и IRL обещает заменить ручное определение демонстрациями, которые, возможно, более объективны, и их легче определять. Также утверждается, что обучение на демонстрациях более надежно, т. к. они не делают никаких предположений об окружающей среде. Большинство методов IRL соответствуют следующему шаблону.

Предположите, что демонстрации эксперта являются решением для MDP.

1. Используйте параметризованную модель в качестве функции вознаграждения.
2. Решите MDP с текущей функцией вознаграждения.
3. Обновите функцию вознаграждения, чтобы минимизировать разницу между траекторией эксперта и изученной траекторией (или политикой, если они доступны).
4. Повторяйте до тех пор, пока процесс не станет стабильным.

На рис. 8.8 показан этот процесс для простой дискретной сетки $3 \times 3$. Агент RL обучен решать MDP с параметризованной функцией вознаграждения. Каждый раз, когда MDP узнает оптимальную политику, она сравнивается с траекториями экспертов. Если изученная политика посещает состояние, которое также посещается экспертом, вознаграждение за это состояние увеличивается (или меняются эквивалентные параметры вознаграждения). Со временем функция вознаграждения стремится к траектории эксперта.

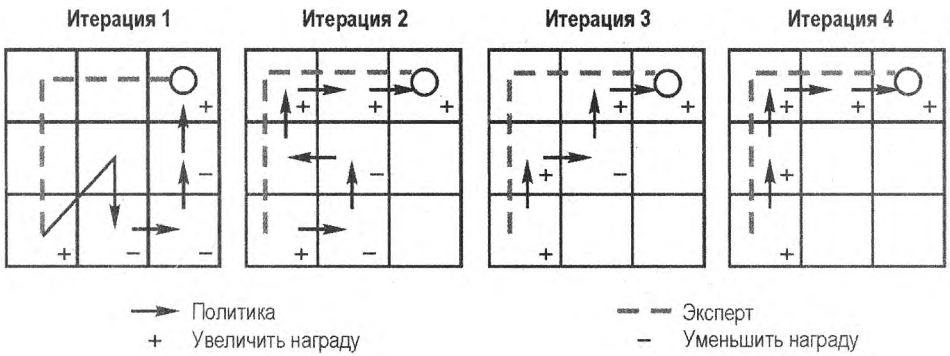



**Рис. 8.8.** Примерные итерации условной задачи IRL. Среда представляет собой сетку 3 × 3. Приведена единая экспертная траектория (пунктирные линии). Политика (сплошные линии) использует параметризованную функцию вознаграждения в MDP. Со временем функция вознаграждения изменяется, чтобы с большей вероятностью получить экспертные траектории. В конце концов, вы получаете функцию вознаграждения, имитирующую поведение эксперта

Большинство проблем с IRL связано с тем, что экспертные траектории могут быть шумными и неоптимальными. Исследователи разработали широкий спектр алгоритмов, которые адаптируют этот базовый процесс для повышения устойчивости к шуму (см. основанные на энтропии или вероятностные IRL-алгоритмы) или для

свободного обучения в сторону оптимальности, а не для сопоставления с экспертной траекторией, что, как правило, делает конкретную задачу более специфичной.

Другая серьезная проблема заключается в том, что эти методы требуют от вас многократного переобучения алгоритмов RL, а это, как вы знаете, может занять много времени. Это означает, что IRL не масштабируется ни к чему, кроме простейшего из пространств состояний. Однако еще не все потеряно; вы можете использовать метод уменьшения размерности, такой как автоматическое кодирование или хеширование, чтобы уменьшить многомерное состояние, с чем может справиться IRL [37, 38].

Ну, и последнее мое замечание заключается в том, что, говоря в своих статьях об экспертных рекомендациях, исследователи склонны предполагать, что эксперты преуспевают в своем деле. Неудачи встречаются гораздо чаще, чем успех, и в IRL могут попасть в том числе и катастрофические траектории; вам нужно просто уменьшить награды за эти траектории. Шиарлис и соавт. предполагают, что создание неудавшихся демонстраций даже проще, чем создание хороших [39].

## Обучение по учебной программе

Когда вы изучаете естественные науки, парусный спорт или учитесь кататься на сноуборде, вам необходимо пройти четко определенный ряд шагов или тестов, прежде чем вы сможете перейти к чему-то более сложному. В сноуборде вы сначала тренируете мышцы ног, чтобы контролировать края доски, затем тренируете ваш центр тяжести, чтобы правильно переносить вес для поворотов, и наконец, вы учитесь скручивать траекторию, чтобы выполнять элементы карвинга и вращать доской. В науке вы сначала узнаете об атомах до начала изучения термоядерного синтеза, о законах движения Ньютона — до гидродинамики и о клеточных структурах — до ДНК. Это учебные планы, высокоуровневые руководства по метаобучению, которые показывают вам общие стратегии изучения продвинутых политик.

Учебные программы можно использовать в RL, чтобы уменьшить объем исследований, которые необходимо выполнить агенту. Вы можете указать ряд более простых подцелей, которые приведут к изучению основной цели. Нечто подобное было описано в *разд. "Имитационное RL" ранее в данной главе*, и в некоторых методах в качестве учебной программы используются методы имитации для получения подзадач.

Игры Маркова (или другие структуры MARL) являются важными примерами для изучения учебных программ. Часто можно изменить формулировку задачи, чтобы настроить агентов друг против друга в самостоятельной игре. Наиболее очевидная форма самостоятельной игры — это широко разрекламированные примеры настольных игр, но исследователи применили ту же идею к немного более реалистичной среде, такой как лабиринты или движение роботов [40, 41]. Существует много вариантов того, как спроектировать взаимодействие в игре, одна распространенная форма — использовать один экземпляр агента в качестве учителя, а другой — в качестве ученика. Учитель предлагает "задачки", выполняя их в окружающей среде (и, следовательно, создавая траекторию), а ученик решает эту задачу,

пытаясь выполнить ту же "задачку". Я говорю "задачка", потому что на практике это просто переходы к определенному состоянию, которое может или не может быть переведено в то, что люди сочтут задачей. Вознаграждения формируются таким образом, чтобы побудить ученика со временем решать более сложные задачи, увеличивая длину пути учителя. Другой распространенной формой является включение методов, похожих на имитацию, для автоматической классификации навыков и неоднократных просьб обучаемого агента попрактиковаться в навыках [42].

Еще одна интересная идея, похожая как на имитацию, так и на обучение по учебной программе, — это сочетание политик. *Остаточное RL* определяет способ объединения двух политик для решения одной задачи. Оно не представлено как иерархическое RL, но также может рассматриваться как таковое. Идея состоит в том, что, если у вас есть точная модель для части задачи, которая является распространенной, например, в робототехнике, вы можете комбинировать эту политику с другой для достижения задач, которые намного сложнее смоделировать, таких как хватание или прикосновение. Это реализуется путем добавления политик. Мне нравится эта идея, потому что она очень проста. Вы видели много примеров изменения функции вознаграждения или функции потерь путем произвольного добавления бонусов или штрафов, но это первый пример объединения действий. Интуитивно я думаю, что это нечто большее, чем кажется на первый взгляд. Я могу представить себе более общую структуру изменения действий, которая применима к другим областям RL (например, безопасное RL) [43].

Следование учебной программе — еще одна актуальная тема, потому что это один из способов, который потенциально может помочь RL и ML масштабироваться для решения действительно сложных задач. Идея, заключающаяся в том, что вы можете спроектировать свою задачу так, чтобы получить неконтролируемое решение для изучения ряда важных навыков или, возможно, даже решений, заманчива. Дополнительные сведения можно найти в последних обзорах в *разд. "Дополнительные материалы для чтения" в конце данной главы*.

# Другие парадигмы

Исследователи любят вводить термины. Это может привести к потоку других парадигм, которые трудно интерпретировать. Часто некоторые из них уже сейчас становятся популярными, например Q-обучение, тогда как другие могут стать популярными в будущем, а могут и не стать, как метаобучение. Я представляю другие парадигмы вместе с трансфертным обучением, т. к. они не подходят больше ни к какой теме. Это большие темы сами по себе, но я считаю их достаточно продвинутыми, и поэтому их подробное обсуждение выходит за рамки этой книги.

## Метаобучение

*Метаобучение*, или обучение для обучения, направлено на разработку общего фреймворка, который оптимизирует эффективность обучения. Другими словами, если вы знали, как лучше всего учиться, вы получите алгоритм, оптимальный для вашей задачи.

Одно из наиболее важных приложений метаобучения — это количественная оценка или улучшение обучения при небольшом количестве наблюдений либо редких событиях. Люди способны изучать модели и политику на основе небольшого количества наблюдений, передавая знания из прошлого опыта или других областей.

Проблема с мета-RL заключается в том, что оно настолько абстрактно, что вы можете представить себе множество способов повышения эффективности обучения. Например, некоторые способы реализуют искусственное любопытство и скуку [44]. Другие используют генетические, эволюционные или состязательные подходы [45]. Некоторые даже комбинируют имитацию и инверсное RL [46].

Из-за всего этого шума трудно точно определить, что такое мета-RL или куда оно движется. Если исходить из предыдущего исследования, то мета-RL продолжит быть комбинацией методов RL и ML с целью более быстрого решения задач с меньшим объемом данных.

## Трансферное обучение

Передача опыта одного агента другому представляет очевидный интерес. Если вы можете обучить робота моделированию, а затем оптимально передать эти знания в реальный мир, вы можете искусственно повысить эффективность выборки вашего алгоритма RL. Исторически трансферное обучение рассматривалось отдельно, но сегодня оно, как правило, поглощается метаобучением. Однако оно по-прежнему важно для промышленного применения.

Распространенная проблема при преобразовании ориентированных на исследования алгоритмов RL в промышленные приложения заключается в том, что почти все исследовательские работы готовятся с нуля. Но в большинстве случаев подобные проблемы уже решены, о чем свидетельствует литература или ваша предыдущая практика. Нет смысла отбрасывать весь этот опыт и вытекающую из него политику.

Однако, как и в случае с мета-RL, есть много способов, с помощью которых вы можете воплотить прошлый опыт в новые модели. Вы можете повторно использовать необработанный опыт с помощью буферов воспроизведения, автоэнкодеров копирования/вставки, применять предыдущие политики, чтобы направлять новые через изучение учебной программы, или придумать какое-то совместное решение MARL. Возможности безграничны, и нет правильного пути.

По сути, существуют три типа трансфера. Первый и наиболее распространенный — это попытка перенести политики, которые используются для одной и той же задачи в том же домене. Примером этого является предварительное обучение с помощью моделирования. Второй — перенос политики в том же домене для решения несколько иной задачи. Это цель обучения по учебной программе, которая часто встречается в приложениях, где один агент может решать несколько задач, например в робототехнике. Третий — перенос политик за границы домена, где могут быть разные состояния и действия. Такая ситуация часто возникает случайно из-за частичной наблюдаемости, но классический пример — это попытка обучить алгоритм RL для игры Atari в настольный теннис, а затем использовать ту же политику для игры в Breakout.

Думаю, это все еще актуальная тема для отраслевых приложений, т. к. она позволяет создавать решения и обучать алгоритмы более эффективным способом, что часто является узким местом в реальных приложениях. Однако обсуждение данной темы выходит за рамки этой книги, поэтому, если вам интересно, предлагаю некоторые отправные точки в *разд. "Дополнительные материалы для чтения" в конце данной главы*.

# Резюме

Цель RL — изучить оптимальные стратегии, поэтому имеет смысл включить как можно больше дополнительной информации, чтобы помочь решить вашу задачу лучше и быстрее. Первый шаг на этом пути — убедиться, что ваши основные предположения верны, и проверить, можно ли улучшить определение вашей задачи. Это часто начинается с сомнений в обоснованности MDP и ведет к поиску новых путей потенциальных решений. Классическим примером является то, что многие реальные области частично наблюдаемы, но редко моделируются. Принятие частичной наблюдаемости может привести вас к решениям, которые успешно справляются с неопределенностью.

Иерархическое RL (hierarchical RL, HRL), особенно RL с учетом навыков, имеет потенциал для значительного масштабирования отраслевых решений. Если производители или поставщики оригинального оборудования смогут получить и повторно использовать "навыки", это может помочь им предоставить больше решений с использованием тех же процессов, оборудования и программного обеспечения. Это поможет открыть новые варианты использования и диверсифицировать потоки доходов. По этим причинам я считаю, что HRL играет очень важную роль в промышленности.

Мультиагентное RL (MARL) — очень популярная тема для исследований, поскольку представляет собой хороший способ решения распределенных задач. Например, лесные пожары увеличиваются как по количеству, так и по интенсивности, поэтому наличие интеллектуальных решений подобных геопространственных проблем может помочь в борьбе с ними [47]. На техническом уровне также имеет смысл решать задачу разведки, используя опыт многих агентов в одной среде, возможно, совместно. Конкурентные сценарии возникают во многих областях, особенно в тех, которые связаны с деньгами.

Использование советов экспертов для разработки стратегии — еще одна закономерность, которую не нужно объяснять. Одной из основных проблем в сложных промышленных задачах является ограничение постановки задачи и, значит, исследовательского пространства. Промышленность ожидает и требует большей производительности и повышения эффективности, и один из лучших способов добиться этого — это бутстрэппирование, как это делали *n*-шаговые методы еще в *главе 3*. На стратегическом уровне бутстрэппинг — это, по сути, процесс копирования успешных стратегий, будь то экспертное руководство или другая модель. После того как вы извлечете этот опыт, можете работать над его точной настройкой с помощью стандартных методов RL.

И наконец, RL — это быстро развивающаяся тема, и по мере того, как уровень абстракций становится все выше, наблюдается тенденция их отклонения в странных и чудесных направлениях. Некоторые из них станут следующим большим достижением, другие будут тихо сидеть в углу. Думаю, что использование метаобучения в обучении с подкреплением, например, станет более важным, поскольку инженеры получат больше практического опыта работы с RL. Считаю, что сейчас мы находимся в фазе *"что"*, и скоро инженеры будут спрашивать: *"Почему?"* Со временем, надеюсь, мета-RL представит набор идей, чтобы ответить на этот вопрос. Но, как следует из названия "мета-RL", тема очень философская, и ее трудно определить. Я уверен, что в будущем ситуация улучшится, если это действительно правильное направление.

# Дополнительные материалы для чтения

- Иерархическое обучение с подкреплением.
  - Отличное обзорное видео о технических трудностях от Doina Precup[6].
- Мультиагентное обучение с подкреплением.
  - Schwartz H. M. Multi-Agent Machine Learning: A Reinforcement Approach. — John Wiley & Sons, 2014.
  - Отличный недавний обзор, посвященный RL, от Оруджлуя и Хаджинежада [48].
  - Еще один обзор, который охватывает игры и EFG от Чжана и соавт. [49].
  - Децентрализованный MARL: отличное введение в децентрализованный MARL[7] от Кристофера Амато и сопроводительная книга Олихука и Амато [50].
  - Экспертное руководство:
    - недавнее исследование обратного RL [51].
  - Учебные курсы:
    - два недавних опроса [52, 53];
    - блог Лилиан Венг (Lillian Weng)[8] является отличным ресурсом и содержит хороший обзор учебной программы.
- Мета-RL:
  - Хуэйминь Пэн (Huimin Peng) смело написал хороший обзор о мета-RL [54].
- Трансферное обучение.
  - Отличная глава Алессандро Лазарича (Alessandro Lazaric), в которой трансферное обучение рассматривается на высоком уровне [55].
  - Интересный обзор трансфертного обучения в целом (неспецифичный RL) [56].

---

[6] См. **https://oreil.ly/v9qm3**.

[7] См. **https://oreil.ly/z7WOr**.

[8] См. **https://oreil.ly/IstUv**.

# Использованные источники

[1] Karl M., Soelch M., Bayer J., van der Smagt P. Deep variational Bayes filters: unsupervised learning of state space models from raw data // ArXiv:1605.06432. — 2017. — March. — URL: **https://oreil.ly/ixzns**.

[2] Hefny A., Marinho Z., Sun W., Srinivasa S., Gordon G. Recurrent predictive state policy networks // ArXiv:1803.01489. — 2018. — March. — URL: **https://oreil.ly/ThHPZ**.

[3] Wayne G., Hung C.-C., Amos D. et al. Unsupervised Predictive memory in a goal-directed agent // ArXiv:1803.10760. — 2018. — March. — URL: **https://oreil.ly/778pX**.

[4] Qiao Z., Muelling K., Dolan J., Palanisamy P., Mudalige P. POMDP and hierarchical options MDP with continuous actions for autonomous driving at intersections // 21st International Conference on Intelligent Transportation Systems (ITSC). — 2018. — P. 2377–2382. — URL: **https://oreil.ly/w3rv8**.

[5] Qiao Z., Tyree Z., Mudalige P., Schneider J., Dolan J. M. Hierarchical reinforcement learning method for autonomous vehicle behavior Planning // ArXiv:1911.03799. — 2019. — November. — URL: **https://oreil.ly/VmupK**.

[6] Hochberg I., Feraru G., Kozdoba M. et al. A Reinforcement learning system to encourage physical activity in diabetes patients // J. Med. Int. Res. — 2017. — Vol. 19, № 10. — P. 338. — URL: **https://oreil.ly/g0jLx**.

[7] Hallak A., Castro D. D., Mannor S. Contextual Markov decision processes // ArXiv:1502.02259. — 2015. — February. — URL: **https://oreil.ly/jiudo**.

[8] Ie E., Hsu C.-wei, Mladenov M. et al. RecSim: a configurable simulation platform for recommender systems // ArXiv:1909.04847. — 2019. — September. — URL: **https://oreil.ly/GjSoP**.

[9] Chandak Y., Theocharous G., Metevier B., Thomas P. S. Reinforcement learning when all actions are not always available // ArXiv: 1906.01772. — 2020. — January. — URL: **https://oreil.ly/gGau0**.

[10] Boutilier C., Cohen A., Daniely A. et al. Planning and learning with stochastic action sets // ArXiv:1805.02363. — 2018. — May. — URL: **https://oreil.ly/Y6Any**.

[11] Chandak Y., Theocharous G., Metevier B., Thomas P. S. Reinforcement learning when all actions are not always available // ArXiv: 1906.01772. — 2020. — January. — URL: **https://oreil.ly/gGau0**.

[12] Chandak Y., Theocharous G., Nota C., Thomas P. S. Lifelong learning with a changing action set // ArXiv:1906.01770. — 2020. — May. — URL: **https://oreil.ly/sYCBs**.

[13] Li X., Yang W., Zhang Z. A Regularized approach to sparse optimal policy in reinforcement learning // ArXiv:1903.00725. — 2019. — October. — URL: **https://oreil.ly/Ha65T**.

[14] Saleh A., Jaques N., Ghandeharioun A. et al. Hierarchical reinforcement learning for open-domain dialog // ArXiv:1909.07547. — 2019. — December. — URL: **https://oreil.ly/fY01Z**.

[15] Zhou G., Azizsoltani H., Ausin M. S. et al. Hierarchical reinforcement learning for pedagogical policy induction // Artifcial intelligence in education / ed. by S. Isotani, E. Millán, A. Ogan et al. — Lecture Notes in Computer Science. — Cham: Springer International Publishing, 2019. — P. 544–556. — URL: **https://oreil.ly/rrzsW**.

[16] Nachum O., Gu S., Lee H., Levine S. Data-efficient hierarchical reinforcement learning // ArXiv:1805.08296. — 2018. — October. — URL: **https://oreil.ly/prP31**.

[17] Eysenbach B., Gupta A., Ibarz J., Levine S. Diversity is all you need: learning Skills without a reward function // ArXiv: 1802.06070. — 2018. — October. — URL: **https://oreil.ly/HSs_I**.

[18] Li S., Wang R., Tang M., Zhang C. Hierarchical reinforcement learning with advantage-based auxiliary rewards // ArXiv: 1910.04450. — 2019. — October. — URL: **https://oreil.ly/r0zu7**.

[19] Saleh A., Jaques N., Ghandeharioun A., Shen J. H., Picard R. Hierarchical reinforcement learning for open-domain dialog // ArXiv:1909.07547. — 2019. — December. — URL: **https://oreil.ly/_nkHe**.

[20] Waradpande V., Kudenko D., Khosla M. Deep reinforcement learning with graph-based state representations // ArXiv:2004.13965. — 2020. — April. — URL: **https://oreil.ly/PX_BB**.

[21] Sharma A., Gu S., Levine S., Kumar V., Hausman K. Dynamics-aware unsupervised discovery of skills // ArXiv:1907.01657. — 2020. — February. — URL: **https://oreil.ly/BRM9C**.

[22] Zheng S., Trott A., Srinivasa S. et al. The AI economist: improving equality and productivity with AI-driven tax policies // ArXiv:2004.13332. — 2020. — April. — URL: **https://oreil.ly/230sL**.

[23] Cox T. Muddling-Through and Deep Learning for managing largescale uncertain risks // J. Benefit-Cost Anal. — 2019. — Vol. 10, № 2. — P. 226–250. — URL: **https://oreil.ly/ms1Yu**.

[24] Xiao K., Mao S., Tugnait J. K. TCP-drinc: smart congestion control based on deep reinforcement learning // IEEE. — 2019. — Vol. 7, № 11. — P. 892–904. — URL: **https://oreil.ly/w4-pt**.

[25] Xiao Y., Hoffman J., Xia T., Amato C. Learning multi-robot decentralized Macro-action-based policies via a centralized QNet // ArXiv:1909.08776. — 2020. — March. — URL: **https://oreil.ly/RgXHj**.

[26] Pham H. X., La H. M., Feil-Seifer D., Deans M. A Distributed control framework for a team of unmanned aerial vehicles for dynamic wildfire tracking // ArXiv:1704.02630. — 2017. — April. — URL: **https://oreil.ly/CtE3Q**.

[27] Tomic T., Schmid K., Lutz P. et al. Toward a fully autonomous UAV: research platform for indoor and outdoor urban search and rescue" // IEEE Robotics Automation Magazine. — 2012. — Vol. 19, № 3. — P. 46–56. — URL: **https://oreil.ly/L34Rn**.

[28] Cui J., Liu Y., Nallanathan A. Multi-agent reinforcement learning based resource allocation for UAV networks // ArXiv: 1810.10408. — 2018. — October. — URL: **https://oreil.ly/7in8V**.

[29] Lowe R., Wu Y., Tamar A., Harb J., Abbeel P., Mordatch I. Multi-agent actor-critic for mixed cooperative-competitive environments // ArXiv:1706.02275. — 2020. — March. — URL: **https://oreil.ly/yTYdm**.

[30] Matignon L., Laurent G. J., Le Fort-Piat N. Hysteretic Q-learning: an algorithm for decentralized reinforcement learning in cooperative multi-agent teams // 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems. — 2007. — P. 64–69. — URL: **https://oreil.ly/ZSWmT**.

[31] Omidshafiei S., Pazis J., Amato C., How J. P., Vian J. Deep decentralized multi-task multi-agent reinforcement learning under partial observability // ArXiv:1703.06182. — 2017. — July. — URL: **https://oreil.ly/2I9aX**.

[32] Ross S., Gordon G. J., Bagnell J. A. A Reduction of imitation learning and structured prediction to no-regret online learning // ArXiv:1011.0686. — 2011. — March. — URL: **https://oreil.ly/md4Pd**.

[33] Hester T., Vecerik M., Pietquin O. et al. Deep Q-learning from demonstrations // ArXiv: 1704.03732. — 2017. — November. — URL: **https://oreil.ly/ENvFp**.

[34] Ho J., Ermon S. Generative adversarial imitation learning // ArXiv:1606.03476. — 2016. — June. — URL: **https://oreil.ly/N0pWb**.

[35] Reddy S., Dragan A. D., Levine S. SQIL: imitation learning via reinforcement learning with sparse rewards // ArXiv:1905.11108. — 2019. — September. — URL: **https://oreil.ly/6s1Oy**.

[36] Ng A. Y., Russell S. J. Algorithms for inverse reinforcement learning // Proceedings of the Seventeenth International Conference on Machine Learning. ICML 2000. — San Francisco, CA: Morgan Kaufmann Publishers Inc., 2000. — P. 663–670.

[37] Tucker A., Gleave A., Russell S. Inverse Reinforcement learning for video games // ArXiv:1810.10593. — 2018. — October. — URL: **https://oreil.ly/yssiM**.

[38] Cai X.-Q., Ding Y.-X., Jiang Y., Zhou Z.-H. Imitation learning from pixel-level demonstrations by hashreward // ArXiv:1909.03773. — 2020. — June. — URL: **https://oreil.ly/06bk4**.

[39] Shiarlis K., Messias J., Whiteson S. Inverse reinforcement learning from failure // Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems. AAMAS 2016. — Singapore: International Foundation for Autonomous Agents and Multiagent Systems, 2016. — P. 1060–1068.

[40] Sukhbaatar S., Lin Z., Kostrikov I., Synnaeve G., Szlam A., Fergus R. Intrinsic motivation and automatic curricula via asymmetric self-play // ArXiv:1703.05407. — 2018. — April. — URL: **https://oreil.ly/Y7oEa**.

[41] Laux M., Arenz O., Peters J., Pajarinen J. Deep adversarial reinforcement learning for object disentangling // ArXiv:2003.03779. — 2020. — March. — URL: **https://oreil.ly/wA_3s**.

[42] Jabri A., Hsu K., Eysenbach B., Gupta A., Levine S., Finn C. Unsupervised curricula for visual meta-reinforcement learning // ArXiv:1912.04226. — 2019. — December. — URL: **https://oreil.ly/O_A7w**.

[43] Johannink T., Bahl S., Nair A. et al. Residual reinforcement learning for robot control // ArXiv:1812.03201. — 2018. — December. — URL: **https://oreil.ly/m1l7m**.

[44] Alet F., Schneider M. F., Lozano-Perez T., Kaelbling L. P. Meta-learning curiosity algorithms // ArXiv:2003.05325. — 2020. — March. — URL: **https://oreil.ly/azstP**.

[45] Wang R., Lehman J., Clune J., Stanley K. O. Paired open-ended trailblazer (POET): endlessly generating increasingly complex and diverse learning environments and their solutions // ArXiv:1901.01753. — 2019. — February. — URL: **https://oreil.ly/rGJoo**.

[46] Paine T. L., Colmenarejo S. G., Wang Z. et al. One-shot high-fidelity imitation: training large-scale deep nets with RL // ArXiv:1810.05017. — 2018. — October. — URL: **https://oreil.ly/Q-u9W**.

[47] Pham H. X., La H. M., Feil-Seifer D., Deans M. A Distributed control framework for a team of unmanned aerial vehicles for dynamic wildfire tracking // ArXiv:1704.02630. — 2017. — April. — URL: **https://oreil.ly/1YSSS**.

[48] OroojlooyJadid A., Hajinezhad D. A Review of Cooperative multi-agent deep reinforcement learning // ArXiv:1908.03963. — 2020. — June. — URL: **https://oreil.ly/p98r_**.

[49] Zhang K., Yang Z., Başar T. Multi-agent reinforcement learning: a selective overview of theories and algorithms // ArXiv: 1911.10635. — 2019. — November. — URL: **https://oreil.ly/rnl2q**.

[50] Oliehoek F. A., Amato C. A concise introduction to decentralized POMDPs. — 1st ed. — Springer Publishing Company, 2016.

[51] Arora S., Doshi P. A survey of inverse reinforcement learning: challenges, methods and progress // ArXiv:1806.06877. — 2019. — August. — URL: **https://oreil.ly/ZolV1**.

[52] Narvekar S., Peng B., Leonetti M., Sinapov J., Taylor M. E., Stone P. Curriculum learning for reinforcement learning domains: a framework and survey // ArXiv:2003.04960. — 2020. — March. — URL: **https://oreil.ly/_uqkd**.

[53] Portelas R., Colas C., Weng L., Hofmann K., Oudeyer P.-Y. Automatic curriculum learning for deep RL: a short survey // ArXiv:2003.04664. — 2020. — May. – URL: **https://oreil.ly/Rq2uZ**.

[54] Peng H. A Comprehensive overview and survey of recent advances in meta-learning // ArXiv:2004.11149. — 2020. — June. — URL: **https://oreil.ly/0lt2U**.

[55] Lazaric A. Transfer in reinforcement learning: a framework and a survey // Reinforcement learning: state-of-the-art / ed. by M. Wiering, M. van Otterlo. — Adaptation, learning, and optimization. — Berlin, Heidelberg: Springer, 2012. — P. 143–173. — URL: **https://oreil.ly/qFqhC**.

[56] Zhuang F., Qi Z., Duan K. et al. A comprehensive survey on transfer learning // ArXiv:1911.02685. — 2020. — June. — URL: **https://oreil.ly/xaQxM**.

ГЛАВА 9

# Практическое обучение с подкреплением

Обучение с подкреплением как предмет весьма старо; ему уже несколько десятилетий. Но только недавно оно получило достаточную известность, чтобы выйти за пределы академических кругов. Я думаю, что это отчасти потому, что пока еще нет достаточно распространенных знаний для промышленности. В подавляющем большинстве литературы до сих пор обсуждаются алгоритмы и надуманное моделирование.

Исследователи и предприниматели начинают осознавать потенциал RL. Накапливается богатый опыт, которого не было в 2015 г. Фреймворки и библиотеки следуют этой тенденции, что повышает осведомленность всех заинтересованных сторон и снижает барьер для входа.

В этой главе я хочу меньше говорить о чудовищных алгоритмических деталях и больше о самом процессе. Я хочу ответить на вопрос: "Что включает в себя настоящий проект RL?" Сначала я рассмотрю, как выглядит проект RL, и предложу новую модель для создания промышленных продуктов RL. Попутно я научу вас тому, как определять проблему RL и как сопоставить ее с парадигмой обучения. Наконец, я опишу, как спроектировать и разработать проект RL с самого начала, указав все области, на которые вам нужно обратить внимание.

## Жизненный цикл проекта RL

Типичные RL-ориентированные проекты нацелены на решение задач средствами обучения с подкреплением с самого начала, поскольку, как правило, основаны на имеющихся наработках, но иногда потому, что проектировщики понимают последовательный характер проблемы. Проекты RL также могут возникать на основе проекта машинного обучения (machine learning, ML), в котором инженеры ищут более эффективные способы моделирования задачи или повышения производительности. В любом случае жизненный цикл проекта RL сильно отличается от ML-проекта и уж совсем далёк от разработки программного обеспечения. Программная инженерия для RL — то же самое, что принципы укладки кирпичей для строительства мостов.

При разработке RL-проекта обычно проходишь через следующие эмоциональные стадии, которые, вероятно, вам знакомы. Без таких взлетов и падений карьеру инженера нельзя считать полноценной. Это выглядит примерно так.

1. *Оптимизм*: "RL невероятно, оно позволяет управлять роботами, так что наверняка поможет решить эту задачу?"

2. *Стресс и депрессия*: "Почему это не работает? Он способен управлять роботами, так почему он этого не может? Это из-за меня?"
3. *Осознание*: "Да, верно. Эта задача намного сложнее, чем управление роботами. Но я использую глубокое обучение, так что со временем оно сработает (?). Я собираюсь заплатить за дополнительные графические процессоры, чтобы ускорить его".
4. *Страх*: "Почему это все еще не работает? Это невозможно!"
5. *Упрощение*: "Что произойдет, если я удалю/заменю/изменю/дополню реальные данные?"
6. *Сюрприз*: "Ух ты! На моем ноутбуке он сошелся всего за 5 минут. Заметка для себя: не тратить деньги на графические процессоры в следующий раз..."
7. *Счастье*: "Отлично, теперь все работает; я скажу своему руководителю проекта, что свет в конце тоннеля уже виден, но мне нужен еще один рывок..."
8. *Возврат к п. 1.*

В более общем виде это верно для любого проекта, над которым я работал. Я нашел один простой прием, который помогает снизить уровень стресса и повысить скорость выполнения проекта. И если вы возьмете из этой книги только одну идею, то сделайте ее такой: *начать с простого.*

Вот и все. Простое начало заставляет вас задуматься о наиболее важных аспектах проблемы MDP. Это помогает создавать действительно большие абстракции; вы можете добавить детали позже. Абстракции помогают, потому что они высвобождают умственные ресурсы, которые в противном случае были бы сосредоточены на деталях; они позволяют передавать концепции другим людям без кровавых технических подробностей.

Я ничего не знаю о нейробиологии или психологии, но считаю, что стресс во время развития, который, как мне кажется, отличается от обычного стресса, пропорционален количеству умственных усилий, затраченных на решение задачи, и обратно пропорционален вероятности успеха. Если у меня был долгий день интенсивной сосредоточенности, которая требовалась из-за того, чтобы понять, почему эта чертова штука не работала должным образом, я бы больше нервничал и раздражался, когда увидел бы свою семью. Наличие более легкой проблемы, ведущей к большему успеху, помогает снизить уровень стресса.

Скорость проекта — это скорость, с которой идет разработка. Это вымышленная мера, созданная инструкторами процесса. Если сравнивать скорость вашей работы или вашей команды за определенный период времени (не между командами или людьми), постороннему человеку становится легче обнаружить временные проблемы. На личном уровне скорость важна по причинам, указанным в предыдущем абзаце. Если вы застряли, то должны заставить себя усерднее работать над задачей, в противном случае вы застрянете в спирали отчаяния надолго без каких-либо шансов на успех.

Одна из проблем с типичными измерениями скорости отслеживании по балльной системе заключается в том, что обычно люди не принимают во внимание слож-

ность задачи или риск. В течение недели вы можете работать над каким-то стандартным программным обеспечением и выполнять кучу одиночных сторонних задач (например, 5×1 баллов), на следующей неделе вы можете работать с моделью и биться головой о стену (1×5 баллов). У них может быть одинаковое количество оценочных баллов, но вторая неделя будет гораздо напряженнее.

Вот мой второй совет: чтобы облегчить эту проблему, *сведите цикл разработки к минимуму.*

Я думаю о цикле разработки как о времени между моими первыми нажатиями клавиш и тем моментом, когда все работает так, как я задумал. Одна репрезентативная метрика — это время между коммитами Git, если вы используете коммиты для представления рабочего кода. Чем длиннее эти циклы, тем больше вероятность неработающего кода и большего стресса. Кроме того, длинные циклы означают, что вам придется долго ждать обратной связи. Это особенно неприятно при обучении больших моделей на большом количестве данных. Если вы хотите подождать пару дней, чтобы узнать, работает ли ваш код, вы поступаете неправильно. Ваше время слишком ценно, чтобы сидеть без дела так долго. И, несомненно, вы обнаружите ошибку, и вам все равно придется отменить итерацию. Сократите время между циклами или запусками, чтобы повысить свою эффективность, и, если необходимо, проводите циклы обучения ночью; новые модели, только что вынутые из духовки утром, заставят вас чувствовать себя ребенком перед Рождеством.

Ни одна из этих идей не нова, но я считаю, что они невероятно важны при выполнении RL — по крайней мере, для моего рассудка. Так легко увлечься решением задачи с одной попытки, погрузившись в водоворот намерений. Вероятным результатом является сбой из-за несоответствия между задачей и MDP или моделью и данными. Причина кроется в слишком большой сложности и недостаточном понимании. Следуйте двум предыдущим советам, и у вас будет хороший шанс на победу.

## Определение жизненного цикла

Помимо доходов от парадигмы MDP, многие инструменты и методы, используемые в машинном обучении, могут быть напрямую перенесены в RL. Даже сам процесс похож.

### Жизненный цикл науки о данных

На рис. 9.1 представлена схема четырех основных этапов проекта из области науки о данных (data science), которые я наблюдал, работая с моими коллегами из Winder Research. Она в значительной степени соответствует классическому межотраслево-

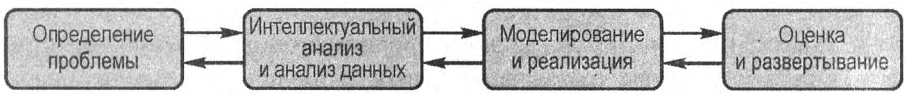



**Рис. 9.1.** Схема четырех основных этапов проекта data science, упрощенная версия модели CRISP-DM

му стандартному процессу интеллектуального анализа данных 1990-х годов, но я упростил его и подчеркнул итеративный характер.

Все проекты начинаются с определения задачи, и часто это неверно. Изменение определения может существенно повлиять на инженерные работы, поэтому вы должны убедиться, что проект решает бизнес-задачу и является технически жизнеспособным. Это одна из наиболее распространенных причин предполагаемой неудачи; *предполагаемой* потому, что технически проекты могут работать так, как задумано, но не решать реальных бизнес-задач. Часто для уточнения определения требуется несколько итераций.

Далее идет сбор и анализ данных. Примерно две трети времени специалиста по data sciense тратится на этом этапе, когда он пытается обнаружить и предоставить новую информацию, которая может помочь решить задачу, а затем проанализировать и очистить данные, готовые для моделирования.

На этапе моделирования и реализации вы создаете и обучаете модели, которые пытаются решить вашу задачу.

Наконец, вы оцениваете, насколько хорошо вы решаете задачу, используя количественные и качественные показатели производительности, и внедряете свое решение в производственную среду.

На любом из этих этапов вам может потребоваться вернуться на шаг назад — к реинжинирингу. Например, вы можете разработать доказательство концепции в Jupyter Notebook, чтобы быстро доказать, что существует приемлемое техническое решение, но затем вернуться к созданию программного обеспечения. Или вы можете обнаружить, что во время первоначальной оценки у вас недостаточно правильных данных, чтобы решить проблему на удовлетворительном уровне. Хуже этого может быть лишь то, что вы решили не ту задачу.

## Жизненный цикл обучения с подкреплением

На рис. 9.2 показан типичный процесс проекта RL. Первое, что вы заметите, — насколько похожи фазы. Цели одинаковы (решить бизнес-задачу с использованием данных), поэтому разумно ожидать, что процесс будет аналогичным. Но сходство существует только на поверхности. Если углубиться в каждую фазу, можно обнаружить, что задачи будут совершенно разными.

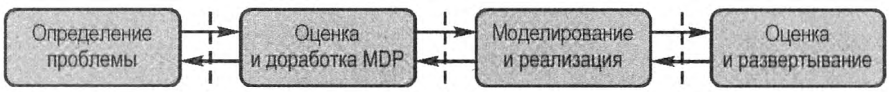



**Рис. 9.2.** Схема четырех основных этапов проекта RL

Для проектов ML нехарактерно резкое изменение определения задачи. Вы, вероятно, можете повозиться, например, со спецификацией производительности, но в основном задача, которую вы пытаетесь решить, определяется бизнес-целями. Вы должны поискать и найти любые данные, которые уже существуют в компании, получить разрешение на их использование и внедрить технологию для доступа

к ним. Если у вас нет данных, вы можете попробовать собрать новые данные или использовать внешний источник. Когда в вашем распоряжении будут данные, вы сможете потратить время на анализ, понимание, очистку и расширение, чтобы помочь решить проблему. Это первые два блока процесса анализа данных.

В RL ситуация иная, временами даже весьма необычная. Сначала перед вами будет поставлена бизнес-задача. Она носит стратегический характер, находится на более высоком уровне абстракции по сравнению с "обычными" задачами науки о данных, и у нее есть реальное определение успеха, такое как прибыль, количество подписчиков или кликов. Затем вы, да, вы, должны взять это определение успеха, тщательно выполнять проектирование задачи, чтобы она вписывалась в MDP, и разрабатывать функцию вознаграждения. До тех пор пока не появится алгоритм обратного RL (*см. разд. "Обратное RL" главы 8*), который работает в любой ситуации, вам нужно будет разработать вознаграждение самостоятельно. Это большая ноша, которую нужно возложить на свои плечи. У вас есть право решать, как количественно оценить производительность в реальном выражении. В каком-то смысле это здорово, потому что прямо показывает, как проект влияет на бизнес в понятных для человека терминах, таких как спасенные жизни (а не "точность" или "F-мера). Но это также ложится большей нагрузкой на инженера. Представьте себе агента стратегического уровня, который выбирает направление бизнеса и приводит к плохому результату. Исторически этот риск несут руководители вашей компании — отсюда и размер зарплаты, но в будущем он будет делегирован алгоритмам, подобным RL, которые разрабатываются инженерами. Я не видел никаких свидетельств того, как это влияет на иерархические структуры управления, присутствующие во многих компаниях, но ожидаю, что вы будете потрясены. Акционеры платят за производительность, а когда производительность определяется инженерами, разрабатывающими алгоритмы, означает ли это также, что они управляют компанией?

Следующий этап процесса — решить, как лучше всего учиться. В машинном обучении вам нужно только перебирать имеющиеся у вас данные. Но в RL, поскольку оно основано на взаимодействии и последовательных решениях, у вас никогда не будет всех данных, необходимых для создания реалистичных развертываний. В подавляющем большинстве случаев имеет смысл создавать симуляции для ускорения цикла обратной связи при разработке, а не использовать реальную среду. Как только у вас будет жизнеспособное решение, вы сможете приступить к экспериментам в реальной жизни.

Обратите внимание на разницу. Либо вы создаете искусственные наблюдения, либо у вас нет реальных данных, пока вы не начнете активно использовать свой алгоритм. Если бы кто-то сказал мне, что мне нужно развернуть свою причудливую модель глубокого обучения в производственной среде, прежде чем собирать какие-либо данные, я, вероятно, немного расстроился бы.

Но именно эта ситуация в RL и делает такой важной разработку состояний, действий и вознаграждений. Вы не можете надеяться на создание надежного решения, если полностью не понимаете состояние и действия, а также принцип их влияния на вознаграждение. Пытаться применить глубокое обучение к задаче, которую вы

не понимаете, — все равно что купить клюшку для гольфа за 1000 долларов и надеяться стать профессиональным гольфистом.

Есть еще кое-что, о чем я расскажу в этой главе. Но я хочу подчеркнуть, что, хотя есть сходства и передаваемые навыки, RL — совершенно другой зверь. Первые два блока на рис. 9.2 — определение проблемы и уточнение среды/MDP — я обсуждаю в этой главе. Два последних блока я оставлю для следующей главы.

# Определение проблемы: что такое проект RL?

Думаю, что к настоящему времени у вас есть довольно хорошее представление об определении RL, но я хочу посвятить этот раздел разговору о применении RL для решения повседневных задач. Напомню, вы должны основывать свою модель на марковском процессе принятия решений (*см. главу 2 и разд. "Переосмысление марковских процессов принятия решений" главы 8*). Задача должна иметь уникальные состояния, которые могут быть дискретными или непрерывными — я считаю, что часто проще воспринимать дискретные состояния, даже если они непрерывны. В каждом состоянии должна быть некоторая вероятность перехода в другое состояние (или в себя). И действие должно изменить состояние среды.

Хотя это определение охватывает почти всё, мне интуитивно трудно мыслить в терминах состояний и вероятностей переходов. Вместо этого я попытаюсь нарисовать другую картину.

## Проблемы с RL являются последовательными

Задачи RL являются последовательными, агент RL оптимизирует траектории в несколько шагов. Только это отделяет RL от ML — парадигма выбора для одношагового принятия решений. ML не может оптимизировать решения ради долгосрочного вознаграждения. Это очень важно для решения многих бизнес-задач. Я не думаю, что какой-либо генеральный директор, имеющий представление о человечности, *прямо сейчас* стал бы выжимать из потенциального клиента каждую копейку. Вместо этого бизнесмены знают, что управление отношениями и управление тем, как и когда совершаются продажи, должны быть оптимизированы для долгосрочной поддержки клиента, и это является наилучшим путем к устойчивому бизнесу.

Такая философия напрямую связана с тем, как вы будете решать отраслевые, ориентированные на клиента задачи. Вы не должны использовать модели ML для повышения эффективности рекламы или рекомендации по покупке товаров, потому что они оптимизированы не для того. Подобно прожорливому малышу, они не могут видеть общую картину. Модели ML обучены принимать наилучшее возможное решение в данный момент, независимо от будущих последствий. Например, модели машинного обучения способны оптимизировать рекомендации по товарам, но они могут советовать приобретение товаров, эквивалентных клик-приманке, — таких, которые получают много кликов, но в конечном счете разочаровывают. Со временем близорукое представление, скорее всего, будет раздражать пользователей и

может снизить прибыль, навредить удержанию клиентов или другим важным бизнес-показателям.

Алгоритмы RL идеально подходят для ситуаций, когда вы видите, что нужно принять несколько решений или среда не сбрасывается после того, как вы приняли решение. Это открывает широкий спектр возможностей, и некоторые из них были изложены в *разд. "Варианты применения обучения с подкреплением" главы 1*. Это могут быть новые задачи или проблемы, которые уже частично решены с помощью ML. В некотором смысле задача, которая уже была частично решена, лучше, чем новый проект, потому что вполне вероятно, что инженеры уже хорошо знакомы с предметной областью и данными. Но имейте в виду, что первоначальные результаты RL могут дать худшие результаты в краткосрочной перспективе, пока алгоритмы улучшаются. Посетите соответствующий веб-сайт[1], чтобы получить больше информации о приложении.

## Проблемы RL имеют стратегический характер

Мне нравится думать о программной инженерии как о способе автоматизации процессов, а машинное обучение может автоматизировать решения. RL может автоматизировать стратегии.

Эта идея подтверждается на рис. 9.3. Современный бизнес состоит из трех основных функций. У предприятий есть множество процессов, от анализа производительности до того, как использовать принтер. Включение любого из этих процессов в модель редко бывает ценным, но, как правило, происходит часто. Из-за частоты и того факта, что человеку требуется время и, следовательно, деньги для выполнения этого процесса, имеет смысл автоматизировать этот процесс с помощью программного обеспечения.

**Рис. 9.3.** Описание различных функций современного бизнеса, ценности каждого действия в этой функции и технологии, которая соответствует функции

Бизнес-решения встречаются реже, чем бизнес-процессы, но все же довольно часто. Принятие решения о выполнении искусственного дыхания, звонка по телефону или блокировки подозрительной транзакции может спасти жизни / принести прибыль / предотвратить мошенничество. Все эти задачи можно решить с помощью машинного обучения. Эти действия количественно ценны, но, как я уже сказал ранее, они

[1] См. **https://rl-book.com/applications/?utm_source=oreilly&utm_medium=book&utm_campaign=rl**.

имеют ограниченный срок службы. Они намеренно сосредоточиваются на одном переходном событии. Для принятия решений необходимы процессы, которые сделают их возможными.

Бизнес-стратегии составляют основу деятельности компании. Они разрабатываются редко, в первую очередь потому, что существует ограниченное число людей (или один), причастных к их созданию и реализации. Такие сотрудники хранят секреты и влияют на будущее множества людей, на принимаемые решения и существующие процессы. Бизнес-стратегии имеют далеко идущие последствия, такие как здравоохранение для государства или годовая прибыль компании. Вот почему оптимальные стратегии так важны и почему так важно RL.

Представьте, что вы можете доверить управление своей компанией/здравоохранением/страной обучению с подкреплением. Как это звучит? Что вы думаете?

Вы можете предположить, что это несбыточная мечта, но уверяю вас, что это не так. Налогообложение решается с помощью MARL [1]. Примеры использования в здравоохранении очевидны, их тысячи [2]. Да и компании должны автоматизироваться, чтобы стать более конкурентоспособными, в этом нет ничего нового — они занимаются автоматизацией уже более века.

Таким образом, будущее в руках инженеров — людей, которые знают, как создавать, адаптировать и ограничивать алгоритмы, чтобы они были надежными, полезными и, прежде всего, безопасными.

## Низкоуровневые индикаторы RL

Предыдущие два предположения о том, что проблемы RL последовательные и стратегические, являются высокоуровневыми. На более низком уровне вы должны искать отдельные компоненты задачи.

### Сущность

Ищите объекты, которые представляют собой конкретные вещи или людей, которые взаимодействуют со средой. В зависимости от проблемы объект может располагаться по обе стороны интерфейса среды. Например, человек, просматривающий рекомендацию, является сущностью, но он выступает частью среды. В других случаях люди играют роли симулированных агентов, как в примере с налогообложением.

### Среда

На языке предметно-ориентированного проектирования среда должна быть ограниченным контекстом. Это должен быть интерфейс, который инкапсулирует все сложности ситуации. Вам не нужно заботиться о том, стоит ли за интерфейсом симуляция или реальная жизнь; данные в интерфейсе должны быть такими же. Остерегайтесь делать среду слишком большой или слишком сложной, потому что это влияет на разрешимость проблемы. Разделение или упрощение среды облегчает работу.

## Состояние

Самый простой способ определить состояние — это наблюдать или вообразить, что происходит, когда состояние меняется. Вы можете не думать об этом как об изменении, но присмотритесь: цена акции меняется, потому что меняется основное состояние всех инвесторов; состояние робота меняется, потому что он переместился из положения *A* в положение *B*; после отображения рекомендации состояние вашей цели могло измениться с "игнорировать рекомендацию" на "нажать на рекомендацию".

Что вы можете наблюдать в среде (которая может включать сущности)? Какие функции наиболее актуальны и информативны? Как и в ML, вы должны быть осторожны и включать только чистые, информативные, некоррелированные функции. Вы можете добавлять, удалять или дополнять эти функции по своему усмотрению, но старайтесь минимизировать их количество, если это возможно, чтобы уменьшить пространство для исследования, повысить эффективность вычислений и надежность. Не забывайте, что вы можете объединять контекстную информацию из других источников, например время суток или погоду.

Область состояния может сильно различаться в зависимости от приложения. Например, это могут быть изображения, матрицы, представляющие геопространственные сетки, скаляры, отражающие показания датчиков, фрагменты текста, элементы в корзине для покупок, история браузера и многое другое. Имейте в виду, что с точки зрения агента состояние может быть неопределенным или скрытым. Часто напоминайте себе, что агент не видит того, что видите вы, потому что у вас есть преимущество в виде взгляда оракула на проблему.

## Действие

Что происходит со средой, когда вы применяете действие? Внутреннее состояние среды должно иметь возможность измениться (а может и не измениться, потому что действие должно было оставаться на месте, или оно было плохим), когда вы применяете действие. В идеале хочется наблюдать это изменение состояния, а если нет, подумайте о поиске наблюдений, которые раскрывают эту информацию. Опять же, попробуйте ограничить количество действий, чтобы уменьшить пространство исследования.

## Количественная оценка успеха или неудачи

Как бы вы определили успех или неудачу в этой среде? Можете ли вы придумать способ количественной оценки? Значение должно соответствовать определению задачи и использовать единицы, понятные как вам, так и всем заинтересованным сторонам. Если вы работаете над увеличением прибыли, оцените ее в денежном выражении. Если вы пытаетесь повысить частоту кликов, вернитесь и попробуйте найти денежную сумму за клик. Подумайте о настоящей причине, по которой ваш клиент/босс/заинтересованное лицо привлекли вас к выполнению этого проекта. Как вы можете количественно оценить результаты по терминологии, которую они понимают и ценят?

## Типы обучения

Цель агента — изучить оптимальную политику. В оставшейся части книги исследуется, *как* агент обучается с помощью различных алгоритмов, основанных на ценностях и градиентах политики. Но что насчет того, *когда* он обучится?

### Онлайн-обучение

Подавляющее большинство примеров, литературы, сообщений в блогах и фреймворков предполагают, что ваш агент учится в Интернете. Здесь ваш агент взаимодействует с реальной или смоделированной средой и одновременно учится. До сих пор в этой книге я рассматривал только этот режим. Одна из основных проблем этого подхода — эффективность выборки, количество взаимодействий со средой, необходимое для изучения оптимальной политики. В предыдущие два десятилетия исследователи преследовали цель повысить эффективность выборки, улучшая гарантии исследования и обучения. Но, по сути, всегда будут накладные расходы на разведку, задержки выборки из-за стохастической аппроксимации и все важные проблемы стабильности. Лишь недавно они начали рассматривать возможность обучения на основе журналов — файлов с записями о событиях в хронологическом порядке.

### Автономное или пакетное обучение

До недавнего времени считалось, что все обучение должно происходить в режиме онлайн из-за неотъемлемой связи между действием агента и реакцией окружающей среды [3]. Но исследователи обнаружили, что можно учиться из пакета сохраненных данных, например данных в буфере воспроизведения. Поскольку эти данные больше не взаимодействуют со средой, их также называют *обучением в автономном*, или *пакетном, режиме*. На рис. 9.4 представлена идея. Во-первых, данные генерируются онлайн с использованием политики (возможно, случайной) и сохраняются в буфере. В автономном режиме новая политика обучается на основе данных в буфере. Затем новая политика развертывается для дальнейшего использования.

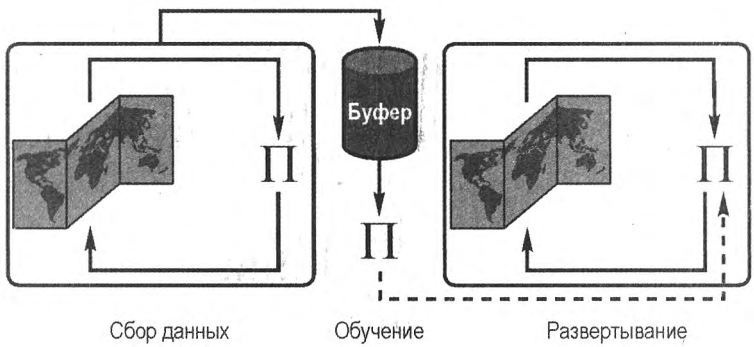



**Рис. 9.4.** Пакетное обучение с подкреплением

Основное преимущество автономного обучения — повышение эффективности выборки. Вы можете собирать или регистрировать набор данных и использовать их для обучения моделей столько раз, сколько захотите, без воздействия на среду. Это невероятно важная функция во многих областях, где сбор новых образцов является дорогостоящим или небезопасным.

Но постойте. Если я собираю и выполняю тренировку на статических данных, разве это не чистое контролируемое обучение? Ответ — да, в некотором роде. Оно контролируется в том смысле, что у вас есть пакет данных с помеченными переходами и вознаграждениями. Но это не значит, что вы можете использовать какой-либо старый алгоритм регрессии или классификации. Данные (предполагается, что они создаются) генерируются MDP, и поэтому алгоритмы, разработанные для поиска оптимальных политик для MDP, по-прежнему полезны. Главное отличие в том, что алгоритмы больше не могут проводить разведку.

Если агент не может исследовать, то он не может пытаться улучшить политику, "восполняя пробелы", что было основной причиной быстрого прогресса в области RL за последние десять лет. Еще один аргумент в пользу выбора PL заключается в том, что агенту не позволяется задавать сомнительные вопросы о действии. Опять же, это теоретический метод обеспечения более надежных политик, позволяющий им задавать вопросы: "А что, если?". По сути, распределение обучающих данных (для состояния, действий и вознаграждений) отличается от того, которое наблюдается при развертывании. Представьте, как это влияет на типичный алгоритм обучения с учителем.

В принципе, *любой* алгоритм RL вне политики может быть использован в качестве автономного алгоритма RL путем удаления из алгоритма части исследования. Однако на практике это проблематично по ряду причин. Методы оценки трудны, потому что они полагаются на точные оценки стоимости. Но многие состояния или действия не будут присутствовать в буфере, и поэтому оценки окажутся неточными. Методы градиента политики сталкиваются с большими трудностями из-за сдвига распределения, который в лучшем случае создает градиенты, указывающие в неправильном направлении, а в худшем — приводящие в никуда. Эти проблемы усугубляются, когда вы используете чрезмерно определенные модели, такие как глубокое обучение [4].

На момент написания этой книги исследователи упорно трудились над решением этих проблем, чтобы обнаружить следующий большой выигрыш в эффективности выборки. Одним из направлений исследований является дальнейшее уточнение гарантий стабильности базовых алгоритмов RL, чтобы они не вели себя некорректно при обнаружении нестабильных распределений, состояний или действий с недостаточной выборкой. Лучшие теоретические гарантии при использовании глубокого обучения — очевидный путь, который резко улучшит ситуацию. В других подходах предпринимаются попытки использовать такие методы, как имитационное обучение для использования политик, генерирующих положительные результаты [5]. Еще более простой подход — обучить автоэнкодер на буфере для создания моделируемой среды, в которой агенты *могут* проводить разведку данных [6]. На практике эти методы уже используются, применяются и демонстрируют впечатляющие

результаты, особенно в робототехнике[2] [7]. Таким образом, с практической точки зрения тест DeepMind RL Unplugged[3] может оказаться полезным инструментом для поиска или прототипирования идей [8].

## Параллельное обучение

В литература встречается сочетание методов обучения, которые находятся где-то между мультиагентным RL (MARL) и многопроцессорными фреймворками. Они достаточно интересны, поэтому стоит упомянуть их здесь.

Продолжительность ваших циклов обратной связи продиктована сложностью задачи: чем сложнее задача, тем больше времени требуется на ее решение. Частично это вызвано возрастающей сложностью модели, но в первую очередь это проблема разведки. Большие пространства состояний и действий требуют большего изучения для "восполнения пробелов".

Многие исследователи сосредоточились на улучшении разведки в случае с одним агентом, что сотворило чудеса. Но это только в определенной степени. В конечном счете агенту все равно необходимо посетить все пространство "состояние — действие". Один из вариантов — запустить несколько агентов одновременно (*см. разд. "Масштабирование RL" главы 10*), но если они используют один и тот же алгоритм исследования, то могут стремиться исследовать одно и то же пространство, что опять же сводит на нет эффективность по мере увеличения пространства действий.

Вместо этого почему бы не проводить разведку *одновременно*, когда группы агентов имеют совместную задачу исследования пространства "состояние — действие". Однако помните, что этот термин перегружен; исследователи использовали слово "параллельный" для обозначения чего угодно, от кооперативного MARL до параллельного программирования для RL [9, 10]. Я использую этот термин для обозначения цели выполнения скоординированного исследования одной и той же среды, впервые предложенной в 2013 г. [11].

Реализация этой идеи в некоторой степени зависит от предметной области. В качестве одного из примеров можно привести то, что многие компании взаимодействуют с тысячами или миллионами клиентов параллельно и пытаются максимизировать вознаграждение, локальное для клиента, например пожизненную ценность или лояльность. Но клиенты находятся в пределах одной и той же среды; у компании есть только ограниченный набор товаров на складе, а вы владеете примерно одинаковой для всех клиентов демографической информацией. Знания, полученные от одного клиента, должны быть переданы другим, поэтому ситуация подразумевает либо централизованное обучение, либо, по крайней мере, совместное использование параметров.

Я не думаю, что существует один правильный ответ, потому что есть много мелких проблем, которые требуют уникальной адаптации. Например, что произойдет, если

---

[2] См. **https://oreil.ly/cXXDc**.

[3] См. **https://oreil.ly/hdtXK**.

вы введете возможность переназначения клиентов с помощью электронных писем или рекламы? Это улучшило бы показатели производительности, но вознаграждение может быть отложено. Таким образом, вам нужно будет усилить способность "пропускать" действия и иметь дело с частичной наблюдаемостью.

Димакопулу и Ван Рой формализовали эту идею в 2018 г. и доказали, что скоординированное исследование может увеличить как скорость обучения, так и масштабирование для большого числа агентов [12]. Их решение на удивление простое: убедитесь, что у каждого агента есть случайное начальное число, которое максимизирует исследование. Преимущество этого подхода в том, что вы можете использовать стандартные алгоритмы RL, и каждый агент независимо стремится к поиску оптимальной производительности. Он также хорошо адаптируется к новым данным, поскольку начальные условия многократно рандомизируются. Однако проблема заключается в том, что оптимальная техника отбора проб семян различна для каждой задачи.

Таким образом, при параллельном обучении агенты независимы и не взаимодействуют. Агенты могут иметь разные взгляды на среду, как и у разных клиентов разные взгляды на ваши продукты. Несмотря на другое название, это очень похоже на MARL с централизованным обучением. Поэтому, если вы работаете над проблемой, которая соответствует такому описанию, обязательно исследуйте MARL, а также параллельное RL.

## Обучение без сброса

В подавляющее большинство примеров и алгоритмов включено неявное использование "эпизодов", которые подразумевают, что окружающая среда может быть "сброшена". Сброс к первоначальному состоянию имеет смысл для некоторых сред, где начальная позиция известна, фиксирована или легко идентифицируется. Но реальная жизнь не допускает "переделок", хотя порой бы я от них не отказался, плюс такая переделка потребовала бы создать Всевидящее око. Для того чтобы избежать оруэлловской версии фильма "День сурка", во многих приложениях требуется какой-то способ не указывать сброс.

Ранние решения предполагали, что агенты могут научиться выполнять сброс [13]. Агенты контролируют свои действия, поэтому вы можете сохранять предыдущие действия в буфере. Для того чтобы вернуться к исходной точке, вы можете "отменить" эти действия. Такой подход работает во многих областях, потому что у пространств действий есть естественная противоположность. Но неопределенность действий и неестественные "отмены" не позволяют этому подходу быть общеприменимым.

Другая идея состоит в том, чтобы предсказать момент, когда произойдет завершение работы, сброс или небезопасное поведение, и остановить агента до того, как это случится [14]. Но опять же, это требует надзора, поэтому еще одно предложение — изучить функцию, предсказывающую, когда произойдет сброс, как часть самого обучения [15]. Как видно из используемой терминологии, здесь есть намеки на метаобучение. Но ни один из этих подходов не является полностью открытым для сброса; тогда почему бы не исключить сбросы полностью?

Технически вы можете никогда не сбрасывать среду и использовать коэффициент дисконтирования меньше единицы, чтобы предотвратить стремление оценок стоимости к бесконечности. Но сброс важнее, чем вы думаете. Он позволяет агентам начать все сначала, по аналогии, когда вы бросаете скучную работу и начинаете захватывающую карьеру в инженерном деле. Агентам-исследователям очень легко застрять и провести значительное время в подобных состояниях. Единственный способ выбраться из колеи — это дать толчок кому-то или чему-то. Сброс — идеальное решение этой проблемы, т. к. он дает агенту еще один шанс опробовать другой путь.

Связанная с этим проблема заключается в том, что когда агент может достичь цели или глобального оптимума, тогда лучшее следующее действие — оставаться на месте, упиваясь целью. Опять же, вам нужен сброс или толчок для исследования большего количества неоптимальных состояний, чтобы сделать политику еще надежнее.

Так что вместо того, чтобы учиться перезагружать, может быть, лучше научиться отталкивать агента от цели и от областей, в которых он может застрять. Или предоставьте простой контроллер для перетасовки среды, такой как предложили Чжу и соавт., что приведет к созданию надежных роботизированных политик[4] без сбросов [16].

В заключение можно сказать, что обучение без сброса возможно в некоторых простых приложениях, но вполне вероятно, что ваши агенты застрянут. В какой-то момент вам нужно будет подумать, можете ли вы манипулировать своей средой, чтобы переместить агента в другое состояние? Вы также можете заставить агента перейти в другое состояние. Опять же, различные проблемные области будут иметь разные решения.

# Проектирование и доработка RL

Задачи RL, как и любые инженерные проблемы, имеют аспекты высокого и низкого уровней. Изменение дизайна верхнего уровня влияет на реализацию нижнего уровня. За неимением лучшего термина я обозначаю анализ и проектирование высокого уровня как *архитектурные* проблемы. К ним относятся уточнение модели марковского процесса принятия решений, определение интерфейсов, определение взаимодействий между компонентами и того, как агент будет учиться. Эти решения влияют на реализацию или *технические* детали.

Я уже много говорил об архитектурных аспектах высокого уровня, поэтому в данном разделе хочу обсудить детали реализации. Я стараюсь охватить как можно больше основных проблем, но многие детали реализации зависят от предметной области и задачи. В общем, я рекомендую вам поискать в литературе соответствующие примеры, а затем изучить детали реализации в поисках интересных лакомых кусочков. Эти детали, которые в отчете исследователя часто замалчиваются, дают жизненно важное представление о том, что работает, а что — нет.

---

[4] См. **https://oreil.ly/VLwJb**.

## Процесс

Прежде чем я углублюсь в детали, думаю, стоит в общих чертах объяснить, как я выполняю инженерную работу. Я считаю, что большинство инженерных задач, независимо от уровня абстракции, имеют тенденцию следовать некоторому варианту цикла "наблюдай, ориентируйся, решай, действуй" (observe, orient, decide, act, OODA). Якобы научный метод, цикл OODA — простой четырехэтапный подход к решению любой проблемы, популяризированный американскими военными. Идея состоит в том, что вы можете повторять этот цикл, чтобы использовать контекст и быстро принять решение, при этом понимая, что решения могут измениться по мере поступления дополнительной информации. На рис. 9.5 показан этот цикл в инженерных терминах, где я предлагаю использовать слова *"анализ"*, *"проектирование"*, *"реализация"* и *"оценка"*, но они означают одно и то же.

Самый важный аспект заключается в том, что это бесконечный процесс. На каждой итерации цикла вы оцениваете, где вы находитесь, и анализируете, что можно улучшить. Затем вы выбираете следующее лучшее улучшение и внедряете его.

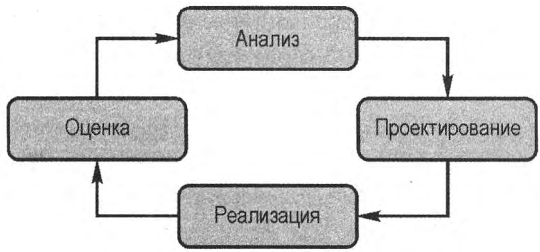



**Рис. 9.5.** Процесс доработки RL

Любого количества улучшений никогда не будет достаточно, и всегда найдется еще одно, которое вы могли бы сделать. Вы должны быть осторожны, чтобы не застрять в этом цикле, вы можете сделать это, вспомнив распределение Парето[5]: 80% решения займет 20% времени, и наоборот. Подумайте, действительно ли реализация этих последних 20% того стоит, учитывая все остальные преимущества, которые вы могли бы предоставить за это время.

## Инженерия среды

Я считаю, что наиболее продуктивным первым шагом проекта RL является создание среды. На протяжении всей книги я использую этот термин для обозначения абстракции мира, в котором действует агент. Это интерфейс между тем, что происходит на самом деле, и тем, что хочет агент (*см. разд. "Обучение с подкреплением" главы 1*).

Однако именно сейчас я говорю о реализации этого интерфейса. Независимо от того, работаете ли вы над реальными проектами, представляя реальные сделки, или создаете симуляцию ветряной турбины, вам нужна реализация интерфейса среды.

---

[5] См. **https://oreil.ly/iAWCX**.

Преимущество сосредоточения на этом в первую очередь состоит в том, что вы ближе познакомитесь с данными. Это соответствует этапу "понимания данных" в модели CRISP-DM. Понимание своей среды имеет решающее значение для эффективного решения промышленных задач. Если вы этого не сделаете, то, как правило, произойдет то, что я называю "очередью улучшений". Подбрасывая кучу различных идей для решения проблемы, вы, вероятно, найдете ту, которая попадет в цель, но большинству из них это не удастся. Этот тип работы непродуктивен, неэффективен и неразумен.

Вместо этого следует погрузиться в среду, ставить научно и технически обоснованные вопросы — да, очевидно, это такое слово — и реализовывать функциональные возможности, отвечающие на эти вопросы. Со временем вы поймете проблему и сможете объяснить, *почему* улучшения важны, а не просто заявить, что они есть.

## Реализация

OpenAI Gym естественным образом стал де-факто интерфейсом среды в сообществе RL [17]. Фактически это одна из самых цитируемых статей о RL всех времен. Вы, вероятно, сталкивались с этим раньше, но на всякий случай, Gym — это фреймворк Python для выражения среды. Он предоставляет полезные примитивы, такие как определения для действия и представления пространства состояний, и обеспечивает простой набор функций через интерфейс, который должен соответствовать большинству приложений. Он также поставляется с большим набором примеров сред, которые используются в Интернете в качестве игрушечных примеров.

С промышленной точки зрения интерфейс — отличный способ смоделировать вашу задачу и помочь усовершенствовать ваш MDP. И поскольку этот фреймворк настолько популярен, вполне вероятно, что вы сможете быстро протестировать различные алгоритмы, т. к. большинство ожидает интерфейса Gym.

Я не буду вдаваться в подробный пример реализации, потому что это выходит за рамки книги, но вы можете найти много отличных примеров, выполнив быстрый поиск в Интернете или посетив веб-сайт Gym GitHub[6].

## Моделирование

Моделирование (симуляция) должно отражать наиболее важные аспекты реальной задачи. Оно не предназначено для идеального представления, хотя многие разработчики-любители, исследователи и компании тратят огромное количество времени на то, чтобы сделать модели более реалистичными. Преимущество симуляции состоит в том, что в рамках моделирования проще, дешевле и быстрее работать, чем в реальной жизни. Она сокращает продолжительность циклов обратной связи

[6] См. **https://oreil.ly/Geupc**.

по развитию и увеличивает продуктивность. Но точные преимущества зависят от задачи; иногда учиться в реальной жизни легче и проще, чем создавать симуляцию.

Сначала вам следует выполнить поиск, чтобы узнать, не пытался ли кто-нибудь уже создать симуляцию. Понимание фундаментальных аспектов проблемы — физики, биологии или химии — часто бывает труднее, чем ее решение, хотя создание симуляции, безусловно, расширит ваши знания о предмете, так что это может иметь смысл лишь по этой причине. Многие коммерческие симуляторы, такие как MuJoCo или Unity, доступны, но не бесплатны. Вам нужно будет решить, стоит ли того решение вашей задачи. В общем, я рекомендую оценить, является ли моделирование конкурентным преимуществом для вашего бизнеса или продукта. Если это так, возможно, стоит инвестировать в развитие; в противном случае сэкономьте много времени и купите готовые продукты. Еще я обнаружил, что поставщики симуляторов чрезвычайно полезны и профессиональны, поэтому попросите их о помощи. В их интересах, чтобы ваша идея или бизнес были успешными, т. к. это означает для них будущий доход.

Если вы все же решите реализовать симулятор самостоятельно, вам нужно будет построить модели для моделирования задачи и решить, какие аспекты раскрыть. Здесь решающую роль играет разработка MDP. В общем, вы должны попытаться выявить те же аспекты задач, которые можно наблюдать в реальности. В противном случае, попытавшись перенести полученные знания в реальную жизнь, вы обнаружите, что ваш агент ведет себя не так, как вы ожидаете.

Точно так же многие исследователи пытаются взять модели, обученные имитационному моделированию, и применить их к реальным задачам. Это, безусловно, возможно, но полностью зависит от репрезентативности моделирования. Если это ваше намерение, вам следует неявно связать реализации моделирования и реальной жизни. Подавляющему большинству алгоритмов и моделей не нравится, когда вы меняете базовое состояние или пространство действий. Моделирование идеально подходит для экспериментальных проектов для подтверждения технической жизнеспособности идеи. Но будьте осторожны, чтобы не возлагать слишком большие надежды на симуляцию, т. к. нет гарантии, что политика, разработанная вашим агентом, возможна в реальной жизни. Вполне вероятно, что реальные проекты могут потерпеть фиаско, даже если симуляции работают идеально. Чаще всего моделирование упускает из виду какой-то важный аспект или информацию.

## Взаимодействие с реальной жизнью

Если ваш опыт моделирования обнадеживает или вы решили, что проще пропустить его, то следующим шагом будет создание другого (Gym) интерфейса для взаимодействия с реальной жизнью. Разработка такой реализации часто оказывается намного проще, т. к. вам не нужно понимать и реализовывать лежащие в основе механизмы; природа делает это за вас. Но адаптация ваших алгоритмов к проблеме полностью зависит от понимания и контроля данных.

Основными проблемами реальной реализации являются эксплуатационные аспекты, такие как масштабирование, мониторинг, доступность и т. д. Но архитектура и

дизайн по-прежнему играют важную роль. Например, в реальной жизни существуют очевидные проблемы безопасности, которых нет в симуляции. В некоторых областях необходимо потратить значительное количество времени на обеспечение безопасной и надежной работы. Другие распространенные проблемы включают частичную наблюдаемость, которой нет во всемогущем моделировании, и чрезмерное количество стохастичности. Вы полагаете, что двигатели перемещаются в положение, о котором вы им сообщаете, но часто это не так.

Не думаю, что этот небольшой раздел в достаточной мере отражает количество инженерного времени, которое может быть потрачено на реализацию в реальном мире. Конечно, реализации сильно зависят от предметной области, но основная проблема в том, что эта работа в значительной степени является чистым исследованием. Трудно точно оценить, сколько времени займет работа, потому что ни вы, ни кто-либо еще в мире раньше не занимались именно этой проблемой. Вы цифровой исследователь. Не забудьте напомнить об этом менеджерам проектов. И помните установку, упомянутую в начале этой главы. *Начни с простого*. Жизнь достаточно тяжела и без того, чтобы ты делал ее еще тяжелее.

### Обучение моделированию

Одно техническое дополнение, которое я хочу добавить, — это идея сделать моделирование задачей оптимизации. Если вы можете сформулировать свое моделирование так, чтобы основные компоненты были параметризованы или, что еще лучше, имели формальные математические определения, то вполне возможно оптимизировать ваше моделирование, чтобы оно больше походило на реальную жизнь.

Все, что вам нужно, — это источник достоверной информации: данные журнала из реальной среды, прошлые эксперименты или, возможно, более сложное моделирование. Затем вы можете автоматически настроить параметры вашего моделирования, чтобы сгенерировать данные, которые примерно соответствуют модели. Можете рассматривать это как задачу поиска гиперпараметров методом грубой силы, используя подходящую меру "расстояния от реальной жизни", или в духе этой книги использовать RL для изучения оптимальных параметров [18]! Опять же, это очень "мета", но мне нравится идея унификации цепочки создания стоимости через абстракцию MDP и RL.

## Инжиниринг состояния или обучение представлениям

В науке о данных *конструирование признаков* — это искусство/акт улучшения необработанных данных для того, чтобы сделать их более информативными и репрезентативными для поставленной задачи. Как правило, более информативные признаки облегчают решение проблемы. Значительное количество времени после проверки концепции тратится на очистку, удаление и добавление признаков для увеличения производительности и повышения надежности решения.

Мне нравится использовать подобную терминологию для обозначения инженерии состояния, хотя я не видел, чтобы она широко применялась где-либо еще. *Инжини-*

*ринг состояния*, часто называемый *обучением представлениям* состояния, — это набор методов, направленных на улучшение того, как наблюдения представляют состояние. Другими словами, цель состоит в том, чтобы создать более качественные представления о состоянии. Однако нет правильного способа сделать это, поэтому в первую очередь ориентируются на такие показатели, как производительность и надежность.

Хотя есть параллели, это не конструирование признаков. Цель состоит не в том, чтобы решить проблему путем предоставления более простых и информативных признаков. Цель — как можно лучше представить состояние среды, а затем позволить политике решить задачу. Причина этого в том, что в большинстве промышленных задач очень маловероятно, что вы сможете создать признаки, способные конкурировать с оптимальной многоэтапной политикой. Конечно, если вам это удастся, тогда следует подумать об использовании стандартных методов обучения с учителем.

Как и в конструировании признаков, важна экспертиза в предметной области. Добавление, казалось бы, простого признака может значительно улучшить производительность, гораздо больше, чем использование различных алгоритмов. Например, Калашников и соавт. обнаружили, что в своих экспериментах с робототехникой, в которых в качестве представления состояния использовались изображения, добавление простого измерения "высота захвата" улучшило способность робота поднимать невидимые объекты на колоссальные 12% [19]. Это говорит о том, что даже несмотря на то, что исследователи всегда пытаются учиться на пикселах, простые, надежные измерения истинного положения вещей являются гораздо лучшими представлениями состояния.

Обратите внимание на то, что во многих научных работах и статьях в Интернете как обучение представлениям, так и политика/действия объединяются в одно. Но я считаю их отдельными задачами, т. к. цель обучения представлениям состоит в том, чтобы сконцентрироваться на лучшем представлении состояния, а не на улучшении того, как политика генерирует действия.

## Перспективные модели обучения

В некоторых случаях есть шанс получить доступ к основной истине временно, например в хорошо оснащенном лабораторном эксперименте, но не в произвольной ситуации. Вы можете построить контролируемую модель для прогнозирования состояния на базе наблюдений и отправить ее полевым агентам. Такой подход также называется изучением *перспективной модели*.

## Ограничения

Ограничение пространства состояний помогает ограничить объем исследований, необходимых агенту. В целом политика улучшается, когда состояния имеют повторные посещения, поэтому ограничение количества состояния ускоряет обучение.

Применяйте ограничения, используя априорные значения, достоверную информацию, формально определяемую распределением вероятностей. Например, если вы

знаете, что признаки должны быть строго положительными, убедитесь, что ваша политика знает об этом и не пытается достичь этих состояний.

## Преобразование (уменьшение размерности, автоэнкодеры и модели мира)

Уменьшение размерности — это задача науки о данных, в процессе решения которой пытаются уменьшить количество функций, не удаляя информацию. Традиционные техники тоже полезны в RL. Например, использование анализа главных компонент сокращает пространство состояний и, следовательно, уменьшает время обучения [20].

Но жемчужиной обучения представлениям являются различные воплощения автоэнкодеров для реконструкции наблюдений. Идея проста: учитывая набор наблюдений, обучите нейронную сеть формировать внутреннее представление и регенерировать входные данные. Затем подключитесь к этому внутреннему представлению и используйте его как новое состояние.

Обычно реализации пытаются уменьшить размерность ввода с помощью нейронной архитектуры, которая постепенно сжимает пространство состояний. Это ограничение заставляет сеть изучать абстракции более высокого уровня и, надеюсь, представление важного для нас состояния. Однако есть предположения, что увеличение размерности может улучшить производительность [21]. Я по-прежнему скептически отношусь к этому утверждению, поскольку оно активно поощряет переобучение.

Изучение представлений отдельных наблюдений — это одно, но MDP последовательны, поэтому важные временные особенности заблокированы. Новый класс автоэнкодеров, основанных на MDP, начинает рассматривать эту проблему в большей степени как задачу прогнозирования. Учитывая упорядоченный набор наблюдений, эти автоэнкодеры пытаются предсказать, что произойдет в будущем. В результате автоэнкодер вынужден изучать временны́е функции, а также функции с отслеживанием состояния. Конкретная реализация индивидуальна, но идея неизменна [22, 23].

Термин *"модели мира"* после одноименной научной статьи указывает на намерение попытаться построить модель среды из автономных данных, чтобы агент мог взаимодействовать со средой так, как если бы он был в сети. Эта идея, очевидно, важна для практических реализаций, где сложно обучаться на реальных данных, и поэтому данный термин стал популярным для описания любого метода, который пытается моделировать среду [24]. Ха и Шмидхубер заслуживают особой похвалы за превосходную интерактивную версию[7] своей работы. Я считаю, что эта работа представляет собой конкретную реализацию обучения представлениям, которая может оказаться полезной или бесполезной в зависимости от вашей ситуации, но это очень эффективный инструмент, помогающий сегментировать процесс RL.

[7] См **https://oreil.ly/wHT_w**.

Перечисленные методы могут быть осложнены проблемами MDP, такими как частичная наблюдаемость, но в целом исследования автоэнкодера могут быть непосредственно применены к проблемам RL. Если это важно для вашего приложения, рекомендую вам изучить эти работы. Фреймворки также начинают видеть значение в представлении состояния, например SRL-zoo[8], коллекции методов обучения представлению, ориентированных на PyTorch, которые могут оказаться полезными.

## Разработка политики

Политика RL отвечает за отображение представления состояния в действие. Хотя это звучит просто, на практике вам нужно принимать осторожные решения относительно функции, которая реализует такое сопоставление. Как и в машинном обучении, выбор функции определяет пределы производительности и надежность политики. Это также влияет на обучаемость; использование нелинейных политик может вызвать расхождения и, как правило, требует гораздо больше времени для обучения.

Я обобщаю этот раздел под заголовком *"разработка политики"*, где цель состоит в том, чтобы разработать политику, которая наилучшим образом решает задачу с учетом набора ограничений. Подавляющее большинство литературы по RL посвящено поиску улучшенных способов определения политики или улучшения их теоретических гарантий.

Политика принципиально ограничена обновлением Беллмана, поэтому, если не произойдет серьезного теоретического сдвига, все политики преследуют схожую цель. Исследователи вводят новизну, добавляя новые способы представления состояния, улучшая исследование или наказания за некоторые другие ограничения. В основе всех алгоритмов RL лежат ценности или политики.

Учитывая это сходство между всеми алгоритмами, я визуализирую политику как имеющую три различные задачи (рис. 9.6). Сначала она должна преобразовать

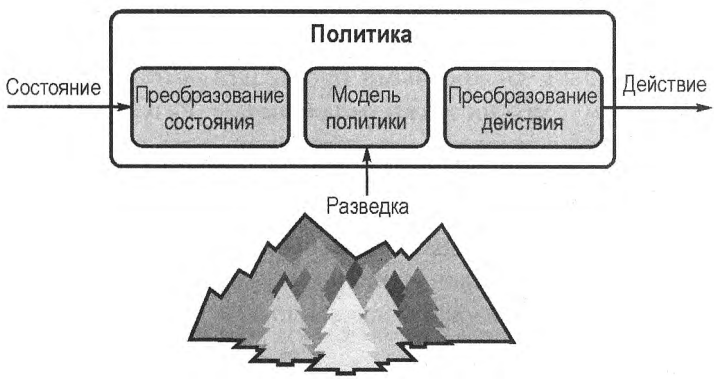



**Рис. 9.6.** Изображение трех фаз политики

[8] См. **https://oreil.ly/Gfr5r**.

представление состояния в формат, который может использоваться внутренней моделью. Внутренняя модель — это механизм принятия решений, который изучает все, что ему нужно, чтобы преобразовать состояние в возможные действия. Политика должна быть изучена, и вы можете полностью контролировать этот процесс. Наконец, политика должна преобразовать скрытое представление в действие, подходящее для среды.

Инженерная задача состоит в том, чтобы разработать функции для эффективного и надежного выполнения этих трех этапов. Определение модели политики было предметом большей части этой книги. В следующих разделах я опишу, как использовать дискретные или непрерывные состояния и действия.

## Дискретные состояния

*Дискретный* признак может принимать только определенные значения, но может иметь бесконечный диапазон. Подавляющее большинство данных поступает в форме, квантованной аналого-цифровыми преобразователями, поэтому вы можете рассматривать большинство данных как дискретные. Но на практике, когда люди говорят "дискретное пространство", они обычно имеют в виду целочисленное пространство состояний, где состояние может принимать только целые значения. В большинстве случаев квантованные данные преобразуются в число с плавающей запятой и считаются непрерывными.

Входы дискретного состояния полезны благодаря их простоте. Меньше состояний, которые агент должен исследовать, и поэтому производительность и время обучения могут быть уменьшены, даже предсказуемы. Задачи с меньшими пространствами состояний будут сходиться быстрее, чем с большими.

Дискретное пространство состояний — это любая среда, подобная GridWorld. Геопространственные задачи — хороший тому пример. Несмотря на то что позиции могут быть закодированы с помощью непрерывных характеристик, таких как широта и долгота, часто с вычислительной точки зрения и интуитивно проще преобразовать карту в сетку, как сделали это Чаудхари и соавт. при оптимизации политики маршрутизации в Нью-Йорке [25].

Считается, что проблемы с дискретными пространствами состояний легче решить главным образом потому, что их легко отлаживать и визуализировать. О дискретных областях часто легко рассуждать, что позволяет задавать точные теоретические априорные и базовые значения. Вы можете напрямую сравнивать политики с этими предположениями, и такой подход упрощает отслеживание прогресса обучения и устранение проблем по мере их возникновения. Еще одно преимущество состоит в том, что ценностные функции могут храниться в словаре или справочной таблице, поэтому вы можете использовать промышленно надежные технологии, такие как базы данных.

Но наиболее важным преимуществом является то, что дискретные пространства фиксируются; нет никакого "промежуточного" состояния, поэтому нет необходимости в какой-либо аппроксимации. Это означает, что вы можете использовать базовое Q-обучение. Применение более простого, более теоретически надежного

алгоритма, такого как Q-обучение, приводит к более простым и надежным реализациям, что является очень важным фактором при использовании этих моделей в производстве.

Проблема, однако, в том, что естественные дискретные состояния встречаются редко. В большинстве реализаций инженер принудительно применяет дискретное состояние к задаче, чтобы получить вышеупомянутые преимущества. Эта произвольная дискретизация, вероятно, будет в лучшем случае неоптимальной и приведет к потере деталей информации. Вопреки заманчивости аргументов в пользу дискретных пространств состояний, маловероятно, что они обеспечат наилучшую производительность. Например, в предыдущем примере ширина каждой ячейки в сетке Нью-Йорка составляла около 2 миль. Даже после оптимизации позиционирования автомобилей с функцией райдшеринга[9] они могут оказаться в 2 милях от своего клиента.

## Непрерывные состояния

*Непрерывный* признак — это бесконечный набор реальных значений. Они могут быть ограниченными, но между соседними значениями нет "шага". Примером может служить скорость автономного транспортного средства или температура в кузнице. Считается, что с непрерывными пространствами состояний работать немного сложнее, чем с дискретными, т. к. это приводит к бесконечному количеству потенциальных состояний, что делает невозможным использование, например, простой таблицы поиска.

Политики непрерывных состояний должны иметь возможность предсказывать наилучшее следующее действие или значение для данного состояния, а это означает, что политики должны аппроксимировать, а не ожидать известных состояний. Вы можете предоставить аппроксимацию, используя любой стандартный метод машинного обучения, но наиболее распространены функциональные аппроксиматоры. Нейронные сети становятся все более популярными благодаря унифицированному подходу и инструментарию.

Основное беспокойство по поводу непрерывных пространств состояний заключается в том, что они доказуемо сходятся только при использовании линейных аппроксиматоров. По состоянию на 2020 г. не существует доказательств сходимости нелинейных аппроксиматоров (хотя эмпирические данные свидетельствуют о том, что они все-таки сходятся), а это означает, что любой "глубокий" алгоритм не гарантирует сходимости, даже если это хорошая модель. Линейные приближения могут моделировать значительную сложность с помощью преобразований в предметной области. Простота, стабильность и настраиваемая сложность означают, что, как и в машинном обучении, линейные методы очень полезны. В общем, есть три способа дать приблизительное значение.

---

[9] Райдше́ринг — совместное использование частного автомобиля с помощью онлайн-сервисов поиска попутчиков

- *Линейная аппроксимация.*

  Изучение набора линейных весов, непосредственно основанного на входных признаках, возможно, является одним из простейших подходов к работе с непрерывными состояниями. Фактически это то же самое, что и линейная регрессия. Вы также можете улучшить качество регрессии с помощью любого стандартного расширения линейной регрессии, например, такого, которое уменьшает влияние выбросов. Линейная политика доказуемо сходится.

- *Непараметрическая аппроксимация.*

  Вы можете использовать модель, похожую на классификацию, для прогнозирования значений, если это имеет смысл. Например, алгоритмы на основе ближайшего соседа и древовидные алгоритмы способны обеспечивать надежные масштабируемые прогнозы. Опять же, их простота и популярность привели к появлению широкого спектра готовых, надежных промышленных реализаций, которые вы можете использовать, чтобы сделать вашу реализацию более масштабируемой и надежной. Нет никакой гарантии, что непараметрические политики сойдутся.

- *Нелинейная аппроксимация.*

  Нелинейные аппроксимации обеспечивают гибкую степень сложности в едином пакете: нейронных сетях. Вы можете контролировать уровень сложности с помощью архитектуры сети. Нейронные сети дали самые современные результаты во многих областях, поэтому было бы разумно оценить их в вашей задаче. Нет никакой гарантии, что нелинейные политики сойдутся.

Гарантии доказуемой сходимости — сильная сторона линейных приближений. И вы можете получить удивительно сложную модель, преобразовав данные перед переходом к линейному приближению. Далее приведены популярные методы увеличения сложности модели при сохранении простоты и гарантий линейных методов.

- *Полиномиальный базис.*

  Полиномиальное разложение — это генерация всех возможных комбинаций признаков до указанной степени. Полиномы представляют собой кривые в пространстве значений, что дает немного больше сложности по сравнению с линейными методами.

- *Базис Фурье.*

  Разложения Фурье порождают колебания в пространстве ценностей. Эти колебания можно комбинировать произвольными способами для создания сложных форм, подобно тому, как звуки состоят из тональных наложений. Базисы Фурье являются полными в том смысле, что они могут аппроксимировать любую (хорошо управляемую) функцию для заданного уровня сложности, контролируемого порядком базовой функции. Они хорошо работают в самых разных областях [26]. Базис Фурье помогает справляться с неоднородностями в средах, например в таких, как среда Cliffworld.

- *Радиальный базис.*

  Радиальные базисные функции часто представляют собой преобразования гауссовой формы (существуют и другие радиальные базисы), которые измеряют расстояние между состоянием и центром базиса относительно его ширины. Другими словами, это параметризованные гауссовские формы, которые можно перемещать и изменять размер, чтобы приблизиться к ценностной функции. Основная проблема с радиальными базисными функциями (и другими схемами) заключается в том, что вам нужно выбрать ширину основы, которая эффективно фиксирует степень "сглаживания". Оптимальные траектории, как правило, вызывают "всплески" в ценностной функции, поэтому приходится использовать довольно узкую ширину, что может привести к проблематичным локальным оптимумам. Несмотря на это, радиальные базисные функции полезны, потому что гиперпараметр ширины обеспечивает простой и эффективный способ управления "степенью сложности".

- *Другие преобразования.*

  Обработка сигналов предоставляет широкий спектр других преобразований, которые могут применяться для генерации различных характеристик функций, например преобразования Гильберта или Вейвлета. Однако они, как правило, не так популярны, потому что большинство исследователей RL предпочитают математическое или машинное обучение. Но специалисты в области электроники, например, будут более чем счастливы использовать эти "традиционные" методы.

Для получения дополнительных сведений о линейных преобразованиях обратитесь к *разд. "Дополнительные материалы для чтения" в конце данной главы.*

## Преобразование в дискретные состояния

Рассмотрите возможность дискретизации своего домена. Преобразование непрерывных состояний особенно распространено там, где есть естественные сеточные представления домена, например на карте. Основное преимущество дискретизации состоит в том, что она способна устранить необходимость в аппроксиматоре функций из-за фиксированного размера пространства состояний. Это означает, что можно использовать простые, быстрые и надежные алгоритмы, основанные на Q-обучении. Даже если вам все еще нужен аппроксиматор функций, дискретизация снижает видимое разрешение данных, что значительно упрощает исследования агентов и подгонку моделей. Доказано, что это приводит к увеличению производительности для сложных непрерывных задач, таких как в среде Humanoid [27]. В следующем списке представлен обзор возможностей дискретизации (дополнительную информацию можно найти в *разд. "Дополнительные материалы для чтения" в конце данной главы*):

- *Группировка.*

  Зафиксируйте диапазон функции и квантуйте его в интервалы, как гистограмму. Это просто и легко понять, но вы теряете точность. Вы можете разместить границы интервала различными способами, например с постоянной шириной, постоянной частотой или логарифмически.

- *Кодирование плитки.*

  Сопоставьте перекрывающиеся плитки с непрерывным пространством и установите для плитки значение 1, если наблюдение присутствует в диапазоне этой плитки. Это похоже на группирование, но обеспечивает повышенную точность из-за перекрытия плиток. Плитка часто бывает квадратной, но может иметь любую форму и располагаться неравномерно.

- *Хеширование.*

  Когда функции не ограничены, например IP-адреса или адреса электронной почты, вы можете принудительно ограничить функции набором ячеек без потери информации с помощью хеширования (но по-прежнему будет потеря точности из-за объединения в ячейки).

- *Контролируемые методы.*

  В некоторых доменах вы можете пометить данные, которые важны для задачи RL. Вы можете подумать о создании модели классификации и передать результат в алгоритм RL. Для этой цели подходит множество алгоритмов [28].

- *Неконтролируемые методы.*

  Если у вас нет методов, вы можете использовать методы машинного обучения без учителя, такие как метод *k* средних, чтобы расположить ячйки в более удобных местах [29].

## Пространства смешанных состояний

Если у вас пространство смешанных состояний, в вашем распоряжении три варианта действий.

1. Унифицируйте пространство состояний посредством преобразования.
2. Превратите дискретное пространство в непрерывное и продолжайте использовать непрерывные методы.
3. Разделите пространство состояний на отдельные задачи.

Варианты 1 и 2 говорят сами за себя. Большинство функциональных аппроксиматоров успешно работают с дискретными значениями, даже если они не могут быть оптимальными или эффективными.

А вот вариант 3 интересен. Если вы можете разделить состояние на разные компоненты, которые соответствуют ограниченным контекстам в проблемной области, это сигнал того, что эффективнее использовать подмодели или подполитики. Например, грубый подход может заключаться в том, чтобы разделить непрерывное и дискретное состояния, передать их в два отдельных алгоритма RL, а затем объединить их в ансамбль [30]. Как видите, это касается иерархического RL, поэтому обязательно используйте идеи из него.

## Сопоставление политик с пространствами действий

Прежде чем выяснять, как политика определяет действия, сделайте шаг назад и подумайте, какой тип действий будет наиболее подходящим для вашей задачи. Задайте себе следующие вопросы.

- *Какие типы ценностей поставлены в соответствие вашим действиям?* Они бинарные? Они дискретны? Они ограничены или неограничены? Они непрерывны? Они смешанные?
- *Возможны ли одновременные действия?* Можно ли выбрать несколько действий одновременно? Можете ли вы выполнять итерацию быстрее и чередовать отдельные действия, чтобы приблизиться к выбору нескольких действий?
- *Когда нужно действовать?* Есть ли временной элемент в выборе действия?
- *Есть ли какие-нибудь метадействия?* Можете ли вы придумать какие-либо высокоуровневые действия, которые могут помочь изучить или обработать временны́е элементы?
- *Можете ли вы разделить действия на подзадачи?* Есть ли в ваших действиях очевидный ограниченный контекст? Можно ли их разделить на отдельные задачи? Это иерархическая задача?
- *Можете ли вы по-другому взглянуть на свои действия?* Например, вместо перехода к позиции с непрерывной оценкой, можете ли вы подумать о "продвижении вперед"? Или наоборот?

Все эти решения серьезно влияют на вашу работу, поэтому будьте осторожны при их принятии. Например, переключение действия с непрерывного на дискретное может вынудить вас изменить алгоритмы. Или понимание того, что вашему агенту требуются одновременные действия, может значительно усложнить решение вашей проблемы.

### Бинарные действия

С бинарными действиями, возможно, проще всего работать, потому что их легко оптимизировать, визуализировать, отлаживать и внедрять. Ожидается, что политики выведут отображение бинарного значения для действия, которое они хотят выбрать. В задачах, где разрешено только одно действие, выбор максимального ожидаемого дохода (для ценностных методов) или действия с наивысшей вероятностью (для методов градиента политики) является простой реализацией.

Как показано в *разд. "Как температурный параметр влияет на исследование?" главы 7*, вы можете стимулировать широкое исследование, задав вероятности результатов политики или стоимостные оценки, которые можно интерпретировать как вероятности. Затем агент может произвольно выбирать из этого пространства действий в соответствии с назначенными вероятностями. Для ценностных методов вы можете назначить наивысшие вероятности наивысшим ожидаемым значениям. Я бы рекомендовал использовать преимущества, а не ценности напрямую, чтобы избежать предвзятости. Для преобразования стоимостных оценок в вероятности

рекомендуется использовать экспоненциальную функцию softmax. В методах градиента политики вы можете обучить модели напрямую предсказывать вероятности.

## Непрерывные действия

Непрерывные действия очень распространены, например определение положения сервопривода или указание суммы ставки для объявления. В этих случаях наиболее естественно использовать какую-либо функцию аппроксимации с целью вывода непрерывного прогноза для данного состояния (аналогично регрессии); это делают детерминированные градиенты политики. Но исследователи обнаружили, что моделирование выходных данных как случайной величины может помочь разведке. Таким образом, вместо того чтобы напрямую предсказывать выходное значение, предсказывают средние значения распределения, представленные как гауссиан. Затем значение выбирается из распределения. Другие параметры распределения, такие как стандартное отклонение для гауссиана, также могут быть предсказаны. Далее вы можете установить большое начальное значение стандартного отклонения, чтобы стимулировать исследование, и позволить алгоритму уменьшать его по мере обучения. Мне нравится думать об этом как о грубой попытке включить вероятностное моделирование. Конечно, предполагается, что ваши действия удовлетворяют некоторому закону распределения, и это соответствие может быть неточным, поэтому хорошо подумайте, какое распределение выбрать для своих действий.

Альтернативой является использование более новых методов, основанных на Q-обучении, но адаптированных для аппроксимации непрерывных функций — Q-обучение непрерывного действия [31]. Распространенность таких методов куда меньше, нежели методов градиента политики, но считается, что они более эффективны.

Еще один, последний прием, который я хочу упомянуть и который полезен в простых примерах, называется *приемом масштабирования размеров*. При использовании линейных аппроксиматоров, которым необходимо прогнозировать несколько выходных данных, таких как среднее значение и стандартное отклонение, вы можете иметь отдельные параметры для каждого действия и обновлять параметры для выбранного действия индивидуально. По сути, это словарь параметров, где ключевым является действие, а значения, такие как среднее значение и стандартное отклонение, служат параметрами для непрерывного отображения.

## Гибридные пространства действий

Пространства с дискретными и непрерывными действиями относительно распространены, например решение о покупке и денежной сумме, которую можно потратить, и с ними бывает сложно справиться. Сначала подумайте, сможете ли вы разделить задачу. Если это иерархическая задача, сначала именно попытайтесь ее разделить. Затем подумайте, можете ли вы обойтись только непрерывным или только дискретным представлениями данных. Вы, вероятно, немного потеряете в детализации, но это может упростить запуск.

Появляются новые алгоритмы, которые учатся выбирать либо непрерывный алгоритм, либо дискретный в зависимости от требуемого пространства действий. Пара-

метризованная глубокая Q-сеть является одним из таких примеров, но эту идею можно применить к любому алгоритму [32]. Другой подход, называемый *ветвлением действий*, просто создает больше "голов" с различными активациями и целями обучения в нейронной сети, что кажется хорошим инженерным решением проблемы. Эта многоголовая архитектура также позволяет выполнять одновременные действия. В некотором смысле это просто рукотворная форма иерархического RL.

## Когда выполнять действия

Вопрос: что делать, если вы не хотите выводить никаких действий? На самом деле это намного сложнее, чем кажется, потому что все, от уравнения Беллмана до обучения нейронной сети, настроено на ожидание действия на каждом этапе. В играх Atari реализован пропуск кадров без особого учета того, сколько кадров нужно пропустить; 4 кадра просто принимаются как данность. Но исследователи обнаружили, что можно оптимизировать количество пропущенных кадров для повышения производительности [34].

Более общий подход заключается в использовании фреймворка *опций*, который включает условие прекращения. Эти методы учатся не только тому, *как* действовать оптимально, но и когда [35, 36]. Потенциально более простым решением является добавление действия, которое заставляет агента повторять предыдущие действия; я видел, как это используется в робототехнике для уменьшения износа двигателей. Также можно научиться пропускать как можно больше действий, чтобы снизить затраты, связанные с действием [37].

Как обычно, лучшее решение зависит от конкретной задачи. В общем, я бы рекомендовал как можно больше оставаться в рамках "нормального" RL, чтобы упростить разработку.

## Обширные пространства действий

Когда пространства состояний становятся слишком большими, вы можете использовать проектирование состояний, чтобы уменьшить их до разумного количества. Но что делать, если у вас есть обширное пространство действий, скажем, если вы рекомендуете покупателю один из миллионов продуктов? Например, YouTube насчитывает $10^8$ элементов [38].

Методы, основанные на ценностях, такие как Q-обучение, сталкиваются с большими проблемами, потому что вам нужно максимизировать набор возможных действий на каждом этапе. Очевидная проблема заключается в том, что когда у вас есть непрерывные действия, количество действий, которые нужно максимизировать, бесконечно. Даже в дискретных настройках может быть так много действий, что выполнение этой максимизации может занять невероятно много времени. Методы градиента политики решают эту проблему, поскольку они выбирают из распределения, и максимизации нет.

Решения обозначенной проблемы включают использование вложений для сокращения общего количества действий до пространства действий [39]. Это инстинктивно привлекает благодаря успеху встраивания машинного обучения в другие за-

дачи, например при обработке естественного языка. Если у вас есть области действий, похожие на задачи, при которых встраивание было успешным, то это может оказаться хорошим подходом. Другая стратегия состоит в том, чтобы рассматривать максимизацию как проблему оптимизации и научиться предсказывать максимальные действия [40].

Связанная проблема заключается в том, что вам нужно выбрать фиксированное количество элементов в области действий. Это влечет комбинаторный взрыв и является достаточно распространенной проблемой, отнимая время исследователей. Существует широкий спектр приближенных решений, таких как иерархические модели и автоэнкодеры [41, 42]. Более общее решение состоит в том, чтобы переместить сложность пространства действий в пространство состояний путем вставки фиктивных состояний и использования групповых действий, прогнозируемых на нескольких этапах для создания выборки рекомендаций [43]. Более простое и надежное решение заключается в предположении, что пользователи могут выбрать одно действие. Затем вам нужно только ранжировать и выбрать топ $k$ действий, а не пытаться решить, какая комбинация действий лучше. Было показано, что это решение увеличивает привлечение пользователей на YouTube [44].

Рассмотрим этот результат в контексте того, что Google (владеющий YouTube) имеет доступ к одному из самых передовых пулов специалистов по данным в мире. Этот алгоритм составляет конкуренцию хорошо настроенной модели рекомендаций, на совершенствование которой командам опытных специалистов в области науки о данных (вероятно) потребовалось много лет. Невероятно!

## Исследование

Дети обеспечивают личное и профессиональное вдохновение целым рядом неожиданных способов, например, когда я работаю над презентацией или новой абстракцией, я обычно пытаюсь объяснить это своим детям. Если я не могу объяснить так, чтобы это звучало хотя бы связно для них, то это сигнал, что мне нужно больше работать над улучшением своего сообщения. Книга "Иллюстрированное руководство по Kubernetes для детей" ("The Illustrated Children's Guide to Kubernetes")[10] — прекрасное свидетельство этой идеи.

Дети также направляют нас для улучшения познания. Это необходимо, потому что я львиную долю этой книги объяснял, что большинство задач — это проблема исследования: сложные состояния, большие пространства действий, сходимость, оптимумы и т. д. Все эти проблемы можно решить более тщательной разведкой. Недавние исследования показали, что у детей в очень раннем возрасте есть врожденная способность активно исследовать окружающую среду ради получения знаний. Они могут проводить простые эксперименты и проверять гипотезы. Они могут объединить несколько источников информации, чтобы создать богатую ментальную модель мира и спроецировать эти знания на новые ситуации [45].

[10] См. **https://www.cncf.io/phippy/the-childrens-illustrated-guide-to-kubernetes/**.

Алгоритмы RL основаны на ошибках, и обучение происходит почти случайным образом. Без цели агенты будут натыкаться на среду, как раздражающая муха, которая, по-видимому, неспособна вылететь в открытое окно, через которое она прилетела.

Похоже, что дети склонны работать с алгоритмом, приближенным к *поиску в глубину*. Им нравится доводить свои действия до конца, пока не возникнет воспринимаемое состояние, которое блокирует текущий путь, как тупик в лабиринте. Невероятно, но это работает без какой-либо цели, а значит, что дети передают знания о состояниях из предыдущего опыта. Когда тем же детям затем ставят цель, они могут использовать ментальную карту, созданную своим исследованием, чтобы значительно улучшить первоначальный поиск цели. Сравните это с существующими алгоритмами RL, которые обычно полагаются на случайное нахождение цели.

С этой целью проводится большая работа, посвященная изучению того, как улучшить исследование, особенно в задачах, которые имеют скудную награду. Но сначала подумайте, можете ли вы предоставить учебную программу в виде улучшенных алгоритмов (например, заменить ε-жадные методы чем-то еще) или промежуточных вознаграждений (например, меры расстояния до цели) или использовать что-то вроде имитационного RL для активного руководства вашим агентом. Также подумайте об изменении определения задачи, чтобы получать более частые награды. Все это улучшит скорость обучения и/или исследования без привлечения следующих методов.

## Является ли внутренняя мотивация исследованием?

Исследователи пытаются решить проблему разведывания данных, разрабатывая альтернативы случайному движению, что приводит к избытку предложений с антропоморфными названиями, такими как удивление, любопытство или расширение прав и возможностей. Несмотря на хитрые названия, методы исследования с привлечением людей слишком сложны для кодирования в простом алгоритме RL, поэтому исследователи пытаются создать подобное поведение с помощью математических изменений функции оптимизации.

Следует подчеркнуть проблему всех этих методов — отсутствие правильного ответа. Я уверен, что вы можете представить себе несколько математических решений для поощрения исследований, таких как бонус энтропии в мягком алгоритме "актор — критик", но они никогда не приблизятся к истинному исследованию, подобному человеческому, без предварительного опыта. Именно поэтому я считаю, что эта область весьма молода и такие слова, как "любопытство" или "внутренняя мотивация", являются неудачными названиями, поскольку они подразумевают, что исследование — это попытка новых действий, а это не так. Исследование — это RL. Вы используете предыдущие знания, чтобы исследовать новые состояния, основываясь на ожидании будущих наград.

Представьте себе классическую RL-игру, в которой ваш агент смотрит на стену. Если он что-то знает о стенах, он никогда не должен исследовать состояния, близкие к стене, потому что от этого нет никакой пользы. Все алгоритмы RL игнориру-

ют простое физическое правило: нельзя проходить сквозь стены, поэтому они тратят время, буквально ударяясь головой о стену. Но также интересно отметить, что эти предположения могут помешать нам обнаружить неожиданные состояния, такие как раздражающие невидимые блоки в серии Super Mario.

Так что, если вы хотите провести оптимальное исследование, вам понадобится предварительный опыт. Я думаю, что передача знаний между задачами является ключом к этой проблеме, и RL без учителя имеет потенциальное решение. А контрфактические рассуждения приобретут бóльшую важность по мере того, как алгоритмы начнут использовать "библиотеки" предшествующих знаний.

Внутренняя мотивация и методы, обсуждаемые далее, на самом деле вовсе не связаны с исследованием. Они пытаются решить задачу *холодного запуска*. Когда у вас нулевые знания и скудные награды, как вы можете поощрять глубокое исследование?

## Количество посещений (выборка)

Один из простейших способов повысить эффективность исследования — убедиться, что агент не посещает состояние слишком много раз. Или, наоборот, поощряйте агента чаще переходить в редко посещаемые состояния. Методы с верхним доверительным интервалом пытаются предотвратить слишком частое посещение состояний агентом (*см. разд. "Улучшение ε-жадного алгоритма" главы 2*). Выборка Томпсона — это оптимальный метод выборки для такого рода задач, а вероятностная случайная выборка, основанная на функции ценности или предсказаниях политики, может служить полезным заместителем.

## Прирост информации (сюрприз)

*Прирост информации* используется в машинном обучении как мера снижения энтропии. Например, древовидные алгоритмы используют получение информации для поиска разделений в пространстве признаков, которые уменьшают количество энтропии между классами; хорошее разделение делает классы ясными или чистыми, поэтому прирост информации велик [45].

Многие исследователи пытались использовать полученную информацию, чтобы направить агента в районы, вызывающие большее "удивление". Основное преимущество перед методами необработанной выборки состоит в том, что они могут включать внутренние представления политики, а не полагаться на внешние наблюдения за перемещением агента.

Одним из примеров этого является *исследование с максимизацией вариационной информации* (variational information maximizing exploration, VIME), которое использует дивергенцию (*см. разд. "Дивергенция Кульбака — Лейблера" главы 6*) между наблюдаемыми траекториями и траекториями, ожидаемыми параметризованной моделью, в качестве дополнения к функции вознаграждения [47]. Обратите внимание на сходство между этой идеей и тем, что обсуждается в *разд. "Методы доверительной области" главы 6*. В результате мы имеем исследование, которое, кажется, "проносится" через пространство состояний.

## Прогноз состояния (любопытство или саморефлексия)

*Любопытство* — это человеческая черта, поощряющая поиск и нахождение информации, которая удивляет. Предыдущий подход был направлен на поощрение неожиданных траекторий, но вы, возможно, не захотите его применять, потому что интересные пути, скорее всего, приведут к результату, который совсем не удивителен. Подумайте, важно ли это для вашей задачи. Например, вы можете искать новые способы достижения того же результата, в таком случае это будет очень полезно. Однако во многих задачах важен результат, а не то, как вы его достигли.

Поэтому вместо того, чтобы поощрять разные пути, например разные маршруты в школу, другое решение — поощрять новые ситуации, например переход в другую школу. Одним из примеров этого является создание модели, которая пытается предсказать следующее состояние на основе текущего состояния и действия, а затем вознаграждает в зависимости от того, насколько неверен прогноз [48]. Проблема с этим методом заключается в том, что прогнозирование будущих состояний, как известно, осложняется стохастическим и динамическим характером среды. Если бы всё было так просто, вы могли бы использовать машинное обучение для решения своей задачи. Поэтому предпринимается попытка смоделировать лишь те части состояния, которые влияют на агента. Это делается путем обучения нейронной сети, которая предсказывает действие из двух последовательных состояний, что создает внутреннее представление политики, а затем награждает на основе разницы между моделью, которая предсказывает, каким *должно быть* внутреннее представление, и тем, чем оно на самом деле является после учета данного состояния. Более простой, похожий подход заключался бы в использовании автоэнкодера с задержкой, когда автоэнкодер сравнивает скрытое представление с наблюдаемым.

## Любопытные задачки

В реальном мире все время встречаются новые состояния. Как и предполагает предыдущий метод, мы строим долгосрочные модели, чтобы предсказать нечто новое. Это означает, что мы также научились игнорировать шум. Представьте процесс управления автомобилем. Вы каким-то образом научились автоматически игнорировать 99% нерелевантной информации вокруг себя и концентрироваться на опасностях. Но поставьте в поле зрения экран телевизора или мобильный телефон, и ваши глаза будут прикованы к нему. Ожидание увидеть что-то новое, пренебрегая обнаружением опасностей, — вот что делает вождение с телефоном или планшетом таким опасным.

Исследователи обнаружили, что аналогичная проблема существует и в RL. На рис. 9.7 показан забавный сценарий, в котором телевизор был помещен на стену [49]. Агент, движимый любопытством, постоянно вознаграждается, потому что он не может предсказать, что будет показано по телевизору в следующий раз. Очевидно, этот результат немного надуман; весь смысл этих моделей состоит в том, чтобы отфильтровать шум, что можно довольно легко сделать, если вы разбираетесь в предметной области. Но в целом исследователям нужно больше сосредоточиться на улучшении глобальных исследований, а не на локальных.

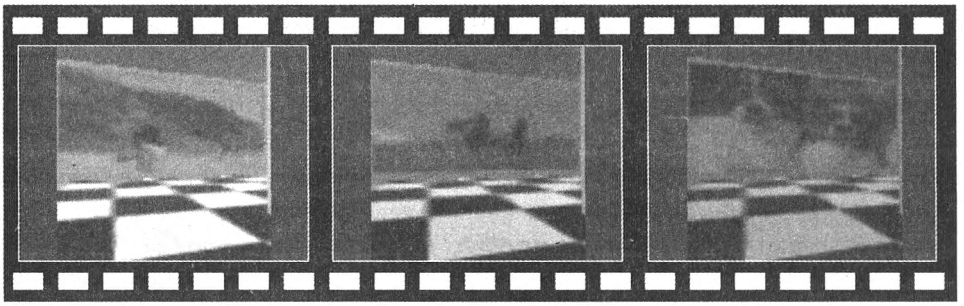


**Рис. 9.7.** Использование наблюдений в качестве источника неожиданности может побудить агентов застрять в шумных состояниях.
Адаптировано из видео ©Deepak Pathak, использовано по лицензии CC BY 2.0

## Случайные вложения (сети случайной дистилляции)

Одна из идей преодолеть проблему "телепомех", когда агенты зацикливаются на стохастических состояниях, заключается в использовании случайных вложений. *Сети случайной дистилляции* (random distillation networks) — это пара нейронных сетей, одна из которых инициализируется случайным образом и больше никогда не затрагивается. Случайная сеть используется для генерации случайных вложений. Вторая сеть обучена предсказывать случайные вложения. В работе Бурды и соавт. ошибка между случайным и предсказанным вложениями используется как мера "новизны"; большая ошибка означает, что этот тип наблюдения недостаточно хорошо представлен в буфере и, следовательно, его нужно посетить снова в будущем. Это значение ошибки добавляют к награде, чтобы способствовать посещению новых состояний.

Основное преимущество этого подхода состоит в том, что он не предсказывает следующее наблюдение; он просто сравнивает следующее наблюдение с текущей моделью среды. Это означает, что стационарный вид стохастических состояний, например отдельное изображение на экране телевизора, можно смоделировать достаточно хорошо, и поэтому агент не получает вознаграждения за то, что он "бездельник".

Я нахожу идею использования случайно инициализированной нейронной сети в качестве вложения трудной для понимания. Но представьте, что ваше зрение подверглось какой-то случайной трансформации. Со временем ваш мозг научится компенсировать это и разовьет новые абстракции; восстановление после инсульта должно пройти через это испытание. Еще одно преимущество состоит в том, что эта архитектура проще в том смысле, что требуется обучать только одну сеть. Но исследование по-прежнему носит чисто локальный характер; оно не способствует глубокому исследованию. Другая проблема заключается в том, что очень важно иметь архитектуру, которая обеспечивает эффективное встраивание в предметную область. Сверточные нейронные сети явно хорошо работают в областях, основанных на изображениях, но как насчет других областей? Эти случаи требуют вашего опыта в предметной области и опыта в моделировании [51].

## Расстояние до новизны (эпизодическое любопытство)

Одно интересное направление исследований, способствующих глобальному изучению проблемы, — использовать расстояние между состояниями как лучшую меру новизны. Савинов и соавт. предлагают использовать буфер опыта для записи новых состояний, которые кодируются с помощью встраивания. Буфер очищается в начале каждого эпизода. Когда агент попадает в новое состояние, оценивается расстояние между состояниями в буфере и текущим состоянием. Если новое состояние находится в пределах нескольких шагов среды от состояния в буфере, оно не считается новым и наказывается. Если новое состояние находится дальше определенного количества шагов, агент получает вознаграждение за обнаружение новых состояний. Это побуждает агентов искать новые состояния на эпизодической основе [52].

На рис. 9.8 представлена траектория, пройденная во время одного эпизода. Она показывает знак награды и время помещения встроенных наблюдений в буфер. Вы можете видеть, что алгоритм заставляет агента продолжать движение, что способствует исследованию в глубину и предотвращает превращение агента в "бездельника".

Реализация довольно сложная. Во-первых, изображения преобразуются в скрытые вложения с помощью сверточной нейронной сети, использующей половину *сиамской* сети, которая представляет собой архитектуру нейронной сети, используемую в задачах сопоставления изображений [53]. Мне не нравится это название, но именно так ее называют в литературе. Затем текущее внедрение сравнивается со всеми вложениями в буфере, создавая меру расстояния до всех других наблюдений. Наконец, выполняется агрегирование, чтобы найти расстояние до следующего ближайшего состояния, которое представляет новизну этого наблюдения и используется для расчета вознаграждения.

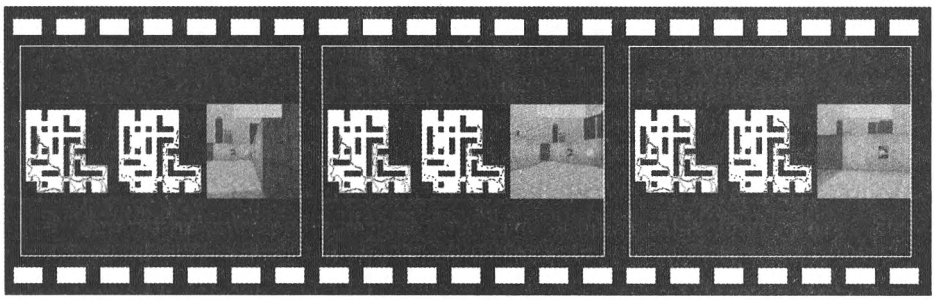


**Рис. 9.8.** Траектории из эпизода среды DMLab с эпизодическим любопытством. В левом кадре показано, как агент был вознагражден за это состояние (зеленый означает положительное вознаграждение, красный — отрицательное). Средний кадр показывает, когда "воспоминания" были добавлены в эпизодический буфер. Правый кадр — это вид от первого лица агента. Видео адаптировано из работы Савинова и соавт.

## Выводы по разведке

Мы знаем, что люди при обучении применяют поиск в глубину, используя полученные ранее знания. Перенести эту идею в RL невероятно сложно, потому что нет

очевидного способа прямого исследования без нарушения предположений MDP. Кажется, что есть пробел, который не может заполнить MDP, когда часть задачи находится под контролем, например подготовка к экзамену, а другая — полностью оппортунистическая, скажем, когда вы погружаетесь в другую культуру.

Это важная проблема, которую необходимо решить, т. к. разреженные состояния часто возникают в RL из-за природы целенаправленной инженерии. С редкими наградами трудно работать, потому что их трудно найти. Одним из решений является создание альтернативы (псевдопредположения), которая поощряет произвольную стратегию исследования, но во многих задачах относительно просто вручную спроектировать вознаграждения, соответствующие вашей проблеме (*см. разд. "Разработка вознаграждений" далее в этой главе*), или изменить определение задачи, чтобы упростить ее.

Текущие попытки автоматизировать разведку принимают локальный, эпизодический характер и не учитывают глобальные или долгосрочные цели разведки. Фактически исследователи заметили, что продвижение новизны активно поощряет опасные состояния, такие как прогулка по краю обрыва, потому что к ним трудно приблизится и они редко отражаются в прошлом опыте агента [54].

Все усилия поощряют исследование, увеличивая функцию вознаграждения, а не направляя выбор действия. Есть четкие параллели между изучением учебной программы и обратным RL, которые могут улучшить эту ситуацию. Безусловно, в перспективе разведка данных будет направлена либо на изменение фундаментального определения MDP, допускающее некоторую координацию на высоком уровне (таким образом, нарушая марковское предположение), либо на включение работы других областей в RL для поощрения долгосрочных исследований.

## Разработка вознаграждений

Разработка вознаграждений для решения вашей задачи, возможно, самое важное в RL. Именно здесь ваша задача как инженера — преобразовать определение проблемы в показатель, который направляет агента к решению. Это невероятно важно. Ваше определение диктует решение, и легко случайно заставить агента решить не ту задачу. Отличный наглядный пример показан на рис. 9.9. Агент во фреймворке Unity поощряется к достижению цели, и он может использовать коробку, чтобы перепрыгнуть через стену. Вместо этого из-за небольшого края среды агент научился красться вдоль ее сторон. Несмотря на разумное вознаграждение, агент научился эксплуатировать среду способами, которые вы не можете себе представить.

В машинном обучении вы напрямую определяете алгоритм решения задачи. Итак, если у вас есть разумное представление о задаче, которая сама по себе является проблемой, легко заметить, когда проекты идут наперекосяк. Но в RL вам нужно дождаться завершения обучения, чтобы увидеть, как агент решает эту проблему. Это резко увеличивает длину цикла обратной связи и демонстрирует, почему так важно понимание результирующей политики.

Что еще хуже, награда также определяет максимальную производительность алгоритма и скорость его достижения. Награды должны помогать агенту руководить,

и чем больше рекомендаций вы дадите, тем быстрее он научится. Однако многие проблемы редки в том смысле, что награда будет только тогда, когда задача выполнена. Исследователи предложили множество способов смягчения разреженности, но самое простое, что вы можете сделать, — это разработать вознаграждение, которое будет соответствовать задаче, которую вы хотите выполнить. Но помните, что нет оптимальной политики без вознаграждения, и ваше вознаграждение меняет оптимальную политику.

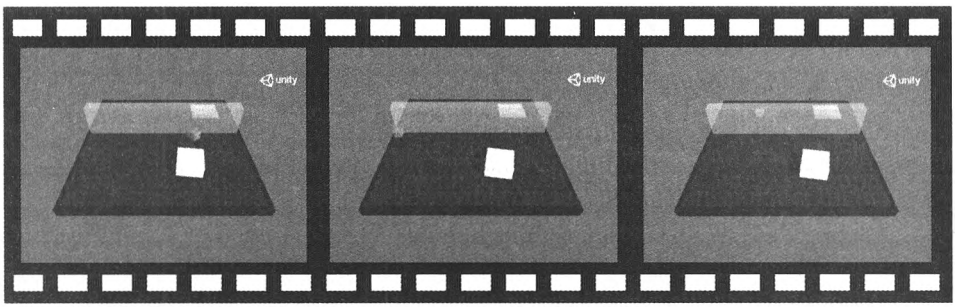


**Рис. 9.9.** В этом примере из Unity цель состояла в том, чтобы научить агента использовать коробку для перепрыгивания через стену, побуждая агента приблизиться к цели. Вместо этого он научился хитрить и красться по краю. Адаптировано из презентации Дэнни Ланге, ©Unity Technologies, с разрешения

## Рекомендации по разработке вознаграждений

У каждого домена есть свои специфические проблемы, что затрудняет создание жестких и быстрых правил определения вознаграждения. Однако в этом разделе я хочу обозначить несколько общих стратегий, применимых к любой ситуации. Вознаграждение должно делать следующее.

- *Быть быстрым.*

  Функции вознаграждения должны быстро вычисляться, потому что в сложных средах они вызываются миллионы раз.

- *Соотноситься с "истинной" наградой.*

  Вознаграждения по доверенности должны соотноситься с истинными вознаграждениями, в противном случае вы рискуете решить не ту задачу. Это означает, что они должны приравнивать или приближать терминологию, используемую в проблемной области; например, денежное выражение, количество клиентов, спасенные жизни.

- *Снижать уровень шума.*

  Вознаграждения по доверенности могут быть шумными, а это означает, что они не совсем соответствуют истинному вознаграждению. Рассмотрите возможность комбинирования различных прокси-вознаграждений, чтобы уменьшить дисперсию вознаграждения.

- *Кодировать нефункциональные требования.*

  Рассмотрите возможность добавления прокси-вознаграждений, которые соответствуют нефункциональным требованиям, таким как безопасность, эффективность или скорость.

- *Учитывать время.*

  Фактор дисконтирования заставляет агента становиться более или менее *близоруким*; он контролирует, насколько большое внимание уделяется оптимизации краткосрочного или долгосрочного вознаграждения. Учтите это при формировании награды.

- *Предотвращать выход на плато.*

  Ваша награда должна четко различать состояния, а разница должна быть больше, чем шум Q-значений. Если у состояний очень похожие награды, агенты обычно ходят по кругу.

- *Добиваться плавности.*

  Для того чтобы добиться "плавного" поведения, политикам необходимо учиться на плавном вознаграждении. Разрывы в пространстве вознаграждений могут проявляться в виде "резких" "толчковых" действий в политике.

- *Быть простым.*

  Как и все остальное в RL, сложность ухудшает понимание. Если у вас простое вознаграждение, правила легко понять. Если всё сложно, вы увидите неожиданные политики.

## Формирование вознаграждения

Это может быть очевидным, но о нем все же стоит сказать. На политику влияет величина вознаграждения, т. к. агенты стремятся максимизировать общее вознаграждение. Поскольку вы разрабатываете вознаграждение, вам также необходимо указать его размер при достижении цели. Когда есть подцели, несколько целей или ситуаций, которых следует избегать, балансирование вознаграждений каждого типа может быть мучительным. В таких обстоятельствах вы можете попытаться разделить задачу или взглянуть на иерархическое RL.

В общем, вы можете произвольно изменять *форму* вознаграждения, например для поощрения определенного поведения или повышения скорости обучения. При разработке вознаграждения следует придерживатся определенных рекомендаций.

- *Нормализовать вознаграждение.*

  Будьте осторожны с размером награды. В общем, все алгоритмы выигрывают от нормализованного вознаграждения, потому что они уменьшают дисперсию градиентов при оптимизации политики или функции ценности.

- *Усекать вознаграждения.*

  Усечение вознаграждений — распространенная форма нормализации, но усечение может отбросить информацию. Например, DQN рекомендует обрезать на-

грады до ±1, но в игре *Pac-Man* это относится как к точкам, так и к фруктам, имеющим одинаковую награду, что неверно; фрукт дает гораздо более высокий балл (эта прокси-награда не коррелирует с истинной наградой). Если вы нормализуете награды в *Pac-Man*, агент DQN станет охотником-собирателем, научится активно охотиться на призраков и искать фрукты, а не просто собирать точки.

- *Преобразовывать награды.*

  Изменение градиента вознаграждения способно помочь сгладить политику и обеспечить бóльшую поддержку, когда агент находится далеко от цели. Например, линейное вознаграждение обеспечивает одинаковую срочность независимо от того, где в настоящее время находится агент в пространстве состояний. Вы можете передать свое вознаграждение через функцию, в которой вознаграждение быстро меняется, когда агент далек от цели. Это скользкий путь, потому что изменения в вознаграждении могут иметь неожиданные проявления в политике.

- *Награды модели.*

  Во многих доменах есть среды, в которых возможно некоторое моделирование, по крайней мере, локально, поэтому можно получить прокси-вознаграждение, которое эффективно кодирует знания предметной области. Например, в робототехническую задачу вы можете включить некоторые предварительные знания, такие как тот факт, что робот должен встать, прежде чем он сможет бежать; вы можете включить датчик, который измеряет, насколько далеко от земли находится робот. В общем, включение такой достоверной информации повышает производительность.

## Общие награды

Проверьте, есть ли какие-либо опубликованные решения задач, похожих на вашу. Общие домены, как правило, придерживаются проверенных определений вознаграждения. Далее приводится список распространенных типов вознаграждений.

- *Редкие.*

  Многие задачи считаются завершенными только при достижении конкретного состояния, иногда нескольких состояний в определенном порядке. Обычно за выполнение задачи можно получить большу́ю награду, но ее трудно решить, потому что награды не направляют агента к цели.

- *Награда типа "расстояние — цель".*

  Показатели расстояния до цели являются очень сильными показателями вознаграждения, потому что они не только сильно коррелируют с истинным вознаграждением, но также направляют агента к цели независимо от того, где тот находится в пространстве состояний. Это улучшает исследование и ускоряет время обучения.

- *Наказывающие шаги.*

  Наказание агента за каждое действие побуждает его делать как можно меньше шагов.

- *Наказание за вредное или опасное поведение.*

  Часто используется сильное отрицательное вознаграждение за деструктивное поведение, но будьте осторожны, чтобы это не повлияло отрицательно на оптимальные пути вознаграждения, например отклонение на милю от обрыва, когда десяти метров было бы вполне достаточно.

- *Награды на основе изображений.*

  Награды обычно основаны на низкоразмерном состоянии. Но все чаще исследователи используют многомерные наблюдения, такие как изображения, для определения состояния цели. Например, Чжу и соавт. применили набор изображений для определения состояния цели и использовали как случайную сетевую дистилляцию, так и скрытую модель, созданную автоэнкодером. Это невероятно эффективно во многих реальных приложениях, потому что вам нужно всего лишь собрать пакет изображений, представляющих состояние вашей цели, а затем предоставить RL сделать все остальное [56].

### Выводы о вознаграждении

Определение вознаграждения, которое решает задачу, способствует быстрому обучению и приводит к оптимальной политике, — чрезвычайно сложная задача. Я бы предположил, что во многих приложениях список проблем также построен в порядке приоритета, и до поры, пока определение задачи является правильным, инженеры, как правило, не беспокоятся об оптимальности или быстром обучении, потому что можно использовать другие методы для решения этих проблем. Но если вам удастся найти баланс, результатом будут более простые и эффективные алгоритмы.

Опять же, вы могли заметить, что некоторые рекомендации связаны с другими темами. Действительно, автоматическое генерирование вознаграждения — это главная цель внутренней мотивации, неконтролируемого RL и даже метаобучения. Вы должны иметь целостное представление об этих методах и использовать их в нужное время для решения нужных задач. В противном случае вы можете чрезмерно усложнить задачу, что часто приводит к потере времени или даже к полному сбою.

Проведено большое количество исследований, в которых предпринимаются попытки объединить некоторые из этих идей в общий алгоритм. Например, согласно одной интересной идее, формирование вознаграждения может привести к локальным оптимумам, от которых трудно отказаться. Тротт и соавт. используют два независимых контрфактических развертывания, чтобы проверить, есть ли еще один оптимум, больший, чем текущий [57].

## Резюме

Несмотря на то что RL существует уже несколько десятков лет, оно только недавно получило известность как жизнеспособный в промышленности и применимый на практике инструмент. Я считаю, что причина задержки не техническая. Полагаю, что главная проблема в том, что абстрагироваться от технических деталей так, что-

бы сделать их понятными, сложно. Также сложно определить, что такое "стратегия" или как она может помочь бизнесу, не говоря уже о том, что "роботы будут править миром". Идея мощная, но вы, как инженер, обязаны реализовать потенциал (и предостережения) таким образом, чтобы это находило отклик у заинтересованных сторон.

В этой главе я сосредоточился не столько на лакомых деталях, сколько на самом процессе. Однако помните, что это новая область, с новыми возможностями. Я представил то, что считаю разумным, но у вас могут быть идеи получше. Не бойтесь адаптировать свой подход к вашей задаче. Это также означает, что передовые методы со временем будут применяться везде. Если вы действительно используете RL в промышленном проекте, я хотел бы услышать о вашем опыте!

Надеюсь, теперь у вас достаточно знаний, чтобы, по крайней мере, начать проект RL. Если вы этого не сделаете, свяжитесь со мной (контактные данные *см. в разд. "Об авторе"*), и я постараюсь помочь. Теперь вы знаете, как выглядит процесс и что нужно искать в проекте RL. Вы также знаете, сколько инженерных усилий необходимо для того, чтобы сделать проект жизнеспособным, а затем и оптимальным. Кроме того, в Интернете можно найти множество исследований, которые могут напрямую соответствовать вашей задаче, и я также включил некоторые важные материалы в *разд. "Дополнительные материалы для чтения" далее в этой главе*, которые я нашел полезными во время своего исследования.

# Дополнительные материалы для чтения

- Автономное обучение с подкреплением.
  - Слегка старая, но все еще актуальная часть [58].
  - Недавний обзор проблем [59].
- Обучение без сброса.
  - Хороший обзор непрерывного обучения с упором на робототехнику [60].
- Государственный инжиниринг.
  - Хороший обзор состояний [61].
- Разработка политики.
  - В книге Саттона и Барто есть краткий, но содержательный раздел о линейной аппроксимации и преобразовании. Также подходит для получения подробной информации о параметризации непрерывного действия [62].
  - Старый, но актуальный обзор методов дискретизации в ML [63].
  - Видео о гауссовой политике[11].
- Исследование.
  - Интересное сравнение психологической внутренней мотивации с вычислительными подходами [64].
  - Полное руководство по отбору выборок Томпсона [65].

[11] См. **https://oreil.ly/VWfLm**.

# Использованные источники

[1] Zheng S., Trott A., Srinivasa S. et al. The AI economist: improving equality and productivity with AI-driven tax policies // ArXiv:2004.13332. — 2020. — April. — URL: **https://oreil.ly/hXqcT**.

[2] Håkansson S., Lindblom V., Gottesman O., Johansson F. D. Learning to search efficiently for causally near-optimal treatments // ArXiv:2007.00973. — 2020. — July. — URL: **https://oreil.ly/hX-GL**.

[3] Lange S., Gabel T., Riedmiller M. Batch reinforcement learning // Reinforcement learning: state-of-the-art / ed. by M. Wiering, M. van Otterlo. — Adaptation, learning, and optimization. — Berlin, Heidelberg: Springer, 2012. — P. 45–73. — URL: **https://oreil.ly/yAHBp**.

[4] Levine S., Kumar A., Tucker G., Fu J. Offline reinforcement learning: tutorial, review, and perspectives on open problems // ArXiv: 2005.01643. — 2020. — May. — URL: **https://oreil.ly/R_vjH**.

[5] Chen X., Zhou Z., Wang Z. et al. BAIL: Best-action imitation learning for batch deep reinforcement learning // ArXiv:1910.12179. — 2020. — February. — **https://oreil.ly/tYOKM**.

[6] Dargazany A. Model-based actor-critic: GAN + DRL (actor-critic) → AGI // ArXiv:2004.04574. — 2020. — April. — URL: **https://oreil.ly/UC65B**.

[7] Mandlekar A., Ramos F., Boots B. et al. IRIS: Implicit reinforcement without interaction at scale for learning control from offline robot manipulation data // ArXiv: 1911.05321. — 2020. — February. — URL: **https://oreil.ly/RP5Q5**.

[8] Gulcehre C., Wang Z., Novikov A. et al. RL unplugged: benchmarks for offline reinforcement learning // ArXiv:2006.13888. — 2020. — July. — **https://oreil.ly/Lxpf0**.

[9] Banerjee B., Sen S., Peng J. On-policy concurrent reinforcement learning // J. Exp. & Theor. Artificial Intelligence. — 2004. — Vol. 16, № 4. — P. 245–60. — URL: **https://oreil.ly/5caMf**.

[10] Marthi B., Russell S., Latham D., Guestrin C. Concurrent hierarchical reinforcement learning // Proceedings of the 19th International Joint Conference on Artificial Intelligence. IJCAI '05. — Edinburgh, Scotland: Morgan Kaufmann Publishers Inc., 2005. — P. 779–785.

[11] Silver D., Newnham L., Barker D., Weller S., McFall J. Concurrent reinforcement learning from customer interactions // International Conference on Machine Learning. — 2013. — P. 924–32. — URL: **https://oreil.ly/BD4R-**.

[12] Dimakopoulou M., Van Roy B. Coordinated exploration in concurrent reinforcement learning // ArXiv:1802.01282. — 2018. — February. — URL: **https://oreil.ly/Cd72Q**.

[13] Han W., Levine S., Abbeel P. Learning compound multi-step controllers under unknown dynamics // 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). — 2015. — P. 6435–6442. — URL: **https://oreil.ly/z2S9L**.

[14] Richter C., Roy N. Safe visual navigation via deep learning and novelty detection // MIT Web Domain. — 2017. — July. — URL: **https://oreil.ly/C6Bi_**.

[15] Eysenbach B., Gu S., Ibarz J., Levine S. Leave no trace: learning to reset for safe and autonomous reinforcement learning // ArXiv:1711.06782. — 2017. — November. — URL: **https://oreil.ly/p2nwJ**.

[16] Zhu H., Yu J., Gupta A. et al. The ingredients of real-world robotic reinforcement learning // ArXiv:2004.12570. — 2020. — April. — URL: **https://oreil.ly/kaodL**.

[17] Brockman G., Cheung V., Pettersson L. et al. OpenAI Gym // ArXiv:1606.01540. — 2016. — June. — URL: **https://oreil.ly/tnnqe**.

[18] Ruiz N., Schulter S., Chandraker M. Learning to simulate // ArXiv:1810.02513. — 2019. — May. — URL: **https://oreil.ly/jwxX0**.

[19] Kalashnikov D., Irpan A., Pastor P. et al. QT-opt: scalable deep reinforcement learning for vision-based robotic manipulation // ArXiv:1806.10293. — 2018. — November. — URL: **https://oreil.ly/Mchmp**.

[20] Karakovskiy S., Togelius J. The Mario AI benchmark and competitions // IEEE transactions on computational intelligence and AI in games. — 2012. — Vol. 4, № 1. — P. 55–67. — URL: **https://oreil.ly/6KKCf**.

[21] Ota K., Oiki T., Jha D. K., Mariyama T., Nikovski D. Can increasing input dimensionality improve deep reinforcement learning? // ArXiv:2003.01629. — 2020. — June. — URL: **https://oreil.ly/_JNf9**.

[22] Hafner D., Lillicrap T., Ba J., Norouzi M. Dream to control: learning behaviors by latent imagination // ArXiv:1912.01603. — 2020. — March. — URL: **https://oreil.ly/2T1oB**.

[23] Hafner D., Lillicrap T., Fischer I. et al. Learning latent dynamics for planning from pixels // ArXiv:1811.04551. — 2019. — June. — URL: **https://oreil.ly/fRp2X**.

[24] Ha D., Schmidhuber J. World models // ArXiv:1803.10122. — 2018. — March. — URL: **https://oreil.ly/rrPrn**.

[25] Chaudhari H. A., Byers J. W., Terzi E. Learn to Earn: enabling coordination within a ride hailing fleet // ArXiv:2006.10904. — 2020. — July. — URL: **https://oreil.ly/L57Kf**.

[26] Value function approximation in reinforcement learning using the fourier basis // Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence. — 2020. — 18 July 2020. — URL: **https://oreil.ly/FmeN3**.

[27] Tang Y., Agrawal S. Discretizing continuous action space for on-policy optimization // ArXiv:1901.10500. — 2020. — March. — URL: **https://oreil.ly/cukso**.

[28] Gonzalez-Abril L., Cuberos F. J., Velasco F., Ortega J. A. Ameva: an autonomous discretization algorithm // Expert Systems with Applications: An International Journal. — 2009. — Vol. 36, № 3. — P. 5327–5332. — URL: **https://oreil.ly/eb7oY**.

[29] Dougherty J., Kohavi R., Sahami M. Supervised and unsupervised discretization of continuous features // Proceedings of the Twelfth International Conference on International Conference on Machine Learning. ICML'95. — Tahoe City, California, USA: Morgan Kaufmann Publishers Inc., 1995. — P. 194–202.

[30] van Seijen H., Bakker B., Kester L. J. H. M. Switching between different state representations in reinforcement learning. — 2017. — November. — URL: **https://oreil.ly/eJ1go**.

[31] Ryu M., Chow Y., Anderson R., Tjandraatmadja C., Boutilier C. CAQL: continuous action Q-learning // ArXiv:1909.12397. — 2020. — February. — URL: **https://oreil.ly/G_AOF**.

[32] Xiong J., Wang Q., Yang Z. et al. Parametrized deep Q-networks learning: reinforcement learning with discrete-continuous hybrid action space // ArXiv:1810.06394. — 218. — October. — URL: **https://oreil.ly/z2kSN**.

[33] Tavakoli A., Pardo F., Kormushev P. Action branching architectures for deep reinforcement learning // ArXiv:1711.08946. — 2019. — January. — URL: **https://oreil.ly/iLcmj**.

[34] Khan A., Feng J., Liu S., Asghar M. Z, Li L.-L. Optimal skipping rates: training agents with fine-grained control using deep reinforcement learning // Journal of Robotics. — 2019. — January. — URL: **https://oreil.ly/GfKDp**.

[35] Sutton R. S., Precup D., Singh S. Between MDPs and semi-MDPs: a framework for temporal abstraction in reinforcement learning // Artificial Intelligence. — 1999. — Vol. 112, № 1. — P. 181–211. — URL: **https://oreil.ly/1iEJc**.

[36] Khetarpal K., Klissarov M., Chevalier-Boisvert M., Bacon P.-L., Precup D. Options of interest: temporal abstraction with interest functions // ArXiv:2001.00271. — 2020. — January. — URL: **https://oreil.ly/KDyb9**.

[37] Biedenkapp A., Rajan R., Hutter F., Lindauer M. Towards TempoRL: learning when to act // Workshop on Inductive Biases, Invariances and Generalization in RL (BIG@ICML '20). — 2020.

[38] Ie E., Jain V., Wang J. et al. Reinforcement learning for slate-based recommender systems: A tractable decomposition and practical methodology // ArXiv:1905.12767. — 2019. — May. — URL: **https://oreil.ly/y7Fpd**.

[39] Dulac-Arnold G., Evans R., Van Hasselt H. et al. Deep reinforcement learning in large discrete action spaces // ArXiv:1512.07679. — 2016. — April. — URL: **https://oreil.ly/BeS1T**.

[40] Van de Wiele T., Warde-Farley D., Mnih A., Mnih V. Q-learning in enormous action spaces via amortized approximate maximization // ArXiv:2001.08116. — 2020. — January. — URL: **https://oreil.ly/Rjmv9**.

[41] Mehrotra R., Lalmas M., Kenney D., Lim-Meng T., Hashemian G. Jointly leveraging intent and interaction signals to predict user satisfaction with slate recommendations // The World Wide Web Conference. WWW 2019. — San Francisco, CA, USA: Association for Computing Machinery, 2019. — P. 1256–1267. — URL: **https://oreil.ly/py0m8**.

[42] Jiang R., Gowal S., Mann T. A., Rezende D. J. Beyond greedy ranking: slate optimization via list-CVAE // ArXiv:1803.01682. — 2019. — February. — URL: **https://oreil.ly/aOeNP**.

[43] Metz L., Ibarz J., Jaitly N., Davidson J. Discrete sequential prediction of continuous actions for deep RL // ArXiv:1705.05035. — 2019. — June. — URL: **https://oreil.ly/i7OAG**.

[44] Ie E., Jain V., Wang J. et al. Reinforcement learning for slate-based recommender systems: a tractable decomposition and practical methodology // ArXiv:1905.12767. — 2019. — May. — URL: **https://oreil.ly/o-qVT**.

[45] Kosoy E., Collins J., Chan D. M. et al. Exploring exploration: comparing children with RL agents in unified environments // ArXiv:2005.02880. — 2020. — July. — URL: **https://oreil.ly/dqM6q**.

[46] Murphy K. P. Machine learning: a probabilistic perspective. — MIT Press, 2012.

[47] Houthooft R., Chen X., Duan Y., Schulman J., De Turck F., Abbeel P. VIME: variational information maximizing exploration // ArXiv: 1605.09674. — 2017. — January. — URL: **https://oreil.ly/fHafR**.

[48] Pathak D., Agrawal P., Efros A. A., Darrell T. Curiosity-driven exploration by self-supervised prediction // ArXiv:1705.05363. — 2017. — May. — URL: **https://oreil.ly/3a5RL**.

[49] Burday Y., Edwards H., Pathak D. et al Large-scale study of curiosity-driven learning. — 2020. — 22 July. — URL: **https://oreil.ly/9zP22**.

[50] Burda Y., Edwards H., Storkey A., Klimov O. Exploration by random network distillation // ArXiv:1810.12894. — 2018. — October. — URL: **https://oreil.ly/w3UVl**.

[51] Saxe A. M., Koh P. W., Chen Z. et al. Random weights and unsupervised feature learning // Proceedings of the 28th International Conference on International Conference on Machine Learning, 1089–1096. ICML '11. — Bellevue, Washington, USA: Omnipress, 2011.

[52] Savinov N., Raichuk A., Marinier R. et al. Episodic Curiosity through reachability // ArXiv:1810.02274. — 2019. — August. — UR: **https://oreil.ly/GGuyL**.

[53] Zagoruyko S., Komodakis N. Learning to compare image patches via convolutional neural networks // ArXiv:1504.03641. — 2015. — April. — URL: **https://oreil.ly/UpYiy**.

[54] Burda Y., Edwards H., Storkey A., Klimov O. Exploration by random network distillation // ArXiv:1810.12894. — 2018. — October. — URL: **https://oreil.ly/w3UVl**.

[55] Van Hasselt H., Guez A., Hessel M., Mnih V., Silver D. Learning values across many orders of magnitude // ArXiv:1602.07714. — 2016. — August. — URL: **https://oreil.ly/mTaNn**.

[56] Zhu H., Yu J., Gupta A. et al. The ingredients of real-world robotic reinforcement learning // ArXiv:2004.12570. — 2020. — April. — URL: **https://oreil.ly/ArO_j**.

[57] Trott A., Zheng S., Xiong C., Socher R. Keeping your distance: solving sparse reward tasks using self-balancing shaped rewards // ArXiv:1911.01417. — 2019. — November. — URL: **https://oreil.ly/tNsJD**.

[58] Lange S., Gabel T., Riedmiller M. Batch reinforcement learning // Reinforcement learning: state-of-the-art / ed. by M. Wiering, M. van Otterlo. — Adaptation, Learning, and Optimization. — Berlin, Heidelberg: Springer, 2012. — P. 45–73. — URL: **https://oreil.ly/QSSBK**.

[59] Levine S., Kumar A., Tucker G., Fu J. Offline reinforcement learning: tutorial, review, and perspectives on open problems // ArXiv: 2005.01643. — 2020. — May. — URL: **https://oreil.ly/BZxIP**.

[60] Wong J. M. Towards lifelong self-supervision: a deep learning direction for robotics // ArXiv:1611.00201. — 2016. — November. — URL: **https://oreil.ly/iIJ-H**.

[61] Lesort T., Díaz-Rodríguez N., Goudou J.-F., Filliat D. State Representation learning for control: an overview // Neural Networks. — 2018. — № 108 (December). — P. 379–92. — URL: **https://oreil.ly/B7LML**.

[62] Sutton R. S., Barto A. G. Reinforcement learning: an introduction. — MIT Press, 2018.

[63] Dougherty J., Kohavi R., Sahami M. Supervised and unsupervised discretization of continuous features // Proceedings of the Twelfth International Conference on International Conference on Machine Learning. ICML'95. — Tahoe City, California, USA: Morgan Kaufmann Publishers Inc., 1995. — P. 194–202.

[64] Oudeyer P.-Y., Kaplan F. What is intrinsic motivation? A typology of computational approaches // Frontiers in Neurorobotics. — 2007. — 1 November. — URL: **https://oreil.ly/Hi5yn**.

[65] Russo D., Van Roy B., Kazerouni A., Osband I., Wen Z. A tutorial on Thompson sampling // ArXiv:1707.02038. — 2020. — July. — URL: **https://oreil.ly/t4DUY**.

ГЛАВА 10

# Этапы в обучении с подкреплением

Чем ближе вы подходите к внедрению, тем ближе вы к краю. Не зря обучение с подкреплением называют передовым. Довольно сложно довести свой проект обучения с подкреплением (reinforcement learning, RL) до этого момента, но внедрение в производство и оперативное развертывание создают целый ряд новых проблем.

Насколько мне известно, эта книга — первая попытка собрать оперативные знания о RL в одном месте. Эта информация разбросана по удивительным работам исследователей и книгам многих ярчайших умов отрасли, но в одном месте — никогда.

В этой главе я проведу вас через процесс внедрения вашей концепции в производство, подробно описав этапы реализации и развертывания проекта RL. К концу я надеюсь, что эти идеи найдут отклик и у вас будет достаточно обширных знаний, чтобы, по крайней мере, начать и понимать, куда вам нужно копать дальше. С помощью этой главы и, конечно же, всей книги я ставлю перед собой цель привнести RL в промышленность и продемонстрировать, что отраслевое RL не только возможно, но и прибыльно.

В первой половине я анализирую этап реализации проекта RL, углубляясь в доступные фреймворки, абстракции и способы масштабирования. Здесь также важно знать, как оценивать агентов, потому что недопустимо полагаться на специальные или статистически необоснованные показатели эффективности.

Вторая половина посвящена операционному RL и развертыванию вашего агента в производственной среде. Я говорю о надежных архитектурах и необходимых инструментах, которые понадобятся вам для создания надежных агентов. И, наконец, что, возможно, наиболее важно, есть раздел о том, как сделать RL безопасным, надежным и этичным; обязательно прочтите его.

## Реализация

Лично я нахожу этап реализации проекта машинного обучения / искусственного интеллекта / больших данных /{вставьте сюда модное слово} одной из самых увлекательных и интересных частей всего жизненного цикла. На этом этапе вам нужно позаботиться о коде, который вы пишете, потому что этот код может просуществовать десятилетия или даже дольше. GitHub недавно создал в Арктике архив, в котором все активные проекты хранятся на микрофильмах, и, по прогнозам, архив просуществует не менее 1000 лет [1]. Если вы когда-либо использовали достаточно

популярный репозиторий GitHub, примите наши поздравления, люди 3000 года смогут прочитать ваш код.

Как бы унизительно это ни звучало, основная причина создания качественного кода — снижение затрат на техническое обслуживание. Каждая строка кода добавляет еще один источник потенциальной ошибки. Вы можете снизить вероятность ошибки с помощью простоты, тестирования и пристального внимания к коду. Еще одна причина качественного кода в том, что он упрощает его использование. Масштаб проектов может увеличиваться довольно быстро, поэтому следование фундаментальным рекомендациям по качеству программного обеспечения может облегчить этот переход.

Предыдущие темы не относятся к RL; существует множество отличных книг по разработке программного обеспечения, которые я рекомендую прочитать каждому инженеру, и некоторые из них я предлагаю *в разд. "Дополнительные источники" в конце данной главы*. Этот раздел концентрируется на аспектах, специфичных для RL.

## Фреймворки

Один из самых быстрых способов разработки качественных RL-приложений — это использование фреймворка. Инженеры потратили много времени, чтобы облегчить вам разработку вашего проекта.

### Фреймворки RL

Фреймворки RL с открытым исходным кодом выигрывают от количества людей, которые их используют. Бóльшая популярность означает больше испытаний и больше стимулов для улучшения проекта. Возможность проверять общедоступный исходный код чрезвычайно ценна, особенно когда вы сталкиваетесь со сложной ошибкой.

Проблема с популярностью заключается в том, что она обычно пропорциональна размеру маркетингового бюджета и предполагаемому опыту. Например, любой репозиторий, имеющий отношение к Google (поищите по ключевой фразе "неофициальный продукт Google" (not an official Google product), что обычно означает, что он был разработан, но официально не поддерживается), как правило, очень популярен, даже если реальных пользователей у него мало. То же самое можно предложить и для OpenAI.

Подумайте, совпадают ли цели фреймворка с открытым исходным кодом с вашими целями. Например, многие из существующих структур сильно ориентированы на исследования, поскольку они были разработаны отдельными лицами или организациями, которым платят за проведение исследований. Они не предназначены для использования в продуктах или в эксплуатации. Классическим примером этого является проект OpenAI Baselines[1], который остается одной из самых популярных доступных кодовых баз RL, но проект с открытым исходным кодом Stable

[1] См. **https://oreil.ly/Zkbbp**.

Baselines[2], который гораздо незаметнее с точки зрения маркетинга, намного проще в использовании и более надежен, на мой взгляд [2, 3].

Итак, что вам следует искать? Это зависит от того, что вы пытаетесь сделать. Если вы хотите быстро протестировать различные алгоритмы для решения своей задачи, тогда у вас будут другие требования к тем, кому нужно запускать фреймворк в производственной среде. Вы должны расставить приоритеты и взвесить факторы в соответствии с вашими обстоятельствами и потребностями, но в следующем списке перечислены некоторые сущности, которые я ищу.

- *Абстракции.*

  Есть ли у фреймворка хорошие абстракции, упрощающие замену компонентов RL? Например, модели политик, методы исследования, типы буферов воспроизведения и т. д. Насколько легко вы можете разработать новые, подходящие для вашей задачи? Это, как правило, труднее всего оценить, потому что вы действительно не узнаете, пока не попробуете.

- *Простота.*

  Достаточно ли проста кодовая база для навигации? Есть ли в ней смысл и интуитивно ли он понятен? Или фреймворк настолько велик или содержит столько абстракций, что вам нужны недели, чтобы понять, как его установить?

- *Наблюдаемость.*

  Включает ли фреймворк ведение журнала и мониторинг? Если нет, легко ли добавить их, когда вам нужно?

- *Масштабируемость.*

  Может ли фреймворк масштабироваться, когда вам это нужно?

- *Минимальные зависимости.*

  Насколько фреймворк зависит от другого программного обеспечения? Закреплен ли он в определенной структуре, например в конкретной библиотеке глубокого обучения? Если да, одобряете ли вы базовые библиотеки?

- *Документация и примеры.*

  Хорошо ли задокументирован проект? Не только документация по API: есть ли примеры? Они актуальны и протестированы? Есть документация по архитектуре? Документация по вариантам использования? Документация по алгоритму?

- *Включенные алгоритмы.*

  Есть ли во фреймворке встроенные реализации алгоритмов? Хорошо ли они охвачены тестами и примерами? Насколько легко переключить алгоритмы?

- *Вспомогательные алгоритмы.*

  Поставляется ли фреймворк с кодом для множества других ситуаций, таких как обратный RL, иерархическое обучение, различные реализации буфера воспроизведения или алгоритмы исследования и т. д.?

---

[2] См. **https://oreil.ly/4hDLC**.

- *Служба поддержки.*

  Есть ли поддержка? Есть ли сообщество? Хорошо ли он представлен на краудсорсинговых сайтах самопомощи? Поддержка приятная или агрессивная?

- *Коммерческие предложения.*

  Поддерживается ли продукт компанией (не обязательно хорошо, но, по крайней мере, у вас должна быть возможность поговорить с кем-то)? Есть ли возможность получить коммерческую поддержку?

Существует множество фреймворков RL, и все они делают акцент на различных элементах в этом списке. Программное обеспечение быстро развивается, и любые комментарии о конкретных фреймворках быстро устареют. Вместо этого посетите соответствующий веб-сайт[3], чтобы найти актуальный обзор многих фреймворков RL.

## Другие фреймворки

Одной только структуры RL недостаточно. Для создания решения вам понадобится множество других фреймворков. Многие части алгоритма RL были абстрагированы, например различные структуры глубокого обучения, используемые в моделях политик и приближенных значениях. PyTorch[4] и TensorFlow[5], как правило, являются наиболее распространенными, и я предпочитаю PyTorch благодаря его более простому и интуитивно понятному API, но Keras[6] API может помочь сгладить эту разницу. Большинство фреймворков глубокого обучения являются зрелыми и имеют схожие наборы функций, но их API совершенно разные. Это серьезный источник блокировки дальнейших действий, поэтому выбирайте внимательно.

Также растет число библиотек, которые абстрагируют другие общие компоненты RL. Например, воспроизведение опыта важно для многих алгоритмов. Reverb[7] — это библиотека очередей, которая была специально разработана для обработки таких реализаций, как приоритетное воспроизведение опыта [4]. Apache Bookkeeper[8] (распределенная бухгалтерская книга) является примером инструмента производственного уровня, который можно использовать как часть операционной архитектуры в виде журнала.

Я хочу подчеркнуть, что вам необходимо тщательно продумать операционную архитектуру и оценить, какие инструменты доступны для решения этой проблемы (*см. разд. "Архитектура" далее в этой главе*). Вы можете найти более свежую информацию об инструментах на веб-сайте[9], который сопровождает эту книгу.

---

[3] См. **https://rl-book.com/rl-frameworks**.

[4] См. **https://pytorch.org/**.

[5] См. **https://oreil.ly/01Uno**.

[6] См. **https://keras.io/**.

[7] См. **https://oreil.ly/CxJAZ**.

[8] См. **https://oreil.ly/1RM4z**.

[9] См. **https://rl-book.com/**.

## Масштабирование RL

Сложные задачи труднее решить, поэтому для них требуются более комплексные модели и скрупулезное исследование. Это приводит к увеличению вычислительной сложности до такой степени, что становится невозможным обучить агента на одной машине.

При разработке время цикла обратной связи является наиболее важным показателем для мониторинга, потому что все время, которое вы ожидаете результата, — это время, не потраченное на улучшение агента. Вы, конечно, можете проводить эксперименты параллельно, но на данном этапе такой подход оказывает ограниченное влияние на результат (*см. разд. "Отслеживание экспериментов" далее в этой главе*). На ранних этапах исследования RL наибольшие улучшения происходят за счет быстрых повторяющихся циклов улучшений. Поэтому я рекомендую вам изначально ограничить объем своей работы, чрезмерно упростив задачу, используя моделирование и значительно снизив сложность ваших моделей. Когда у вас будет рабочий код, вы можете масштабировать его.

Но как увеличить масштаб? В отличие от других методов машинного обучения, увеличение мощности не обязательно улучшает скорость или производительность тренировки. Далее приводится список причин, согласно которым вам нужны специальные алгоритмы для масштабирования RL.

- *Последовательный.*

  Действия агента должны быть упорядочены, что затрудняет масштабирование отдельного агента на разных машинах.

- *Объединение опыта.*

  Если вы используете несколько агентов, им потребуется синхронизация, иначе вы получите множество независимых копий, которые не смогут обмениваться опытом. Процесс синхронизации может оказать большое влияние на скорость обучения и общую производительность. Пропускная способность сети и задержка также могут стать проблемой.

- *Независимость.*

  Независимость от агентов означает, что некоторые агенты могут зависнуть или никогда не завершить работу, что способно вызвать блокировку, если синхронизация ожидает завершения агентов.

- *Сложность среды.*

  Некоторые среды, такие как 3D-симуляторы, настолько сложны, что сами по себе нуждаются в ускорении, а это может вызвать конкуренцию за графический процессор (graphics processing unit, GPU), если не будет разумной организации.

- *Экономическая эффективность.*

  Масштабирование может быть дорогостоящим, поэтому вы должны быть уверены, что полностью используете оборудование. Необходимо создание архитектур и алгоритмов, которые лучше используют преимущества вычислений как на GPU, так и на CPU (central processing unit — центральный процессор).

Последний пункт — экономическая эффективность — является большой проблемой для машинного обучения в целом. Например, не так давно представленнаяя языковая модель OpenAI GPT-3[10] будет стоить в пределах 10 млн долларов для обучения с нуля по прейскурантам текущего поставщика облачных услуг. И это при одном запуске. Представьте себе ужас, когда обнаружите ошибку!

Очевидно, это крайний пример. В своей стате Эспехолт и соавт. обнародовали измеренную стоимость обработки 1 млрд кадров в средах DeepMind Lab и Google Research Football. Обе среды обеспечивают трехмерное моделирование для ряда задач. Исследователи протестировали два алгоритма под названием IMPALA и SEED (обсуждаемые ниже) и обнаружили, что общая стоимость одного запуска (1 млрд кадров) составила 236 и 54 доллара для DeepMind Lab и 899 и 369 долларов для Google Research Football соответственно. Это не астрономические цифры в контексте корпоративного ИТ-проекта, но их стоит учитывать [5].

## Распределенное обучение (Gorila)

Ранние попытки масштабирования RL были сосредоточены на создании параллельных акторов, чтобы лучше использовать многоядерные машины или многомашинные кластеры. Одной из первых была *общая архитектура обучения с подкреплением* (general reinforcement learning architecture, Gorila), которая была продемонстрирована с помощью распределенной версии глубоких Q-сетей (deep Q-networks, DQN), основанной на распределенной системе обучения нейронных сетей под названием *DistBelief* [6, 7]. DistBelief состоит из двух основных компонентов: централизованного сервера параметров и реплик моделей. Централизованный сервер параметров поддерживает и обслуживает главную копию модели и отвечает за обновление модели (с новыми градиентами) из реплик. Реплики, распределенные по ядрам или машинам, отвечают за вычисление градиентов для заданного пакета данных, отправку их на сервер параметров и периодическое получение новой модели. Для того чтобы использовать эту архитектуру в настройках RL, Gorila вносит следующие изменения. Централизованный сервер параметров поддерживает и обслуживает параметры модели (которые зависят от выбранного алгоритма RL). Акторские процессы получают текущую модель и генерируют развертывания в среде. Их опыт сохраняется в централизованном буфере воспроизведения (если это соответствует алгоритму). Процессы обучения используют выбранные траектории для изучения новых градиентов и передачи их обратно на сервер параметров. Преимущество этой архитектуры заключается в том, что она обеспечивает гибкость для масштабирования тех частей процесса обучения, которые представляют собой узкое место. Например, если у вас есть среда, требующая больших вычислительных ресурсов, вы можете масштабировать количество процессов-акторов. Или, если у вас большая модель, а узким местом является обучение, вы можете увеличить количество учащихся. На рис. 10.1 изображены эти две архитектуры на схеме компонентов.

[10] См. **https://oreil.ly/KtNHP**.

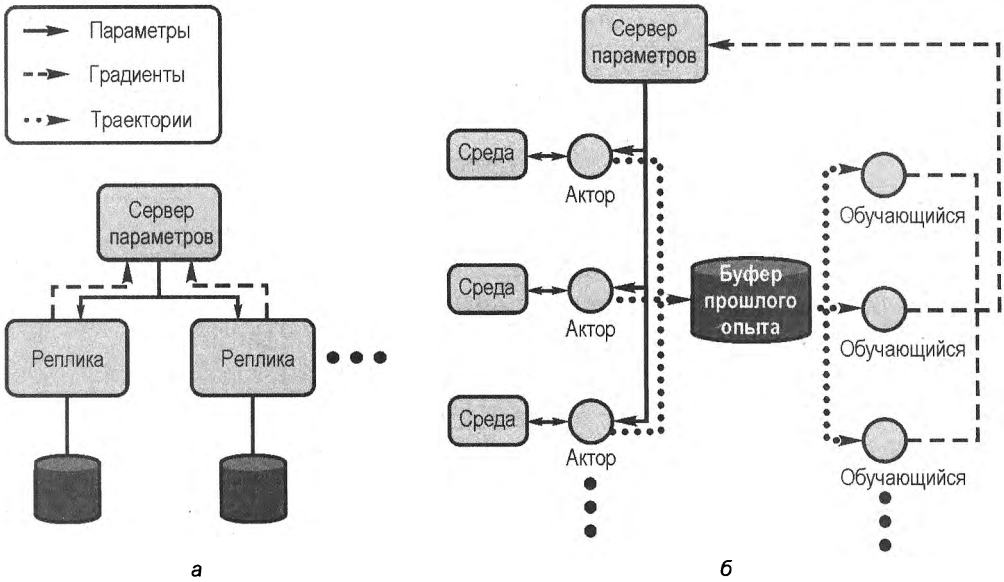



**Рис. 10.1.** Компонентная схема, обобщающая архитектуру DistBelief (*а*) и Gorila (*б*). Gorila позволяет масштабировать, разделяя задачи действия и обучения, а затем репликации

## Обучение на одной машине (A3C, PAAC)

Использование Gorila в средах Atari привело к превосходной производительности, более чем в 20 раз быстрее (в реальном времени), чем только лишь DQN, но исследователи обнаружили, что распределенная архитектура, хотя и гибкая, требует быстрых сетей. Передача траекторий и синхронизация параметров стали узким местом. Для того чтобы ограничить затраты на обмен данными и упростить архитектуру, исследователи продемонстрировали, что распараллеливание с использованием одной многоядерной машины является жизнеспособной альтернативой алгоритму, называемому алгоритмом *асинхронного алгоритма "актор — критик" с преимуществом* (asynchronous advantage actor-critic, A3C) [8]. Это параллельная версия алгоритма "актор — критик" с преимуществом (advantage actor-critic, A2C). Поскольку обновления находятся на одном компьютере, обучающиеся могут совместно использовать память, что значительно снижает накладные расходы на связь из-за быстрого чтения/записи в оперативное запоминающее устройство (ОЗУ). Интересно, что они не используют блокировку при записи в разделяемую память по потокам, что обычно является невероятно опасным делом, вместо этого они применяют технику под названием Hogwild [9]. Они обнаруживают, что обновления являются разреженными и стохастическими, поэтому перезаписи и конкуренция отсутствуют. На рис. 10.2 показаны различия между алгоритмами A2C и A3C, которые работают на одной машине.

Удивительно, но, используя одну машину с 16-ядерным процессором без GPU, алгоритм A3C получил сопоставимые с Gorila результаты за одну восьмую времени (1 день) и лучшие результаты примерно за половину времени (4 дня). Это ясно

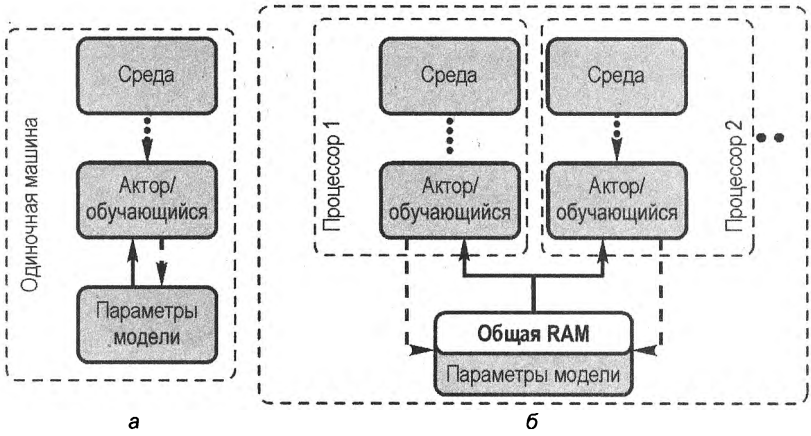



**Рис. 10.2.** Компонентная схема, обобщающая архитектуру для A2C (*а*) и A3C (*б*). Оба алгоритма работают на одной машине, но A3C использует несколько потоков с реплицированными агентами, которые записывают в общую память

демонстрирует, что накладные расходы на синхронизацию и связь — большая проблема.

Простота отказа от использования GPU сдерживает A3C. Весьма вероятно, что GPU ускорит обучение чего-либо "глубокого", конечно, с большей стоимостью. Реализация *параллельного алгоритма "актор — критик" с преимуществом* (parallel advantage actor critic, PAAC), название которого не очень удачное, потому что вы можете использовать эту архитектуру с любым алгоритмом RL, возвращает возможность использования GPU. Он базируется на одном обучающем потоке, который может использовать один подключенный GPU и множество акторов в таком же количестве сред. В некотором смысле это намеренно менее масштабируемая версия, наподобие Gorila и A3C, но она дает результаты, сопоставимые с A3C за одну возьмую времени (опять же!), создавая эффективные политики для многих игр Atari примерно за 12 часов, в зависимости от архитектуры нейронной сети. Эта реализация показывает, что увеличение количества исследований в независимых средах — очень хороший способ сократить время обучения в реальном времени. Время обучения пропорционально количеству пространства "состояние — действие", которое вы можете исследовать. Подобно A3C эта реализация демонстрирует, что передача данных и синхронизация являются узким местом, и если ваша задача позволяет, вы выиграете, если будете держать компоненты как можно ближе друг к другу, предпочтительно внутри одной физической машины. Это тоже простое решение в том смысле, что нет проблем с оркестрацией или сетью. Но очевидно, что реализация алгоритма параллельного преимущества "актор — критик" может не подходить для всех задач. На рис. 10.3 представлена архитектура PAAC [10].

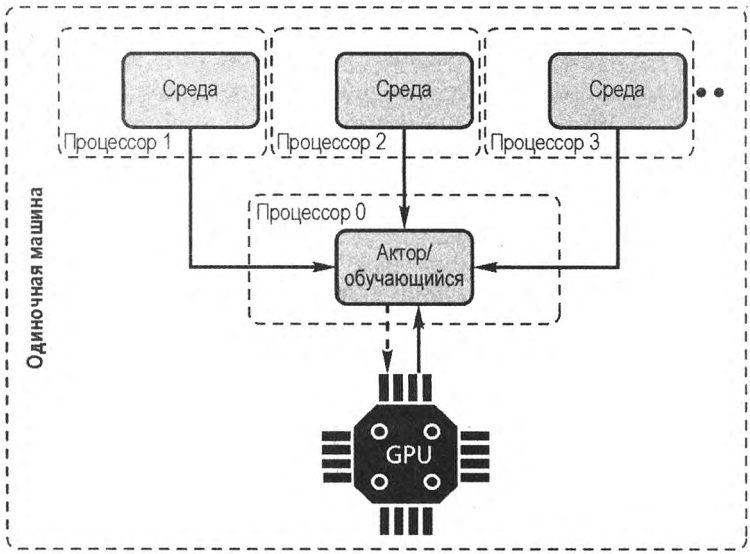



**Рис. 10.3.** Компонентная схема, обобщающая архитектуру PAAC. Одна машина с GPU и множеством процессоров используется для масштабирования количества сред

## Распределенное воспроизведение (Ape-X)

Использование других типов буфера воспроизведения в распределенной среде так же просто, как расширение архитектуры Gorila (или другой). *Ape-X* является одним из таких примеров, в котором более широкое исследование с помощью распределенных акторов со средами привело к лучшим результатам, поскольку было собрано более разнообразное взаимодействие. Показатели производительности в средах Atari были примерно в три раза выше, чем у AC3, и в четыре раза выше, чем у Gorila. Но эти оценки были обучены на 16-кратном количестве CPU (256 против 16 для A3C), что привело к увеличению взаимодействия со средой на два порядка. Неясно, была бы разница в производительности при использовании того же количества сред/CPU [11].

## Синхронное распределение (DD-PPO)

Попытка использовать одну машину с множеством CPU для обучения алгоритму RL, как в PAAC или AC3, заманчива благодаря простотой архитектуре. Но отсутствие GPU становится ограничивающим фактором, когда в вашей задаче используются 3D-симуляторы, которым требуются GPU по соображениям производительности. Точно так же сервер с одним параметром и одним GPU, как в Gorila или PAAC/AC3, становится ограничивающим фактором, когда вы пытаетесь обучить массивные архитектуры глубокого обучения, например *ResNet50*. Эти факторы требуют наличия распределенной архитектуры, способной использовать несколько GPU.

Целью *децентрализованной распределенной проксимальной оптимизации политики* (decentralized distributed proximal policy optimization, DD-PPO) является децентрали-

зация обучения, во многом подобно децентрализованному кооперативному мультиагентному RL. Эта архитектура чередует свои действия между сбором опыта и оптимизацией модели, а затем предусматривает синхронное взаимодействие с другими воркерами (исполнителями) для распространения обновлений модели (градиентов). Воркеры используют распределенное параллельное обновление градиента от PyTorch и комбинируют градиенты с помощью простого объединения всех других воркеров. Результаты показывают, что DD-PPO может масштабировать производительность GPU, увеличивая количество воркеров почти линейно, протестировано около 256 воркеров/GPU. Основная проблема с этой реализацией заключается в том, что обновление не может произойти, пока все агенты не соберут свой пакет. Для того чтобы облегчить эту блокировку, Вейманс и соавт. реализуют превентивное отключение, которое вынуждает часть отставших выйти раньше. На рис. 10.4 эта архитектура представлена в общем виде, а Вейманс и соавт. предполагают, что можно реализовать любой алгоритм RL [12]. Другие исследователи предложили аналогичную архитектуру [13].

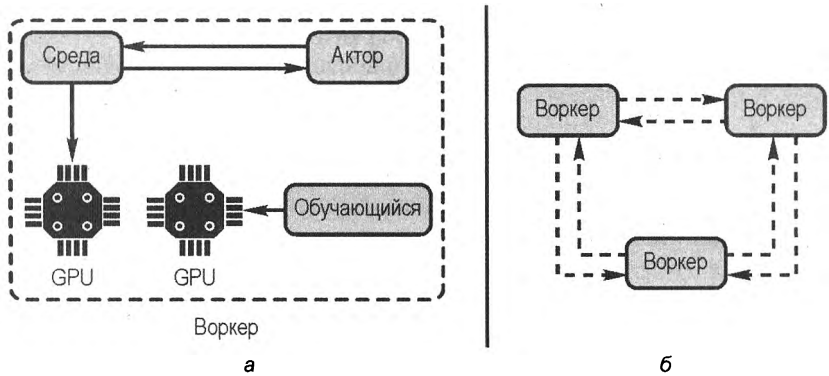



**Рис. 10.4.** Компонентная схема, обобщающая архитектуру DD-PPO. Воркер (*а*) включает среду, актора и обучающегося, которые могут использовать графические процессоры. После того как агент соберет пакет данных, каждый воркер выполняет распределенное обновление для распределения весов (*б*)

## Повышение эффективности использования (IMPALA, SEED)

*Взвешенная по значимости архитектура "актор — ученик"* (importance weighted actor-learner architecture, IMPALA) похожа на комбинацию Gorila и DD-PPO. Ключевым достижением является то, что она использует очередь для устранения необходимости синхронизации обновлений. Актор многократно выкатывает опыт и помещает данные в очередь. Отдельная машина (или группа машин) считывает данные из очереди и повторно обучает модель. Это означает, что обновление не соответствует политике; обучающийся использует данные из прошлого. Таким образом, IMPALA использует модифицированную версию алгоритма восстановления (*см. разд. "Алгоритм Retrace (λ)" главы 6*), чтобы компенсировать задержку в обновлении политики. Этот подход также позволяет использовать буфер воспроизведения, что повышает производительность.

Эспехольт и соавт. демонстрируют, что алгоритм "актор — критик", действующий вне политики (*см. разд. "Алгоритм „актор — критик" вне политики " главы 6*), работает почти так же, а это означает, что более сложные версии, такие как TD3, могут работать лучше. Действительно, OpenAI использовала фреймворк, вдохновленный IMPALA, с PPO для соревнования по видеоиграм Dota 2 [14].

Одиночные обучающиеся с несколькими акторами обычно работают на одной машине. Вы можете продублировать архитектуру, чтобы увеличить количество учащихся, и использовать распределенное обучение, такое как DD-PPO. Эта архитектура, изображенная на рис. 10.5, улучшает скорость обучения на порядок по сравнению с A3C, но пропорциональна количеству оборудования в кластере (в экспериментах были использованы 8 GPU). Эспехольт и соавт. также утверждают, что эту архитектуру можно масштабировать до тысяч машин [15].

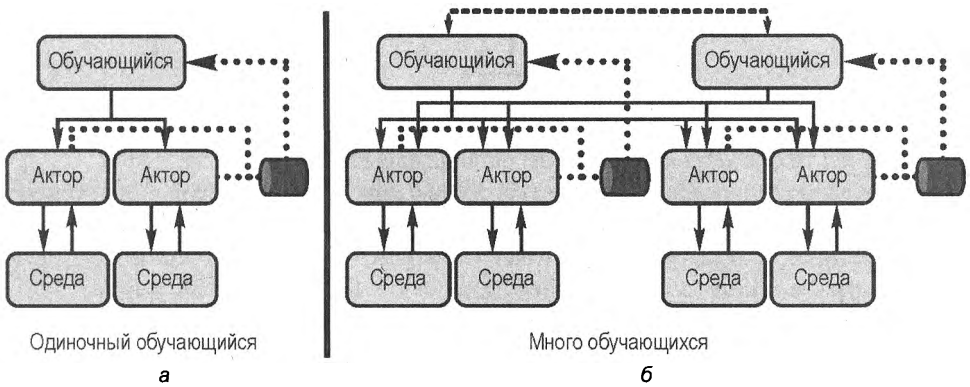



**Рис. 10.5.** Схема компонентов, обобщающая архитектуру IMPALA. Один обучающийся (*а*) имеет несколько акторов (обычно на одной машине). Акторы заполняют очередь, и обучение проводится независимо от них. При использовании нескольких учащихся (*б*) градиенты распределяются и обучаются синхронно (как в DD-PPO)

Несмотря на то что IMPALA эффективна и способна давать самые современные результаты, все же есть части архитектуры, которые можно оптимизировать. Одна из проблем — это передача параметров нейронной сети. В больших сетях они представлены миллиардами чисел с плавающей запятой, а это гигантский объем данных для передачи. С другой стороны, наблюдения, как правило, намного меньше, поэтому было бы более эффективно сообщать о наблюдениях и действиях. Кроме того, к средам и акторам предъявляются разные требования, поэтому размещать их на одном и том же CPU или GPU может быть неэффективно. Например, актору может потребоваться GPU для быстрого прогнозирования, а среде нужен только CPU (или наоборот). Это означает, что бóльшую часть времени GPU не будет использоваться.

*Масштабируемая и эффективная глубокая RL с ускоренным центральным выводом* (scalable and efficient deep RL with accelerated central inference, SEED) предназначена для архитектуры, в которой среды распределены, но запрашивают действия от центральной распределенной модели. Такой подход сохраняет параметры при-

меняемой модели близкими к базовой и позволяет вам выбирать оборудование, которое напрямую сопоставляется либо с моделированием среды, либо с обучением и прогнозированием модели. Это приводит к повышению загрузки сети и немного более быстрому обучению (примерно двукратному при аналогичных условиях). На рис. 10.6 изображена упрощенная версия архитектуры [16].

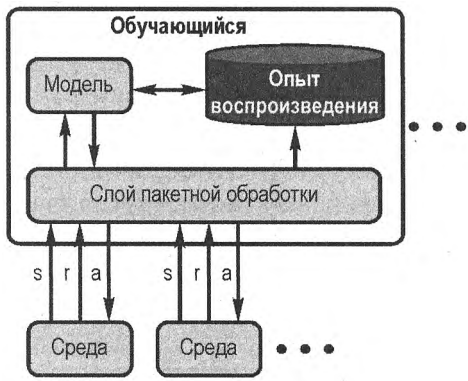



**Рис. 10.6.** Схема компонентов, обобщающая архитектуру SEED RL. Оборудование распределяется в соответствии с потребностями модели или среды

Один интересный аспект SEED заключается в том, что она подчеркивает, насколько архитектура связана с производительностью. Для того чтобы получить свои результаты, исследователям пришлось рассматривать количество сред на CPU и количество CPU как гиперпараметры. Как правило, большее количество реплик дает более высокие результаты за меньшее время, но точные конфигурации, вероятно, зависят от среды и поэтому их трудно точно определить. Еще один момент, который следует отметить, заключается в том, что SEED делает все возможное для оптимизации и на программном уровне, используя двоичный протокол связи (gRPC) и компиляцию C++.

И IMPALA, и SEED могут работать на одной машине, по аналогии с A3C и PAAC, и, как правило, более производительны из-за более высокой загрузки.

## Масштабирование сделанных выводов

Представленные алгоритмы позволяют достичь самых современных результатов за короткий период времени. Однако сравнение проводилось в игрушечных средах — в основном в среде Atari, а также иногда и в более непрерывных средах, таких как Google Research Football. Вы можете экстраполировать результаты из этих сред на реальные сценарии, но действительность такова, что вам придется провести собственное тестирование, чтобы быть уверенным в оправданности такого масштабирования.

Исследователи имеют большой опыт работы с этими средами, поэтому им не нужно тратить слишком много времени на их настройку, и они уже знают, на какие подводные камни следует обращать внимание. Они также нацелены на высокие результаты, такие как высочайшая производительность или скорость. Особое вни-

мание уделяется использованию и сверхмасштабированию, и иногда за это приходится платить. Например, полностью распределенное развертывание, такое как DD-PPO, может быть лучше для оперативного использования, потому что было бы легко сделать его самовосстанавливающимся (просто сформировать кластер из некоторого количества распределенных компонентов), что увеличивает надежность и высокую доступность. Возможно, это не представляет интереса для вашего приложения, но я считаю, что самый быстрый — не всегда лучший. Есть и другие важные аспекты.

Точно так же возможно проявление убывающей отдачи. Ранние улучшения привели к снижению скорости обучения на порядок. Разница между IMPALA и SEED составила примерно половину. Исследователям приходится углубляться в детали, чтобы добиться большей производительности, например применять статическую компиляцию кода и использовать бинарные протоколы. Опять же, это может не подходить для вашей задачи. Если, например, вам приходилось писать код на C++, а не на Python, затраты вашего времени или потеря возможностей могут быть больше, чем выгода, полученная за счет оптимизации на уровне кода.

Наконец, некоторые из этих архитектур имеют тенденцию размывать абстракции, возникшие во фреймворках. Обычно существуют абстракции для агентов, сред, буферов воспроизведения и т. п., но эти архитектуры обеспечивают расположение этих абстракций, что может быть неидеальным с точки зрения дизайна. Например, объединение модели, обучения, буфера воспроизведения и некоторой настраиваемой логики пакетной обработки в учащегося в архитектуре SEED может стать беспорядочным. Было бы здорово, если бы они были физически отдельными компонентами, такими как распределенные очереди и реестры. Очевидно, что ничто не мешает вам реализовать что-то подобное, но, как правило, это не является приоритетом для исследований, ориентированных на производительность.

## Оценка

Для того чтобы повысить ценность продукта, услуги или стратегии, вам и вашим заинтересованным сторонам необходимо тщательно обдумать, чего вы пытаетесь достичь. Это звучит абстрактно и мало понятно, но слишком легко работать над задачей только для того, чтобы обнаружить, что объект вашей оптимизации не соответствует цели. Причина в том, что инженеры могут забыть или даже не обращать внимания на связь своей работы с целью.

Психологи и неврологи глубоко изучили многозадачность. Они обнаружили, что у людей есть ограниченное количество умственной энергии, которую они могут потратить на задачи, и что переключение между несколькими сильно влияет на производительность. Но при работе над инженерными проблемами переключение между уровнями абстракции также затруднено. Ваш мозг должен выгрузить текущий набор мыслей, сохранить его в кратковременной памяти, загрузить новую ментальную модель для другого уровня абстракции и запустить мыслительный процесс. Это переключение, наряду с усилиями по поддержанию кратковременной памяти, утомительно для ума. Это одна из причин, по которой легче игнорировать высокоуровневые цели при работе над повседневными задачами низкого уровня.

Еще одна причина, даже после зарождения DevOps, заключается в том, что разработка считается успешной, когда достигается прогресс: новые продукты, новые функции, лучшая производительность и т. д. Движение назад рассматривается как отсутствие прогресса; я хочу прояснить, что это не так. "Возвращение к чертежной доске" не признак неудачи, это признак того, что вы усвоили важные уроки, которые теперь используете для разработки лучшего решения.

Несмотря на то что это может быть сложно, постарайтесь найти время двигаться в процессе разработки и вверх, и вниз по слоям абстракции, чтобы убедиться, что все они согласованы с целью, которую вы пытаетесь достичь.

Вот один прием, помогающий мне переключаться между абстракциями или контекстами: ставлю вопрос "почему?" и тем самым перехожу на уровень выше, ставлю вопрос "как?" и опускаюсь на уровень ниже в системе абстракций.

Вернемся к текущей задаче: оценивая свою работу, вы должны сначала спросить, *почему* вы хотите измерять производительность? Это нужно для объяснения результатов вашим заинтересованным сторонам? Или для того, чтобы повысить эффективность? Ответы на данные вопросы помогут определить, как вы хотите осуществить это.

## Показатели эффективности политики

Эффективность агента RL можно описать двумя абстрактными показателями: *эффективность политики* и *эффективность обучения*. Эффективность политики — это показатель того, насколько хорошо она решает вашу проблему. В идеале вы должны использовать единицы, которые соответствуют проблемной области. Эффективность обучения измеряет, насколько быстро вы можете обучить агента выработке оптимальной политики.

В задаче с *конечным горизонтом*, что означает конечное состояние, эффективность политики наиболее естественно представлена суммой вознаграждения. Это классическое описание оптимальной политики. В задачах с *бесконечным горизонтом* вы можете включить дисконтирование, чтобы предотвратить стремление вознаграждения к *бесконечности*. Дисконтирование также часто используется в конечных задачах, что может привести к несколько иной политике, поэтому вам следует рассмотреть возможность его удаления при вычислении окончательного показателя производительности. Например, ваши заинтересованные стороны, вероятно, не стремятся к дисконтированной прибыли; они хотят говорить о прибыли.

Другой вариант — максимизация и измерение среднего вознаграждения. В модели среднего вознаграждения вы должны вести себя осторожно и убедиться, что среднее значение является подходящей сводной статистикой для вашего распределения вознаграждений. Например, если ваше вознаграждение невелико, то среднее вознаграждение — плохая сводная статистика. Вместо этого вы можете выбрать другую сводную статистику, которая более стабильна, например медиану или некоторый

процентиль. Если ваши награды соответствуют определенному распределению, вы можете использовать параметры этого распределения.

Измерение того, насколько быстро агент может изучить оптимальную политику, также является важным показателем. Он определяет время цикла обратной связи, поэтому, если вы используете алгоритм, который может учиться быстрее, вы можете быстрее выполнять итерацию и улучшать модель. В задачах, когда агенты обучаются онлайн, например, если вы настраиваете политики для конкретных людей, быстрое обучение важно для снижения стоимости упущенной возможности из-за неоптимальных политик.

Самый простой показатель эффективности обучения — это количество времени, которое требуется агенту для достижения оптимальности. Однако оптимальность обычно является асимптотическим результатом, поэтому *скорость сходимости к оптимальности* служит лучшим описанием. Вы можете произвольно определить, что является "ближайшим" для вашей проблемы, но вы должны установить это в соответствии с количеством шума в вашей оценке вознаграждения; возможно, вам потребуется выполнить усреднение, чтобы уменьшить шум.

Вы можете определить время несколькими способами в зависимости от того, что важно для вашей задачи. Для моделирования использование количества шагов в среде является хорошей временно́й мерой, поскольку оно не зависит от помех, а взаимодействие с окружающей средой, как правило, обходится дорого. Реальное время можно использовать, если вас беспокоит время цикла обратной связи и вы можете ограничить количество взаимодействий со средой. Количество шагов обучения модели является интересным временны́м параметром для анализа эффективности обучения модели. Время вычислений может быть важным, если вы хотите повысить эффективность алгоритма.

Ваш выбор временного измерения зависит от того, чего вы пытаетесь достичь. Например, вы можете сначала использовать шаги среды, чтобы повысить эффективность выборки. Затем можете начать смотреть на время на настенных часах, когда будете готовы ускорить тренировку. Наконец, вы можете снизить оставшуюся производительность, посмотрев на время процесса. Обычно все эти параметры используют для того, чтобы получить полную картину производительности.

*Ошибка алгоритма предсказателя* (regret — регрет) используется в онлайновам машинном обучении для измерения общей разницы между вознаграждением, если агент вел себя оптимально в ретроспективе, и фактическим вознаграждением, полученным за все время. Это положительное значение, которое представляет собой область между оптимальным эпизодическим вознаграждением и фактическим, как показано на рис. 10.7. Представьте, что вы управляете солнечной электростанцией. Общее количество энергии, которое вы генерируете в эпизоде, например за 1 день, имеет фиксированный максимум, который зависит от количества солнечного света и коэффициента полезного действия (КПД) солнечных панелей. Но вы тренируете политику изменения шага панелей, чтобы максимизировать выходную мощность. В этом случае регрет будет заключаться в разнице между теоретической максимальной выходной мощностью и результатом действий вашей политики, суммированных за определенный период времени.

Регрет использовался исследователями как математический инструмент при формулировании алгоритмов на основе политики, где политика оптимизируется, чтобы ограничить ошибку алгоритма предсказателя. Это часто принимает форму термина регуляризации или некоторой адаптации вознаграждений для содействия статистически эффективному исследованию (см., например, *разд. "Отложенное Q-обучение" главы 3*). Регрет также важен концептуально, поскольку люди опираются на имеющийся опыт для вывода причинно-следственной связи и ставят под сомнение полезность долгосрочных решений [17].

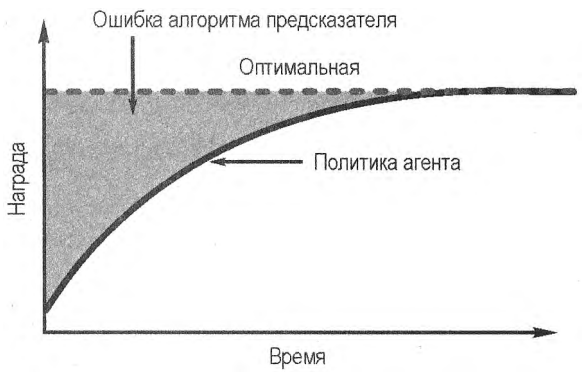



**Рис. 10.7.** Описание того, как вычислить ошибку алгоритма предсказателя, тренируя политику с течением времени

Вы также можете использовать регрет в качестве меры производительности, но принцип выбора оптиимальной политики, даже задним числом, зависит от обстоятельств обучения. Например, вы не сможете получить теоретически оптимальную политику или она может не быть стационарной. Если политики работают в нестационарной среде, то следут попытаться вычислить регрет в пределах небольшого интервала. В более простых задачах вы могли бы задним числом определить, какое действие было лучшим, и использовать его в качестве оптимальной награды. В более сложных задачах вы должны иметь возможность использовать другой базовый алгоритм и проводить с ним все сравнения. Если допустимо, вы можете использовать игру самим с собой, чтобы записать позиции предполагаемого противника, а затем, после завершения игры, посмотреть, есть ли более эффективные ходы (действия), которые вы могли бы сделать. Пока не существует простого единого метода расчета регрета для произвольной задачи.

## Статистические сравнения политик

Всякий раз, когда вы проводите сравнения, рекомендую сначала визуализировать. Статистика — наука неточная: в ней полно предостережений и предположений; запросто может сложиться такая ситуация: вам кажется, что вы поступаете правильно, а кто-то, более склонный к математике, скажет, что полученный вами результат имеет смысл лишь в очень конкретных обстоятельствах. Визуализировать производительность намного проще, потому что это интуитивно понятно. Вы мо-

жете использовать визуальные приближения и модели, которым вы научились на протяжении всей своей жизни. Например, подобрать модель линейной регрессии на глаз, даже с учетом шума и выбросов, намного проще, чем предполагает математика. Точно так же сравнить производительность двух алгоритмов так же просто, как построить две метрики друг против друга и решить, какая из них лучше.

Однако бывают случаи, когда вам действительно нужны статистические доказательства того, что один алгоритм лучше другого. Например, если вы хотите автоматизировать поиск лучшей модели, нет необходимости смотреть на тысячи графиков. Или если вы хотите получить ответ "да/нет" (с учетом оговорок!). Вот список возможных статистических тестов с краткими заметками об их предположениях при сравнении двух распределений.

- *t-Тест Стьюдента.*

  Предполагает нормальное распределение и равные дисперсии.

- *T-тест Уэлча.*

  Аналогичен тесту Стьюдента, но ослабляет предположение о равной дисперсии.

- *Критерий Уилкоксона — Манна — Уитни.*

  Сравнивает медианы, является непараметрическим, поэтому нет предположений о формах распределения. Но предполагается, что распределения непрерывны, имеют одинаковую форму и ширину.

- *Ранговый t-тест.*

  Сравнивает медианы, ранжирует, а затем вводит в *t*-критерий. Результаты, по сути, такие же, как в критерии Уилкоксона — Манна — Уитни.

- *Тест доверительного интервала бутстрэппирования.*

  Произвольная выборка из распределений, измерение разницы в процентилях и многократное повторение. Требуется очень большой размер выборки, примерно больше $10^3$. Никаких других предположений.

- *Проверка перестановки.*

  Произвольно выбирает данные из распределений, случайным образом назначает их гипотезам, измеряет разницу в процентилях и повторяет много раз. Алгоритм подобен тесту доверительного интервала бутстрэппинга, но вместо этого проверяет, совпадают ли разницы с некоторым уровнем достоверности. Опять же, требуется очень большой размер выборки.

Как видно, существует множество различных оговорок, которые позволяют с легкостью случайно использовать неправильный тест. Результаты не покажут вам, что именно вы сделали не так. Поэтому очень важно со всей серьезностью подойти к выполнению этих тестов. Если результаты оказались бессмысленными, возможно, вы ошиблись.

В чем заключаются проблемы с этими статистическими допущениями?

- RL часто дает результаты, которые не являются нормально распределенными. Например, агент может застрять в лабиринте 1 раз из 100 и вызвать выброс.

- Награды могут быть урезаны. Например, может быть жесткое максимальное вознаграждение, которое приводит к прерыванию в распределении результатов.
- Стандартные отклонения прогонов не равны у разных алгоритмов и могут даже не быть стационарными. Например, различные алгоритмы могут использовать разные методы исследования, что приводит к отличающимся вариациям результатов. И в рамках оценки одного алгоритма может произойти какая-то случайность, например птица, летящая в руку робота.

Колас и соавт. написали отличную статью о статистическом тестировании специально для сравнения алгоритмов RL. Они сравнивают производительность TD3 и SAC в среде Half-Cheetah-v2 и обнаруживают, что им необходимо повторять каждое обучение по крайней мере 15 раз, чтобы получить статистически значимые доказательства того, что SAC лучше, чем TD3 (в этой среде). Они предполагают, что для этой среды вознаграждение обычно в некоторой степени имеет нормальное распределение, поэтому *t*-тест Уэлча, как правило, является наиболее надежным для небольшого количества выборок. Непараметрические тесты имеют минимум предположений, но требуют больших размеров выборки, чтобы можно было хоть как-то предположить, в соответствии с каким законом данные распределены [18]. У них также есть полезный *репозиторий*, который вы можете использовать в своих экспериментах.

Вы можете спросить: "Что он имеет в виду под повторением?" Это хороший вопрос. Если вы прочитаете достаточно статей по алгоритмам RL, вы в конечном счете увидите общую тему оценки. Исследователи повторяют эксперимент 5 или 10 раз, а затем строят кривые обучения, где один эксперимент представляет собой полное переобучение алгоритма в одной среде. При каждом повторении они изменяют *случайное начальное число* различных генераторов случайных чисел.

Звучит, конечно, просто, но на практике я обнаружил, что это довольно сложно. Если вы используете одно и то же начальное число в одном и том же алгоритме, теоретически оно должно давать одинаковые результаты. Однако часто это не так. Обычно я обнаруживаю, что где-то внутри кодовой базы есть еще один случайный генератор, который я не установил. Если вы обучаете алгоритм с использованием типичного набора библиотек RL, тогда вам необходимо установить начальное число в генераторах случайных чисел для Python, Numpy, Gym, вашей инфраструктуры глубокого обучения (возможно, в нескольких местах), в структуре RL и в любом другом фреймворке, который вы можете использовать, например в буфере воспроизведения. Поэтому сначала убедитесь, что вы можете получить точно такой же результат за два прогона. Затем вы можете повторить обучение несколько раз с разными значениями случайного начального числа. Это дает ряд кривых производительности, которые (будем надеяться) нормально распределены.

Проблема в том, что исследователи, как правило, используют небольшое фиксированное количество повторов, потому что обучение стоит дорого. Если между двумя распределениями существует большое разделение, то хватит небольшой выборки, достаточной для уверенности в среднем значении и стандартном отклонении. Но когда разница в производительности намного меньше, вам понадобится гораздо

больший размер выборки, чтобы быть уверенным, что производительность отличается. Еще одна статья Коласа и соавт. предлагает вам провести пилотное исследование для оценки стандартного отклонения алгоритма, а затем использовать его для оценки окончательного количества испытаний, необходимых для доказательства того, что он превосходит другой алгоритм, до определенного уровня значимости [19].

Наконец, статистические показатели эффективности — это один из способов количественной оценки устойчивости политики. Стандартное отклонение полученного вознаграждения является хорошим показателем того, выдержит ли ваша политика естественный дрейф наблюдений, который так часто происходит в реальной жизни. Но я бы порекомендовал сделать еще один шаг, активно изменяя состояния, модели и действия, чтобы увидеть, насколько на самом деле надежна политика. Здесь полезны состязательные методы, потому что, если вы не можете сломать свою модель, это снижает вероятность того, что реальная жизнь сломает и вашу модель — жизнь в конечном счете ломает все [20].

## Показатели производительности алгоритма

Я бы предположил, что в большинстве промышленных задач у вас есть одна или небольшое количество задач, на которые нужно ориентироваться, и, следовательно, небольшое количество сред. Однако, если вы больше вовлечены в чистое исследование или пытаетесь адаптировать свою реализацию для обобщения различных задач, вам нужно рассмотреть более абстрактные меры производительности, которые объясняют, насколько хорошо эта реализация работает в задачах, специфичных для RL.

На данный момент одним из наиболее полезных решений является *поведенческий набор для обучения с подкреплением*[11] (behavior suite) Осбанда и соавт. Они описывают семь фундаментальных проблем RL: передачу ответственности, базовые поведения (тест компетенции, например, задачи $N$-рукого бандита), масштабируемость, шум, память, обобщение и исследование. Агент тестируется в специально подобранном наборе сред, соответствующих задаче. Затем агент получает окончательную совокупную оценку за каждую задачу, которая может быть представлена на радарном графике, как на рис. 10.8 [21].

## Измерения производительности для конкретных задач

Существует также ряд тестов, доступных для специфических задач, или для задач, которые имеют особенно сложную проблему. Например, тесты для *автономного RL*[12], *робототехники*[13] и *безопасности*[14] [22–24]. И еще много отличных тестов.

---

[11] См. **https://oreil.ly/HLb_0**.

[12] См. **https://oreil.ly/i6csS**.

[13] См. **https://oreil.ly/N27IC**.

[14] См. **https://oreil.ly/008IE**.

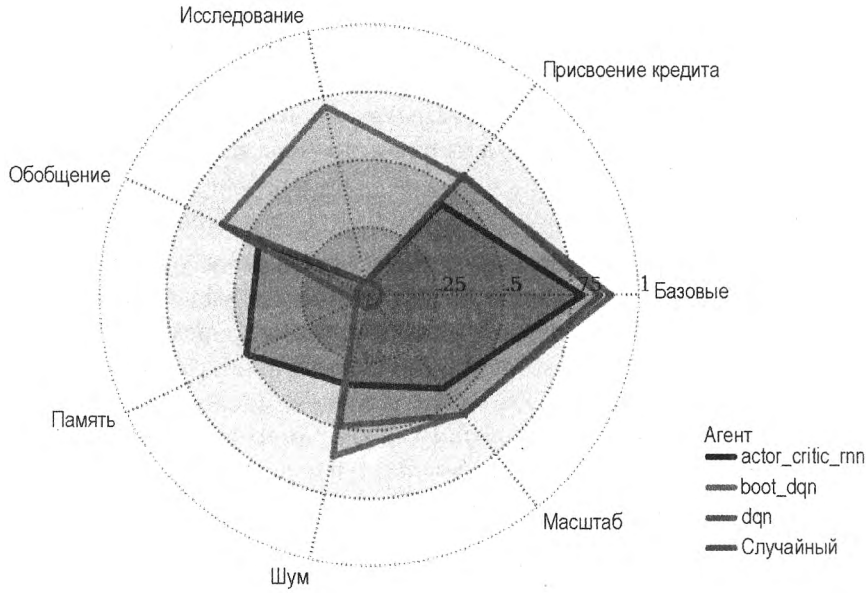



**Рис. 10.8.** Сравнение нескольких алгоритмов с использованием набора поведений. Каждая ось представляет разные задачи RL. Обратите внимание, что случайный агент плохо справляется со всеми задачами, а стандартный DQN хорошо справляется со многими, за исключением исследования и памяти. Используется по лицензии Apache 2.0

Они могут быть полезны, если вы знаете, что у вас также есть аналогичная общая задача в вашей области, и можете быть уверены, что улучшения в тестах приводят к повышению производительности в реальном мире.

## Объяснимость

Новым этапом в процессе обработки данных является *объяснимость*, часто называемая *объяснимым искусственным интеллектом* (explainable artificial intelligence, XAI), и вы можете использовать методы, чтобы попытаться описать, как и почему модель приняла решение. Некоторые алгоритмы, такие как модели на основе деревьев, обладают неотъемлемой способностью давать объяснения, тогда как другие, такие как стандартные нейронные сети, должны полагаться на подходы *черного ящика*, когда вы будете исследовать известные входные данные и наблюдать, что происходит на выходе.

Объяснимость важна по нескольким причинам. Первая — доверие; вы не должны запускать в производство то, чему не доверяете. В программной инженерии это доверие устанавливается посредством тестов, но сложность моделей и RL означает, что "охват" всегда будет низким. Другая причина может заключаться в том, что существует нормативная или коммерческая причина для объяснения решений, таких как общий регламент ЕС по защите данных или функция объяснения рекомендаций Facebook. Возможность объяснить действие или решение также может помочь при отладке и разработке.

Объяснение в RL сложнее, чем в машинном обучении, потому что агенты выбирают последовательные действия, эффекты которых взаимодействуют и складываются в течение длительных периодов времени. Некоторые алгоритмы RL обладают внутренней способностью объяснять. Иерархический RL, например, разбивает политики на подполитики, которые в совокупности описывают поведение высокого уровня. Но большинство исследований сосредоточено на объяснении после обучения с помощью черного ящика, потому что такой подход более широко применим.

Методы черного ящика обычно работают либо на глобальном, либо на локальном уровне. Глобальные методы пытаются вывести и описать поведение во всех состояниях, тогда как локальные методы могут объяснить только определенные решения.

Глобальные методы, как правило, состоят из обучения других, более простых моделей, которые по своей сути объяснимы, например древовидных или линейных моделей. Скажем, *линейная модель U-деревьев* использует древовидную структуру для представления функции ценности с линейными аппроксиматорами в листе каждого дерева для улучшения обобщения. Из этой модели вы можете извлечь значимость функций и ориентировочные действия для всего пространства состояний [25].

Локальные методы заключают в себе две основные идеи. Слово "локальный" обычно означает глобальный метод, оцениваемый в небольшой области пространства состояний. Например, Юн и соавт. используют RL для управления процессом, который обучает локальной апрокксимации [26]. Другие идеи расширяют прогностическую природу глобальных методов, рассматривая *контрфактическое*, противоположное действие или то, что могло произойти. Мадумал и соавт. добиваются этого путем обучения модели для представления причинно-следственного графа, который аппроксимирует динамику перехода среды. Звучит многообещающе, но работает только с дискретными состояниями [27].

Подобно объяснению решений, также может быть полезно указать, какие части состояния влияют на решение. Для проблем с визуальными наблюдениями или состояний, которые имеют некоторое представление о положении или смежности, вы можете использовать *карты значимости*, чтобы выделить регионы, которые важны для выбора действий [28].

## Выводы оценки

Оценка — неотъемлемая часть жизни, от поясов карате до кулинарных критиков. В промышленных приложениях оценка вашей работы помогает вам решить, когда вы закончите. В программной инженерии это обычно происходит после прохождения приемочных испытаний. В машинном обучении приемлемая производительность часто определяется оценочной метрикой. Но в RL из-за стратегического характера принимаемых решений очень сложно определить, когда прибыли достаточно. Вы всегда можете получить больше прибыли, вы всегда можете спасти больше жизней.

Как правило, эффективность агента определяется вознаграждением, полученным политикой, и тем, как быстро он его получил. Но есть ряд других мер, которые вам

также следует рассмотреть, если вы хотите обучить и использовать стабильную, надежную политику в производственной ситуации. Это также включает способность объяснять решения политики, что может быть жизненно важным для приложений с высокой степенью воздействия.

# Развертывание

Под *развертыванием* я подразумеваю этап, на котором вы производите или вводите в эксплуатацию свое приложение. Проекты достигают точки, когда все соглашаются, что они (проекты) жизнеспособны и ценны, но, чтобы оставаться жизнеспособными и ценными, они должны быть надежными. В этом разделе рассказывается, как вывести ваше многообещающее подтверждение концепции RL в высшую лигу, чтобы вы могли служить реальным пользователям.

Сложность заключается в том, что ранее не проводилось большого количества работ по запуску приложений RL в оперативном режиме. Многое из этого нужно экстраполировать на основе запущенного программного обеспечения и приложений машинного обучения, моего опыта работы с Winder Research и многих светил, написавших о своем собственном опыте.

## Цели

Прежде чем я углублюсь в детали развертывания, думаю, что важно рассмотреть цели этого этапа процесса. Они помогут вам сосредоточиться на вашей ситуации и решить, что для вас важно, потому что ни одна идея или архитектура не подходит для всех ситуаций.

### Цели на разных этапах развития

Во-первых, подумайте, как меняются приоритеты развертывания в зависимости от того, на каком этапе разработки вы находитесь. Существуют три этапа, на которых вам необходимо развернуть агента: во время разработки, усиления и производства.

Во время разработки важно иметь архитектуру, позволяющую быстро тестировать новые идеи; гибкость и скорость превосходят все остальные потребности. Это связано с тем, что на ранних этапах проекта важно доказать его потенциальную жизнеспособность и ценность. Не важно быть бесконечно масштабируемым или точно воспроизводимым; это придет позже. Наибольшие выгоды достигаются за счет быстрого и разнообразного экспериментирования, поэтому наибольшие затраты на данный момент — это длина цикла обратной связи.

После того как проект доказал свою жизнеспособность и ценность, он проходит этап упрочнения. Это включает повышение устойчивости процессов, связанных с поддержанием политики. Например, автономные политики или вспомогательные модели должны быть обучены, что может потребовать больших объемов вычислительных ресурсов, а происхождение кодовой базы или модели жизненно важно для того, чтобы вы знали, что будет работать в производственной среде.

Наконец, политики и модели развертываются в производственной среде и используются реальными людьми или процессами. Поскольку RL оптимизирует последовательность действий, маловероятно, что вы сможете переобучить свою политику, периодически группируя данные, что является преобладающим методом, используемым в производственных системах машинного обучения, поэтому необходимо соблюдать осторожность, чтобы сохранить состояние политики. Этот этап поощряет разработку и внедрение сложных механизмов мониторинга и обратной связи, т. к. политики очень легко расходятся (или, по крайней мере, не сходятся).

При развертывании решения RL в продукте или услуге ценность выполнения задачи меняется. На раннем этапе самый быстрый способ извлечь пользу — это быстрые эксперименты и исследования; позже надежность, производительность и отказоустойчивость становятся более важными. Это приводит к целому ряду передовых практик, которые важны в разное время.

## Лучшие практики

Операции машинного обучения (machine learning operations, MLOps) — это новое описание передовых практик и соответствующих инструментов, необходимых для запуска моделей производственного качества. Необходимость в этом возникла, потому что модели машинного обучения имеют разные операционные требования по сравнению с традиционными программными проектами, но, по сути, основные потребности точно такие же. Мне нравится объединять эти потребности в три темы: надежность, масштабируемость и гибкость.

*Надежность* лежит в основе современной разработки программного обеспечения. По этой теме написаны тысячи книг, и эти идеи, например тестирование, непрерывная интеграция и доставка, архитектура и дизайн, напрямую применимы к машинному обучению и RL. При переходе к машинному обучению такие идеи, как построение конвейеров данных, происхождение и воспроизводимость, направлены на укрепление и дальнейшую гарантию того, что модели будут продолжать работать в будущем. RL создает собственные задачи, и я ожидаю, что со временем будут выработаны и рекомендованы новые практики использования.

Например, недавно я встретился со старым коллегой и спросил его о некоторых предыдущих проектах, над которыми мы вместе работали. Примерно в 2012 г. я разработал несколько алгоритмов для мониторинга трафика и заторов, и они были развернуты на некоторых главных дорогах Великобритании на специальном оборудовании. Я узнал, что оборудование дважды выходило из строя, но мои алгоритмы (на Java!) все еще работали, почти 10 лет спустя. Это связано не с моими алгоритмами, а с опытом моих коллег в области разработки программного обеспечения и их приверженностью к надежному программному обеспечению.

*Масштабируемость* особенно важна как для машинного обучения, так и для RL, возможно, даже больше, чем для типичного программного обеспечения, потому что им требуется очень много лошадиных сил для выполнения своей работы. Если учесть все когда-либо написанные программы, подавляющее большинство этих приложений, вероятно, будет работать на ресурсах, предоставляемых стандартным ноутбуком. Лишь в редких случаях вам нужно создавать что-то столь же масшта-

бируемое, как глобальная система обмена сообщениями или банковская система. Но в машинном обучении и RL почти каждый продукт, когда-либо производимый, требует гораздо большего, чем может предоставить ноутбук. Я утверждаю, что средние вычислительные требования для продукта RL на несколько порядков больше, чем для программного продукта. Я намеренно говорю расплывчато, потому что у меня нет неопровержимых фактов, подтверждающих это. Правда, я думаю, что моя гипотеза верна, и именно поэтому масштабируемость является важным фактором при развертывании RL.

Я считаю, что *гибкость* — одна из самых недооцененных передовых практик. В промышленности надежное программное обеспечение было темой первого десятилетия XXI века, а архитектура программного обеспечения, разработка через тестирование и непрерывная интеграция были весьма обсуждаемыми темами на конференциях по всему миру. Во втором десятилетии масштабируемость была "золотым стандартом" и привела к важным изменениям, таким как внедрение облачных технологий и последующая потребность в оркестрации. Думаю, что третье десятилетие (я игнорирую конкретные темы, такие как искусственный интеллект, RL и машинное обучение) будет сосредоточено на гибкости просто потому, что я вижу тенденцию в отрасли к гиперответственности, когда самые ценные инженеры выделяются своей способностью выполнять все роли, ожидаемые в корпоративной среде, от развития бизнеса до операций. Сфера DevOps может только расширяться; она никогда не сможет вернуться к двум отдельным ролям.

Аналогичный аргумент можно привести и в отношении темпов сегодняшних изменений. Инженеры постоянно создают новые технологии, и трудно предсказать, какие из них выдержат испытание временем. Но я обнаружил, что, когда инструменты пытаются навязать фиксированный, негибкий или проприетарный способ работы, они быстро становятся громоздкими для работы и устаревают. Нет, гибкость — это единственный путь к перспективным системам, процессам и решениям.

## Иерархия потребностей

Иерархия потребностей Маслоу была ранней психологической моделью, предлагавшей человеческие потребности изображать в виде иерархической модели; чтобы чувствовать себя в безопасности, вам нужен доступ к воздуху и воде, чтобы водить дружбу, вам нужно быть в безопасности, и т. д. У RL аналогичная иерархия потребностей.

Для того чтобы иметь жизнеспособную задачу RL, вам нужна среда, в которой вы можете наблюдать, действовать и получать вознаграждение. Вам нужны вычислительные способности, чтобы принимать решения о действиях на основании наблюдения. Эти вещи необходимы, и если у вас их нет, вы не сможете выполнять RL.

Выше этого расположены вещи, которые являются ценными, но не совсем необходимыми. Например, конвейеры для унифицированного и надежного преобразования данных предотвратят ошибки данных. Возможность точно указать, что работает в производстве — происхождение — очень полезна с операционной точки зрения.

Слой выше этого представляет собой оптимальное состояние, которое, вероятно, во многих случаях является скорее концептуальным, чем практическим, потому что решение других проблем или разработка новых функций могут быть более ценными в краткосрочной перспективе. Здесь вы можете подумать о полной автоматизации сквозного процесса того, чем вы занимаетесь, или о включении сложного механизма обратной связи, который позволит вашему продукту автоматически улучшаться.

В табл. 10.1 представлены некоторые примеры проблем реализации и развертывания, с которыми вы можете столкнуться. Я хочу подчеркнуть, что у них есть ряд решений с вариациями в степени автоматизации. Но ценность каждого уровня автоматизации зависит от оценки вами или вашей заинтересованной стороной других насущных потребностей. Просто потому, что есть более сложное решение, еще не значит, что имеющееся у вас уже недостаточно хорошо. Сначала усвойте основы.

***Таблица 10.1.*** *Пример иерархии потребностей для различных задач реализации*

| Потребности | Инфраструктура | Репрезентация состояния | Среда | Обучение | Улучшение алгоритма |
|---|---|---|---|---|---|
| Оптимальные | Самовосстановление | Постоянное обучение | Комбинация | Мета | Автоматизированное |
| Цельные | Автоматическое развертывание | Уменьшение размерности на основе модели | Реальная | Офлайн | Перебор |
| Необходимые | Ручное развертывание | Фиксированное отображение | Модель | Онлайн | Ручное |

**Примечание** нижняя строка представляет основные требования, средняя строка представляет собой некоторую степень автоматизации, верхняя строка — современное состояние

## Архитектура

Большого количества исследований общих архитектур RL до настоящего времени не было проведено, и я могу придумать несколько причин, чтобы объяснить это. Во-первых, RL все еще не повсеместно признано полезным инструментом для решения промышленных проблем. Надеюсь, эта книга начнет изменять ситуацию. Это означает, что не существует достаточного совокупного опыта разработки таких систем, и поэтому оно не было распространено среди более широкой аудитории. Во-вторых, исследователи, особенно ученые-исследователи, как правило, больше сосредоточены на новаторских исследованиях, поэтому вы найдете немного статей по данной теме. В-третьих, я не думаю, что существует универсальный подход. Сравните, например, различия между одноагентным и мультиагентным RL. И наконец, архитектура развертывания — это в основном проблема разработки программного обеспечения. Конечно, существуют специфические потребности RL, но я думаю, что типичные рекомендации по архитектуре программного обеспечения хорошо подходят.

С учетом сказанного, есть некоторая литература, доступная из области контекстных бандитов. Это, как правило, неприменимо, но поскольку промышленность быстрее внедрила этот метод, многие уроки уже были извлечены. Две самые большие проблемы, которые приводят к влиянию на архитектуру, заключаются в том, что агенты могут наблюдать только частичную обратную связь и что вознаграждения часто откладываются или в худшем случае редки [29].

- Машинное обучение считается более простым, потому что обратная связь, или основополагающая истина, является полной. Метка представляет собой единственный правильный ответ (или ответы, если правильных несколько). RL наблюдает только за результатом одного действия; ничему не научиться из неизведанных действий (которые, возможно, бесконечны).
- Награды, наблюдаемые агентом, часто задерживаются, причем задержка колеблется от миллисекунд в торгах до дней, когда в них участвуют люди. Даже в этом случае награда может быть скудной, потому что последовательность действий в течение некоторого времени не приводит к реальной награде, например к покупке.

Некоторые проблемы связаны с предыдущей работой в области машинного обучения, но подчеркиваются RL. Например, онлайн-обучение или непрерывное обучение важно в некоторых дисциплинах машинного обучения, но RL выводит это на совершенно новый уровень, поскольку решения имеют далеко идущие последствия, а политики нестационарны. Даже если вы включаете пакетное RL, важно иметь цикл обучения с малой задержкой. Воспроизводимость также становится большой проблемой, поскольку и политика, и окружающая среда постоянно меняются.

Все эти аспекты приводят к следующему диапазону архитектурных абстракций, которые могут быть полезны в вашей проблеме.

- *Обслуживающий интерфейс.*

  Подобно персонализированному помощнику по покупкам, одна абстракция должна содержать возможность предлагать действия и внедрять исследования. Это должно реализовать интерфейс марковского процесса принятия решений и взаимодействовать с более широкой системой или средой. Это как точка входа, так и точка выхода из системы и представляет собой отличное место для отделения RL от более широких развертываний.

- *Журнал событий.*

  Как и в очереди шины сообщений, жизненно важно иметь возможность сохранять состояние по нескольким причинам. Во-первых, для аварийного восстановления вам может потребоваться воспроизвести взаимодействия, чтобы обновить политику до последнего известного состояния. Во-вторых, журнал событий можно использовать как часть более широкого пакета или трансфертного обучения для улучшения моделей. В-третьих, он действует как очередь для буферизации потенциально медленных онлайн-обновлений.

- *Сервер параметров.*

  Место для хранения параметров модели, например базы данных, появляется в мультиагентном RL, но это полезно в ситуациях, когда вам также необходимо реплицировать политики одного агента, чтобы справиться с большей нагрузкой.

- *Серверы политик.*

  Поскольку алгоритмы RL обычно настраиваются для решения конкретной задачи, имеет смысл создать абстракцию, инкапсулирующую реализацию. В более простых, более статических задачах вы можете заложить параметры модели прямо в инкапсуляции, в противном случае вы можете использовать сервер параметров. Вы можете реплицировать серверы политик для увеличения емкости.

- *Буферы воспроизведения.*

  Многие алгоритмы используют буферы воспроизведения для повышения эффективности выборки. Это может быть расширение журнала или выделенная база данных.

- *Акторы и критики.*

  Возможно, вы захотите разделить сервер политики на более мелкие компоненты.

- *Обучение.*

  В идеале вы должны использовать тот же код политики для обучения или тренировки, что и для обслуживания. Это гарантирует отсутствие расхождений между обучением и обслуживанием.

- *Увеличение данных.*

  У вас могут быть части вашего алгоритма, которые полагаются на расширенные данные; алгоритм представления состояния, например. Они должны существовать как традиционно обслуживаемые модели ML с соответствующими конвейерами непрерывной интеграции и обучения.

Все абстракции могут появиться в вашем архитектурном проекте, либо ни одна из них не попадет в ваш арсенал. К сожалению, они очень специфичны для предметной области и приложений, поэтому нет смысла применять эти абстракции. Кроме того, большее количество абстракций обычно увеличивает нагрузку на обслуживание; в конце концов, проще работать с единой кодовой базой. Вместо этого предоставьте инструменты и технологии, чтобы задействовать эти архитектурные возможности для каждого приложения.

## Вспомогательные инструменты

Я не решаюсь подробно рассказывать об инструментах из-за быстрых изменений в отрасли; компонент, который я описываю здесь, может быть устаревшим к тому времени, когда эта книга выйдет в печать. Вместо этого я абстрактно расскажу о функциональности групп инструментов и приведу несколько (исторических) примеров. Вернитесь к *разд. "Фреймворки RL" данной главы* для обсуждения специфических инструментов RL.

## Разработка против покупки

Думаю, что традиционные рекомендации по разработке против покупки верны и в RL. Если интересующий инструмент представляет собой основную компетенцию или ценностное предложение, вам следует создать его таким образом, чтобы сохранить интеллектуальную собственность, иметь гораздо большую глубину и полный контроль над настройкой. Другой способ подумать об этом — спросить себя: за что вам платят ваши клиенты?

Но если этот конкретный инструмент не отражает вашу основную компетенцию или если он не обеспечивает конкурентного преимущества, вам следует подумать о покупке или использовании готовых компонентов. Также существует золотая середина, когда вы можете взять за основу проект с открытым исходным кодом, чтобы завершить его на 80%, и адаптировать, или закрепить компоненты, чтобы получить остальное.

## Мониторинг

Когда ваш продукт поступит в производство, вам следует инвестировать в мониторинг. Так вы без посторонней помощи получите заблаговременное предупреждение о проблемах, которые влияют на доверие к алгоритму. А доверие важно, потому что, в отличие от традиционного программного обеспечения, которое просто перестает работать, машинное обучение и RL изучаются так же, как гонщик критикует управление своего автомобиль. Если пользователи получают странный или неожиданный результат, это может подорвать их уверенность. Иногда лучше смоделировать неисправный продукт, чем давать плохие прогнозы или рекомендации.

Однажды я общался с техническим руководителем игровой компании, и он сказал, что пользователи негативно реагируют, если они играют против автоматизированного агента, который не соответствует их социальным нормам. Например, если агент должен был воспользоваться возможностью в игре, которая оставила пользователя слабым, пользователь считал это плохим спортивным мастерством и прекращал играть. И часто исследователям приходилось перенастраивать своих агентов намеренно принимать неверные решения, чтобы пользователь чувствовал, что у него есть шанс на победу. RL может помочь в этом отношении, оптимизируя то, что важно, например удержание пользователей, а не чистую производительность.

В *разд. "Оценка" ранее в данной главе* рассказывается о ряде специфичных для RL показателях производительности, которые вы можете реализовать в решении для мониторинга. Это в дополнение ко всей стандартной статистике машинного обучения, которую вы можете отслеживать, и низкоуровневым статистическим показателям базовых данных. Конечно, мониторинг на уровне программного обеспечения жизненно важен. Это формирует иерархию показателей, которые вы должны рассмотреть как возможные к внедрению, чтобы получить оперативную информацию и выявить проблемы до того, как это сделают ваши пользователи.

## Регистрация и отслеживание

Журнал — важный инструмент отладки. Как правило, он наиболее важен во время разработки для того, чтобы помочь вам настроить и отладить ваших агентов, а после сбоя — выполнить посмертный анализ. Ошибки в традиционном программном обеспечении появляются, когда состояние программы изменяется неожиданным образом. Поэтому регистрация этих изменений состояния — хороший способ отслеживать проблемы, когда они действительно возникают. Однако в RL каждое взаимодействие с агентом — это изменение состояния, а значит, регистрация изменений может быстро превратиться в огромное количество информации. Вот почему я рекомендую иметь специальный журнал для изменений марковских состояний, чтобы освободить ваше программное обеспечение от этого бремени.

Одна из наиболее распространенных ошибок в RL связана с различиями во времени выполнения и ожидаемой форме данных. Рекомендуется регистрировать эти формы, возможно, в низкоуровневом журнале отладки, чтобы вы могли быстро найти, где возникают расхождения. Некоторые разработчики доходят до того, что останавливают код, как только обнаруживается исключение. Следующие по распространенности ошибки возникают из данных, но я бы не стал сохранять данные непосредственно в текстовом журнале. Слишком много данных. Обобщите это и сохраните как метрику мониторинга или сбросьте в специальный каталог для автономной проверки.

Отслеживание решений по журналу может быть довольно сложным из-за потенциальной длины траекторий. Плохая рекомендация не обязательно вызвана предыдущим действием. Действие, совершенное давным-давно, могло привести к такому плохому состоянию. Эта проблема очень похожа на системы трассировки, используемые при развертывании микросервисов для отслеживания активности через границы сервисов. Эти системы используют идентификатор корреляции (уникальный ключ), который передается по всем вызовам; этот ключ используется для связи действий. Аналогичный подход можно использовать для сопоставления траекторий.

Могут быть и другие источники изменения состояния: например, изменения в политике, отзывы пользователей или изменения в инфраструктуре. Постарайтесь запечатлеть все это, чтобы иметь возможность обрисовать картину, когда что-то пойдет не так. Если у вас все же есть сбои, и вы не знаете почему, обязательно добавьте их в соответствующий журнал.

## Непрерывная интеграция и непрерывная доставка

*Непрерывная интеграция* (continuous integration, CI) — это стандартная практика разработки программного обеспечения для создания автоматизированных конвейеров для развертывания ваших решений. Магистрали проводят испытания, чтобы поддерживать качество вашего продукта. Идея *непрерывной доставки* (continuous delivery, CD) заключается в том, что, если вы достаточно уверены в своих критериях качества, то можете выпускать новую версию своего программного продукта при каждой фиксации.

Комплекс "непрерывная интеграция/непрерывная доставка" (CI/CD) — важный и необходимый инструмент для обеспечения развертывания на производстве, но он применим только с автономными статическими артефактами, такими как пакеты программного обеспечения. Он не может предоставить никаких гарантий, что данные, поступающие к вашему агенту, через него или исходящие от него, действительны.

Вы можете заняться внутренней частью с помощью тестирования. Вы можете отслеживать и предупреждать об исходящих данных. Но даже если вы сможете отслеживать поступающие данные, это все равно может привести к катастрофическому сбою.

Для того чтобы справиться с проблемами с входящими данными, вам необходимо протестировать их. Я называю это "модульным тестированием данных". Идея заключается в том, что вы указываете, как должны выглядеть входящие данные в форме схемы, и проверяете их соответствие. Если соответствие не прослеживается, вы можете исправить это или сообщить пользователю, что с его данными что-то не так. Вы должны принять архитектурное решение о том, где проводить это тестирование. Самое простое место — обслуживающий агент, но это может привести к раздуванию кода. У вас может быть централизованный дозорный, который также управляет этим процессом.

## Отслеживание экспериментов

Во время разработки вы будете проводить большое количество экспериментов. Даже после этого этапа вам необходимо отслеживать и контролировать производительность вашего агента или модели по мере сбора новых наблюдений. Единственный способ делать это последовательно — отслеживать показатели производительности.

Основные цели аналогичны мониторингу: поддерживать качество и повышать производительность. Но мониторинг предоставляет информацию только о текущем экземпляре агента. Отслеживание экспериментов призвано обеспечить долгосрочное целостное представление с учетом возможных улучшений.

Идея состоит в том, что при каждом запуске агента вы должны регистрировать и отслеживать производительность. Со временем вы начнете замечать тенденции в производительности агентов по мере развития данных. И вы можете сравнить и сопоставить производительность с другими экспериментами.

Но основная причина, почему это важно, заключается в том, что обучение иногда может длиться несколько дней, поэтому для поддержания эффективности вы должны проводить эксперименты параллельно. Например, представьте, что вы разработали агента по рекомендациям и выдвинули гипотезу о том, что пожилым людям легче изучать политику, потому что они знают, чего хотят. Итак, вы сегментировали данные обучения по возрасту и начали обучение на нескольких разных наборах данных. Вместо того, чтобы ожидать завершения каждого из них, вы должны проводить все эксперименты параллельно и отслеживать их результаты. Да, ваша эффективность снижается из-за времени, необходимого для получения обратной

связи по вашей гипотезе, но, по крайней мере, вы получите все результаты одновременно. По этой причине вам нужно отслеживать эти эксперименты с помощью внешней системы, чтобы вы могли эффективно проводить время в другом месте или взять выходной — это ваше дело.

Фактически вы хотите отслеживать показатели, то, что мониторите, и хранить их в базе данных. TensorBoard — одно из распространенных решений, которое хорошо подходит для простых экспериментов, но с трудом масштабируется. Даже для отдельных людей бывает сложно организовать разные эксперименты. Еще хуже, когда несколько человек используют один и тот же аппаратный экземпляр оборудования. Доступны проприетарные системы, но многие из них ориентированы на традиционное машинное обучение, а не на RL, и, поскольку они проприетарные, их сложно улучшить или дополнить.

## Настройка гиперпараметров

Предпосылка настройки гиперпараметров довольно проста: выберите значения для гиперпараметров, обучите и оцените своего агента, затем повторите для других значений гиперпараметров. Опять же, основная проблема — это количество времени, которое требуется на то, чтобы сделать это методом перебора, когда вы исчерпывающе тестируете каждую перестановку, из-за времени, необходимого для выполнения одного запуска.

Вероятно, что понимание вашей задачи, которое вы можете представить, приведет к тщательно отобранному подмножеству параметров, для которого может быть достаточно перебора. Но есть более новые подходы, которые пытаются построить модель производительности в сравнении со значениями гиперпараметров, чтобы быстрее перейти к оптимальной модели. Они также могут сократить продолжительность тренировок, если увидят, что итоговая производительность окажется неоптимальной. Я использовал Optuna[15] для этой цели, и она хорошо работает, когда ваши гиперпараметры постоянны, а показатель производительности надежен. Вы можете столкнуться с проблемами, например, когда гиперпараметры замедляют обучение, а фреймворк считает результат неоптимальным, но в долгосрочной перспективе это может привести к повышению производительности.

Наличие хорошей настройки инфраструктуры с автоматическим масштабированием важно для сокращения времени, необходимого для проведения экспериментов с гиперпараметрами.

## Развертывание нескольких агентов

Архитектурный шаблон микросервиса хорошо подходит для машинного обучения и RL. Запуск нескольких агентов в производственной среде стал проще, чем раньше, благодаря усовершенствованным инструментам, позволяющим создавать расширенные развертывания. Вот список возможных стратегий развертывания.

[15] См. **https://optuna.org/**.

- *Канареечный тест.*

  Перенаправьте определенную часть пользователей к новому агенту, изменяющемуся со временем. Убедитесь, что у вас есть возможность обеспечить *привязку*, чтобы траектория пользователя определялась одним и тем же агентом. Обычно это реализуется с использованием тех же идей, что и трассировка.

- *Затенение.*

  Запуск других агентов, скрытых в фоновом режиме, на реальных данных, но без предоставления результатов пользователям. Это может быть сложно проверить, потому что скрытому агенту не разрешено исследовать.

- *A/B-тестирование и RL.*

  Да, вы можете использовать бандитов или RL для оптимального выбора агентов RL. Опять же, убедитесь в привязке [30].

- *Ансамбли.*

  Эта стратегия с большей вероятностью попадет под знамя алгоритмической разработки, но для принятия решения можно использовать ансамбль RL-агентов [31]. Теория предполагает, что результирующее действие должно быть более надежным, если агенты разнообразны.

## Развертывание политик

Внедрение новой политики сопряжено со значительным риском, поскольку трудно гарантировать, что она станет оптимальной или не будет отличаться при онлайн-обучении. Мониторинг и оповещение помогают, но еще один способ — взглянуть на эту проблему с точки зрения *эффективности развертывания*.

Подобно измерению эффективности выборки, вы можете подсчитать, сколько раз во время обучения необходимо обновить политику, чтобы она стала оптимальной. Политики, которые необходимо обновлять реже, скорее всего, будут более надежными в производственной среде. Мацусима и соавт. исследуют это в своей работе и предлагают обновлять политику только тогда, когда расхождение между траекториями текущей и предполагаемой политик становится слишком большим. Это ограничивает частоту обновления политик и, следовательно, снижает некоторые риски развертывания [32].

Обдумывание того, когда вы хотите обновить свою политику, — интересное упражнение. Вы обнаружите, что ваш агент находится где-то в континууме, который варьируется от непрерывного обучения (обновление при каждом взаимодействии) до пакетного обучения (обновление по расписанию) и заканчивается отсутствием обновления, как показано на рис. 10.9. Нет правильного ответа, т. к. это зависит от вашей задачи. Но я рекомендую вам держаться как можно ближе к тому, чтобы никогда не обновлять свою политику, поскольку это ограничивает частоту, с которой вы должны отслеживать свое развертывание на предмет ошибок или расхождений [33].

Я рекомендую вам попытаться сделать развертывания без сохранения состояния и неизменяемыми, если это возможно. Если вы обновляете параметры модели, это

называется *побочным эффектом* вызова функции. Побочные эффекты считаются плохими, потому что могут привести к неожиданным сбоям и затруднить восстановление после аварии. Например, один из способов добиться неизменности — попытаться рассматривать обновления политики как новые развертывания и проверять их так же, как при изменении кода. Это экстремально, но очень надежно.

**Рис. 10.9.** Частота обновления производственной политики влияет на стратегию развертывания. Любая стратегия развертывания находится в континууме между отсутствием обновления (развертывание вручную) и полным онлайн-обновлением

Традиционные методы развертывания из *разд. "Развертывание нескольких агентов" ранее в этой главе* также могут помочь ограничить проблемы, с которыми сталкиваются пользователи, рассматривая каждую новую реализацию политики, как определено параметрами политики, т. е. как отдельное развертывание. Автоматическое A/B-тестирование может гарантировать, что новые параметры не окажут негативного воздействия на пользователей.

## Безопасность, защита и этика

Агенты RL иногда могут управлять опасным оборудованием, таким как роботы или автомобили, что увеличивает риск принятия неправильных решений. Область, называемая *безопасным RL*, пытается справиться с этой проблемой. Агенты RL также подвержены риску атак, как и любая подключенная система. Но RL добавляет несколько новых возможных векторов атак помимо традиционных программных средств и эксплойтов машинного обучения. И я думаю, что инженеры — это первая линия защиты от этических проблем, и их (проблемы) следует рассматривать заранее, поэтому я считаю важным поговорить здесь об этом.

### Безопасное RL

Цель безопасного RL — изучить политику, которая максимизирует вознаграждение при работе в рамках заранее определенных ограничений безопасности. Эти ограничения могут существовать либо отсутствовать во время обучения и эксплуатации. Проблема с этим определением состоит в том, что оно включает идеи о предотвращении катастрофических обновлений в смысле вознаграждения. Были разработаны алгоритмы для внутреннего решения данной проблемы, например такие, как методы доверительной области (*см. разд. "Методы доверительной области" главы 6*).

Другая форма исследования включает внешние методы, которые предотвращают определенные действия от причинения физического вреда, денежных убытков или любого другого поведения, которое приводит к разрушительным результатам, и именно об этом я расскажу здесь. Очевидно, что лучше всего начать с техник, которые уже были описаны в этой книге.

Если ваша задача связана с проблемами безопасности, которые естественным образом можно описать как ограничение, вы можете использовать любой алгоритм, который выполняет ограниченную оптимизацию. TRPO и PPO — хороший выбор, в них вы можете адаптировать функцию оптимизации с учетом ограничений вашей задачи. Например, если вы обучаете алгоритм развертывания реплик веб-службы для удовлетворения спроса, он не должен развертывать больше, чем доступная мощность инфраструктуры. Эта информация может быть предоставлена как ограничение для функции оптимизации. Ха и соавт. представили хороший пример обучения робота предотвращению падения с помощью ограниченной версии SAC [35].

Точно так же вы можете сформировать вознаграждение, чтобы наказывать за определенное поведение. Например, роботизированная рука будет испытывать значительную нагрузку, когда она полностью вытянута. Для того чтобы продлить срок службы робота, не допускайте растяжения, которое может вызвать усталость металла. Вы можете включить в функцию вознаграждения элемент, запрещающий растяжение.

Проблема с этими двумя подходами в том, что они жестко запрограммированы и, вероятно, будут неоптимальными в более сложных задачах. Лучшим подходом могло бы быть использование имитации или обратного RL для изучения оптимальных ограниченных действий от эксперта.

Однако все предыдущие подходы не гарантируют безопасности, особенно во время обучения. Безопасность гарантируется лишь приблизительно, после достаточного количества тренировок. По сути, эти подходы учат, как действовать безопасно, совершая небезопасные действия, например ребенок или взрослый, дотрагивающийся до горячей чашки, несмотря на то, что их предупреждают о возможном ожоге.

В более позднем наборе подходов используется внешнее определение безопасности путем определения *набора безопасных* состояний высшего уровня. Например, вы можете определить максимальный диапазон, в котором роботу разрешено работать, или минимально допустимое расстояние между транспортными средствами в алгоритме следования за автомобилем. Учитывая это определение, внешний фильтр может оценить, безопасны действия или нет, и предотвратить их.

Грубая фильтрация может полностью предотвратить действие и наказать агента за его выполнение, что в основном аналогично формированию награды, за исключением того, что вы предотвращаете опасное действие. Но это не включает знание безопасного набора в процессе оптимизации. При таком подходе, например, вам придется прыгать с дерева на любой высоте, прежде чем вы точно узнаете, что определенные действия небезопасны, позвольте мне предложить страховочный трос.

Более прагматичный подход заключается в создании моделей, которые могут прерывать работу и безопасно перезагружать агентов. Зачастую вы можете спланиро-

вать свое обучение так, чтобы по определению избегать небезопасных действий. Например, Айзенбах и соавт. построили робота с ногами и обучили его ходить, изменив цель так, чтобы он находился подальше от небезопасных состояний. Например, если он переместится слишком далеко влево, то цель изменяется, чтобы быть справа. Они также изучают контроллер сброса, чтобы обеспечить автоматический метод возврата в безопасное состояние [36].

Исследователи рассмотрели множество способов включения этой информации. Одно из предложений — использовать развертывания для прогнозирования вероятности небезопасного события и отдавать предпочтение действиям, ведущим к безопасным траекториям [37]. Это можно включить в основанные на значениях алгоритмы RL, подтолкнув параметры аппроксимации значений к безопасным состояниям, а также включить в методы градиента политики, указав, как градиент должен переходить в безопасные состояния с поправкой, которая учитывает предвзятую генерацию политики [38].

Другой набор предложений, называемый *формально ограниченным RL*, пытается математически определить и, следовательно, гарантировать безопасность, но эти методы, как правило, полагаются на идеальные символические представления небезопасных состояний, которые возможны только в игрушечных примерах. Одна практическая реализация от Ханта и соавт. прибегает к использованию методики фильтрации действий на основе модели [39]. Без сомнения, это остается открытой проблемой, и я ожидаю, что в будущем появятся многие другие улучшения.

## Защитное RL

Если злоумышленник способен изменить исходный код приложения, он может внедрить вредоносный код. Это называется атакой *белого ящика*. Развертывание программного обеспечения снижает этот риск за счет строгих механизмов контроля доступа к кодовой базе. Но если разработчик предоставит программное обеспечение общедоступному Интернету, он рискует попасть в ситуацию, когда злоумышленник сможет исследовать и найти слабое место, которое позволит ему изменить работу приложения. Такой внешний подход называется атакой *черного ящика*.

Машинное обучение или любой процесс, управляемый данными, также уязвим для атак. В целом они классифицируются как атаки, нацеленные либо на *фазу обучения*, либо на *фазу прогнозирования*. На этапе обучения злоумышленник может иметь возможность изменять или манипулировать данными обучения для получения известного результата. Это обычно называется *заражением*. Например, если злоумышленник может ввести спам-письмо с неправильной меткой в общедоступный набор данных, применяемый для изучения спама, то он может повторно использовать аналогичное электронное письмо для продажи таблеток, которые, по-видимому, повышают либидо. Если же в учебном наборе, использованном для подготовки модели, попадались подобные письма, то она может ошибочно классифицировать их как "не-спам". Атаки с предсказанием предоставляют или дополняют входные данные для получения неверных результатов. Многие методы представляют собой методы белого хакера и предполагают знание базовой модели [40]. Но

совсем недавно исследователи обнаружили методы черного ящика, например, когда злоумышленник может создать вторую модель, аналогичную первой. После обучения злоумышленник может генерировать атаки белого хакера для второй модели и использовать их для атаки первой. Этот подход показывает, что атаки можно переносить [41]. Вместе они известны как *состязательные атаки.*

Агенты RL также могут быть атакованы теми же способами. Вашим первым шагом должно быть исправление основ. Злоумышленник не будет утруждать себя созданием обученной реплики, если он найдет открытый порт SSH на одном из ваших серверов или чьи-то учетные данные GitHub были подвергнуты фишингу, а у вас не включена многофакторная аутентификация. Затем убедитесь, что у вас есть надежные средства тестирования и мониторинга целостности данных; используйте ограничение скорости и защиту от отказа в обслуживании, чтобы ограничить объем модели, который злоумышленник может исследовать. Убедитесь, что ваши модели просты и надежны, насколько это возможно. Рассмотрим несколько исследований, чтобы выяснить, как вы могли бы улучшить контрмеры, введенные для ваших моделей машинного обучения.

RL представляет большие риски из-за увеличенной поверхности атаки. Модели машинного обучения обычно имеют одну точку входа — вход (input) и единственный ответ — выход (output). В RL злоумышленники могут нацеливаться на среду, наблюдение за средой, награды или действия. Успешное изменение любого из них может привести к тому, что политика будет стремиться к другому результату. Если злоумышленник имеет доступ к внутренним системам, он также может нацеливаться на вспомогательные службы, такие как буфер воспроизведения опыта или сервер параметров.

Способы предотвращения или смягчения этих типов атак активно исследуются. Одна ветвь напоминает нам, что надежность модели лежит в основе стратегии безопасности и тесно связана с безопасным RL. Если у вас есть модель или политика, которые хорошо работают в неблагоприятных условиях, то вероятность того, что она подвергнется внешней атаке, меньше. Бехзадан и Мунир предлагают возмущать агентов во время обучения, чтобы научить их быть более устойчивыми к прогнозирующим атакам, что является распространенной стратегией для укрепления моделей машинного обучения [42].

Другой подход, предложенный Лю и соавт., состоит в том, чтобы попытаться обнаружить враждебную атаку и остановить ее до того, как она нанесет какой-либо ущерб. RL является последовательным и часто коррелирует с течением времени, поэтому вы можете построить модель для прогнозирования следующего состояния с учетом текущего наблюдения. Если вы вычислите расхождение между предсказанием и тем, что произошло на самом деле, а затем сравните это с "нормальным" поведением (модель типичного распределения расхождений), вы можете создать алгоритм, который будет предупреждать вас, когда наблюдаются большие расхождения [43].

Вы можете подумать о том, чтобы направлять исследование, направляя действия агентов к состояниям, которые могут вызвать максимальную состязательную выго-

ду [44]. Или даже смягчить примеры состязательного обучения, рассматривая проблему как иерархическую; состязательное заражение, вероятно, будет "выглядеть" очень отличным от нормального взаимодействия, поэтому иерархическое RL должно иметь возможность рассматривать это как субполитику. Субполитики затем можно проанализировать, чтобы гарантировать легитимность и удаление зараженных данных [45].

Как видите, существует масса идей по борьбе с множеством потенциальных векторов атак. Но в этом и проблема: потенциал. Вложение времени, энергии и денег в противодействие этим атакам необходимо оценивать с учетом вероятности их возникновения. Например, если у вас есть агент, который работает во внутренней сети и никогда не доступен в общедоступном Интернете, то я бы посчитал этот уровень смягчения плохим использованием времени, учитывая столь низкую вероятность атаки. Вот почему вам необходимо провести анализ рисков для вашей конкретной ситуации.

Смягчение последствий также имеет убывающую отдачу, потому что знания и ресурсы для выполнения атаки возрастают в геометрической прогрессии. Согласно отчету Symantec об угрозах безопасности в Интернете за 2019 г., колоссальные 65% всех целевых взломов были вызваны фишингом, 23% — программным обеспечением без исправлений и 5% — поддельными обновлениями программного обеспечения. Только когда вы доберетесь до сути, вы обнаружите эксплойты, которые напрямую связаны с уязвимым агентом RL: 2% эксплойтов веб-сервера и 1% эксплойтов хранилища данных. Эти статистические данные демонстрируют, что базовые методы обеспечения безопасности более ценны для компании, чем методы RL или, возможно, даже методы предотвращения атак с использованием машинного обучения. Для ясности я не утверждаю, что безопасность RL не важна, просто сначала разберитесь с основами [46].

## Этическое RL

Возможно, это не совсем строгое определение термина, но мне нравится думать об *этике* как о профессиональном консенсусе индивидуальной морали. Это важное различие, потому что то, что вы считаете правильным и неправильным, не обязательно является таковым для меня. Но мне хотелось бы думать, что есть компромисс, на который мы оба можем согласиться. Итак, мораль представляет собой распределение убеждений, а среднее — этические стандарты. У инженеров, как правило, нет единого унифицированного способа или места для определения этических средств, поэтому важно, чтобы вы знали, что они существуют, по крайней мере на концептуальном уровне, и что ваша точка зрения — лишь одна из многих.

Я говорю это не для того, чтобы проповедовать философию или религию, а чтобы напомнить вам, что вы находитесь в положении властителя. У вас есть возможность и полномочия не только автоматизировать решения (машинное обучение), но и целые стратегии (RL). Каждый раз, когда вы применяете эти методы, вы фактически лишаете человека возможности принимать решения. Это может быть хорошо, если решение тривиально или обременительно, но иногда это не так. Иногда решение настолько важно, что его должен принять человек, чтобы он мог взять на себя ответственность за решение, правильное или неправильное. Человек волен исполь-

зовать информацию, предоставленную алгоритмом, но это не снимает с него ответственности. В других случаях решение может быть не столь важным в глобальном масштабе, но оно может быть важным в человеческом масштабе; человек может чувствовать потерю или беспомощность, если теряет способность принимать решения.

К сожалению, заинтересованные стороны (как пользователи, так и владельцы) ваших систем, как правило, не могут или не в состоянии независимо оценить влияние. У них могут отсутствовать полномочия или возможность протестовать, или они не могут понять технические тонкости. Вы в состоянии предотвратить это. Вы можете простым языком объяснить, каковы плюсы и минусы вашего решения таким образом, чтобы это касалось их лично. Вы можете выслушать и ответить на их вопросы. Вы можете объяснить, когда ваша система работает, а когда нет, и все предположения, которые вы сделали на этом пути.

Что касается вас, вы, вероятно, достаточно удачливы, чтобы иметь возможность выбирать, какую работу выполнять, или, по крайней мере, активно решаете не работать над чем-то или с кем-то, что (кто) нарушает ваши моральные принципы. Это негласное преимущество возможностей, образования и инженерии. Тот факт, что вы можете прочитать эту книгу, — поистине замечательное достижение. Вы можете отчасти снять с себя ответственность за выбор, активно размышляя о том, что для вас важно и как ваша работа влияет на мир. Ниже приводится список пунктов, которые могут быть важны для вас, а не оказаться таковыми. Каждый раз, когда вы разрабатываете приложение, спрашивайте себя.

- *Предвзятость.* Влияет ли это решение на одни группы людей больше, чем на другие? Есть ли в ваших данных аспекты, которые могут случайно добавить предвзятости?
- *Личное влияние.* Как приложение влияет на отдельных людей? Что они будут чувствовать, когда увидят/воспользуются вашим решением? Если люди будут затронуты, есть ли способ смягчить это? Можете ли вы включить людей в цикл принятия решений?
- *Климат.* Как ваше решение влияет на климат? Достаточно ли ценно приложение, чтобы окупить затраты на электроэнергию?
- *Использование не по назначению.* Как еще можно использовать ваше приложение? Было бы вам неудобно, если бы его использовали в определенных отраслях? Можете ли вы предотвратить это?
- *Гнусное использование.* Будет ли ваше решение прямо или косвенно использоваться для деятельности, с которой вы не согласны?
- *Законность.* Это законно? Может ли это быть признано незаконным в определенных правовых системах или странах?
- *Права человека.* Может ли ваше решение нарушать закон или права человека? Например, может ли это повлиять на частную жизнь людей?
- *Обоснованность и объяснимость.* Как можно оправдать действия своего алгоритма? Сколько объяснений нужно предоставить людям, чтобы его посчитали заслуживающим доверия?

- *Дезинформация о производительности.* Объяснили ли вы влияние вашего алгоритма или потенциальную производительность наиболее ясным и независимым способом? Есть ли шанс, что другие могут неверно истолковать ваши результаты и подвергнуться ненадлежащему наказанию?

В техническом плане предпринимаются попытки напрямую включить некоторые из этих идей в машинное обучение и RL. Например, ML-fairness-gym[16] — это набор компонентов, которые создают простые симуляции и оценивают долгосрочное влияние систем принятия решений в социальной среде. Так, в странах по всему миру кредитные рейтинги применяются для оценки вероятности выплаты долга физическими лицами. Но тот факт, что компании используют эту меру, побуждает людей искусственно завышать свои кредитные рейтинги с помощью таких методов, как использование кредита и даже завышение показателей доступности в приложениях. В свою очередь, людям, которые не используют эту тактику, тем, у кого нет возможности получить доступ к такой информации или доступ весьма ограничен, становится сложнее заручиться доверием, когда оно им требуется [47].

Долгосрочные последствия настолько трудно предсказать, что возникает соблазн не делать этого вообще. Но подождите, у вас есть доступ к фреймворку, который способен отслеживать и оптимизировать собственную работу для получения долгосрочного вознаграждения. Да, RL-исследователи зашли так далеко, что предположили, что вы можете использовать RL для оптимизации и принятия этических решений и даже рассмотреть возможность использования вознаграждения в качестве прокси для *формирования этики* [48, 49]. Может быть, RL спасет мир?

# Резюме

Насколько мне известно, в этой главе впервые даны последовательные и исчерпывающие советы по реализации и развертыванию агентов RL. В исследовательских статьях разбросано много лакомых кусочков, но инженеры в среднем просто не имели достаточного опыта работы с RL в производстве. Я надеюсь, что эта глава, да и эта книга, начнут менять ситуацию.

Конечно, это также означает, что я пишу с точки зрения моего опыта и того, что считаю важным. Не стесняйтесь прислушиваться к моему совету и адаптировать, изменять или полностью игнорировать его по своему усмотрению. Сама широта применения означает, что обстоятельства проекта должны диктовать, как эти идеи могут быть применены.

Несмотря на отсутствие промышленного использования (по сравнению, например, с машинным обучением), уже существует широкий спектр фреймворков, которые помогут вам быстро приступить к работе. Они варьируются от алгоритмических до операционных структур, а также включают специализированные вспомогательные структуры для реализации определенной части проблемы RL. Но вполне вероятно, особенно при запуске в производство, что вам все равно понадобится много усилий по настройке кода, поскольку RL все еще молод. Масштабирование RL аналогич-

[16] См. **https://oreil.ly/n3edb**.

ным образом поддерживается на обучающей стороне фреймворками, а традиционные программные стеки и оркестраторы должны помочь вам в масштабировании в производственную среду.

Оценка агентов имеет решающее значение, и ее можно было бы включить в *главу 9*. Но я оставил ее здесь, поскольку она становится действительно важной только тогда, когда вы планируете развернуть своего первого агента. Пока вы находитесь на этапе проверки концепции, обычно можно обойтись специальной оценкой со стандартными показателями и отладкой, зависящей от среды.

Когда решите развернуть проект в среде, где его могут использовать другие люди, вам следует сделать шаг назад и рассмотреть операционные аспекты выполнения RL. Найдите время на обдумывание того, чего вы пытаетесь достичь, и построить архитектуру, поддерживающую эти цели. Не бойтесь признаться, что некоторые аспекты не важны для экономии времени, но делайте это только с учетом рисков. Для поддержки развертывания вам потребуется использовать широкий спектр инструментов для разработки программного обеспечения и машинного обучения. Воспользуйтесь этим и адаптируйте их к своему процессу RL.

Безопасное RL — важная тема для агентов, которые взаимодействуют с реальным миром и могут напрямую влиять на реализацию алгоритмов. Постарайтесь уловить это требование как можно раньше, поскольку популярные алгоритмы могут работать хорошо, но могут быть небезопасными. Здесь важна защита вашего приложения, как и в любом проекте, и нужно знать о векторах атак, специфичных для RL. Но основные методы обеспечения безопасности должны иметь приоритет, поскольку они являются основной целью большинства атак.

И наконец, подумайте, как ваше приложение влияет на этические аспекты с точки зрения отрасли и на моральные аспекты с личной точки зрения. Чем выше вы поднимаетесь в стеке решений, тем важнее становится это делать. С этическим машинным обучением сложно разобраться, но, поскольку RL автоматизирует целые стратегии, есть потенциал для долгосрочных и далеко идущих последствий.

## Дополнительные материалы для чтения

- Программная инженерия.
  - Martin R. C. Clean code: a handbook of agile software craftsmanship. — Pearson Education, 2008.
  - McConnell S. Code Complete. — Pearson Education, 2004.
  - Hunt A., Thomas D. The pragmatic programmer: from journeyman to master. — Addison-Wesley Professional, 1999.
- Психология.
  - Kahneman D. Thinking, fast and slow. — Penguin UK, 2012.
  - Baumeister R. F., Tierney J. Willpower: rediscovering our greatest strength. — Penguin UK, 2012.

- Оценка.
  - Provost F., Fawcett T. Data science for business: what you need toknow about data mining and data-analytic thinking. — O'Reilly Media, 2013[17].
  - Я счел, что эта старая статья поможет понять традиционные оценочные методы, используемые в RL [50].
- Объяснимость.
  - Хороший обзор проблемы объяснимости результатов, правда, с уклоном в медицину [51].
  - Посмотрите проекты PyTorch Captum[18] и TensorFlow tf[19] — они помогают понять, как объясняются реализации, выполненные по принципу черного ящика.
  - Сравнительно новое исследование, посвященное объяснимости в RL [52].
- Микросервисы и стратегии развертывания.
  - Newman S. Building microservices: designing fine-grained systems. — O'Reilly, 2021[20].
- Непрерывная интеграция и развертывание.
  - Duvall P. M., Matyas S., Glover A. Continuous integration: improving software quality and reducing risk. — Pearson Education, 2007.
  - Humble J., Farley D. Continuous delivery: reliable software releases through build, test, and deployment automation. — Pearson Education, 2010.
- Безопасное RL.
  - Подробный, правда, слегка устаревший обзор [53].
  - Pfleeger C. P., Pfleeger L. S. Analyzing computer security: a threat/vulnerability/countermeasure approach. — Prentice Hall Professional, 2012.
  - Отличная диссертация Вахида Бехзадана (Vahid Behzadan) на эту тему [54].
- Этика.
  - Nielsen A. Practical Fairness. — O'Reilly, 2020[21].
  - Institute for Ethical AI & ML[22] является новаторской работой в этой области.
  - Я был действительно вдохновлен этой статьей о том, как ML можно использовать для борьбы с изменением климата [55].

# Использованные источники

[1] GitHub Archive Program / GitHub Archive Program : [site]. — Accessed 26 July 2020. — URL: **https://oreil.ly/GzE9T**.

---

[17] См. **https://oreil.ly/xh5ya**.

[18] См. **https://oreil.ly/AsYTe**.

[19] См. **https://oreil.ly/lUpRf**.

[20] См. **https://oreil.ly/96Tc1**.

[21] См **https://oreil.ly/yjc_K**

[22] См. **https://ethical.institute/**.

[2] Dhariwal P., Hesse C., Klimov O. et al. OpenAI Baselines // GitHub Repository. GitHub. — 2017.

[3] Hill A., Raffin A., Ernestus M. et al. Stable Baselines // GitHub Repository. GitHub. — 2018.

[4] Cassirer A., Barth-Maron G., Sottiaux T., Kroiss M., Brevdo E. Reverb: an efficient data storage and transport system for ML research. — 2020.

[5] Espeholt L., Marinier R., Stanczyk P., Wang K., Michalski M. SEED RL: scalable and efficient deep-RL with accelerated central inference // ArXiv:1910.06591. — 2020. — February. — URL: **https://oreil.ly/6wbbU**.

[6] Nair A., Srinivasan P., Blackwell S. et al. Massively parallel methods for deep reinforcement learning // ArXiv:1507.04296. — 2015. — July. — URL: **https://oreil.ly/pviAX**.

[7] Dean J., Corrado G., Monga R. et al. Large scale distributed deep networks // Advances in Neural Information Processing Systems 25, ed. by F. Pereira, C. J. C. Burges, L. Bottou, K. Q. Weinberger. — Curran Associates, Inc., 2012. — P. 123–1231. — URL: **https://oreil.ly/RRh-I**.

[8] Mnih V., Badia A. P., Mirza M. et al. Asynchronous methods for deep reinforcement learning // ArXiv:1602.01783. — 2016. — June. — URL: **https://oreil.ly/wplVP**.

[9] Niu F., Recht B., Re C., Wright S. J. HOG-WILD!: a lock-free approach to parallelizing stochastic gradient descent // ArXiv: 1106.5730. — 2011. — November. — URL: **https://oreil.ly/fbJ_i**.

[10] Clemente A. V., Castejón H. N., Chandra A. Efficient parallel methods for deep reinforcement learning // ArXiv:1705.04862. — 2017. — May. — URL: **https://oreil.ly/gHuxR**.

[11] Horgan D., Quan J., Budden D. et al. Distributed prioritized experience replay // ArXiv:1803.00933. — 2018. — March. — URL: **https://oreil.ly/nR8_I**.

[12] Wijmans E., Kadian A., Morcos A. et al. DD-PPO: learning near-perfect PointGoal navigators from 2.5 billion frames // ArXiv:1911.00357. — 2020. — January. — URL: **https://oreil.ly/sHcgQ**.

[13] Stooke A., Abbeel P. Accelerated methods for deep reinforcement learning // ArXiv:1803.02811. — 2019. — January. — URL: **https://oreil.ly/Id0xg**.

[14] OpenAI. 2018 // OpenAI Five. — 2018. — URL: **https://oreil.ly/k8Zwo**.

[15] Espeholt L., Soyer H., Munos R. et al. IMPALA: scalable distributed deep-RL with importance weighted actor-learner architectures // ArXiv:1802.01561. — 2018. — June. — URL: **https://oreil.ly/IU2-T**.

[16] Espeholt L., Marinier R., Stanczyk P., Wang K., Michalski M. SEED RL: scalable and efficient deep-RL with accelerated central inference // ArXiv:1910.06591. — 2020. — February. — URL: **https://oreil.ly/2CMLU**.

[17] Hung C.-C., Lillicrap T., Abramson J. et al. Optimizing agent behavior over long time scales by transporting value // Nature Communications. — 2019. — Vol. 10, № 1. — P. 5223. — URL: **https://oreil.ly/hX1ep**.

[18] Colas C., Sigaud O., Oudeyer P.-Y. A Hitchhiker's guide to statistical comparisons of reinforcement learning algorithms // ArXiv: 1904.06979. — 2019. — April. — URL: **https://oreil.ly/f-_z0**.

[19] Colas C., Sigaud O., Oudeyer P.-Y. How many random seeds? Statistical power analysis in deep reinforcement learning experiments // ArXiv:1806.08295. — 2018. — July. — URL: **https://oreil.ly/49cDY**.

[20] Lee X. Y., Havens A., Chowdhary G., Sarkar S. Learning to cope with adversarial attacks // ArXiv:1906.12061. — 2019. — June. — URL: **https://oreil.ly/0vN_I**.

[21] Osband I., Doron Y., Hessel M. et al. Behaviour suite for reinforcement learning // ArXiv:1908.03568. — 2020. — February. — URL: **https://oreil.ly/Lfqfj**.

[22] Gulcehre C., Wang Z., Novikov A. et al. RL unplugged: benchmarks for offline reinforcement learning // ArXiv:2006.13888. — 2020. — July. — URL: **https://oreil.ly/F3PKn**.

[23] James S., Ma Z., Arrojo D. R., Davison A. J. RLBench: the robot learning benchmark & learning environment // ArXiv: 1909.12271. — 2019. — September. — URL: **https://oreil.ly/N8ubF**.

[24] Leike J., Martic M., Krakovna V. et al. AI safety gridworlds // ArXiv:1711.09883. — 2017. — November. — URL: **https://oreil.ly/HfuRk**.

[25] Liu G., Schulte O., Zhu W., Li Q. Toward interpretable deep reinforcement learning with linear model U-Trees // ArXiv: 1807.05887. — 2018. — July. — URL: **https://oreil.ly/2ck-z**.

[26] Yoon J., Arik S. O., Pfister T. RL-LIM: reinforcement learning-based locally interpretable modeling // ArXiv:1909.12367. — 2019. — September. — URL: **https://oreil.ly/fDoKK**.

[27] Madumal P., Viller T., Sonenberg L., Vetere F. Explainable reinforcement learning through a causal lens // ArXiv:1905.10958. — 2019. — November. — URL: **https://oreil.ly/-4imX**.

[28] Atrey A., Clary K., Jensen D. Exploratory not explanatory: counterfactual analysis of saliency maps for deep reinforcement learning // ArXiv:1912.05743. — 2020. — February. — URL: **https://oreil.ly/eo4mI**.

[29] Agarwal A., Bird S., Cozowicz M. et al. Making contextual decisions with low technical debt // ArXiv:1606.03966. — 2017. — May. — URL: **https://oreil.ly/0Xj1M**.

[30] Merentitis A., Rasul K., Vollgraf R., Sheikh A.-S., Bergmann U. A bandit framework for optimal selection of reinforcement learning agents // ArXiv:1902.03657. — 2019. — February. — URL: **https://oreil.ly/V5Afb**.

[31] Lee K., Laskin M., Srinivas A., Abbeel P. SUNRISE: a simple unified framework for ensemble learning in deep reinforcement learning // ArXiv:2007.04938. — 2020. — July. — URL: **https://oreil.ly/GncZJ**.

[32] Matsushima T., Furuta H., Matsuo Y., Nachum O., Gu S. Deployment-efficient reinforcement learning via model-based offline optimization // ArXiv:2006.03647. — 2020. — June. — URL: **https://oreil.ly/nUg_v**.

[33] Tian H., Yu M., Wang W. Continuum: a platform for cost-aware, low-latency continual learning // Proceedings of the ACM Symposium on Cloud Computing. SoCC'18. — New York, NY, USA: Association for Computing Machinery, 2018. — P. 26–40. — URL: **https://oreil.ly/weHtP**.

[34] García J., Fernández F. A comprehensive survey on safe reinforcement learning // J. Mach. Learn. Res. — 2015. — Vol. 16, № 1. — P. 1437–1480.

[35] Ha S., Xu P., Tan Z., Levine S., Tan J. Learning to walk in the real world with minimal human effort // ArXiv:2002.08550. — 2020. — February. — URL: **https://oreil.ly/Ezr_4**.

[36] Eysenbach B., Gu S., Ibarz J., Levine S. Leave no trace: learning to reset for safe and autonomous reinforcement learning // ArXiv:1711.06782. — 2017. — November. — URL: **https://oreil.ly/JfDIW**.

[37] Wabersich K. P., Hewing L., Carron A., Zeilinger M. N. Probabilistic model predictive safety certification for learning-based control // ArXiv:1906.10417. — 2019. — June. — URL: **https://oreil.ly/dmjmF**.

[38] Gros S., Zanon M., Bemporad A. Safe reinforcement learning via projection on a safe set: how to achieve optimality? // ArXiv: 2004.00915. — 2020. — April. — URL: **https://oreil.ly/6vaDf**.

[39] Hunt N., Fulton N., Magliacane S. et al. Verifiably safe exploration for end-to-end reinforcement learning // ArXiv:2007.01223. — 2020. — July. — URL: **https://oreil.ly/lKX7i**.

[40] Goodfellow I. J., Shlens J., Szegedy C. Explaining and harnessing adversarial examples // ArXiv:1412.6572. — 2015. — March. — URL: **https://oreil.ly/6dXk-**.

[41] Papernot N., McDaniel P., Goodfellow I. et al. Practical black-box attacks against machine learning // ArXiv:1602.02697. — 2017. — March. — URL: **https://oreil.ly/dnIgv**.

[42] Behzadan V., Munir A. Whatever does not kill deep reinforcement learning, makes it stronger // ArXiv:1712.09344. — 2017. — December. — URL: **https://oreil.ly/WcQ8j**.

[43] Lin Y.-C., Liu M.-Y., Sun M., Huang J.-B. Detecting adversarial attacks on neural network policies with visual foresight // ArXiv: 1710.00814. — 2017. — October. — URL: **https://oreil.ly/haYxX**.

[44] Behzadan V., Hsu W. Analysis and improvement of adversarial training in DQN agents with adversarially-guided exploration (AGE) // ArXiv:1906.01119. — 2019. — June. — URL: **https://oreil.ly/PHjCj**.

[45] Havens A. J., Jiang Z., Sarkar S. Online robust policy learning in the presence of unknown adversaries // ArXiv:1807.06064. — 2018. — July. — URL: **http://arxiv.org/abs/1807.06064**.

[46] Internet Security Threat Report. — Symantec. — 2019. — Vol. 24. — URL: **https://oreil.ly/rdW_8**.

[47] Milli S., Miller J., Dragan A. D., Hardt M. The social cost of strategic classification // ArXiv:1808.08460. — 2018. — November. — URL: **http://arxiv.org/abs/1808.08460**.

[48] Abel D., MacGlashan J., Littman M. L. Reinforcement learning as a framework for ethical decision making // Workshops at the Thirtieth AAAI Conference on Artificial Intelligence. — 2016.

[49] Yu H., Shen Z., Miao C., Leung C., Lesser V. R., Yang Q. Building ethics into artificial intelligence // ArXiv:1812.02953. — 2018. — December. — URL: **https://oreil.ly/Xw5D1**.

[50] Kaelbling L. P., Littman M. L., Moore A. W. Reinforcement learning: A survey // ArXiv:Cs/9605103. — 1996. — April. — URL: **https://oreil.ly/w6Ffn**.

[51] Tjoa E., Guan C. A survey on explainable artificial intelligence (XAI): towards medical XAI // ArXiv:1907.07374. — 2020. — June. — URL: **https://oreil.ly/1Az_5**.

[52] Puiutta E., Veith E. M. S. P. Explainable Reinforcement Learning: A Survey // ArXiv:2005.06247. — 2020. — May. — URL: **https://oreil.ly/CIh1I**.

[53] García J., Fernández F. A comprehensive survey on safe reinforcement learning // J. Mach. Learn. Res. — 2015. — Vol. 16, № 1. — P. 1437–1480.

[54] Behzadan V. Security of deep reinforcement learning. — Manhattan, Kansas: Kansas State University, 2019. — URL: **https://oreil.ly/gS4WB**.

[55] Rolnick D., Donti P. L., Kaack L. H. et al. Tackling climate change with machine learning // ArXiv:1906.05433. — 2019. — November. — URL: **https://oreil.ly/VIeiJ**.

ГЛАВА 11

# Выводы и будущее

На этом этапе вы, возможно, ожидаете (и радуетесь), что это конец книги. Что ж, это не совсем так, потому что за время написания этой книги я собрал целый шведский стол из мелочей и идей для будущей работы. В первой половине этой главы я углубляюсь в различные советы и рекомендации, которые накопил и которые не вписывались в другие главы книги. Во второй половине я описываю текущие проблемы и определяю направление будущих исследований.

## Советы и рекомендации

Я уже несколько раз говорил об этом: RL сложно применять в реальной жизни из-за лежащих в его основе зависимостей от машинного обучения и разработки программного обеспечения. Но последовательная стохастичность добавляет еще одно измерение, задерживая и скрывая проблемы.

### Формулирование задачи

Часто вы будете сталкиваться с отраслевыми задачами, в которых компоненты марковского процесса принятия решений в лучшем случае нечеткие. Ваша цель должна состоять в том, чтобы уточнить концепцию и доказать (по крайней мере концептуально) жизнеспособность использования RL для этой задачи.

Для начала попробуйте визуализировать случайную политику, действующую в условной среде. Куда она ведет? С чем взаимодействует? Когда поступает правильно? Как быстро находит правильное решение? Если ваша случайная политика никогда не делает правильных вещей, то RL вряд ли поможет.

Попробовав случайную политику, попытайтесь решить задачу самостоятельно. Представьте, что именно *вы* — агент. Могли бы вы принять правильные решения после изучения каждого состояния? Можете ли вы использовать эти наблюдения для принятия решений? Что облегчит или усложнит вашу работу? Какие особенности вы могли бы извлечь из этих наблюдений? Опять же, если вы не можете теоретически придумать жизнеспособные стратегии, то RL (или машинное обучение, если на то пошло) вряд ли сработает.

Используйте базовые показатели для определения производительности. Начните со случайного агента. Затем используйте метод кросс-энтропии (который представляет собой простой алгоритм, который напоминает лучшее предыдущее развертыва-

ние из случайного исследования). Затем попробуйте простой алгоритм Q-обучения или градиент политики. Каждый раз, когда вы внедряете новый алгоритм, относитесь к нему как к научному эксперименту: создайте теорию, проверьте ее, извлеките уроки из эксперимента и используйте ее для руководства будущими реализациями.

Упростите задание настолько, насколько это в человеческих силах, а затем решите для начала то, что получилось. Стремитесь к быстрым победам. Под *упрощением* я подразумеваю максимально возможное сокращение пространства состояний и действий. Вы можете применить для этого инженерный подход или использовать свои знания предметной области, чтобы отбросить части состояния, которые не имеют смысла. И упростите определение цели или задачи. Например, вместо того чтобы пытаться поднять предмет с помощью робота, попробуйте сначала научить его перемещаться в нужное положение. Вознаграждение тоже упростите. Всегда старайтесь устранить разреженность, используя такие показатели, как расстояние до цели, а не положительную награду, когда агент достигает цели. Постарайтесь сделать вознаграждение плавным. Удалите любые отклонения или разрывы; это затруднит аппроксимацию вашей модели.

Спросите себя, нужно ли вам запоминать предыдущие действия, чтобы совершать новые. Например, если вам нужен ключ, чтобы открыть дверь, то вам следует помнить, что вы взяли в руку ключ. Если вы хотите поместить предмет в коробку, вам тоже нужно сначала взять его в руки. В этих ситуациях есть состояния, которые необходимо посещать в определенном порядке. Для того чтобы обеспечить эту возможность, вы должны включить рабочую память, и одной из распространенных реализаций является использование рекуррентных нейронных сетей.

Однако не путайте это с двумя действиями. Легко попасть в ловушку, думая, что вам нужна память, но на самом деле у вас есть действия, чтобы достичь этих состояний напрямую. Например, вы могли бы представить, что перед тем, как двигаться вперед в автомобиле, вам нужно включить двигатель. Звучит как упорядоченные задачи, но это не так, потому что у вас, вероятно, есть отдельные действия, чтобы завести машину и двигаться вперед. Агент без памяти вполне способен научиться держать машину включенной, чтобы двигаться вперед. Это также подчеркивает способ устранения необходимости в памяти. Если вы сможете найти действие, которое обеспечивает быстрый доступ к этому состоянию, то, возможно, вам удастся избавиться от рабочей памяти.

Также помните об иерархической политике на этом этапе. Подумайте, есть ли в вашем приложении разрозненные "навыки", которым можно было бы научиться для решения проблемы. Рассмотрите возможность их самостоятельного решения; если получится, это окажется концептуально и практически проще, чем использование иерархического RL, а конечный результат может быть таким же.

## Ваши данные

Я думаю, что подавляющее большинство времени в разработке уходит на переподготовку моделей, например после настройки некоторых гиперпараметров или изменения признаков. По этой причине, что почти само собой разумеется, вы должны

стараться делать все правильно с первого раза. Навык приходит только с опытом: опытом работы с RL в целом, а также в вашей области и с вашей задачей. Но вот несколько советов, которые должны немного сократить путь.

Как и в машинном обучении, масштаб и корреляция действительно важны. Не все алгоритмы *нуждаются* в масштабировании (например, деревья), но большинство из них все же нуждаются и ожидают, что данные будут нормально распределены. Поэтому, как правило, вы всегда должны масштабировать свои наблюдения, чтобы иметь среднее значение, равное нулю, и стандартное отклонение, равное единице. Вы можете сделать это с помощью чего-то настолько простого, как скользящее среднее и стандартное отклонение, или, если ваши входные данные являются стационарными либо ограниченными, вы можете использовать эти наблюдаемые пределы. Имейте в виду, что статистика ваших данных может изменяться с течением времени и делать вашу проблему нестационарной.

Также постарайтесь декоррелировать и чистить свои наблюдения. Модели не любят корреляции во времени или между признаками, потому что они представляют собой сложные искажения и дублируют информацию. Вы можете использовать z-преобразование, анализ главных компонент или даже простое различие между наблюдениями для декорреляции отдельных признаков. Используйте свои навыки анализа данных, чтобы находить и устранять корреляции между наблюдениями.

Отслеживайте все аспекты ваших данных. Посмотрите на распределение вознаграждений; убедитесь, что нет никаких отклонений. Посмотрите на входные данные еще раз, распределения графиков или визуализации, и удостоверьтесь, что входные данные не являются невозможными. Посмотрите на прогнозы, сделанные различными частями модели. Хорошо ли предсказывает ценностная функция? А как насчет градиентов? Они слишком большие? Посмотрите на энтропию пространства действий: оно хорошо исследуется или исследование пространства буксует на определенных действиях? Посмотрите на дивергенцию Кульбака — Лейблера: она слишком велика или слишком мала?

Подумайте об увеличении объема дискретизации, другими словами, увеличьте время между действиями. Это может сделать обучение более стабильным, потому что на него меньше влияет случайный шум, и ускорить его, потому что требуется меньше вычислительных действий. Фактический пропуск кадров в средах Atari является примером этого. Хорошее эмпирическое правило — представить, что вы выбираете действия; с какой скоростью вам нужно принимать решения, чтобы выработать жизнеспособную политику?

## Тренировка

Тренировка может быть довольно деморализующим процессом. Вы думаете, что у вас есть хорошая идея, а потом обнаруживаете, что это не имеет никакого значения, и вам потребовался день, чтобы реализовать ее и переобучить модель. Но, по крайней мере, это видно. Одна из самых больших проблем в RL прямо сейчас заключается в том, что легко адаптировать разработку алгоритма к вашей конкретной среде и, следовательно, создать политику, которая плохо обобщается.

Попробуйте разработать свой алгоритм для различных сред. Очень легко адаптировать вашу разработку к среде, которую вы используете для специального тестирования. Это требует дисциплины, но старайтесь поддерживать набор репрезентативных сред, от простых до реалистичных, которые также имеют различные настройки, такие как бóльшие или мéньшие, более простые или более сложные, полностью обслуживаемые или частично наблюдаемые. Если ваш алгоритм хорошо работает во всех них, это дает вам уверенность в том, что он способен обобщать.

Используйте разные предустановки. Иногда я думал, что у меня все хорошо в той или иной среде, но оказывалось, что одно конкретное начальное значение было сильно переобучено и случайно наткнулось на жизнеспособную политику, чисто случайно. Когда вы используете разные начальные данные, это заставляет вашего агента проводить разведку немного иначе, чем раньше. Конечно, ваш алгоритм должен работать независимо от начального значения, но не удивляйтесь, если это не так.

Проверьте чувствительность ваших гиперпараметров. Опять же, вы должны стремиться к алгоритму, который является надежным и ошибается изящно. Если он работает только с очень специфическим набором гиперпараметров, то это снова свидетельствует о переобучении.

Поиграйте с графиками скорости обучения. Фиксированное значение редко работает хорошо, потому что оно побуждает политику меняться, даже если она функционирует должным образом. Это может привести к тому, что агент отклонится от оптимальной политики.

## Оценка

Оценка, как правило, является одним из наименее проблемных этапов, потому что вы уже должны иметь представление о том, насколько хорошо идут дела после всего мониторинга, который вы проводили во время обучения. Но еще осталось достаточно топлива, чтобы обгореть.

Будьте очень осторожны, делая выводы при сравнении алгоритмов. Вы можете использовать статистику для количественной оценки улучшения (*см. разд. "Статистические сравнения политик" главы 10*), но я обнаружил, что делаю меньше ошибок при визуализации производительности. Основная идея заключается в том, что если вы можете увидеть значительное увеличение производительности, где есть четкое пространство между кривыми производительности по многим различным началам, то, скорее всего, это будет лучше. Если вы не видите разрыва, то проведите больше испытаний с бóльшим количеством начальных данных или просто не доверяйте этим данным. Звучит достаточно просто, но в действительно сложных условиях это не так. Часто, не по вине вашего алгоритма, вы можете получить реально плохой запуск. Это может произойти в сложных условиях, когда неудачное первоначальное исследование привело к состояниям, из которых трудно выбраться. Взгляните на состояния, которые ваш агент выбирает для изучения; возможно, вам захочется изменить стратегию исследования.

Ни один эталон или график не может по-настоящему доказать эффективность. Я нахожу, что целостное сочетание опыта работы с этой областью и средой и интуиции, подсказывающей, как *должен* вести себя алгоритм, определяет мое мнение об изменении производительности. Вот почему важно иметь исходные данные для сравнения со своими, потому что это дает еще одно доказательство.

Помните, что некоторые политики по умолчанию являются стохастическими, например многие алгоритмы градиента политики, поэтому для оценки используйте детерминированные версии этих политик, если вам нужна чистая оценка производительности. Аналогично, если у вас есть какое-либо случайное исследование в вашей политике, например ε-жадное, обязательно отключите его. Хотя вы и можете отключить его во время оценки, подумайте о сохранении стохастичности в реальной среде, если есть вероятность, что оптимальная политика может измениться со временем.

## Развертывание

Я думаю, что ключ к быстрой разработке успешных приложений определяется способностью комбинировать готовые компоненты и модули для решения конкретной задачи. Это требует определенной степени композиционности, которая в первую очередь обеспечивается на уровне программного обеспечения за счет надлежащей практики кодирования. Идея заключается в том, что вы смотрите на свою задачу, выбираете компоненты, которые могли бы помочь — внутреннюю мотивацию или буфер воспроизведения, например объединяете их все вместе с помощью красиво инкапсулированных интерфейсов и развертываете в унисон. На данный момент, к сожалению, до этой мечты еще далеко. Любое промышленное развертывание потребует значительных знаний в области разработки программного обеспечения и DevOps.

Фреймворки помогают и здесь, потому что в многих из них предусмотрена комплексная структура. Но это обычно означает, что они могут быть составлены в рамках своей кодовой базы. Фреймворки часто с трудом взаимодействуют с внешними компонентами из-за отсутствия стандартов. OpenAI Gym является очевидным исключением из этого правила.

Так что хорошенько подумайте о своей архитектуре, потому что я думаю, что это ключ к успешному развертыванию и эксплуатации реального продукта.

# Отладка

Несмотря на то что алгоритмы RL подпитывают бóльшую часть разговоров, связанных с искусственным интеллектом, методы глубокого RL "хрупкие, трудно воспроизводимые, ненадежные от запуска к запуску и иногда хуже по простейшим базовым показателям" [1]. Основная причина этого заключается в том, что составные части алгоритма недостаточно понятны: как различные части реализации влияют на поведение и производительность?

Детали реализации невероятно важны. Энгстром и соавт. считают, что, казалось бы, небольшие модификации, такие как оптимизация на уровне кода, могут объяснить бóльшую часть производительности во многих популярных алгоритмах. Например, несмотря на утверждение о том, что PPO является эффективным благодаря отсечению обновлений или ограничений, эти исследователи доказали, что оптимизация на уровне кода улучшила производительность больше, чем выбор алгоритма (они сравнили TRPO с PPO) [2].

Это приводит к очевидному выводу, что ошибки влияют на производительность. Например, во время исследования случайной сетевой дистилляции OpenAI (*см. разд. "Случайные вложения (сети случайной дистилляции)" главы 9*), Коббе и соавт. заметили, что, казалось бы, мелкие детали определили разницу между агентом, который застревает, и агентом, который может пройти сложную игру Atari. Они заметили одну проблему: приближение ценностной функции, которая должна быть стабильной после достаточного обучения, колебалось. Оказалось, что они случайно обнулили массив, который использовался для получения бонуса любопытства, что означало, что ценностная функция была понижена в начале каждого эпизода. Это объясняло колебания и, будучи зафиксированным, значительно улучшало производительность благодаря стабильным оценкам ценности [3].

Итак, вопрос в том, как вы предотвращаете эти ошибки? На самом деле, каждая отдельная строка кода содержит возможность ошибки, поэтому единственный способ устранить все ошибки — это вообще не иметь кода. Я часто проповедую эту мантру во время продвижения своих проектов в Winder Research, которые также имеют дополнительное преимущество в снижении операционной нагрузки. Очевидно, вам нужен код, но я хочу сказать, что чем меньше, тем лучше. Если у вас есть возможность использовать готовую библиотеку или компонент, используйте их, даже если они не соответствуют всем вашим требованиям. Точно так же, если вы пишете код или разрабатываете фреймворк, отдайте предпочтение лаконичному коду, а не причудливому. Вот почему я люблю такие языки, как Go или Python. Они активно продвигают простоту, потому что у них нет всей так называемой сложности C++ или Java.

Помимо этого, обычные методы отладки трассировки, визуализации ваших данных, тестирования и вывода отчетов станут вашим повседневным опытом работы с RL. По сути, отладка — это просто проявление научного метода: вы наблюдаете что-то странное (визуализации действительно важны), вы разрабатываете теорию, объясняющую такое поведение, вы разрабатываете способы проверки этой теории (инструкции print) и оцениваете результаты. Если вы хотите узнать больше, возьмите книгу по отладке программного обеспечения; некоторые рекомендации приведены в *разд. "Дополнительные материалы для чтения" в конце данной главы.*

RL является последовательным, это означает, что ошибки могут не проявляться в течение некоторого времени, а еще RL является стохастическим, т. е. ошибки могут быть скрыты шумом процесса обучения. Объедините это с базовыми зависимостями машинного обучения и разработки программного обеспечения, и вы получите летучий ведьмин котел с пузырящимися ошибками.

Если бы вы поговорили со своим руководителем проекта и оценили, сколько времени вы потратили бы на разработку алгоритма для новой задачи, вы бы свою занятость оценили бы так, как показано в верхней части рис. 11.1; бóльшая часть времени тратится на исследования и внедрение, верно? Нет. Подавляющее большинство времени уходит на отладку в попытках понять, почему ваш агент не может решить простейшую задачу. В нижней части рис. 11.1 показано нечто более репрезентативное, с добавлением времени, необходимого для правильной повторной реализации вашего проекта, с чистым кодом и тестами. Измерения, которые я выбрал, произвольны, но, по моему опыту, они наглядны.

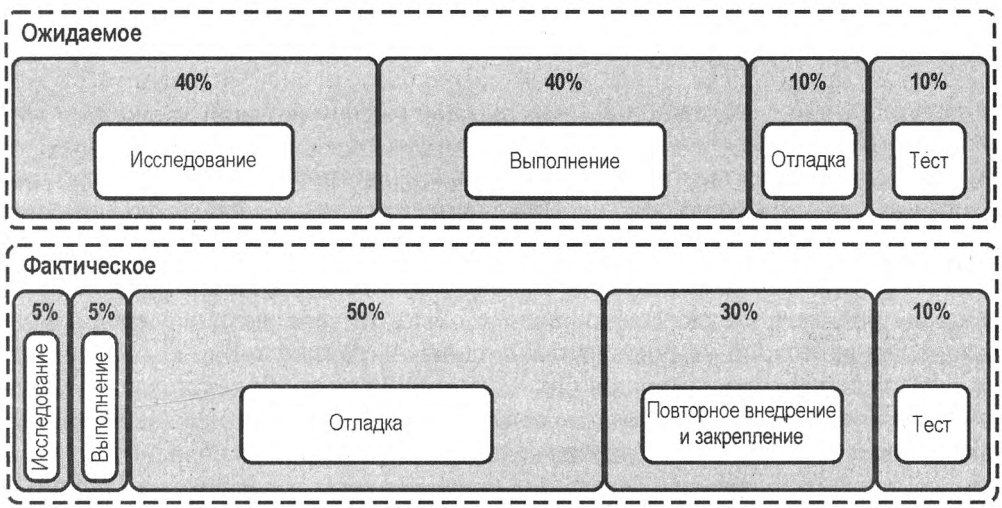



**Рис. 11.1.** Сравнение ожидаемого и фактического времени, затраченного на деятельность во время разработки алгоритма. Вдохновлено Мэтью Рацем [4]

## Алгоритм не может решить проблемы среды!

Несмотря на то что описываемые алгоритмы могут достигать удивительных результатов, изучая методы решения, которые вы могли бы даже счесть разумными, важно помнить, что они не всесильны. Они не могут решить все задачи и, конечно, не делают этого автоматически.

Используя RL в прикладной сфере, вы должны придерживаться комплексного подхода. Очень вероятно, что вы сможете сделать нечто, значительно упрощающее задачу. Основная причина, по которой методы RL терпят неудачу, заключается не в том, что алгоритм недостаточно хорош, а в том, что задача не решается таким способом. Например, среда MountainCar-v0, как известно, чрезвычайно сложна для решения алгоритмами RL, несмотря на то, что она концептуально проста. Это не проблема с алгоритмом; ценностная функция на удивление проста. Это задача на исследование и зачетная работа. Если вы разработали лучший способ исследования или изменили вознаграждение, чтобы стимулировать движение вверх по склону, один из этих алгоритмов может решить проблему в кратчайшие сроки.

Также не существует единого метода получения решения для среды. Иногда проблема заключается в пространстве наблюдения, иногда в исследовании, иногда в гиперпараметре. Добро пожаловать в RL, где интеллектуальные агенты иногда автоматизируют сложные стратегические цели. Моя единственная рекомендация — использовать эту книгу, чтобы шаг за шагом двигаться к разрешимой задаче. Если бы вы заставили меня выбрать один из них, я бы начал с упрощения задачи или пространства наблюдений. Сделайте их достаточно простыми, чтобы можно было визуализировать, что происходит, а затем решить эту простую задачу. Учитывая сказанное выше, вы, вероятно, будете обладать достаточными знаниями в предметной области, чтобы понять задачи с более сложной версией.

## Мониторинг для отладки

Мониторинг в производственной среде для оперативных целей не предоставляет достаточно низкоуровневой информации, которая помогла бы вам отладить алгоритм RL. При разработке агента вам захочется регистрировать, строить графики и визуализировать как можно больше. Ниже приведен список советов, которые помогут, когда вам понадобится дополнительная информация.

- Регистрируйте и визуализируйте наблюдения, действия и награды. Действия должны обладать высокой энтропией, чтобы стимулировать исследования. Награды для состояний не должны быть одинаковыми; если они таковы, агент не может сказать, какие состояния более ценны. Проверьте представленное агенту наблюдение; легко случайно выбросить важную информацию. Например, убедитесь, что преобразование изображения в оттенках серого сохраняет элементы внутри этого изображения; зеленый и красный цвета светофора явно различаются по цвету, но в оттенках серого не так сильно.
- Оцените ценностную функцию для различных частей вашего пространства состояний. Если возможно, визуализируйте все пространство. В противном случае выберите несколько репрезентативных состояний и постройте их график. Не имеем ли мы дело с переобучением?
- Постройте график ценностной функции от времени (для разумного числа состояний). Как она меняется? Не застревает ли? Колеблется ли? Какова разница во время запуска? Какова разница в течение нескольких прогонов? Оценка должна изменяться с течением времени, а затем стабилизироваться. Помните, что при изменении политики меняется и функция идеальной ценности. Аппроксимация ценностной функции должна оставаться достаточно стабильной, чтобы справляться с нестационарностью, но достаточно гибкой, чтобы реагировать на изменения.
- Плохие или глючные реализации нейронных сетей создают плохое впечатление об RL. Начните с чего-нибудь простого, линейного аппроксиматора или эквивалентной однослойной нейронной сети. Убедитесь, что все работает должным образом, прежде чем пробовать более сложные сети. Почитайте хорошую книгу с практическими рекомендациями по глубокому обучению, чтобы помочь себе в этом деле (*см. разд. "Дополнительные материалы для чтения" в конце данной главы*).

- Убедитесь, что все работает с фиксированными начальными числами и однопоточностью, прежде чем масштабировать до нескольких начальных чисел или многопоточных/многоузловых запусков.
- Начните с гиперпараметров, которые точно работают хорошо. Пользуйтесь вашим опытом с другими реализациями.

# Будущее обучения с подкреплением

Что прямо сейчас думают об RL? Это заставляет меня вспомнить почти основополагающий доклад Эндрю Ына о практическом глубоком обучении, где ему удается идеально сбалансировать здоровую дозу краткосрочного цинизма с долгосрочным оптимизмом [5]. Это заставило меня осознать сложность предмета, особенно когда речь заходит о промышленных внедрениях, но в то же время оптимизм заставил меня почувствовать, что эти проблемы были просто частью процесса. Проблемы существуют, так что примите их, справьтесь с ними наилучшим образом и двигайтесь дальше. Иногда глубокое обучение не имеет смысла, и это нормально. Но когда смысл появляется, слияние модели и данных — это красота, которую стоит созерцать.

Я думаю, что RL сейчас находится в таком положении. Мы находимся на пороге того момента, когда промышленное производство и внедрение возможны, но сложны. Есть еще много случаев, когда более простые алгоритмы превосходят RL; например, если вы можете играть в своей среде в автономном режиме, то вы можете просто использовать алгоритм поиска Монте-Карло по дереву (Monte Carlo tree search, MCTS), например AlphaGo, для оценки ваших возможных ходов. Так, эксперимент от 2014 г. снизил производительность DQN с использованием MCTS, и потребовалось некоторое время, прежде чем методы RL смогли нагнать [6]. Но это нормально. RL — это не палочка-выручалочка, а машинное обучение еще пока не родило детей и не создало искусственный интеллект. Это всего лишь инструмент. И, подобно плотнику, вы должны использовать лучший инструмент для своей работы.

## Рыночные возможности RL

Глубокое обучение стало мейнстримом, поэтому даже мои корпоративные клиенты инвестируют во внедрение. Я верю, что RL последует за DL в течение следующих нескольких лет. Это открывает новые рынки и возможности. К сожалению, как и машинное обучение, RL — это межотраслевой метод, а не решение. Вам понадобятся знания в предметной области, в области отраслевого рынка и немного вдохновения, чтобы использовать RL.

Однако уже есть признаки того, что RL особенно хорошо подходит для определенных сфер. Робототехника — одна из очевидных областей. Но автоматизированные торги и рекомендации могут обогнать робототехнику по объему глобальных доходов. Пока не было никаких отраслевых исследований, посвященных RL, но это произойдет, если RL будет успешным.

Аналитики компании по исследованию рынка Gartner считают, что RL все еще находится на подъеме кривой ажиотажа и треть предприятий стремятся автоматизировать принятие стратегических решений (которые они называют *интеллектуальным принятием решений*, decision intelligence) к 2023 г. [7, 8]. Они достаточно расплывчато говорят о рыночной стоимости, особенно когда речь заходит о количественной оценке влияния конкретных технологий, но, как сообщается, мировая ценность для бизнеса составляет многие триллионы долларов в год [9]. Интересно, что в некоторых визуализациях исследователи Gartner начинают отделять "агентов" и "автоматизацию принятия решений" от других форм более традиционного машинного обучения, которые они называют "поддержкой/дополнением решений", чтобы можно было четко увидеть появляющуюся ценность оптимизации последовательных решений.

Это обещание означает, что оперативное пространство несет ценность и возможности. Компании не смогут использовать нынешний набор предложений машинного обучения как услуги, потому что большинство из них не очень хорошо подходят для непрерывного обучения и последовательного принятия решений. Некоторые, конечно, адаптируются, но у компаний есть возможности заполнить это пространство.

Проекты с открытым исходным кодом также не будут сильно отставать, а во многих случаях могут даже оказаться впереди. Можно привести аргумент в пользу того, что только проекты с открытым исходным кодом могут быть по-настоящему успешными, потому что, как и машинное обучение, глобальное внедрение требует открытого сотрудничества. В противном случае ваши потребности и проблемы могут вступить в противоречие с проприетарными процессами и мнениями. Если вы рассматриваете этот вариант, убедитесь, что вы позиционируете свой продукт на некотором расстоянии от того, что может заинтересовать крупных поставщиков. И обязательно подумайте о том, как вы и мировое сообщество можете получить взаимную выгоду от разработки с открытым исходным кодом.

Это также означает, что компаниям еще долго будут нужны поддержка и опыт RL в будущем. Может пройти целых 10 лет, прежде чем RL станет товаром. Поэтому будут востребованы консультанты и инженеры с очевидным опытом в области RL, такие как сотрудники моей компании. Если вы хотите специализироваться на RL для карьерного роста, думаю, что это неплохое решение, учитывая все возможности и прогнозы, которые мы только что обсуждали. Но не позволяйте ложному чувству безопасности убаюкивать себя. RL, ML и разработка программного обеспечения — это сложно, особенно когда вы пытаетесь внедрить эти вещи в производство. И, как и в любой отрасли, основанной на знаниях, цели постоянно меняются. Вы должны принять это и двигаться вместе с ними, чтобы обеспечить долгосрочную позицию в этой высококонкурентной области. Для того чтобы приложить столько усилий, вы действительно должны любить свое дело. Вы должны наслаждаться самим ремеслом и испытанием невзгодами; радостно встречать вызовы, выдвигаемые вам конкретной промышленной ситуацией, с которой никто и никогда раньше не сталкивался.

## Будущее RL и направления исследований

У RL много проблем, о чем я говорил на протяжении всей книги. Не позволяйте им отвлекать вас от изучения потенциала RL, т. к. я искренне верю, что эти идеи приведут к следующему уровню искусственного интеллекта; они просто ждут прорывных приложений. Но, надеюсь, я четко обозначил, какие части RL могут замедлить вас или заставить вас споткнуться при работе над отраслевой задачей. Эти проблемы в основном воплощаются в будущих исследовательских возможностях.

Направления исследований можно в целом разделить на два: промышленные и академические исследования. Промышленные исследования состоят из способов повышения эффективности разработки и эксплуатации и определяются потребностями бизнеса. Академические исследования в большей степени направлены на улучшение или переопределение фундаментального понимания и основаны на теоретическом понятии новизны или прогресса, как их определяют коллеги. Это означает, что научные исследователи не собираются тратить свое время, например, на улучшение читаемости своего кода. Точно так же промышленные исследователи вряд ли наткнутся на что-то действительно новаторское. Но, конечно, есть люди, которые находятся посередине.

Я считаю, что в настоящее время в академическом сообществе машинного обучения существует проблема. Основными местами для академического успеха являются конференции и журналы. Самые популярные могут позволить себе выбирать, поэтому у рецензентов есть много возможностей для управления текущим и будущим контентом. И, к сожалению, приложения считаются менее ценными, чем, например, новое алгоритмическое усовершенствование Так, Ханна Кернер недавно рассказала о своей попытке продвинуть на конференциях научные работы, ориентированные на приложения Однако она обнаружила, что "значение кажется ограниченным для сообщества машинного обучения" [10]. Конечно, фундаментальное улучшение — главная цель для академических кругов и научных открытий в целом, но я чувствую, что эта чаша весов перевесила слишком сильно.

### Исследования в промышленности

Начнем с промышленности; вот те области, в которых, по моему мнению, есть исследовательские возможности и импульс.

- *Введение в эксплуатацию и архитектура.* Здесь RL используется все больше и больше, и вскоре в производство будет внедрено множество приложений. Я не думаю, что какая-либо из существующих систем машинного обучения обладает такими возможностями, а системы RL, как правило, ориентированы на разработку, а не на производство. Конечно, есть несколько исключений, но есть и много возможностей для улучшения, особенно по части архитектуры.
- *Проектирование и реализация RL.* В этой области уже проделана большая работа — взгляните, например, на все фреймворки RL: многие из них, как правило, фокусируются не на той задаче. Многие из них пытаются реализовать алгоритмы RL, но это не является целью для промышленной проблемы. Цель состоит в том, чтобы найти, усовершенствовать и создать RL-решение для решения биз-

нес-задач. Это означает, что такие вещи, как разработка функций, формирование вознаграждения и все остальное, перечисленное в предыдущих двух главах, также важно. Возможно, даже больше, чем алгоритм. Фреймворкам необходимо переключить свое внимание с разработки алгоритмов на разработку приложений.

- *Курирование среды и симуляция.* Очевидно, что симуляции имеют решающее значение для начальной разработки. Точное моделирование обеспечивает возможность переноса знаний в реальный мир. По мере решения более сложных промышленных задач они будут извлекать выгоду из более реалистичных симуляций. Это уже сейчас относится к программам для 3D-моделирования, которые способны имитировать пугающе хорошие образы реального мира. Но для всех остальных областей трудно найти достойные, реалистичные симуляторы. Одна из причин этого заключается в том, что они содержат много проприетарной информации. Модели, которые управляют симуляциями, часто строятся на основе данных, которыми владеет компания, поэтому они знают, что это представляет собой конкурентное преимущество. Одним из способов, при помощи которых исследования могли бы здесь помочь, является улучшение процесса создания и развития среды. Это обходит проблему данных стороной, но все равно поможет на этом решающем этапе развития.
- *Мониторинг и объяснимость.* Инструментарий и поддержка, связанные с мониторингом агентов RL, отсутствуют и могут быть объяснены отсутствием практической реализации. Когда вы работаете над промышленной задачей, вы должны быть уверены, что ваша политика надежна, и единственный способ убедиться в этом — обеспечить надлежащий надзор. Некоторые приложения могут даже требовать объяснимости, что в настоящее время довольно сложно, особенно в промышленных условиях. Но более того, вам нужны инструменты, помогающие отлаживать разработку приложений, инструменты, которые дают вам представление о том, что происходит внутри агента. Проблема в том, что это, как правило, зависит от алгоритма и предметной области, но такая задача стоит того, чтобы заняться ею.
- *Безопасность и надежность.* Это тоже горячая тема в академических исследованиях, но влияние исследования ощущается только тогда, когда оно применяется в реальных приложениях. Доказательство жизнеспособности вашей модели важно для каждого приложения, поэтому я представляю библиотеки и инструменты, которые помогут решить эту задачу. Безопасность — проблема посерьезнее. Меня меньше беспокоят технические аспекты, т. к. я уверен, что инженеры могут создавать решения, безопасные по дизайну. Меня беспокоят правовые и нормативные последствия, потому что на данный момент это становится скорее этической проблемой. Кто несет юридическую ответственность? Насколько это должно быть безопасно? Как вы проверите безопасность?
- *Приложения.* И, конечно же, предприятия платят за применение RL. Направления исследований, обсуждаемые здесь, помогают в реализации этого применения. Интерес к RL не увядает, а значит, исследований по применению RL к бизнес-задачам будет все больше. Самое хитрое, как и в случае с любой техноло-

гией, — это сопоставление технологии с бизнес-задачей. Это намного сложнее, чем кажется, потому что RL — очень обобщенная вещь; существует множество возможных применений. Поэтому используйте предложения, представленные в *разд. "Определение проблемы: что такое проект RL?" главы 9*, чтобы найти жизнеспособную проблему RL, а затем расставить приоритеты в соответствии с запросами бизнеса.

## Исследования в науке

С академической точки зрения RL — вещь весьма увлекательная, потому что кажется, что мы находимся всего в нескольких шагах от чего-то действительно революционного: агента, который может научиться выполнять любое действие так же хорошо, как и человек. Этот момент подобен переборкам корабля; как только она пробита, открывается целый мир возможностей, в основном хороших, но иногда пугающих.

Современные приложения машинного обучения и в меньшей степени RL ограничены их неспособностью использовать научный метод и сохранять эти знания разумным способом. Когда агент станет способен разрабатывать собственные теории, проверять собственные убеждения и сможет применять эти знания для открытия новых теорий, тогда-то мы сможем применять RL для решения любой последовательной задачи на любых данных. И я чувствую, что мы почти на месте. Но, если быть точным, проблемы, препятствующие достижению этой цели, с прикладной точки зрения заключаются в следующем.

- *Офлайн-обучение.* Это, наверное, самая большая проблема. Основная причина, по которой нам нужны симуляторы, заключается в том, что агент должен протестировать действие, чтобы изучить политику. Если мы сможем отойти от этого режима, обучая политике без такого взаимодействия, это означает, что мы можем полностью удалить симулятор. Просто регистрируйте эти данные и тренируйтесь на этом, как в машинном обучении. Исследователи уже начали совершенствовать этот процесс; посмотрите, например, *главу 8*. Но многое еще оставляет желать лучшего. Я предполагаю, что вполне возможно представить себе алгоритм, способный понимать стохастичность среды, наблюдать примеры оптимальной траектории, а затем применять модель для соответствия обобщенной траектории данным. Это должно обеспечить компромисс между поиском оптимальной политики и сохранением достаточной неопределенности в отношении состояний, ненаблюдаемых среди зарегистрированных.
- *Эффективность выборки.* Повышение эффективности выборки алгоритмов желательно, поскольку приложениям, как правило, требуется оптимальная политика при наименьшем количестве взаимодействий. Рассмотрим пример с корзиной покупок: я действительно не хочу, чтобы мне доставляли пакеты с кормом для собак и кошек, когда у меня нет домашних животных. Чем быстрее алгоритм сможет усвоить это предпочтение, тем меньше я буду расстраиваться. Опять же, здесь существует целый ряд направлений исследований, от улучшения исследований до консолидации знаний, таких как ансамбли, но они часто определяются предметной областью. Было бы здорово, если бы существовала какая-то универ-

сальная теоретическая гарантия эффективности, к которой могли бы стремиться алгоритмы.

- *Высокоразмерная эффективность.* Большие пространства состояний и действий по-прежнему проблематичны. Большие пространства состояний неудобны из-за количества вычислительной энергии, необходимой для того, чтобы свести их к чему-то полезному. Полагаю, что для некоторых областей можно было бы использовать предварительно подготовленные пребразователи или вложения для упрощения пространства состояний без необходимости обучения сокращению. Большое пространство действий более проблематично, т. к. агенты должны постоянно активно пробовать все доступные действия, чтобы убедиться в наличии у них оптимальной политики. Ситуация проще, когда действия взаимосвязаны. Например, одно непрерывное действие бесконечно, но его довольно легко преобразовать, потому что вы можете смоделировать действие с помощью распределения. Правда, многие отдельные действия вызывают больше проблем главным образом потому, что это приводит к более сложной задаче исследования.
- *Чувствительность функции вознаграждения.* Контролируемое машинное обучение — это легко, потому что вы знаете, какова цель. Мне бы очень хотелось иметь такой простой способ определения целей в RL. Вознаграждения RL часто скудны, имеют несколько целей и очень многомерны. По сей день определение функции вознаграждения в значительной степени является утомительным ручным процессом, даже с использованием таких методов, как внутренние вознаграждения. Я предполагаю, что в будущем еще останется некоторое ручное вмешательство для определения целей высокого уровня, но я рассчитываю делать это менее инвазивным способом; мне не нужно будет формировать вознаграждение. Проблема становится еще более важной, когда вознаграждение откладывается.
- *Сочетание различных видов обучения.* Есть ли способ объединить исследовательские преимущества мультиагентного RL, обобщение иерархического RL и экспертное руководство? Могли бы вы объединить эволюционное или генетическое обучение с этими методами RL?
- *Предобученные агенты и улучшение обучения по учебной программе.* Где-то я прочитал, что обучение возможно только тогда, когда разрыв между знанием и незнанием невелик. Верю, что это верно и в RL. Я не ожидаю, что ребенок сможет водить машину; точно так же вы не должны ожидать, что вашу задачу решит неподготовленный агент. И переобучение этой одной задаче не является решением, поскольку вождение автомобиля — нечто большее, чем просто обучение включению комплексного стереовидения и управлению рулем. Агенту необходим широкий круг знаний, которые могут быть предоставлены агентами, предобученными по определенным учебным программам. Если хотите, в конце можете даже выставить им оценки.
- *Повышенное внимание к реальным приложениям и контрольным показателям.* Думаю, в академических кругах понимают, что применение наработок в реальной жизни желательно, но я хотел бы, чтобы этому уделялось больше внимания.

Я вижу алгоритмы, которые превосходят эталонные показатели Atari, и это сделано специально, чтобы иметь возможность претендовать на самую современную производительность.

## Этические стандарты

Этические или моральные стандарты машинного обучения и RL распространяются как на промышленность, так и на академические круги. Похоже, что существует плавильный котел желания продвинуть дискуссию, но это трудно. Может быть, потому что это всегда превращается в философский спор, а может, потому что у каждого свой набор моральных принципов или контекстов, из-за которых трудно договориться о том, какие этические выводы следует делать. Это действительно деликатная тема, которую я не вправе комментировать. Но это не означает, что вы не должны об этом думать; вы должны. Во всем, что вы делаете.

Что касается будущего, сомневаюсь, что существет поток попыток хотя бы определить этические стандарты, не говоря уже об их решении. Общее положение о защите данных (General Data Protection Regulation, GDPR) является одним из примеров попытки правительства определить этичное использование чьих-либо данных. И хотя я полностью согласен с этой идеей, на практике она слишком расплывчата и содержит чересчур много юридического жаргона, чтобы быть полезной. Например, статья 7 (условия для согласия GDPR), гласит:

> "Согласие должно быть представлено в форме, которая четко отличается от других вопросов, в понятной и легкодоступной форме, должен использоваться ясный и понятный язык".

Справедливо. Но на той же самой странице пункт 4 гласит:

> "При оценке того, свободно ли дается согласие, следует в максимальной степени учитывать, зависит ли, в частности, выполнение контракта, включая предоставление услуги, от согласия на обработку персональных данных, которые не являются необходимыми для выполнения этого контракта".

Ясный и понятный язык? И в качестве доказательства полного пренебрежения буквой этого закона в пояснительном пункте 32, "условия согласия", говорится, что "предварительно отмеченные флажки или бездействие не должны, следовательно, означать согласие". Я бы рискнул предположить, что почти все формы регистрации GDPR на веб-сайте содержат поле "принять все", которое, конечно, проверяет все поля, прежде чем у вас будет возможность прочитать, для чего они предназначены.

Эта псевдобезопасность тоже имеет свою цену. Меня постоянно раздражает необходимость находить крестик, чтобы удалить всплывающее окно. (Кстати, что делает кнопка закрытия: принимает все или отклоняет все?) Это все равно что вернуться в 90-е годы, когда сайты GeoCities были до краев заполнены всплывающими окнами.

Итак, какое отношение имеет GDPR к машинному обучению и этическим стандартам RL? Это первая и крупнейшая попытка правительства обуздать практику, связанную с данными, которую оно сочло неэтичной. Сработало ли это? Может быть,

немного. Это приструнило некоторые выходки отдельных крупных компаний или правительств, которые пытались подлить масла в огонь важных разговоров о демократии и свободе. Но какой ценой? Есть ли лучший способ? Возможно, но не думаю, что существует единственный правильный ответ.

В отношении более общих применений машинного обучения и RL, чувствую, это будет медленный процесс; они не станут важными, пока ситуация не выйдет из-под контроля, немного похоже на изменение климата. Думаю, успех будет достигнут на низовом уровне, однако ситуация немного смахивает на то, как этический взлом заставляет компании инвестировать в цифровую безопасность. Все предприятия хотят быть безопасными и защищенными, но для того, чтобы что-то с этим сделать, требуется провести сложные работы. То же самое будет справедливо и для данных и моделей, полученных на их основе. Кое-кто даже будет увлеченно попытаться привлечь компании или правительства к ответственности.

И потом, есть такие люди, как вы. Практикующие. Инженеры. Люди, создающие эти решения. Вы в первых рядах наблюдающих за результатами своей работы, поэтому на вас лежит большая ответственность за то, чтобы решить, является ли ваш алгоритм этичным. Это бремя знаний; используйте его с умом и принимайте правильные решения.

# Заключительные замечания

По иронии я заканчиваю книгу об оптимизации последовательного принятия решений замечанием, в котором прошу вас принимать правильные решения. Кто знает, может быть, в будущем мы сможем использовать RL для принятия этических решений за нас.

До недавнего времени RL был в основном предметом академического интереса. Цель этой книги состояла в том, чтобы попытаться привнести некоторые из этих знаний в промышленность и бизнес, чтобы продолжить путь повышения производительности за счет автоматизации. RL обладает потенциалом для оптимизации процессов принятия решений на стратегическом уровне с использованием целей, непосредственно вытекающих из бизнес-показателей. Это означает, что RL может повлиять на целые бизнес-структуры, от заводских рабочих до генеральных директоров.

Это создает уникальные технические и этические проблемы. Например, как решения могут иметь долгосрочные последствия, которые в то время неизвестны, и как автоматическая стратегия может повлиять на людей. Но это не должно мешать вам экспериментировать, чтобы увидеть, как RL может помочь вам в вашей ситуации. Это действительно просто инструмент; важно то, как вы его используете.

## Дальнейшие шаги

У вас, вероятно, возникнет много вопросов. Это хорошо. Соберите эти вопросы и отправьте их мне (контактные данные *см. в разд. "Об авторе"*). Но в целом я рекомендую вам не останавливаться на достигнутом. Попытайтесь найти практическую

задачу для решения, в идеале — на работе. Если вы сможете убедить свое руководство поддержать ваши исследования, возможно, в формате 20% вашего времени, или в качестве полноценного доказательства концепции, это даст вам время и стимул для решения реальных проблем, с которыми вы столкнетесь. Если вы будете придерживаться игрушечных примеров, боюсь, это будет слишком легко для вас. Если не можете делать это в рабочее время, то найдите интересующую вас проблему, возможно, связанную с вашим хобби. Вероятно, у вас не будет специально выделенного времени, потому что жизнь мешает, но, надеюсь, это будет полезно и даст вам какой-никакой практический опыт.

Помимо практического опыта рекомендую вам продолжать читать. Очевидно, что RL сильно зависит от машинного обучения, поэтому любые книги по науке о данных или машинному обучению полезны. Есть и другие книги по RL, которые тоже очень хороши и должны раскрывать новые перспективы. А если вы хотите заправиться по полной, рекомендую улучшить свои общие навыки в области программного обеспечения и инженерии.

Для того чтобы закрепить все, чему вы научились, вам нужно это использовать. Опыт — это один из способов, о котором я уже упоминал, но наставничество, преподавание и общее представление — еще несколько отличных способов учиться. Если вам нужно что-то объяснить кому-то другому, особенно формально, например в формате презентации, это заставит вас консолидировать свои идеи и запомнить свои модели и обобщения. Остерегайтесь ловушек, думая, что вы недостаточно опытны, чтобы говорить об этом. Именно вы, вы являетесь экспертом в том опыте, который у вас был. Весьма вероятно, что некоторые из ваших опытов могут быть обобщены в RL, чтобы помочь применять, объяснять или преподавать эти концепции так, как никто другой не смог бы. Дайте себе волю!

## Теперь ваша очередь

Итак, теперь дело за вами.

Надеюсь, вам уже известно, что больше ресурсов ждут вас на сайте[1] к этой книге. А также я жду обратной связи. Я хочу услышать о проектах, над которыми вы работаете, о ваших проблемах, ваших советах и хитростях, а также о ваших идеях относительно будущего RL. Я особенно хочу поговорить с вами, если вы думаете о применении RL в промышленности или боретесь с конкретной задачей. Я могу облегчить вашу жизнь с помощью решений, советов и оценок, чтобы сократить усилия, снизить риски и повысить производительность. Мои полные контактные данные напечатаны в *разд. "Об авторе"*. Удачи!

# Дополнительные материалы для чтения

- Отладка.
  - Agans D. J. Debugging: the 9 indispensable rules for finding even the most elusive software and hardware problems. — AMACOM, 2002.

[1] См. **https://rl-book.com/**.

- Fowler M. Refactoring: improving the design of existing code. — Addison-Wesley Professional, 2018.
- Feathers M. C. Working effectively with legacy code. — Prentice Hall PTR, 2004.

♦ Отладка глубокого обучения.

- Goodfellow I., Bengio Y., Courville A. Deep learning. — MIT Press, 2016.

♦ Отладка RL.

- Великолепный тред с Reddit, который дал жизнь многому из моей книги [11].
- Потрясающая презентация Джона Шульмана о гайках и болтах [12].
- Полезный опыт Мэтью Ратца [13].

♦ Направления исследований.

- Многие мои идеи были вдохновлены работой Дулака-Арнольда и соавт. [14].
- Офлайн-RL [15].
- Лично мне очень понравилась книга Макса Тегмарка о будущем искусственного интеллекта [16].

# Использованные источники

[1] Engstrom L., Ilyas A., Santurkar S. et al. Implementation matters in deep policy gradients: a case study on PPO and TRPO // ArXiv:2005.12729. — 2020. — May. — URL: **https://oreil.ly/8DR7T**.

[2] Ibid.

[3] Cobbe K. et al. Reinforcement learning with prediction-based rewards // OpenAI. — 2018. — 31 October. — URL: **https://oreil.ly/9VV5z**.

[4] Lessons learned reproducing a deep reinforcement learning paper. — 2020. — URL: **https://oreil.ly/OPnQm**.

[5] Ng A. Nuts and bolts of applying deep learning. — 2016. — URL: **https://oreil.ly/qEz3Z**.

[6] Guo X., Singh S., Lee H. et al. Deep learning for real-time atari game play using offline monte-carlo tree search planning // Advances in Neural Information Processing Systems 27, ed. by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, K. Q. Weinberger. Curran Associates, Inc., 2014. — P. 3338–3346. — URL: **https://oreil.ly/dl-az**.

[7] Hype cycle for data science and machine learning, 2019 // Gathner : [site]. — 2020. — URL: **https://oreil.ly/s7kEo**.

[8] Gartner top 10 trends in data and analytics for 2020// Gathner : [site]. — 2020. — URL: **https://oreil.ly/nYjE_**.

[9] Gartner says AI augmentation will create $2.9 trillion of business value in 2021// Gathner : [site]. — 2020. — URL: **https://oreil.ly/OjbkG**.

[10] Too many AI researchers think real-world problems are not relevant // MIT Technology Review. — 2020. — URL: **https://oreil.ly/B0k1N**.

[11] Deep reinforcement learning practical tips // R/Reinforcementlearning. — 2020. — URL: **https://oreil.ly/rSrgg**.

[12] Deep RL bootcamp lecture 6: nuts and bolts of deep rl experimentation. — 2017. — URL: **https://oreil.ly/PjuAc**.

[13] Lessons learned reproducing a deep reinforcement learning paper. — 2020. — URL: **https://oreil.ly/SNSVk**.

[14] Dulac-Arnold G., Mankowitz D., Hester T. Challenges of real-world reinforcement learning // ArXiv:1904.12901. — 2019. — April. — URL: **https://oreil.ly/WKoG5**.

[15] Levine S., Kumar A., Tucker G., Fu J. Offline reinforcement learning: tutorial, review, and perspectives on open problems // ArXiv: 2005.01643. — 2020. — May. — URL: **https://oreil.ly/Kae6m**.

[16] Tegmark M. Life 3.0: being human in the age of artifcial intelligence. — Penguin UK, 2017.

## ПРИЛОЖЕНИЕ 1

# Градиент логистической политики для двух действий

Уравнение 5.6 представляет собой политику для двух действий. Для того чтобы обновить политику, мне нужно рассчитать градиент натурального логарифма политики (см. уравнение 5.4). Я представил это в уравнении П1.1. Вы можете выполнить дифференцирование несколькими различными способами в зависимости от того, какие алготирмы преобразований вы используете, поэтому результат может выглядеть по-разному, даже если он обеспечивает один и тот же результат.

**Уравнение П1.1. Градиент логистической политики для двух действий**

$$\nabla \ln \pi(a \mid s, \boldsymbol{\theta}) = \begin{bmatrix} \frac{\delta}{\delta \boldsymbol{\theta}_0} \ln\left(\frac{1}{1+e^{-\boldsymbol{\theta}_0^T s}}\right) \\ \frac{\delta}{\delta \boldsymbol{\theta}_1} \ln\left(\frac{1}{1+e^{-\boldsymbol{\theta}_1^T s}}\right) \end{bmatrix}.$$

Я вычисляю градиенты каждого действия независимо, и мне будет проще, если я реорганизую логистическую функцию, как в уравнении П1.2.

**Уравнение П1.2. Рефакторинг логистической функции**

$$\begin{aligned} \pi(x) \doteq \frac{1}{1+e^{-x}} &= \frac{e^x}{e^x\left(1+e^{-x}\right)} = \\ &= \frac{e^x}{e^x + e^x e^{-x}} = \\ &= \frac{e^x}{e^x + e^{x-x}} = \\ &= \frac{e^x}{e^x + e^0} = \\ &= \frac{e^x}{e^x + 1} = \\ &= \frac{e^x}{1+e^x}. \end{aligned}$$

Производная от преобразованной логистической функции для действия 0 показана в уравнении П1.3.

**Уравнение П1.3. Дифференцирование действия 0**

$$\frac{\delta}{\delta\boldsymbol{\theta}_0}\ln\pi_0\left(\boldsymbol{\theta}_0^T s\right)=\frac{\delta}{\delta\boldsymbol{\theta}_0}\ln\left(\frac{e^{\boldsymbol{\theta}_0^T s}}{1+e^{\boldsymbol{\theta}_0^T s}}\right)=$$

$$=\frac{\delta}{\delta\boldsymbol{\theta}_0}\ln e^{\boldsymbol{\theta}_0^T s}-\frac{\delta}{\delta\boldsymbol{\theta}_0}\ln\left(1+e^{\boldsymbol{\theta}_0^T s}\right)=$$

$$=\frac{\delta}{\delta\boldsymbol{\theta}_0}\boldsymbol{\theta}_0^T s-\frac{\delta}{\delta\boldsymbol{\theta}_0}\ln\left(1+e^{\boldsymbol{\theta}_0^T s}\right)=$$

$$=s-\frac{\delta}{\delta\boldsymbol{\theta}_0}\ln\left(1+e^{\boldsymbol{\theta}_0^T s}\right)=$$

$$=s-\frac{\delta}{\delta\boldsymbol{\theta}_0}\ln u=$$

$$=s-\frac{1}{u}\frac{\delta}{\delta\boldsymbol{\theta}_0}u=$$

$$=s-\frac{1}{1+e^{\boldsymbol{\theta}_0^T s}}\frac{\delta}{\delta\boldsymbol{\theta}_0}\left(1+e^{\boldsymbol{\theta}_0^T s}\right)=$$

$$=s-\frac{1}{1+e^{\boldsymbol{\theta}_0^T s}}\frac{\delta}{\delta v}\left(e^{v}\right)\frac{\delta}{\delta\boldsymbol{\theta}_0}v=$$

$$=s-\frac{1}{1+e^{\boldsymbol{\theta}_0^T s}}e^{\boldsymbol{\theta}_0^T s}s=$$

$$=s-\frac{se^{\boldsymbol{\theta}_0^T s}}{1+e^{\boldsymbol{\theta}_0^T s}}=$$

$$=s-s\pi\left(\boldsymbol{\theta}_0^T s\right),$$

где

$u=1+e^{\boldsymbol{\theta}_0^T s}$, $v=\boldsymbol{\theta}_0^T s$.

Метод расчета производной политики для действия 1 показан в уравнении П1.4. Обратите внимание, что вывод ближе к концу совпадает с уравнением П1.3.

**Уравнение П1.4. Дифференцирование действия 1**

$$\frac{\delta}{\delta\boldsymbol{\theta}_1}\ln\pi_1\left(\boldsymbol{\theta}_1^T s\right)=\frac{\delta}{\delta\boldsymbol{\theta}_1}\ln\left(1-\frac{e^{\boldsymbol{\theta}_1^T s}}{1+e^{\boldsymbol{\theta}_1^T s}}\right)=$$

$$=\frac{\delta}{\delta\boldsymbol{\theta}_1}\ln\left(\frac{1+e^{\boldsymbol{\theta}_1^T s}}{1+e^{\boldsymbol{\theta}_1^T s}}-\frac{e^{\boldsymbol{\theta}_1^T s}}{1+e^{\boldsymbol{\theta}_1^T s}}\right)=$$

$$=\frac{\delta}{\delta\boldsymbol{\theta}_1}\ln\left(\frac{1+e^{\boldsymbol{\theta}_1^T s}-e^{\boldsymbol{\theta}_1^T s}}{1+e^{\boldsymbol{\theta}_1^T s}}\right)=$$

$$=\frac{\delta}{\delta\boldsymbol{\theta}_1}\ln\left(\frac{1}{1+e^{\boldsymbol{\theta}_1^T s}}\right)=$$

$$=\frac{\delta}{\delta\boldsymbol{\theta}_1}\ln 1-\frac{\delta}{\delta\boldsymbol{\theta}_1}\ln\left(1+e^{\boldsymbol{\theta}_1^T s}\right)=$$

$$=-\frac{\delta}{\delta\boldsymbol{\theta}_1}\ln\left(1+e^{\boldsymbol{\theta}_1^T s}\right)=$$

$$=\ldots \text{ то же самое, что для производной для } \pi_0 =$$

$$=-s\pi\left(\boldsymbol{\theta}_1^T s\right).$$

Это приводит к градиенту, показанному в уравнении П1.5.

**Уравнение П1.5. Логистическая политика**

$$\nabla\ln\pi(a\mid s,\boldsymbol{\theta})=\begin{bmatrix} s-s\pi\left(\boldsymbol{\theta}_0^T s\right) \\ -s\pi\left(\boldsymbol{\theta}_1^T s\right)\end{bmatrix}.$$

# Градиент политики softmax

Вывод градиента политики softmax показан в уравнении П2.1. Обратите внимание, что эта форма аналогична логистическим градиентам в *приложении 1*.

**Уравнение П2.1. Градиент политики softmax**

$$\nabla_{\theta} \ln\left(\theta^{T} s\right) = \nabla_{\theta} \ln \frac{e^{\theta^{T} s}}{\sum_{a} e^{\theta_{a}^{T} s}} =$$

$$= \nabla_{\theta} \ln e^{\theta^{T} s} - \nabla_{\theta} \ln \sum_{a} e^{\theta_{a}^{T} s} =$$

$$= \nabla_{\theta} \theta^{T} s - \nabla_{\theta} \ln \sum_{a} e^{\theta_{a}^{T} s} =$$

$$= s - \nabla_{\theta} \ln \sum_{a} e^{\theta_{a}^{T} s} =$$

$$= s - \frac{\nabla_{\theta} \sum_{a} e^{\theta_{a}^{T} s}}{\sum_{a} e^{\theta_{a}^{T} s}} =$$

$$= s - \frac{\sum_{a} s e^{\theta_{a}^{T} s}}{\sum_{a} e^{\theta_{a}^{T} s}} =$$

$$= s - \sum_{a} s \pi\left(\theta^{T} s\right).$$

# Предметный указатель


## A

Action-value function 75
Actor-critic using kronecker-factored trust regions (ACKTR) 213
Actor-critic with experience replay (ACER) 212
Advantage actor-critic (A2C) 160, 194, 331
Artificial intelligence (AI) 31
Artificial neural network (ANN) 118
Asynchronous advantage actor-critic (A3C) 331

## B

Batch-constrained deep Q-learning (BCQ) 141
Bootstrapped DQN (BDQN) 139

## C

CartPole 125
Click-through rate (CTR) 98
Clipped double Q-learning (CDQ) 189
Coach 124
Contextual MDP (CMDP) 249
Convolutional neural network (CNN) 119

## D

Decentralized distributed proximal policy optimization (DD-PPO) 333
Dec-HDRQN 264
Deep belief networks (DBN) 119
Deep deterministic policy gradients (DDPG) 184
Deep learning (DL) 184
Deep Q-learning from demonstrations (DQfD) 268
Deep Q-network (DQN) 122, 330
Deterministic policy gradients (DPG) 183
Directed acyclic graph (DAG) 120, 152, 245
Diversity is all you need (DIAYN) 254
Dynamic programming (DP) 87

## E

Echo state networks (ESN) 120
Explainable artificial intelligence (XAI) 344
Extensive-form game (EFG) 258

## F

First in, first out (FIFO) 123

## G

Gated recurrent units (GRU) 120
Generalized off-policy actorcritic (Geoff-PAC) 214
Generalized policy iteration (GPI) 81
Gorila 330
Gradient-temporal-difference (GTD) 181

## H

Hierarchical reinforcement learning (HRL) 251, 273
HRL with advantage-based rewards (HAAR) 255

## I

Importance weighted actor-learner architecture (IMPALA) 334
Independent and identically distributed (IID) 122
Inverse RL (IRL) 268

## K

Keras 121
Key performance indicator (KPI) 53, 98
Kronecker-factored approximation (K-FAC) 213

## L

Long short-term memory (LSTM) 120

## M

Machine learning (ML) 21
Markov decision process (MDP) 33, 47, 53, 62
Markov game (MG) 258
Mean squared error (MSE) 124
MLOps 347
Monte Carlo (MC) 78
Multi-agent reinforcement learning (MARL) 256
◊ проблемы 265
Multilayer perceptrons (MLP) 119
MXNet 120

## N

Natural policy gradients (NPG) 197
Neural fitted Q-iteration (NFQ) 129

## O

Observe, orient, decide, act (OODA) 293
Off-policy actor-critic (Off-PAC) 181
Opposition-based RL 104

## P

Parallel advantage actor critic (PAAC) 332
Partially observable Markov decision process (POMDP) 246
path consistency learning (PCL) 228
Probably approximately correct (PAC) 103
Proximal policy optimization (PPO) 201
PyTorch 120

## Q

Q-обучение 90
◊ быстрое 113
◊ глубокое на основе демонстраций 268
◊ глубокое с пакетными ограничениями 141
◊ двойное 102
◊ имитационное мягкое 268
◊ мягкое 233
◊ ограниченное двойное 189
◊ отложенное 103
Q-сеть
◊ глубокая 122
▫ бутстрапированная 139
◊ клонирование 123
Q-функция 75

## R

Random distillation network 312
Real-time bidding (RTB) 98
Recurrent neural network (RNN) 120, 248
Reinforcement learning (RL) 21, 32, 146
◊ фреймворк 326
Restricted Boltzmann machines (RBM) 119

## S

SARSA 92
◊ онлайн-вариант 109
Scalable and efficient deep RL with accelerated central inference (SEED) 335
Soft actor-critic (SAC) 223
Soft Q imitation learning (SQIL) 268
soft Q-learning (SQL) 233
Softmax 119
◊ политика 151
state-value function 71
Stochastic action set (SAS) 249

## T

Target policy smoothing (TPS) 189
Temporal-difference (TD) 87
TensorFlow 120
Trust region policy optimization (TRPO) 198
Twin delayed deep deterministic policy gradients (TD3) 188
◊ реализация 190

## U

Upper-confidence-bound (UCB) 61, 139

## V

Vanilla gradient ascent 147
Variational information maximizing exploration (VIME) 310

## А

Автоэнкодер 119
◊ условный вариационный 141
Агент 33, 62
Актор 47, 160
Актор — критик 160
Алгоритм
◊ ε-жадный 57
◊ n-шаговый 105
▫ в распределенной среде 108
◊ Q(λ) Уоткинса 112
◊ REINFORCE 181
◊ Retrace (λ) 212
◊ актор — критик 160
▫ вне политики 181
▫ использующий доверительные области по фактору Кронекера (ACKTR) 213
▫ мягкий 223
▫ с опытом воспроизведения (ACER) 212
▫ с преимуществом
• асинхронный (A3C) 331
• параллельный (PAAC) 332
▫ с преимуществом (A2C) 160, 194
◊ бандита 57
◊ безмодельный 37
◊ градиентно-временных различий 181
◊ жадный-GQ 180
◊ жесткий 223
◊ Монте-Карло на основе политики 79
◊ на основе
▫ имитации 40
▫ моделей 37
▫ политики 40, 93
◊ нейронной аппроксимации 129
◊ с верхним доверительным интервалом (UCB) 61
◊ с одним агентом 260
◊ ценностный 40
Архитектура "актор — ученик", взвешенная по значимости 334
Атака 359

## Б

Библиотека глубокого обучения с открытым кодом 120
Блок рекуррентный 120
Бутстрэппинг 87
Буфер воспроизведения опыта 106

## В

Вес 42, 118
Вознаграждение 33, 46, 62
◊ выбор 101
Воспроизведение опыта 122
Выборка по значимости 176

## Г

Гибкость 348
Гиперпараметр
◊ поиск 239
Гистерезис с временной разницей 263
Градиент политики
◊ глубокий детерминированный 184
◊ глубокий мягкий 227
◊ дважды отложенный глубокий детерминированный 188
◊ детерминированный 183
◊ естественный 197
Граница пессимистическая 201
Граф
◊ ориентированный ациклический 120, 152, 245
◊ переходов 66

## Д

Действие 32
◊ стохастическое 249
Децентрализованная распределенная проксимальная оптимизация политики 333
Дивергенция Кульбака — Лейблера 196
Дискретизация возможности 303
Дифференцирование автоматическое 152
Доходность 70

## З

Заражение 359

## И

Игра
◊ в развернутой форме 258
◊ Маркова 258
Иерархия потребностей 348
Инжиниринг состояния 297
Интеллект искусственный 31
Информация скрытая 119

Исследование с максимизацией вариационной информации 310
Итерация по стратегии, обобщенная 81

## К

Карта значимости 345
Квантиль 215
Клонирование поведения 267
Кодирование фиктивное 150
Количество показов 98
Контейнер программный 96
Концепция предустановок 125
Коэффициент
◊ выборки важности 201
◊ дисконтирования 71
◊ плотности 214
Критик 47, 160
◊ обобщенный внеполитических субъектов 214

## М

Максимизация энтропии 222
Марковский процесс принятия решений 33, 47, 62
Масштабирование 96
Масштабируемость 347
Матрица перехода 66
Машина Больцмана ограниченная 119
Машинное обучение
◊ парадигма 21
◊ операции 347
Мета-RL 272
Метаобучение 271
Метод
◊ временных различий 214
◊ градиента политики 158
◊ динамического программирования 87
◊ итерации по ценности 82
◊ Монте-Карло 78
◊ накопления трассировок 113
◊ табличный 92
◊ черного ящика 344
Моделирование 31
Модель
◊ подписки 166
◊ Рескорла — Вагнера 42

## Н

Наблюдаемость частичная 246
Наблюдение 62
Награда 53
Надежность 347
Нейрон 118
◊ искусственный 45
Непрерывная доставка 353
Непрерывная интеграция 353

## О

Обновление политики отложенное 188
Обоснование контрфактическое 214
Обусловливание 41
Обучение
◊ без сброса 291
◊ вероятно приближённо корректное 103
◊ генеративное состязательное имитационное 268
◊ глубокое 184
◊ децентрализованное 260, 263
◊ машинное 31
◊ методом проб и ошибок 47
◊ непрерывное 250
◊ параллельное 290
◊ передающее 170
◊ по учебной программе 270
◊ представлениям 297
◊ с временными различиями 87
◊ с подкреплением 21, 29, 32, 146
▫ иерархическое 251
• с вознаграждением на основе преимуществ 255
▫ обратное 268
▫ остаточное 271
▫ на основе противодействия 104
◊ согласованности пути 228
◊ трансферное 272
◊ централизованное 260
◊ эффективность 338
Обучение с подкреплением
◊ иерархическое 273
◊ мультиагентное 256, 273
◊ общая архитектура 330
Общая архитектура обучения с подкреплением 330
Объяснимость 344
Объяснимый искусственный интеллект 344
Онлайн-сеть 123

Оператор ожидания 72
Операции машинного обучения 347
Оптимизация политики проксимальная 201
Отжиг 61, 233
Оценка
◊ политики 81
◊ ценности 42
Ошибка
◊ среднеквадратичная 124
◊ алгоритма предсказателя 339

## П

Память долгая краткосрочная 120
Переобучение 73
Персептрон многослойный 119
Поведенческие программы Осбанда 343
Подход SARSA 92
Подъем градиентный упрощенный 147
Показатель эффективности ключевой 53, 98
Политика 38, 70
◊ softmax 151
◊ децентрализованная распределенная проксимальная оптимизация 333
◊ оптимальная 76
◊ поведенческая 178
◊ стохастическая 146
◊ целевая 178
◊ эффективность 338
Правило обновления весов 44
Приближение
◊ информационной матрицы Фишера 213
◊ с факторизацией Кронекера 213
Прирост информации 310
Проблема предсказания 44
Прогноз одношаговый 105
Произведение скалярное 42
Проклятие размерности 250
Процесс марковский 63
◊ частично наблюдаемый 69, 246
Процесс принятия решений марковский
◊ контекстный 249
◊ регуляризованный 250
◊ с изменяющимися действиями 249
◊ частично наблюдаемый 69, 246

## Р

Развертывание 90, 346
Регрет 339
Регулировка температуры автоматическая 224
Регуляризатор 250
Рейтинг
◊ дисконтирования 71
◊ кликов 98
Рефакторинг 152
Руководство экспертное 267

## С

Свертка 119
Сглаживание целевой политики 189
Сеть
◊ глубокая, доверия 119
◊ дуэльная 133
◊ зашумленная 133
◊ локальная 187
◊ нейронная
▫ глубокая 119
▫ искусственная 118
▫ рекуррентная 120, 248
▫ сверточная 119
◊ случайной дистилляции 312
◊ целевая 123
◊ эхо-состояний 120
Слой 118
Совершенствование политики 81
Состояние 62
◊ терминальное 70
Спуск
◊ градиентный 124
◊ Штейна градиентный вариационный 224
Среднее по совокупности 176
Стратегия 38

## Т

Таблица переходов 65
Температура 224
Торги в режиме реального времени 98
Трассировка соответствия 111
Трюк с репараметризацией 224

## У

Уравнение оптимальности Беллмана 77

## Ф

Фреймворк
◊ MARL 257
◊ реализации глубокого обучения 120
Функция
◊ softmax 61, 119
◊ активации нелинейная 118
◊ аппроксимационная 117
◊ значения действия 76
◊ значения состояния 76
◊ логистическая 150
◊ отображения 42
◊ преимущества 133, 195
◊ сигмоидальная 150
◊ ценности действия 75
◊ ценности состояния 71

## Ш

Шлюз 120

## Э

Энтропия
◊ максимальная 222
◊ Шеннона 221
Эпизод 70, 95
Эффективность
◊ обучения 338
◊ политики 338

O'REILLY®

# Обучение с подкреплением для реальных задач

Обучение с подкреплением (Reinforcement Learning, RL) — это парадигма, способная обеспечить один из крупнейших прорывов в искусственном интеллекте на протяжении текущего десятилетия. Алгоритмы RL способны обучаться на взаимодействии с окружающей средой как на множестве данных, тем самым достигая произвольно сформулированных целей. Эти поразительные разработки не стеснены ограничениями, характерными для традиционных алгоритмов машинного обучения. Книга, которую вы держите в руках, поможет профессионалам, занимающимся искусственным интеллектом и наукой о данных, разобраться, как обучать алгоритм методом подкрепления и превращать машину в подлинно самообучаемую систему.

Тема RL излагается во всех подробностях: от базовых составляющих до ультрасовременных практик. Вы узнаете, каково состояние RL в настоящий момент, каким образом оно применяется в промышленности, изучите многочисленные алгоритмы, а также почерпнете много полезного из рассмотренных конкретных примеров развертывания RL в индустрии. Но это не сборник рецептов. Автор не избегает сложной математики и рассчитывает, что читатель знаком с машинным обучением.

*«Проделана превосходная работа по изложению контекста, ландшафта и возможностей использования RL в обработке данных, что прямо способствует развитию современного бизнеса».*

**— Дэвид Арончик**, соавтор технологии Kubeflow

*«Незаменимая книга для всех, кто желает применять обучение с подкреплением для решения реальных задач. Читатель узнает все от азов до новейших практик; книга изобилует практическими примерами и подробными объяснениями».*

**— Дэвид Фостер**, совладелец компании Applied Data Science Partners

*«Отличная книга. Проверенный эволюцией метод обучения наконец-то пополняет стандартный инструментарий программиста».*

**— Дэнни Лэнж**, старший вице-президент по искусственному интеллекту в компании Unity Technologies

- **Освойте обучение с подкреплением и узнайте, как его алгоритмы применяются при решении задач**
- **Основательно проработайте фундаментальные темы RL, рассмотренные в книге: марковские процессы принятия решений, динамическое программирование и обучение с применением временной разницы**
- **Разберитесь в различных методах формулирования ценностных функций и градиентов политик**
- **Применяйте продвинутые RL-решения, в частности метаобучение, иерархическое обучение, мультиагентное и имитационное обучение**

**Фил Уиндер** — междисциплинарный инженер, эксперт и автор онлайн-курсов на платформе O'Reilly. Возглавляет компанию Winder Research, оказывающую консультации в области науки о данных (data science) для облачно-ориентированных приложений. Компания помогает оптимизировать процессы, связанные с обработкой данных, а также обслуживает платформы и продукты, работающие в этой нише. Автор имеет степени PhD и MEng в электротехнике, полученные в Университете Халла.

ISBN 978-5-9775-6885-2

191036, Санкт-Петербург,
Гончарная ул., 20
Тел.: (812) 717-10-50,
339-54-17, 339-54-28
E-mail: mail@bhv.ru
Internet: www.bhv.ru

bhv®