

Оглавление

Об авторе	14
О научных редакторах.....	14
Предисловие	16
Для кого написана эта книга	16
Что в книге.....	16
Необходимое программное обеспечение	18
Загрузка файлов с примерами кода	19
От издательства	19
Глава 1. Введение в обучение с подкреплением.....	20
Что такое RL?	20
Алгоритм RL	22
Чем RL отличается от других парадигм машинного обучения	23
Элементы RL.....	24
Агент.....	24
Функция политики.....	24
Функция ценности.....	24
Модель.....	25
Интерфейс агента со средой.....	25
Типы сред RL.....	27
Детерминированная среда	27
Стохастическая среда.....	27
Среда с полной информацией	27

Среда с неполной информацией.....	27
Дискретная среда.....	28
Непрерывная среда.....	28
Эпизодические и неэпизодические среды.....	28
Одноагентные и многоагентные среды.....	28
Платформы RL.....	28
OpenAI Gym и Universe.....	29
DeepMind Lab.....	29
RL-Glue.....	29
Проект Malmo.....	29
ViZDoom.....	30
Практическое применение RL.....	30
Образование.....	30
Медицина и здравоохранение.....	30
Производство.....	31
Управление ресурсами.....	31
Финансы.....	31
Обработка естественного языка и машинное распознавание образов.....	31
Итоги.....	32
Вопросы.....	32
Дополнительные источники.....	32
Глава 2. Знакомство с OpenAI и TensorFlow.....	33
Подготовка системы.....	34
Установка Anaconda.....	34
Установка Docker.....	35
Установка OpenAI Gym и Universe.....	36
OpenAI Gym.....	39
Базовое моделирование.....	39
Робот учится ходить.....	41
OpenAI Universe.....	44
Построение бота для видеоигры.....	44
TensorFlow.....	48
Переменные, константы и заместители.....	49
Граф вычислений.....	50

Сеансы	51
TensorBoard	52
Итоги.....	55
Вопросы	56
Дополнительные источники.....	56
Глава 3. Марковский процесс принятия решений и динамическое программирование	57
Марковские цепи и марковские процессы.....	57
Марковский процесс принятия решений	59
Награды и возврат	60
Эпизодические и непрерывные задачи	61
Поправочный коэффициент.....	61
Функция политики.....	62
Функция ценности состояния	62
Функция ценности состояния/действия (Q-функция)	63
Уравнение Беллмана и оптимальность.....	64
Вывод уравнения Беллмана для функции ценности и Q-функции	65
Решение уравнения Беллмана	68
Динамическое программирование	68
Решение задачи о замерзшем озере	76
Итерация по ценности.....	78
Итерация по политикам.....	84
Итоги.....	87
Вопросы	88
Дополнительные источники.....	88
Глава 4. Методы Монте-Карло в играх.....	89
Метод Монте-Карло	89
Оценка значения π методом Монте-Карло	90
Прогнозирование методом Монте-Карло.....	94
Метод Монте-Карло с первым посещением.....	96
Метод Монте-Карло с каждым посещением	96
Игра в блек-джек по стратегии Монте-Карло.....	96

Управление методом Монте-Карло	105
MC-ES.....	106
Метод Монте-Карло с привязкой к политике.....	108
Метод Монте-Карло без привязки к политике.....	111
Итоги.....	112
Вопросы	113
Дополнительные источники.....	113
Глава 5. Обучение на основе временных различий	114
Обучение на основе временных различий	114
Прогнозирование на основе временных различий	115
TD-управление	118
Q-обучение	119
SARSA	127
Решение задачи о такси методом SARSA	131
Различия между Q-обучением и SARSA.....	133
Итоги.....	135
Вопросы	135
Дополнительные источники.....	135
Глава 6. Задача о многоруком бандите.....	136
Задача MAB	137
Эпсилон-жадная стратегия.....	139
Алгоритм softmax-исследования	140
Алгоритм верхней границы доверительного интервала	141
Алгоритм выборки Томпсона	145
Практические применения MAB	147
Выбор подходящего рекламного баннера с использованием MAB	148
Контекстные бандиты.....	151
Итоги.....	151
Вопросы	152
Дополнительные источники.....	152

Глава 7. Основы глубокого обучения.....	153
Искусственные нейроны	154
ANN	155
Входной слой	156
Скрытый слой.....	157
Выходной слой	157
Функции активации	157
Подробнее об ANN.....	159
Градиентный спуск.....	162
Нейросети в TensorFlow	168
RNN	171
Обратное распространение во времени.....	174
RNN с долгой краткосрочной памятью.....	176
Генерирование текстов песен посредством LSTM RNN.....	178
Сверточные нейросети	182
Сверточный слой.....	182
Слой подвыборки	188
Полносвязный слой	188
Архитектура CNN.....	189
Классификация предметов одежды с использованием CNN.....	189
Итоги.....	196
Вопросы	196
Дополнительные источники.....	196
Глава 8. Игры Atari с использованием Deep Q Network	197
Что такое DQN?	197
Архитектура DQN.....	199
Сверточная сеть	199
Воспроизведение опыта	200
Целевая сеть.....	201
Нормализация наград.....	202
Понимание алгоритма	202
Построение агента для игр Atari	203
Двойная сеть DQN	211

Приоритетное воспроизведение опыта	212
Архитектура дуэльных сетей.....	213
Итоги.....	215
Вопросы	215
Дополнительные источники.....	215
Глава 9. Игра Doom в глубокой рекуррентной Q-сети.....	216
DRQN.....	216
Архитектура DRQN	218
Обучение агента для игры в Doom	219
Базовая игра Doom	220
Doom с DRQN	222
DARQN.....	232
Архитектура DARQN	232
Итоги.....	233
Вопросы	234
Дополнительные источники.....	234
Глава 10. Асинхронная преимущественная сеть «актор-критик»	235
Асинхронный преимущественный алгоритм «актор-критик»	236
Три «А».....	236
Архитектура АЗС	237
Как работает АЗС	238
Подъем на гору с использованием АЗС	239
Визуализация в TensorBoard	247
Итоги.....	250
Вопросы	250
Дополнительные источники.....	250
Глава 11. Градиенты политик и оптимизация	251
Градиент политики	252
Посадка на Луну с градиентами политик	252
Глубокий детерминированный градиент политики.....	257
Раскачивание маятника.....	259

Оптимизация политики доверительной области	266
Оптимизация ближайшей политики	270
Итоги.....	272
Вопросы	273
Дополнительные источники.....	273
Глава 12. «Автогонки» с использованием DQN.....	274
Функции-обертки среды	274
Дуэльная сеть.....	278
Память воспроизведения.....	280
Обучение сети.....	281
«Автогонки»	287
Итоги.....	290
Вопросы	291
Дополнительные источники.....	291
Глава 13. Последние достижения и следующие шаги	292
Агенты, дополненные воображением.....	292
Обучение на человеческих предпочтениях.....	297
Глубокое Q-обучение на примере демонстраций	298
Ретроспективное воспроизведение опыта.....	299
Иерархическое обучение с подкреплением.....	301
Декомпозиция функции ценности MAXQ.....	302
Инвертированное обучение с подкреплением.....	305
Итоги.....	306
Вопросы	307
Дополнительные источники.....	307
Ответы	308

6

Задача о многоруком бандите

В предыдущих главах были представлены фундаментальные концепции **обучения с подкреплением (RL)** и некоторые алгоритмы RL, а также моделирование RL-задач в форме **марковского процесса принятия решений (MDP)**. Также были описаны различные алгоритмы (с моделью и без модели), используемые для решения MDP. В этой главе будет рассмотрена одна из классических задач RL — **задача о многоруком бандите**, или **МAB** (multi-armed bandit). Вы узнаете, что собой представляет эта задача, как она решается при помощи разных алгоритмов и как выбрать наиболее подходящий рекламный баннер, который будет получать наибольшее число переходов, средствами МAB. Также будет описан так называемый контекстный бандит, широко применяемый при построении рекомендационных систем.

В этой главе рассматриваются следующие темы:

- Задача о многоруком бандите (МAB).
- Эпсилон-жадный алгоритм.
- Алгоритм softmax-исследования.
- Алгоритм верхней границы доверительного интервала.
- Алгоритм выборки Томпсона.
- Практические применения МAB.
- Выбор подходящего рекламного баннера с использованием МAB.
- Контекстные бандиты.

Задача MAB

Задача о многоруком бандите (MAB) — одна из классических задач в RL. Многорукий бандит — игровой автомат; азартная игра, в которой игрок нажимает на рычаг и получает награду (выигрыш) на основании случайно сгенерированного распределения вероятностей. Отдельный автомат называется «одноруким бандитом»; если же таких автоматов несколько, это называется «многоруким бандитом» или k -руким бандитом.

Многорукий бандит выглядит так:



Так как каждый автомат дает выигрыш из собственного вероятностного распределения, наша цель — определить, какой автомат принесет максимальную накапливаемую награду за конкретный промежуток времени. Таким образом, на каждом временном кванте t агент выполняет действие a_t , то есть нажимает рычаг игрового автомата и получает награду r_t , при этом цель агента заключается в максимизации накапливаемой награды.

Определим ценность руки $Q(a)$ как среднюю награду, получаемую при нажатии рычага:

$$Q(a) = \frac{\text{Сумма наград, полученных на руке}}{\text{Общее количество активизаций руки}}.$$

Таким образом, *оптимальной* будет называться рука, обеспечивающая максимальную накапливаемую награду:

$$Q(a^*) = \text{Max}Q(a).$$

Цель агента — найти оптимальную руку и минимизировать потери, которые могут определяться как затраты на получение информации о том, какая из k рук является оптимальной. Как найти лучшую руку? Исследовать все руки или выбрать ту руку, которая уже дает максимальную накапливаемую награду? Мы снова сталкиваемся с дилеммой компромисса между исследованием и эксплуатацией. В этой главе будет показано, как эта дилемма решается с помощью различных стратегий исследования:

- эpsilon-жадная стратегия;
- softmax-исследование;
- алгоритм верхней границы доверительного интервала;
- алгоритм выборки Томпсона.

Прежде чем двигаться дальше, установите среды `bandit` в OpenAI Gym; для этого введите в терминале следующие команды:

```
git clone https://github.com/JKCooper2/gym-bandits.git
cd gym-bandits
pip install -e .
```

После установки импортируйте `gym` и `gym_bandits`:

```
import gym_bandits
import gym
```

Теперь нужно инициализировать среду; мы будем использовать МАВ с десятью руками:

```
env = gym.make("BanditTenArmedGaussian-v0")
```

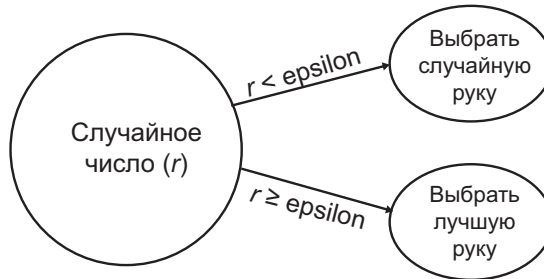
Размер пространства действий равен 10, так как у бандита 10 рук. Соответственно, команда

```
env.action_space
```

выводит следующий результат:

Эпсилон-жадная стратегия

Вы уже неоднократно сталкивались с эпсилон-жадной стратегией; здесь выбирается либо лучшая рука с вероятностью $1 - \text{epsilon}$, либо случайная рука с вероятностью epsilon :



Посмотрим, как выбрать лучшую руку с применением эпсилон-жадной стратегии:

1. Инициализировать все переменные:

```
# Количество раундов (итераций)
num_rounds = 20000

# Количество нажатий на рычаг
count = np.zeros(10)

# Сумма наград для каждой руки
sum_rewards = np.zeros(10)

# Q, то есть средняя награда
Q = np.zeros(10)
```

2. Затем определить функцию `epsilon_greedy`:

```
def epsilon_greedy(epsilon):
    rand = np.random.random()
    if rand < epsilon:
        action = env.action_space.sample()
    else:
        action = np.argmax(Q)
    return action
```

3. Перейти к нажатию рычагов:

```
for i in range(num_rounds):
    # Выбрать руку с применением эпсилон-жадной политики
    arm = epsilon_greedy(0.5)
    # Получить награду
    observation, reward, done, info = env.step(arm)
    # Обновить счетчик для этой руки
    count[arm] += 1
    # Включить в сумму наград от руки
    sum_rewards[arm] += reward
    # Вычислить Q - среднюю награду от руки
    Q[arm] = sum_rewards[arm] / count[arm]

print( 'The optimal arm is {}'.format(np.argmax(Q)))
```

Результат выполнения:

The optimal arm is 3

Алгоритм softmax-исследования

Softmax-исследование, также называемое *исследованием Больцмана*, — еще одна стратегия, применяемая для нахождения оптимальной руки. В эпсилон-жадной стратегии все не лучшие руки считаются равноправными, а в softmax-исследовании рука выбирается на основании вероятности из распределения Больцмана. Эта вероятность определяется следующей формулой:

$$P_t(a) = \frac{\exp(Q_t(a) / \tau)}{\sum_{i=1}^n \exp(Q_t(i) / \tau)}.$$

Здесь τ — температурный коэффициент, который определяет, сколько случайных рук можно исследовать. При высоких значениях τ все руки будут исследоваться с равной вероятностью, а при низких значениях τ будут выбираться руки с высокой наградой. Последовательность действий выглядит так:

1. Инициализация переменных:

```
# Количество раундов (итераций)
num_rounds = 20000

# Количество нажатий на рычаг
count = np.zeros(10)
```

```
# Сумма наград для каждой руки
sum_rewards = np.zeros(10)

# Q, то есть средняя награда
Q = np.zeros(10)
```

2. Определение softmax-функции:

```
def softmax(tau):
    total = sum([math.exp(val/tau) for val in Q])
    probs = [math.exp(val/tau)/total for val in Q]
    threshold = random.random()
    cumulative_prob = 0.0
    for i in range(len(probs)):
        cumulative_prob += probs[i]
        if (cumulative_prob > threshold):
            return i
    return np.argmax(probs)
```

3. Переход к нажатию рычагов:

```
for i in range(num_rounds):
    # Выбрать руку с применением softmax-политики
    arm = softmax(0.5)
    # Получить награду
    observation, reward, done, info = env.step(arm)
    # Обновить счетчик для этой руки
    count[arm] += 1
    # Просуммировать награды от руки
    sum_rewards[arm] += reward
    # Вычислить Q - среднюю награду от руки
    Q[arm] = sum_rewards[arm]/count[arm]
print( 'The optimal arm is {}'.format(np.argmax(Q)))
```

Результат:

```
The optimal arm is 3
```

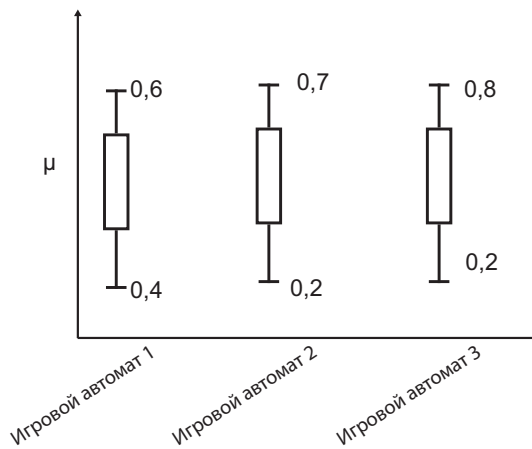
Алгоритм верхней границы доверительного интервала

Эпсилон-жадные и softmax-алгоритмы обеспечивают исследование случайных действий с определенной вероятностью. Исследование разных вариантов может привести к опробованию действий, вообще не несущих хорошей награды.

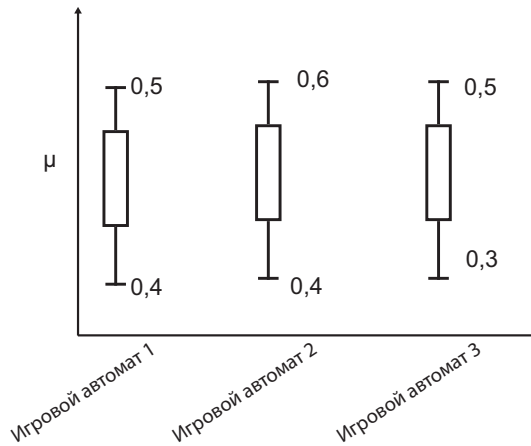
При этом не хотелось бы упустить те хорошие руки, которые принесли низкую награду в начальных итерациях. Для решения таких проблем существует алгоритм *верхней границы доверительного интервала* (*UCB*, Upper Confidence Bound), основанный на принципе оптимизма перед лицом неопределенности.

Алгоритм UCB помогает выбрать лучшую руку на основании доверительного интервала. Что это такое? Допустим, есть две руки. Мы опробуем оба варианта; первая рука обеспечивает награду 0,3, а вторая — 0,8. Тем не менее по одной попытке нельзя прийти к выводу, что рука 2 обеспечивает более высокую награду. Нужно попробовать нажать на рычаг несколько раз, вычислить среднее значение награды, полученной для каждой руки, и выбрать вариант с более высоким средним значением. Но как определить правильное среднее значение для каждой из этих рук? На помощь приходит *доверительный интервал* — интервал, в котором лежит средняя награда для руки. Если доверительный интервал руки 1 имеет вид $[0,2, 0,9]$, то это означает, что среднее значение для руки 1 лежит в этом интервале. Значение 0,2 называется *нижней границей доверительного интервала*, а 0,9 называется *верхней границей доверительного интервала*, или *UCB*. Алгоритм UCB выбирает для исследования игровой автомат с более высоким значением UCB.

Допустим, есть три игровых автомата, и на каждом из автоматов вы сыграли 10 раз. Доверительные интервалы для этих трех игровых автоматов изображены на следующем рисунке:



Мы видим, что игровой автомат 3 имеет высокое значение UCS. Тем не менее не стоит делать вывод, что этот игровой автомат обеспечивает хорошую награду, на основании всего 10 попыток. С другой стороны, после нескольких попыток доверительный интервал будет более точным. Таким образом, со временем доверительный интервал сужается к фактическому значению, как показано на следующем рисунке. Итак, мы можем выбрать игровой автомат 2 с высоким значением UCS:



Принцип работы алгоритма UCS очень прост:

1. Выбрать действие (руку) с высокой суммой средней награды и верхней границей доверительного интервала.
2. Нажать на рычаг и получить награду.
3. Обновить награду и границу доверительного интервала для руки.

Но как вычислить UCS?

Для этого можно воспользоваться формулой $\sqrt{\frac{2 \log(t)}{N(a)}}$, где $N(a)$ — количество нажатий на рычаг, а t — общее количество раундов.

Итак, в алгоритме UCS рука выбирается по следующей формуле:

$$Arm = \arg \max_a \left[Q(a) + \sqrt{\frac{2 \log(t)}{N(a)}} \right].$$