от теории к коду, они упорядочены в соответствии с изложением глубокого RL в академической литературе.

Другой важный аспект изучения глубокого RL — эксперименты. С этой целью в SLM Lab представлен фреймворк для экспериментов, призванный помочь начинающим в создании и тестировании собственных гипотез.

Библиотека SLM Lab — это проект с открытым исходным кодом на Github. Мы рекомендуем установить ее (на машины с Linux или MacOS) и запустить первое демо, следуя инструкциям, приведенным на сайте репозитория https://github.com/kengz/SLM-Lab. В Git была создана специальная ветка book с версией кода, совместимой с этой книгой. В листинге 0.1 приведена краткая инструкция по установке, скопированная с сайта репозитория.

Листинг 0.1. Установка SLM-Lab с ветки book

```
# клонируйте репозиторий
git clone https://github.com/kengz/SLM-Lab.git
cd SLM-Lab
# переключитесь на специальную ветку для этой книги
git checkout book
# установите зависимости
./bin/setup
# далее следуйте инструкциям, приводимым на сайте репозитория
```

Рекомендуется сначала задать эти настройки, чтобы можно было обучать агентов по алгоритмам в соответствии с их появлением в книге. Помимо установки и запуска первого примера, необходимости в знакомстве с SLM Lab до прочтения глав об алгоритмах (части I и II) не возникнет: все команды для обучения агентов даны там, где нужно. Кроме того, SLM Lab обсуждается более подробно в главе 11, после перехода от алгоритмов к практическим аспектам глубокого обучения с подкреплением.

Благодарности

Завершить этот проект нам помогло немало людей. Спасибо Милану Цвитковичу, Алексу Лидсу, Навдипу Джайтли, Джону Крону, Кате Василаки и Кейтлин Глисон за поддержку и вдохновение. Выражаем благодарность OpenAI, PyTorch, Илье Кострикову и Джамромиру Дженишу за предоставление высококачественной реализации с открытым исходным кодом различных компонентов алгоритмов глубокого RL. Благодарим также Артура Джулиани за предварительные обсуждения структуры сред. Эти ресурсы и обсуждения были бесценны при создании SLM Lab.

Хотелось бы поблагодарить Александра Саблайроллеса, Ананта Гупта, Брендона Стрикланда, Чонга Ли, Джона Крона, Джорди Франка, Кэтрин Джаясурия, Мэтью Ратца, Пидонга Вонга, Раймонда Чуа, Регину Р. Монако, Рико Джоншковски, Софи Табак и Утку Эвси за обстоятельные замечания о ранних набросках этой книги.

Мы очень признательны производственной команде Pearson — Алине Кирсановой, Крису Зану, Дмитрию Кирсанову и Джулии Нахил. Благодаря вашему профессионализму, старанию и вниманию к деталям текст стал гораздо лучше.

Наконец, эта книга не появилась бы на свет без нашего редактора Деборы Вильямс Коли. Благодарим за вашу помощь и терпение.

Об авторах

Лаура Грессер работает разработчиком исследовательского программного обеспечения для робототехнических систем в Google. Получила степень магистра компьютерных наук в Нью-Йоркском университете со специализацией в машинном обучении.

Ван Лун Кенг — разработчик систем искусственного интеллекта в Machine Zone, использует глубокое обучение с подкреплением для решения проблем промышленного производства. Специалист в области теоретической физики и компьютерных наук.

Они вместе разработали две библиотеки программного обеспечения для глубокого обучения с подкреплением и выпустили ряд лекций и учебников по этой теме.

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На веб-сайте издательства www.piter.com вы найдете подробную информацию о наших книгах.

О научном редакторе русскоязычного издания

Александр Игоревич Панов, кандидат физико-математических наук, доцент, заведующий отделом ФИЦ ИУ РАН, руководитель Центра когнитивного моделирования МФТИ, ведущий научный сотрудник Института искусственного интеллекта (AIRI), член Научного совета Российской ассоциации искусственного интеллекта. Специалист в области когнитивной робототехники, обучения с подкреплением, общего искусственного интеллекта.

1 Введение в обучение с подкреплением

В этой главе вводятся основные концепции обучения с подкреплением. Сначала рассматриваются простые примеры, которые позволят вам интуитивно понимать основные компоненты, используемые в обучении с подкреплением, — агента и среду.

В частности, мы рассмотрим процесс оптимизации целевой функции посредством взаимодействия агента со средой. Далее дано более формальное определение, а обучение с подкреплением объясняется с помощью понятия марковского процесса принятия решений. Это теоретические основы обучения с подкреплением.

Затем представлены три основные функции, которые должен выучить агент: *стратегия*, *функции полезности* и *модель среды*. Далее показано, как разные варианты обучения этих функций порождают различные семейства алгоритмов глубокого обучения с подкреплением.

В конце дан краткий обзор основного метода глубокого обучения — метода аппроксимации функций, который используется на протяжении всей книги. Также здесь обсуждаются основные различия между обучением с подкреплением и обучением с учителем.

1.1. Обучение с подкреплением

Обучение с подкреплением (reinforcement learning, RL) занимается задачами последовательного принятия решений. Многие реальные проблемы, возникающие в компьютерных играх, спорте, вождении автомобиля, оптимизации товарных запасов, роботизированном управлении, то есть везде, где действуют люди и машины, могут быть представлены в подобном виде.

Решая каждую из этих задач, мы преследуем какую-то цель: победить в игре, безопасно доехать до пункта назначения или минимизировать стоимость строительных материалов. Мы предпринимаем действия и получаем из окружающего мира ответ о том, насколько близки к цели: текущий счет, расстояние до пункта назначения или цену одного изделия. Достижение цели, как правило, подразумевает

выполнение ряда действий, каждое из которых изменяет окружающий мир. Мы наблюдаем эти изменения мира, а также получаем обратную информацию, опираясь на которую принимаем решение о следующем шаге.

Представьте, что вы на вечеринке, а ваш друг принес флагшток и предлагает на спор как можно дольше балансировать им, поставив на ладонь. Если вы никогда до сих пор не делали этого, то первоначальные попытки будут не слишком удачными. Вероятно, первые несколько минут вы потратите на то, чтобы методом проб и ошибок почувствовать флагшток, ведь он все время падает.

Эти ошибки позволят накопить полезную информацию и приобрести интуитивное понимание того, как удерживать флагшток в равновесии. Вы узнаете, где находится его центр масс, с какой скоростью он наклоняется, при каком угле наклона падает, как быстро вы может подстроиться и т. д. Вы используете эту информацию, чтобы внести коррективы при следующих попытках, совершенствуетесь и снова корректируете свое поведение. Вы даже не заметите, как начнете удерживать равновесие по 5, 10, 30 с, 1 мин и т. д.

Этот процесс наглядно демонстрирует, как работает обучение с подкреплением. В обучении с подкреплением вас можно назвать агентом, а флагшток и ваше окружение — средой. Фактически первая среда, задачу которой мы научимся решать с помощью обучения с подкреплением, — это игровая версия данного сценария под названием CartPole (рис. 1.1). Агент управляет скользящей вдоль оси тележкой так, чтобы удерживать стержень в вертикальном положении в течение заданного времени. Реальные способности людей гораздо шире, ведь мы можем интуитивно понимать физическую сторону происходящего. А можем и применить навыки выполнения схожих заданий, таких как балансирование подносом, уставленным напитками. Но, по сути, формулировка задачи остается той же самой.

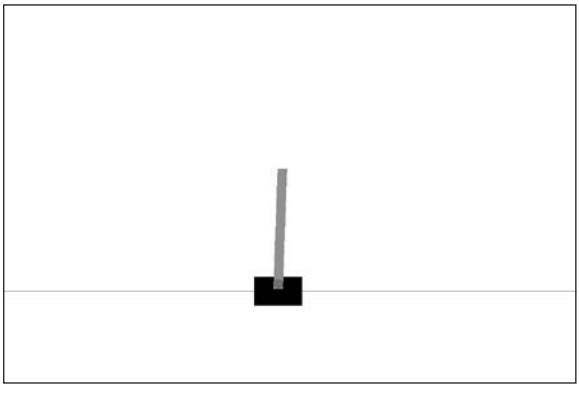


Рис. 1.1. CartPole-v0 — простая игровая среда. Цель — удержание в равновесии стержня на протяжении 200 шагов посредством управления перемещениями тележки вправо и влево

В обучении с подкреплением изучаются подобного рода задачи, а также методы, с помощью которых искусственные агенты учатся их решать. Это область искусственного интеллекта, которая восходит к теории оптимального управления и использует понятие марковского процесса принятия решений (МППР). RL появился в 1950-х годах в контексте динамического программирования и квазилинейных уравнений благодаря Ричарду Беллману. Его имя будет еще не раз упомянуто при изучении получившего известность в обучении с подкреплением уравнения Беллмана.

Задачи RL могут быть представлены как система, состоящая из агента и среды. Среда предоставляет информацию, описывающую состояние системы. Агент взаимодействует со средой, наблюдая состояние и используя данную информацию при выборе *действия*. Среда принимает действие и переходит в следующее состояние, а затем возвращает агенту следующее состояние и *вознаграждение*. Когда цикл *«состояние → действие → вознаграждение»* завершен, предполагается, что сделан один шаг. Цикл повторяется, пока среда не завершится, например, когда задача решена. Полностью этот процесс показан на диаграмме цикла управления (рис. 1.2).

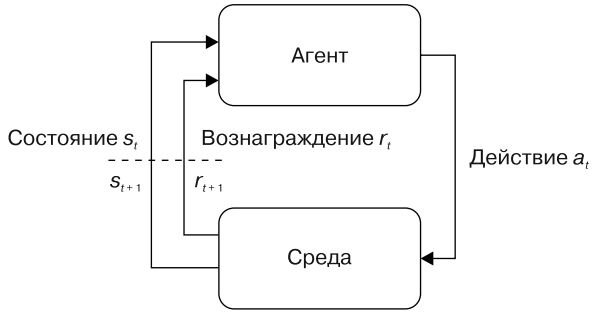


Рис. 1.2. Цикл управления агентом в обучении с подкреплением

Функция, в соответствии с которой агент выбирает действия, называется *стратегией*. Формально стратегия — это функция, отображающая множество состояний в множество действий. Действие изменяет среду и влияет на то, что агент наблюдает и делает дальше. Обмен информацией между агентом и средой разворачивается во времени, однако его можно рассматривать как процесс последовательного принятия решений.

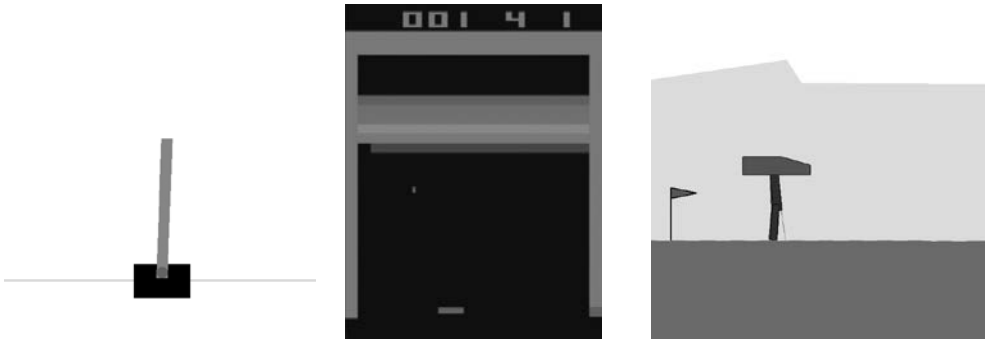
В задачах RL есть *целевая функция*, которая является суммой полученных агентом вознаграждений. Задача агента — максимизировать целевую функцию, выбирая наилучшие действия. Он *учится* этому, взаимодействуя со средой методом проб и ошибок, и использует поощряющие сигналы для *подкрепления* лучших действий.

Агент и среда определены как взаимоисключающие сущности, чтобы мы могли четко отделить друг от друга состояния, действия и вознаграждения. Среду можно

рассматривать как все, что не является агентом. Например, для езды на велосипеде возможны несколько различных, но равнозначных определений агента и среды. Если рассматривать все человеческое тело как агента, который наблюдает свое окружение, а производимые им напряжения мышц — как действия, то среда — это дорога и велосипед. Если считать агентом мыслительный процесс, то средой будут физическое тело, велосипед и дорога, действиями — нервные импульсы, пересылаемые от головного мозга к мускулам, а состояниями — поступающие в мозг сигналы от органов чувств.

По сути, система обучения с подкреплением реализует цикл управления с обратной связью, где агент и среда взаимодействуют и обмениваются сигналами, причем агент пытается максимизировать целевую функцию. Сигналы — это тройка (s_t, a_t, r_t), что соответствует состоянию, действию и вознаграждению, а индекс t указывает на номер шага (момент времени), на котором возник сигнал. Кортеж (s_t, a_t, r_t) называется *прецедентом* или частью получаемого агентом опыта. Цикл управления может повторяться до бесконечности[1] или закончиться по достижении либо конечного состояния, либо максимального значения шага $t = T$. Временной горизонт от $t = 0$ до момента завершения среды носит название *эпизода*. *Траектория* — это последовательность прецедентов, или часть опыта, накопленного в течение эпизода, $\tau = (s_0, a_0, r_0), (s_1, a_1, r_1)$... Обычно агенту для обучения хорошей стратегии требуется от сотен до миллионов эпизодов в зависимости от сложности задачи.

Рассмотрим для обучения с подкреплением три примера сред (рис. 1.3) и определения состояний, действий и вознаграждений. Все эти среды можно получить в OpenAI Gym — библиотеке с открытым исходным кодом, которая предоставляет стандартизованный набор сред.



а) CartPole б) Atari Breakout в) BipedalWalker

Рис. 1.3. Три примера сред с разными состояниями, действиями и вознаграждениями

[1] Бесконечные циклы управления существуют в теории, но не на практике. Как правило, для среды устанавливается максимальное количество шагов T.

CartPole (см. рис. 1.3, *а*) — одна из простейших сред для обучения с подкреплением, была впервые описана Барто, Саттоном и Андерсоном в 1983 году. В этой среде стержень закреплен на тележке, которая может двигаться по дорожке без трения. Основные особенности среды приведены далее.

1. **Цель** — удерживать стержень в вертикальном положении в течение 200 шагов.
2. **Состояние** — массив из четырех элементов: [позиция тележки, скорость тележки; угол наклона стержня; угловая скорость стержня], например [–0,034; 0,032; –0,031; 0,036].
3. **Действие** — целое число: 0 при перемещении тележки на фиксированное расстояние влево и 1 — при перемещении на фиксированное расстояние вправо.
4. **Вознаграждение** равно +1 на каждом шаге, на котором стержень остается в вертикальном положении.
5. Среда **завершается** либо при падении стержня (отклонение от вертикали на угол больше 12°), либо при выходе тележки за пределы экрана, либо при достижении максимального числа шагов, равного 200.

Atari Breakout (см. рис. 1.3, *б*) — это старая консольная аркадная игра, в которой есть мячик, расположенная внизу экрана платформа и блоки. Цель — попадать в блоки и разрушать их, отбивая мячик платформой. В начале игры у игрока пять жизней, одна из которых теряется, когда мячик попадает мимо платформы.

1. **Цель** — набрать максимальный счет в игре.
2. **Состояние** — цифровое изображение в формате RGB с разрешением 160×210 пикселов, то есть то, что мы видим на экране игры.
3. **Действие** — целое число из набора {0, 1, 2, 3}, которое сопоставляется игровым действиям {бездействие, запустить мячик, перемещение вправо, перемещение влево}.
4. **Вознаграждение** — разность в счете между двумя идущими друг за другом состояниями.
5. Среда **завершается**, когда все жизни потеряны.

BipedalWalker (см. рис. 1.3, *в*) — задача непрерывного управления, где агент-робот сканирует окрестности с помощью датчика-лидара и старается удержаться от падения при движении вправо.

1. **Цель** — идти вправо не падая.
2. **Состояние** — массив из 24 элементов: [угол наклона корпуса; угловая скорость корпуса; скорость по оси *X*; скорость по оси *Y*; угол поворота шарнира бедра 1; скорость шарнира бедра 1; угол поворота шарнира колена 1; скорость шарнира колена 1; есть ли контакт с землей ноги 1; угол поворота шарнира бедра 2; скорость шарнира бедра 2; угол поворота шарнира колена 2; скорость

шарнира колена 2; есть ли контакт с землей ноги 2... 10 показаний лидара], например [2,745e–03; 1,180e–05; –1,539e–03; –1,600e–02... 7,091e–01; 8,859e–01; 1,000e+00; 1,000e+00].

3. **Действие** — вектор из четырех чисел с плавающей запятой со значениями в интервале [–1,0; 1,0], имеющий вид [крутящий момент и скорость бедра 1; крутящий момент и скорость колена 1; крутящий момент и скорость бедра 2; крутящий момент и скорость колена 2], например [0,097; 0,430; 0,205; 0,089].
4. **Вознаграждение** — за движение вправо максимально до +300 и –100 за падение робота. Кроме того, за каждый шаг дается небольшое отрицательное вознаграждение (плата за движение), пропорциональное приложенному крутящему моменту.
5. Среда **завершается**, когда тело робота касается земли, либо при достижении цели, расположенной справа, либо после максимального количества шагов, равного 1600.

Эти среды демонстрируют различные варианты того, в каком виде могут быть представлены состояния и действия. В CartPole и BipedalWalker состояния — это векторы, кодирующие свойства, такие как позиции и скорости. В Atari Breakout состояние — это изображение экрана игры. В CartPole и Atari Breakout действия являются одиночными дискретными целыми числами, тогда как в BipedalWalker действие — это вектор из четырех непрерывных вещественных значений. Вознаграждение всегда представлено скалярной величиной, но пределы его значений варьируются в зависимости от задания.

Просмотрев несколько примеров, можно ввести формальные обозначения для состояний, действий и вознаграждений.

Состояние:

$$s_t \in S, \tag{1.1}$$

где S — пространство состояний.

Действие:

$$a_t \in A, \tag{1.2}$$

где A — пространство действий.

Вознаграждение:

$$r_t = R(s_t, a_t, s_{t+1}), \tag{1.3}$$

где R — функция вознаграждения.

Пространство состояний S — это набор из всех возможных в среде состояний. В зависимости от среды оно может быть определено как набор целых или вещественных чисел, векторов, матриц, структурированных или неструктурированных данных. Аналогично пространство действий A — это набор всех возможных действий, определяемых средой. Оно также может иметь разный вид, но чаще всего действие определяется как скалярная величина или вектор. Функция вознаграждения $R(s_t, a_t, s_{t+1})$ присваивает положительное, отрицательное или равное нулю скалярное значение каждому переходу (s_t, a_t, s_{t+1}). Пространство состояний, пространство действий и функция вознаграждения задаются средой. Вместе они составляют кортежи (s, a, r), являющиеся основными информационными единицами при описании систем обучения с подкреплением.

1.2. Обучение с подкреплением как МППР

Теперь рассмотрим, как среда переходит из одного состояния в другое с помощью так называемой *функции переходов*. В обучении с подкреплением функция переходов — это основной компонент марковского процесса принятия решений (МППР), который является математической основой моделирования последовательного принятия решений.

Чтобы понять, как функции переходов связаны с МППР, рассмотрим общую постановку задачи, приведенную в следующем выражении:

$$s_{t+1} \sim P(s_{t+1} \mid (s_0, a_0), (s_1, a_1) \ldots (s_t, a_t)). \tag{1.4}$$

В выражении (1.4) утверждается, что на временном шаге t следующее состояние s_{t+1} берется из распределения вероятностей P, обусловленного всей историей. Вероятность перехода среды из состояния s_t в состояние s_{t+1} зависит от всех предыдущих состояний s и действий a, которые имели место в данном эпизоде до этого момента. Записать функцию переходов в подобной форме сложно, особенно если эпизоды растягиваются на большое количество шагов. При проектировании таких функций переходов нужно учитывать огромное количество комбинаций факторов, возникавших в течение всех предыдущих шагов. Кроме того, в такой формулировке становится очень сложной функция выбора действий агентом — его стратегия. Поскольку для понимания того, как действие может изменить следующее состояние среды, важна вся история состояний и действий, то агенту придется принимать во внимание всю эту информацию, принимая решение о выборе действия.

Чтобы функция переходов среды стала лучше реализуемой на практике, преобразуем ее в МППР. Сделаем такое предположение: переход в следующее состояние s_{t+1} зависит только от предыдущих значений состояния s_t и действия a_t. Данное

предположение известно как *марковское свойство*. С учетом этого функция переходов примет следующий вид:

$$s_{t+1} \sim P(s_{t+1} \mid s_t, a_t). \tag{1.5}$$

Выражение (1.5) гласит, что следующее состояние s_{t+1} берется из распределения вероятностей $P(s_{t+1} \mid s_t, a_t)$. Это упрощенная форма первоначальной функции переходов. Марковское свойство подразумевает, что текущие состояние и действие на шаге t содержат достаточно информации, чтобы в полной мере определить вероятность перехода в следующее состояние на шаге $t + 1$.

Несмотря на простоту данной формулировки, она довольно эффективна. В подобной форме могут быть выражены многие процессы, такие как игры, управление робототехническими системами и планирование. Это связано с тем, что определение состояния может включать любую информацию, позволяющую сделать функцию переходов марковской.

Рассмотрим пример с последовательностью чисел Фибоначчи, описываемой формулой $s_{t+1} = s_t + s_{t-1}$, где каждый член s_t рассматривается как состояние. Чтобы эта функция стала марковской, переопределим состояние как $s'_t = [s_t, s_{t-1}]$. Теперь состояние содержит достаточно информации для расчета следующего элемента последовательности. В более общей форме эта стратегия может быть применена к любой системе с конечным набором k последовательных состояний, содержащих достаточно сведений для перехода в следующее состояние. В примечании 1.1 более подробно описано определение состояний в МППР и в его обобщенной форме — частично наблюдаемом МППР. Обратите внимание: на протяжении всей книги встречаются примечания, где вопросы рассматриваются углубленно. При первом прочтении их можно пропустить — это не помешает понять основную тему.

Примечание 1.1. МППР и частично наблюдаемые МППР

До сих пор понятие состояния применялось в двух случаях. С одной стороны, состояние — это то, что порождается средой и что наблюдает агент. Назовем его *наблюдаемым состоянием* s_t. С другой стороны, состояние — это то, что используется функцией переходов. Назовем его *внутренним состоянием* s_t^{int} среды.

В МППР $s_t = s_t^{\text{int}}$, то есть наблюдаемое состояние идентично внутреннему состоянию среды. Агенту доступна информация, которая используется для перехода среды в следующее состояние.

Это не всегда верно. Наблюдаемое состояние может отличаться от внутреннего состояния среды, $s_t \neq s_t^{\text{int}}$. В этом случае среда описывается как *частично наблюдаемый МППР*, так как предоставляемое агенту состояние *st* содержит лишь часть информации о состоянии среды.

В этой книге в большинстве случаев данное различие не учитывается и предполагается, что $s_t = s_t^{\text{int}}$. Однако знать о частично наблюдаемых МППР важно по двум

причинам. Во-первых, некоторые рассматриваемые примеры среды не являются идеальными МППР. Например, в среде Atari наблюдаемое состояние s_t — это одно изображение в формате RGB, в котором передается информация о позиции объекта, количестве жизней агента и т. д., но нет скоростей объекта. Скорости будут включены во внутреннее состояние среды, поскольку они требуются для определения следующего состояния, заданного действием. Тогда для достижения высокой производительности нужно будет преобразовать s_t так, чтобы оно содержало больше сведений. Это обсуждается в главе 5.

Во-вторых, многие интересные реальные проблемы по своей сути являются частично наблюдаемыми МППР, в их число входят случаи проявления ограниченности датчиков или данных, ошибок моделирования и зашумленности среды. Детальное рассмотрение частично наблюдаемых МППР лежит за пределами этой книги, но они будут вкратце затронуты при обсуждении архитектуры нейронных сетей в главе 12.

Наконец, при рассмотрении структуры состояний в главе 14 различие между s_t и s_t^{int} будет иметь большое значение, поскольку агент обучается по s_t. Информация, которая содержится в s_t, и степень его отличия от s_t^{int} влияют на то, насколько сложным или простым будет решение задачи.

Теперь можно представить формулировку задачи обучения с подкреплением в виде МППР. МППР определяется кортежем из четырех элементов — S, A, $P(.)$, $\mathcal{R}(.)$, где:

- S — набор состояний;
- A — набор действий;
- $P(s_{t+1} \mid s_t, a_t)$ — функция перехода состояний среды;
- $\mathcal{R}(s_t, a_t, s_{t+1})$ — функция вознаграждения среды.

Рассматриваемые в этой книге задачи обучения с подкреплением используют одно важное предположение: функция переходов $P(s_{t+1} \mid s_t, a_t)$ и функция вознаграждений $\mathcal{R}(s_t, a_t, s_{t+1})$ недоступны для агентов. Они могут получить информацию об этих функциях только через состояния, действия и вознаграждения, воздействию которых подвергаются на данный момент в среде, то есть через кортежи (s_t, a_t, r_t).

Чтобы формулировка задачи была полной, нужно формализовать понятие максимизируемой агентом целевой функции. Во-первых, определим *отдачу*[1] (return) $R(\tau)$, используя траекторию из эпизода $\tau = (s_0, a_0, r_0) \ldots (s_T, a_T, r_T)$:

$$R(\tau) = r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots + \gamma^T r_T = \sum_{t=0}^{T} \gamma^t r_t. \tag{1.6}$$

В уравнении (1.6) отдача определена как дисконтированная сумма вознаграждений на траектории, где γ — коэффициент дисконтирования, $\gamma \in [0, 1]$.

[1] Отдача обозначается R, а $\mathcal{R}$ оставлено для функции вознаграждения.

Тогда *целевая функция* $J(\tau)$ становится просто математическим ожиданием отдачи по нескольким траекториям:

$$J(\tau) = \mathbb{E}_{\tau\sim\pi}\left[R(\tau)\right] = \mathbb{E}_{\tau}\left[\sum_{t=0}^{T}\gamma^t r_t\right]. \tag{1.7}$$

Отдача $R(\tau)$ — это сумма дисконтированных вознаграждений $\gamma^t r_t$ за все временные шаги $t = 0...T$. Целевая функция $J(\tau)$ — это отдача, усредненная по нескольким эпизодам. Математическое ожидание предполагает случайный характер выбора действий и поведения среды, то есть при повторных запусках отдача не всегда будет одинаковой. Максимизация целевой функции — то же самое, что и максимизация отдачи.

Коэффициент дисконтирования $\gamma \in [0, 1]$ — важная переменная, влияющая на то, как оцениваются будущие вознаграждения. Чем меньше γ, тем меньший вес имеют будущие вознаграждения, что ведет к «близорукости» агента. В крайнем случае, когда $\gamma = 0$, целевая функция рассматривает только начальное вознаграждение r_0, как показано в уравнении (1.8):

$$R(\tau)_{\gamma=0} = \sum_{t=0}^{T}\gamma^t r_t = r_0. \tag{1.8}$$

Чем больше значение γ, тем больший вес придается вознаграждениям на будущих временных шагах: агент становится «дальнозорким». Если $\gamma = 1$, вознаграждения на всех шагах имеют одинаковый вес, как показано в уравнении (1.9):

$$R(\tau)_{\gamma=1} = \sum_{t=0}^{T}\gamma^t r_t = \sum_{t=0}^{T} r_t. \tag{1.9}$$

Для задач с *бесконечным* временным горизонтом нужно устанавливать $\gamma < 1$, чтобы целевая функция не стала бесконечной. Для задач с *конечным* временным горизонтом γ — важный параметр, так как в зависимости от используемого значения коэффициента дисконтирования решение может стать проще или сложнее. Пример этого будет рассмотрен в главе 2.

Определив целевую функцию и представив задачу обучения с подкреплением как МППР, можно выразить цикл управления агентом (рис. 1.2) как цикл управления в МППР (алгоритм 1.1).

Алгоритм 1.1. Цикл управления МППР

```
Считаем, что агент и среда env уже созданы
for episode = 0,...,MAX_EPISODE do
    state = env.reset()
    agent.reset()
    for t = 0,...,T do
        action = agent.act(state)
        state, reward = env.step(action)
        agent.update(action, state, reward)
```

```
 9:         if env.done() then
10:             break
11:         end if
12:     end for
13: end for
```

Алгоритм 1.1 показывает взаимодействие между агентом и средой на протяжении множества эпизодов и некоторого количества шагов. В начале каждого эпизода среда и агент возвращаются к исходным позициям (строки 3 и 4), при этом среда порождает начальное состояние. Затем начинается их взаимодействие: агент выполняет действие, исходя из данного состояния (строка 6), затем, основываясь на этом действии, среда производит следующее состояние и вознаграждение (строка 7), переходя на следующий шаг. Цикл `agent.act-env.step` длится, пока не будет достигнуто максимальное количество шагов T либо среда не завершится. Здесь появляется новый компонент — `agent.update` (строка 8), который включает в себя алгоритм обучения агента. На протяжении большого количества шагов и эпизодов этот метод накапливает данные и на внутреннем уровне обучается максимизации целевой функции.

Этот алгоритм общий для всех задач обучения с подкреплением, поскольку он определяет непротиворечивый интерфейс взаимодействия между агентом и средой. Данный интерфейс служит основой для реализации многих алгоритмов обучения с подкреплением, включенных в унифицированный фреймворк, как будет видно в прилагаемой к книге библиотеке SLM Lab.

1.3. Обучаемые функции в обучении с подкреплением

Когда обучение с подкреплением сформулировано как МППР, возникает естественный вопрос: чему должен учиться агент?

Мы видели, что агент может сформировать функцию выбора действий, известную как *стратегия*. Однако у среды есть и другие свойства, которые могут быть полезны для агента. В частности, существует *три основных функции*, которые изучаются в обучении с подкреплением.

1. Стратегия π, которая сопоставляет состоянию действие: $a \sim \pi(s)$.
2. Функция полезности $V^{\pi}(s)$ или $Q^{\pi}(s, a)$ для вычисления ожидаемой отдачи $\mathbb{E}_{\tau}\left[R(\tau)\right]$.
3. Модель среды[1] $P(s' \mid s, a)$.

[1] Чтобы сделать нотацию более краткой, принято записывать пару следующих друг за другом кортежей (s_t, a_t, r_t), $(s_{t+1}, a_{t+1}, r_{t+1})$ как (s, a, r), (s', a', r'), где штрих означает следующий шаг. Это обозначение будет встречаться в книге повсеместно.

Стратегия π — это то, каким образом агент производит действия в среде, чтобы максимизировать целевую функцию. Согласно циклу управления агент должен производить действия на каждом шаге после наблюдения состояния *s*. Стратегия имеет фундаментальное значение для цикла управления, поскольку генерирует действия, которые заставляют его продолжаться.

Стратегия может быть стохастической. Это значит, что она может с определенной вероятностью давать на выходе разные действия для одного состояния. Это может быть записано как $\pi(a \mid s)$ и означает вероятность действия a для данного состояния *s*. Действие, выбранное по стратегии, записывается как $a \sim \pi(s)$.

Функции полезностей представляют информацию о цели. Они помогают агенту понять, насколько хороши состояния и доступные действия с точки зрения ожидаемой отдачи. Они записываются в двух формах:

$$V^{\pi}(s) = \mathbb{E}_{s_0 = s, \tau \sim \pi}\left[\sum_{t=0}^{T} \gamma^t r_t\right]; \tag{1.10}$$

$$Q^{\pi}(s,a) = \mathbb{E}_{s_0 = s, a_0 = a, \tau \sim \pi}\left[\sum_{t=0}^{T} \gamma^t r_t\right]. \tag{1.11}$$

Функция полезности V^{π}, приведенная в уравнении (1.10) оценивает, насколько хорошим или плохим является состояние. V^{π} дает оценку ожидаемой отдачи от пребывания в положении *s*, предполагая, что агент продолжает действовать в соответствии со своей текущей стратегией π. Отдача $R(\tau) = \sum_{t=0}^{T} \gamma^t r_t$ подсчитывается, начиная с текущего состояния *s* и до конца эпизода. Это прогнозная оценка, поскольку не учитываются все вознаграждения, полученные до состояния *s*.

Рассмотрим простой пример, который позволит получить представление о функции полезности V^{π}. На рис. 1.4 изображена дискретная среда с конечным числом состояний, в которой агент может перемещаться из клетки в клетку по вертикали и горизонтали. Каждая клетка — это состояние, с которым связано вознаграждение, как показано на рис. 1.4, *слева*. Среда завершается, когда агент попадает в целевое состояние с вознаграждением $r = +1$.

Справа показаны полезности $V^{\pi}(s)$, рассчитанные для каждого состояния по вознаграждениям с помощью уравнения (1.10) при $\gamma = 0{,}9$. Функция полезности V^{π} всегда зависит от конкретной стратегии π. В этом примере взята стратегия π, при которой всегда выбирается кратчайший путь к целевому состоянию. Если стратегия будет другой — к примеру, перемещение только вправо, — то значения полезностей будут иными.

Это показывает, что функция полезности является прогнозной и помогает агенту различать состояния с одинаковым вознаграждением. Чем ближе агент к целевому состоянию, тем выше полезность рассматриваемого состояния.

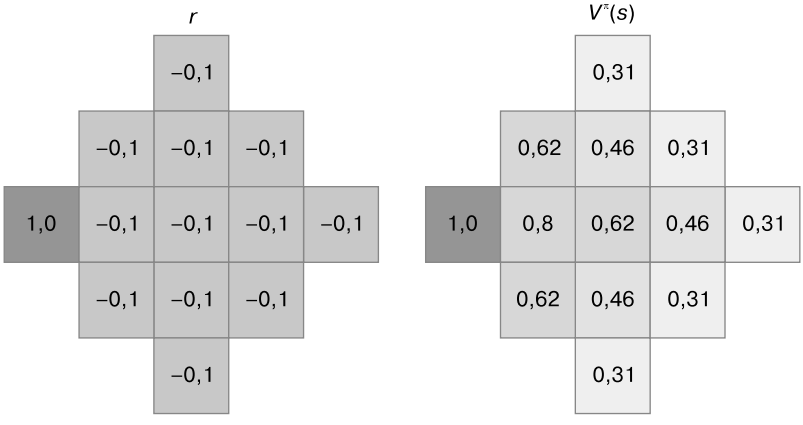


Рис. 1.4. Вознаграждения r и полезности $V^{\pi}(s)$ для каждого состояния s в простой клеточной среде. Полезность состояния рассчитывается по вознаграждениям с помощью уравнения (1.10) при $\gamma = 0{,}9$. Здесь применяется стратегия π, при которой всегда выбирается кратчайший путь к целевому состоянию с $r = +1$

Функция полезности Q^{π} в уравнении (1.11) оценивает, насколько хороша или плоха пара «состояние — действие». Q^{π} дает оценку ожидаемой отдачи от выбора действия a в состоянии s, предполагая, что агент продолжает действовать в соответствии со своей текущей стратегией π. По аналогии с V^{π} отдача подсчитывается, начиная с текущего состояния s и до конца эпизода. Это тоже прогнозная оценка, поскольку не учитываются все вознаграждения, полученные до состояния s.

Функции V^{π} и Q^{π} более подробно рассматриваются в главе 3. А сейчас достаточно знать, что они существуют и могут быть использованы агентом для решения задач обучения с подкреплением.

Функция переходов $P(s' \mid s, a)$ предоставляет информацию о среде. Сформировав эту функцию, агент обретает способность предсказывать следующее состояние s', в которое перейдет среда после выбора действия a в состоянии s. Применяя полученную функцию переходов, агент может вообразить последствия действий, не вступая в действительное взаимодействие со средой. В дальнейшем он может использовать эту информацию для планирования оптимальных действий.

1.4. Алгоритмы глубокого обучения с подкреплением

В RL агент настраивает функции, которые помогают ему действовать и максимизировать целевую функцию. Эта книга посвящена глубокому обучению *с подкреплением* (глубокому RL), что означает, что в качестве аппроксимирующего семейства функций будут использоваться глубокие нейронные сети.