

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное
образовательное учреждение высшего образования
«Московский физико-технический институт
(национальный исследовательский университет)»

А. И. Панов

ВВЕДЕНИЕ В МЕТОДЫ МАШИННОГО ОБУЧЕНИЯ С ПОДКРЕПЛЕНИЕМ

Учебное пособие

МОСКВА
МФТИ
2019

УДК 004.853(075)

ББК 22.18я73

П16

Рецензенты:

Доктор физико-математических наук, профессор

Института проблем искусственного интеллекта

ФИЦ «Информатика и управление» РАН Г. С. Осипов

Доктор физико-математических наук, профессор

Института проблем искусственного интеллекта

ФИЦ «Информатика и управление» РАН М. Г. Дмитриев

Панов, Александр Игоревич

П16 Введение в методы машинного обучения с подкреплением:

учебное пособие / А. И. Панов. – Москва: МФТИ, 2019. – 52 с.

ISBN 978-5-7417-0703-6

Рассмотрены основные понятия и алгоритмы машинного обучения с подкреплением – активно развивающегося направления в искусственном интеллекте.

На основе материалов лекций, читаемых автором в МФТИ и Высшей школе экономики, представлены основные методы обучения с подкреплением: методы динамического программирования, метод временных различий, градиентные и иерархические методы. Особое внимание уделено рассмотрению алгоритмических аспектов указанных подходов, приводятся иллюстративные примеры и отмечаются основные преимущества и недостатки описываемых методов.

Предназначено для студентов старших курсов и аспирантов, изучающих методы искусственного интеллекта, машинное обучение и интеллектуальные робототехнические системы.

УДК 004.853(075)

ББК 22.18я73

Печатается по решению Редакционно-издательского совета Московского физико-технического института (национального исследовательского университета)

ISBN 978-5-7417-0703-6

© Панов А. И., 2019

© Федеральное государственное автономное образовательное учреждение высшего образования «Московский физико-технический институт (национальный исследовательский университет)», 2019

Оглавление

Введение	4
Глава 1. Динамическое программирование	7
§1.1. Постановка задачи	7
§1.2. Итерации по значениям и уравнение Беллмана	11
§1.3. Итерации по стратегиям	15
Глава 2. Функция полезности	17
§2.1. Адаптивное динамическое программирование	17
§2.2. Проблема применения–исследования	20
§2.3. Q -функция	21
Глава 3. Алгоритмы временных различий	24
§3.1. Обучение с учетом временных различий	24
§3.2. Подход $TD(\lambda)$	27
§3.3. Алгоритм SARSA	29
§3.4. Q -обучение	29
Глава 4. Градиентные методы	32
§4.1. Актор-критик	32
§4.2. Аппроксимация и обобщение	36
§4.3. Метод градиента стратегии	40
Глава 5. Иерархические методы	43
§5.1. Феодальный подход и умения	44
§5.2. Абстрактные автоматы и MAXQ	46
Заключение	49
Литература	50

Введение

Обучение с подкреплением (reinforcement learning, RL) является разделом машинного обучения, активно развивающимся направлением в искусственном интеллекте. Несмотря на то, что формально обучение с подкреплением относится к разделу приобретения знаний, оно кардинально отличается от таких методов, как обучение с учителем или без учителя. В первую очередь, здесь явно выделен субъект приобретения знаний (агент), который принимает решения и некоторым образом влияет на источник анализируемых данных (среду). Эта агентная постановка очень близка по своей методологии к одному из определений искусственного интеллекта, который давали одни из основоположников искусственного интеллекта Рассел и Норвиг [19]:

Искусственный интеллект — это наука об «интеллектуальных агентах», т. е. о некотором устройстве или программе, которая воспринимает свою среду и выполняет действия, которые максимизируют её шансы на успех при достижении какой-то цели.

Наличие у агента некоторого набора возможных способов воздействия на среду (действий) и его стремления достигнуть некоторой поставленной заранее цели в этой среде позволяют естественным образом применять обучение с подкреплением в более сложных интеллектуальных системах, которые разрабатываются для синтеза целенаправленного поведения: в интеллектуальных динамических системах и, в частности, в робототехнике [18]. В обучении с подкреплением наиболее тесно переплетаются методы планирования поведения, представления и приобретения знаний. В настоящее время именно подсистемы, реализующие методы обучения с подкреплением, становятся центральными элементами комплексных систем управления поведением автономных объектов вместо ранее занимавших главенствующую позицию подсистем представления знаний. Таким образом, наблюдается переход от более статичных когнитивных архитектур (например, Soar [8]) к более активным обучающимся архитектурам (например, знаковым [17]).

Успехи в развитии методов обучения с подкреплением возродили интерес к разработке агентов, действующих в искусственных средах,

в том числе игровых. Были разработаны обучающиеся агенты, демонстрирующие иногда результаты, превосходящие уровень человека, как для сред компьютерных игр (Atari [11], DOOM [2], Starcraft [12], Minecraft [3]), так и для более серьезных, приближенных к условиям, в которых действуют люди в реальной жизни (Go [9], OpenAI Universe¹). Демонстрируемые успехи, понятные и знакомые даже далеким от искусственного интеллекта людям, породили большую волну новых исследований, и сейчас секции по обучению с подкреплением занимают самую большую часть научных конференций (ICML, NIPS, IJCAI).

В качестве короткой исторической справки необходимо отметить, что идеи, лежащие в основе современной теории обучения с подкреплением, высказывались еще на первых этапах становления искусственного интеллекта в 60-х годах XX века (отечественные работы по автоматам Цетлина и Стефанюка [21], зарубежные инженерные работы Вальца и Фу и др. [16]) и были заимствованы из области психологии, где еще с начала XX века существовало понятие обусловленности поведения и условных рефлексов (Павлов, Скиннер).

Настоящее учебное пособие призвано дать краткий обзор современных подходов в обучении с подкреплением и является сжатым описанием основных алгоритмов, в том числе тех, которые появились буквально в последние несколько лет. Все подходы распределены на группы (обучения с моделью, итерации по стратегиям и т. п.), каждой из которых выделена отдельная глава. Границы между этими группами зачастую условны, и некоторые подходы могут быть отнесены сразу к нескольким направлениям. Так как обучение с подкреплением развивается чрезвычайно динамично и каждый день появляются новые работы, настоящий обзор не может быть полным.

Пособие состоит из трех основных блоков: краткого введения в постановку задачи и алгоритмов, основанных на динамическом программировании (гл. 1), итерационных алгоритмов временной разности (гл. 2), группы алгоритмов градиента стратегии (§ 4.3) и иерархических подходов (гл. 5).

В подготовке этого пособия были использованы материалы ряда монографий по обучению с подкреплением, которые могут служить дополнительной литературой: книга Саттона и Барто [20], являющихся основоположниками данного направления машинного обучения, несколько глав фундаментального труда Рассела и Норвига [19, гл. 21],

¹<https://blog.openai.com/universe/>

краткий обзор Жепешвари [15] и ряд интернет-ресурсов и онлайн-курсов (в основном на английском языке):

- блог Массимилиано Патачёла²;
- курс Школы Яндекса³;
- курс Паскаля Попарта⁴;
- материалы к курсу Саттона⁵;
- лекции Дэвида Сильвера⁶;
- материалы к курсу Беркли по глубокому обучению с подкреплением⁷;
- небольшой обзор методов иерархического обучения с подкреплением⁸.

Материал данного пособия соответствует лекционной части курса «Обучение с подкреплением». Для развития прикладных навыков программирования рекомендуется также изучать вторую часть пособия «Машинное обучение с подкреплением на Python», в котором наибольшее внимание будет уделено алгоритмической части, т. е. описанию и разбору современных алгоритмов, реализующих основные модели обучения с подкреплением. В качестве эталонных программных реализаций можно ссылаться на базу программного кода OpenAI и RLCode⁹.

²[https://mpatacchiola.github.io/blog/2016/12/09/
dissecting-reinforcement-learning.html](https://mpatacchiola.github.io/blog/2016/12/09/dissecting-reinforcement-learning.html)

³https://github.com/yandexdataschool/Practical_RL

⁴<https://cs.uwaterloo.ca/~ppoupart/teaching/cs885-spring18>

⁵<https://drive.google.com/drive/folders/0B3w765rOKuKANmxNbXdwaE1YU1k>

⁶<http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html>

⁷<https://github.com/berkeleydeeprlcourse>

⁸<https://thegradient.pub/the-promise-of-hierarchical-reinforcement-learning/>

⁹<https://github.com/openai/baselines> и <https://github.com/rllcode/reinforcement-learning>

Глава 1. Динамическое программирование

§1.1. Постановка задачи

Начнем рассмотрение задачи обучения с подкреплением с примера задачи последовательного принятия решений, следуя классическому изложению [19]. Пусть агент находится в клеточной среде E размером $n \times m$, в которой каждая клетка является либо проходимой, либо содержит препятствие (для общности можно считать, что $n \times m$ клеток окружены $2n + 2m$ непроходимыми клетками). Среди проходимых клеток есть одна стартовая клетка c^s , в которой находится агент в начальном состоянии среды, и несколько конечных клеток $\{c_1^f, c_2^f, \dots\}$, попадание агента в которые приводит к завершению *эпизода* взаимодействия среды и агента. Агенту доступно множество действий $A = \{Up, Down, Left, Right\}$ – движений в соседние клетки из текущей клетки (вверх, вниз, налево и направо соответственно). При попытке двинуться в непроходимую клетку положение агента не меняется. Среда E является *полностью наблюдаемой*, т. е. агенту доступна полная информация о всех клетках среды, и *недетерминированной*, т. е. действия выполняются агентом только с некоторой неединичной вероятностью. Стохастическая модель совершения действия следующая: каждое действие достигает своей цели с вероятностью 0.8, с вероятностями 0.1 агент перемещается в перпендикулярных направлениях от намеченной цели. Состояние среды меняется только в результате совершения действий агентом.

Выполнение агентом, например, последовательности действий

$$\{Up, Up, Right, Right, Right\},$$

направленной на достижение конечной клетки, например, для среды на рис. 1.1, приведет к цели только с вероятностью $0.8^5 = 0.3277$.

С учетом случайного достижения цели при обходе препятствия с другой стороны полная вероятность достижения цели равна $8^5 + 0.1^4 \cdot 0.8 = 0.3278$.

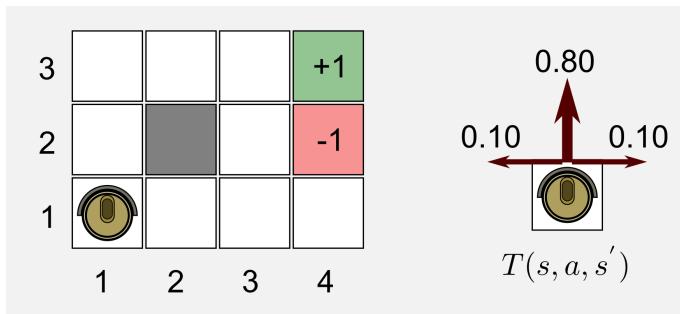


Рис. 1.1. Пример клеточной среды с двумя конечными клетками (зеленая и красная)

Набор известных вероятностей перехода из одного состояния среды в другое, что эквивалентно набору вероятностей результатов действий в каждом состоянии среды, называется *моделью переходов* $T(s, a, s')$, где s – состояние, в котором было выполнено действие $a \in A$, в результате чего было достигнуто состояние s' . Будем предполагать эти переходы марковскими, т. е. такими, для которых T зависит только от текущего состояния s , а не от истории пребывания агента в предыдущих состояниях.

Как было указано в начале, в среде существуют конечные клетки, т. е. целью совершения агентом действий является одна из целевых клеток. Функцию полезности действий агента, которая в отличии от модели переходов зависит от истории пребывания в среде, определим через получаемые агентом от среды в каждом состоянии вознаграждения $r(s, a, s')$ (в общем виде вознаграждение зависит от действия и состояния). Эти вознаграждения являются обычными действительными числами. Например для среды E , $r = -0.04$ во всех состояниях, кроме конечных. В конечных клетках агент получает вознаграждения -1 и $+1$. Полезность агента определяется суммой всех полученных к данному моменту вознаграждений $R = \sum_i r_i$. Например, если агент достиг состояния $+1$ после 10 шагов, $R = 1 - 10 * 0.04 = 0.6$. С учетом этого целью агента является достижение целевого состояния с максимальной итоговой полезностью, что приводит к тому, что

агент должен стремиться уменьшить количество действий, приводящих к низким вознаграждениям.

Определенная таким образом задача последовательного принятия решений для полностью наблюдаемой недетерминированной среды с марковской моделью переходов и дополнительными вознаграждениями, задающими функцию полезности агента, называется марковским процессом принятия решений (Markov Decision Process, МППР) (см. рис. 1.2). Любая задача МППР определяется как кортеж: $\langle S, A, s_0, T(s, a, s'), r(s, a, s') \rangle$, где S – множество состояний среды, A – множество действий агента, $s_0 \in S$ – начальное состояние среды, $T(s, a, s')$ – модель переходов, $r(s, a, s')$ – функция вознаграждения, которую далее для упрощения будем считать зависящей только от текущего состояния s .

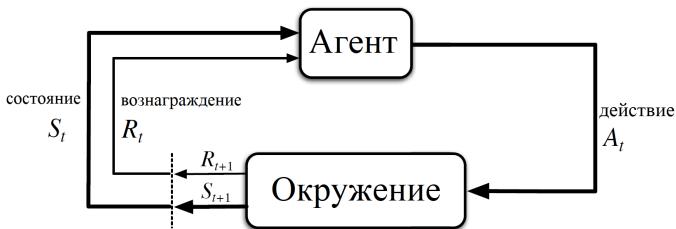


Рис. 1.2. Схема взаимодействия агента и среды

Решением задачи МППР является *стратегия* $\pi(s) : S \rightarrow A$ – функция, которая по каждому состоянию среды s выдает действие, которое необходимо совершить в этом состоянии. Оптимальной стратегией является та стратегия π^* , которая позволяет агенту при следовании этой стратегии достичь максимальной ожидаемой полезности. Примеры оптимальных стратегий в зависимости от значений функции вознаграждения в нетерминальных состояниях приведены на рис. 1.3. В случае очень большого штрафа ($r(s) < -1.628$, очень низкая емкость батареи) агенту выгоднее как можно быстрее завершить эпизод даже в конечном состоянии -1 . При небольшом штрафе ($-0.428 < r(s) < -0.085$, батарея средней емкости) агент выбирает кратчайший путь к цели -1 и старается избегать попадания в состояние -1 . При низком штрафе ($-0.022 < r(s) < 0$, батарея высокой емкости) агент вообще избегает любого риска, который бы мог даже случайно привести его в состояние -1 , поэтому он огибает препятствие по самой дальней траектории.

Когда же штрафов вообще нет ($r(s) > 0$, бесконечная емкость батареи) агент не стремится завершить эпизод и избегает конечных состояний.

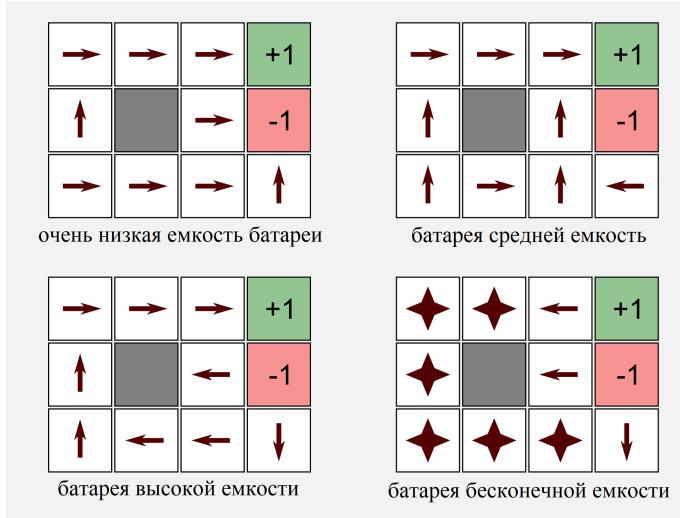


Рис. 1.3. Примеры оптимальных стратегий для различных значений «промежуточных» вознаграждений

Определение функции полезности агента как суммы всех вознаграждений, полученных агентом в процессе взаимодействия со средой, нуждается в небольшом пояснении. Во-первых, будем различать конечные и бесконечные эпизоды, которым соответствуют конечный и бесконечный горизонты принятия решений агентом. В случае конечного эпизода длина последовательностей действий ограничена некоторым максимальным значением N . Оптимальная стратегия, вообще говоря, зависит от N (при $N = 5$ в примере со средой E , несмотря ни на какие риски, агент должен идти к цели кратчайшим путем), что означает, что оптимальная стратегия *неstationарна* при $N < \infty$. В случае бесконечного эпизода длина последовательности действий потенциально не ограничена, хотя, конечно, при достижении конечного состояния эпизод завершается. При этом оптимальная стратегия *стационарна* (не зависит от ограничений, налагаемых на длину эпизода).

Во-вторых, функция полезности агента, кроме обычной суммы $R = \sum_t r(s_t)$ (*аддитивные* вознаграждения), может задаваться как $R = \sum_t \gamma^t r(s_t)$ (*обесцениваемые* вознаграждения), где γ – коэффициент обесценивания (дисконтирующий множитель), который описы-

вает предпочтение агентом текущих вознаграждений перед будущими. В случае $\gamma = 1$ необходимо перейти к обычному аддитивному варианту. В многоатрибутной теории полезности вводится гипотеза *стационарности* предпочтений между последовательностями состояний, которая в нашем случае означает, что если есть две последовательности состояний (атрибутов) $S = [s_0, s_1, \dots]$ и $S' = [s'_0, s'_1, \dots]$, при этом $s_0 = s'_0$, то S предпочтительнее S' только в том случае, если $[s_1, \dots]$ предпочтительнее $[s'_1, \dots]$. В случае МПР принятие этой гипотезы естественно и приводит к тому, что в этих условиях стационарности других способов задания функции полезности не существует.

Использование обесцениваемых вознаграждений удобно еще и потому, что в случае с бесконечными последовательностями действий полезность оказывается ограниченной:

$$R = \sum_{t=0}^{\infty} \gamma^t r(s_t) \leq \sum_{t=0}^{\infty} \gamma^t r_{\max} = \frac{r_{\max}}{1 - \gamma}.$$

Не нарушая общности, будем использовать в дальнейшем обесцениваемые вознаграждения и записывать задачу поиска оптимальной стратегии как

$$\pi^* = \arg \max_{\pi} \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t) | \pi \right].$$

§1.2. Итерации по значениям и уравнение Беллмана

Простейший алгоритм поиска оптимальной стратегии в том случае, когда агенту известна вся информация о переходах среды, состоит в том, что происходит оценка полезности каждого состояния среды и использование этой информации для выбора оптимального действия в каждом состоянии. Полезность состояния s представляет собой ожидаемую полезность последовательности состояний, в которых оказывается агент, если он следует некоторой стратегии π , начиная с данного состояния s :

$$V^\pi(s) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t) | \pi, s_0 = s \right]. \quad (1.1)$$

Так как наибольшую полезность агент имеет, следуя оптимальной стратегии π^* , то истинное значение полезности состояния s определяется как $V^{\pi^*}(s)$. На рис. 1.4 представлен пример значений $V^{\pi^*}(s)$.

С приближением к конечному состоянию +1 ценность состояния увеличивается в связи с тем, что уменьшается количество шагов, за которые агент получает отрицательное вознаграждение.

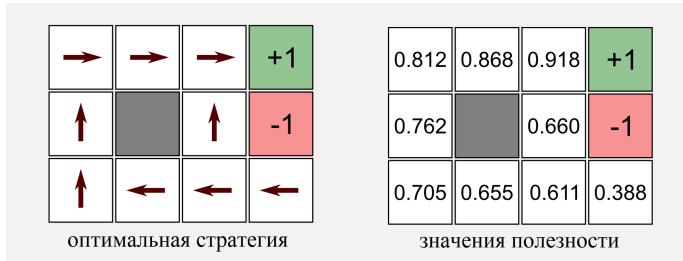


Рис. 1.4. Пример значений полезности состояний при условии $\gamma = 1$, для нетерминальных состояний s , $r(s) = -0.04$

С другой стороны, зная значения полезностей состояний, можно использовать принцип максимальной ожидаемой полезности и выбирать действия, которые максимизируют ожидаемую полезность следующего состояния:

$$\pi(s) = \arg \max_A \sum_{s'} T(s, a, s') V(s'). \quad (1.2)$$

Принимая во внимание последнее выражение, можно теперь определить полезность состояния как сумму текущего вознаграждения, полученного при совершении выбранного по стратегии π действия, и ожидаемой полезности очередного состояния, в которое агент попадает в зависимости от модели переходов T :

$$V(s) = r(s) + \gamma \max_A \sum_{s'} T(s, a, s') V(s'). \quad (1.3)$$

Уравнение (1.3) называется *уравнением Беллмана* и было предложено Ричардом Беллманом в рамках его изложения основ динамического программирования [1]. Полезности всех состояний являются решением системы уравнений Беллмана, записанных для каждого состояния, в которое может попасть агент.

На рис. 1.5 представлен пример выбора действия агентом, находящимся в левом нижнем углу карты. В соответствии с формулой (1.2) суммирование полезностей возможных по модели переходов следующих состояний приводит к тому, что действие *Up* является наилучшим, т. к. приводит в состояние с максимальной полезностью $V(s') = 0.7456$.

Применяя уравнение Беллмана, получаем, что полезность левого нижнего угла карты равна $V(s) = -0.04 + V(s') = 0.7056$.

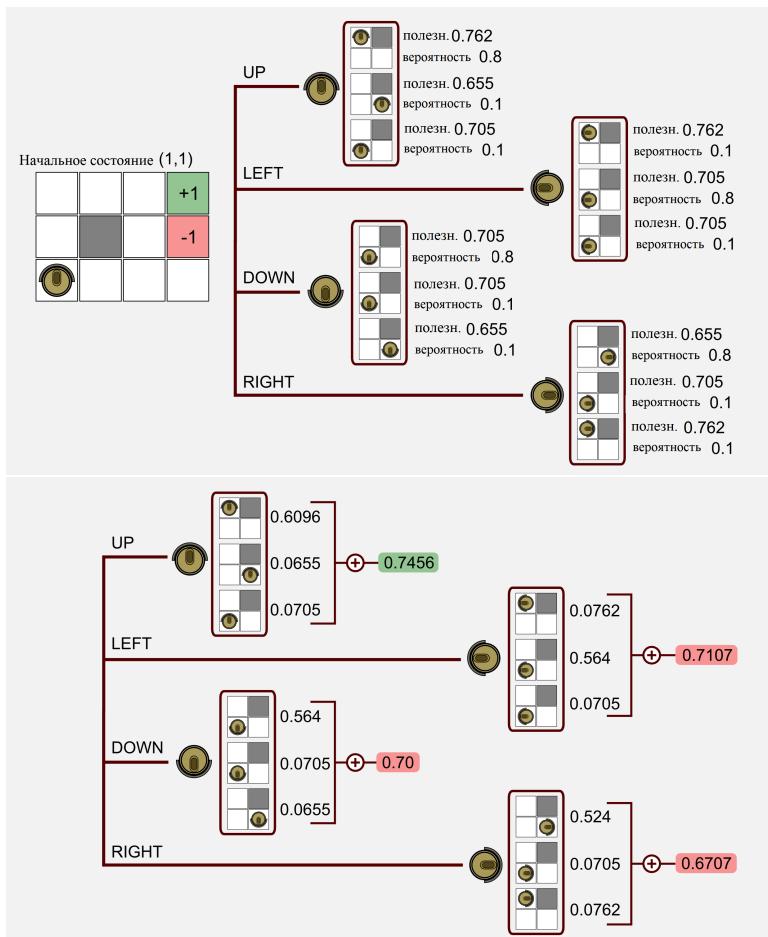


Рис. 1.5. Выбор наилучшего действия по полезностям состояний

Вместо того чтобы искать значения полезностей состояний путем решения системы уравнений Беллмана, которые из-за наличия функций тах являются нелинейными, применяют итерационный метод.

На некотором i -м шаге итерации происходит *обновление по Беллману*:

$$V_{i+1}(s) \leftarrow r(s) + \gamma \max_a \sum_{s'} T(s, a, s') V_i(s'). \quad (1.4)$$

Листинг 1. Итерации по значениям полезности

```

1: function VALUEITERATION( $MDP, \epsilon$ )
    $\triangleright MDP = < S, T, r, \gamma >$  – задача МПР,  $\epsilon$  – максимально
   допустимая ошибка определения значений полезности
2:    $V = 0; V' = 0$             $\triangleright$  начальные значения ценностей – нулевые
   векторы
3:   repeat
4:      $V \leftarrow V'; \delta \leftarrow 0$ 
5:     for all  $s \in S$  do
6:       
$$V'[s] \leftarrow r(s) + \gamma \max_a \sum_{s'} T(s, a, s') V[s']$$

7:       if  $|V'[s] - V[s]| > \delta$  then
8:          $\delta \leftarrow |V'[s] - V[s]|$ 
9:     until  $\delta < \frac{\epsilon(1-\gamma)}{\gamma}$ 
return  $V$ 

```

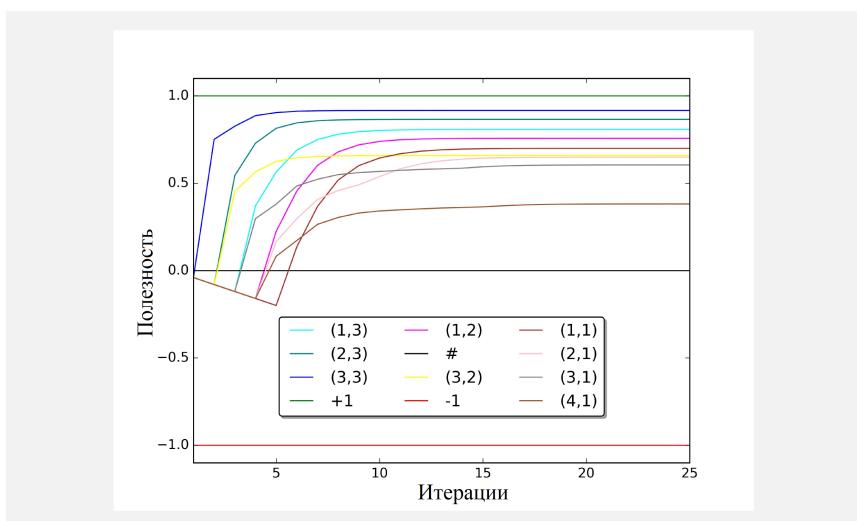


Рис. 1.6. Алгоритм итерации по значениям полезности на примере среды E

В пределе бесконечного повторения шагов обновления значения полезностей для каждого состояния будут являться решением системы уравнений Беллмана, а построенная на их основе стратегия 1.2 будет оптимальной. Полный алгоритм представлен в листинге 1. На рис. 1.6 представлен график изменения значений полезностей состояний в зависимости от номера итерации. Координаты состояний отсчитываются от нижнего левого угла карты. Необходимо отметить, что практически всегда стратегия становится оптимальной до того, как значения полезностей состояний приблизятся к своим конечным значениям, что позволяет останавливать итерации и до достижения критерия останова.

§1.3. Итерации по стратегиям

Последнее замечание предыдущего параграфа наводит на мысль, что нет необходимости точно определять полезности состояний для выбора наилучшего действия. Эта идея положена в основу второго базового алгоритма обучения с подкреплением с известной моделью переходов – *итерации по стратегиям*. Он состоит из двух этапов:

- *оценка стратегии*: используя стратегию π_i получить значения полезностей V^{π_i} ;
- *обновление стратегии*: прогнозируя на один шаг и выбирая максимальную ожидаемую полезность, по формуле (1.2) вычислить новую стратегию π_{i+1} .

Полный алгоритм представлен в листинге 2. Его особенностью является то, что оценка стратегии происходит с использованием упрощенного уравнения Беллмана:

$$V'[s] \leftarrow r(s) + \gamma \sum_{s'} T(s, \pi[s], s') V[s'], \quad (1.5)$$

в котором используется текущая стратегия и не происходит выбора наилучшего действия. Таким образом, пропадает операция максимизации, и система упрощенных уравнений Беллмана становится линейной системой, которая может быть решена аналитически и достаточно быстро даже при большом количестве состояний.

Листинг 2. Итерации по стратегиям

```
1: function POLICYITERATION( $MDP, \epsilon$ )
   ▷  $MDP = < S, T, r, \gamma >$  – задача МПР,  $\epsilon$  – максимально
      допустимая ошибка определения значений полезности
2:    $V \leftarrow [0, \dots], V' \leftarrow [0, \dots]$     ▷ начальные значения ценности –
      нулевые векторы
3:    $\pi \leftarrow random(|S|)$     ▷ начальная стратегия – случайный вектор
                                ▷ оценка стратегии
4:   repeat
5:     repeat
6:        $V \leftarrow V'; \delta \leftarrow 0$ 
7:       for all  $s \in S$  do
8:
9:          $V'[s] \leftarrow r(s) + \gamma \sum_{s'} T(s, \pi[s], s')V[s']$ 
10:        if  $|V'[s] - V[s]| > \delta$  then
11:           $\delta \leftarrow |V'[s] - V[s]|$ 
12:        until  $\delta < \epsilon$                                 ▷ обновление стратегии
13:         $stable \leftarrow False$ 
14:        for all  $s \in S$  do
15:           $R = \max_{a \in A} \sum_{s'} T(s, a, s')V[s']$ 
16:          if  $R > R'$  then
17:             $R' = \sum_{s'} T(s, \pi[s], s')V[s']$ 
18:             $\pi[s] \leftarrow \arg \max_A \sum_{s'} T(s, a, s')V[s']$ 
19:          until  $stable$ 
20:          return  $\pi$ 
```

Существует ряд модификаций и упрощений рассмотренных выше алгоритмов итераций по значениям и стратегиям, например: обобщенные итерации по стратегиям [20, п. 4.6] или асинхронные итерации по стратегиям [19, п. 17.3].

Глава 2. Функция полезности

§2.1. Адаптивное динамическое программирование

Рассмотрев в предыдущих параграфах простейшие варианты среды, в которой агенту давалась полная информация как о состояниях среды, так и о функции вознаграждения, перейдем к более общей постановке, когда у агента нет априорных знаний о среде и подкреплении. Формально отличие от предыдущей постановки задачи состоит в том, что агенту неизвестна модель переходов $T(s, a, s')$ – он должен восстановить ее в явном виде в ходе самого процесса обучения или иметь иное ее представление, по которому будет строиться стратегия выбора действий. Также агенту заранее недоступна информация о функции $r(s, a, s')$, т. е. то вознаграждение, которое он будет получать в данном состоянии при совершении данного действия (см. рис. 2.1).

Агент, в отсутствии информации о модели переходов, может осуществлять *пассивное обучение*, т. е. пользоваться неизменяемой стратегией π и пытаться восстановить только значения полезностей состояний среды. Если же агент также пытается выучить и оптимальную стратегию по выбору действий в среде, то этот случай называется *активным обучением*. Оба этих подхода часто называются *Монте-Карло подходами*, т. к. агент использует набранную статистику для оценки модели и выбора действий. В связи с этим возникает классическая проблема обучения с подкреплением – проблема соотношения между процессом исследования среды, когда агент набирает необходимые для принятия решений знания, и процессом применения найденной стратегии по достижению цели (*проблема применения–исследования, exploration-exploitation problem*).

В случае пассивного обучения задача агента сводится к тому, чтобы по набираемой агентом статистике $\{(s_i, r(s_i))\}$ совершения действий

по некоторой, возможно случайной, стратегии π (см. рис. 2.2) с использованием стандартной формулы 1.1 оценить значение полезности состояния $V^\pi(s)$. На примере из рис. 2.2 для трех эпизодов в нашу подвыборку для состояния $(1, 3)$ войдут три последовательности (две из первого эпизода и одна из второго), и оценка полезности будет равна (при $\gamma = 1$):

$$V(1, 3) = \frac{(1 - 5 \cdot 0.04) + (1 - 3 \cdot 0.04) + (1 - 5 \cdot 0.04)}{3} = 0.827.$$

Такая оценка значений V для каждого состояния при большом количестве эпизодов сходится к истинным значениями полезности состояний и называется *непосредственной оценкой*. Такой подход индуктивного обучения значениям полезности не учитывает взаимозависимости состояний, которая описывается уравнением Беллмана, поэтому он сходится очень медленно.

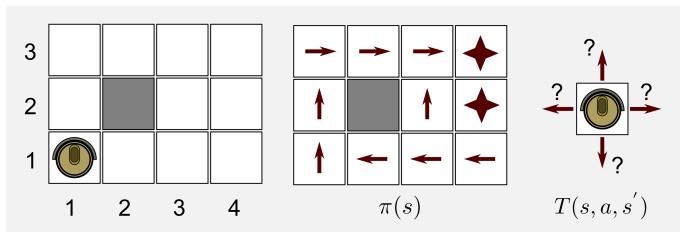


Рис. 2.1. Общая постановка задачи обучения с подкреплением: неизвестны модель переходов $T(s, a, s')$ и информация о вознаграждениях $r(s, a, s')$



Рис. 2.2. Примеры траекторий, получаемых агентов при постоянной стратегии. Значения сверху квадратов состояний – получаемые вознаграждения

Естественно, что более эффективным является учет зависимости значения полезности одного состояния от значений полезности последующих состояний. Но для этого агенту необходимо строить модель среды, т. е. восстанавливать модель переходов $T(s, a, s')$ и вознаграждения $r(s, a, s')$. Для этого можно воспользоваться так называемым алгоритмом адаптивного динамического программирования (см. листинг 3). Для построения модели здесь используется таблица вероятностей переходов, которые оцениваются по частоте, с которой достигается состояние s' при совершении действия a в состоянии s . Для примера на рис. 2.2 вероятность перехода из состояния $(1, 3)$ в состояние $(2, 3)$ равна $T((1, 3), Right, (2, 3)) = \frac{2}{3}$. Оценка стратегии (*ValueDetermination*) происходит с использованием упрощенных уравнений Беллмана (1.5) по аналогии с первой частью алгоритма итерации по стратегиям. Данный алгоритм сходится очень быстро и служит своего рода эталоном эффективности для задач с невысокой размерностью множества состояний и действий.

Листинг 3. Пассивное обучение с подкреплением на основе адаптивного динамического программирования (ADP)

```

1: function PASSIVEADPAGENT( $s', r'$ )
                                ▷ глобальные переменные:
2:    $MDP = < S, T, r, \gamma >, \pi$           ▷ задача МПР и фиксированная
   стратегия
3:    $V \leftarrow [0, \dots]$ ;                  ▷ таблица значений полезности
4:    $N_{sa} \leftarrow [0, \dots]$ ;           ▷ таблица частот пар «состояние–действие»
5:    $N_{sas'} \leftarrow [0, \dots]$ ;        ▷ таблица частот троек
   «состояние–действие–состояние»
6:    $s \leftarrow \emptyset, a \leftarrow \emptyset;$ 
7:   if  $s'$  - новое then
8:      $V[s'] \leftarrow r', R[s'] \leftarrow r'$ ;
9:   if  $s \neq \emptyset$  then
10:     $N_{sa}[s, a] += 1, N_{sas'}[s, a, s'] += 1$ 
11:    for all  $t : N_{sas'}[s, a, t] \neq 0$  do
12:       $T[s, a, t] \leftarrow N_{sas'}[s, a, t] / N_{sa}[s, a]$ 
13:     $V \leftarrow ValueDetermination(\pi, V, MDP)$ 
14:    if  $s'$  - конечное then
15:       $s \leftarrow \emptyset, a \leftarrow \emptyset;$ 
16:    else
17:       $s \leftarrow s', a \leftarrow \pi[s']$ ;
return  $a$ 

```

§2.2. Проблема применения–исследования

Переходя к активному обучению, когда агент кроме определения значений полезности состояний определят и оптимальную стратегию, необходимо вернуться к *проблеме применение–исследование*. Если бы агент был жадным, т. е. всегда использовал только оптимальные действия, с точки зрения текущих оценок полезности, то тогда у нас была бы очень высокая вероятность попадания в локальный минимум. Иными словами, *жадный агент* с высокой долей вероятности найдет стратегию, которая является оптимальной относительно его найденной модели среды, но которая не является оптимальной с точки зрения истинной модели переходов. В статистической теории такая проблема была рассмотрена в постановке задачи об *n-руком бандите*, где игроку необходимо выбрать одну из n рукожаток игорного автомата, которую необходимо потянуть и с некоторой вероятностью получить выигрыш. Перед игроком в этой задаче стоит та же дилемма: выбрать ту рукоятку, которая ранее выдавала наибольший выигрыш, или попробовать другую, которую ранее еще не использовал?

Задача об *n-руком бандите*, в которой нужно найти стратегию агента по получению выигрыша со всеми возможными автоматами, почти не поддается точному решению. Тем не менее понятно, что схема решения должна быть жадной в пределе бесконечного исследования (Greedy in the Limit of Infinite Exploration, GLIE) и предусматривать опробование каждого действия в каждом состоянии для предотвращения пропуска оптимального действия. Понятно, что ADP-агент, использующий GLIE-схему, в итоге построит истинную модель среды.

К простейшим вариантам реализации схемы GLIE относится схема, в которой агент в $1/t$ случаях придерживается случайной стратегии, а в остальных текущей оптимальной. Другой, более интеллектуальный подход, представляет собой введение *функции исследования*:

$$f(v, n) = f\left(\sum_{s'} T(s, a, s') V^*(s'), N(a, s)\right),$$

где $N(s, a)$ – количество попыток попробовать действие a в состоянии s . Функция исследования используется, например, в итерации по значениям полезностей вместо обычного обновления Беллмана (1.4) для определения оптимистических оценок полезности:

$$V^* \leftarrow r(s) + \gamma \max_a f(v, n).$$

Очевидно, что функция $f(u, n)$ должна возрастать по v и убывать по n , например иметь следующий вид:

$$f(v, n) = \begin{cases} r^*, & \text{если } n < N_e, \\ u, & \text{иначе.} \end{cases}$$

Здесь агент будет вынужден проверить пару действие–состояние как минимум N_e раз.

§2.3. Q -функция

Для одновременной оценки полезности состояний и определения оптимальной стратегии (активное обучение) нам необходимо оценивать полезность каждого действия отдельно. В связи с этим вводят так называемую Q -функцию, которая определяется аналогично значениям V :

$$Q^\pi(s, a) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) | s_0 = s, a_0 = a, \pi \right].$$

Из этого выражения видно, что Q -функция оценивает полезности пар состояния–действие или Q -значения, которые напрямую связаны с оценками полезности состояния следующим соотношением:

$$V(s) = \max_a Q(s, a).$$

На рис. 2.3 представлен пример кодирования Q -функции в виде таблицы состояния–действие. Каждой строчке соответствует отдельное действие, колонке – возможное состояние, встречаемое в среде. Каждая ячейка таблицы (s, a) содержит Q -оценку полезности совершения действия a в состоянии s .

На рис. 2.4 приведена схема расчета суммарного вознаграждения для каждой пары состояния–действие, которое встретилось в одном из эпизодов взаимодействия агента и среды методом Монте-Карло. Рассматриваемый эпизод состоит из 7 шагов, и суммарное вознаграждение рассчитывается отдельно для всех встречаемых пар: $((1, 1) - Up)$, $((1, 2) - Up)$ и т. д. Внизу на рис. 2.4 представлена таблица

новых Q -значений после рассмотрения данного эпизода. При рассмотрении следующих эпизодов таблица пополняется новыми значениями. Эта часть алгоритма аналогична части оценки стратегии в алгоритме итерации по стратегиям.

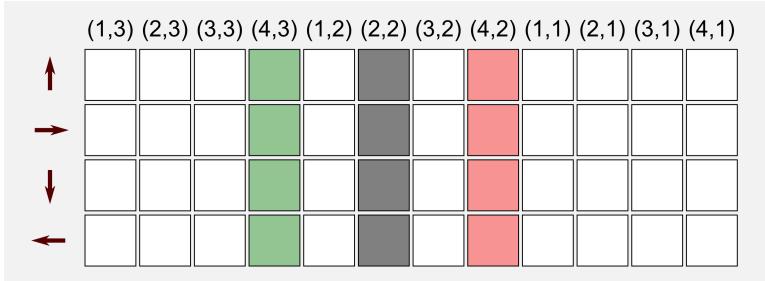


Рис. 2.3. Табличное представление Q -функции

Вторая часть алгоритма (обновление стратегии) в случае использования Q -таблицы очень проста: мы выбираем в текущем состоянии то действие, которое имеет максимальную оценку:

$$\pi(s) = \arg \max_a Q(s, a).$$

На рис. 2.5 представлен пример обновления стратегии для состояния $(1,3)$, для которого новые значения Q -таблицы предсказывают наилучшее действие *Left*.

Обновление стратегии агента на основе значений Q -функции происходит жадным образом – агент выбирает ту пару состояние–действие (s, a) , для которой значение $Q(s, a)$ максимально (рис. 2.5). При этом эффективность агента при таком подходе остается высокой, хотя известно, что жадные алгоритмы подвержены проблеме локальных минимумов. В данном случае жадным является не весь алгоритм, а только один из его этапов; а правильное обновление Q -таблицы позволяет нам получить хорошую сходимость за счет того, что агент в набирающейся статистике эпизодов в неявном виде получает всю необходимую информацию о лучших переходах.

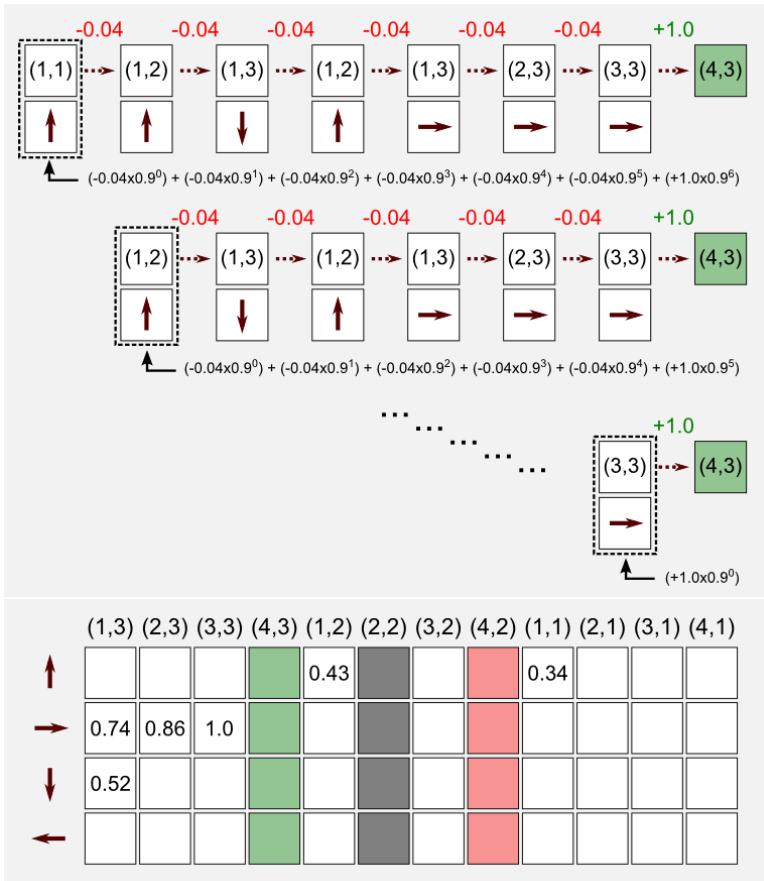


Рис. 2.4. Заполнение значений Q -таблицы

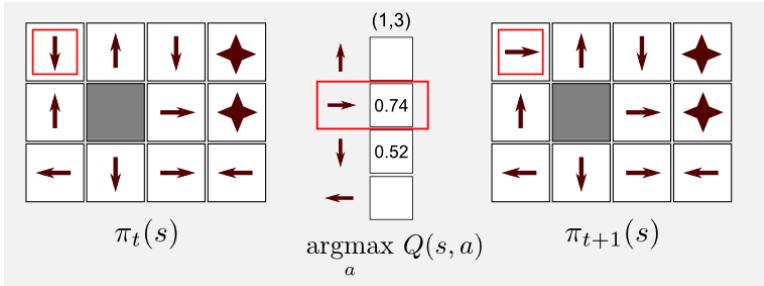


Рис. 2.5. Обновление стратегии с использованием Q -функции

Глава 3. Алгоритмы временных различий

§3.1. Обучение с учетом временных различий

Система упрощенных уравнений Баллмана (1.5) задает некоторое множество ограничений, которым должны удовлетворять полезности состояний. В следующем подходе, который получил название *обучения с учетом временной разности* (Temporal Difference, TD), предлагаются не решать эти уравнения для всех возможных состояний, а оценивать полезности только с учетом ограничений на наблюдаемые переходы. Например, для второй попытки на рис. 2.2 полезности состояний (1, 3) и (2, 3) должны удовлетворять следующему ограничению: $V^\pi(1, 3) = -0.04 + V^\pi(2, 3)$. Поэтому, если к этому моменту были получены оценки $V^\pi(1, 3) = 0.84$ и $V^\pi(2, 3) = 0.92$, агент может повысить заниженную оценку для (1, 3), чтобы удовлетворить данному уравнению. Обобщая, получаем следующий вариант обновления на основе временной разности между полезностями последовательных состояний:

$$V^\pi(s) = V^\pi(s) + \alpha(r + \gamma V^\pi(s') - V^\pi(s)), \quad (3.6)$$

где α – параметр, характеризующий скорость обучения.

При обновлении значений полезности состояния агент забывает историю своего взаимодействия со средой и использует только значения полезности предыдущего состояния (см. рис. 3.1); в связи с чем такой подход еще иногда называют *TD(0)*. Например, для состояния (1, 1) после первого шага агента новое значение полезности будет $V(1, 1) = 0.0 + 0.1(-0.04 + 0.9 * 0.0 - 0.0) = -0.004$.

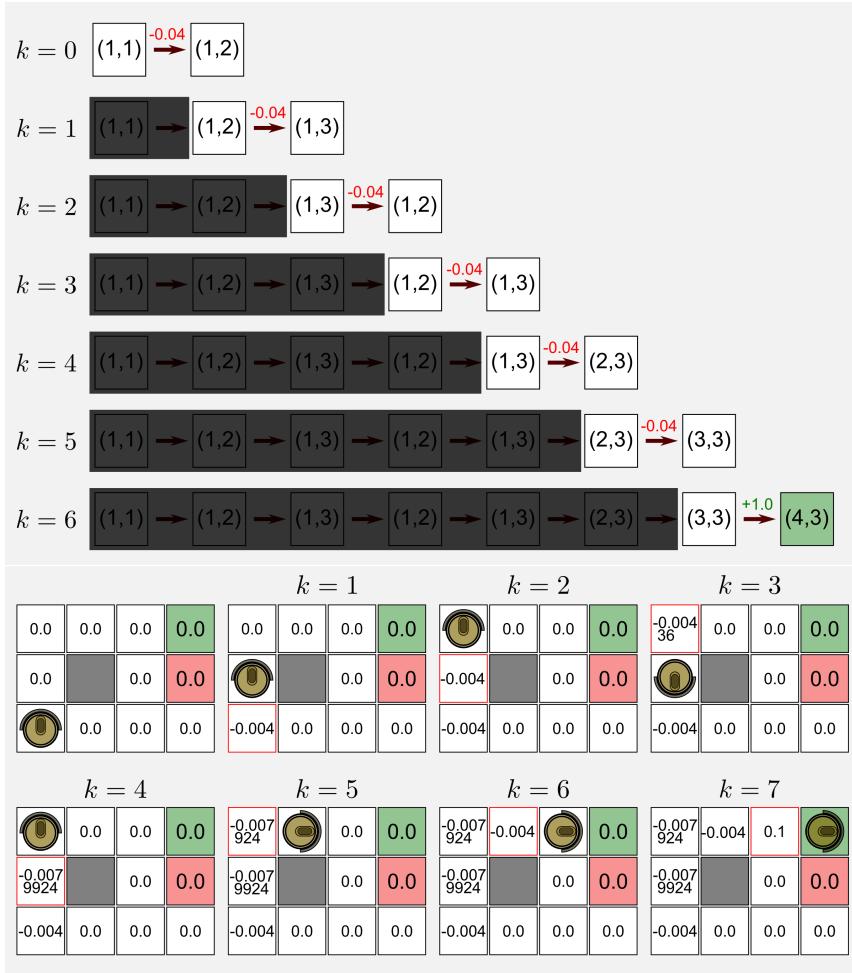


Рис. 3.1. Оценка значений полезности состояний на основе $TD(0)$

Полный алгоритм агента на основе временных различий представлен в листинге 4. Несмотря на то, что здесь, в отличие от уравнений Беллмана, для обновления полезности используется лишь одно из возможных состояний, это несильно сказывается на скорости сходимости, и данный агент обучается лишь немного медленнее, чем ADP-агент, при этом являясь более простым и требующим существенно меньше

вычислений. Параметр скорости обучения в общем случае может зависеть от того, сколько раз данное состояние встречалось в наблюдениях агента. Для обеспечения сходимости процесса должны соблюдаться условия: $\sum_{n=1}^{\infty} \alpha(n) = \infty$ и $\sum_{n=1}^{\infty} \alpha^2(n) < \infty$; например $\alpha = 1/n$ удовлетворяет этим соотношениям.

Листинг 4. Пассивное обучение с подкреплением на основе адаптивного динамического программирования (ADP)

```

1: function PASSIVETDAGENT( $s', r'$ )
   ▷ глобальные переменные:
2:    $MDP = < S, T, r, \gamma >, \pi$            ▷ задача МПР и фиксированная
   стратегия
3:    $V \leftarrow [0, \dots]$ ;                  ▷ таблица значений полезности
4:    $N_s \leftarrow [0, \dots]$ ;                ▷ таблица частот состояний
5:    $s \leftarrow \emptyset, a \leftarrow \emptyset, r \leftarrow \emptyset$ ;
6:   if  $s'$  - новое then
7:      $V[s'] \leftarrow r'$ ;
8:   if  $s \neq \emptyset$  then
9:      $N_s[s] += 1$ ;
10:     $V(s) \leftarrow V(s) + \alpha(N_s[s])(r + \gamma V(s') - V(s))$ 
11:   if  $s'$  - конечное then
12:      $s \leftarrow \emptyset, a \leftarrow \emptyset, r \leftarrow \emptyset$ ;
13:   else
14:      $s \leftarrow s', a \leftarrow \pi[s'], r \leftarrow r'$ ;
return  $a$ 

```

Алгоритм TD-агента не требует построения модели переходов, информация о связях состояний присутствует в неявном виде в статистике наблюдений агента, поэтому этот алгоритм относится к классу *свободных от модели* (model-free). TD-агент является довольно грубым приближением к затратному алгоритму ADP-агента. Существует ряд промежуточных вариантов, когда с использованием эвристик выбирается ряд предшественников (а не только один), которые будут влиять на изменение оценки полезности текущего состояния. Это позволяет достичь почти той же скорости обучения, что в оригинальном ADP-агент, но при этом существенно уменьшить количество вычислений.

§3.2. Подход $TD(\lambda)$

Теперь предположим, что агенту ставится задача обновлять полезности состояния не только на основе предыдущего, но и учитывая полезности какого-то количества предыдущих состояний. Для этого агента необходимо оснастить так называемой *кратковременной памятью*. Введем для каждого состояния s в момент времени t так называемый *след применимости* (eligibility trace):

$$e_t(s) = \begin{cases} \gamma \lambda e_{t-1}(s), & \text{если } s \neq s_t, \\ \gamma \lambda e_{t-1}(s) + 1, & \text{если } s = s_t. \end{cases}$$

Здесь γ – дисконтирующий множитель, а $\lambda \in [0, 1]$ – параметр, характеризующий скорость распада следа и определяющий вес обновления для каждого посещенного состояния. Если $0 < \lambda < 1$, то след распадается со временем и нечастым состояниям соответствует небольшой вес. При $\lambda = 0$ получается классический метод временных различий $TD(0)$, когда на обновление влияет только ценность предыдущего состояния. Наконец, если $\lambda = 1$, оценки всех предыдущих состояний обновляются независимо от того, когда они были встречены. Последний случай сближает метод временных различий с методами Монте-Карло с тем лишь отличием, что теперь необязательно ждать завершения эпизода для того, чтобы обновить оценки полезностей состояний.

На рис. 3.2 представлен пример изменения значения следа состояния s_1 при реализации эпизода, состоящего из 7 шагов. Перед первым посещением след состояния равен 0, на втором шаге к следу прибавляется 1 и на следующих шагах след распадается до тех пор, пока при втором посещении к текущему значению 0.25 снова не добавляется 1.

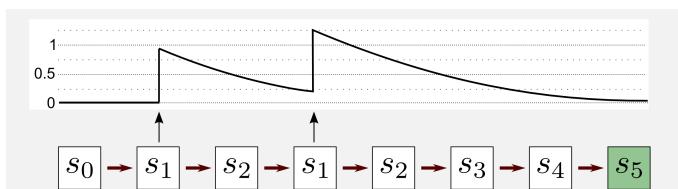


Рис. 3.2. Изменение следа состояния с течением эпизода взаимодействия агента и среды

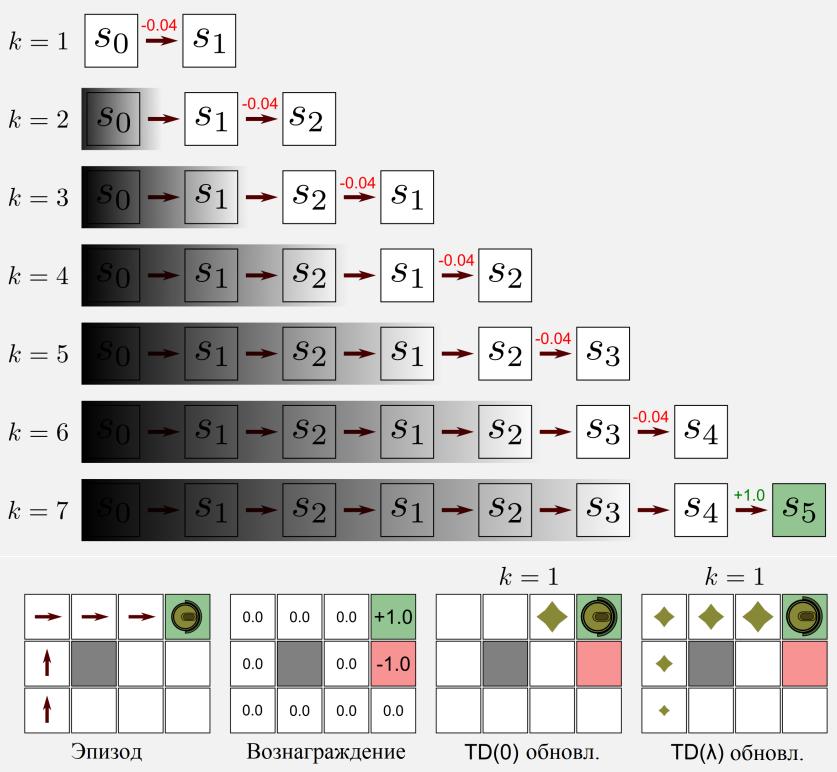


Рис. 3.3. Оценка значений полезности состояний на основе $TD(\lambda)$

Правило обновления значений полезности для всех состояний $s \in S$ на каждом шаге с учетом следа применимости выглядит следующим образом:

$$V_{t+1}(s) = V_t(s) + \alpha(r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t))e_t(s). \quad (3.7)$$

На рис. 3.3 представлен пример того, как происходит распространение в обратном направлении волны изменения оценок полезности состояний в методе $TD(\lambda)$. В том случае, когда вознаграждение дается только при достижении финального состояния, в $TD(0)$ меняется только значение полезности предыдущего состояния, в то время как в $TD(\lambda)$ в соответствии со следами состояний обновляются оценки полезности всех состояний, которые были посещены в рамках текущего эпизода. Очевидное преимущество использования $TD(\lambda)$ вместо

$TD(0)$ является более быстрая сходимость первого метода по сравнению со вторым, особенно в средах с редкими вознаграждениями и с большим количеством состояний.

§3.3. Алгоритм SARSA

Переходя к активному случаю, т. е. к такому варианту обучения, когда нам необходимо также строить оптимальную стратегию, можно записать для Q -функции (см. § 2.3) выражение итерационного обновления, аналогичного используемому в $TD(0)$ алгоритме (3.6):

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)). \quad (3.8)$$

Главная особенность здесь состоит в том, что нам необходимо знать примененное в следующем состоянии действие a_{t+1} , информация о котором может быть получена из Q -таблицы. Потому что в данном правиле обновления используются переменные $s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}$ алгоритм получил название SARSA.

Аналогично алгоритму $TD(0)$ SARSA игнорирует все состояния, кроме текущего и следующего за ним. В листинге 5 представлен полный вариант алгоритма, как обычно, для активного случая, состоящий из части обновления информации о среде (Q -таблицы) и обновления стратегии жадным способом. В этом алгоритме агент обновляет свою стратегию π , используя статистику, накопленную при использовании той же стратегии π . Такой подход называется *с явной стратегией* (op-policy).

Аналогично с $TD(\lambda)$ -подходом можно оснастить агента памятью, т. е. ввести след применимости $e_t(s)$ для каждой пары состояние-действие (s, a) :

$$e_t(s, a) = \begin{cases} \gamma \lambda e_{t-1}(s, a) + 1, & \text{если } s = s_t, a = a_t; \\ \gamma \lambda e_{t-1}(s, a), & \text{иначе.} \end{cases}$$

При добавлении этого следа в правило обновления 3.8 получается алгоритм $SARSA(\lambda)$, который обладает более быстрой сходимостью по сравнению с оригинальным подходом.

§3.4. Q -обучение

Наряду с алгоритмом op-policy существует так называемый off-policy алгоритм или алгоритм *без явной стратегии*. Естественно,

для этого случая можно сгенерировать стратегию в явном виде, выбирая действия с максимальными Q -значениями. Но в этом нет необходимости, если для обновления Q -значений используется вспомогательная исследовательская стратегия μ . Off-policy подход основывается на наблюдениях, которые генерируются на основе вспомогательной стратегии μ , и можно на накопленной с ее помощью статистике сгенерировать несколько оптимальных стратегий (многоагентное обучение) или сделать дополнительную обработку этой статистики для улучшения сходимости (как в случае аппроксимации нейронными сетями). Наиболее известным вариантом алгоритмов без явной стратегии является Q -обучение.

Листинг 5. SARSA с использованием функции исследования среды $f(v, n)$

```

1: function SARSAAGENT( $s', r'$ )                                ▷ глобальные переменные:
2:    $MDP = < S, T, r, \gamma >$                                      ▷ задача МПР
3:    $Q \leftarrow [0, \dots];$                                          ▷ таблица значений полезности действий
4:    $N_{sa} \leftarrow [0, \dots];$                                      ▷ таблица частот пар состояние–действие
5:    $s \leftarrow \emptyset, a \leftarrow \emptyset, r \leftarrow \emptyset;$ 
6:   if  $s \neq \emptyset$  then
7:      $N_{sa}[s, a] += 1;$ 
8:      $a' \leftarrow \pi(s')$ 
9:      $Q[s, a] \leftarrow Q(s, a) + \alpha(N_{sa}[s, a])(r + \gamma Q[s', a'] - Q[s, a])$ 
10:    if  $s'$  - конечное then
11:       $s \leftarrow \emptyset, a \leftarrow \emptyset, r \leftarrow \emptyset;$ 
12:    else
13:       $s \leftarrow s', a \leftarrow \arg \max_{a'} f(Q[a', s'], N_{sa}[a', s']), r \leftarrow r';$ 
return  $a$ 

```

Агент, который принимает решения на основе Q -функции, не требует модель для обучения и выбора действий, т. е. такой агент также свободен от модели (model-free), как и TD -агент. Уравнение Беллмана для значения Q -функции в равновесии записывается как

$$Q(s, a) = r(s) + \gamma \sum_s' T(s, a, s') \max_{a'} Q(a', s'). \quad (3.9)$$

Как и раньше, при наличии модели переходов можно решать эту систему уравнений непосредственно. При отсутствии модели на основе

подхода временных различий уравнение для итерационного обновления значений Q -функции выглядит следующим образом:

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r(s) + \gamma \max_{a'} Q(a', s') - Q(s, a)). \quad (3.10)$$

В листинге 6 представлен вариант Q -агента, который исследует среду с использованием функции $f(v, n)$, аналогичной функции для ADP -агента. Скорость обучения Q -агента существенно меньше, чем ADP -агента, строящего модель среды и старающегося удовлетворить полному набору ограничений для полезности действий. Как показывает практика, методы, основанные на знаниях (агенты, которые явно строят полную модель среды или моделируют какой-то отдельный ее аспект), являются более перспективными для сложных задач, хотя в настоящее время методы, свободные от модели, демонстрируют очень хорошие результаты и являются более универсальными и требующими меньше затрат на создание и обучение.

Листинг 6. Q -обучение с использованием функции исследования среды $f(v, n)$

```

1: function QLEARNINGAGENT( $s', r'$ )
   ▷ глобальные переменные:
2:    $MDP = < S, T, r, \gamma >$            ▷ задача МПР
3:    $Q \leftarrow [0, \dots];$            ▷ таблица значений полезности действий
4:    $N_{sa} \leftarrow [0, \dots];$        ▷ таблица частот пар состояния–действие
5:    $s \leftarrow \emptyset, a \leftarrow \emptyset, r \leftarrow \emptyset;$ 
6:   if  $s \neq \emptyset$  then
7:      $N_{sa}[s, a] += 1;$ 
8:      $Q[s, a] \leftarrow Q(s, a) + \alpha(N_{sa}[s, a])(r + \gamma \max_{a'} Q[a', s'] - Q[s, a])$ 
9:   if  $s'$  - конечное then
10:     $s \leftarrow \emptyset, a \leftarrow \emptyset, r \leftarrow \emptyset;$ 
11:   else
12:      $s \leftarrow s', a \leftarrow \arg \max_{a'} f(Q[a', s'], N_{sa}[a', s']), r \leftarrow r';$ 
return  $a$ 

```

Аналогично с $TD(\lambda)$ и $SARSA(\lambda)$ существует и $Q(\lambda)$ -алгоритм, для которого след применимости имеет следующий вид:

$$e_t(s, a) = \delta_{s_t}(s)\delta_{a_t}(a) + \begin{cases} \gamma\lambda\gamma\lambda e_{t-1}(s, a), & \text{если } Q_{t-1}(s_t, a_t) = \max_a Q_{t-1}(s_t, a) \\ \gamma\lambda 0, & \text{иначе,} \end{cases}$$

где δ – дельта-функция.

Глава 4. Градиентные методы

В данном разделе рассматриваются современные методы обучения с подкреплением, которые, безусловно, основываются на фундаментальных понятиях, введенных в предыдущей главе. Наиболее важными алгоритмами, которые в свою очередь породили большое число модификаций и прикладных применений, являются алгоритмы *актор-критик* и *градиента стратегии*.

§4.1. Актор-критик

Одним из наиболее успешных современных алгоритмов обучения с подкреплением является схема актор-критик. Эта схема, основной идеей которой является отделение блока оценки действия (*критик*) от блока собственно выбора действия (*актор*), имеет глубокие аналогии с работой головного мозга высших животных и человека. Подкорковые структуры, включающие такие подсистемы, как *базальные ганглии*, на основе известного нейромедиатора *дофамина*, реализуют такую же схему при обучении корковых структур.

Алгоритмы семейства актор-критик иногда рассматриваются как метаалгоритмы, которые объединяют в себе преимущества (и некоторые недостатки) алгоритмов-критиков (Critic-only, все рассмотренные ранее) и алгоритмов-акторов (Actor-only, генетические алгоритмы, REINFORCE). Алгоритмы-критики, или алгоритмы, основанные на полезности, всегда строят стратегию на основе функции полезности, в то время как алгоритмы-акторы, основанные на стратегии, напрямую обновляют параметры стратегии, закодированные в некотором параметрическом пространстве, например, весами нейронной сети (см. рис. 4.1).

Формальная реализация схемы актор-критик представляет собой функцию полезности (utility function), которая обновляется на основе вознаграждения, поступающего от среды на каждой итерации

(см. рис. 4.2), и отображения состояния в действия, по которому происходит выбор применяемого на данной итерации действия. Для свободного от модели варианта реализации можно, например, использовать подход на основе временных различий для выражения функции полезности и табличный метод представления отображения состояния–действия с ϵ -жадной стратегией.



Рис. 4.1. Методы, основанные на стратегии (actor-only), и методы, основанные на полезности (critic-only)

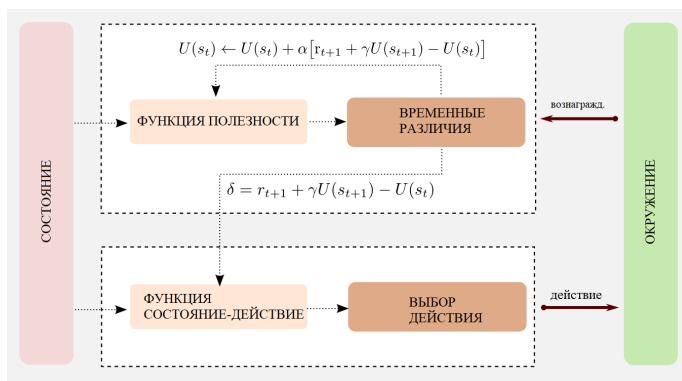


Рис. 4.2. Процесс обучения на основе схемы актор-критик

Общая схема следующая:

- 1) выбор действия a_t для текущего состояния s_t ;

- 2) применение действия, наблюдение нового состояния s_{t+1} и вознаграждения r_t ;
- 3) обновление полезности состояния s_t (критик);
- 4) обновление вероятностей использования действий в состоянии s_t на основе ошибки δ .

В качестве стратегии выбора действия на первом шаге можно использовать функцию softmax:

$$P(a + t = a | s_t = s) = \frac{e^{p(s,a)}}{\sum_b e^{p(s,a)}}.$$

Обучение критика идет по стандартной $TD(0)$ -схеме:

$$V(s_t) = V(s_t) + \alpha \underbrace{(r_t + \gamma V(s_{t+1}) - V(s_t))}_{\delta_t}.$$

Четвертый шаг реализуется уменьшением или увеличением вероятности выбора действия пропорционально определенной ошибке: $p(s_t, a_t) = p(s_t, a_t) + \beta \delta_t$.

По аналогии с методами $TD(\lambda)$ и $SARSA(\lambda)$ можно добавить следы применимости к схеме актор-критика. Особенность здесь заключается в том, что кроме обычного следа $e_t(s)$ для состояния, который используется в функции обновления критика, нам нужен второй след $e_t(s, a)$ для пары состояние–действие:

$$e_t(s, a) = \begin{cases} \gamma \lambda \gamma \lambda e_{t-1}(s, a) + 1, & \text{если } s = s_t, a = a_t; \\ \gamma \lambda \gamma \lambda e_{t-1}(s, a), & \text{иначе.} \end{cases}$$

Он используется при обновлении актора: $p(s_t, a_t) = p(s_t, a_t) + \beta \delta_t e_t(s)$.

Пример работы описанного алгоритма для задачи клеточной среды представлен на рис. 4.3. Для стандартного эпизода: $(1, 1) \rightarrow (1, 2) \rightarrow (1, 3) \rightarrow (1, 2) \rightarrow (1, 3) \rightarrow (2, 3) \rightarrow (3, 3)$, который рассматривается в качестве примера набранной статистики, обновление функции полезности состояний и Q -таблицы представлены на рис. 4.4. Начальные значения полезностей состояний нулевые, а Q -таблица заполняется случайными значениями в диапазоне $[0, 1]$.

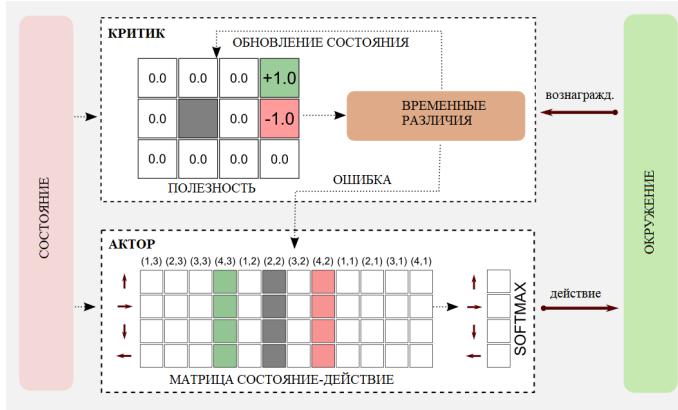


Рис. 4.3. Пример работы схемы актор-критик

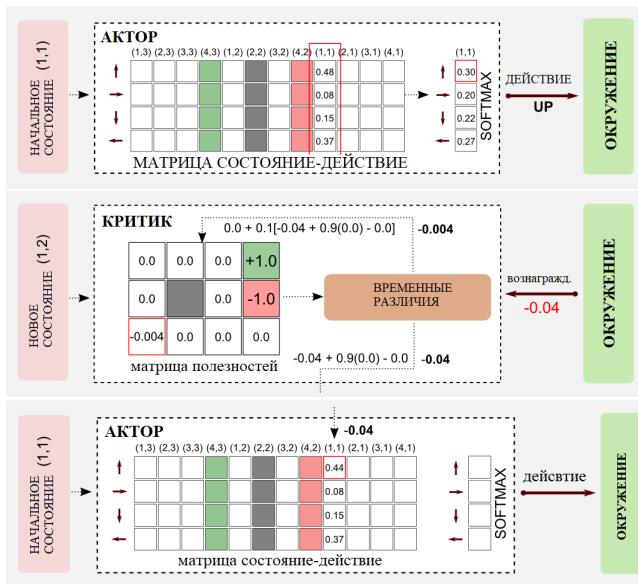


Рис. 4.4. Обновление функции полезности состояний и Q -таблицы в схеме актор-критик с параметрами: $\alpha = 0.1$, $\beta = 1.0$, $\gamma = 0.9$

Преимущество использования схемы актор-критик заключается в том, что актору в явном виде передается ошибка, которая соотносится

ся с выбором действия в том или ином состоянии. Таким образом, при реализации актора можно использовать обучение с учителем и весь класс наработанных в этой области методов. Кроме того, выбор действия в этом подходе требует минимальных вычислительных затрат, что важно при большом количестве доступных действий или когда действия не дискретны, а непрерывны.

§4.2. Аппроксимация и обобщение

Ранее рассматривались так называемые табличные методы, в которых для каждого состояния или пары состояния–действие определялась полезность и сохранялась в таблицу или в список. Однако в тех случаях, когда множество состояний и действий очень велико или даже бесконечно, т. е. формализующий взаимодействие агента среды марковский процесс очень велик, такой табличный подход оказывается неприменимым. Решением этой проблемы является использование приближенного представления функции полезности, когда агент по небольшому количеству наблюдаемых состояний может провести обобщение и вычислить функцию полезности для оставшегося множества состояний и действий.

Для хранения функции полезности состояний $V(s)$ для нашего примера клеточного мира размером $r \times c$ нам понадобится список длиной $r \times c = N$. Для представления функции полезности действий $Q(s, a)$ нам уже будет нужна таблица размерности $N \times M$, где $M = |A|$ – общее количество действий (см. рис. 4.5). Приближение к функции полезности состояний с параметрами w будем обозначать как $\hat{V}(s, w)$, к функции полезности действий – как $\hat{Q}(s, a, w)$. Предполагается, что количество параметров w существенно меньше, чем N и M .

Смысл введения параметризации заключается в том, что можно использовать различные механизмы обучения с учителем и минимизировать некоторый функционал ошибки, подстраивая веса на каждой итерации в сторону уменьшения этой ошибки. Например, если нам известно истинное значение полезности состояния (4.1) $V(4, 1) = 0.388$ (рис. 4.6), можно к текущему значению оценки $\hat{V}((4, 1), w) = 0.352$ добавить небольшое значение Δ в направлении уменьшения ошибки $V(s) - \hat{V}(s, w)$ и получить более точное значение: $\hat{V}((4, 1), w) = 0.371$. Функционал ошибки для задачи обучения с подкреплением должен быть таким, чтобы его уменьшение приводило к более точной оценке функции полезности.

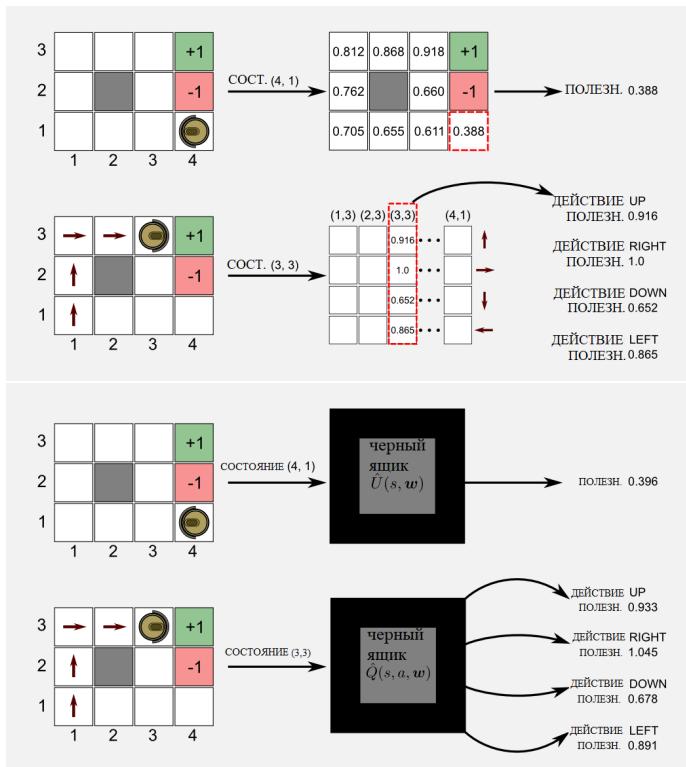


Рис. 4.5. Табличный способ представления функций полезности и использование параметризованного аппроксиматора

Параметрами аппроксиматора могут служить вектор признаков линейной или нелинейной модели либо веса нейронной сети (рис. 4.7).

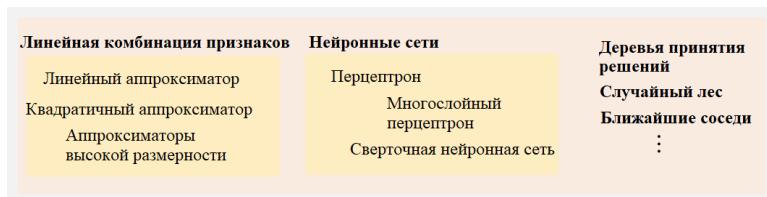


Рис. 4.7. Варианты аппроксиматоров

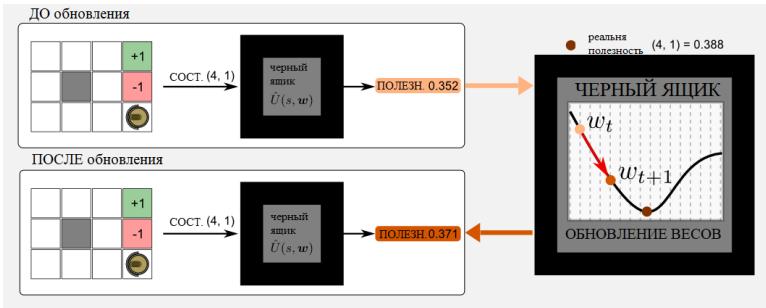


Рис. 4.6. Изменение параметров аппроксиматора для уменьшения ошибки обучения

При определении параметров аппроксиматора нам необходимо задать функцию потерь и способ обновления параметров по текущему значению этой функции (см. рис. 4.8). Одной из наиболее часто используемых функций потерь, или функций ошибок, является среднеквадратичная ошибка (Mean Squared Error – MSE), которая определяется текущей разницей между целевым значением функции полезности и ее аппроксимацией:

$$MSE(w) = \frac{1}{N} \sum_{s \in S} [V^*(s) - \hat{V}(s, w)]^2.$$

Применительно к задаче обучения с подкреплением можно воспользоваться взвешенной суммой квадратов ошибок, используя в качестве весов коэффициенты $\mu(s)$, определяющие важность правильной оценки полезности в состоянии s (так называемой среднеквадратичной ошибкой полезности, MSVE).



Рис. 4.8. Цикл обновления параметров при аппроксимации полезности

В качестве правила обновления параметров w в случае использования дифференцируемого аппроксиматора (например, нейронной сети) применяется *градиентный спуск* (gradient descent). Его идея заключается в том, что агент меняет параметры в направлении, обратном (при решении задачи минимизации) тому, которое задается градиентом функции потерь:

$$w_{t+1} = w_t - \alpha \nabla_w L(w_t),$$

где $L(w)$ – оптимизируемая функция потерь, α – параметр длины шага, задающий скорость обучения. В случае использования среднеквадратичной функции потерь можно посчитать градиент в явном виде:

$$w_{t+1} = w_t - 2\alpha [V^*(s) - \hat{V}(s, w_t)] \nabla_w \hat{V}(s, w_t).$$

При записи процесса обучения выше использовалась схема обучения с учителем, для реализации которой нам необходимо знание истинных значений функции полезности $V^*(s)$. Однако в задаче управления и предсказания в обучении с подкреплением эти целевые значения нам неизвестны, и нам необходимо заменять их некоторыми оценками $\tilde{V}(s)$. Подходы для оценки функции полезности нам уже известны: это метод Монте-Карло и TD -подход.

В методе Монте-Карло нам необходимо накопить как минимум один эпизод взаимодействия агента со средой, чтобы можно было непосредственно посчитать функцию полезности каждого встреченного в эпизоде состояния как $\tilde{V}(s_t) = R_t = \sum_t^T \gamma^{T-t} r_t$. Оценка Монте-Карло является несмешенной (т. е. $\mathbb{E}[\tilde{V}(s)] = V^*(s)$), что гарантирует сходимость этой оценки к истинному значению при достаточном количестве накопленного опыта (эпизодов взаимодействия).

В случае TD -подхода используется *принцип бутстрепа*; оцениваем полезность состояния s_t на основе уже имеющейся оценки полезности следующего состояния: s_{t+1} как $r_{t+1} + \gamma \tilde{V}(s_{t+1})$. Несмотря на то что оценка методом $TD(0)$ является смешенной и сходимость, вообще говоря, не гарантируется, этот подход является интерактивным и не требует накопления опыта по полным эпизодам. В реальных экспериментальных условиях $TD(0)$ демонстрирует большую вычислительную эффективность и сходится быстрее, чем метод Монте-Карло.

Полная схема взаимодействия агента со средой происходит по уже известной нам схеме обобщенной итерации по стратегиям: агент использует текущие оценки полезности: $\hat{V}(s, w)$, или $\hat{Q}(s, w)$, для выбора действия (например, ϵ -жадно), после каждого шага или в конце эпизода агент обновляет веса аппроксиматора с помощью градиентного

шага, получает новые оценки полезности и обновляет свою стратегию и т. д. В качестве примера рассмотрим линейный аппроксиматор: $\hat{V}(s, w) = x(s)^\top w = x_1 w_1 + \dots + x_n w_n$, где $x(s)$ – некоторый вектор признаков состояния s , в качестве которых могут выступать, например, координаты робота, угол и скорость перевернутого маятника, расположение камней на доске в игре Го и т. д. Градиент линейной функции подсчитывается аналитически и равен просто вектору признаков: $\nabla_w \hat{V}(s, w) = x(s)$. В таком случае функция потерь и правило обновления примут вид

$$MSE(w) = \sum_{s \in S} [\hat{V}(s, w) - x(s)^\top w]^2,$$

$$w_{t+1} = w_t - \alpha [r_{t+1} + \gamma x(s_{t+1})^\top w - x(s)^\top w] x(s).$$

§4.3. Метод градиента стратегии

В предыдущем разделе использовалась параметризация и градиентные алгоритмы обновления весов для аппроксимации функции полезности $V(s)$, или $Q(s)$. Однако оказывается, что можно напрямую параметризовать стратегию агента без промежуточного вычисления значений полезности. Такой подход называется *основанным на стратегии* (policy based) и в отличие от методов, *основанных на полезности* (value based), позволяет напрямую представлять стратегию как отображение $\pi : S \rightarrow A$.

Итак, пусть у нас задана некоторая параметризация стратегии: $\pi(s, a) \approx \hat{\pi}(s, a, w)$. Необходимо найти такое значение параметров w , которые обеспечивали бы наилучшую стратегию. Для того чтобы сравнивать стратегии друг с другом, т. е. определять их качество, введем *целевую функцию стратегии* (target function) $J(w)$. Ее можно определять разными способами:

- в эпизодических средах можно использовать полезность начального состояния:

$$J_1(w) = V^{\hat{\pi}(w)}(s_1) = \mathbb{E}_{\hat{\pi}(w)}[R_1];$$

- в непрерывных средах используют среднюю полезность:

$$J_{avV}(w) = \sum_s d^{\hat{\pi}(w)}(s) V^{\hat{\pi}(w)}(s),$$

где $d^{\hat{\pi}(w)}(s)$ – стационарное распределение марковской цепи для стратегии $\hat{\pi}(w)$;

- как среднее вознаграждение:

$$J_{avR}(w) = \sum_s d^{\hat{\pi}(w)}(s) \sum_a \hat{\pi}(w)(s, a, w) r(s, a).$$

Таким образом, задача обучения с подкреплением свелась к задаче оптимизации: для поиска лучшей стратегии необходимо найти значение параметров w , максимизирующих функционал $J(w)$. Для решения задачи оптимизации можно использовать множество методов, в т. ч. и градиентные. При использовании метода градиентного спуска правило обновления весов будет выглядеть как $\Delta w = \alpha \nabla_w J(w)$.

Существует *теорема о градиенте стратегии*, которая связывает градиент целевой функции (любой из трех представленных выше вариантов) и градиент самой стратегии:

$$\nabla_w J(w) = \mathbb{E}_{\hat{\pi}(w)}[\nabla_w \log \hat{\pi}(s, a, w) Q^{\hat{\pi}(w)}(s, a)].$$

Если использовать по методу Монте-Карло в качестве несмешенной оценки $Q^{\hat{\pi}(w)}(s, a)$ отдачу R_t , то тогда происходит переход к алгоритму REINFORCE (см. листинг 7), и обновление весов будет осуществляться по правилу

$$\Delta w_t = \alpha \nabla \log \hat{\pi}(s, a, w) R_t.$$

Листинг 7. REINFORCE

```

1: function REINFORCE
2:   инициализируем  $w$ ;
3:   for каждого эпизода  $\{s_1, a_1, r_2, \dots, s_{T-1}, a_{T-1}, r_T\} \approx \hat{\pi}(w)$  do
4:     for  $t = 1$  и до  $T - 1$  do
5:        $w \leftarrow w + \alpha \nabla_w \log \hat{\pi}(s_t, a_t, w) R_t$ 
return  $w$ 
```

Основными преимуществами методов, основанных на стратегии, являются лучшая сходимость процесса обучения, большая эффективность в задачах высокой размерности, возможность работы с непрерывным множеством действий и возможность обучаться стохастическим стратегиям, которые в некоторых задачах могут приводить к оптимальному решению в отличие от детерминированных стратегий.

В качестве примера можно рассмотреть клеточный мир с неразличимыми состояниями (см. рис. 4.9). В неразличимых состояниях агент при построении детерминированной стратегии вынужден использовать одно и то же действие (пойти налево или направо), что может привести к ситуации, где агент застрянет в одном из углов этого мира и никогда не сможет прийти к награде. В случае же стохастической стратегии агент может определить, что в этих состояниях он выбирает действие случайно, поэтому у него всегда есть возможность выбраться из тупика.

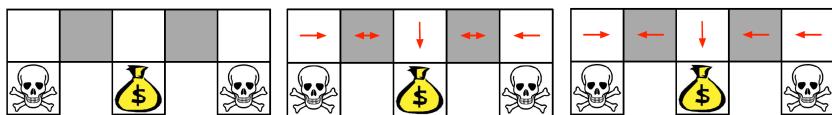


Рис. 4.9. Клеточный мир с парой неразличимых состояний, обозначенных серым цветом (слева), оптимальная стохастическая стратегия (в центре) и неоптимальная детерминированная (справа)

Глава 5. Иерархические методы

Одним из наиболее перспективных направлений в обучении с подкреплением является *иерархический подход*. Идея его очень проста. Многие задачи решаются человеком путем переиспользования знаний, полученных при решении других, похожих задач. В связи с тем, что наша деятельность устроена иерархически, т. е. каждое действие может быть представлено в виде последовательно более мелких операций, которые, в свою очередь, также могут состоять из последовательности еще более «мелких» операций. Например, для задачи построения башни из кубиков нам необходимо уметь поднимать кубики, ставить их на стол или другие кубики. Однако действия по поднятию и опусканию кубиков могут быть переиспользованы для решения задачи составления других конфигураций, например постройки дома из кубиков. Даже сами действия постройки башни могут стать частью более сложного действия по сооружению веранды возле дома из кубиков.

Перечислим следующие основные преимущества иерархического обучения с подкреплением:

- можно использовать все преимущества так называемого переиспользования знаний (*transfer learning*), что может существенно снизить требования к количеству данных для достаточного обучения моделей;
- более эффективно решается проблема больших размерностей пространства действий: декомпозиция и введение иерархической структуры действий позволяет нам фактически снизить размерность;
- получаемые в результате обучения модели обычно более интерпретируемые, т. к. структура действий отражается в структуре решения, в которой можно выделить подцели и подпланы по достижению целей;

- выделение классов состояний и действий позволяет более эффективно решать сложные задачи за счет более эффективного представления и переиспользования знаний.

Концепция иерархического обучения с подкреплением находит свои подтверждения как в других разделах искусственного интеллекта, так и в психологии, и в нейрофизиологии. Например, в теории автоматического планирования также применяются иерархические подходы (иерархические сети подзадач [4] или абстрактные действия [7]). Во всех этих предметных областях находится множество подтверждений в пользу того, что процесс выработки действий и в целом хранение информации организовано иерархических образом, и во многом эффективность действий человека в различных ситуациях основывается на принципе переиспользования абстрактных действий, успешность которых была доказана для других, но схожих задач.

§5.1. Феодальный подход и умения

Для формальной постановки задачи иерархического обучения с подкреплением необходимо учесть, что процесс обучения нужно проводить на различных временных масштабах (temporal abstraction) [14]. Это означает, что кроме доступных агенту элементарных действий, он может использовать сложные абстрактные действия. При этом необходимо учитывать, что в отличие от элементарных действий, выполнение абстрактных действий растягивается на несколько моментов времени (см. рис. 5.1). Таким образом, если a – это примитивное действие, σ – макродействие (подплан), то нам необходимо иметь отдельную стратегию π_σ по выбору обеспечивающих это макродействие операций. Формально это означает, что процесс взаимодействия среды и агента уже моделируется не просто марковским процессом принятия решений (MDP), а полумарковским процессом (SMDP), в котором возможны протяженные во времени действия.

Одним из очевидных подходов к решению SMDP является [5] полная декомпозиция задачи (феодальный подход): верхнеуровневый агент для достижения своей цели (формирования своей стратегии π_σ) выставляет подцели для своих подагентов более низкого уровня иерархии, которые формируют свои собственные стратегии π_a . Получается, что каждый подагент максимизирует свою отдачу и получает максимальное вознаграждение для достижения своей подцели; при этом ему достаточно знать только о тех состояниях, точнее о том уровне абстракции состояний среды, которая необходима для решения его под-

задачи. Несмотря на очевидную интерпретацию идеи иерархии, такой подход не привел к успеху, и перенос знаний демонстрировать не удавалось.

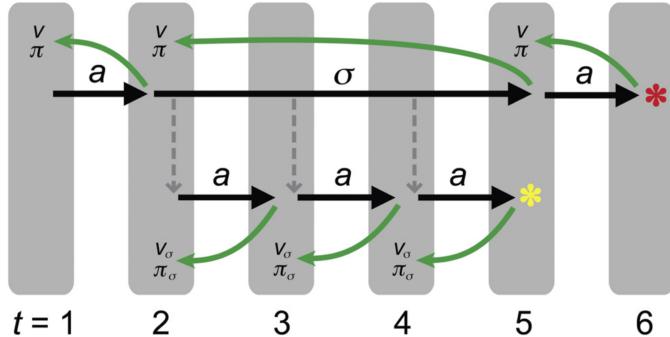


Рис. 5.1. Схема полумарковского процесса принятия решений

Более успешным оказался подход макродействий или *умений* (option framework) [13], [14], который сейчас является одним из наиболее употребительных. В модели умений вводится определение (макровского) умения *o* как тройки $\langle I_o, \pi_o, \beta_o \rangle$, где

- I_o – множество начальных состояний, в которых может начаться выполнение умения;
- $\pi_o : S \times A \rightarrow [0, 1]$ – стратегия, определяющая операционный состав умения;
- $\beta_o : S \rightarrow [0, 1]$ – условия завершения умения.

В качестве примера можно рассмотреть классическую иерархическую задачу поиска пути в лабиринте, представляющем собой несколько комнат с соединяющими их проходами (см. рис. 5.2). Пусть агент находится в одном из проходов между комнатами. При рассмотрении только элементарных действий после первой итерации можно переместиться только на одну клетку рядом с проходом. Если же агент уже обладает умением по достижению проходов в комнатах, то за одну итерацию применения стратегии, включающей такое умение, можно достигнуть выходов из соответствующих комнат.

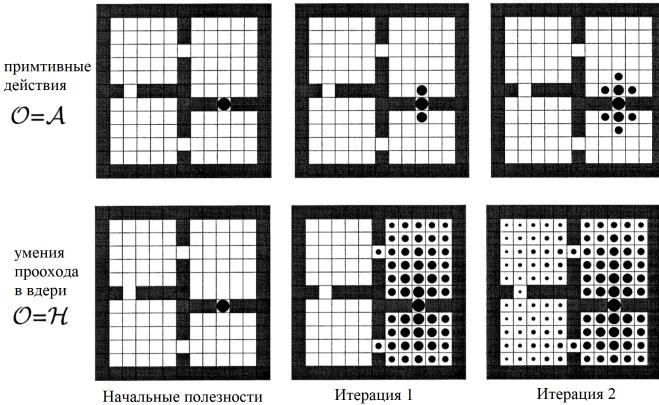


Рис. 5.2. Демонстрация различий между решением задачи на основе примитивных действий и умений

В отличие от элементарного феодального подхода, описанного выше, в модели умений агент может одновременно выбирать действия как из множества элементарных, так и из множества метадействий, что улучшает свойства сходимости.

§5.2. Абстрактные автоматы и НАМ

Еще одним базовым иерархическим подходом является *метод абстрактных автоматов* (hierarchical abstract machines, НАМ-метод) [10]. НАМ представляет собой набор недетерминированных конечных автоматов, переходы между состояниями которых могут приводить к вызову низкоуровневых автоматов. При этом сама последовательность переходов настраивается в процессе обучения. В каждом автомате есть четыре типа состояний:

- *состояния действий* (action states) – переход в этом состояние приводит к выполнению соответствующего действия в среде;
- *состояния вызова* (call states) – приводят к вызову низкоуровневого автомата (выполнению подплана);
- *состояния выбора* (choice states) – недетерминированный выбор следующего состояния автомата;

- *состояния остановки* (stop states) – приводят к завершению выполнения текущего автомата и передачи управления к верхневому состоянию вызова.

Такие автоматы представляют собой простые подпрограммы, комбинация которых приводит к возможности составлять более сложные программы – последовательности действий агента. Обучение заключается в настройке состояния выбора, которое может быть сопоставлено со своей Q -таблицей, по которой происходит выбор оптимального действия в соответствующей ситуации. Например, на рис. 5.3 представлен пример автомата по обходу препятствия. В отличие от модели умений, НАМ также может гарантированно находить оптимальные решения, но он не столь популярен ввиду более сложной реализации.

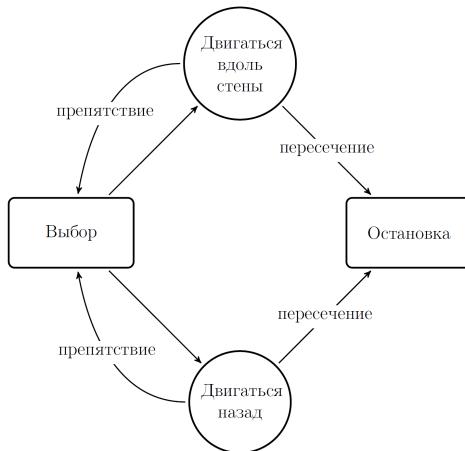


Рис. 5.3. Пример абстрактного автомата для задачи обхода препятствия

К базовым подходам к иерархическому обучению с подкреплением относится и метод MAXQ [6], в котором строится декомпозиция функции полезности по подзадачам: $Q(p, s, a) = R(s, a) + Q(p, s', a')$, где $R(s, a)$ – обычная полезность действия a в состоянии s , а $Q(p, s', a')$ – полное вознаграждение, полученное при завершении подзадачи p . Если a – непримитивное действие, то функция вычисления полезности представляет собой рекурсивную функцию и оптимальность стратегии высокого уровня определяется оптимальным решением дочерних подзадач (так называемая рекурсивная оптимальность). Каждая подзадача характеризуется условиями завершения (терминальным преди-

катом), множеством действий (стратегией) и псевдовознаграждением, что сближает этот подход с простейшим феодальным.

На рис. 5.4 представлен пример иерархии подзадач для задачи «Такси», в которой водителю необходимо подобрать пассажира и доставить его в необходимую точку в клеточном окружении.



Рис. 5.4. Декомпозиция задачи «Такси» для решения ее методом MAXQ

Как и все ранее рассмотренные подходы к иерархическому обучению с подкреплением, MAXQ требует заранее заданной структуры подзадач, и при некоторых условиях рекурсивно-оптимальное решение может оказаться далеко от оптимального.

Заключение

В настоящем пособии были рассмотрены базовые принципы и алгоритмы машинного обучения с подкреплением. Основное внимание было уделено формальным постановкам задач. Были введены определения агента, его стратегии, среды, вознаграждения и марковского процесса взаимодействия агента со средой; при этом целью агента является максимизация суммарного вознаграждения – отдачи. Были рассмотрены основные особенности применения метода динамического программирования для решения марковского процесса принятия решений при известных матрицах переходов. В самой задаче обучения с помощью методов, основанных на полезности, стандартной является схема обобщенных итераций по стратегиям, в которой вначале производится оценка полезности состояний по имеющейся стратегии, а затем происходит обновление текущей стратегии по найденным полезностям.

В пособии рассмотрены методы аппроксимации функций полезности и собственно стратегии, которые позволяют свести задачу обучения с подкреплением к задаче обучения с учителем. В завершение пособия более подробно представлено одно из новых перспективных направлений в машинном обучении с подкреплением – иерархический подход.

Данное пособие будет полезно студентам и аспирантам, специализирующимся по направлению искусственного интеллекта и, в частности, может выступать в качестве дополнительных глав для курсов машинного обучения и интеллектуальных систем в робототехнике.

Литература

1. *Bellman R.E., Dreyfus S.E.* Applied Dynamic Programming: tech. rep. London: The RAND Corporation, 1962. 384 p.
2. *For N., et al.* Clyde: A deep reinforcement learning DOOM playing agent // What's Next For AI In Games. 2017.
3. *Oh J., et al.* Control of Memory, Active Perception, and Action in Minecraft // arXiv.org. 2016. URL: <https://arxiv.org/abs/1605.09128>
4. *Currie K., Tate A.* O-Plan: The open planning architecture // Artificial Intelligence. 1991. V. 52, N 1. P. 49–86.
5. *Dayan P., Hinton G.* Feudal Reinforcement Learning // Advances in neural information processing systems. 1993. P. 271–278.
6. *Dietterich T.G.* Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition // Journal of Artificial Intelligence Research. 2000. V. 13. P. 227–303. URL: <https://arxiv.org/abs/9905014>
7. *Knoblock C.A.* Learning abstraction hierarchies for problem solving. 1990.
8. *Laird J.E.* The Soar Cognitive Architecture. MIT Press, 2012. 374 p.
9. *Silver D., et al.* Mastering the game of Go with deep neural networks and tree search // Nature. 2016. V. 529, N 7587. P. 484–489.
10. *Parr R., Russell S'.* Reinforcement learning with hierarchies of machines // Neural Information Processing Systems (NIPS). 1998. P. 1043–1049.
11. *Mnih V., et al.* Playing Atari with Deep Reinforcement Learning // arXiv.org. 2013. P. 1–9. URL: <https://arxiv.org/abs/1312.5602>
12. *Vinyals O., et al.* StarCraft II: A New Challenge for Reinforcement Learning // arXiv. 2017. URL: <https://arxiv.org/abs/1708.04782>
13. *Stolle M., Precup D.* Learning Options in Reinforcement Learning // Proceedings of Abstraction, Reformulation, and Approximation, 5th International Symposium, SARA 2002. Springer, 2002. P. 212–223.
14. *Sutton R.S., Precup D., Singh S.* Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning // Artificial Intelligence. 1999. V. 112. P. 181–211. URL: <https://arxiv.org/abs/1011.1669v3>

15. *Szepesvari C.* Algorithms for Reinforcement Learning. V. 4. 2010. P. 1–103.
16. *Waltz M., Fu K.*A heuristic approach to reinforcement learning control systems // Automatic Control, IEEE Transactions on. 1965. V. AC-10, N 4. P. 390–398.
17. *Осипов Г.С. [и др.]*. Знаковая картина мира субъекта поведения. Москва: Физматлит, 2018. 264 с.
18. *Осипов Г.С.* Методы искусственного интеллекта. Москва: Физматлит, 2011. 297 с.
19. *Rascel C., Norvig P.* Искусственный интеллект: современный подход. 2-е изд, Москва : Издательский дом «Вильямс», 2006. 1408 с.
20. *Саттон Р.С., Барто Э.Г.* Обучение с подкреплением. 2-е изд. Москва : БИНОМ. Лаборатория знаний, 2011. 399 с.
21. *Степанюк В.Л.* Локальная организация интеллектуальных систем. Москва : Физматлит, 2004. 328 с.

Учебное издание

Панов Александр Игоревич

ВВЕДЕНИЕ В МЕТОДЫ МАШИННОГО
ОБУЧЕНИЯ С ПОДКРЕПЛЕНИЕМ

Редактор *Н. Е. Кобзева*. Корректор *И. А. Волкова*
Компьютерная верстка: *А. И. Панов, Н. Е. Кобзева*
Дизайн обложки *Е. А. Казённова*

Подписано в печать 25.06.2019. Формат 60×84 $\frac{1}{16}$.
Усл. печ. л. 3,25. Уч.-изд. л. 2,5. Тираж 100 экз. Заказ № 206.

Федеральное государственное автономное образовательное учреждение
высшего образования «Московский физико-технический институт
(национальный исследовательский университет)»
141700, Московская обл., г. Долгопрудный, Институтский пер., 9
Тел. (495) 408-58-22, e-mail: rio@mipt.ru

Отдел оперативной полиграфии «Физтех-полиграф»
141700, Московская обл., г. Долгопрудный, Институтский пер., 9
E-mail: polygraph@mipt.ru