Глава 5

Стимулируемое обучение

§ 5.1. Введение и постановка задачи

В предыдущих главах мы рассматривали задачи обучения на некотором наборе тестовых примеров. Иначе говоря, до сих пор задача ставилась так: есть набор «правильных ответов», а задача обучения состоим в том, чтобы его продолжить на всё пространство.

Но разве так работает обучение в реальной жизни? Мы далеко не всегда знаем набор правильных ответов заранее, мы просто делаем то или иное действие и получаем результат. Если программа обучается игре в шашки, человек не сможет сказать ей, какой ход единственно правильный в сложившейся ситуации. Всё, что у программы получится выяснить, — это то, выиграла она в итоге или проиграла. Если робот учится ходить, человек не сможет сказать ему, какой именно мускул и насколько нужно напрячь в данную секунду — а просто скажет, получилось сделать шаг или нет.

Такие ситуации, близкие к реальному обучению, можно охарактеризовать единой общей моделью. В этой модели обучающийся агент взаимодействует с окружающей средой, предпринимая какие-то действия (выбирая их, конечно, из фиксированного набора); окружающая среда его каким-то образом поощряет за эти действия, а агент продолжает их предпринимать. Единственный способ для агента понять, что он делает правильно, — следить за поощрениями от окружающей среды. Такой вид машинного обучения называется *стимулирующим обучением* (reinforcement learning).

Попробуем поставить задачу формально. Пусть на каждом шаге агент может находиться в состоянии s из некоторого множества состояний S. На каждом шаге он выбирает из имеющегося набора действий A некоторое действие a. В ответ на это окружающая среда сообщает агенту, какую награду r он за это получил и в каком состоянии s' после этого оказался.

Пример 5.1. Диалог обучающегося агента и окружающей среды.

Среда: Агент, ты в состоянии 1; есть 5 возможных действий.
Агент: Делаю действие 2.
Среда: Даю тебе 2 единицы за это. Попал в состояние 5, есть 2 возможных действия.
Агент: Делаю действие 1.
Среда: Даю тебе за это -5 единиц. Попал в состояние 1, есть 5 возможных действий.

Агент: Делаю действие 4.
Среда: Даю тебе 14 единиц за это. Попал в состояние 3, есть 3 возможных действия...

В этом примере агент успел вернуться в состояние 1 и исследовать ранее не пробовавшуюся опцию 4 (получив за это существенную награду). Так и в общем случае — агент должен исследовать окружающую среду и выбирать оптимальное поведение.

Знакомому с теорией вероятностей человеку уже очевидно, что дело пахнет марковскими процессами. Действительно, мы к ним ещё вернёмся, и они будут играть важную роль в построении новых алгоритмов, но пока обратимся к тому, как сравнивать и оценивать алгоритмы стимулируемого обучения.

§ 5.2. Как оценивать поведение агента?

Мы исследуем алгоритмы, которые обучаются получать награду от окружающей среды. Разумеется, мы хотим получить алгоритмы, которые «хорошо себя ведут», то есть получают большую награду. Но то такое «хорошо»? Как оценивать поведение алгоритма в приведённом выше сеттинге?

На этот вопрос есть несколько ответов, выбор между которыми зависит от того, на какой срок жизни агента мы рассчитываем и на что хотим сделать упор в своих оценках.

Первая, простейшая модель — это так называемая *модель конечного горизонта* (finite horizon model). В ней качество поведения агента измеряется только по отношению к следующим h шагам, и максимизировать нужно величину

$$\mathrm{E}\left[\sum_{t=0}^{h} r_t\right].$$

Эта модель соответствует ситуации, когда у агента ограничен срок жизни: мы знаем, что у нас всего десять попыток, и поэтому нас не интересует, успешно ли мы обучимся производить одиннадцатую.

В более естественной ситуации, однако, хотелось бы учесть все возможные шаги в будущем, потому что время жизни агента может быть не определено. Но при этом в большинстве реальных задач чем раньше мы получим награду от окружающей среды, тем лучше. Например, рубль сегодня — это гораздо лучше, чем рубль через год, ведь его за это время можно разумно вложить и приумножить[1].

Как это учесть в математической модели? Для этого достаточно придать более быстрой прибыли больший вес, а более отдалённой во времени — меньший. Так мы

[1] «Нет! расчёт, умеренность и трудолюбие: вот мои три верные карты, вот что утроит, усемерит мой капитал...» — говорил пушкинский Герман; правда, впрок ему это не пошло.

одновременно убьём двух зайцев, ведь за счёт введённого коэффициента γ ещё и возникающий при суммировании по бесконечной длине жизни агента ряд начнёт сходиться (мы считаем, что выплаты r_t ограничены сверху — в конце концов, у нас конечное число разных состояний и разных выплат). Итак, мы подсчитываем математическое ожидание суммы ряда

$$E\left[\sum_{t=0}^{\infty}\gamma^t r_t\right],$$

где γ — некоторая константа (называемая в англоязычных текстах discount factor).

Такая модель называется *моделью бесконечного горизонта* (infinite horizon model). В нашем дальнейшем рассмотрении мы будем в основном пользоваться именно ею.

Третья, реже встречающаяся, модель — *модель среднего вознаграждения* (average-reward model). В ней максимизации подлежит предел ожидания среднего вознаграждения

$$\lim_{h\to\infty} E\left[\frac{1}{h}\sum_{t=0}^{h} r_t\right].$$

Эта модель не даёт быстрой прибыли преимущества перед отложенной, поэтому на практике нечасто оказывается лучше модели бесконечного горизонта.

Пример 5.2. Различия между моделями оценки качества алгоритмов.

Все вышеописанные модели разные, и несложно построить граф состояний, на котором все три модели будут приводить к разным оптимальным стратегиям.

Кроме поведения уже готового алгоритма, нужно также научиться оценивать и качество его обучения — то, насколько качественно он стремится к своему пределу.

Первейшее (и обычно не так уж сложно доказуемое) свойство алгоритмов стимулируемого обучения — сходимость к оптимальному (для данной модели, конечно). Это полезное свойство, но, к сожалению, оно ничего не говорит ни о скорости сходимости, ни о том, с какими потерями мы подойдёи к оптимальному алгоритму.

Более разумны оценки скорости сходимости. Здесь возможны два альтернативных подхода: оценки скорости сходимости к какой-то фиксированной доле оптимальности (собственно оптимального алгоритма обучающийся агент обычно достигает лишь на бесконечности) и оценки качества работы алгоритма по истечении некоторого фиксированного времени. К сожалению, в обоих случаях возникают конкретные вопросы: какая нужна доля оптимальности? После какого времени нужно сравнивать качество алгоритмов? Однозначный ответ на эти вопросы найти трудно.

Третий возможный подход — минимизировать цену (regret) выбора той или иной стратегии обучения. Любая обучающаяся стратегия не может быть лучше следования оптимальной стратегии с самого начала. Иными словами, задача оптимального обучения — это задача минимизации разницы между общей суммой выигрыша при обучении по сравнению с применением оптимальной стратегии с самого начала. Если нам удастся подсчитать эту разницу, то можно очень просто сравнивать алгоритмы обучения — у кого разница меньше, тот и победил. Это очень хорошая мера, ведь, собственно, она и является настоящей конечной целью происходящего. Но, к сожалению, она плохо поддаётся анализу: разумные результаты об этой мере получить очень сложно.

И последнее замечание, прежде чем мы перейдём к непосредственному изучению отдельных алгоритмов. Каждый алгоритм стимулируемого обучения должен и изучать окружающую среду, и пользоваться своими знаниями, чтобы максимизировать прибыль. Возникает вопрос — как достичь оптимального соотношения? Та или иная стратегия может быть хороша, но вдруг она не оптимальная, и нужно искать другую? А, может быть, она уже идеальна, и нужно просто следовать ей, не тратя времени на бессмысленные поиски? Эта дилемма (на английском она носит красивое название «exploitation vs. exploration») всегда присутствует в стимулируемом обучении. Однозначного ответа на этот вопрос, конечно, нет и быть не может.

§ 5.3. Многорукие бандиты

Будем двигаться от простого к сложному. Первой остановкой на этом пути станут агенты, у которых есть только одно состояние. Формально они ничем не отличаются от введённых в § 5.1, но $|S| = 1$. У такого агента есть фиксированный набор действий A и возможность выбора из этого набора действий.

В искусственном интеллекте общепринята изящная и доступная для понимания модель поведения такого агента. Предположим, что агент находится в комнате с несколькими игровыми автоматами («однорукими бандитами», далее — просто «бандитами»). У каждого бандита своё, неизвестное агенту ожидание выигрыша. Агент должен за ограниченное количество попыток получить максимальную прибыль.

Отметим, что такая (казалось бы, сильно упрощённая) модель действительно реализует все возможные ситуации с одним состоянием. Действительно, за каждое предпринятое действие среда даёт вознаграждение (возможно, случайное) и возвращает агента в исходное состояние (результат следующего подхода к бандиту не зависит от предыдущих).

Прежде чем перейти к рассмотрению конкретных алгоритмов, рассмотрим общие принципы, которые должны руководить агентом в такой ситуации. Разумеется, хочется применить в каком-то смысле «жадную» стратегию, т.е. всегда выбирать

стратегию, максимизирующую прибыль. Однако совершенно очевидно, что агент в комнате не знает с самого начала, что ему делать, и не может выбрать самую лучшую стратегию. Более того, даже если он уже дёрнул по одному разу за ручку каждого бандита и только один раз победил, это вовсе не означает, что нужно теперь всё время выбирать только принесший прибыль автомат. Как же соблюсти баланс между изучением окружающей среды и непосредственным извлечением прибыли?

Как ни странно, единый ответ существует. Он вполне логичен: полезная эвристика в такой ситуации — *оптимизм при неопределённости*. Иначе говоря, выбирать стратегию поведения нужно жадно, но при этом следует быть весьма оптимистичным относительно ожидаемой прибыли. Должны быть получены серьёзные отрицательные свидетельства, прежде чем та или иная стратегия может быть отклонена.

Чуть позже мы увидим, как эта рекомендация реализуется на практике, а пока рассмотрим методы, которые *гарантированно* находят оптимальное решение или сходятся к нему.

§ 5.4. Доказуемо оптимальные алгоритмы

В этом параграфе мы рассматриваем алгоритмы, анализ работы которых проводится достаточно несложно и приводит к хорошим результатам. Это, правда, вовсе не означает, что мы докажем все утверждения о рассмотренных здесь алгоритмах, но значительная их часть будет непосредственно очевидной.

Динамическое программирование широко используется для самых различных задач. Не стало исключением и стимулируемое обучение. Динамическое программирование здесь можно применить, если заранее известен срок жизни агента: предположим, что агент действует на протяжении h шагов.

Тогда для определения оптимальной стратегии можно использовать несложный байесовский подход. Нужно вычислить отображение из всех возможных *состояний опыта* (belief states) агента во множество действий, таким образом задав ему стратегию поведения.

Состояние опыта агента выражается как $\mathcal{S} = \{n_1, w_1, \ldots, n_k, w_k\}$, что означает, что каждого бандита i запустили n_i раз, получив при этом выигрыш w_i раз (здесь и далее мы считаем, что результат бинарный — либо выиграл, либо проиграл).

Обозначим через $V^*(\mathcal{S})$ ожидаемый оставшийся выигрыш для данного опытного состояния. Базой для рекуррентных соотношений будет служить случай, когда $\sum_{i=1}^{k} n_i = h$. В этом случае агенту больше нечего делать, и $V^*(\mathcal{S}) = 0$. Если же мы знаем V^* для всех состояний, когда у агента ещё осталось t попыток, мы сможем пересчитать V^* и для состояний с $t+1$ оставшейся попыткой:

$$V^*(n_1, w_1, \ldots, n_k, w_k) =$$

`DynamicReinforcement`(h, k):

1. Инициализировать таблицу $\{V^*(n_1, w_1, \ldots, n_k, w_k)\}_{n_i, w_i=1}^{h}$.
2. Для всех элементов, для которых $\sum_{i=1}^{k} n_i \geq h$, присвоить $V^*(n_1, w_1, \ldots, n_k, w_k) := 0$.
3. Для всех t от h до 0:
   а) Для всех элементов, для которых $\sum_{i=1}^{k} n_i = t$, присвоить
$$V^*(n_1, w_1, \ldots, n_k, w_k) := \\ := \max_i \left\{ \tfrac{w_i+1}{n_i+2} \left(1 + V^*(\ldots, n_i + 1, w_i + 1, \ldots)\right) + \right. \\ \left. + \left(1 - \tfrac{w_i+1}{n_i+2}\right) V^*(\ldots, n_i + 1, w_i, \ldots) \right\}.$$
4. На каждом шаге с данным фиксированным $(n_1, w_1, \ldots, n_k, w_k)$ выбирать действие i, для которого достигается
$$\max_i \left\{ \tfrac{w_i+1}{n_i+2} \left(1 + V^*(\ldots, n_i + 1, w_i + 1, \ldots)\right) + \right. \\ \left. + \left(1 - \tfrac{w_i+1}{n_i+2}\right) V^*(\ldots, n_i + 1, w_i, \ldots) \right\}.$$

Рис. 5.1. Динамическое программирование для стимулируемого обучения.

$$= \max_i \left(\rho_i(1 + V^*(\ldots, n_i + 1, w_i + 1, \ldots)) + (1 - \rho_i)V^*(\ldots, n_i + 1, w_i, \ldots)\right),$$

где ρ_i — апостериорные вероятности того, что действие i оправдается, при данных опыта (n_i, w_i). В случае с испытаниями Бернулли в виде одноруких бандитов $\rho_i = \frac{w_i+1}{n_i+2}$.

Получившийся алгоритм представлен на рис. 5.1. Как видно, он получился не слишком-то эффективным: цена построения такой таблицы линейна от произведения количества состояний опыта на количество действий $|A|$, что экспоненциально зависит от h. Иначе говоря, динамическое программирование можно использовать только в случаях, когда нужно планировать не слишком далеко. Но зато такой подход гарантированно приводит к оптимальной стратегии.

Вы, наверное, уже обратили внимание, что предвычисленная таблица V^* совершенно не зависит от конкретной ситуации — её можно вычислить хоть раз и навсегда, а от конкретных ожиданий выигрыша конкретных одноруких бандитов зависеть будет только путь по этой таблице, который изберёт оптимальная стратегия. Может быть, можно сделать всё заранее, раз и навсегда, а потом брать значения из таблицы? Да, можно.

Пусть мы уже n раз дёрнули за ручку какого-то бандита и получили w единичек. Существуют предвычисленные таблицы *индексов распределения Гиттинса* (Gittins allocation indices) $I(n, w)$, которые учитывают как ожидаемую прибыль, так и

`LinearRewardInaction`(α):

1. Случайно инициализировать вероятности p_i.
2. Пока агент продолжает работу:
   а) Выбрать действие i с вероятностью p_i.
   б) Если действие привело к успеху:

$$\begin{aligned} p_i &:= p_i + \alpha(1 - p_i), \\ p_j &:= p_j - \alpha p_j, \quad j \neq i. \end{aligned}$$

Рис. 5.2. Алгоритм линейного вознаграждения–бездействия.

количество новой информации, которую мы получим, если предпримем это действие ещё раз. Таким образом, оптимальная стратегия проста до безобразия: достаточно просто выбирать бандита, для которого $I(n, w)$ максимален.

Индексы распределения Гиттинса работают отлично, но, к сожалению, никак не обобщаются — их аналоги для нескольких состояний агента неизвестны.

Третий приём из тех, анализ которых провести достаточно просто, связан с «тренировкой» оптимальной стратегии. Действительно, в модели с комнатой с однорукими бандитами любой алгоритм можно представить как набор вероятностей, с которыми он выбирает то или иное действие (всё остальное не важно для конечного результата). И эти вероятности можно тренировать примерно так же, как мы в Главе 2 тренировали перцептроны. *Алгоритм линейного вознаграждения–бездействия* (linear reward–inaction algorithm) линейно увеличивает вероятность действия a_i, если оно привело к успеху:

$$\begin{aligned} p_i &:= p_i + \alpha(1 - p_i), \\ p_j &:= p_j - \alpha p_j, \quad j \neq i, \end{aligned}$$

а если оно безуспешно, то вероятности сохраняются. В этих формулах α — константа, задающая скорость обучения (вспомните аналогичную константу в алгоритмах обучения нейронных сетей). Алгоритм также представлен на рис. 5.3.

Алгоритм линейного вознаграждения–бездействия с вероятностью 1 сходится к вектору из одной единички и остальных нулей. Он не всегда сходится к оптимальной стратегии; вполне вероятно, что не вполне характерные первые несколько запусков «обучат» алгоритм настолько, что он уже никогда не вернётся к оптимальному варианту, а доведёт до единицы вероятность субоптимального выбора. Но вероятность ошибиться можно сделать сколь угодно малой, уменьшая α.

§ 5.5. Эвристические стратегии

Кроме алгоритмов, которые доказуемо сходятся к оптимальному решению, бывают также ситуации, в которых оптимальность совершенно не гарантирована, но на практике оказывается, что метод работает хорошо.

Мы рассмотрим два представителя этого семейства стратегий. Первый из них — случайные стратегии. Простейшая случайная стратегия выглядит так: выбрать действие с наилучшей ожидаемой прибылью с вероятностью p, а с вероятностью $1 - p$ выбрать случайное действие. Чтобы вероятность сойтись к субоптимальному действию не зашкаливала, обычно начинают с маленьких p, а затем увеличивают.

Основной недостаток этого алгоритма очевиден: он выделяет лидера, но не отличает хорошую его альтернативу от бесполезной. Чтобы принимать во внимание различия между всеми стратегиями, нужно «размазать» вероятность по всем действиям, принимая во внимание их ожидаемую прибыль, но при этом не запрещая дополнительные исследования. Одним из подходящих методов является *исследование по Больцману*[2] (Boltzmann exploration):

$$p(a) = \frac{e^{ER(a)/T}}{\sum_{a'} e^{ER(a')/T}},$$

где ER — ожидаемая прибыль, T — *температура* (фиксированная константа). Температура определяет, насколько велика разница между вероятностями выбора хороших и плохих стратегий; очевидно, что чем выше температура, тем ближе вероятности друг к другу, а чем ниже, тем более вероятен выбор оптимального и близких к нему действий. Здесь тоже обычно температура начинается с достаточно высокого значения, а затем со временем система «остывает».

Вторая эвристическая стратегия, которую мы рассмотрим, — представитель класса «оптимистично жадных» алгоритмов. Предположим, что вам нужно выбрать стратегии достаточно оптимистично, но при этом всё же учитывать имеющийся негативный опыт. Теория вероятностей тут же предлагает весьма разумный выход — доверительные интервалы!

Для каждого действия нужно хранить статистику n и w, и перед принятием решения вычислять доверительный интервал для вероятности успеха (с некоторой наперёд заданной границей $1 - \alpha$), а для выбора реализуемого действия максимизировать верхнюю границу этого интервала.

[2] *Людвиг Эдуард Больцман* (Ludwig Eduard Boltzmann, 1844–1906) — австрийский физик, один из отцов–основателей статистической механики и статистической термодинамики. Разумеется, ни о каком искусственном интеллекте Больцман ни сном ни духом не слыхивал; но случайные процессы, которые происходят в этом алгоритме, весьма напоминают статистическую физику; о том же напоминает и термин «температура».

Пример 5.3. Пример вычисления доверительных интервалов.

Предположим, что мы находимся в ситуации с k однорукими бандитами, каждый из которых производит бинарный результат (1 или 0). С точки зрения теории вероятностей это значит, что у нас имеются k серий испытаний Бернулли с разными средними (разными ожиданиями выигрыша). Будем подсчитывать доверительные интервалы среднего значения с вероятностью .95 (напомним, что семантика доверительного интервала — «интересующая нас случайная величина лежит в доверительном интервале с заданной вероятностью», в данном случае с вероятностью .95). Тогда среднее лежит в интервале

$$\left(\bar{x} - 1.96\frac{s}{\sqrt{n}}, \bar{x} + 1.96\frac{s}{\sqrt{n}}\right),$$

где 1.96 берётся из таблиц распределения Стьюдента, n — количество испытаний, $s = \sqrt{\frac{\sum (x-\bar{x})^2}{n-1}}$.

Предположим, что мы уже двадцать раз выбирали первого бандита и получили двенадцать положительных ответов ($\bar{x} = 0.6$), а второго бандита выбирали шесть раз и получили три положительных ответа ($\bar{x} = 0.5$). Тогда

$$\begin{aligned} s_1 &= \sqrt{\frac{12\times(1-0.6)^2+8\times(0-0.6)^2}{19}} \approx 0.503, \\ s_2 &= \sqrt{\frac{3\times(1-0.5)^2+3\times(0-0.5)^2}{4}} \approx 0.612, \end{aligned}$$

и в итоге доверительный интервал для первого бандита составляет

$$\left(0.6 - 1.96\frac{0.503}{\sqrt{20}}, 0.6 + 1.96\frac{0.632}{\sqrt{20}}\right) \approx (0.323, 0.877),$$

а для второго —

$$\left(0.5 - 1.96\frac{0.612}{\sqrt{5}}, 0.5 + 1.96\frac{0.612}{\sqrt{5}}\right) = (0, 1).$$

Разброс доверительного интервала во втором случае оказался настолько велик, что имеющийся опыт (шесть опытов, три удачи) вообще ничего нам не говорит о доверительном интервале для среднего с вероятностью 0.95. Получается, что, несмотря на то, что средний выигрыш, казалось бы, выше у первого бандита, выбирать по алгоритму доверительных интервалов следует второго — хотя бы с целью дальнейшего исследования этого варианта.

§ 5.6. Модель агентов с несколькими состояниями

В предыдущих параграфах мы рассматривали простейшую ситуацию — когда у агента ровно одно состояние, т.е. на протяжении всей работы алгоритма множество действий и их исходов не меняется. Однако в реальной жизни обучающимся алгоритмам обычно нужно переходить из одного состояния в другое, прежде чем будет достигнут какой-то определённый результат. Таким образом, нам требуется построить адекватную модель, описывающую переходы между этими состояниями и пригодную для максимизации полученной прибыли. Читатели, знакомые с теорией вероятности, наверняка уже догадались, что мы сейчас будем вести речь о марковских процессах.

ConfidenceIntervalReinforcement(α):

1. Инициализировать доверительные интервалы $I_k = (I_{1k}, I_{2k}) := (0, 1)$.
2. Пока агент продолжает работу:
   а) Выбрать действие $j = \operatorname{argmax}_{i=1..k} I_k^2$.
   б) Пересчитать доверительный интервал I_j в зависимости от результатов этого действия.

Рис. 5.3. Алгоритм стимулируемого обучения методом доверительных интервалов.

Определение 5.1. Марковский процесс принятия решений *(Markov decision process) состоит из:*

- *множества состояний* S;
- *множества действий* A;
- *функции поощрения* $R : S \times A \to \mathbb{R}$;
- *функции перехода между состояниями* $T : S \times A \to \Pi(S)$, *где* $\Pi(S)$ — *множество распределений вероятностей над* S.

Таким образом, вероятность попасть из состояния s *в состояние* s' *после совершения действия* a *обозначается через* $T(s, a, s')$.

Отметим, что в нашей модели переходы между состояниями вероятностные: после того или иного действия новое состояние наступает не с абсолютной необходимостью, а с некоторым (заданным) распределением вероятностей. А вот функцию награды мы для простоты считаем постоянной — это не влияет на рассматриваемые нами алгоритмы.

Прилагательное *марковский* в теории вероятностей (и у нас) означает, что переходы не зависят от истории предыдущих переходов, т.е. находясь в определённом состоянии, агент будет за одни и те же действия получать одну и ту же награду, вне зависимости от того, каким путём агент пришёл в это состояние.

Наша задача, как водится, состоит в том, чтобы максимизировать прибыль. Разумеется, в реальной ситуации мы в начале процесса находимся в абсолютном «тёмном лесу» — нам не известна реакция системы ни на какие наши действия, в том числе и переходы между состояниями.

Однако давайте на минутку предположим, что мы уже точно знаем нашу модель. Задача поиска оптимальную стратегию поведения для агента в этой модели тоже не лишена смысла и отнюдь не тривиальна. Более того, может оказаться так, что решение этой задачи будет необходимо для решения более общей. Поэтому сначала давайте рассмотрим алгоритмы, которые находят оптимальную стратегию для

`ValueIteration()`:

1. Инициализировать $V(s)$.
2. Пока стратегия недостаточно хороша:
   а) Для всех $s \in S$ и $a \in A$
   $$Q(s,a) := R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V(s').$$
3. $V(s) := \max_a Q(s,a)$.

Рис. 5.4. Поиск оптимального значения состояния итерациями по значениям.

известной модели, а потом, уже вооружённые этими методами, приступим к более сложной задаче обучения.

§ 5.7. Поиск оптимальных стратегий в известной модели

Начнём с определения одного из основных понятий этого параграфа.

Определение 5.2. Оптимальное значение состояния — *ожидаемая суммарная прибыль, которую получит агент, если начнёт с этого состояния и будет следовать оптимальной стратегии:*

$$V^*(s) = \max_\pi E\left[\sum_{t=0}^{\infty} \gamma^t r_t\right].$$

Иными словами, оптимальное значение состояния — это та награда, которую мы получим, если будем играть наилучшим образом. Это значение можно определить как решение уравнений

$$V^*(s) = \max_{a \in A}\left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V^*(s')\right).$$

Если её знать, то выбирать оптимальную стратегию уже несложно:

$$\pi^*(s) = \operatorname{argmax}_a\left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V^*(s')\right).$$

Осталось только понять, как же решать эти уравнения.

На рис. 5.4 и 5.5 приведены два очень похожих алгоритма, которые решают эту задачу итеративным способом. Первый из них проводит итерации по значениям, т.е. пытается на каждом шаге подсчитывать функцию $Q(s,a)$ — текущую версию ожидания прибыли в случае выбора действия a в состоянии s — а затем, когда значения

`PolicyIteration()`:

1. Инициализировать π.
2. Повторять:
   а) Вычислить значения состояний для стратегии π, решив систему линейных уравнений
   $$V_\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V_\pi(s').$$
   б) Улучшить стратегию на каждом состоянии:
   $$\pi'(s) := \operatorname{argmax}_a \left(R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V_\pi(s') \right).$$
3. пока $\pi \neq \pi'$.

Рис. 5.5. Поиск оптимального значения состояния итерациями по стратегиям.

этой функции нас уже удовлетворяют, выбирает то действие, которое максимизирует это ожидание. Проблема, как водится, в том, когда же можно остановить алгоритм. Обычно его останавливают тогда, когда значения $Q(s, a)$ перестают изменяться (т.е. разность между последовательными значениями по модулю перестаёт превышать некоторый заранее фиксированный предел ϵ).

Отметим, что пересчёт в этом алгоритме использует информацию от всех состояний–предшественников. Но можно использовать и другой, «стохастический» вариант:

$$Q(s, a) := Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a)).$$

Доказано, что он работает, если каждая пара (s, a) встречается бесконечное число раз, s' выбирают из распределения $T(s, a, s')$, а r сэмплируют со средним $R(s, a)$ и ограниченной вариацией.

Алгоритм итерации по стратегиям (рис. 5.5) использует точно такую же идею, но итерации идут не по ожидаемым значениям, а по собственно стратегиям π, которым может следовать агент. Для него каждая итерация стоит дороже, чем для алгоритма итерации по значениям, так как нужно на каждом шаге решать системы линейных уравнений. Зато для итераций по стратегиям очевидна сходимость, т.к. на каждом шаге целевая функция строго улучшается, а всего существует конечное число ($|A|^{|S|}$) стратегий. Правда, это рассуждение даёт очень уж большую оценку на время работы алгоритма; на практике он работает куда быстрее, но теоретически не известно, полиномиален ли он.