4. ЛЕКЦИЯ. Классическая теория 2

Динамическое программирование

Некоторые алгоритмы разбивают задачу на более мелкие подзадачи и используют решения подзадач, чтобы собрать решение для главной. Во время этого процесса количество подзадач может стать очень большим, и некоторые алгоритмы решают одну и ту же подзадачу многократно, что чрезмерно увеличивает время выполнения. Динамическое программирование упорядочивает вычисления и позволяет не вычислять уже известные значения повторно. Зачастую это экономит массу времени.

Метод простой итерации

Видно, что знание оценочных функций открывает путь к улучшению стратегии. Напрямую по определению считать их затруднительно; надо пробовать научиться их решать через уравнения Беллмана. И хотя уравнения оптимальности Беллмана нелинейные, они, тем не менее, имеют весьма определённый вид. Нам понадобится несколько понятий внезапно из функционального анализа о том, как решать системы нелинейных уравнений вида $x = f(x)$.

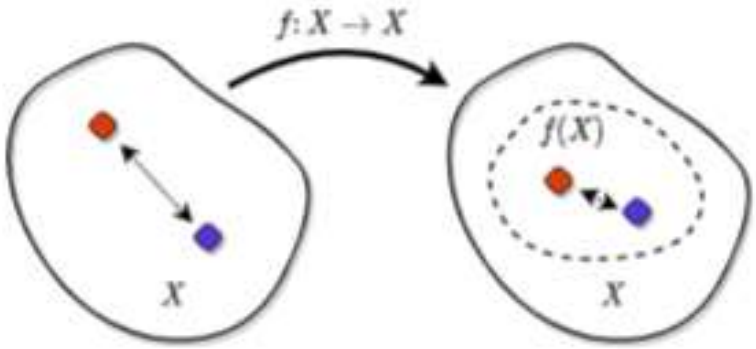

Рис.4.1

Определение: Оператор $f: X \to X$ называется сжимающим (contraction) с коэффициентом сжатия $\gamma < 1$ по некоторой метрике ρ, если $\forall x_1, x_2$:

$$\rho(f(x_1), f(x_2)) < \gamma\rho(x_1, x_2)$$

Определение: Точка $x \in X$ для оператора $f: X \to X$ называется неподвижной (fixed point), если

$$x = f(x)$$

Определение: Построение последовательности $x_{k+1} = f(x_k)$ для начального приближения $x_0 \in X$ называется методом простой итерации (point iteration) решения уравнения $x = f(x)$.

Теорема – Теорема Банаха о неподвижной точке: В полном метрическом пространстве X усжимающего оператора $f: X \to X$ существует и обязательно

единственна неподвижная точка x^*, причём метод простой итерации сходится к ней из любого начального приближения

Сходимость метода простой итерации. Пусть x_0 – произвольное, $x_{k+1} = f(x_k)$. Тогда для любого $k > 0$:

$$\begin{aligned}\rho(x_k, x_{k+1}) &= \{\text{определение } x_k\} = \rho(f(x_{k-1}), f(x_k)) \leq \\ &\leq \{\text{свойство сжатия}\} \leq \gamma\rho(x_{k-1}, x_k) \leq \cdots \leq \\ &\leq \{\text{аналогичным образом}\} \leq \cdots \leq \gamma^k \rho(x_0, x_1)\end{aligned} \tag{4.1}$$

Теперь посмотрим, что произойдёт после применения оператора f n раз:

$$\rho(x_k, x_{k+1}) \leq$$

{неравенство треугольника}

$$\leq \rho(x_k, x_{k+1}) + \rho(x_{k+1}, x_{k+2}) + \cdots + \rho(x_{k+n-1}, x_{k+n}) \leq$$

$$\leq \{(4.1)\} \leq (\gamma^k + \gamma^{k+1} + \ldots \gamma^{k+n-1})\rho(x_0, x_1) \leq$$

$$\leq \{\text{геом. прогрессия}\} \leq \frac{\gamma^k}{1-\gamma}\rho(x_0, x_1) \xrightarrow{\text{k}\to\infty} 0$$

Итак, последовательность x_k – фундаментальная, и мы специально используем такое метрическое пространство («полное»), в котором обязательно найдётся предел $x^* := \lim\limits_{k\to\infty} x_k$.

Существование неподвижной точки. Покажем, что x^* и есть неподвижная точка f, то есть покажем, что метод простой итерации конструктивно её построил. Заметим, что для любого $k > 0$:

$$\rho(x^*, f(x^*)) \leq$$

$$\leq \{\text{неравенство треугольника}\} \leq \rho(x^*, x_k) + \rho(x_k, f(x^*)) =$$

$$= \{\text{определение } x_k\} = \rho(x^*, x_k) + \rho(x_{k-1}, f(x^*)), f(x^*)) \leq$$

$$\leq \{\text{свойство сжатия}\} \leq \rho(x^*, x_k) + \gamma\rho(x_{k-1}, x^*)$$

Устремим $k \to \infty$; слева стоит константа, не зависящая от k. Тогда расстояние между x_k и x^* устремится к нулю, ровно как и между x_{k-1}, x^*поскольку x^* — предел x_k. Значит, константа равна нулю, $\rho(x^*, f(x^*)) = 0$, следовательно, $x^* = f(x^*)$.

Единственность. Пусть x_1, x_2 — две неподвижные точки оператора f. Ну тогда:

$$\rho(x_1, x_2) = \rho(f(x_1), f(x_2)) \leq \gamma\rho(x_1, x_2)$$

Получаем, что такое возможно только при $\rho(x_1, x_2) = 0$, то есть только если x_1 и x_2 совпадают

Policy Evaluation (Оценка политики)

Известно, что V^π для данного MDP и фиксированной политики π удовлетворяет уравнению Беллмана. Для нас это система уравнений

относительно значений $V^{\pi}(s)$. $V^{\pi}(s)$ — объект (точка) в функциональном пространстве $\mathcal{S} \to \mathbb{R}$.

Будем решать её методом простой итерации. Для этого определим оператор $\mathfrak{B}$, то есть преобразование из одной функции $\mathcal{S} \to \mathbb{R}$ в другую. На вход этот оператор принимает функцию $V: \mathcal{S} \subseteq \mathbb{R}^n$ и выдаёт некоторую другую функцию от состояний $\mathfrak{B}V$. Чтобы задать выход оператора, нужно задать значение выходной функции в каждом $s \in \mathcal{S}$; это значение мы будем обозначать $[\mathfrak{B}V](s)$ (квадратные скобки позволяют не путать применение оператора с вызовом самой функции) и определим его как правую часть решаемого уравнения Беллмана. И так:

Определение: Введём оператор Беллмана для заданного MDP и стратегии π как

$$[\mathfrak{B}V](s) := \mathbb{E}_{a\sim\pi(a|s)}[r(s,a) + \gamma \mathbb{E}_{s'} V(s')]$$

Также нам нужна метрика на множестве функций $\mathcal{S} \to \mathbb{R}$; возьмём

$$d_{\infty}(V_1, V_2) := \max_s |V_1(s) - V_2(s)|$$

Теорема: Если $\gamma < 1$, оператор $\mathfrak{B}$ — сжимающий с коэффициентом сжатия γ.

Итак, мы попали в теорему Банаха, и значит, метод простой итерации

$$V_{k+1} := \mathfrak{B}V_k$$

гарантированно сойдётся к единственной неподвижной точке при любой стартовой инициализации V_0. По построению знаем, что $V^{\pi}(s)$ такова, что $\mathfrak{B}V^{\pi} = V^{\pi}$ (это и есть уравнение Беллмана), поэтому к ней и придём.

Важно помнить, что на каждой итерации такой процедуры текущее приближение не совпадает с истинной оценочной функцией: $V_k(s) \approx V^{\pi}(s)$, но точного равенства поставить нельзя. Распространено (но, к сожалению, не везде применяется) соглашение обозначать аппроксимации оценочных функций без верхнего индекса: просто V или Q. Однако, иногда, что бы подчеркнуть, что алгоритм учит именно V^{π}, верхний индекс оставляют, что может приводить к путанице.

Обсудим, что случится в ситуации, когда $\gamma = 1$; напомним, что в таких ситуациях мы требовали эпизодичность сред, с гарантиями завершения всех эпизодов за $T^{\max}$ шагов. Оператор Беллмана формально сжатием являться уже не будет, и мы не подпадаем под теорему.

Теорема: В эпизодичных средах метод простой итерации сойдётся к единственному решению уравнений Беллмана не более чем за $T^{\max}$ шагов даже при $\gamma = 1$.

Мы определились, как решать задачу оценивания стратегии (Policy Evaluation): вычислять значения оценочной функции по данной стратегии π в ситуации, когда мы знаем динамику среды. На практике мы можем воспользоваться этим результатом только в «табличном» случае (tabular RL), когда пространство состояний и пространство действий конечны и достаточно малы, чтобы все пары состояние-действие было возможно хранить в памяти компьютера и перебирать за разумное время. В такой ситуации $V^{\pi}(s)$ — конечный вектор, и можно считать оператор Беллмана и делать обновления $V_{k+1} = \mathfrak{B}V_k$.

Алгоритм: Policy Evaluation (Оценка политики)

Вход: $\pi(a \mid s)$ — стратегия
Гиперпараметры: ε — критерий останова

Инициализируем $V_0(s)$ произвольно для всех $s \in \mathcal{S}$
На k-ом шаге:

1. $\forall s$: $V_{k+1}(s) \coloneqq \mathbb{E}_a\left[r(s,a) + \gamma \mathbb{E}_{s'} V_k(s')\right]$
2. **критерий останова:** $\max_s |V_k(s) - V_{k+1}(s)| < \varepsilon$

Выход: $V_k(s)$

Value Iteration (Итерация значения)

Теорема Банаха позволяет аналогично Policy Evaluation (Оценка политики) решать уравнения оптимальности Беллмана через метод простой итерации. Действительно, проведём аналогичные рассуждения (мы сделаем это для Q^*, но совершенно аналогично можно было бы сделать это и для V^*):

Определение: Определим оператор оптимальности Беллмана (Bellman optimality operator, Bellman control operator) $\mathfrak{B}^*$:

$$[\mathfrak{B}^* Q](s,a) := r(s,a) + \gamma \mathbb{E}_{a'} \max_{a'} Q(s',a')$$

В качестве метрики на множестве функций $\mathcal{S} \times \mathcal{A} \to \mathbb{R}$ аналогично возьмём

$$d_\infty(Q_1, Q_2) := \max_{s,a} |Q_1(s,a) - Q_2(s,a)|$$

Тогда утвердим следующий факт:

Утверждение: $\left|\max_x f(x) - \max_x g(x)\right| \le \max_x |f(x) - g(x)|$

Аналогично.

Теорема: Если $\gamma < 1$, оператор $\mathfrak{B}^*$: — сжимающий.

Теорема: В эпизодичных средах метод простой итерации сойдётся к

единственному решению уравнений оптимальности Беллмана не более чем за $T^{\max}$ шагов даже при $\gamma = 1$.

Аналогично.

Утверждение: Метод простой итерации сходится к Q^* из любого начального приближения

Вообще, если известна динамика среды, то нам достаточно решить уравнения оптимальности для V^* — это потребует меньше переменных. Итак, в табличном случае мы можем напрямую методом простой итерации решать уравнения оптимальности Беллмана и в пределе сойдёмся к оптимальной оценочной функции, которая тут же даёт нам оптимальную стратегию.

Алгоритм: Value Iteration (Итерация значения)

Вход: ε — критерий останова

Инициализируем $V_0(s)$ произвольно для всех $s \in \mathcal{S}$

На k-ом шаге:

1. для всех s: $V_{k+1}(s) \coloneqq \max_a [r(s,a) + \gamma \mathbb{E}_{s'} V_k(s')]$
2. **критерий останова:** $\max_s |V_{k+1}(s) - V_k(s)| < \varepsilon$

Выход: $\pi(s) \coloneqq \operatorname{argmax}_a [r(s,a) + \gamma \mathbb{E}_{s'} V(s')]$

Это табличный алгоритм планирования — алгоритм, решающий задачу RL в условиях известной модели среды. На каждом шаге мы обновляем («бэкапим») нашу текущую аппроксимацию V-функции на её одношаговое приближение (one-step approximation): смотрим на один шаг в будущее (a, r, s') и приближаем всё остальное будущее текущей же аппроксимацией.

Рис.4.2

Такой «бэкап динамического программирования» (dynamic programming backup, DP-backup) — обновление «бесконечной ширины»: мы должны

перебрать все возможные варианты следующего одного шага, рассмотреть все свои действия (по ним мы возьмём максимум) и перебрать всевозможные ответы среды — s' (по ним мы должны рассчитать математическое ожидание). Поэтому этот алгоритм в чистом виде напоминает то, что обычно и понимается под словами «динамическое программирование»: мы «раскрываем дерево игры» полностью на один шаг вперёд.

Policy Iteration (Итерация политики)

Обобщим Value Iteration (итерацию значения) и придумаем более общую схему алгоритма планирования для табличного случая.

Для очередной стратегии π_{k} посчитаем её оценочную функцию $Q^{\pi_{\mathrm{k}}}$, а затем воспользуемся теоремой Policy Improvement (улучшение политики) и построим стратегию лучше; например, жадно:

$$\pi_{k+1}(s) := arg\max_a Q^{\pi_{\mathrm{k}}}(s,a)$$

Тогда у нас есть второй алгоритм планирования, который, причём, перебирает детерминированные стратегии, обладающие свойством монотонного возрастания качества: каждая следующая стратегия не хуже предыдущей. Он работает сразу в классе детерминированных стратегий, и состоит из двух этапов:

• Policy Evaluation (оценка политики): вычисление Q^{π} для текущей стратегии π;

• Policy Improvement (улучшение политики): улучшение стратегии $\pi(s) \leftarrow arg\max_a Q^{\pi}(s,a)$;

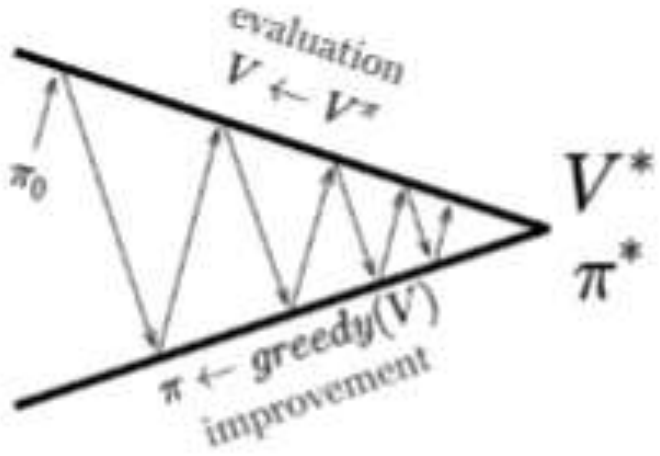

Рис.4.3

При этом у нас есть гарантии, что когда алгоритм «останавливается» (не может провести Policy Improvement), то он находит оптимальную стратегию. Будем считать, что в такой момент остановки после проведения Policy Improvement наша стратегия не меняется: $\pi_{k+1} \equiv \pi_k$.

Теорема: В табличном сеттинге Policy Iteration завершает работу за

конечное число итераций.

Алгоритм: Policy Iteration (Итерация политики)

Гиперпараметры: ε — критерий останова для процедуры **PolicyEvaluation**

Инициализируем $\pi_0(s)$ произвольно для всех $s \in \mathcal{S}$
На k-ом шаге:

1. $V^{\pi_k} \coloneqq \textbf{PolicyEvaluation}(\pi_k, \varepsilon)$
2. $Q^{\pi_k}(s,a) \coloneqq r(s,a) + \gamma \mathbb{E}_{s'} V^{\pi_k}(s')$
3. $\pi_{k+1}(s) \coloneqq \underset{a}{\operatorname{argmax}}\, Q^{\pi_k}(s,a)$
4. **критерий останова:** $\pi_k \equiv \pi_{k+1}$

Generalized Policy Iteration (Обобщенная итерация политики)

Policy Iteration — идеализированный алгоритм: на этапе оценивания в табличном сеттинге можно попробовать решить систему уравнений Беллмана V^{π} с достаточно высокой точностью за счёт линейности этой системы уравнений, или можно считать,что проводится достаточно большое количество итераций метода простой итерации. Тогда, вообще говоря, процедура предполагает бесконечное число шагов, и на практике нам нужно когда-то остановиться; теоретически мы считаем, что доводим вычисления до некоторого критерия останова, когда значения вектора не меняются более чем на некоторую погрешность $\varepsilon > 0$.

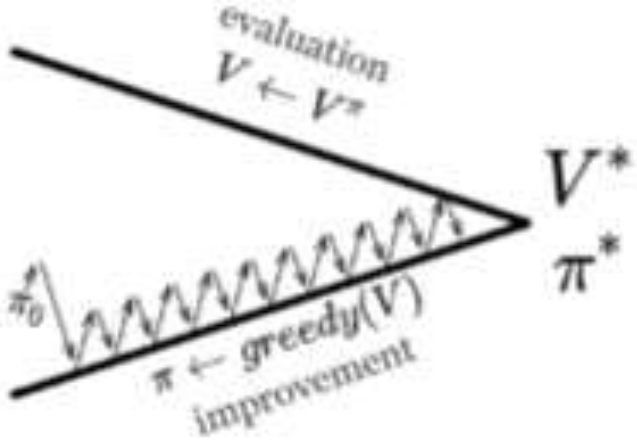

Рис.4.4

Но рассмотрим такую, пока что, эвристику: давайте останавливать Policy Evaluation после ровно N шагов, а после обновления стратегии не начинать оценивать π_{k+1} с нуля, а использовать последнее $V(s) \approx V^{\pi_k}$ в качестве инициализации. Тогда алгоритм примет следующий вид:

Алгоритм: Generalized Policy Iteration (Обобщенная итерация политики)

Гиперпараметры: N — количество шагов

Инициализируем $\pi(s)$ произвольно для всех $s \in \mathcal{S}$
Инициализируем $V(s)$ произвольно для всех $s \in \mathcal{S}$
На k-ом шаге:

1. **Повторить N раз:**
   - $\forall s: V(s) \leftarrow \mathbb{E}_a\left[r(s,a) + \gamma \mathbb{E}_{s'} V(s')\right]$
2. $Q(s,a) \leftarrow r(s,a) + \gamma \mathbb{E}_{s'} V(s')$
3. $\pi(s) \leftarrow \underset{a}{\operatorname{argmax}}\, Q(s,a)$

Мы формально теряем гарантии улучшения стратегии на этапе Policy Improvement, поэтому останавливать алгоритм после того, как стратегия не изменилась, уже нельзя: возможно, после следующих N шагов обновления оценочной функции, аргумента максимума поменяется, и стратегия всё-таки сменится. Но такая схема в некотором смысле является наиболее общей, и вот почему:

Утверждение: Generalized Policy Iteration (обобщенная итерация политики) совпадает с Value Iteration (итерация значения) при $N = 1$ и с Policy Iteration (итерация политики) при $N = \infty$.

Итак, Generalized Policy Iteration при $N = 1$ и при $N = \infty$ —это ранее разобранные алгоритмы, физический смысл которых нам ясен. В частности, теперь понятно, что в Value Iteration очередное приближение $Q_k(s,a) \approx Q^*(s,a)$ можно также рассматривать как приближение $Q_k(s,a) \approx Q^{\pi}(s,a)$ для $(s) := arg\max_a Q_k(s,a)$; то есть в алгоритме хоть и не потребовалось в явном виде хранить «текущую» стратегию, она всё равно неявно в нём присутствует. Попробуем понять, что происходит в Generalized Policy Iteration при промежуточных N. Заметим, что повторение N раз шага метода простой итерации для решения уравнения $\mathfrak{B}V^{\pi} = V^{\pi}$ эквивалентно одной итерации метода простой итерации для решения уравнения $\mathfrak{B}^{\mathrm{N}}V^{\pi} = V^{\pi}$ (где запись $\mathfrak{B}^{\mathrm{N}}$ означает повторное применение оператора $\mathfrak{B}$ N раз), для которого, очевидно, искомая V^{π} также будет неподвижной точкой. Что это за оператор $\mathfrak{B}^{\mathrm{N}}$? В уравнениях Беллмана мы «раскручивали» наше будущее на один шаг вперёд и дальше заменяли оставшийся «хвост» на определение V-функции. Понятно, что мы могли бы раскрутить не на один шаг, а на N шагов вперёд.

Теорема — N-шаговое уравнение Беллмана: Для любого состояния s0:

$$V^{\pi}(\mathrm{s}_0) = \mathbb{E}_{\mathcal{T}:\mathrm{N}\sim\pi|\mathrm{s}0}\left[\sum_{\mathrm{t}=0}^{\mathrm{N}-1}\gamma^t r_t + \gamma^{\mathrm{N}}\mathbb{E}_{s_N}V^{\pi}(s_N)\right]$$

Тогда

Утверждение: $\mathfrak{B}^{\mathrm{N}}$ — оператор с коэффициентом сжатия γ^{N}.

Означает ли это, что метод простой итерации решения N-шаговых уравнений сойдётся быстрее? Мы по сути просто «за один шаг» делаем N итераций метода простой итерации для решения обычного одношагового уравнения; в этом смысле, мы ничего не выигрываем. В частности, если мы устремим N к бесконечности, то мы получим просто определение V-функции; формально, в правой части будет стоять выражение, вообще не зависящее от поданной на вход оператору $V(s)$, коэффициент сжатия будет ноль, и метод простой итерации как бы сходится тут же за один шаг. Но для проведения этого шага нужно выинтегрировать все траектории — «раскрыть дерево полностью».

Но теперь у нас есть другой взгляд на Generalized Policy Iteration: мы чередуем одну итерацию решения N-шагового уравнения Беллмана с алгоритмом Policy Improvement (улучшение политики).

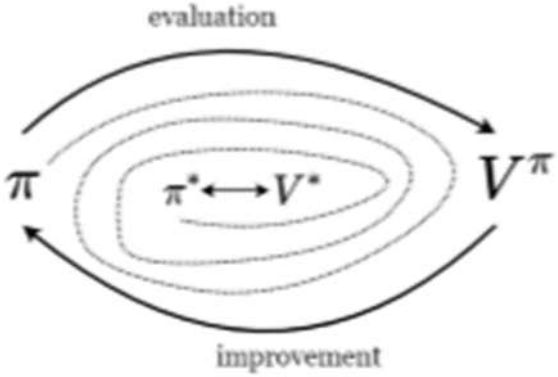

Рис.4.5

Теорема: Алгоритм Generalized Policy Iteration при любом N сходится к оптимальной стратегии и оптимальной оценочной функции.

Интуитивно, такой алгоритм «стабилизируется», если оценочная функция будет удовлетворять уравнению Беллмана для текущей π (иначе оператор $\mathfrak{B}^{\mathrm{N}}$ изменит значение функции), и если π выбирает $\underset{a}{\mathrm{argmax}}\,\mathrm{Q}(\mathrm{s},\mathrm{a})$ из неё; а если аппроксимация V-функции удовлетворяет уравнению Беллмана, то она совпадает с V^{π} , и значит $Q(s,a) = Q^{\pi}(s,a)$. То есть, при сходимости стратегия π будет выбирать действие жадно по отношению к своей же Q^{π}, а мы помним, что это в точности критерий оптимальности. Все алгоритмы, которые мы будем обсуждать далее, так или иначе подпадают под обобщённую парадигму «оценивание-улучшение». У нас будет два процесса оптимизации: обучение

актёра (actor), политики π, и критика (critic), оценочной функции (Q или V). Критик обучается оценивать текущую стратегию, текущего актёра: сдвигаться в сторону решения какого-нибудь уравнения, для которого единственной неподвижной точкой является V^{π} или Q^{π}. Актёр же будет учиться при помощи policy improvement-а: вовсе не обязательно делать это жадно, возможно учиться выбирать те действия, где оценка критика «побольше», оптимизируя в каких-то состояниях (в каких - пока открытый вопрос) функционал:

$$\mathbf{E}_{\pi(a|s)}Q(s,a) \rightarrow \max_{\pi}$$

Причём, возможно, в этом функционале нам не понадобится аппроксимация (модель) Q-функции в явном виде, и тогда мы можем обойтись лишь какими-то оценками Q^{π}; в таких ситуациях нам достаточно будет на этапе оценивания политики обучать лишь модель V^{π} для текущей стратегии. А, например, в эволюционных методах мы обошлись вообще без обучения критика именно потому, что смогли обойтись лишь Монте-Карло оценками будущих наград. Этот самый простой способ решать задачу RL — погенерировать несколько случайных стратегий и выбрать среди них лучшую — тоже условно подпадает под эту парадигму: мы считаем Монте-Карло оценки значения $J(\pi)$ для нескольких разных стратегий (evaluation) и выбираем наилучшую стратегию (improvement). Поэтому Policy Improvement, тоже может выступать в разных формах: например, возможно, как в Value Iteration, у нас будет приближение Q-функции, и мы будем просто всегда полагать, что policy improvement проводится жадно, и текущей стратегией неявно будет $\operatorname{argmax}_a \mathrm{Q}(\mathrm{s},\mathrm{a})$. Но главное, что эти два процесса, оценивание политики (обучение критика) и улучшение (обучение актёра) можно будет проводить стохастической оптимизацией. Достаточно, чтобы лишь в среднем модель оценочной функции сдвигалась в сторону V^{π} или Q^{π}, а актёр лишь в среднем двигался в сторону $\operatorname{argmax}_a \mathrm{Q}(\mathrm{s},\mathrm{a})$. И такой «рецепт» алгоритма всегда будет работать: пока оба этих процесса проводятся корректно.

Табличныеалгоритмы

Монте-Карло алгоритм

Value iteration (итерация значения) и Policy iteration (итерация политики) имели два ограничения: 1) необходимо уметь хранить табличку размером $|\mathcal{S}|$ в памяти и перебирать все состояния и действия за разумное время 2) должна быть известна динамика среды $p(s'|s,a)$. Первое устраняется нейронками, а сейчас будем устранять второе: в сложных средах проблема даже не столько в том,

чтобы приблизить динамику среды, а в том, что интегралы $\mathbb{E}_{s'\sim p}(s'|s,a)$ мы не возьмём в силу огромного числа состояний и сможем только оценивать по Монте-Карло. Итак, мы хотим придумать табличный model-free (без модели) RL-алгоритм: мы можем отправить в среду пособирать траектории какую-то стратегию, и дальше должны проводить итерации алгоритма, используя лишь эти сэмплы траекторий. Иначе говоря, для данного s мы можем выбрать a и получить ровно один сэмпл из очередного $p(s'|s,a)$, причём на следующем шаге нам придётся проводить сбор сэмплов именно из s'. Как в таких условиях «решать» уравнения Беллмана — неясно. Рассмотрим самый простой способ превратить Policy Iteration в model-free метод. Давайте очередную стратегию π_k отправим в среду, сыграем несколько эпизодов, и будем оценивать Q^{π_k} по Монте-Карло:

$$Q^{\pi_\mathrm{k}}(\mathrm{s},\mathrm{a}) \approx \frac{1}{N}\sum_{i=0}^{N} R(\mathcal{T}_i) \quad \mathcal{T}_i \sim \pi_\mathrm{k}|\mathrm{s}_0 = \mathrm{s}, \mathrm{a}_0 = \mathrm{a}$$

Теперь, доиграв эпизод до конца, мы для каждой встретившейся пары s, a в полученной траектории можем посчитать reward-to-go (награда на вынос) и использовать этот сэмпл для обновления аппроксимации Q-функции — проведения Монте-Карло бэкапа (MC-backup резервная копия). Такое обновление полностью противоположно по свойствам бэкапу динамического программирования: это «бэкап ширины один» бесконечной длины — мы использовали лишь один сэмпл будущего и при этом заглянули в него на бесконечное число шагов вперёд.

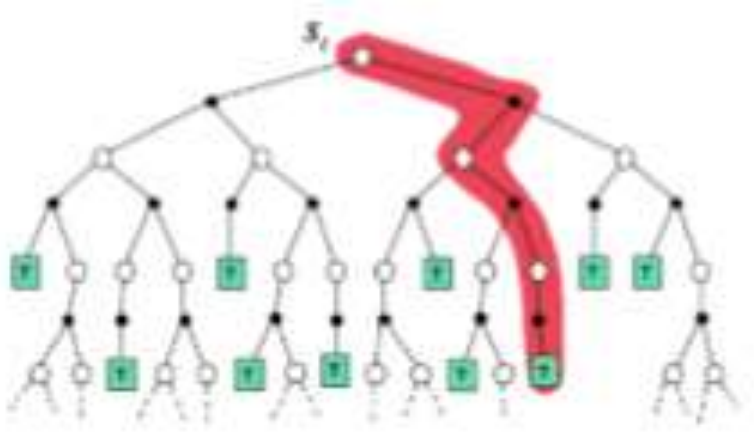

Рис.4.6

Монте-Карло алгоритм, на первый взгляд, плох примерно всем. Нужно доигрывать игры до конца, то есть алгоритм неприменим в неэпизодичных средах; Монте-Карло оценки также обладают огромной дисперсией, поскольку мы заменили на сэмплы все мат.ожидания, стоящие в мат.ожидании по

траекториям; наконец, мы практически перестали использовать структуру задачи. Если мы получили сэмпл +100 в качестве очередного сэмпла для $Q^{\pi}(s,a)$, то мы «забыли», какую часть из этой +100 мы получили сразу же после действия a, а какая была получена в далёком будущем — не использовали разложение награды за эпизод в сумму наград за шаг. Также мы посеяли «информацию о соединениях состояниях»: пусть у нас было две траектории, имевших пересечение в общем состоянии. Тогда для начал этих траекторий мы всё равно считаем, что собрали лишь один сэмпл reward-to-go, хотя в силу марковости у нас есть намного больше информации.

Ещё одна проблема алгоритма: если для некоторых $s,a: \pi(a|s) = 0$, то мы ничего не узнали об $Q^{\pi}(s,a)$. А, как было видно в алгоритмах динамического программирования, мы существенно опираемся в том числе и на значения Q-функции для тех действий, которые π никогда не выбирает; только за счёт этого мы умеем проводить policy improvement (политику улучшения) для детерминированных стратегий.

Экспоненциальное сглаживание

Рассмотрим такую задачу. Нам приходят сэмплы $x_1, x_2 \dots x_n \sim p(x)$. Хотим по ходу получения сэмплов оценивать мат.ожидание случайной величины x. Давайте хранить Монте-Карло оценку, усредняя все имеющиеся сэмплы; для этого достаточно пользоваться следующим рекурсивным соотношением:

$$m_k := \frac{1}{k}\sum_{i=1}^{k} x_i = \frac{k-1}{k} m_{k-1} + \frac{1}{k} x_k$$

Обозначим за $\alpha_k := \frac{1}{k}$. Тогда формулу можно переписать так:

$$m_k := (1-\alpha_k) m_{k-1} + \alpha_k x_k$$

Определение: Экспоненциальным сглаживанием (exponential smoothing) для последовательности x_1, x_2, x_3 ... будем называть следующую оценку:

$$m_k := (1-\alpha_k) m_{k-1} + \alpha_k x_k$$

где m_0 – некоторое начальное приближение, последовательность $\alpha_k \in [0,1]$ — гиперпараметр, называемый скоростью обучения.

Определение: Будем говорить, что learning rate (скоростью обучения) $\alpha_k \in [0,1]$ удовлетворяет условиям Роббинса-Монро (Robbins-Monro conditions), если:

$$\sum_{k\geq 0}^{\infty} \alpha_k = +\infty, \sum_{k\geq 0}^{\infty} \alpha_k^2 < +\infty$$

Теорема: Пусть $x_1, x_2 \dots$ — независимые случайные величины, $\mathbb{E}x_k = m$, $\mathbb{D}x_k \leq C < +\infty$, где C — некоторая конечная константа. Пусть m_0 —

произвольно, а последовательность чисел $\alpha_k \in [0,1]$ удовлетворяет условиям Роббинса-Монро. Тогда экспоненциальное сглаживание

$$m_k := (1-\alpha_k)m_{k-1} + \alpha_k x_k$$

сходится к m с вероятностью 1.

Стохастическая аппроксимация

Можно считать, что мы научились решать уравнения такого вида:

$$x = \mathbb{E}_\varepsilon f(\varepsilon),$$

где справа стоит мат.ожидание по неизвестному распределению $\varepsilon \sim p(\varepsilon)$ от функции f, которую для данного значения сэмпла ε мы умеем считать. Это просто стандартная задача оценки среднего, для которой мы даже доказали теоретические гарантии сходимости следующего итеративного алгоритма:

$$x_k := (1-\alpha_k)x_{k-1} + \alpha_k f(\varepsilon), \varepsilon \sim p(\varepsilon)$$

Аналогично у нас есть итеративный алгоритм для решения систем нелинейных уравнений

$$x = f(x)$$

где справа стоит, если угодно, «хорошая» функция — сжатие. Формула обновления в методе простой итерации выглядела вот так:

$$x_k := f(x_{k-1})$$

Определение: Стохастическая аппроксимация (Stochastic approximation) — задача решения уравнения вида

$$x = \mathbb{E}_{\varepsilon \sim p(\varepsilon)} f(x, \varepsilon),$$

где справа стоит мат.ожидание по неизвестному распределению $p(\varepsilon)$, из которого доступны лишь сэмплы, от функции, которую при данном сэмпле ε и некотором значении неизвестной переменной x мы умеем считать.

Можно ли объединить идеи метода простой итерации и экспоненциального сглаживания? Давайте запустим аналогичный итеративный алгоритм: на k-ой итерации подставим текущее приближение неизвестной переменной x_k в правую часть $f(x_k, \varepsilon)$ для $\varepsilon \sim p(\varepsilon)$, но не будем «жёстко» заменять x_{k+1} на полученное значение, так как оно является лишь несмещённой оценкой правой части; вместо этого сгладим старое значение x_k и полученный новый «сэмпл»:

$$x_k = (1-\alpha_k)x_{k-1} + \alpha_k f(x_{k-1}, \varepsilon), \qquad \varepsilon \sim p(\varepsilon)$$

Есть хорошая надежда, что, если функция f «хорошая», распределение $p(\varepsilon)$ не сильно страшное (например, имеет конечную дисперсию, как в теореме о сходимости экспоненциального сглаживания), а скорость обучения α_k удовлетворяют условиям Роббинса-Монро, то такая процедура будет в пределе сходиться.

Поймём, почему задача стохастической аппроксимации тесно связана со

стохастической оптимизацией. Для этого перепишем формулу в альтернативной форме:

$$x_k = x_{k-1} + \alpha_k(f(x_{k-1}, \varepsilon) - x_{k-1}), \qquad \varepsilon \sim p(\varepsilon)$$

Видно, что это формула стохастического градиентного спуска. Действительно, оптимизируя некоторую функцию $f(x)$, прибавляем к очередному приближению $x_{\mathrm{k}-1}$ с некоторым скоростью обучения α_k несмещённую оценку градиента $\nabla f(x_{k-1})$, которую можно обозначить как $\nabla f(x_{k-1}, \varepsilon)$:

$$x_k = x_{k-1} + \alpha_k \nabla f(x_{k-1}, \varepsilon), \qquad \varepsilon \sim p(\varepsilon)$$

О стохастической оптимизации можно думать не как об оптимизации, а как о поиске решения уравнения $\nabla f(x) = 0$. Действительно, как и любая локальная оптимизация, может идти как к локальному оптимуму, так и к седловым точкам, но в них $\mathbb{E}_\varepsilon \nabla f(x, \varepsilon) = 0$.

Тогда можно понять, что, видимо, выражение $f(x_{k-1}, \varepsilon) - x_{k-1}$ есть «стохастический градиент». Стохастический он потому, что это выражение случайно: мы сэмплируем $\varepsilon \sim p(\varepsilon)$; при этом это несмещённая оценка «градиента», поскольку она обладает следующим свойством: в точке решения, то есть в точке x^*, являющейся решением уравнения $x = \mathbb{E}_{\varepsilon \sim p(\varepsilon)} f(x, \varepsilon)$, в среднем его значение равно нулю:

$$\mathbb{E}_\varepsilon \left(f(x^*, \varepsilon) - x^*\right) = 0$$

И по аналогии можно предположить, что если стохастический градиентный спуск ищет решение $\mathbb{E}_\varepsilon \nabla f(x, \varepsilon) = 0$, то процесс ищет решение уравнения $\mathbb{E}_\varepsilon \nabla f(x, \varepsilon) - x = 0$. Это наблюдение будет иметь для нас ключевое значение. В алгоритме без модели (model-free) режиме как при оценивании стратегии, когда мы решаем уравнение Беллмана

$$Q^\pi(s, a) = \mathbb{E}_{s'}[r(s, a) + \gamma \mathbb{E}_{a'} Q^\pi(s', a')]$$

так и когда мы пытаемся реализовать Value Iteration и напрямую решать уравнения оптимальности Беллмана

$$Q^*(s, a) = \mathbb{E}_{s'}[r(s, a) + \gamma \max_{a'} Q^*(s', a')]$$

мы сталкиваемся в точности с задачей стохастической аппроксимации. Справа стоит мат.ожидание по недоступному нам распределению, содержимое которого мы, тем не менее, при данном сэмпле $s' \sim p(s'|s, a)$ и текущем приближении Q-функции, умеем считать. Возникает идея проводить стохастическую аппроксимацию: заменять правые части решаемых уравнений на несмещённые оценки и сглаживать текущее приближение с получаемыми сэмплами. Отметим интересный факт: уравнение V^*V^* (уравнения оптимальности Беллмана),

имеющее вид «максимум от мат.ожидания по неизвестному распределению»

Temporal Difference (Временная разница)

Попробуем применить идею стохастической аппроксимации для решения уравнений Беллмана. На очередном шаге после совершения действия a из состояния s мы получаем значение награды $r := r(s,a)$, сэмпл следующего состояния s' и генерируем $a' \sim \pi$, после чего сдвигаем с некоторой скоростью обучения (learning rate) наше текущее приближение $Q(s,a)$ в сторону сэмпла

$$y := r + \gamma Q(s',a'),$$

который также будем называть таргетом (Bellman target, подразумевается «догадка» какое будущее нас ждёт). Получаем следующую формулу обновления:

$$Q_{k+1}(s,a) \leftarrow Q_k(s,a) + \alpha_k \underbrace{\left(\overbrace{r + \gamma Q_k(s',a')}^{\text{target}} - Q_k(s,a) \right)}_{\text{временная разность}}$$

Выражение $r(s,a) + \gamma Q_k(s',a') - Q_k(s,a)$ называется временной разностью (temporal difference): это отличие сэмпла, который нам пришёл, от текущей оценки среднего.

Таким образом, TD-backup (временная разница – резервное копирование): обновление, имеющее как ширину, так и длину один. Рассматриваем лишь одну версию будущего (один сэмпл) и заглядываем на один шаг вперёд, приближая всё дальнейшее будущее своей собственной текущей аппроксимацией. Этот ход позволяет учиться, не доигрывая эпизоды до конца: мы можем обновить одну ячейку нашей Q-функции сразу же после одного шага в среде, после сбора одного перехода (s,a,r,s',a').

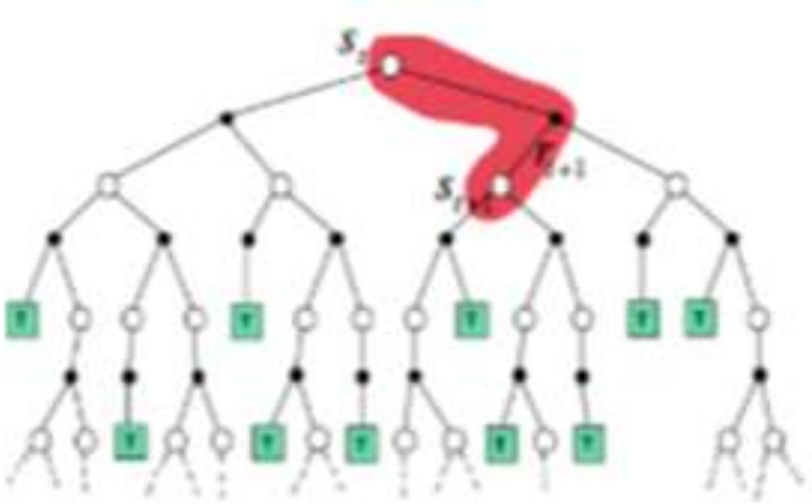

Рис.4.7

Рассматриваемая формула отличается от Монте-Карло обновлений Q-

функции лишь способом построения таргета: в Монте-Карло мы бы взяли в качестве у reward-to-go (награда на вынос), когда здесь же используем одношаговое приближение. Поэтому смотреть на эту формулу также можно через интуицию бутстрапирования (bootstrapping – таргеты построены на принципе такого бутстрапирования, поскольку мы начинаем «из воздуха» делать себе сэмплы из уже имеющихся сэмплов.). Мы хотим получить сэмплы для оценки нашей текущей стратегии π, поэтому делаем один шаг в среде и приближаем всю оставшуюся награду нашей текущей аппроксимацией среднего. Такой «псевдосэмпл» уже не будет являться корректным сэмплом для $Q^{\pi}(s,a)$, но в некотором смысле является «более хорошим», чем то, что у нас есть сейчас, за счёт раскрытия дерева на один шаг и получения информации об r, s'. Такое движение нашей аппроксимации в сторону чего-то хоть чуть-чуть более хорошего и позволяет нам чему-то учиться.

Обсудим следующий важный технический момент. Какие есть ограничения на переходы (s,a,r,s',a'), которые мы можем использовать для обновлений по формуле разности? Пока мы говорили, что мы хотим заниматься оцениванием стратегии π, и поэтому предлагается, видимо, ею и взаимодействовать со средой, собирая переходы. С точки же зрения стохастической аппроксимации, для корректности обновлений достаточно выполнения всего лишь двух требований:

- $s' \sim p(s'|s,a)$; если s' приходит из какой-то другой функции переходов, то мы учим Q-функцию для какого-то другого MDP.
- $a' \sim \pi(a'|s')$; если a' приходит из какой-то другой стратегии, то мы учим Q-функцию для вот этой другой стратегии.

Оба этих утверждения вытекают из того, что обновление временной разницы неявно ищет решение уравнения

$$Q(s,a) = \mathbb{E}_{s'}\mathbb{E}_{a'}y$$

как схема стохастической аппроксимации.

Q-learning (Q-обучение)

Поскольку довести $Q(s,a)$ до точного значения $Q^{\pi}(s,a)$ с гарантиями мы всё равно не сможем, однажды в алгоритме нам всё равно придётся сделать policy improvement (политику улучшения). Что, если мы будем обновлять нашу стратегию $\pi_k(s) := arg\max_a Q_k(s,a)$ после каждого шага в среде и каждого обновления Q-функции? Наше приближение Policy Iteration (итерация политики) схемы, аналогично ситуации в динамическом программировании, превратится в приближение Value Iteration (итерация значения) схемы

$$y = r + \gamma Q_k(s', a') =$$
$$= r + \gamma Q_k(s', \pi_k(s')) =$$
$$= r + \gamma Q_k(s', arg\max_{a'} Q_k(s', a')) =$$
$$= r + \gamma \max_{a'} Q_k(s', a')$$

Мы получили в точности таргет для решения методом стохастической аппроксимации уравнения оптимальности Беллмана; поэтому для такого случая будем обозначать нашу Q-функцию как аппроксимацию Q^*, и тогда наше обновление принимает следующий вид: для перехода s, a, r, s' обновляется только одна ячейка нашего приближения Q-функции:

$$Q_{k+1}(s, a) := Q_k(s, a) + \alpha_k \left(\mathrm{r} + \gamma \max_{a'} Q_k(s', a') - Q_k(s, a)\right)$$

Теорема – Сходимость Q-learning: Пусть пространства состояний и действий конечны, $Q_0(s, a)$ — начальное приближение, на k-ой итерации $Q_k(s, a)$ для всех пар s, a строится по правилу

$$Q_{k+1}(s, a) := Q_k(s, a) + \alpha_k(s, a) \left(\mathrm{r}(s, a) + \gamma \max_{a'} Q_k(s'_k(s, a), a') - Q_k(s, a)\right)$$

где $s'_k(s, a) \sim \mathrm{p}(\mathrm{s}'|\mathrm{s}, \mathrm{a})$, а $\alpha_k(s, a) \in [0,1]$ — случайные величины, с вероятностью один удовлетворяющие для каждой пары s, a условиям Роббинса-Монро:

$$\sum_{\mathrm{k}\geq 0}^{\infty} \alpha_k(s, a) = +\infty \quad \sum_{\mathrm{k}\geq 0}^{\infty} \alpha_k(s, a)^2 < +\infty$$

Тогда Q_k сходится к Q^* с вероятностью один.

Заметим, что для выполнения условий сходимости необходимо для каждой пары s, a проводить бесконечное число обновлений $Q(s, a)$: в противном случае, в ряду $\sum_{\mathrm{k}\geq 1} \alpha_k(s, a)$ будет конечное число ненулевых членов, и мы не попадём под теорему. Значит, наш сбор опыта должен удовлетворять условию infinite visitation (бесконечное посещение) — все пары s, a должны гарантированно встречаться бесконечно много раз. По сути теорема говорит, что это требование является достаточным условием на процесс порождения переходов s, a, r, s':

Утверждение: Пусть сбор опыта проводится так, что пары s, a встречаются бесконечное число раз. Пусть $n(s, a)$ — счётчик количества выполнений действия a в состоянии s во время взаимодействия агента со средой. Тогда можно положить $\alpha_k(s, a) := \frac{1}{n(s,a)}$, чтобы гарантировать сходимость алгоритма к Q^*.