
Конспект по обучению с подкреплением

Издание дописывающееся и дорабатывающееся

11 октября 2020 г.



Дисклеймер

Work in Progress

Оглавление

1 Задача обучения с подкреплением	4
1.1 Модель взаимодействия агента со средой	4
1.1.1 Связь с оптимальным управлением	4
1.1.2 Марковская цепь	5
1.1.3 Среда	6
1.1.4 Действия	7
1.1.5 Траектории	7
1.1.6 Марковский процесс принятия решений (MDP)	8
1.1.7 Эпизодичность	9
1.1.8 Дисконтирование	10
1.2 Алгоритмы обучения с подкреплением	11
1.2.1 Условия задачи RL	11
1.2.2 On-policy vs Off-policy	12
1.2.3 Концепция model-free алгоритмов	13
1.2.4 Классификация RL-алгоритмов	14
1.2.5 Критерии оценки RL-алгоритмов	14
1.2.6 Сложности задачи RL	15
1.2.7 Бенчмарки	16
2 Мета-эвристики	18
2.1 Бэйзлайны	18
2.1.1 Задача безградиентной оптимизации	18
2.1.2 Случайный поиск	19
2.1.3 Hill Climbing	20
2.1.4 Имитация отжига	21
2.1.5 Эволюционные алгоритмы	22
2.1.6 Weight Agnostic Neural Networks (WANN)	23
2.1.7 Видовая специализация	24
2.1.8 Генетические алгоритмы	25
2.2 Эволюционные стратегии	27
2.2.1 Идея эволюционных стратегий	27
2.2.2 Оценка вероятности редкого события	27
2.2.3 Метод Кросс-Энтропии для стохастической оптимизации	29
2.2.4 Метод Кросс-Энтропии для обучения с подкреплением (CEM)	29
2.2.5 Натуральные эволюционные стратегии (NES)	30
2.2.6 OpenAI-ES	30
2.2.7 Адаптация матрицы ковариации (CMA-ES)	32
3 Классическая теория	34
3.1 Оценочные функции	34
3.1.1 Свойства траекторий	34
3.1.2 V-функция	36
3.1.3 Уравнения Беллмана	37
3.1.4 Оптимальная стратегия	38
3.1.5 Q-функция	38
3.1.6 Принцип оптимальности Беллмана	39
3.1.7 Отказ от однородности	40
3.1.8 Вид оптимальной стратегии (доказательство через отказ от однородности)	41
3.1.9 Уравнения оптимальности Беллмана	43
3.1.10 Критерий оптимальности Беллмана	43
3.2 Relative Performance Identity	45
3.2.1 Advantage-функция	45

3.2.2	Relative Performance Identity (RPI)	45
3.2.3	Policy Improvement	46
3.3	Динамическое программирование	48
3.3.1	Метод простой итерации	48
3.3.2	Policy Evaluation	49
3.3.3	Value Iteration	51
3.3.4	Policy Iteration	53
3.3.5	Generalized Policy Iteration	54
3.4	Табличные алгоритмы	56
3.4.1	Монте-Карло алгоритм	56
3.4.2	Экспоненциальное слаживание	57
3.4.3	Стохастическая аппроксимация	59
3.4.4	Temporal Difference	60
3.4.5	Exploration-exploitation дилемма	62
3.4.6	Q-learning	63
3.4.7	SARSA	64
4	Value-based подход	68
4.1	Deep Q-learning	68
4.1.1	Q-сетка	68
4.1.2	Переход к параметрической Q-функции	69
4.1.3	Декорреляция сэмплов	70
4.1.4	Таргет-сеть	71
4.1.5	DQN	72
4.2	Модификации DQN	73
4.2.1	Twin DQN	73
4.2.2	Clipped Twin DQN	74
4.2.3	Double DQN	74
4.2.4	Dueling DQN	74
4.2.5	Шумные сети (Noisy Nets)	75
4.2.6	Приоритизированный реплей (Prioritized DQN)	76
4.2.7	Multi-step DQN	78
A	Приложение	80
A.1	Натуральный градиент	80
A.1.1	Проблема параметризации	80
A.1.2	Матрица Фишера	81
A.1.3	Натуральный градиент	82
A.2	Обоснование формул CMA-ES	83
A.2.1	Вычисление градиента	83
A.2.2	Произведение Кронекера	84
A.2.3	Вычисление матрицы Фишера	85
A.2.4	Covariance Matrix Adaptation Evolution Strategy (CMA-ES)	87
A.3	Сходимость Q-learning	88
A.3.1	Action Replay Process	88
A.3.2	Ключевые свойства ARP	89
A.3.3	Схожесть ARP и настоящего MDP	90

ГЛАВА 1

Задача обучения с подкреплением

Ясно, что обучение с учителем это не та модель «обучения», которая свойственна интеллектуальным сущностям. Термин «обучение» здесь подменяет понятие интерполяции или, если уж на то пошло, построение алгоритмов неявным образом («смотрите, оно само обучилось, я сам ручками не прописывал, как кошечек от собачек отличать»). К полноценному обучению, на которое способен не только человек, но и в принципе живые организмы, задачи классического машинного обучения имеют лишь косвенное отношение. Значит, нужна другая формализация понятия «задачи, требующей интеллектуального решения», в которой обучение будет проводится не на опыте, заданном прецедентно (в виде обучающей выборки).

Термин **подкрепление** (reinforcement) пришёл из **поведенческой психологии** и обозначает награду или наказание за некоторый получившийся результат, зависящий не только от самих принятых решений, но и внешних, не обязательно подконтрольных, факторов. Под обучением здесь понимается поиск способов достичь желаемого результата **методом проб и ошибок** (trial and error), то есть попыток решить задачу и использование накопленного опыта для усовершенствования своей стратегии в будущем.

В данной главе будут введены основные определения и описана формальная постановка задачи.

Под желаемым результатом мы далее будем понимать максимизацию некоторой скалярной величины, называемой **наградой** (reward). Интеллектуальную сущность (систему/робота/алгоритм), принимающую решения, будем называть **агентом** (agent). Агент взаимодействует с **миром** (world) или **средой** (environment), которая задаётся зависящим от времени **состоянием** (state). Агенту в каждый момент времени в общем случае доступно только некоторое **наблюдение** (observation) текущего состояния мира. Сам агент задаёт процедуру выбора **действия** (action) по доступным наблюдениям; эту процедуру далее будем называть **стратегией** или **политикой** (policy). Процесс взаимодействия агента и среды задаётся **динамикой среды** (world dynamics), определяющей правила смены состояний среды во времени и генерации награды.

Буквы **s**, **a**, **r** зарезервируем для состояний, действий и наград соответственно; буквой **t** будем обозначать время в процессе взаимодействия.

§1.1. Модель взаимодействия агента со средой

1.1.1. Связь с оптимальным управлением

Математика поначалу пришла к очень похожей формализации следующим образом. Для начала, нужно ввести какую-то модель мира; в рамках **лапласовского детерминизма** можно предположить, что положение, скорости и ускорения всех атомов вселенной задают её текущее состояние, а изменение состояния происходит согласно некоторым диффурам:

$$\dot{s} = f(s, t)$$

Коли агент может как-то взаимодействовать со средой или влиять на неё, мы можем промоделировать взаимодействие следующим образом:

$$\dot{s} = f(s, a(s, t), t)$$

Для моделирования награды положим, что в каждый момент времени агент получает наказание в объёме $L(s, a(s, t), t)$. Итоговой наградой агента полагаем суммарную награду, то есть интеграл по времени:

$$\begin{cases} - \int_t L(s, a(s, t), t) dt \rightarrow \max_{a(s, t)} \\ \dot{s} = f(s, a(t), t) \end{cases}$$

За исключением того, что для полной постановки требуется задать ещё начальные и конечные условия, поставленная задача является общей формой задачи **оптимального управления** (optimal control). При этом обратим внимание на сделанные при постановке задачи предположения:

- время непрерывно;
- мир детерминирован;
- мир нестационарен (функция f напрямую зависит от времени t);
- модель мира предполагается известной, то есть функция f задана явно в самой постановке задачи;

Принципиально последнее: теория рассматривает способы для заданной системы дифференциальных уравнений поиска решений $a(s, t)$ аналитически; никакого «обучения методом проб и ошибок» здесь не предполагается.

Обучение с подкреплением исходит из других предположений: время дискретно, а среда — стохастична, но стационарна. В частности, в рамках этой теории можно построить алгоритмы для ситуации, когда модель мира агенту неизвестна, и единственный способ поиска решений — обучение на собственном опыте взаимодействия со средой.

1.1.2. Марковская цепь

Мы зададим модель мира следующим образом: будем считать, что существуют некоторые «законы физики», возможно, стохастические, которые определяют следующее состояние среды по предыдущему. При этом предполагается, что в текущем состоянии мира содержится вся необходимая информация для выполнения перехода, или, иначе говоря, выполняется **свойство Марковости** (Markov property): процесс зависит только от текущего состояния и не зависит от всей предыдущей истории.

Определение 1: *Марковской цепью (Markov chain)* называется пара $(\mathcal{S}, \mathcal{P})$, где:

- \mathcal{S} — множество состояний.
- \mathcal{P} — вероятности переходов $\{p(s_{t+1} | s_t) | t \in \{0, 1, \dots\}, s_t, s_{t+1} \in \mathcal{S}\}$.

Если дополнительно задать стартовое состояние s_0 , можно рассмотреть процесс, заданный марковской цепью. В момент времени $t = 0$ мир находится в состоянии s_0 ; сэмплируется случайная величина $s_1 \sim p(s_1 | s_0)$, и мир переходит в состояние s_1 ; сэмплируется $s_2 \sim p(s_2 | s_1)$, и так далее до бесконечности.

Мы далее делаем важное предположение, что законы мира не изменяются с течением времени. В аналогии с окружающей нас действительностью это можно интерпретировать примерно как «физические константы не изменяются со временем».¹

Определение 2: Марковская цепь называется **однородной** (time-homogeneous) или **стационарной** (stationary), если вероятности переходов не зависят от времени:

$$\forall t: p(s_{t+1} | s_t) = p(s_1 | s_0)$$

По определению, переходы \mathcal{P} стационарных марковских цепей задаются единственным условным распределением $p(s' | s)$. Апостроф ' канонично используется для обозначения «следующих» моментов времени, мы будем активно пользоваться этим обозначением.

¹вопрос на засыпку для любителей философии: а это вообще правда? Вдруг там через миллион лет гравитационная постоянная или скорость света уже будут другими в сорок втором знаке после запятой?..

Пример 1 — Конечные марковские цепи: Марковские цепи с конечным числом состояний можно задать при помощи ориентированного графа, дугам которого поставлены в соответствие вероятности переходов (отсутствие дуги означает нулевую вероятность перехода).

На рисунке приведён пример стационарной марковской цепи с 4 состояниями. Такую цепь можно также задать в виде матрицы переходов:

	■	■	■	■
	0.5	0.5		
■	0.25		0.75	
■	0.1	0.2	0.1	0.6
■		0.5		0.5

1.1.3. Среда

Как и в оптимальном управлении, для моделирования влияния агента на среду в вероятности переходов достаточно добавить зависимость от выбираемых агентом действий. Итак, наша модель среды — это «управляемая» марковская цепь.

Определение 3: *Средой* (environment) называется тройка $(\mathcal{S}, \mathcal{A}, \mathcal{P})$, где:

- \mathcal{S} — пространство состояний (state space), некоторое множество.
- \mathcal{A} — пространство действий (action space), некоторое множество.
- \mathcal{P} — функция переходов (transition function) или динамика среды (world dynamics): вероятности $p(s' | s, a)$.

В таком определении среды заложена марковость (независимость переходов от истории) и стационарность (независимость от времени). Время при этом дискретно, в частности, нет понятия «времени принятия решения агентом»: среда, находясь в состоянии s , ожидает от агента действие $a \in \mathcal{A}$, после чего совершает шаг, сэмплируя следующее состояние $s' \sim p(s' | s, a)$.

По дефолту, среда также предполагается **полностью наблюдаемой** (fully observable): агенту при выборе a_t доступно всё текущее состояние s_t в качестве входа. Иными словами, в рамках данного предположения понятия состояния и наблюдения для нас будут эквивалентны. Мы будем строить теорию в рамках этого упрощения; если среда не является полностью наблюдаемой, задача существенно усложняется, и необходимо переходить к формализму **частично наблюдаемых MDP** (partially observable MDP, PoMDP). Обобщение алгоритмов для PoMDP будет рассмотрено отдельно в разделе ??.

Пример 2 — Конечные среды: Среды с конечным числом состояний и конечным числом действий можно задать при помощи ориентированного графа, где для каждого действия задан свой комплект дуг.

На рисунке приведён пример среды с 2 действиями $\mathcal{A} = \{\blacksquare, \blacksquare\}$; дуги для разных действий различаются цветом. Возле каждого состояния выписана стратегия агента.

Пример 3 — Кубик-Рубик: Пространство состояний — пространство конфигураций Кубика-Рубика. Пространство действий состоит из 12 элементов (нужно выбрать одну из 6 граней, каждую из которых можно повернуть двумя способами). Следующая конфигурация однозначно определяется текущей кон-

фигурацией и действием, соответственно среда Кубика-Рубика **детерминирована**: задаётся вырожденным распределением, или, что тоже самое, обычной детерминированной функцией $s' = f(s, a)$.

1.1.4. Действия

Нас будут интересовать два вида пространства действий \mathcal{A} :

- конечное**, или **дискретное пространство действий** (discrete action space): $|\mathcal{A}| < +\infty$. Мы также будем предполагать, что число действий $|\mathcal{A}|$ достаточно мало (ситуации, когда это не так, представляют пока для алгоритмов серьёзные затруднения).
- непрерывное пространство действий (continuous domain): $\mathcal{A} \subseteq [-1, 1]^m$. Выбор именно отрезков $[-1, 1]$ является не ограничивающим общности распространённым соглашением. Задачи с таким пространством действий также называют задачами **непрерывного управления** (continuous control).



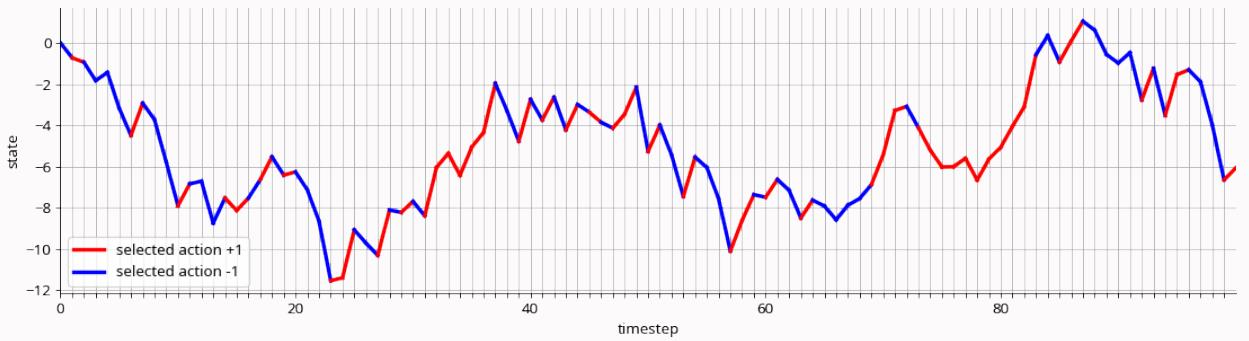
Заметим, что множество действий не меняется со временем и не зависит от состояния. Если в практической задаче предполагается, что множество допустимых действий разное в различных состояниях, в «законы физики» прописывается реакция на некорректное действие, например, случайный выбор корректного действия за агента.

В общем случае процесс выбора агентом действия в текущем состоянии может быть стохастичен. Таким образом, объектом поиска будет являться распределение $\pi(a | s), a \in \mathcal{A}, s \in \mathcal{S}$. Заметим, что факт, что нам будет достаточно искать стратегию в классе стационарных (независимых от времени), вообще говоря, потребует обоснования.

1.1.5. Траектории

Определение 4: Набор $\mathcal{T} := (a_0, s_1, a_1, s_2, a_2, s_3, a_3 \dots)$ называется **траекторией**.

Пример 4: Пусть в среде состояния описываются одним вещественным числом, $\mathcal{S} \equiv \mathbb{R}$, у агента есть два действия $\mathcal{A} = \{+1, -1\}$, а следующее состояние определяется как $s' = s + a + \varepsilon$, где $\varepsilon \sim \mathcal{N}(0, 1)$. Начальное состояние полагается равным нулю $s_0 = 0$. Сгенерируем пример траектории для случайной стратегии (вероятность выбора каждого действия равна 0.5):



Поскольку траектории — это случайные величины, которые заданы по постановке задачи конкретным процессом порождения (действия генерируются из некоторой стратегии, состояния — из функции переходов), можно расписать распределение на множестве траекторий:

Определение 5: Для данной среды, политики π и начального состояния $s_0 \in \mathcal{S}$ распределение, из которого приходят траектории \mathcal{T} , называется **trajectory distribution**:

$$p(\mathcal{T}) = p(a_0, s_1, a_1 \dots) = \prod_{t \geq 0} \pi(a_t | s_t) p(s_{t+1} | s_t, a_t)$$

Мы часто будем рассматривать мат.ожидания по траекториям, которые будем обозначать $\mathbb{E}_{\mathcal{T}}$. Под этим подразумевается бесконечная цепочка вложенных мат.ожиданий:

$$\mathbb{E}_{\mathcal{T}}(\cdot) := \mathbb{E}_{\pi(a_0 | s_0)} \mathbb{E}_{p(s_1 | s_0, a_0)} \mathbb{E}_{\pi(a_1 | s_1)} \dots (\cdot) \quad (1.1)$$

Поскольку часто придётся раскладывать эту цепочку, договоримся о следующем сокращении:

$$\mathbb{E}_{\mathcal{T}}(\cdot) = \mathbb{E}_{a_0} \mathbb{E}_{s_1} \mathbb{E}_{a_1} \dots (\cdot)$$

Однако в такой записи стоит помнить, что действия приходят из некоторой зафиксированной политики π , которая неявно присутствует в выражении. Для напоминания об этом будет, где уместно, использоваться запись $\mathbb{E}_{\tau \sim \pi}$.

1.1.6. Марковский процесс принятия решений (MDP)

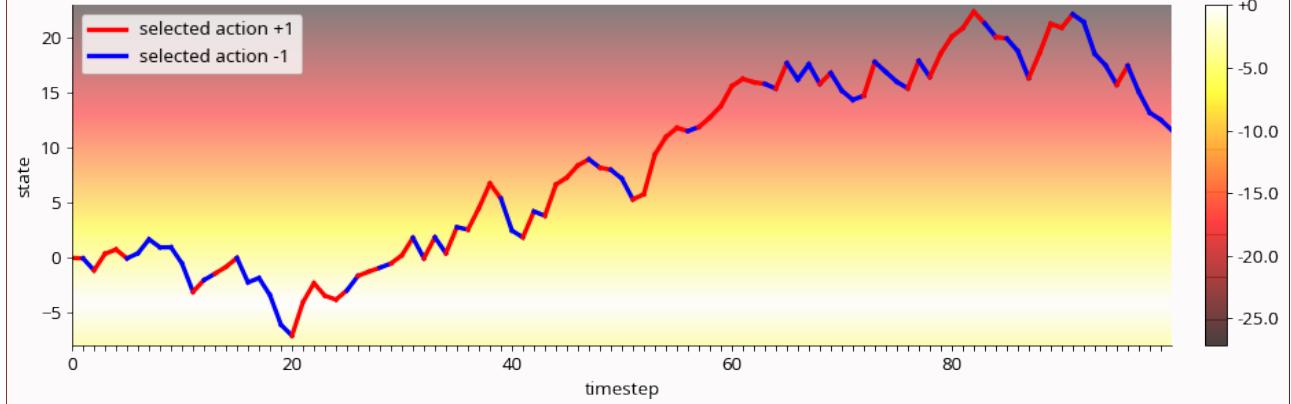
Для того, чтобы сформулировать задачу, нам необходимо в среде задать агенту цель — некоторый функционал для оптимизации. По сути, марковский процесс принятия решений — это среда плюс награда. Мы будем пользоваться следующим определением:

Определение 6: *Марковский процесс принятия решений* (Markov Decision Process, MDP) — это четвёрка $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r)$, где:

- $\mathcal{S}, \mathcal{A}, \mathcal{P}$ — среда.
- $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ — *функция награды* (reward function).

Сам процесс выглядит следующим образом. Для момента времени $t = 0$ начальное состояние мира полагается s_0 ; будем считать, оно дано дополнительно и фиксировано (формально его можно рассматривать как часть заданного MDP). Агент наблюдает всё состояние целиком и выбирает действие $a_0 \in \mathcal{A}$. Среда отвечает генерацией награды $r(s_0, a_0)$ и сэмплирует следующее состояние $s_1 \sim p(s' | s_0, a_0)$. Агент выбирает $a_1 \in \mathcal{A}$, получает награду $r(s_1, a_1)$, состояние s_2 , и так далее до бесконечности.

Пример 5: Для среды из примера 4 зададим функцию награды как $r(s, a) = |s + 4.2|$. Независимость награды от времени является требованием стационарности к рассматриваемым MDP:



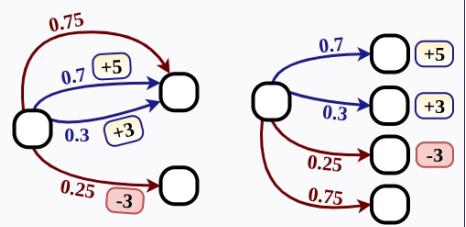
Формальное введение MDP в разных источниках чуть отличается, и из-за различных договорённостей одни и те же утверждения могут выглядеть совсем непохожим образом в силу разных исходных обозначений. Важно, что суть и все основные теоретические результаты остаются неизменными.

Теорема 1 — Эквивалентные определения MDP: Эквивалентно рассматривать MDP, где

- функция награды зависит только от текущего состояния;
- функция награды является стохастической;
- функция награды зависит от тройки (s, a, s') ;
- переходы и генерация награды задаётся распределением $p(r, s' | s, a)$;
- начальное состояние стохастично и генерируется из некоторого распределения $s_0 \sim p(s_0)$.

Скетч доказательства. Покажем, что всю стохастику можно «засовывать» в $p(s' | s, a)$. Например, стохастичность начального состояния можно «убрать», создав отдельное начальное состояние, из которого на первом шаге агент вне зависимости от выбранного действия перейдёт в первое по стохастичному правилу.

Покажем, что от самого общего случая (генерации награды и состояний из распределения $p(r, s' | s, a)$) можно перейти к детерминированной функции награды только от текущего состояния. Добавим в описание состояний информацию о последнем действии и последней полученной агентом награде, то есть для каждого возможного (имеющего ненулевую вероятность) перехода (s, a, r, s') размножим s' по числу¹ возможных исходов $p(r |$



$|s, a)$ и по числу действий. Тогда можно считать, что на очередном шаге вместо $r(s, a)$ агенту выдаётся $r(s')$, и вся стохастика процесса формально заложена только в функции переходов. ■

¹которое в худшем случае континуально, так как награда — вещественный скаляр.

Считать функцию награды детерминированной удобно, поскольку позволяет не городить по ним мат. ожидания (иначе нужно добавлять сэмплы наград в определение траекторий). В любом формализме всегда принято считать, что агент сначала получает награду и только затем наблюдает очередное состояние.

Определение 7: MDP называется **конечным** (finite MDP), если пространства состояний и действий конечны: $|\mathcal{S}| < \infty, |\mathcal{A}| < \infty$.

Пример 6: Для конечных MDP можно над дугами в графе среды указать награду за переход по ним или выдать награду состояниям (в рамках нашего формализма награда «за состояние» будет получена, давайте считать, при выполнении любого действия из данного состояния).

1.1.7. Эпизодичность

Во многих случаях процесс взаимодействия агента со средой может при определённых условиях «заканчиваться», причём факт завершения доступен агенту.

Определение 8: Состояние s называется **терминальным** (terminal) в MDP, если $\forall a \in \mathcal{A}$:

$$\mathbf{P}(s' = s | s, a) = 1 \quad r(s, a) = 0,$$

то есть с вероятностью 1 агент покинуть состояние не сможет.

Пример 7: В примере 6 самое правое состояние является терминальным.

Считается, что на каждом шаге взаимодействия агент дополнительно получает для очередного состояния s значение предиката $\text{done}(s) \in \{0, 1\}$, является ли данное состояние терминальным. По сути, после попадания в терминальное состояние дальнейшее взаимодействие бессмысленно (далнейшие события тривиальны), и, считается, что возможно произвести *reset* среды в s_0 , то есть начать процесс взаимодействия заново. Введение терминальных состояний именно таким способом позволит всюду в теории писать суммы по времени до бесконечности, не рассматривая отдельно случай завершения за конечное время.



Для агента все терминальные состояния в силу постановки задачи неразличимы, и их описания среда обычно не возвращает (вместо этого на практике она обычно проводит ресет и возвращает s_0 следующего эпизода).

Определение 9: Один цикл процесса от стартового состояния до терминального называется **эпизодом** (episode).

Продолжительности эпизодов (количество шагов взаимодействия) при этом, конечно, могут различаться от эпизода к эпизоду.

Определение 10: Среда называется **эпизодичной** (episodic), если для любой стратегии процесс взаимодействия гарантированно завершается не более чем за некоторое конечное T^{\max} число шагов.

Теорема 2 — Граф эпизодичных сред есть дерево: В эпизодичных средах вероятность оказаться в одном и том же состоянии дважды в течение одного эпизода равна нулю.

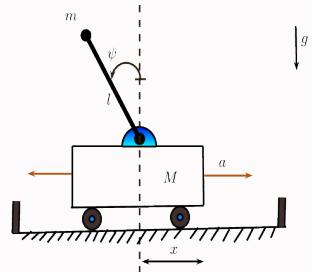
Доказательство. Если для некоторого состояния s при некоторой комбинации действий через T шагов агент с вероятностью $p > 0$ вернётся в s , при повторении такой же комбинации действий в силу марковости с вероятностью $p^n > 0$ эпизод будет длиться не менее nT шагов для любого $n \in \mathbb{N}$. Иначе говоря, эпизоды могут быть неограниченно долгими. ■

Пример 8 — Cartpole: К тележке на шарнире держится палка с грузиком.

Два действия позволяют придать тележке ускорение вправо или влево. Состояние описывается двумя числами: x-координатой тележки и углом, на которой палка отклонилась от вертикального положения.

Состояние считается терминальным, если x-координата стала слишком сильно отличной от нуля (тележка далеко уехала от своего исходного положения), или если палка отклонилась на достаточно большой угол.

Агент получает +1 каждый шаг и должен как можно дольше избегать терминальных состояний. Гарантии завершения эпизодов в этом MDP нет: агент в целом может справляться с задачей бесконечно долго.



На практике, в средах обычно существуют терминальные состояния, но нет гарантии завершения эпизодов за ограниченное число шагов. Это лечат при помощи таймера — жёсткого ограничения, требующего по истечении T^{\max} шагов проводить в среде ресет. Чтобы не нарушить теоретические предположения, необходимо тогда заложить информацию о таймере в описание состояний, чтобы агент знал точное время до прерывания эпизода. Обычно так не делают; во многих алгоритмах возможно использовать только «начала» траекторий, учитывая, что эпизод не был доведён до конца. Формально при этом нельзя полагать $\text{done}(s_{T^{\max}}) = 1$, но в коде зачастую так всё равно делают. Например, для Cartpole в реализации OpenAI Gym по умолчанию по истечении 200 шагов выдаётся флаг **done**, что формально нарушает марковское свойство.

1.1.8. Дисконтирование

Наша задача заключается в том, чтобы найти стратегию π , максимизирующую среднюю суммарную награду. Формально, нам явно задан функционал для оптимизации:

$$\mathbb{E}_{\mathcal{T} \sim \pi} \sum_{t \geq 0} r_t \rightarrow \max_{\pi} \quad (1.2)$$

где $r_t := r(s_t, a_t)$ — награда на шаге t .

Мы хотим исключить из рассмотрения MDP, где данный функционал может улететь в бесконечность² или не существовать вообще. Во-первых, введём ограничение на модуль награды за шаг, подразумевая, что среда не может поощрять или наказывать агента бесконечно сильно:

$$\forall s, a: |r(s, a)| \leq r^{\max} \quad (1.3)$$

Чтобы избежать парадоксов, этого условия нам не хватит³. Введём **дисконтирование** (discounting), коэффициент которого традиционно обозначают γ :

Определение 11: *Дисконтированной кумулятивной наградой* (discounted cumulative reward) или *total return* для траектории \mathcal{T} и $\gamma \in (0, 1]$ называется

$$R(\mathcal{T}) := \sum_{t \geq 0} \gamma^t r_t \quad (1.4)$$

У дисконтирования есть важная интерпретация: мы полагаем, что на каждом шаге с вероятностью $1 - \gamma$ взаимодействие обрывается, и итоговым результатом агента является та награда, которую он успел собрать до прерывания. Это даёт приоритет получению награды в ближайшее время перед получением той же награды через некоторое время. Математически смысл дисконтирования, во-первых, в том, чтобы в совокупности с требованием (1.3) гарантировать ограниченность оптимизируемого функционала, а во-вторых, выполнение условий некоторых теоретических результатов, которые явно требуют $\gamma < 1$. В силу последнего, гамму часто рассматривают как часть MDP.

²если награда может быть бесконечной, начинаются всякие парадоксы, рассмотрения которых мы хотим избежать. Допустим, в некотором MDP без терминальных состояний мы знаем, что оптимальная стратегия способна получать +1 на каждом шаге, однако мы смогли найти стратегию, получающую +1 лишь на каждом втором шаге. Формально, средняя суммарная награда равна бесконечности у обоих стратегий, однако понятно, что найденная стратегия «неоптимальна».

³могут возникнуть ситуации, где суммарной награды просто не существует (например, если агент в бесконечном процессе всегда получает +1 на чётных шагах и -1 на нечётных).

Определение 12: Скором (score или performance) стратегии π в данном MDP называется

$$J(\pi) := \mathbb{E}_{\mathcal{T} \sim \pi} R(\mathcal{T}) \quad (1.5)$$

Итак, задачей обучения с подкреплением является оптимизация для заданного MDP средней дисконтированной кумулятивной награды:

$$J(\pi) \rightarrow \max_{\pi}$$

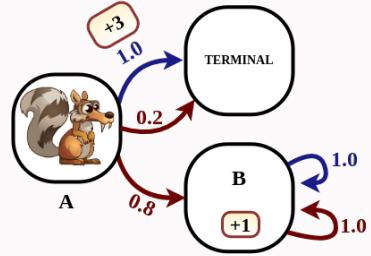
Пример 9: Посчитаем $J(\pi)$ для приведённого на рисунке MDP, $\gamma = \frac{10}{11}$ и начального состояния A. Ясно, что итоговая награда зависит только от стратегии агента в состоянии A, поэтому можно рассмотреть все стратегии, обозначив $\pi(a = \blacksquare | s = A)$ за параметр стратегии $\theta \in [0, 1]$.

С вероятностью θ агент выберет действие \blacksquare , после чего попадёт в состояние B с вероятностью 0.8. Там вне зависимости от стратегии он начнёт крутиться и получит в пределе

$$\gamma + \gamma^2 + \dots + = \sum_{t \geq 1} \gamma^t = \frac{\gamma}{1 - \gamma} = 10.$$

Ещё с вероятностью $1 - \theta$ агент выберет \blacksquare , получит +3 и попадёт в терминальное состояние, после чего эпизод завершится. Итого:

$$J(\pi) = \underbrace{(0.8\theta) \cdot 10}_{\blacksquare} + \underbrace{(1 - \theta) \cdot 3}_{\blacksquare} = 3 + 5\theta$$



Видно, что оптимально выбрать $\theta = 1$, то есть всегда выбирать действие \blacksquare . Заметим, что если бы мы рассмотрели другое γ , мы бы могли получить другую оптимальную стратегию; в частности, при $\gamma = \frac{15}{19}$ значение $J(\pi)$ было бы константным для любых стратегий, и все стратегии были бы оптимальными.

Всюду далее подразумевается⁴ выполнение требования ограниченности награды (1.3), а также или дисконтирования $\gamma < 1$, или эпизодичности среды.

Утверждение 1: При сделанных предположениях скор ограничен.

Доказательство. Если $\gamma < 1$, то по свойству геометрической прогрессии для любых траекторий \mathcal{T} :

$$R(\mathcal{T}) = \left| \sum_{t \geq 0} \gamma^t r_t \right| \leq \frac{1}{1 - \gamma} r^{\max}$$

Если же $\gamma = 1$, но эпизоды гарантированно заканчиваются не более чем за T^{\max} шагов, то суммарная награда не превосходит по модулю $T^{\max} r^{\max}$. Следовательно, скор как мат.ожидание от ограниченной величины также удовлетворяет этим ограничениям. ■

§1.2. Алгоритмы обучения с подкреплением

1.2.1. Условия задачи RL

Основной постановкой в обучении с подкреплением является задача нахождения оптимальной стратегии на основе собственного опыта взаимодействия. Это означает, что алгоритму обучения изначально доступно только:

- вид пространства состояний — количество состояний в нём в случае конечного числа, или размерность пространства \mathbb{R}^d в случае признакового описания состояний.
- вид пространства действий — непрерывное или дискретное. Некоторые алгоритмы будут принципиально способны работать только с одним из этих двух видов.
- взаимодействие со средой, то есть возможность для предоставленной алгоритмом стратегии π генерировать траектории $\mathcal{T} \sim \pi$; иными словами, принципиально доступны только сэмплы из trajectory distribution (1.1).

⁴в качестве акта педантичности оговоримся, что также всюду подразумевается измеримость всех функций, необходимая для существования всех рассматриваемых интегралов и мат.ожиданий.

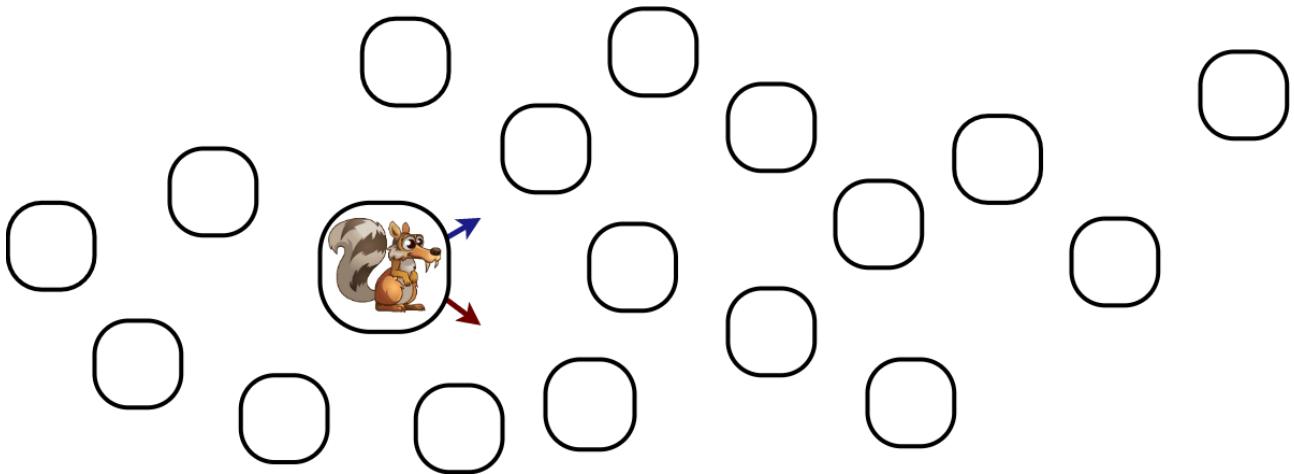


Рис. 1.1: Примерно так выглядит MDP для начинающего обучение агента.

Итак, в отличие от обучения с учителем, где датасет «дан алгоритму на вход», здесь агент должен сам собрать данные. Находясь в некотором состоянии, обучающийся агент обязан выбрать ровно одно действие, получить ровно один сэмпл s' и продолжить взаимодействие (накопление опыта — сбор сэмплов) из s' . Собираемые в ходе взаимодействия данные и представляют собой всю доступную агенту информацию для улучшения стратегии.

Определение 13: Пятерки $\mathbb{T} := (s, a, r, s', \text{done})$, где $r := r(s, a)$, $s' \sim p(s' | s, a)$, $\text{done} := \text{done}(s')$, называются *переходами* (transitions).

Таким образом, в RL-алгоритме должна быть прописана стратегия взаимодействия со средой во время обучения (*behavior policy*), которая может отличаться от «итоговой» стратегии (*target policy*), предназначенной для использования в среде по итогам обучения.

1.2.2. On-policy vs Off-policy

Доступной помощью для алгоритма могут быть *данные от эксперта*, то есть записи взаимодействия со средой некоторой стратегии (или разных стратегий), не обязательно, вообще говоря, оптимальной. Алгоритм RL, возможно, сможет эти данные как-то использовать, или хотя бы на них преодолеться. Предобучение, в простейшем случае, выглядит так: если в алгоритме присутствует параметрически заданная стратегия π_θ , можно решать задачу *имитационного обучения* (imitation learning), т.е. воспроизведения действий эксперта. Восстановление по парам s, a функции $\mathcal{S} \rightarrow \mathcal{A}$ — это обычная задача обучения с учителем, однако, если эксперт не оптимален, обученная стратегия вряд ли будет действовать хоть сколько-то лучше. Понятна прямая аналогия с задачей обучения с учителем, где верхняя граница качества алгоритма определяется качеством разметки; если разметка зашумлена и содержит ошибки, обучение вряд ли удастся. В обучении с подкреплением нам не даны «правильные действия», и, когда мы будем каким-либо образом сводить задачу к задачам регрессии и классификации, нам будет важно обращать внимание на то, как мы «собираем себе разметку» из опыта взаимодействия со средой и какое качество мы можем от этой разметки ожидать. В идеале, с каждым шагом алгоритм сможет собирать себе всё более и более «хорошую» разметку, за счёт неё выбирать всё более и более оптимальные действия, и так «вытягивать сам себя из болота».

Определение 14: Алгоритм RL называется *off-policy*, если он может использовать для обучения опыт взаимодействия произвольной стратегии.

Определение 15: Алгоритм RL называется *on-policy*, если для очередной итерации алгоритма ему требуется опыт взаимодействия некоторой конкретной, предоставляемой самим алгоритмом, стратегии.

Некоторое пояснение названия этих терминов: если мы обучаем некоторую стратегию π по сэмплам любой другой произвольной стратегии μ , то мы проводим «off-policy» обучение, обучаем политику «не по ней же самой». On-policy алгоритмам будет нужно «отправлять в среду конкретную стратегию», поскольку они будут способны обучать π лишь по сэмплам из неё же самой.

Off-policy алгоритм должен уметь проводить очередной шаг обучения на произвольных траекториях, сгенерированных произвольными (возможно, разными, возможно, неоптимальными) стратегиями. Понятие принципиально важно тем, что алгоритм может потенциально переиспользовать траектории, полученные старой версией стратегии со сколь угодно давних итераций. Если алгоритм может переиспользовать опыт, но с ограничениями (например, только с недавних итераций, или только из наилучших траекторий),

то мы всё равно будем относить его к on-policy. Это не означает, что для on-policy алгоритма совсем бесполезны данные от (возможно, неоптимального) эксперта; почти всегда можно придумать какую-нибудь эвристику, как воспользоваться ими для хотя бы инициализации (при помощи того же имитационного обучения). Важно, что off-policy алгоритм сможет на данных произвольного эксперта провести «полное» обучение, то есть условно сойтись к оптимуму при достаточном объёме и разнообразии экспертной информации, не потребовав вообще никакого дополнительного взаимодействия со средой.

1.2.3. Концепция model-free алгоритмов

Ещё одним существенным изменением «правил игр» является наличие у агента прямого доступа к функции переходов $p(s' | s, a)$ и функции награды.

Определение 16: Будем говорить, что у агента есть *симулятор* или «доступ к функции переходов», если он знает функцию награды и может в любой момент процесса обучения сэмплировать произвольное число сэмплов из $p(s' | s, a)$ для любого набора пар s, a .

Симулятор — по сути копия среды, которую агент во время обучения может откатывать к произвольному состоянию. Симулятор позволяет агенту строить свою стратегию при помощи *планирования* (planning) — рассмотрения различных вероятных версий предстоящего будущего и использования их для текущего выбора действия. При использовании планирования в идеальном симуляторе никакого процесса непосредственного обучения в алгоритме может не быть.

Пример 10: Примером задач, в которых у алгоритма есть симулятор, являются пятнашки или кубик-рубик. То есть, пытаясь создать алгоритм, собирающий кубик-рубик, мы, естественно, можем пользоваться знаниями о том, в какую конфигурацию переводят те или иные действия текущее положение, и таким образом напрописываются какие-то алгоритмы разумного перебора — «планирование».

Пример 11: Любые задачи, в которых среда реализована виртуально, можно рассматривать как задачи, где у агента есть симулятор. Например, если мы хотим обучить бота в Марио, мы можем сказать: да у нас есть исходники кода Марио, мы можем взять любую игровую ситуацию (установить симулятор в любое состояние) и для любого действия посмотреть, что будет дальше. В RL по умолчанию считается, что в видеоиграх такого доступа нет: динамика среды изначально агенту неизвестна.

Пример 12: Примером задач, в которых у алгоритма принципиально нет симулятора, являются любые задачи реальной робототехники. Важно, что даже если окружение реального робота симулируется виртуально, такая симуляция неточна — отличается от реального мира. В таких ситуациях можно говорить, что имеется *неидеальный симулятор*. Отдельно стоит уточнить, доступен ли симулятор реальному роботу в момент принятия решения (возможно, симулятор реализован на куда более вычислительно мощной отдельной системе) — тогда он может использоваться во время обучения, но не может использоваться в итоговой стратегии.

По умолчанию всегда считается, что доступа к динамике среды нет, и единственное, что предоставляет алгоритму — среда, с которой возможно взаимодействовать. Можно ли свести такую задачу к планированию? В принципе, алгоритм может пытаться обучать себе подобный симулятор — строить генеративную модель, по s, a выдающую $s', r(s, a), \text{done}(s')$ — и сводить таким образом задачу к планированию. Приближение тогда, естественно, будет неидеальным.

Обучение симулятора сопряжено и с рядом других нюансов. Например, в сложных средах в описании состояний может храниться колоссальное количество информации. Построение моделей, предсказывающих будущее, может оказаться вычислительно неподъёмной и неоправданно дорогой задачей.

Одна из фундаментальных парадигм обучения с подкреплением, вероятно, столь же важная, как парадигма end-to-end обучения для глубокого обучения — идея *model-free* обучения. Давайте не будем учить динамику среды и перебирать потенциальные варианты будущего для поиска хороших действий, а выучим напрямую связь между текущим состоянием и оптимальными действиями.

Определение 17: Алгоритм RL классифицируется как *model-free*, если он не использует и не пытается выучить модель динамики среды $p(s' | s, a)$.

1.2.4. Классификация RL-алгоритмов

При рассмотрении алгоритмов RL мы начнём с рассмотрения именно model-free алгоритмов и большую часть времени посвятим им. Их часто делят на следующие подходы:

- **мета-эвристики** (metaheuristic) никак не используют внутреннюю структуру взаимодействия среды и агента, и рассматривают задачу максимизации $J(\pi)$ как задачу «black box оптимизации»: можно примерно оценивать, чему равно значение функционала для разных стратегий, а структура задачи — формализм MDP — не используется; мы рассмотрим мета-эвристики в главе 2 как не требующие построения особой теории.
- **value-based** алгоритмы получают оптимальную стратегию неявно через теорию оценочных функций, которую мы рассмотрим в главе 3. Эта теория позволит нам построить value-based алгоритмы (глава 4) и будет использоваться всюду далее.
- **policy gradient** алгоритмы максимизируют $J(\pi)$, используя оценки градиента функционала по параметрам стратегии; мы сможем помочь процессу оптимизации, правильно воспользовавшись оценочными функциями (глава ??).

Затем в главе ?? мы отдельно обсудим несколько алгоритмов специально для непрерывных пространств действий, находящихся на стыке value-based и policy gradient подхода, и увидим, что между ними довольно много общего. Наконец, **model-based** алгоритмы, которые учат или используют предоставленную модель среды $p(s' | s, a)$, и которые обычно выделяют в отдельную категорию, будут рассмотрены после в главе ??.

1.2.5. Критерии оценки RL-алгоритмов

При оценивании алгоритмов принципиально соотношение трёх критериев:

- **performance**: Монте-Карло оценка значения $J(\pi)$;
- **wall-clock time**: реальное время работы, потребовавшееся алгоритму для достижения такого результата (в полной аналогии с классическими методами оптимизации);
- **sample efficiency**: количество сэмплов (или шагов) взаимодействия со средой, потребовавшихся алгоритму. Этот фактор может быть ключевым, если взаимодействие со средой дорого (например, обучается реальный робот);



Поскольку **победами над кожаными мешками в дотах** уже никого не удивишь, вторые два фактора начинают играть всю большую роль.

В отличие от классических методов оптимизации, речи о критерии останова идти не будет, поскольку адекватно разумно проверить около-оптимальность текущей стратегии не представляется возможным в силу слишком общей постановки задачи. Считается, что оптимизация (обучение за счёт получения опыта взаимодействия) происходит, пока доступны вычислительные ресурсы; в качестве итога обучения предоставляется или получившаяся стратегия, или наилучшая встретившаяся в ходе всего процесса.

1.2.6. Сложности задачи RL

Обсудим несколько «именованных» проблем задачи обучения с подкреплением, с которыми сталкивается любой алгоритм решения.

Проблема **застревания в локальных оптимумах** приходит напрямую из методов оптимизации. В оптимизируемом функционале (1.5) может существовать огромное количество стратегий π , для которых его значение далеко не максимально, но все в некотором смысле «соседние» стратегии дают в среднем ещё меньшую награду.

Пример 13: Часто агент может выучить тривиальное «пассивное» поведение, которое не приносит награды, но и позволяет избегать какие-то штрафов за неудачи. Например, агент, который хочет научиться перепрыгивать через грабли, чтобы добраться до тортика (+1), может несколько раз попробовать отправиться за призом, но наступить на грабли (-1), и выучить ничего не делать (+0). Ситуация весьма типична: например, Марио может пару раз попробовать отправиться покорять первый уровень, получить по башке и решить не ходить вправо, а тупить в стену и получать «безопасный» +0. Для нашей задачи оптимизации это типичнейшие локальные экстремумы: «похожие стратегии» набирают меньше текущей, и, чтобы добраться до большей награды, нужно как-то найти совершенно новую область в пространстве стратегий.



Другие проблемы куда более характерны именно для RL. Допустим, агент совершает какое-то действие, которое запускает в среде некоторый процесс. Процесс протекает сам по себе без какого-либо дальнейшего вмешательства агента и завершается через много шагов, приводя к награде. Это проблема **отложенного сигнала** (delayed reward) — среда даёт фидбэк агенту спустя какое-то (вообще говоря, неограниченно длительное) время.

Пример 14: Пример из видеоигр — в игре Atari Space Invaders очки даются в момент, когда удачный выстрел попал во вражеское НЛО, а не когда агент этот выстрел, собственно, совершает. Между принятием решения и получением сигнала проходит до нескольких секунд, и за это время агент принимает ещё несколько десятков решений.



Смежная проблема — какое именно из многих совершивших агентом действий привело к сигналу награды? **Credit assignment problem** — даже для уже собранного опыта может быть тяжело оценить, какие действия были правильными, а какие нет.

Пример 15: Вы поймали скунса, сварили яичницу, завезли банку горчицы в ближайший магазин штор, написали научную статью про шпроты, накормили скунса яичницей, сыграли в боулинг глобусом, вернулись домой после тяжёлого дня и получили +1. Вопрос: почему вы научились?

Поскольку функция награды может быть произвольная, довольно типично, когда сигнал от среды — неконстантная награда за шаг — приходит очень редко. Это проблема **разреженной награды** (sparse reward).

Пример 16 — Mountain Car: Визуализация задачи в OpenAI Gym: тележка хочет забраться на горку, но для этого необходимо поехать в противоположном направлении, чтобы набрать разгона. Состояния описываются двумя числами (x-координата тележки и её скорость); действий три (придать ускорения вправо, влево, или ничего не делать). Функция награды равна -1 всюду: задачей агента является как можно скорее завершить эпизод. Однако, терминальное состояние — это вершина горки, и для того, чтобы достичь его, нужно «уже уметь» задачу решать.

Наконец, проблема, обсуждению которой мы посвятим довольно много времени — дилемма «исследовать или использовать» (exploration-exploitation trade-off). Пока ограничимся лишь примером для иллюстрации.

Пример 17: Вы решили пойти сегодня в ресторан. Следует ли отправится в ваш любимый ресторан, или попробовать новый, в котором вы ещё ни разу не были?

Практическая проблема, отчасти связанная с тем, что алгоритму необходимо постоянно «пробовать новое» — проблема **«безопасного обучения»** (Safe RL). Грубо говоря, некоторые взаимодействия агента со средой крайне нежелательны даже во время обучения, в том числе когда агент ещё только учится. Эта проблема возникает в первую очередь для реальных роботов.

Пример 18: Вы хотите научить реального робота мыть посуду. В начале обучения робот ничего не умеет и рандомно размахивает конечностями, бьёт всю посуду, переворачивает стол, сносит все лампочки и «в исследовательских целях» самовыкидывается из окна. В результате, начать второй обучающий эпизод становится довольно проблематично.



Здесь надо помнить, что безопасный RL не о том, как «не сбивать пешеходов» при обучении автономных автомобилей; он о том, как сбивать меньше пешеходов. RL по определению обучается на собственном опыте и в том числе собственных ошибках, и эти ошибки ему либо нужно совершить, либо получить в форме некоторой априорной информации (экспертных данных), хотя последнее всё равно не защищает от дальнейших исследований любых областей пространства состояний. Это означает, что на практике единственная полноценная защита от нежелательного поведения робота может быть проведена исключительно на этапе построения среды. То есть среда должна быть устроена так, что робот в принципе не может совершить нежелательных действий: опасные ситуации должны детектироваться средой (то есть — внешне, внешним отдельным алгоритмом), а взаимодействие — прерываться (с выдачей, например, отрицательной награды для агента).



Одна из причин распространения видеоигр для тестирования RL — отсутствие проблемы Safe RL: не нужно беспокоиться о том, что робот «что-то сломает» в процессе сбора опыта, если среда уже задана программной симуляцией.

И ещё одна, вероятно, главная проблема⁵. **Откуда берётся награда?** Алгоритмы RL предполагают, что награда, как и среда, заданы, «поданы на вход», и эту проблему наши алгоритмы обучения, в отличие от предыдущих, решать идеологически не должны. Но понятно, что если для практического применения обучения с учителем боттлнеком часто является необходимость размечивать данные — «предоставлять обучающий сигнал» — то в RL необходимо аккуратно описать задачу при помощи функции награды.

Определение 18: «*Reward hypothesis*»: любую интеллектуальную задачу можно задать (определить) при помощи функции награды.

В обучении с подкреплением принято полагать эту гипотезу истинной. Но так ли это? RL будет оптимизировать ту награду, которую ему предоставят, и дизайн функции награды в ряде практических задач оказывается проблемой.

Пример 19 — «Взлом» функции награды: Классический пример того, как RL оптимизирует не то, что мы ожидаем, а ту награду, которую ему подсунули. Причина зацикливания агента видна в нижнем левом углу, где отображается счёт игры.

Пример 20: Попробуйте сформулировать функцию награды для следующих интеллектуальных задач:

- очистка мебели от пыли;
- соблюдение правил дорожного движения автономным автомобилем;
- захват мира;

1.2.7. Бенчмарки

Для тестирования RL алгоритмов есть несколько распространившихся бенчмарков. Неистощаемым источником тестов для обучения с подкреплением с дискретным пространством действий являются видеоигры, где, помимо прочего, уже задана функция награды — счёт игры.

Пример 21 — Игры Atari: Atari — набор из 57 игр с дискретным пространством действий. Наблюдением является экран видео-игры (изображение), у агента имеется до 18 действий (в некоторых играх действия, соответствующие бездействующим кнопкам джойстика, по умолчанию убраны). Награда — счёт в игре. [Визуализация игр из OpenAI Gym](#).

⁵а точнее, в принципе главная проблема всей нашей жизни (what is your reward function?)

При запуске алгоритмов на Atari обычно используется препроцессинг. В общем случае MDP, заданное исходной игрой, не является полностью наблюдаемым: например, в одной из самых простых игр Pong текущего экрана недостаточно, чтобы понять, в какую сторону летит шарик. Для борьбы с этим состоянием считают последние 4 кадра игры (*frame stack*), что для большинства игр достаточно.

Добавляется и препроцессинг самого входного изображения — в оригинале оно сжимается до размера 84x84 и переводится в чёрно-белое.

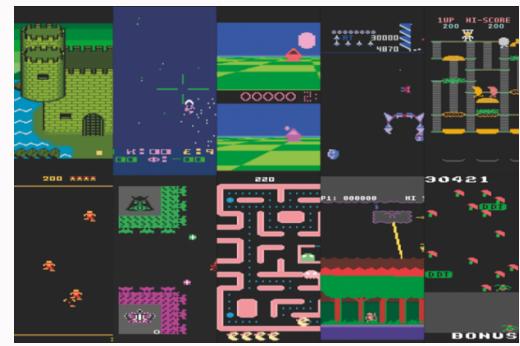
Чтобы агент не имел возможности менять действие сильно чаще человека, применяют *frame skip* — агент выбирает следующее действие не каждый кадр, а, скажем, раз в 4 кадра. В течение этих 4 кадров агент нажимает одну и ту же комбинацию кнопок. В Space Invaders это привело к неожиданным последствиям: агент перестаёт видеть выстрелы, которые «исчезают» как раз каждые 4 кадра.

Функция переходов в Atari — детерминированная. Есть опасения, что это может приводить к «запоминанию» хороших траекторий, поэтому распространено использование *sticky actions* — текущее выбранное действие повторяется для k кадров, где k определяется случайно (например, для очередного кадра подбрасывается монетка, и с вероятностью 0.5 агент повторяет предыдущее действие).

Для оценки алгоритмов используется *Human normalized score*: пусть *agentScore* — полученная оценка $J(\pi)$ для обучившегося агента, *randomScore* — случайной стратегии, *humanScore* — средний результат человека, тогда Human normalized score равен

$$\frac{\text{agentScore} - \text{randomScore}}{\text{humanScore} - \text{randomScore}}.$$

Эта величина усредняется по 57 играм для получения качества алгоритма.



Пример 22 — Atari RAM: Игры Atari представлены в ещё одной версии — «RAM-версии». Состоянием считается не изображение экрана, а 128 байт памяти Atari-консоли, в которой содержится вся информация, необходимая для расчёта игры (координаты игрока и иные параметры). По определению, такое состояние «полностью наблюдаемое», и также может использоваться для тестирования алгоритмов.

Для задач непрерывного управления за тестовыми средами обращаемся к физическим движкам и прочим симуляторам. Здесь нужно оговориться, что схожие задачи в разных физических движках могут оказаться довольно разными, в том числе по сложности для RL алгоритмов.

Пример 23: Задача научить ходить какое-нибудь существо весьма разнообразна. Под «существом» понимается набор сочленений, каждое из которых оно способно «напрягать» или «расслаблять» (для каждого выдаётся вещественное число в диапазоне $[-1, 1]$). Состояние обычно описано положением и скоростью всех сочленений, то есть является всего лишь небольшим векторчиком. Однако, несмотря на «компактное» пространство состояний и небольшую размерность пространства действий (случаи, когда нужно выдавать в качестве действия векторы размерности порядка 20, уже считаются достаточно тяжёлыми), типичная задача в рамках представленной симуляции физики научиться добираться как можно дальше в двумерном или трёхмерном пространстве обычно является довольно-таки сложной.



Пример 24: В робототехнике и симуляциях роботов может получать информацию об окружающем мире как с камеры, наблюдая картинку перед собой, так и узнавать о расположении объектов вокруг с помощью разного рода сенсоров, например, при помощи *ray cast*-ов — расстояния до препятствия вдоль некоторого направления, возможно, с указанием типа объекта. Преимущество последнего представления перед видеокамерой в компактности входного описания (роботу не нужно учиться обрабатывать входное изображение). В любом случае, входная информация редко когда полностью описывает состояние всего окружающего мира, и в подобных реальных задачах требуется формализм частично наблюдаемых сред.

ГЛАВА 2

Мета-эвристики

В данной главе мы рассмотрим первый подход к решению задачи, часто называемый «эволюционным». В этом подходе мы никак не будем использовать формализацию процесса принятия решений (а следовательно, не будем использовать какие-либо результаты, связанные с изучением MDP) и будем относиться к задаче как к black-box оптимизации: мы можем отправить в среду поиграть какую-то стратегию и узнать, сколько примерно она набирает, и задача алгоритма оптимизации состоит в том, чтобы на основе лишь этой информации предлагать, какие стратегии следует попробовать следующими.

§2.1. Бэйзлайны

2.1.1. Задача безградиентной оптимизации

Определение 19: *Мета-эвристикой* (metaheuristic) называется метод black-box оптимизации

$$J(\theta) \rightarrow \max_{\theta \in \Theta}$$

со *стохастическим оракулом нулевого порядка* (stochastic zeroth-order oracle), то есть возможностью для каждой точки $\theta \in \Theta$ получить несмешённую оценку $\hat{J} \approx J(\theta)$.

Такие методы также называются *безградиентными* (gradient-free), поскольку не используют градиент функции и в принципе не предполагают её дифференцируемости. Понятно, что такие методы — «универсальный» инструмент (читать, «инструмент последней надежды»), который можно использовать для любой задачи оптимизации. В первую очередь, этот инструмент полезен, если пространство аргументов Θ нетривиально (например, графы) или если оптимизируемая функция принципиально недифференцируема, состоит из седловых точек («inadequate landscape») или есть другие препятствия для градиентной оптимизации.

Заметим, что если Θ — конечное множество (о градиентной оптимизации тогда речи идти не может), задача сводится к следующей: надо найти тот аргумент $\theta \in \Theta$, для которого настоящее значение $J(\theta)$ максимально, при этом используя как можно меньше стохастических оценок \hat{J} оракула. Это задача многорукого бандита, которую мы обсудим отдельно в секции ???. В теории мета-эвристик опция «запросить оракул в одной и той же точке несколько раз» в ходе алгоритма обычно не рассматривается; предполагается достаточно богатое пространство Θ , для которого более прагматичной альтернативой кажется запросить значение условно в «соседней» точке вместо уточнения значения оракула для одной и той же.

В контексте обучения с подкреплением, чтобы свести задачу к black-box оптимизации, достаточно представить стратегию $\pi_\theta(a | s)$ в параметрическом семействе с параметрами $\theta \in \Theta$. В качестве J , конечно, выступает наш оптимизируемый функционал (1.5)

$$J(\theta) := \mathbb{E}_{\mathcal{T} \sim \pi_\theta} R(\mathcal{T}) \rightarrow \max_{\theta}$$

а в качестве \hat{J} — его Монте-Карло оценка:

$$\hat{J}(\theta) := \frac{1}{B} \sum_{i=1}^B R(\mathcal{T}_i), \quad \mathcal{T}_i \sim \pi_\theta, i \in \{1 \dots B\}$$



Если дисперсия оценки достаточно высока (число сэмплов B недостаточно велико), почти все далее рассматриваемые алгоритмы сломаются (будут выживать «везучие», а не «сильнейшие»). Поэтому может оказаться крайне существенным использовать $B > 1$, даже если π_θ — семейство детерминированных стратегий.

Будем проникаться местной терминологией:

Определение 20: Точку θ , в которой алгоритм оптимизации запрашивает значение оракула, будем называть **особью** (individuals, particles), а само значение $\hat{J}(\theta)$ для данной особи — её **оценкой** или **приспособленностью** (fitness).

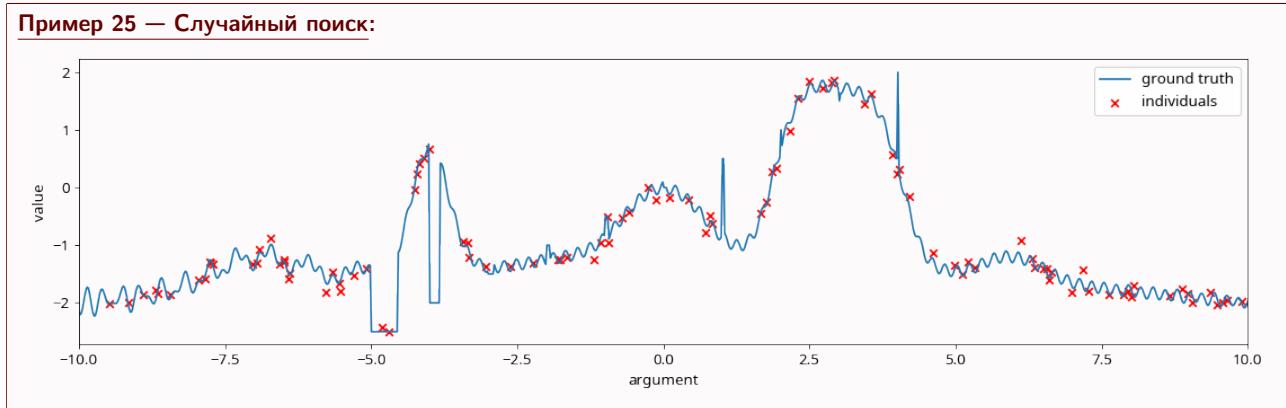
В силу гигантского разнообразия мета-эвристик (от метода светлячков до колоний императорских пингвинов) на полноту дальнейшее изложение, конечно же, не претендует, и стоит воспринимать рассуждение как попытку структурировать мотивации некоторых из основных идей. В частности, нас в первую очередь будут интересовать алгоритмы, в той или иной степени успешно применявшиеся в RL.

2.1.2. Случайный поиск

Случайный поиск (random search) — метод оптимизации и самый простой пример мета-эвристики.

Определение 21: Распределение $q(\theta)$ в пространстве Θ будем называть *стратегией перебора*.

Случайный поиск сводится к сэмплированию из стратегии перебора особей $\theta_k \sim q(\theta)$ ($k \in \{0, 1, 2 \dots\}$), после чего в качестве результата выдаётся особь с наилучшей оценкой.



Забавно, что случайный поиск — метод глобальной оптимизации: если $\forall \theta \in \Theta: q(\theta) > 0$, после достаточного числа итераций метод найдёт сколь угодно близкое к глобальному оптимуму решение¹. Есть ещё один парадокс грубого перебора: если в наличии есть неограниченное число серверов, то возможно запустить на каждом вычисление приспособленности одной особи, и за время одного вычисления провести «глобальную» оптимизацию.

Идея случайного поиска, на самом деле, вводит основные понятия мета-эвристик. Нам придётся так или иначе запросить у оракула приспособленности некоторого набора особей и так или иначе в итоге отобрать лучший. Для имитации умности происходящего введём весёлую нотацию.

Определение 22: Набор особей $\mathcal{P} := (\theta_i \mid i \in \{1, 2 \dots N\})$ называется *популяцией* (population) размера N .

Определение 23: Запрос оракула для всех особей популяции называется *оцениванием* (evaluation) популяции:

$$\hat{J}(\mathcal{P}) := (\hat{J}(\theta_i) \mid i \in \{1, 2 \dots N\})$$

Определение 24: Процедурой *отбора* (selection) называется выбор (возможно, случайный, возможно, с повторами) M особей из популяции. Формально, это распределение $\text{select}(\mathcal{P}^+ \mid \mathcal{P}, \hat{J}(\mathcal{P}))$, такое что $\forall \theta \in \mathcal{P}^+: \theta \in \mathcal{P}$ с вероятностью 1.

Определение 25: Жадный (greedy) отбор select_M^{top} — выбор топ- M самых приспособленных особей.

Жадный отбор плох тем, что у нас нет гарантий, что мы на самом деле выбираем наилучшую точку из рассмотренных — наши оценки \hat{J} могут быть неточные, и наилучшей на самом деле может оказаться

¹с оговоркой, что метод сможет понять, что нашёл оптимум, для чего придётся предположить некоторые условия регулярности для $J(\theta)$; понятно, что функцию «иголка в стоге сена» (needle in a haystack)

$$J(\theta) = \begin{cases} 0 & \theta \neq \theta^* \\ 1 & \theta = \theta^* \end{cases}$$

никакой метод оптимизации в $\Theta \equiv \mathbb{R}$ не прооптимизирует.

особь с не самой высокой приспособленностью. В частности поэтому могут понадобиться альтернативы жадного отбора.

Пример 26 — Пропорциональный отбор: Зададимся некоторым распределением на особях популяции, которые сэмплирует особь тем чаще, тем выше её приспособленность. Например:

$$p(\theta) \propto \exp(\hat{J}(\theta))$$

Для отбора M особей засэмплируем (обычно с возвращением) из этого распределения M раз.

Пример 27 — Турнирный отбор: Для отбора M особей M раз повторяется следующая процедура: случайно выбираются K особей популяции (из равномерного распределения) и отбирается та из них, чья приспособленность выше. Число K называется *размером турнира* и регулирует вероятность плохо приспособленной особи быть отобранный.

Одна и та же особь в ходе отбора может быть выбрана несколько раз: это можно читать как «особи повезло оставить больше потомства»².

Итак, случайный поиск можно переформулировать на языке мета-эвристик так: сгенерировать из данной стратегии перебора популяцию заданного размера N и отобрать из неё 1 особь жадно.

2.1.3. Hill Climbing

Процедура отбора позволяет только «сокращать» разнообразие популяции. Хочется как-то обусловить процесс генерации новых кандидатов на уже имеющуюся информацию (которая состоит только из особей и их приспособленностей). Мотивация введения мутаций в том, что даже в сложных пространствах Θ зачастую можно что-то «поделать» с точкой θ так, чтобы она превратилась в другую точку $\hat{\theta}$.

Определение 26: *Мутацией* (mutation) называется распределение $m(\hat{\theta} | \theta)$, где θ называется *родителем* (parent), $\hat{\theta}$ — *потомком* (child).

Пример 28: Пусть Θ — множество путей обхода вершин некоторого графа. Такое пространство аргументов возникает во многих комбинаторных задачах (таких как *задача коммивояжёра*). Пусть $\theta \in \Theta$ — некоторый путь обхода, то есть упорядоченное множество вершин графа. Мутацией может выступать выбор случайных двух вершин и смена их местами в порядке обхода (например, обходили пять вершин графа в порядке $(4, 3, 1, 5, 2)$, а после мутации — в порядке $(4, 2, 1, 5, 3)$).

Рассмотрим простейший способ использования мутации. На k -ом шаге алгоритма будем генерировать N потомков особи θ_k при помощи мутации и отбирать из них жадно особь θ_{k+1} . Очень похоже на градиентный подъём: мы сэмплим несколько точек вокруг себя и идём туда, где значение функции максимально. Поэтому отчасти можно считать, что Hill Climbing с большим N — локальная оптимизация: мы не можем взять наилучшее направление изменения θ из, например, градиентов, но можем поискать хорошее направление, условно, случайным перебором.

Пример 29 — Hill Climbing:

Что получается: если мутация такова, что $\forall \theta, \hat{\theta}: m(\hat{\theta} | \theta) > 0$, остаются гарантии оказаться в любой точке пространства, и алгоритм остаётся методом глобальной оптимизации. При этом, мутация может

²С точки зрения эволюционной теории, критерием оптимизации для особи является исключительно преумножение количества своих генов за счёт размножения. Отбор — не столько про «выживание сильнейших», сколько про количество успешных передач генов потомкам.

быть устроена так, что вероятность оказаться «неподалёку» от родителя выше, чем в остальной области пространства.

Пример 30: В предыдущем примере 28 мутация не удовлетворяла свойству $\forall \theta, \hat{\theta}: m(\hat{\theta} | \theta) > 0$: из текущего пути обхода мы могли получить только очень похожий. Применение мета-эвристик к такой мутации чревато скорым застреванием в локальном оптимуме: мы можем попасть в точку, применение мутации к которой с вероятностью один даёт ещё менее приспособленные особи. Чтобы полечить это, создадим другую мутацию: засэмплируем натуральное n из распределения Пуассона или из $p(n) = \frac{1}{2^n}$ и применим такое количество мутаций вида «сменить две вершины местами». Так мы гарантируем, что с небольшой вероятностью мы сможем мутировать в произвольную точку пространства аргументов.

Понятно, что если мутация генерирует очень непохожие на родителя особи, алгоритм схлопывается примерно в случайный поиск. И понятно, что если мутация, наоборот, с огромной вероятностью генерирует очень близкие к родителю особи, алгоритм, помимо того, что будет сходиться медленно, будет сильно надолго застревать в локальных оптимумах. Возникает trade-off между *использованием* (exploitation) и *исследованием* (exploration): баланс между выбором уже известных хороших точек и поиском новых «вдали»; изучением окрестностей найденных локальных оптимумов и поиском новых.

Пример 31: Если $\Theta \equiv \mathbb{R}^h$, то типичным выбором мутации является $m(\hat{\theta} | \theta) := \mathcal{N}(\theta, \sigma^2 I_{h \times h})$, где $\sigma > 0$ — гиперпараметр, и, чем ближе σ к нулю, тем «ближе» к родителю потомки.



Возникает вопрос: а как подбирать гиперпараметры мета-эвристик, тоже мета-эвристикой? Любопытный ответ — *самоадаптирующиеся параметры* (self-adaptive mutations). Параметры мутации кодируются в пространстве аргументов Θ ; то есть одна из координат каждого $\theta \in \Theta$ и отвечает за, например, дисперсию σ в добавляемом шуме из нормального распределения. Появляется надежда, что мета-эвристика «сама подберёт себе хорошие гиперпараметры».

2.1.4. Имитация отжига

Имитация отжига (simulated annealing) решает «проблему исследования» при помощи более умной процедуры отбора: вероятность выбрать потомка θ'_{k+1} , а не оставаться в родительской точке θ_k , вводится так:

$$\text{select } (\theta_{k+1} = \theta'_{k+1}) := \min \left(1, \exp \frac{\hat{J}(\theta'_{k+1}) - \hat{J}(\theta_k)}{\tau_k} \right)$$

где $\tau_k > 0$ — *температура*, гиперпараметр, зависящий от номера итерации. Иными словами, если новая точка более приспособлена, то мы принимаем новую точку θ'_{k+1} с вероятностью 1; если же новая точка менее приспособлена, мы не выкидываем её, а переходим в неё с некоторой вероятностью. Эта вероятность тем ближе к единице, чем «похожее» значения оракула, и температура регулирует понятие похожести между скалярами относительно масштаба оптимизируемой функции J .

Наша цепочка $\theta_0, \theta_1, \theta_2 \dots$ задаёт марковскую цепь: мы генерируем каждую следующую особь на основе только предыдущей, используя некоторое стохастичное правило перехода. Теория марковских цепей говорит, что может существовать *стационарное распределение* (stationary distribution): распределение, из которого приходят θ_k , при стремлении $k \rightarrow \infty$ всё ближе к некоторому $p(\theta)$, которое определяется лишь нашей функцией переходов и не зависит от инициализации θ_0 .

Теорема 3 — Алгоритм Метрополиса-Гастингса: Пусть в пространстве Θ задано распределение $p(\theta)$ и распределение $q(\hat{\theta} | \theta)$, удовлетворяющее $\forall \hat{\theta}, \theta: q(\hat{\theta} | \theta) > 0$. Пусть строится цепочка $\theta_0, \theta_1, \theta_2 \dots$ по следующему правилу: генерируется $\theta'_{k+1} \sim q(\theta'_{k+1} | \theta_k)$, после чего с вероятностью

$$\min \left(1, \frac{p(\theta'_{k+1})}{p(\theta_k)} \frac{q(\theta_k | \theta'_{k+1})}{q(\theta'_{k+1} | \theta_k)} \right)$$

θ_{k+1} полагается равным θ'_{k+1} , а иначе $\theta_{k+1} := \theta_k$. Тогда для любого θ_0 :

$$\lim_{k \rightarrow \infty} p(\theta_k) = p(\theta)$$

Без доказательства. ■

Утверждение 2: Пусть оракул точный, то есть $\hat{J}(\theta) \equiv J(\theta)$, а мутация удовлетворяет

$$\forall \theta, \hat{\theta}: m(\hat{\theta} | \theta) = m(\theta | \hat{\theta}) > 0.$$

Тогда, если температура τ не зависит от итерации, для любой инициализации θ_0 алгоритм имитации отжига строит марковскую цепь со следующим стационарным распределением:

$$\lim_{k \rightarrow \infty} p(\theta_k) \propto \exp \frac{J(\theta_k)}{\tau}$$

Алгоритм Метрополиса даёт, вообще говоря, «гарантии сходимости» для имитации отжига: то, что через достаточно большое количество итераций мы получим сэмпл из распределения $\exp \frac{J(\theta_k)}{\tau}$, нас, в общем-то, устраивает: если температура достаточно мала, это распределение очень похоже на вырожденное в точке максимального значения оптимизируемой функции. Однако, малая температура уменьшает долю исследований в алгоритме, и тогда алгоритм вырождается в наивный поиск при помощи мутации. Поэтому на практике температуру снижают постепенно; подобный *отжиг* (annealing) часто применяется для увеличения доли исследований в начале работы алгоритма и уменьшения случайных блужданий в конце.

Пример 32 — Имитация отжига:



Если в операторе мутации есть параметр, отвечающий за «силу мутаций», например дисперсия σ гауссовского шума в примере 31, а то есть тоже связанный с балансом исследования-использования, его аналогично можно подвергнуть отжигу: в начале алгоритма его значение выставляется достаточно большим, после чего по некоторому расписанию уменьшается с ходом алгоритма.

2.1.5. Эволюционные алгоритмы

Hill Climbing — эвристика с «одним состоянием»: есть какая-то одна текущая основная особь-кандидат, на основе которой и только которой составляется следующая особь-кандидат. Нам, вообще говоря, на очередном шаге нам доступна вся история проверенных точек. Первый вариант — построить суррогат-приближение $\hat{J}(\theta)$, которую легко можно прооптимизировать и найти так следующего кандидата (к нему относятся, например, алгоритмы на основе гауссовских процессов), но этот вариант не сработает в сложных пространствах Θ . Второй вариант — перейти к «эвристикам с N состояниями», то есть использовать последние N проверенных особей для порождения новых кандидатов.

Определение 27: Эволюционный (evolutionary) алгоритм строит последовательность популяций $\mathcal{P}_1, \mathcal{P}_2, \dots$, на k -ом шаге строя очередное *поколение* (generation) на основе предыдущего.

Практически все мета-эвристики сводятся к эволюционным алгоритмам. При этом заметим, что, пока единственным инструментом «генерации» новых особей выступает мутация, алгоритмы различаются только процедурой отбора. Таким образом, в алгоритме всегда поддерживается текущая популяция из N особей (первая популяция генерируется из некоторой стратегии перебора $q(\theta)$), из них отбираются N особей (отбор может выбирать одну особь несколько раз, поэтому этот шаг нетривиален), и дальше каждая отобранныя особь мутируется. Эвристика *элитизма* (elitism) предлагает некоторые из отобранных особей не мутировать, и оставить для следующей популяции; это позволяет уменьшить шансы популяции «потерять» найденную область хороших значений функции, но увеличивает шансы застrevания в локальном оптимуме. При элитизме для каждой особи хранится её *возраст* (age) — число популяций, через которые особь прошла без мутаций; далее этот возраст влияет на отбор, например, выкидывая все особи старше определённого возраста. Далее будем рассматривать алгоритмы без элитизма.

Придумаем простейший эволюционный алгоритм. Любой алгоритм локальной оптимизации (градиентный спуск или Hill Climbing), результат работы которого зависит от начального приближения $\theta_0 \sim q(\theta)$, можно «заменить» на метод глобальной оптимизации, запустив на каждом условном сервере по «*потоку*» (thread) со своим начальным θ_0 . Время работы алгоритма не изменится (считая, конечно, что сервера работают параллельно), а обнаружение неограниченного числа локальных оптимумов с ненулевым шагом найти любой гарантирует нахождение глобального. Набор из текущих состояний всех потоков можно считать текущей популяцией.

При этом, любая процедура отбора позволит потокам «обмениваться информацией между собой». Для примера рассмотрим распространённую схему **(M, K) -эволюционной стратегии**, в которой процедура отбора заключается в том, чтобы топ- M особей отобрать по K раз каждую (дать каждой особи из топа породить K детей). Иными словами, мы параллельно ведём M Hill Climbing-ов, в каждом из которых для одного шага генерируется K потомков; если число потоков $M = 1$, алгоритм вырождается в обычный Hill Climbing. Порождённые $N = MK$ особей образуют текущую популяцию алгоритма. Среди них отбираются M лучших особей, которые могут как угодно распределиться по потокам. Получится, что некоторые потоки, которые не нашли хороших областей Θ , будут прерваны, а хорошие получат возможность сгенерировать больше потомков и как бы «размножаться» на несколько процессов.

Алгоритм 1: (M, K) -эволюционная стратегия

Дано: оракул $\hat{J}(\theta)$

Гиперпараметры: $m(\hat{\theta} | \theta)$ — мутация, $q(\theta)$ — стратегия перебора, M — число потоков, K — число сэмплов на поток

Инициализируем $\mathcal{P}_0 := (\theta_i \sim q(\theta) \mid i \in \{1, 2, \dots, MK\})$

На k -ом шаге:

1. проводим жадный отбор: $\mathcal{P}_k^+ := \text{select}_M^{top}(\mathcal{P}_k, \hat{J}(\mathcal{P}_k))$
2. размножаем: $\mathcal{P}_{k+1} := (\hat{\theta}_i \sim m(\hat{\theta} | \theta) \mid \theta \in \mathcal{P}_k^+, i \in \{1, 2, \dots, K\})$

Пример 33 — $(4, 5)$ -эволюционная стратегия:

2.1.6. Weight Agnostic Neural Networks (WANN)

Поскольку мета-эвристики — универсальный метод оптимизации, они могут применяться к нейронным сетям. Такой подход даёт такие преимущества, как возможность использовать дискретные веса или недопустимые для градиентной оптимизации функции активации вроде функции Хевисайда.

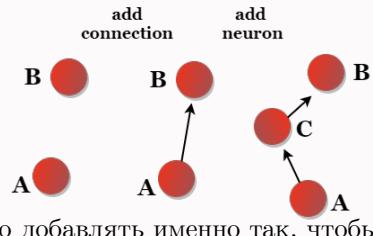
Особенностью **нейроэволюции** (neuroevolution) является возможность искать топологию сети вместе со значениями самих весов: для этого достаточно предложить некоторый оператор мутации. Рассмотрим распространённый пример: пусть дана некоторая нейросеть с произвольной топологией³ и некоторыми весами. Для весов процедуру мутирования можно взять стандартную (добавление шума с заданной дисперсией; дополнительно можно для каждого веса сэмплировать бинарную величину, будет ли данный вес

³на заре нейросетевого подхода разумность использования полно связанных слоёв была под вопросом. Считаем, что нейрон принимает несколько входов, домножает каждый вход на некоторый вес, складывает и применяет функцию активации, выдавая скалярную величину. Выход нейрона отправляется некоторым из других нейронов на вход; понятие «слоёв» обычно не вводится.

мутировать). Далее случайно выбирается, будет ли проходить мутация архитектуры (топологии) сети, и если да, то какого типа (обычно рассматривается несколько типов мутации).

Пример 34 — Топологические мутации нейросети: Распространён набор из трёх видов топологических мутаций: добавление связи, добавление нейрона и смена функции активации.

Добавление связи означает, что случайно выбираются два ранее не соединённых связью нейрона и выход одного подаётся добавляется к входу в другой (вес инициализируется случайно). Какой из двух нейронов является входом, а какой — выходом, однозначно определяется требованием ацикличности к вычислительному графу. Под добавлением нейрона понимается именно разбиение уже имеющейся связи: имевшаяся связь A–B, где A, B — нейроны, выключается, и появляются связи A–C и C–B, где C — новый нейрон. Новые нейроны необходимо добавлять именно так, чтобы они сразу же участвовали в вычислительном процессе. Смена функции активации меняет функцию активацию в случайном нейроне на произвольную из набора.



Заметим, что операторе мутации из примера 34 отсутствуют шансы на уменьшение числа связей. Эволюционный процесс с таким оператором мутации будет рассматривать пространство Θ нейросетей с различными топологиями от «более простых» архитектур к «более сложным». В частности поэтому рекомендуется изначально алгоритм инициализировать «пустой» топологией: между входами и выходами связей нет, а появляться они будут в ходе постепенных мутаций. Похоже, что к такой эвристике привёл эмпирический опыт, и введение мутаций, удаляющих часть архитектуры, не приводило к особым успехам; результатом работы нейроэволюционного алгоритма типично является крайне минимальная по современным меркам нейросеть, с довольно малым количеством связей.

Weight Agnostic сети зашли ещё дальше и, по сути, отказались от настройки весов нейросети в принципе, ограничившись только поиском топологии. Алгоритм WANN основан на (M, K) -эволюционной стратегии с оператором мутации из рассмотренного примера 34. Единственным изменением является процедура отбора. Для данной топологии оракул запускается шесть раз. В каждом запуске всем весам сети присваивается одно и то же значение (авторы использовали значения $[-2, -1, -0.5, 0.5, 1, 2]$). Рассматривается три критерия приспособленности особи: среднее из шести значение \hat{J} , максимальное из шести значение \hat{J} и число связей. Эти три критерия не смешиваются со скалярными гиперпараметрами, вместо этого проводится турнирный отбор (пример 27): из популяции выбираются два кандидата, выбирается случайный критерий, и выживает сильнейший по данному критерию. Так отбираются M особей, которые и порождают K потомков каждый.

2.1.7. Видовая специализация

Возникает простой вопрос к (M, K) -эволюционной стратегии: а не случится ли такого, что он схлопнется к Hill Climbing-у, если в очередной популяции среди топ- M останутся только дети одного и того же родителя? Получится, что, хоть мы и исходили из идеи исследовать параллельно много локальных оптимумов, жадный отбор может убить все потоки, кроме одного, и «область пространства аргументов», покрываемая текущей популяцией, «схлопнется».

Для борьбы с этим эффектом в мета-эвристиках рассматривают методы *защиты инноваций* (innovation protection). Если для получения новых хороших свойств необходимо сделать «несколько» шагов эволюции, то мы каким-то образом помогаем выживать особям, оказавшимся в не исследующихся местах пространства Θ . Самый простой способ — использование более «мягких» процедур отбора, когда у неприспособленной особи есть небольшой шанс выжить. Более интеллектуально было бы как-то оценить, насколько «новой» является область пространства аргументов, в которой оказалась особь, и дать ей больше шансов выжить, если алгоритм эту область ещё не рассматривал: ведь проблема мягких процедур отбора, очевидно, в том, что выживают слабые особи и там, где функция уже была в достаточной степени исследована.

Рассмотрим общую идею *видов* (species). Допустим, мы сможем в Θ придумать метрику (или хоть сколько-то функцию близости) $\rho(\theta_1, \theta_2)$ и на её основе разбить все особи популяции \mathcal{P} на непересекающиеся множества — «виды». Процесс разбиения, что важно, не обязан удовлетворять каким-то особым свойствам и может быть стохастическим, в том числе чтобы быть дешёвым.

Пример 35 — Процедура разделения на виды: Изначально множество видов пусто. На первом шаге берём очередную особь из \mathcal{P} и в случайном порядке перебираем имеющиеся виды. Для каждого вида сэмплируем одну из ранее отнесённых к нему особей и сравниваем расстояние ρ с порогом-гиперпараметром процесса: меньше порога — относим рассматриваемую особь к этому виду и переходим к следующей особи, больше порога — переходим к следующему виду. Если ни для одного вида проверка не прошла, особь относится к новому виду. Пересчёт видов проводится для каждой популяции заново.

Разделение на виды позволяет делать, например, *explicit fitness sharing*: особи соревнуются только внутри своих видов. Пусть $\widetilde{\mathcal{P}} \subseteq \mathcal{P}$ — вид, а $\hat{J}_{mean}(\widetilde{\mathcal{P}})$ — среднее значение приспособленности в данном виде. Тогда на основе этих средних значений между видами проводится (в некоторой «мягкой» форме — слабые виды должны выживать с достаточно высокой вероятностью) некоторый мягкий отбор; например, виду $\widetilde{\mathcal{P}}$ позволяет сгенерировать потомков пропорционально $\exp \hat{J}_{mean}(\widetilde{\mathcal{P}})$ с учётом того, что в сумме все виды должны породить заданное гиперпараметром число особей. Генерация необходимого числа потомков внутри каждого вида происходит уже, например, стандартным, «агрессивным» образом: отбирается некоторая доля топ-особей, к которым и применяется по несколько раз мутация.

Виды защищены мягким отбором и поэтому заспавнившеся вдали особи, образующие новый вид, будут умирать реже; при этом в скоплениях слабых особей в одном месте пройдёт жёсткий внутривидовой отбор, а сам вид получит не так много «слотов потомства», и число точек сократится.

Пример 36 — Эволюция с видовой специализацией: В данном примере текущая популяция из 20 особей разделяется на виды согласно процедуре из примера 35 с метрикой $\rho(\theta_1, \theta_2) = |\theta_1 - \theta_2|$ и порогом 1. Для видов считается $\hat{J}_{mean}(\widetilde{\mathcal{P}})$ (указан как fit в легенде), после чего при помощи сэмплирования из распределения $\propto \exp \hat{J}_{mean}(\widetilde{\mathcal{P}})$ 20 раз разыгрываются между видами «слоты потомства». Внутри каждого вида жадно отбирается 1 особь, она и порождает разыгранное число потомков. Разбиение на виды проводится для новой полученной популяции заново.

2.1.8. Генетические алгоритмы

До сих пор мы умели создавать новые особи только при помощи мутации. В генетических алгоритмах дополнительно вводится этап *рекомбинации* (recombination), когда новые особи можно строить на основе сразу нескольких особей, как-то «совмещая» свойства тех и других в надежде получить «лучшее от двух миров»; найти хороший оптимум между двумя локальными оптимумами. В ванильной версии генетических алгоритмов у детей по два родителя, хотя можно рассматривать и скрещивание большего числа особей:

Определение 28: *Кроссинговером* (crossover) называется распределение $c(\hat{\theta} | \theta_1, \theta_2)$, где θ_1, θ_2 называются *родителями* (parent), $\hat{\theta}$ — *потомком* (child).

Пример 37: Для $\Theta \equiv \mathbb{R}^d$ или $\Theta \equiv \{0, 1\}^d$ (или их смеси) можно придумать много разных кроссинговеров; генетика подсказывает, что если элементы векторов это «гены», то нужно, например, некоторые гены взять от одного родителя, а другие от другого. Некоторые гены можно *сцеплять* (сцепленные гены должны быть взяты из одного родителя), или вводить порядок на генах (брать от одного родителя «правую» часть, от другого «левую»).

Чтобы сохранить свойство «глобальности» оптимизации, желательно было бы, опять же, чтобы мы могли при помощи такого инструмента порождения оказаться в любой точке пространства, т.е. $\forall \hat{\theta}, \theta_1, \theta_2 \in \Theta: c(\hat{\theta} | \theta_1, \theta_2) > 0$. Однако, для практически любых примеров кроссинговера это не так. Поэтому считается, что это требование НЕ выполняется: процедура рекомбинации, возможно, стохастична, но всегда приводит к точке «между» θ_1 и θ_2 . Это означает, что, используя только кроссинговер, область, покрываемая потомством, будет уже области, покрываемой родителями: теряется исследование. Чтобы полечить это, в генетических алгоритмах всё равно остаётся этап применения мутации.

Алгоритм 2: Генетический поиск

Дано: оракул $\hat{J}(\theta)$

Гиперпараметры: $c(\hat{\theta} | \theta_1, \theta_2)$ — кроссинговер, $m(\hat{\theta} | \theta)$ — мутация, $\text{select}(\mathcal{P}^+ | \mathcal{P}, \hat{J}(\mathcal{P}))$ — процедура отбора, $q(\theta)$ — стратегия перебора, N — размер популяции

Инициализируем $\mathcal{P}_0 := (\theta_i \sim q(\theta) | i \in \{1, 2, \dots, N\})$

На k -ом шаге:

1. проводим отбор: $\mathcal{P}_k^+ \sim \text{select}(\mathcal{P}_k^+ | \mathcal{P}_k, \hat{J}(\mathcal{P}_k))$
2. проводим размножение: $\mathcal{P}_{k+1} := \left(\hat{\theta}_i \sim c(\hat{\theta}_i | \theta_{2i-1}, \theta_{2i}) | i \in \{1, 2, \dots, N\} \right)$
3. проводим мутирование: $\mathcal{P}_{k+1} \leftarrow \left(\hat{\theta} \sim m(\hat{\theta} | \theta) | \theta \in \mathcal{P}_{k+1} \right)$

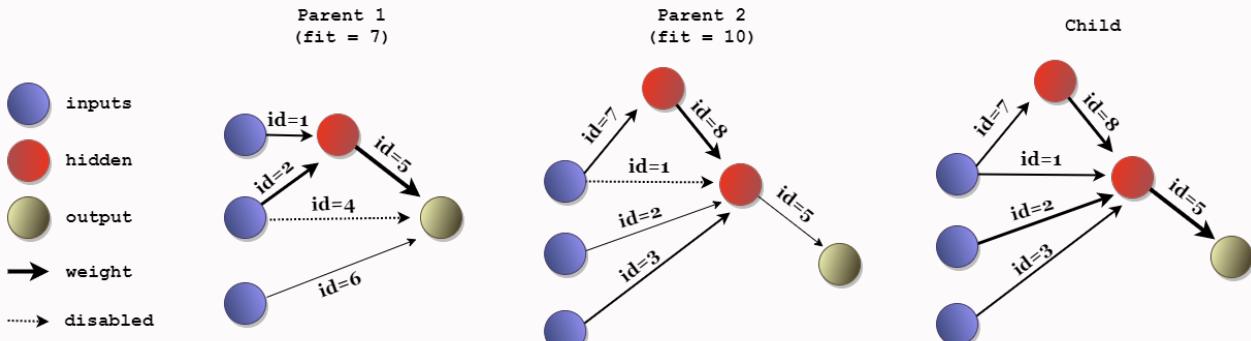


При эволюционном обучении нейросетей, в отличие от ряда других задач, кроссинговер во многом неудобен. Нельзя взять «половинку» одной хорошей нейросети и присоединить к «половинке» другой хорошей нейросети — каждый нейрон рассчитывает на тот набор входов, для которого он был обучен (неважно, эволюционно или градиентно). Поэтому генетические алгоритмы для нас не представляют особого интереса, по крайней мере на момент написания данного текста.

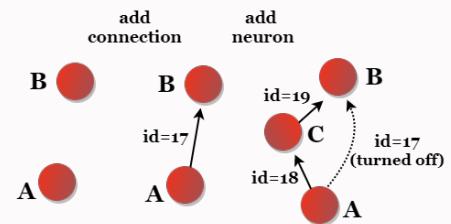
Пример 38 — Neuroevolution of Augmented Topologies (NEAT): Рассмотрим в качестве примера набор эвристик нейроэволюционного алгоритма NEAT, который всё-таки был основан на генетическом поиске (алг. 2), то есть дополнительно вводил оператор кроссинговера для нейросетей (двух произвольных топологий).

В NEAT все связи всех особей, помимо веса, имеют статус «включена-выключена» и уникальный идентификационный номер (id), или *исторический маркер* (historical marker). Связи могут появляться только в ходе мутаций (используется оператор мутации из примера 34); в момент создания связи ей присваивается уникальный (в рамках всего алгоритма) id и статус «включена». Наличие у двух особей связи с одним id будет означать наличие общего предка. У изначальной «пустой» топологии связей нет вообще. Связь может попасть в статус «выключена» только во время мутации вида «добавление нейрона», когда имевшаяся связь A–B «исчезает»: то есть, соответствующий «ген» не удаляется из генома особи, а переходит в статус «выключена». Выключенная связь означает, что у особи был предок, у которого связь была включена.

Как введение статусов и id-шников связей позволяет устраивать между разными топологиями кроссинговер? Если связь с данным id имеется у обоих скрещиваемых особей, связь сохраняется и у потомка, с весом и статусом случайного родителя. Связи, имеющиеся только у одного из родителей, будем называть *непарными*; они копируются из того родителя, чья приспособленность выше (или из обоих сразу, если приспособленности одинаковые).



NEAT также использует видовую специализацию (как описано в разделе 2.1.7) для процесса отбора, для чего на таких генотипах необходимо задать метрику ρ . Пусть θ_1, θ_2 — две особи, G_1, G_2 — количество связей у этих особей, D — число непарных связей, w_1, w_2 — веса особей в парных связях. Понятно, что веса мы можем сравнить только для парных связей, и понятно, что чем больше непарных связей,



тем больше должно быть расстояние. В NEAT предлагается просто объединить эти два критерия:

$$\rho(\theta_1, \theta_2) := \alpha_1 \frac{D}{\max(G_1, G_2)} + \alpha_2 \|w_1 - w_2\|_1$$

где α_1, α_2 — гиперпараметры. Внутри самих видов отбор жадный.

NEAT — исторически один из первых алгоритмов, которые можно с каким-то результатом применить для RL задач без подготовленного удобного признакового описания состояний, например, изображений. Можно найти много интересных примеров применения алгоритма к простейшим задачам, с визуализацией получающейся сети (например, *Mario*).

§2.2. Эволюционные стратегии

2.2.1. Идея эволюционных стратегий

Когда мы пытаемся генерировать $k+1$ -ое поколение, используя особей k -го поколения в качестве исходного материала, единственная зависимость от всей истории заложена в составе k -ой популяции. Мы можем рассмотреть распределение, из которого появляются особи очередной популяции: весь смысл нашей процедуры отбора в том, чтобы это распределение для очередной итерации поменялось так, что вероятность появления более хороших особей стала выше. Давайте обобщим эту идею: будем в явном виде хранить распределение для порождения особей новой популяции и по информации со всей популяции аккумулировать всю информацию внутри его параметров — «скрещивать все особи».

Определение 29: Распределение $q(\theta | \lambda_k)$, из которого генерируются особи k -ой популяции, называется **эволюционной стратегией** (evolutionary strategy):

$$\mathcal{P}_k := \{\theta_i \sim q(\theta | \lambda_k) \mid i \in \{1, 2 \dots N\}\}$$

где λ_k — параметры эволюционной стратегии.

Из каких соображений подбирать λ_k на очередном шаге? В принципе, мы хотели бы найти такой генератор особей, что их оценки как можно больше, то есть напечатать область Θ с высоким значением $J(\theta)$. Эти соображения можно формализовать довольно по-разному и таким образом оправдывать разные мета-эвристики. В частности, мы можем сказать, что λ есть особь или несколько особей (что приведёт нас к примерно ранее рассматривавшимся алгоритмам⁴), но мы можем отойти от пространства Θ и учить модель-генератор особей с какой-то хорошей параметризацией λ . Мы дальше рассмотрим две основные идеи, как это можно делать.

2.2.2. Оценка вероятности редкого события

Допустим, стоит задача оценки вероятности редкого события:

$$l = \mathbf{P}(f(x) \geq \gamma) = \mathbb{E}_{x \sim p(x)} \mathbb{I}[f(x) \geq \gamma] - ? \quad (2.1)$$

где $p(x)$ — некоторое распределение, $f: X \rightarrow \mathbb{R}$ — функционал, γ — некоторый порог.

Под словами «редкое событие» подразумевается, что выражение в индикаторе не равно нулю с вероятностью, крайне близкой к нулю. Это означает, что Монте-Карло оценка с разумным числом сэмплов N выдаст или ноль или $\frac{1}{N}$, если один раз повезёт; короче, лобовой подход не годится.

Хочется сэмплировать x не из того распределения, которое нам дали — $p(x)$, — а из чего-нибудь получше. Для этого применим *importance sampling* с некоторым распределением $q(x)$, которое мы будем выбирать сами:

$$l = \mathbb{E}_{x \sim q(x)} \frac{p(x)}{q(x)} \mathbb{I}[f(x) \geq \gamma] \quad (2.2)$$

Нам хочется выбрать такое $q(x)$, чтобы дисперсия Монте-Карло оценки такого интеграла была как можно меньше. Желание может быть исполнено:

Утверждение 3: Дисперсия Монте-Карло оценки (2.2) минимальна при

$$q(x) \propto p(x) \mathbb{I}[f(x) \geq \gamma] \quad (2.3)$$

Доказательство. Искомое значение l (2.1) является нормировочной константой такого распреде-

⁴ понятие эволюционных стратегий довольно общее и размытое — любой эволюционный алгоритм «неявно» определяет распределение для порождения особей очередной популяции и формально подпадает под эволюционные стратегии.

ленияя. Подставим данное $q(x)$ в подынтегральную функцию:

$$\frac{p(x)}{q(x)} \mathbb{I}[f(x) \geq \gamma] = l$$

Поскольку всё сократилось, для любых сэмплов $x \sim q(x)$ значение Монте-Карло оценки будет равно l ; то есть, дисперсия равна нулю. ■

Посчитать такое $q(x)$ мы не можем, однако сможем приблизить в параметрическом семействе $q(x | \lambda)$, минимизируя KL-дивергенцию:

$$\text{KL}(q(x) \| q(x | \lambda)) = \text{const}(\lambda) - \mathbb{E}_{q(x)} \log q(x | \lambda) \rightarrow \min_{\lambda}$$

Пока что мы променяли шило на мыло, поскольку для такой оптимизации всё равно нужно уметь сэмплировать из $q(x)$. Однако от задачи оценки числа (которую мы кроме как через Монте-Карло особо решать не умеем) мы перешли к поиску распределения. Поскольку это распределение мы строили так, чтобы оно помогало сэмплировать нам точки из редкого события, можно воспользоваться им же с прошлой итерации, чтобы помочь себе решать ту же задачу лучше. То есть: строим последовательность $q(x | \lambda_k)$, λ_0 — любое, на очередной итерации:

$$\lambda_{k+1} = \operatorname{argmin}_{\lambda} -\mathbb{E}_{q(x)} \log q(x | \lambda) =$$

$$\{\text{подставляем вид оптимального } q(x) \text{ из (2.3)}\} = \operatorname{argmin}_{\lambda} -\mathbb{E}_{p(x)} \mathbb{I}[f(x) \geq \gamma] \log q(x | \lambda) =$$

$$\{\text{importance sampling через } q(x | \lambda_k)\} = \operatorname{argmin}_{\lambda} -\mathbb{E}_{q(x | \lambda_k)} \frac{p(x)}{q(x | \lambda_k)} \mathbb{I}[f(x) \geq \gamma] \log q(x | \lambda)$$

Каждая задача нахождения λ_k всё ещё тяжела в связи с тем, что подынтегральное выражение всё ещё почти всегда ноль. Ключевая идея: поскольку мы теперь строим целую последовательность, мы можем поначалу решать сильно более простую задачу, разогревая γ . Будем на k -ом шаге брать γ не из условия задачи, а поменьше, так, чтобы с итерациями γ увеличивалась (и мы решали бы задачу всё более похожую на ту, что требовалось решить исходно), и одновременно достаточное число сэмплов значения подынтегральной функции были отличны от нуля.

Важно, что мы можем не задавать заранее последовательность γ_k , а определять очередное значение прямо на ходу, например, исходя из сэмплов $x_1 \dots x_N \sim q(x | \lambda_k)$ и значений $f(x)$ в них.

Алгоритм 3: Метод Кросс-Энтропии для оценки вероятности редкого события

Вход: распределение $p(x)$, функция $f(x)$, порог γ

Гиперпараметры: $q(x | \lambda)$ — параметрическое семейство, N — число сэмплов, M — порог отбора

Инициализируем λ_0 произвольно.

На k -ом шаге:

1. сэмплируем $x_1 \dots x_N \sim q(x | \lambda_k)$
2. сортируем значения $f(x_i)$: $f_{(1)} \leq f_{(2)} \leq \dots \leq f_{(N)}$
3. полагаем $\gamma_k := \min(\gamma, f_{(M)})$
4. решаем задачу оптимизации:

$$\lambda_{k+1} \leftarrow \operatorname{argmax}_{\lambda} \frac{1}{N} \sum_{j=1}^N \mathbb{I}[f(x_j) \geq \gamma_k] \frac{p(x_j)}{q(x_j | \lambda_k)} \log q(x_j | \lambda)$$

5. **критерий останова:** $\gamma_k = \gamma$

Получение итоговой оценки:

1. сэмплируем $x_1 \dots x_N \sim q(x | \lambda_k)$

2. возвращаем

$$l \approx \frac{1}{N} \sum_{j=1}^N \mathbb{I}[f(x_j) \geq \gamma_k] \frac{p(x_j)}{q(x_j | \lambda_k)}$$

2.2.3. Метод Кросс-Энтропии для стохастической оптимизации

Ну, в рассуждении было видно, что мы практически учим $q(\boldsymbol{x} \mid \boldsymbol{\lambda})$ нащупывать область с высоким значением заданной функции без использования какой-либо информации о ней. Поэтому мы можем адаптировать метод, чтобы он стал мета-эвристикой. Для этого вернёмся к нашей задаче безградиентной оптимизации:

$$J(\boldsymbol{\theta}) \rightarrow \max_{\boldsymbol{\theta}}$$

и перепишем алгоритм 3 в условиях, когда порог γ «не ограничен», ну или что тоже самое, $\gamma = \max_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$. Мы получим первый способ обучения эволюционной стратегии.

Алгоритм 4: Метод Кросс-Энтропии для оптимизации с оракулом нулевого порядка

Дано: оракул $\hat{J}(\boldsymbol{\theta})$

Гиперпараметры: $q(\boldsymbol{\theta} \mid \boldsymbol{\lambda})$ — параметрическое семейство, N — число сэмплов, M — порог отбора

Инициализируем $\boldsymbol{\lambda}_0$ произвольно.

На k -ом шаге:

1. сэмплируем $\mathcal{P}_k := (\boldsymbol{\theta}_i \sim q(\boldsymbol{\theta} \mid \boldsymbol{\lambda}_k) \mid i \in \{1, 2, \dots, N\})$
2. проводим отбор $\mathcal{P}_k^+ := \text{select}_M^{top}(\mathcal{P}_k)$
3. решаем задачу оптимизации:

$$\boldsymbol{\lambda}_{k+1} \leftarrow \operatorname{argmax}_{\boldsymbol{\lambda}} \sum_{\boldsymbol{\theta} \in \mathcal{P}_k^+} \log q(\boldsymbol{\theta} \mid \boldsymbol{\lambda})$$

Видно, что мы по сути действуем эволюционно: хотим генерировать при помощи распределения q точки из области, где значение функции велико; из сгенерированных отбираем те, где значение функции было наибольшим и учим методом максимального правдоподобия повторять эти точки.

2.2.4. Метод Кросс-Энтропии для обучения с подкреплением (CEM)

В обучении с подкреплением в кросс-энтропийном методе можно сделать ещё один шаг. В отличие от всех остальных рассматриваемых в этой главе мета-эвристик, мы можем проводить эволюционный отбор не в пространстве возможных стратегий (в пространстве Θ), а в пространстве траекторий. У нас будет одна текущая стратегия, из которой мы сгенерируем несколько траекторий, и в силу стохастичности некоторые из этих траекторий выдадут лучший результат, чем другие. Мы отберём лучшие и будем методом максимального правдоподобия (по сути, имитационным обучением), учиться повторять действия из лучших траекторий.

Алгоритм 5: Cross Entropy Method

Гиперпараметры: $\pi(a \mid s, \boldsymbol{\theta})$ — стратегия с параметрами $\boldsymbol{\theta}$, N — число сэмплов, M — порог отбора

Инициализируем $\boldsymbol{\theta}_0$ произвольно.

На k -ом шаге:

1. сэмплируем N траекторий $\mathcal{T}_1 \dots \mathcal{T}_N$ игр при помощи стратегии $\pi(a \mid s, \boldsymbol{\theta}_k)$
2. считаем кумулятивные награды $R(\mathcal{T}_i)$
3. сортируем значения: $R_{(1)} \leq R_{(2)} \leq \dots \leq R_{(N)}$
4. полагаем $\gamma_k := R_{(M)}$
5. решаем задачу оптимизации:

$$\boldsymbol{\theta}_{k+1} \leftarrow \operatorname{argmax}_{\boldsymbol{\theta}} \frac{1}{N} \sum_{j=1}^N \mathbb{I}[R(\mathcal{T}_j) \geq \gamma_k] \sum_{s, a \in \mathcal{T}_j} \log \pi(a \mid s, \boldsymbol{\theta})$$

2.2.5. Натуральные эволюционные стратегии (NES)

Рассмотрим альтернативный вариант⁵ подбора параметров эволюционной стратегии $q(\theta | \lambda)$. Будем подбирать λ , исходя из следующего функционала:

$$g(\lambda) := \mathbb{E}_{\theta \sim q(\theta | \lambda)} J(\theta) \rightarrow \max_{\lambda} \quad (2.4)$$

Будем оптимизировать этот функционал градиентно по λ . Давайте подробно разберём, как дифференцировать функции подобного вида, поскольку в дальнейшем мы будем активно пользоваться этой техникой.

Теорема 4:

$$\nabla_{\lambda} g(\lambda) = \mathbb{E}_{\theta \sim q(\theta | \lambda)} \nabla_{\lambda} \log q(\theta | \lambda) J(\theta) \quad (2.5)$$

Доказательство.

$$\begin{aligned} \nabla_{\lambda} g(\lambda) &= \nabla_{\lambda} \mathbb{E}_{\theta \sim q(\theta | \lambda)} J(\theta) = \\ &= \{\text{мат.ожидание — это интеграл}\} = \nabla_{\lambda} \int_{\Theta} q(\theta | \lambda) J(\theta) d\theta = \\ &= \{\text{проносим градиент внутрь интеграла}\} = \int_{\Theta} \nabla_{\lambda} q(\theta | \lambda) J(\theta) d\theta = (*) \end{aligned}$$

Теперь мы применим стандартный технический трюк, называемый *log-derivative trick*: мы хотим преобразовать данное выражение к виду мат.ожидания по $q(\theta | \lambda)$ от чего-то. Как мы сейчас увидим, это «что-то» — градиент логарифма правдодобия. Мы воспользуемся следующим тождеством:

$$\nabla_{\lambda} \log q(\theta | \lambda) = \frac{\nabla_{\lambda} q(\theta | \lambda)}{q(\theta | \lambda)} \quad (2.6)$$

Домножим и поделим наше выражение на $q(\theta | \lambda)$, чтобы получить внутри интеграла мат.ожидание:

$$\begin{aligned} (*) &= \int_{\Theta} q(\theta | \lambda) \frac{\nabla_{\lambda} q(\theta | \lambda)}{q(\theta | \lambda)} J(\theta) d\theta = \\ &= \{\text{замечаем градиент логарифма (2.6)}\} = \int_{\Theta} q(\theta | \lambda) \nabla_{\lambda} \log q(\theta | \lambda) J(\theta) d\theta = \\ &= \{\text{выделяем мат.ожидание}\} = \mathbb{E}_{\theta \sim q(\theta | \lambda)} \nabla_{\lambda} \log q(\theta | \lambda) J(\theta) \quad \blacksquare \end{aligned}$$

Итак, есть следующая идея: сгенерируем популяцию \mathcal{P}_k при помощи $q(\theta | \lambda_k)$, после чего воспользуемся особями как сэмплами для несмешённой оценки градиента функционала (2.4), чтобы улучшить параметры λ и впоследствии сгенерировать следующее поколение \mathcal{P}_{k+1} из более хорошего распределения.

2.2.6. OpenAI-ES

Рассмотрим подход на примере OpenAI-ES, где рассматривается обучение нейросети (с фиксированной топологией) с вещественными весами $\Theta \equiv \mathbb{R}^h$, и полагается

$$q(\theta | \lambda) := \mathcal{N}(\lambda, \sigma^2 I_{h \times h}) \quad (2.7)$$

где σ — гиперпараметр, $\lambda \in \mathbb{R}^h$ — по сути, кодирует одну особь, $I_{h \times h}$ — диагональная единичная матрица размера $h \times h$.

Теорема 5: Для эволюционной стратегии (2.7) оценка градиента (2.4) равна

$$\nabla_{\lambda} g(\lambda) = \frac{1}{N\sigma^2} \sum_{\theta \in \mathcal{P}} \hat{J}(\theta)(\theta - \lambda) \quad (2.8)$$

Доказательство.

$$\nabla_{\lambda} \log q(\theta | \lambda) = -\nabla_{\lambda} \frac{(\theta - \lambda)^T (\theta - \lambda)}{2\sigma^2} = \frac{\theta - \lambda}{\sigma^2}$$

⁵Смысл названия «натуральные эволюционные стратегии» (natural evolution strategies) будет объяснён позже.

Достаточно подставить выражение в общую формулу (2.5). ■

Полученная формула легко интерпретируема: мы находимся в некотором «центре» λ , который является нашим «текущим найденным решением». Дальше мы сэмплируем несколько векторов $\theta - \lambda$ из стандартного нормального распределения и складываем эти вектора («скрепляем всех детей») с весами, пропорциональными приспособленности.



Подход работает при большом количестве серверов и ещё одном трюке, позволяющем не обмениваться векторами θ (имеющими размерность, например, по числу параметров нейросети) между процессами. Для этого достаточно зафиксировать random seed на всех процессорах, в каждом процессе генерировать всё поколение, оценивать только особь с соответствующим процессу номеру и обмениваться с другими процессами исключительно оценками \hat{J} (скалярами!). Цель такой процедуры — избежать обмена весами нейросетей (пусть даже не очень больших). За счёт параллелизации удаётся так «обучать» сетки играть в одну игру Atari за 10 минут. Если у вас есть 1440 процессоров. Естественно, ни про какой sample efficiency речь не идёт.

Пример 39 — OpenAI-ES:

Рассмотрим альтернативный взгляд на этот алгоритм. Допустим, мы находимся в точке θ и хотим сдвинуться как бы по градиенту $J(\theta)$, который вычислить мы не можем. Но мы можем приблизить градиент вдоль любого направления ν , например, так:

$$\nabla|_\nu J(\theta) \approx \frac{\hat{J}(\theta + \sigma\nu) - \hat{J}(\theta)}{\sigma}$$

для некоторого небольшого скаляра σ .

Лобовая идея⁶: давайте возьмём N случайных направлений $\nu_1 \dots \nu_N \sim \mathcal{N}(0, I)$ и сделаем шаг по всем этим направлениям:

$$\theta_{k+1} := \theta_k + \alpha \underbrace{\sum_{i=0}^N \frac{\hat{J}(\theta + \sigma\nu_i) - \hat{J}(\theta)}{\sigma} \nu_i}_{\text{приближение градиента}} \quad (2.9)$$

где α — learning rate.

Утверждение 4: Формулы (2.9) и (2.8) эквивалентны.

Доказательство. Поскольку ν сэмплируется из стандартной гауссианы, то в среднем:

$$\mathbb{E}_{\nu \sim \mathcal{N}(0, I)} \frac{\hat{J}(\theta)}{\sigma} \nu = \frac{\hat{J}(\theta)}{\sigma} \mathbb{E}_{\nu \sim \mathcal{N}(0, I)} \nu = 0$$

Следовательно, при достаточно большом числе слагаемых формула (2.9) упростится до

$$\theta_{k+1} := \theta_k + \alpha \sum_{i=0}^N \frac{\hat{J}(\theta + \sigma\nu_i)}{\sigma} \nu_i$$

⁶в алгоритме ARS (Advanced Random Search, хотя такой подход не совсем «случайный поиск») делается ровно это, за тем небольшим исключением, что используется приближение градиента по направлению за два вызова оракула:

$$\nabla|_\nu J(\theta) \approx \frac{\hat{J}(\theta + \sigma\nu) - \hat{J}(\theta - \sigma\nu)}{2\sigma}$$

что совпадает с формулой OpenAI-ES с точностью до замены обозначений: достаточно заметить, что $\theta + \sigma\nu$ есть сэмпл из $\mathcal{N}(\theta, \sigma^2 I)$. ■

2.2.7. Адаптация матрицы ковариации (CMA-ES)

Более глубокомысленно было бы для $\Theta \equiv \mathbb{R}^h$ адаптировать не только среднее, но и матрицу ковариации, которая имеет смысл «разброса» очередной популяции. Covariance Matrix Adaptation Evolution Strategy (CMA-ES) — алгоритм, включающий довольно большой набор эвристик, в основе которого лежит эволюционная стратегия, адаптирующая не только среднее, но и матрицу ковариации:

$$q(\theta | \lambda) := \mathcal{N}(\mu, \Sigma) \quad (2.10)$$

где $\lambda := (\mu, \Sigma)$.



Нам придётся хранить матрицу ковариации размера $\mathbb{R}^{h \times h}$, где h — количество параметров нашей стратегии ($\Theta \equiv \mathbb{R}^h$). Если стратегия задана нейронной сетью, то, чтобы такое было возможно, сеть должна быть по современным меркам минимальнейшей. Однако, во многих задачах непрерывного управления это довольно типичная ситуация, когда на вход в качестве состояния подаётся небольшой (размера 100-200) вектор, а на выходе также ожидается вектор размера порядка 10-30. Тогда стратегия может быть задана полно связной нейросетью всего в несколько слоёв, и хранить Σ теоретически становится возможно.

Мы рассмотрим только основную часть формул алгоритма, касающихся формулы обновления Σ . Посмотрим на формулу для обновления среднего (2.8) (с точностью до learning rate):

$$\mu_{k+1} = \mu_k + \alpha \sum_{\theta \in \mathcal{P}_k} \underbrace{\hat{J}(\theta)}_{\substack{\text{вес} \\ \text{«предлагаемое»}}} \underbrace{(\theta - \mu_k)}_{\substack{\text{особью изменение}}}, \quad (2.11)$$

Для обновления матрицы ковариации будем рассуждать также: каждая особь популяции θ «указывает» на некоторую ковариацию $(\theta - \mu_k)(\theta - \mu_k)^T$ и таким образом «предлагает» следующее изменение:

$$(\theta - \mu_k)(\theta - \mu_k)^T - \Sigma_k,$$

где Σ_k — матрица ковариации на текущей итерации. Усредним эти «предложения изменения» по имеющейся популяции, взвесив их на $\hat{J}(\theta)$, и получим «градиент» для обновления матрицы:

$$\Sigma_{k+1} := \Sigma_k + \alpha \sum_{\theta \in \mathcal{P}_k} \hat{J}(\theta) ((\theta - \mu_k)(\theta - \mu_k)^T - \Sigma_k) \quad (2.12)$$

Здесь нужно оговориться, что мы используем оценку ковариации как бы «методом максимального правдоподобия при условии известного среднего μ_k » (мы знаем, что именно с таким средним генерировались особи прошлой популяции)⁷. Исторически к этим формулам пришли эвристически, но позже у формулы появилось теоретическое обоснование.

Теорема 6: Формулы (2.11) и (2.12) для обновления $\lambda = (\mu, \Sigma)$ являются формулами натурального градиентного спуска для (2.4).

Доказательство требует обсуждения такой большой темы, как натуральный градиентный спуск, а вывод формулы потребует небольшого введения в Кронекерову алгебру, поэтому доказательство вынесено в приложение. ■

Именно поэтому данный вид алгоритмов для обучения эволюционных стратегий называется «**натуральными**» (natural): считается, что функционал (2.4) корректнее оптимизировать именно при помощи натурального градиентного спуска, инвариантного к параметризации $q(\theta | \lambda)$, а не обычного.

Пример 40 — CMA-ES (упрощ.):

⁷ мы бы пришли к немного другим формулам, если бы использовали метод максимального правдоподобия при условии неизвестного среднего (μ_k заменилось бы на μ_{k+1}):

$$(\theta - \mu_{k+1})(\theta - \mu_{k+1})^T - \Sigma_k$$

Если мы попробуем проделать с данным подходом (оптимизацией (2.4)) проделать тот же трюк, что и с кросс-энтропийным методом, и попытаемся считать градиент «в пространстве траекторий», а не в пространстве стратегий, то получим методы оптимизации $J(\theta)$ уже первого порядка — «policy gradient» методы. К ним мы перейдём в главе ??.

ГЛАВА 3

Классическая теория

В данной главе будут доказаны основные теоретические результаты о MDP и получены важные «табличные» алгоритмы, работающие в случае конечных пространств состояний и действий; они лягут в основу всех дальнейших алгоритмов.

§3.1. Оценочные функции

3.1.1. Свойства траекторий

Как это всегда бывает, чем более общую задачу мы пытаемся решать, тем менее эффективный алгоритм мы можем придумать. В RL мы сильно замахиваемся: хотим построить алгоритм, способный обучаться решению «произвольной» задачи, заданной средой с описанной функцией награды. Однако, в формализме MDP в постановке мы на самом деле внесли некоторые ограничения: марковость и стационарность. Эти предположения практически не ограничивают общность нашей задачи с точки зрения здравого смысла с одной стороны и при этом внесут в нашу задачу некоторую «структуру»; и мы сможем придумать более эффективные алгоритмы решения за счёт эксплуатации этой структуры. Что значит «структурой»? Представим, что мы решаем некоторую абстрактную задачу, максимизируя некоторую кумулятивную награду. Вот мы находимся в некотором состоянии и должны выбрать некоторое действие. Интуитивно ясно, что на прошлое — ту награду, которую мы уже успели собрать — мы уже повлиять не можем, и нужно максимизировать награду в будущем. Более того, мы можем отбросить всю нашу предыдущую историю и задуматься лишь над тем, как максимизировать награду с учётом сложившейся ситуации — «текущего состояния».

Пример 41 — Парадокс обжоры: Обжора пришёл в ресторан и заказал кучу-кучу еды. В середине трапезы выяснилось, что оставшиеся десять блюд явно лишние и в него уже не помещаются. Обидно: они будут в счёте, да и не пропадать же еде, поэтому надо бы всё равно всё съесть. Однако, с точки зрения функции награды нужно делать противоположный вывод: блюда будут в счёте в любом случае, вне зависимости от того, будут ли они съедены — это награда за уже совершённое действие, «прошлое», — а вот за переедание может прилететь ещё отрицательной награды. Обжора понимает, что в прошлом совершил неоптимальное действие, и пытается «прооптимизировать» неизбежную награду за прошлое, в результате проигрывая ещё.

Давайте сформулируем эту интуицию формальнее. Как и в обычных Марковских цепях, в средах благодаря марковости действует закон «независимости прошлого и будущего при известном настоящем». Формулируется он так:

Утверждение 5: Независимость прошлого и будущего при известном настоящем Пусть $\mathcal{T}_{:t} := \{s_0, a_0 \dots s_{t-1}, a_{t-1}\}$ — «прошлое», s_t — «настоящее», $\mathcal{T}_{t:} := \{a_t, s_{t+1}, a_{t+1} \dots\}$ — «будущее». Тогда:

$$p(\mathcal{T}_{:t}, \mathcal{T}_{t:} | s_t) = p(\mathcal{T}_{:t} | s_t) p(\mathcal{T}_{t:} | s_t)$$

Доказательство. По правилу произведения:

$$p(\mathcal{T}_{:t}, \mathcal{T}_{t:} | s_t) = p(\mathcal{T}_{:t} | s_t) p(\mathcal{T}_{t:} | s_t, \mathcal{T}_{:t})$$

Однако в силу марковости будущее зависит от настоящего и прошлого только через настоящее:

$$p(\mathcal{T}_{t:} | s_t, \mathcal{T}_{:t}) = p(\mathcal{T}_{t:} | s_t)$$

■

Для нас утверждение означает следующее: если мы сидим в момент времени t в состоянии s и хотим посчитать награду, которую получим в будущем (то есть величину, зависящую только от $\mathcal{T}_{t:}$), то нам совершенно не важна история попадания в s . Это следует из свойства мат. ожиданий по независимым переменным:

$$\mathbb{E}_{\mathcal{T}|s_t=s} R(\mathcal{T}_{t:}) = \{\text{утв. 5}\} = \mathbb{E}_{\mathcal{T}_{:t}|s_t=s} \underbrace{\mathbb{E}_{\mathcal{T}_{t:}|s_t=s} R(\mathcal{T}_{t:})}_{\text{не зависит от } \mathcal{T}_{:t}} = \mathbb{E}_{\mathcal{T}_{t:}|s_t=s} R(\mathcal{T}_{t:})$$

Определение 30: Для траектории \mathcal{T} величина

$$R_t := R(\mathcal{T}_{t:}) = \sum_{t' \geq t} \gamma^{t'-t} r_{t'}$$

называется *reward-to-go* с момента времени t .

Благодаря второму сделанному предположению, о стационарности (в том числе стационарности стратегии агента), получается, что будущее также не зависит от текущего момента времени t : всё определяется исключительно текущим состоянием. Иначе говоря, агенту неважно не только, как он добрался до текущего состояния и сколько награды встретил до настоящего момента, но и сколько шагов в траектории уже прошло. Формально это означает следующее: распределение будущих траекторий имеет в точности тот же вид, что и распределение всей траектории при условии заданного начала.

Утверждение 6: Будущее определено текущим состоянием:

$$p(\mathcal{T}_{t:} | s_t = s) \equiv p(\mathcal{T} | s_0 = s)$$

Доказательство. По определению:

$$p(\mathcal{T}_{t:} | s_t = s) = \prod_{t' \geq t} p(s_{t'+1} | s_{t'}, a_{t'}) \pi(a_{t'} | s_{t'}) = (*)$$

Воспользуемся однородностью MDP и однородностью стратегии, а именно:

$$\begin{aligned} \pi(a_{t'} | s_{t'} = s) &= \pi(a_0 | s_0 = s) \\ p(s_{t'+1} | s_{t'} = s, a_{t'}) &= p(s_1 | s_0 = s, a_0) \\ \pi(a_{t'+1} | s_{t'+1}) &= \pi(a_1 | s_1) \\ p(s_{t'+2} | s_{t'+1}, a_{t'+1}) &= p(s_2 | s_1, a_1) \end{aligned}$$

и так далее, получим:

$$(*) = \prod_{t' \geq 0} p(s_{t'+1} | s_{t'}, a_{t'}) \pi(a_{t'} | s_{t'}) = p(\mathcal{T} | s_0 = s)$$

■

Утверждение 7: Для любого t и любой функции f от траекторий:

$$\mathbb{E}_{\mathcal{T}|s_0=s} f(\mathcal{T}) = \mathbb{E}_{\mathcal{T}|s_t=s} f(\mathcal{T}_{t:})$$

Доказательство.

$$\mathbb{E}_{\mathcal{T}|s_0=s} f(\mathcal{T}) = \{\text{утв. 6}\} = \mathbb{E}_{\mathcal{T}_{:t}|s_t=s} f(\mathcal{T}_{t:}) = \{\text{утв. 5}\} = \mathbb{E}_{\mathcal{T}|s_t=s} f(\mathcal{T}_{t:})$$

■

Мы показали, что все свойства reward-to-go определяются исключительно стартовым состоянием.

3.1.2. V-функция

Итак, наша интуиция заключается в том, что, когда агент приходит в состояние s , прошлое не имеет значения, и оптимальный агент должен максимизировать в том числе и награду, которую он получит, стартуя из состояния s . Поэтому давайте «обобщим» наш оптимизируемый функционал, варьируя стартовое состояние:

Определение 31: Для данного MDP **V-функцией** или Value-функцией (value function) или оценочной функцией состояний (state value function) для данной стратегии π называется величина

$$V^\pi(s) := \mathbb{E}_{\mathcal{T} \sim \pi | s_0 = s} R(\mathcal{T}) \quad (3.1)$$

По определению функция ценности состояния, или V-функция — это сколько набирает в среднем агент из состояния s . Причём в силу марковости и стационарности неважно, случился ли старт на нулевом шаге эпизода или на произвольном t -ом:

Утверждение 8: Для любого t верно:

$$V^\pi(s) = \mathbb{E}_{\mathcal{T} \sim \pi | s_t = s} R_t$$

Пояснение. Применить утверждение 7 для $R(\mathcal{T})$. ■

Утверждение 9: $V^\pi(s)$ ограничено.

Утверждение 10: Для терминальных состояний $V^\pi(s) = 0$.



Любая политика π индуцирует V^π . То есть, для данного MDP и данной стратегии π функция V^π однозначно задана своим определением; совсем другой вопрос, можем ли мы вычислить эту функцию.

Пример 42: Посчитаем V-функцию для MDP и стратегии π с рисунка, $\gamma = 0.8$. Её часто удобно считать «с конца», начиная с состояний, близких к терминальным, и замечая связи между значениями функции для разных состояний.

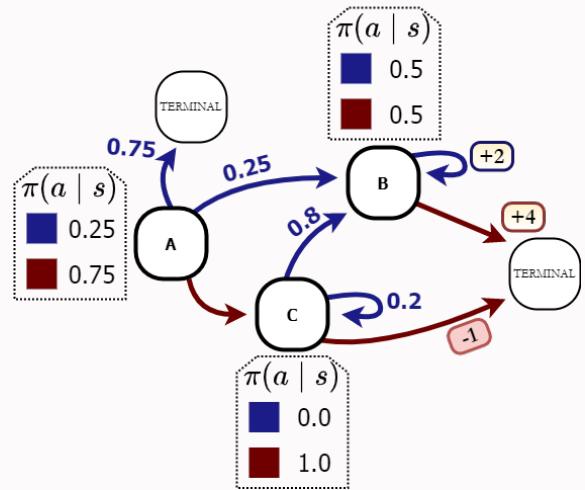
Начнём с состояния С: там агент всегда выбирает действие $\boxed{\text{■}}$, получает -1, и эпизод заканчивается: $V^\pi(s = C) = -1$.

Для состояния В с вероятностью 0.5 агент выбирает действие $\boxed{\text{■}}$ и получает +4. Иначе он получает +2 и возвращается снова в состояние В. Вся дальнейшая награда будет дисконтирована на $\gamma = 0.8$ и тоже равна $V^\pi(s = B)$ по определению. Итого:

$$V^\pi(s = B) = \underbrace{0.5 \cdot 4 + 0.5 \cdot (2 + \gamma V^\pi(s = B))}_{\boxed{\text{■}}} \quad \boxed{\text{■}}$$

Решая это уравнение относительно $V^\pi(s = B)$, получаем ответ 5.

Для состояния А достаточно аналогично рассмотреть все дальнейшие события:



Подставляя значения, получаем ответ $V^\pi(s = A) = -0.35$.

$$V^\pi(s = A) = \underbrace{0.25 \cdot \left(\underbrace{0.75 \cdot 0 + 0.25 \gamma V^\pi(s = B)}_{B} \right)}_{\boxed{\text{■}}} + \underbrace{0.75 \gamma V^\pi(s = C)}_{\boxed{\text{■}}}$$

3.1.3. Уравнения Беллмана

Если s_0 — стартовое состояние, то $V^\pi(s_0)$ по определению есть функционал (1.5), который мы хотим оптимизировать. Формально, это единственная величина, которая нас действительно волнует, так как она нам явно задана в самой постановке задачи, но мы понимаем, что для максимизации $V^\pi(s_0)$ нам нужно промаксимизировать и $V^\pi(s)$ (строго мы это пока не показали). Другими словами, у нас в задаче есть **подзадачи эквивалентной структуры**: возможно, они, например, проще, и мы можем сначала их решить, а дальше как-то воспользоваться этими решениями для решения более сложной. Вот если граф MDP есть дерево, например, то очевидно, как считать V^π : посчитать значение в листьях (терминальные состояния — там ноль), затем в узлах перед листьями, ну и так далее индуктивно добраться до корня.

Мы заметили, что в примере 42 на значения V -функции начали появляться рекурсивные соотношения. В этом и есть смысл введения понятия оценочных функций — «дополнительных переменных»: в том, что эти значения связаны между собой **уравнениями Беллмана** (Bellman equations).

Теорема 7 — Уравнение Беллмана (Bellman expectation equation) для V^π :

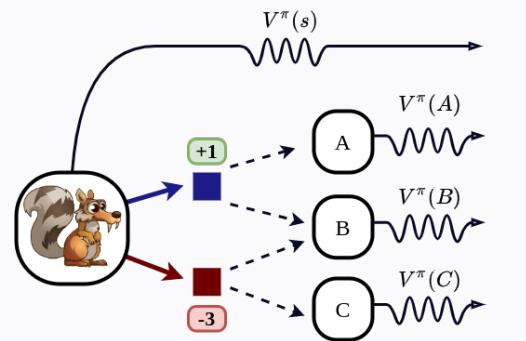
$$V^\pi(s) = \mathbb{E}_a [r(s, a) + \gamma \mathbb{E}_{s'} V^\pi(s')] \quad (3.2)$$

Доказательство. Интуиция: награда за игру равна награде за следующий шаг плюс награда за оставшуюся игру; награда за хвост равна следующей награде плюс награда за хвост. Действительно, для всех траекторий \mathcal{T} и для любых t верно:

$$R_t = r_t + \gamma R_{t+1}$$

Соответственно, для формального доказательства раскладываем сумму по времени как первое слагаемое плюс сумма по времени и пользуемся утверждением 8 о независимости V -функции от времени:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_{\mathcal{T}|s_t=s} R_t = \mathbb{E}_{a_t} [r_t + \gamma \mathbb{E}_{s_{t+1}} \mathbb{E}_{\mathcal{T} \sim \pi|s_{t+1}} R_{t+1}] = \\ &= \mathbb{E}_a [r(s, a) + \gamma \mathbb{E}_{s'} V^\pi(s')] \quad \blacksquare \end{aligned}$$



Пример 43: Выпишем уравнения Беллмана для MDP и стратегии π из примера 42. Число уравнений совпадает с числом состояний. Разберём подробно уравнение для состояния A :

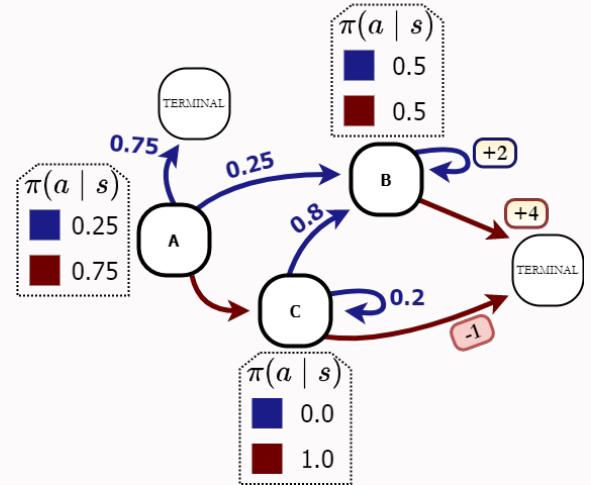
$$V^\pi(A) = \underbrace{0.25(0 + \gamma 0.25 V^\pi(B))}_{\text{Action } A} + \underbrace{0.75(0 + \gamma V^\pi(C))}_{\text{Action } B}$$

С вероятностью 0.25 будет выбрано действие ■, после чего случится дисконтирование на γ ; с вероятностью 0.75 эпизод закончится и будет выдана нулевая награда, с вероятностью 0.25 агент перейдёт в состояние B. Второе слагаемое уравнения будет отвечать выбору действия ■; агент тогда перейдёт в состояние C и, начиная со следующего шага, получит в будущем $V^\pi(C)$. Аналогично расписываются два оставшихся уравнения.

$$V^\pi(A) = \frac{1}{16} \gamma V^\pi(B) + \frac{3}{4} \gamma V^\pi(C)$$

$$V^\pi(B) = 0.5 (2 + \gamma V^\pi(B)) + 0.5 (4 + \gamma V^\pi(C))$$

$$V^\pi(C) = -1$$



Заметим, что мы получили систему из трёх линейных уравнений с тремя неизвестными.

Позже мы покажем, что V^π является единственной функцией $\mathcal{S} \rightarrow \mathbb{R}$, удовлетворяющей уравнениям Беллмана для данного MDP и данной стратегии π , и таким образом однозначно ими задаётся.

3.1.4. Оптимальная стратегия

У нас есть конкретный функционал $J(\pi) = V^\pi(s_0)$, который мы хотим оптимизировать. Казалось бы, понятие оптимальной политики очевидно как вводить:

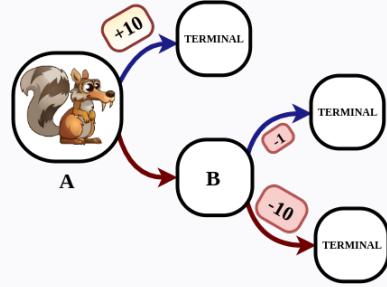
Определение 32: Политика π^* оптимальна, если $\forall \pi: V^{\pi^*}(s_0) \geq V^\pi(s_0)$.

Введём альтернативное определение:

Определение 33: Политика π^* оптимальна, если $\forall \pi, s: V^{\pi^*}(s) \geq V^\pi(s)$.

Теорема 8: Определения не эквивалентны.

Доказательство. Из первого не следует второе (из второго первое, конечно, следует). Контрпример приведён на рисунке. С точки зрения нашего функционала, оптимальной будет стратегия сразу выбрать B и закончить игру. Поскольку оптимальный агент выберет B с вероятностью 0, ему неважно, какое решение он будет принимать в состоянии B , в котором он никогда не окажется. Согласно первому определению, оптимальная политика может действовать в B как угодно. Однако, чтобы быть оптимальной согласно второму определению и в том числе максимизировать $V^\pi(s = B)$, стратегия обязана выбирать в B только действие B . ■



Интуиция подсказывает, что различие между определениями проявляется только в состояниях, которые оптимальный агент будет избегать с вероятностью 1 (позже мы увидим, что так и есть). Задавая оптимальность вторым определением, мы чуть-чуть усложняем задачу, но упрощаем теоретический анализ: если бы мы оставили первое определение, у оптимальных политик могли бы быть разные V-функции (см. пример из последнего доказательства); согласно второму определению, V-функция всех оптимальных политик совпадает.

Определение 34: Оптимальные стратегии будем обозначать π^* , а соответствующую им оптимальную V-функцию — V^* :

$$V^*(s) = \max_{\pi} V^\pi(s) \quad (3.3)$$

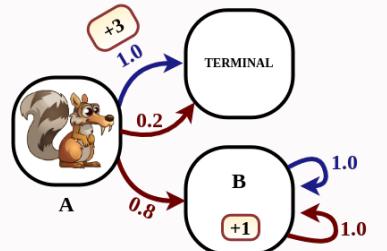
Пока нет никаких обоснований, что найдётся стратегия, которая максимизирует $V^\pi(s)$ сразу для всех состояний. Вдруг в одних s максимум (3.3) достигается на одной стратегии, а в другом — на другой? Тогда оптимальных стратегий в сильном смысле вообще не существует, хотя существует формальная величина (3.3). Пока заметим лишь, что для ситуации, когда MDP — дерево, существование оптимальной стратегии в смысле второго определения можно опять показать «от листьев к корню».

Пример 44: Рассмотрим MDP из примера 9; $\gamma = \frac{10}{11}$, множество стратегий параметризуется единственным числом $\theta := \pi(a = B | s = A)$.

По определению оптимальная V-функция для состояния А равна

$$V^*(s = A) = \max_{\theta \in [0,1]} J(\pi) = \max_{\theta \in [0,1]} [3 + 5\theta] = 8.$$

Оценочные функции для состояния В для всех стратегий совпадают и равны $V^*(s = B) = 1 + \gamma + \gamma^2 + \dots = 11$. Для терминальных состояний $V^*(s) = 0$.



3.1.5. Q-функция

V-функции нам не хватит. Если бы мы знали оптимальную value-функцию $V^*(s)$, мы не смогли бы восстановить хоть какую-то оптимальную политику из-за отсутствия в общем случае информации о динамике среды. Допустим, агент находится в некотором состоянии и знает его ценность $V^*(s)$, а также знает ценности всех других состояний; это не даёт понимания того, какие действия в какие состояния приведут — мы никак не дифференцируем действия между собой. Поэтому мы увеличим количество переменных: введём схожее определение для ценности не состояний, но пар состояния-действие.

Определение 35: Для данного MDP **Q-функцией** (state-action value function, action quality function) для данной стратегии π называется

$$Q^\pi(s, a) := \mathbb{E}_{\tau \sim \pi | s_0 = s, a_0 = a} \sum_{t \geq 0} \gamma^t r_t$$

Теорема 9 — Связь оценочных функций: V-функции и Q-функции взаимозависимы, а именно:

$$Q^\pi(s, a) = r(s, a) + \gamma \mathbb{E}_{s'} V^\pi(s') \quad (3.4)$$

$$V^\pi(s) = \mathbb{E}_a Q^\pi(s, a) \quad (3.5)$$

Доказательство. Следует напрямую из определений.

Итак, если V -функция — это сколько получит агент из некоторого состояния, то Q -функция — это сколько получит агент после выполнения данного действия из данного состояния. Как и V -функция, Q -функция не зависит от времени, ограничена по модулю при рассматриваемых требованиях к MDP, и, аналогично, для неё существует уравнение Беллмана:

Теорема 10 — Уравнение Беллмана (Bellman expectation equation) для Q-функции:

$$Q^\pi(s, a) = r(s, a) + \gamma \mathbb{E}_{s'} \mathbb{E}_{a'} Q^\pi(s', a') \quad (3.6)$$

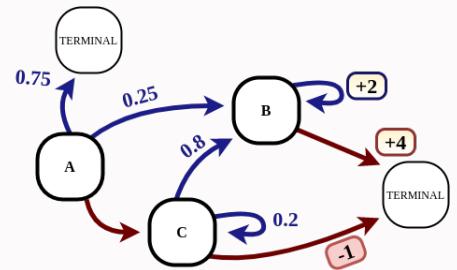
Доказательство. Можно воспользоваться (3.4) + (3.5), можно расписать как награду на следующем шаге плюс хвостик. ■

Пример 45: Q-функция получает на вход пару состояние-действие и ничего не говорится о том, что это действие должно быть как-то связано с оцениваемой стратегией π .

Давайте в MDP с рисунка рассмотрим стратегию π , которая всегда детерминировано выбирает действие \blacksquare . Мы тем не менее можем посчитать $Q^\pi(s, \blacksquare)$ для любых состояний (например, для терминальных это значение формально равно нулю). Сделаем это при помощи QV уравнения:

$$\begin{aligned} Q^\pi(s = A, \square) &= 0.25\gamma V^\pi(s = B) \\ Q^\pi(s = B, \square) &= 2 + \gamma V^\pi(s = B) \\ Q^\pi(s = B, \square) &= 0.8\gamma V^\pi(s = B) + 0.2\gamma V^\pi(s = C) \end{aligned}$$

Внутри V^π сидит дальнейшее поведение при помощи стратегии π , то есть выбор исключительно действий B : соответственно, $V^\pi(s = B) = 4$, $V^\pi(s = C) = -1$.



Мы получили все уравнения Беллмана для оценочных функций (с условными названиями VV , VQ , QV и, конечно же, QQ). Как видно, они следуют напрямую из определений; теперь посмотрим, что можно сказать об оценочных функциях оптимальных стратегий.

3.1.6. Принцип оптимальности Беллмана

Определение 36: Для данного MDP *оптимальной Q-функцией* (optimal Q-function) называется

$$Q^*(s, a) \coloneqq \max_{\pi} Q^\pi(s, a) \quad (3.7)$$

Формально очень хочется сказать, что \mathbf{Q}^* — оценочная функция для оптимальных стратегий, но мы пока никак не связали введённую величину с \mathbf{V}^* и показать это пока не можем. Нам доступно только такое неравенство пока что:

Утверждение 11:

$$Q^*(s, a) \leq r + \gamma \mathbb{E}_{s'} V^*(s')$$

Доказательство.

$$\begin{aligned}
 Q^*(s, a) &= \{\text{определение } Q^* \text{ (3.7)}\} = \max_{\pi} Q^\pi(s, a) = \\
 &= \{\text{связь QV (3.4)}\} = \max_{\pi} [r + \gamma \mathbb{E}_{s'} V^\pi(s')] \leq \\
 &\text{не превосходит среднее максимума } \} \leq r + \gamma \mathbb{E}_{s'} \max_{\pi} V^\pi(s, a) = \\
 &< \{\text{определение } V^* \text{ (3.3)}\} = r + \gamma \mathbb{E}_{s'} V^*(s')
 \end{aligned}$$

Равенство в месте с неравенством случилось бы, если бы мы доказали следующий факт: что вообще существует такая стратегия π , которую мы называем оптимальной, которая, как мы определили,

максимизирует V^π сразу для всех состояний s одновременно. Другими словами, нужно показать, что максимизация $V^\pi(s)$ для одного состояния «помогает» максимизировать награду для других состояний. Но с ней ситуация схожая.

Утверждение 12:

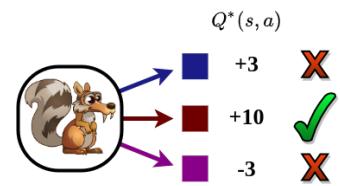
$$V^*(s) \leq \max_a Q^*(s, a)$$

Доказательство.

$$\begin{aligned} V^*(s) &= \{\text{определение } V^* \text{ (3.3)}\} = \max_\pi V^\pi(s) = \\ &= \{\text{связь VQ (3.5)}\} = \max_\pi \mathbb{E}_{a \sim \pi(a|s)} Q^\pi(s, a) \leq \\ &\leq \{\text{по определению } Q^* \text{ (3.7)}\} \leq \max_\pi \mathbb{E}_{a \sim \pi(a|s)} Q^*(s, a) \leq \\ &\leq \{\text{свойство } \mathbb{E}_x f(x) \leq \max_x f(x)\} \leq \max_a Q^*(s, a) \end{aligned}$$

■

Можно ли получить $\max_a Q^*(s, a)$, то есть достигнуть этой верхней оценки? Проведём нестрогое следующее рассуждение: представим, что мы сидим в состоянии s и знаем величины $Q^*(s, a)$, определённые как (3.7). Это значит, что если мы сейчас выберем действие a , то в дальнейшем сможем при помощи какой-то стратегии π , на которой достигается максимум¹ для конкретно данной пары s, a , получить $Q^*(s, a)$. Следовательно, мы, выбрав сейчас то действие a , на которых достигается максимум, в предположении «далнейшей оптимальности своего поведения», надеемся получить из текущего состояния $\max_a Q^*(s, a)$.



Определение 37: Для данного приближения Q-функции стратегия $\pi(s) := \operatorname{argmax}_a Q(s, a)$ называется **жадной** (greedy).

Определение 38: *Принцип оптимальности Беллмана*: жадный выбор действия в предположении оптимальности дальнейшего поведения оптимален.

Догадку несложно доказать для случая, когда MDP является деревом: принятие решения в текущем состоянии s никак не связано с выбором действий в «поддеревьях». В общем случае, однако, нужно показать, что жадный выбор в s «позволит» в будущем набрать то $Q^*(s, a)$, которое мы выбрали — вдруг для того, чтобы получить в будущем $Q^*(s, a)$, нужно будет при попадании в то же состояние s выбирать действие как-то по-другому. Если бы это было так, было бы оптимально искать стратегию в классе нестационарных стратегий.

3.1.7. Отказ от однородности

Утверждение, позволяющее, во-первых, получить вид оптимальной стратегии, а как следствие связать оптимальные оценочные функции, будет доказано двумя способами. В этой секции докажем через отказ от однородности («классическим» способом), затем в секции 3.2 про Relative Performance Identity (RPI) мы поймём, что все желаемые утверждения можно получить и через него (однако, это, похоже, нестандартный способ).

Отказ от однородности заключается в том, что мы в доказательстве будем искать максимум $\max_\pi V^\pi(s)$ не только среди стационарных, но и нестационарных стратегий. Заодно мы убедимся, что достаточно искать стратегию в классе стационарных стратегий. Ранее стационарность означала, что вне зависимости от момента времени наша стратегия зависит только от текущего состояния. Теперь же, для каждого момента времени $t = 0, 1 \dots$ мы запасёмся своей собственной стратегией $\pi_t(a | s)$. Естественно, что теорема 8 о независимости оценочной функции от времени тут перестаёт быть истинной, и, вообще говоря, оценочные функции теперь зависят от текущего момента времени t .

Определение 39: Для данного MDP и нестационарной стратегии $\pi = \{\pi_t(a | s) | t \geq 0\}$ обозначим её **оценочные функции** как

$$V_t^\pi(s) := \mathbb{E}_{\pi_t(a_t | s_t=s)} \mathbb{E}_{p(s_{t+1} | s_t=s, a_t)} \mathbb{E}_{\pi_{t+1}(a_{t+1} | s_{t+1})} \dots R_t$$

$$Q_t^\pi(s, a) := \mathbb{E}_{p(s_{t+1} | s_t=s, a_t=a)} \mathbb{E}_{\pi_{t+1}(a_{t+1} | s_{t+1})} \dots R_t$$

¹ мы знаем, что Q-функция ограничена, и поэтому точно существует супремум. Для полной корректности рассуждений надо говорить об ε -оптимальности, но для простоты мы это опустим.

Пример 46: Действительно, мы можем в состоянии s смотреть на часы, если $t = 0$ — кушать тортики, а если $t = 7$ — бросаться в лаву, т.е. $V_{t=0}^\pi(s) \neq V_{t=7}^\pi(s)$ для неоднородных π .

Утверждение 13: Для нестационарных оценочных функций остаются справедливыми уравнения Беллмана:

$$V_t^\pi(s) = \mathbb{E}_{\pi_t(a|s)} Q_t^\pi(s, a) \quad (3.8)$$

$$Q_t^\pi(s, a) = r(s, a) + \gamma \mathbb{E}_s V_{t+1}^\pi(s') \quad (3.9)$$

Доказательство. Всё ещё следует из определений. ■

Определение 40: Для данного MDP *оптимальными оценочными функциями среди нестационарных стратегий* назовём

$$V_t^*(s) := \max_{\substack{\pi_t \\ \pi_{t+1} \\ \dots}} V_t^\pi(s) \quad (3.10)$$

$$Q_t^*(s, a) := \max_{\substack{\pi_{t+1} \\ \pi_{t+2} \\ \dots}} Q_t^\pi(s, a) \quad (3.11)$$

Заметим, что в определении Q-функции максимум берётся по стратегиям, начиная с π_{t+1} , поскольку по определению Q-функция не зависит от π_t (действие в момент времени t уже «дано» в качестве входа).

Утверждение 14: В стационарных MDP (а мы рассматриваем только их) оптимальные оценочные функции не зависят от времени, т.е. $\forall s, t_1, t_2$ верно:

$$V_{t_1}^*(s) = V_{t_2}^*(s) \quad Q_{t_1}^*(s, a) = Q_{t_2}^*(s, a)$$

Доказательство. Вообще говоря, по построению, так как зависимость от времени заложена исключительно в стратегиях, по которым мы берём максимум (а его мы берём по одним и тем же симплексам вне зависимости от времени):

$$V_t^*(s) = \max_{\substack{\pi_t \\ \pi_{t+1} \\ \dots}} \mathbb{E}_{a_t, s_{t+1}, \dots | s_t=s} R_t = \max_{\substack{\pi_0 \\ \pi_1 \\ \dots}} \mathbb{E}_{a_0, s_1, \dots | s_0=s} R_0 \quad ■$$

Последнее наблюдение само по себе нам ничего не даёт. Вдруг нам в условном MDP с одним состоянием выгодно по очереди выбирать каждое из трёх действий?

3.1.8. Вид оптимальной стратегии (доказательство через отказ от однородности)

Мотивация в отказе от однородности заключается в том, что наше MDP теперь стало деревом: эквивалентно было бы сказать, что мы добавили в описание состояний время t . Теперь мы не оказываемся в одном состоянии несколько раз за эпизод; максимизация $Q_t^*(s, a)$ требует оптимальных выборов «в поддереве», то есть настройки π_{t+1}, π_{t+2} и так далее, а для $\pi_t(a | s)$ будет выгодно выбрать действие жадно. Покажем это формально.

Теорема 11: Стратегия $\pi_t(s) := \operatorname{argmax}_a Q_t^*(s, a)$ оптимальна, то есть для всех состояний s :

$$V_t^\pi(s) = V_t^*(s) = \max_a Q_t^*(s, a) \quad (3.12)$$

Доказательство. В силу VQ уравнения (3.8), максимизация $V_t^\pi(s)$ эквивалентна максимизации

$$V_t^*(s) = \max_{\substack{\pi_t \\ \pi_{t+1} \\ \dots}} V_t^\pi(s) = \max_{\substack{\pi_t \\ \pi_{t+1} \\ \dots}} \mathbb{E}_{\pi_t(a|s)} Q_t^\pi(s, a)$$

Мы уже замечали, что Q_t^π по определению зависит только от $\pi_{t+1}(a | s), \pi_{t+2}(a | s) \dots$. Максимум $Q_t^\pi(s, a)$ по ним по определению (3.11) есть $Q_t^*(s, a)$. Значит,

$$V_t^*(s) \leq \max_{\pi_t} \mathbb{E}_{\pi_t(a|s)} \max_{\substack{\pi_{t+1} \\ \pi_{t+2} \\ \dots}} Q_t^\pi(s, a) = \max_{\pi_t} \mathbb{E}_{\pi_t(a|s)} Q_t^*(s, a)$$

Покажем, что эта верхняя оценка достигается. Найдём π_t такую, что:

$$\begin{cases} \mathbb{E}_{\pi_t(a|s)} Q_t^*(s, a) \rightarrow \max_{\pi_t} \\ \int_{\mathcal{A}} \pi_t(a | s) da = 1; \quad \forall a \in \mathcal{A}: \pi_t(a | s) \geq 0 \end{cases}$$

Решением такой задачи, в частности¹, будет детерминированная стратегия

$$\pi_t^*(s) := \operatorname{argmax}_a Q_t^*(s, a)$$

а сам максимум, соответственно, будет равняться $\max_a Q_t^*(s, a)$. Соответственно, $\max_{\substack{\pi_t \\ \pi_{t+1} \\ \dots}} V_t^\pi(s)$ достигает этой верхней оценки при этой π_t^* и том наборе $\pi_{t+1}^*, \pi_{t+2}^*, \dots$, на котором достигается значение $Q_t^*(s, \pi_t^*(s))$. ■

¹здесь записана просто задача линейного программирования на симплексе (π_t обязано быть распределением); общим решением задачи, соответственно, будет любое распределение, которое размазывает вероятности между элементами множества $\operatorname{Argmax}_a Q_t^*(s, a)$. Но мы пока не знаем, достигается ли значение $Q_t^*(s, a)$ для разных пар s, a на одной и той же стратегии или нет.

Теорема 12: Для нестационарных оценочных функций верно:

$$Q_t^*(s, a) = r(s, a) + \gamma \mathbb{E}_{s'} V_{t+1}^*(s') \quad (3.13)$$

Доказательство. Получим аналогично оценку сверху на $Q_t^*(s, a)$:

$$\begin{aligned} Q_t^*(s, a) &= \max_{\substack{\pi_{t+1} \\ \pi_{t+2} \\ \dots}} Q_t^\pi(s, a) = \{\text{связь QV (3.9)}\} = \max_{\substack{\pi_{t+1} \\ \pi_{t+2} \\ \dots}} [r(s, a) + \gamma \mathbb{E}_{s'} V_{t+1}^\pi(s')] \leq \\ &\leq \{\text{определение } V^* \text{ (3.10)}\} \leq r(s, a) + \gamma \mathbb{E}_{s'} V_{t+1}^*(s') \end{aligned}$$

Эта верхняя оценка достигается на стратегии $\pi(s) = \operatorname{argmax}_a Q_t^*(s, a)$, на которой, как мы доказали в предыдущей теореме 11, достигается максимум $V_t^\pi(s) = V^*(s)$ сразу для всех s одновременно. ■

Таким образом мы показали, что в нестационарном случае наши Q^* и V^* являются оценочными функциями оптимальных стратегий, максимизирующих награду из всех состояний. Осталось вернуться к стационарному случаю, то есть показать, что для стационарных стратегий выполняется то же утверждение.

Утверждение 15: Оптимальные оценочные функции для стационарных и нестационарных случаев совпадают, то есть, например, для V-функции:

$$\max_\pi V^\pi(s) = \max_{\substack{\pi_t \\ \pi_{t+1} \\ \dots}} V_t^\pi(s),$$

где в левой части максимум берётся по стационарным стратегиям, а в правой — по нестационарным.

Доказательство. По теореме 11 максимум справа достигается на детерминированной $\pi_t^*(s) = \operatorname{argmax}_a Q_t^*(a, s)$. В силу утверждения 14, для всех моментов времени Q_t^* совпадают, следовательно π_t^* тоже совпадает для всех моментов времени. ■

Итак, в полученных результатах можно смело заменять все нестационарные оптимальные оценочные функции на стационарные. Интуитивно: мы показали, что об MDP «с циклами в графе» можно думать как о дереве.

3.1.9. Уравнения оптимальности Беллмана

Теорема 13 — Связь оптимальных оценочных функций:

$$V^*(s) = \max_a Q^*(s, a) \quad (3.14)$$

$$Q^*(s, a) = r(s, a) + \gamma \mathbb{E}_{s'} V^*(s') \quad (3.15)$$

Теперь V^* выражено через Q^* и наоборот. Значит, можно получить выражение для V^* через V^* и Q^* через Q^* :

Теорема 14 — Уравнения оптимальности Беллмана (Bellman optimality equation):

$$Q^*(s, a) = r(s, a) + \gamma \mathbb{E}_{s'} \max_{a'} Q^*(s', a') \quad (3.16)$$

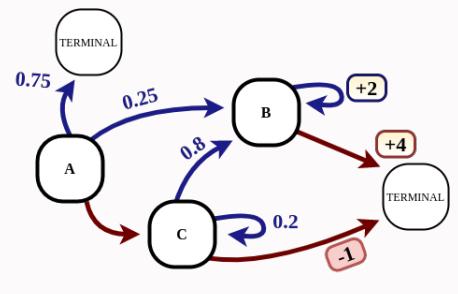
$$V^*(s) = \max_a [r(s, a) + \gamma \mathbb{E}_{s'} V^*(s')] \quad (3.17)$$

Доказательство. Подставили (3.14) в (3.15) и наоборот. ■

Хотя для строгого доказательства нам и пришлось поднапрячься и выписать относительно громоздкое рассуждение, уравнения оптимальности Беллмана крайне интуитивны. Для Q^* , например, можно рассудить так: что даст оптимальное поведение из состояния s после совершения действия a ? Что с нами случится дальше: мы получим награду за этот выбор $r(s, a)$, на что уже повлиять не можем. Остальная награда будет дисконтирована. Затем среда переведёт нас в какое-то следующее состояние s' — нужно проматожидать по функции переходов. После этого мы, пользуясь принципом Беллмана, просто выберем то действие, которое позволит в будущем набрать наибольшую награду, и тогда сможем получить $\max_{a'} Q^*(s', a')$.

Пример 47: Сформулируем для MDP с рисунка уравнения оптимальности Беллмана для V^* . Мы получим систему из трёх уравнений с трёмя неизвестными.

$$\begin{aligned} V^*(s = A) &= \max(0.25\gamma V^*(s = B), \gamma V^*(s = C)) \\ V^*(s = B) &= \max(2 + \gamma V^*(s = B), 4) \\ V^*(s = C) &= \max(0.8\gamma V^*(s = B) + 0.2\gamma V^*(s = C), -1) \end{aligned}$$



Заметим, что в полученных уравнениях не присутствует мат.ожиданий по самим оптимальным стратегиям — предположение дальнейшей оптимальности поведения по сути «заменяет» их на взятие максимума по действиям. Более того, мы позже покажем, что оптимальные оценочные функции — единственные решения систем уравнений Беллмана. А значит, вместо поиска оптимальной стратегии можно искать оптимальные оценочные функции! Таким образом, мы свели задачу оптимизации нашего функционала к решению системы нелинейных уравнений особого вида. Беллман назвал данный подход «*динамическое программирование*» (dynamic programming).

3.1.10. Критерий оптимальности Беллмана

Давайте сформулируем критерий оптимальности стратегий в общей форме, описывающей вид всего множества оптимальных стратегий. Для доказательства нам понадобится факт, который мы технически докажем в рамках повествования чуть позже: для данного MDP Q^* — единственная функция $\mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, удовлетворяющая уравнениям оптимальности Беллмана.

Теорема 15 — Критерий оптимальности Беллмана: π оптимальна тогда и только тогда, когда $\forall s, a: \pi(a | s) > 0$ верно:

$$a \in \operatorname{Argmax}_a Q^\pi(s, a)$$

Необходимость. Пусть π — оптимальна. Тогда её оценочные функции совпадают с V^*, Q^* , для которых выполнено уравнение (3.14):

$$V^\pi(s) = V^*(s) = \max_a Q^*(s, a) = \max_a Q^\pi(s, a)$$

С другой стороны из связи VQ (3.5) верно $V^\pi(s) = \mathbb{E}_{\pi(a|s)} Q^\pi(s, a)$; получаем

$$\mathbb{E}_{\pi(a|s)} Q^\pi(s, a) = \max_a Q^\pi(s, a),$$

из чего вытекает доказываемое. ■

Достаточность. Пусть условие выполнено. Тогда для любой пары s, a :

$$Q^\pi(s, a) = \{\text{связь QQ (3.6)}\} = r(s, a) + \gamma \mathbb{E}_{s'} \mathbb{E}_{\pi(a|s)} Q^\pi(s', a') = r(s, a) + \gamma \mathbb{E}_{s'} \max_{a'} Q^\pi(s', a')$$

Из единственности решения этого уравнения следует $Q^\pi(s, a) = Q^*(s, a)$, и, следовательно, π оптимальна. ■

Иначе говоря: теорема говорит, что оптимальны ровно те стратегии, которые пользуются принципом оптимальности Беллмана. Если в одном состоянии два действия позволят в будущем набрать максимальную награду, то между ними можно любым способом размазать вероятности выбора. Давайте при помощи этого критерия окончательно ответим на вопросы о том, существует ли оптимальная стратегия и сколько их вообще может быть.

Утверждение 16: Если $|\mathcal{A}| < +\infty$, всегда существует оптимальная стратегия.

Доказательство. $\operatorname{Argmax}_a Q^*(s, a)$ для конечных множеств \mathcal{A} всегда существует, следовательно существует детерминированная оптимальная стратегия $\pi(s) := \operatorname{argmax}_a Q^*(s, a)$. ■

Утверждение 17: Оптимальной стратегии может не существовать.

Доказательство. Контрпример: одно состояние, $\mathcal{A} = [-1, 1]$, после первого выбора эпизод заканчивается; в качестве награды $r(a)$ можем рассмотреть любую не достигающую своего максимума функцию. Просто придумали ситуацию, когда $\operatorname{Argmax}_a Q^*(a)$ пуст. ■

Утверждение 18: Если существует хотя бы две различные оптимальные стратегии, то существует континuum оптимальных стратегий.

Доказательство. Две различные оптимальные стратегии означает, что в каком-то состоянии s множество $\operatorname{Argmax}_a Q^*(s, a)$ содержит по крайней мере два элемента. Между ними можно размазать вероятности выбора любым способом и в любом случае получить максимальную награду. ■

Утверждение 19: Если существует хотя бы одна оптимальная стратегия, то существует детерминированная.

Доказательство. Пусть π^* — оптимальна. Значит, $\operatorname{Argmax}_a Q^*(s, a)$ не пуст для всех s , и существует детерминированная оптимальная стратегия $\pi(s) := \operatorname{argmax}_a Q^*(s, a)$. ■

§3.2. Relative Performance Identity

3.2.1. Advantage-функция

Введём ещё одну «оценочную функцию», уже больше из соображений удобства (просто везде вылезает):

Определение 41: Для данного MDP *Advantage-функцией* политики π называется

$$A^\pi(s, a) := Q^\pi(s, a) - V^\pi(s) \quad (3.18)$$

Утверждение 20: Для любой политики π и любого состояния s :

$$\mathbb{E}_{\pi(a|s)} A^\pi(s, a) = 0$$

Доказательство.

$$\begin{aligned} \mathbb{E}_{\pi(a|s)} A^\pi(s, a) &= \mathbb{E}_{\pi(a|s)} Q^\pi(s, a) - \mathbb{E}_{\pi(a|s)} V^\pi(s) = \\ \{V^\pi \text{ не зависит от } a\} &= \mathbb{E}_{\pi(a|s)} Q^\pi(s, a) - V^\pi(s) = \\ \{\text{связь } V \text{ через } Q \text{ (3.5)}\} &= V^\pi(s) - V^\pi(s) = 0 \end{aligned}$$

■

Утверждение 21: Для любой политики π и любого состояния s :

$$\max_a A^\pi(s, a) \geq 0$$

Advantage — это, если угодно, «центрированная» Q-функция. У неё есть важная интуиция: насколько хорошо («удачно») для политики π выбрать в состоянии s действие a ? Мы знаем, что вообще в среднем она набирает из данного состояния $V^\pi(s)$, но какой-то выбор действий даст $Q^\pi(s, a) > V^\pi(s)$, какой-то меньше. Если $A^\pi(s, a) > 0$ — действие a «лучше среднего» для нашей текущей политики в состоянии s , меньше нуля — хуже. И интуиция, что первые надо выбирать чаще, а вторые — реже, нас не обманывает.

3.2.2. Relative Performance Identity (RPI)

Мы сейчас докажем одну очень интересную лемму, которая не так часто нам будет нужна в будущем, но которая прям открыывает глаза на мир. Нам понадобится следующий трюк (*telescoping sums*): для любой последовательности a_t , т. ч. $\lim_{t \rightarrow \infty} a_t = 0$, верно

$$\sum_{t \geq 0}^{\infty} (a_{t+1} - a_t) = -a_0$$

Мы применим его к value-функции следующим образом:

Утверждение 22:

$$-V^\pi(s_0) = \sum_{t \geq 0} [\gamma^{t+1} V^\pi(s_{t+1}) - \gamma^t V^\pi(s_t)] \quad (3.19)$$

Доказательство. Достаточно проверить, что $\lim_{t \rightarrow \infty} \gamma^t V^\pi(s_t) = 0$, что верно в силу ограниченности V-функции в рамках рассматриваемых ограничений на MDP. Если в игре конечное число шагов, не сокращающееся слагаемое $V^\pi(s)$ есть функция ценности терминального состояния, которая по определению равна нулю. ■

Лемма показывает, как связан performance $J(\pi) = V^\pi(s_0)$ двух разных стратегий. В общем виде лемма сравнивает V-функции двух стратегий в одном состоянии:

Теорема 16 — Relative Performance Identity: Для любых двух политик π_1, π_2 :

$$V^{\pi_2}(s) - V^{\pi_1}(s) = \mathbb{E}_{\pi \sim \pi_2 | s_0 = s} \sum_{t \geq 0} \gamma^t A^{\pi_1}(s_t, a_t) \quad (3.20)$$

Доказательство.

$$V^{\pi_2}(s) - V^{\pi_1}(s) = \mathbb{E}_{\pi \sim \pi_2 | s_0 = s} \sum_{t \geq 0} \gamma^t r_t - V^{\pi_1}(s) =$$

$$\begin{aligned}
&= \mathbb{E}_{\mathcal{T} \sim \pi_2 | s_0 = s} \left[\sum_{t \geq 0} \gamma^t r_t - \mathbf{V}^{\pi_1}(s_0) \right] = \\
\{\text{трюк (3.19)}\} &= \mathbb{E}_{\mathcal{T} \sim \pi_2 | s_0 = s} \left[\sum_{t \geq 0} \gamma^t r_t + \sum_{t \geq 0} [\gamma^{t+1} \mathbf{V}^{\pi_1}(s_{t+1}) - \gamma^t \mathbf{V}^{\pi_1}(s_t)] \right] = \\
\{\text{перегруппируем слагаемые}\} &= \mathbb{E}_{\mathcal{T} \sim \pi_2 | s_0 = s} \sum_{t \geq 0} \gamma^t (r_t + \gamma \mathbf{V}^{\pi_1}(s_{t+1}) - \mathbf{V}^{\pi_1}(s_t)) = \\
\{\text{фокус } \mathbb{E}_x f(x) = \mathbb{E}_x \mathbb{E}_x f(x)\} &= \mathbb{E}_{\mathcal{T} \sim \pi_2 | s_0 = s} \sum_{t \geq 0} \gamma^t (r_t + \gamma \mathbb{E}_{s_{t+1}} \mathbf{V}^{\pi_1}(s_{t+1}) - \mathbf{V}^{\pi_1}(s_t)) = \\
\{\text{выделяем Q-функцию (3.4)}\} &= \mathbb{E}_{\mathcal{T} \sim \pi_2 | s_0 = s} \sum_{t \geq 0} \gamma^t (\mathbf{Q}^{\pi_1}(s_t, a_t) - \mathbf{V}^{\pi_1}(s_t)) \\
\{\text{по определению (3.18)}\} &= \mathbb{E}_{\mathcal{T} \sim \pi_2 | s_0 = s} \sum_{t \geq 0} \gamma^t \mathbf{A}^{\pi_1}(s_t, a_t)
\end{aligned}$$

■

Мы смогли записать наш функционал как матожидание по траекториям, сгенерированным одной политикой, по оценочной функции другой стратегии. Грубо говоря, мы можем награду заменить Advantage-функцией произвольной другой стратегии, и это сдвинет оптимизируемый функционал на константу! Прикольно.

3.2.3. Policy Improvement

Определение 42: Будем говорить, что стратегия π_2 «не хуже» π_1 (запись: $\pi_2 \succeq \pi_1$), если $\forall s$:

$$\mathbf{V}^{\pi_2}(s) \geq \mathbf{V}^{\pi_1}(s),$$

и «лучше» (запись: $\pi_2 \succ \pi_1$), если также найдётся s , для которого неравенство выполнено строго:

$$\mathbf{V}^{\pi_2}(s) > \mathbf{V}^{\pi_1}(s)$$

Мы ввели частичный порядок на множестве стратегий (понятно, что можно придумать две стратегии, которые будут «не сравнимы»: когда в одном состоянии одна будет набирать больше второй, в другом состоянии вторая будет набирать больше первой).

Зададимся следующим вопросом. Пусть для стратегии π_1 мы знаем оценочную функцию \mathbf{Q}^{π_1} ; тогда мы знаем и \mathbf{V}^{π_1} из VQ уравнения (3.5) и \mathbf{A}^{π_1} по определению (3.18). Давайте попробуем построить $\pi_2 \succ \pi_1$. Интуицию построения попробуем «подсмотреть» из леммы RPI: мы не очень понимаем, как выбор тех или иных действий в состоянии влияет на структуру траектории $p(\mathcal{T} | \pi_2)$, но можем определить, какие пары (s, a) в принципе могут встречаться в этих траекториях. Что, если π_2 во всех состояниях s выбирает только действия a , на которых $\mathbf{A}^{\pi_1}(s, a) \geq 0$ неотрицателен, то есть что, если в формуле (3.20) под интегралом справа стоят только неотрицательные слагаемые? Такие действия во всех состояниях есть, поскольку Advantage «центрирован» к нулю — утв. 21. Тогда и интеграл в правой части RPI точно будет неотрицателен, каковыми бы ни были траектории, порождаемые π_2 , и тогда $\mathbf{V}^{\pi_2}(s) - \mathbf{V}^{\pi_1}(s) \geq 0$ для всех s .

Покажем более «классическим» способом, что стратегии π_2 достаточно лишь в среднем выбирать действия, дающие неотрицательный Advantage стратегии π_1 , чтобы быть не хуже.

Теорема 17 — Policy Improvement: Пусть стратегии π_1 и π_2 таковы, что для всех состояний s выполняется:

$$\mathbb{E}_{\pi_2(a|s)} \mathbf{Q}^{\pi_1}(s, a) \geq \mathbf{V}^{\pi_1}(s),$$

или, в эквивалентной форме:

$$\mathbb{E}_{\pi_2(a|s)} \mathbf{A}^{\pi_1}(s, a) \geq 0.$$

Тогда $\pi_2 \succeq \pi_1$; если хотя бы для одного s неравенство выполнено строго, то $\pi_2 \succ \pi_1$.

Доказательство. Покажем, что $\mathbf{V}^{\pi_2}(s) \geq \mathbf{V}^{\pi_1}(s)$ для любого s :

$$\begin{aligned}
\mathbf{V}^{\pi_1}(s) &= \{\text{связь VQ (3.5)}\} = \mathbb{E}_{\pi_1(a|s)} \mathbf{Q}^{\pi_1}(s, a) \leq \\
&= \{\text{по построению } \pi_2\} = \mathbb{E}_{\pi_2(a|s)} \mathbf{Q}^{\pi_1}(s, a) = \\
&= \{\text{связь QV (3.4)}\} = \mathbb{E}_{\pi_2(a|s)} [r + \gamma \mathbb{E}_{s'} \mathbf{V}^{\pi_1}(s')] \leq \\
&\leq \{\text{раскручиваем цепочку далее}\} \leq \dots \leq \mathbb{E}_{\mathcal{T} \sim \pi_2 | s_0 = s} \sum_{t \geq 0} \gamma^t r_t =
\end{aligned}$$

$$= \{\text{по определению (3.1)}\} = V^{\pi_2}(s)$$

Если для какого-то s неравенство из условия теоремы было выполнено строго, то для него первое неравенство в этой цепочке рассуждений выполняется строго, и, значит, $V^{\pi_2}(s) > V^{\pi_1}(s)$. ■

Что означает эта теорема? Знание оценочной функции позволяет улучшить стратегию. В частности, мы можем это сделать **жадно**, взяв детерминированную π_2 :

$$\pi_2(s) := \underset{a}{\operatorname{Argmax}} Q^{\pi_1}(s, a) = \underset{a}{\operatorname{Argmax}} A^{\pi_1}(s, a)$$

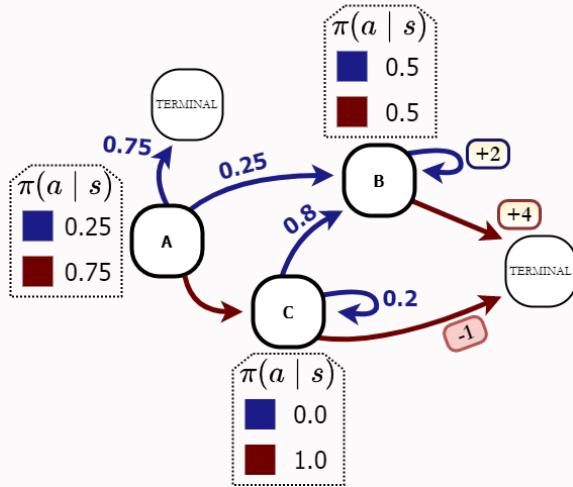
Тогда, какой бы ни была π_1 , в силу утверждения 21, гласящего, что $\max_a A^{\pi_1}(s, a) \geq 0$, мы попадаем под теорему 17 и имеем гарантию $\pi_2 \succeq \pi_1$. Мы так не получим, конечно же, «за один ход» сразу оптимальную стратегию, поскольку выбор $\pi_2(a | s)$ сколько угодно хитро может изменить распределение траекторий, но тем не менее.

Пример 48: Попробуем улучшить стратегию π из примера 42, $\gamma = 0.8$. Например, в состоянии С она выбирает ■ с вероятностью 1 и получает -1; попробуем посчитать $Q^\pi(s = C, ■)$:

$$Q^\pi(s = C, ■) = 0.2\gamma V^\pi(C) + 0.8\gamma V^\pi(B)$$

Подставляя ранее подсчитанные $V^\pi(C) = -1$, $V^\pi(B) = 5$, видим, что действие ■ принесло бы нашей стратегии π куда больше -1, а именно $Q^\pi(s = C, ■) = 3.04$. Давайте построим π_2 , скопировав π в А и В, а в С будем с вероятностью 1 выбирать ■.

Что говорит нам теория? Важно, что она не даёт нам значение $V^{\pi_2}(C)$; в частности, нельзя утверждать, что $Q^{\pi_2}(s = C, ■) = 3.04$, и повторение вычислений подтвердит, что это не так. Однако у нас есть гарантии, что, во-первых, $V^{\pi_2}(C) > V^{\pi_1}(C)$ строго, во-вторых, что мы не «сломали» стратегию в других состояниях: во всех остальных состояниях гарантированно $V^{\pi_2}(s) \geq V^{\pi_1}(s)$. Для Q-функции, как можно показать, выполняется аналогичное неравенство.



Что, если для некоторой π_1 мы «не можем» провести Policy Improvement? Под этим будем понимать, что мы не можем выбрать π_2 так, что $\mathbb{E}_{\pi_2(a|s)} Q^{\pi_1}(s, a) > V^{\pi_1}(s)$ строго хотя бы для одного состояния s (ну, равенства в любом состоянии s мы добьёмся всегда, скопировав $\pi_k(\cdot | s)$). Такое может случиться, если и только если π_1 удовлетворяет следующему свойству:

$$\max_a Q^{\pi_1}(s, a) = V^{\pi_1}(s) \Leftrightarrow \max_a A^{\pi_1}(s, a) = 0$$

Но это в точности критерий оптимальности Беллмана, теорема 15! Причём мы можем, воспользовавшись теоремами RPI 16 и о Policy Improvement 17, теперь доказать этот критерий альтернативным способом, не прибегая к формализму оптимальных оценочных функций². Это доказательство «не совсем канонично», но зато не требует рассуждение про отказ от стационарности и обоснование единственности решения уравнений оптимальности Беллмана.

Теорема 18 — Критерий оптимальности (альт. доказательство): π оптимальна тогда и только тогда, когда $\forall s: \max_a A^\pi(s, a) = 0$.

Достаточность. Допустим, это не так, и существует $\pi_2, s: V^{\pi_2}(s) > V^\pi(s)$. Тогда по RPI (3.20)

$$\mathbb{E}_{\pi_2 \sim \pi_2 | s_0 = s} \sum_{t \geq 0} \gamma^t A^\pi(s_t, a_t) > 0,$$

² в доказательствах RPI и Policy Improvement мы не использовали понятия Q^* и V^* и их свойства; тем не менее, из RPI все свойства оптимальных оценочных функций следуют: например, пусть A^*, Q^*, V^* — оценочные функции оптимальных стратегий, тогда в силу выводимого из RPI критерия оптимальности $\forall s: \max_a A^*(s, a) = 0$, или, что тоже самое, $Q^*(s, a) - V^*(s) = 0$; отсюда $V^*(s) = \max_a Q^*(s, a)$. Аналогично достаточно просто получить все остальные утверждения об оптимальных оценочных функциях, не прибегая к рассуждению с отказом от стационарности.

однако все слагаемые в сумме неположительны. Противоречие. ■

Необходимость. Допустим, что π оптимальна, но для некоторого \hat{s} условие не выполнено, и $\max_a A^\pi(\hat{s}, a) > \mathbf{0}$ (меньше нуля он, ещё раз, быть не может в силу утв. 21). Рассмотрим детерминированную π_2 , которая в состоянии \hat{s} выбирает какое-нибудь \hat{a} , такое что $A^\pi(\hat{s}, \hat{a}) > \mathbf{0}$ (это можно сделать по условию утверждения — сам максимум может вдруг оказаться недостижим для сложных пространств действий, но какое-то действие с положительным advantage-ем мы найдём), а в остальных состояниях выбирает какое-нибудь действие, т.ч. advantage-функция неотрицательна. Тогда

$$V^{\pi_2}(\hat{s}) - V^\pi(\hat{s}) = \mathbb{E}_{\tau \sim \pi_2 | s_0 = \hat{s}} \sum_{t \geq 0} \gamma^t A^\pi(s_t, a_t) > 0$$

поскольку все слагаемые неотрицательны, и во всех траекториях с вероятностью² 1 верно $s_0 = \hat{s}, a_0 = \hat{a}$, то есть первое слагаемое равно $A(\hat{s}, \hat{a}) > 0$. ■

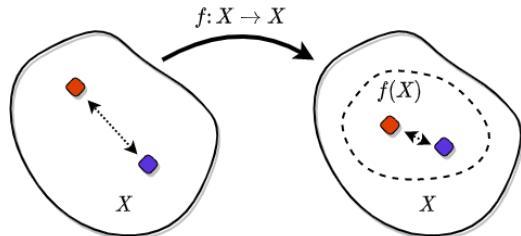
²мы специально стартовали из \hat{s} , чтобы пары \hat{s}, \hat{a} «встретились» в траекториях, иначе могло бы быть такое, что агент в это состояние \hat{s} «никогда не попадает», и отделиться от нуля не получилось бы.

Итак, мораль полученных результатов такая: зная Q^π , мы можем придумать стратегию лучше. Не можем — значит, наша текущая стратегия π уже оптимальна.

§3.3. Динамическое программирование

3.3.1. Метод простой итерации

Хотим научиться решать уравнения Беллмана, но мы видим, что уравнения оптимальности Беллмана нелинейные. Тем не менее, они имеют весьма определённый вид и, как мы сейчас увидим, обладают очень приятными свойствами. Нам понадобится несколько понятий внезапно из функана о том, как решать системы нелинейных уравнений вида $x = f(x)$.



Определение 43: Оператор $f: X \rightarrow X$ 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 44: Точка $x \in X$ для оператора $f: X \rightarrow X$ называется *неподвижной* (fixed point), если

$$x = f(x)$$

Определение 45: 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 19 — Теорема Банаха о неподвижной точке: В полном² метрическом пространстве X у сжимающего оператора $f: X \rightarrow X$ существует и притом ровно одна 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 $k > 0$:

$$\begin{aligned} \rho(x_k, x_{k+1}) &= \{\text{определение } x_k\} = \rho(f(x_{k-1}), f(x_k)) \leq \\ &\leq \{\text{свойство сжатия}\} \leq \gamma \rho(x_{k-1}, x_k) \leq \dots \leq \\ &\leq \{\text{аналогичным образом}\} \leq \dots \leq \gamma^k \rho(x_0, x_1) \end{aligned} \tag{3.21}$$

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

$$\begin{aligned} \rho(x_k, x_{k+n}) &\leq \\ \{\text{неравенство треугольника}\} &\leq \rho(x_k, x_{k+1}) + \rho(x_{k+1}, x_{k+2}) + \dots + \rho(x_{k+n-1}, x_{k+n}) \leq \\ &\leq \{(3.21)\} \leq (\gamma^k + \gamma^{k+1} + \dots + \gamma^{k+n-1}) \rho(x_0, x_1) \leq \\ &\leq \{\text{геом. прогрессия}\} \leq \frac{\gamma^k}{1-\gamma} \rho(x_0, x_1) \xrightarrow{k \rightarrow \infty} 0 \end{aligned}$$

Итак, последовательность \mathbf{x}_k — фундаментальная, и мы специально попросили такое метрическое пространство («полное»), в котором обязательно найдётся предел $\mathbf{x}^* := \lim_{k \rightarrow \infty} \mathbf{x}_k$. ■

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

$$\begin{aligned} \rho(\mathbf{x}^*, f(\mathbf{x}^*)) &\leq \\ &\leq \{\text{неравенство треугольника}\} \leq \rho(\mathbf{x}^*, \mathbf{x}_k) + \rho(\mathbf{x}_k, f(\mathbf{x}^*)) = \\ &= \{\text{определение } \mathbf{x}_k\} = \rho(\mathbf{x}^*, \mathbf{x}_k) + \rho(f(\mathbf{x}_{k-1}), f(\mathbf{x}^*)) \leq \\ &\leq \{\text{свойство сжатия}\} \leq \rho(\mathbf{x}^*, \mathbf{x}_k) + \gamma \rho(\mathbf{x}_{k-1}, \mathbf{x}^*) \end{aligned}$$

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

Единственность. Пусть $\mathbf{x}_1, \mathbf{x}_2$ — две неподвижные точки оператора f . Ну тогда:

$$\rho(\mathbf{x}_1, \mathbf{x}_2) = \rho(f(\mathbf{x}_1), f(\mathbf{x}_2)) \leq \gamma \rho(\mathbf{x}_1, \mathbf{x}_2)$$

Получаем, что такое возможно только при $\rho(\mathbf{x}_1, \mathbf{x}_2) = 0$, то есть только если \mathbf{x}_1 и \mathbf{x}_2 совпадают. ■

²любая фундаментальная последовательность имеет предел

3.3.2. Policy Evaluation

Вернёмся к RL. Известно, что \mathbf{V}^π для данного MDP и фиксированной политики π удовлетворяет уравнению Беллмана (3.2). Для нас это система уравнений относительно значений $\mathbf{V}^\pi(s)$. $\mathbf{V}^\pi(s)$ — объект (точка) в функциональном пространстве $\mathcal{S} \rightarrow \mathbb{R}$.

Будем решать её методом простой итерации³. Для этого определим оператор \mathfrak{B} , то есть преобразование из одной функции $\mathcal{S} \rightarrow \mathbb{R}$ в другую. На вход этот оператор принимает функцию \mathbf{V} и выдаёт некоторую другую функцию от состояний $\mathfrak{B}\mathbf{V}$. Чтобы задать выход оператора, нужно задать значение выходной функции в каждом $s \in \mathcal{S}$; это значение мы будем обозначать $[\mathfrak{B}\mathbf{V}](s)$ (квадратные скобки позволяют не путать применение оператора с вызовом самой функции) и определим его как правую часть решаемого уравнения (3.2). Итак:

Определение 46: Введём *оператор Беллмана* (Bellman operator) как

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

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

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

Теорема 20: Если $\gamma < 1$, оператор \mathfrak{B} — сжимающий с коэффициентом сжатия γ .

Доказательство.

$$\begin{aligned} d_\infty(\mathfrak{B}V_1, \mathfrak{B}V_2) &= \max_s |[\mathfrak{B}V_1](s) - [\mathfrak{B}V_2](s)| = \\ &= \{\text{подставляем значение операторов, т.е. правые части решаемого уравнения}\} = \\ &= \max_s |\mathbb{E}_a [r(s, a) + \gamma \mathbb{E}_{s'} V_1^\pi(s')] - \mathbb{E}_a [r(s, a) + \gamma \mathbb{E}_{s'} V_2^\pi(s')]| = \\ &= \{\text{слагаемые } r(s, a) \text{ сокращаются}\} = \\ &= \gamma \max_s |\mathbb{E}_a \mathbb{E}_{s'} [V_1^\pi(s') - V_2^\pi(s')]| \leq \\ &\leq \{\text{используем свойство } \mathbb{E}_x f(x) \leq \max_x f(x)\} \leq \\ &\leq \gamma \max_s \max_{s'} |V_1^\pi(s') - V_2^\pi(s')| = \gamma d_\infty(V_1, V_2) \end{aligned}$$

■

Итак, мы попали в теорему Банаха, и значит, мы гарантированно сойдёмся к единственной неподвижной точке. По построению мы знаем, что $\mathbf{V}^\pi(s)$ такова, что $\mathfrak{B}\mathbf{V}^\pi = \mathbf{V}^\pi$ (это и есть уравнение Беллмана), поэтому к ней и придём.

³вообще говоря, это система линейных уравнений относительно значений $\mathbf{V}^\pi(s)$, которую в случае табличных MDP можно решать любым методом решения СЛАУ. Однако, дальнейшие рассуждения через метод простой итерации обобщаются, например, на случай непрерывных пространств состояний $\mathcal{S} \subseteq \mathbb{R}^n$.

¹нет, через согласованные нормы стохастических несимметричных матриц, наверное, тоже можно...

Обсудим, что случится в ситуации, когда $\gamma = 1$; напомню, 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, поэтому этот случай придётся разобрать отдельно.

Теорема 21: В эпизодичных средах метод простой итерации сойдётся к единственному решению уравнений Беллмана не более чем за T^{\max} шагов даже при $\gamma = 1$.

Доказательство. Мы уже доказывали теорему 2, что граф таких сред является деревом. Будем говорить, что состояние s находится на ярусе T , если при старте из s у любой стратегии есть гарантии завершения за T шагов. Понятно, что для состояния s на ярусе T верно, что $\forall s', a$, для которых $p(s' | s, a) > 0$, ярус s' не превосходит $T - 1$.

Осталось увидеть, что на k -ой итерации метода простой итерации вычисляет точные значения $V^\pi(s)$ для всех состояний на ярусах до k : действительно, покажем по индукции. Считаем, что терминальные состояния имеют нулевой ярус; а на k -ом шаге при обновлении $V^\pi(s)$ для s на k -ом ярусе в правой части уравнения Беллмана будет стоять мат. ожидание по s' с ярусов до $k - 1$ -го, для которых значение по предположению индукции уже посчитано точно.

Соответственно, за T^{\max} шагов точные значения распространяются на все состояния, и конструктивно значения определены однозначно. ■



Если $\gamma = 1$, а среда неэпизодична (такие MDP мы не допускали к рассмотрению), метод простой итерации может не сойтись, а уравнения Беллмана могут в том числе иметь бесконечно много решений. Пример подобного безобразия. Пусть в MDP без терминальных состояний с нулевой функцией награды (где, очевидно, $V^\pi(s) = 0$ для всех π, s) мы проинициализировали $V^\pi(s) = 100$ во всех состояниях s . Тогда при обновлении наша аппроксимация не будет меняться: мы уже в неподвижной точке уравнений Беллмана. В частности поэтому на практике практически никогда не имеет смысла выставлять $\gamma = 1$, особенно в сложных средах, где, может быть, даже и есть эпизодичность, но, тем не менее, есть «похожие состояния»: они начнут работать «как петли», когда мы перейдём к приближённым методам динамического программирования в дальнейшем.

Утверждение 23: Если некоторая функция $\tilde{V}: \mathcal{S} \rightarrow \mathbb{R}$ удовлетворяет уравнению Беллмана (3.2), то $\tilde{V} \equiv V^\pi$.

Мы научились 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 состояния-действие было возможно хранить в памяти компьютера и перебирать за разумное время. В такой ситуации $V^\pi(s)$ — конечный векторочек, и мы умеем считать оператор Беллмана и делать обновления $V_{k+1} = \mathfrak{B}V_k$.

Алгоритм 6: Policy Evaluation

Вход: $\pi(a | s)$ — стратегия

Гиперпараметры: ε — критерий останова.

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

На k -ом шаге:

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

Выход: $V_k(s)$

Пример 49: Проведём оценивание стратегии, случайно выбирающей, в какую сторону ей пойти, с $\gamma = 0.9$. Угловые клетки с ненулевой наградой терминальны; агент остаётся в той же клетке, если упирается в стенку. На каждой итерации отображается значение текущего приближения $V^\pi(s)$.

Итак, мы научились считать V^π в предположении известной динамики среды. Полностью аналогичное рассуждение верно и для уравнений QQ (3.6); то есть, расширив набор переменных, в табличных MDP можно методом простой итерации находить Q^π и «напрямую». Пока модель динамики среды считается известной, это не принципиально: мы можем посчитать и Q-функцию через V-функцию по формуле QV (3.4).

3.3.3. Value Iteration

Теорема Банаха позволяет аналогично Policy Evaluation (алг. 6) решать уравнения оптимальности Беллмана (3.16) 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 47: Определим *оператор оптимальности Беллмана* (Bellman optimality operator, Bellman control operator) \mathfrak{B}^* :

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

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

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

Нам понадобится следующий факт:

Утверждение 24:

$$|\max_x f(x) - \max_x g(x)| \leq \max_x |f(x) - g(x)| \quad (3.22)$$

Доказательство. Рассмотрим случай $\max_x f(x) > \max_x g(x)$. Пусть x^* — точка максимума $f(x)$. Тогда:

$$\max_x f(x) - \max_x g(x) \leq f(x^*) - \max_x g(x) \leq f(x^*) - g(x^*) \leq \max_x |f(x) - g(x)|$$

Второй случай рассматривается симметрично. ■

Теорема 22: Если $\gamma < 1$, оператор \mathfrak{B}^* — сжимающий.

Доказательство.

$$\begin{aligned} d_\infty(\mathfrak{B}^*Q_1, \mathfrak{B}^*Q_2) &= \max_{s, a} |[\mathfrak{B}^*Q_1](s, a) - [\mathfrak{B}^*Q_2](s, a)| = \\ &= \{\text{подставляем значения операторов, т.е. правые части решаемой системы уравнений}\} = \\ &= \max_{s, a} \left| \left[r(s, a) + \gamma \mathbb{E}_{s'} \max_{a'} Q_1(s', a') \right] - \left[r(s, a) + \gamma \mathbb{E}_{s'} \max_{a'} Q_2(s', a') \right] \right| = \\ &= \{\text{слагаемые } r(s, a) \text{ сокращаются}\} = \\ &= \gamma \max_{s, a} \left| \mathbb{E}_{s'} \left[\max_{a'} Q_1(s', a') - \max_{a'} Q_2(s', a') \right] \right| \leq \\ &\leq \{\text{используем свойство } \mathbb{E}_x f(x) \leq \max_x f(x)\} \leq \\ &\leq \gamma \max_{s, a} \max_{s'} \left| \max_{a'} Q_1(s', a') - \max_{a'} Q_2(s', a') \right| = \\ &\leq \{\text{используем свойство максимумов (3.22)}\} \leq \end{aligned}$$

$$\begin{aligned}
&\leq \gamma \max_{s,a} \max_{s'} \max_{a'} |Q_1(s',a') - Q_2(s',a')| \leq \\
&= \{\text{внутри стоит определение } d_\infty(Q_1, Q_2), \text{ а от внешнего максимума ничего не зависит}\} = \\
&= \gamma d_\infty(Q_1, Q_2)
\end{aligned}$$

Теорема 23: В эпизодичных средах метод простой итерации сойдётся к единственному решению уравнений оптимальности Беллмана не более чем за T^{\max} шагов даже при $\gamma = 1$.

Доказательство. Полностью аналогично доказательству теоремы 21. ■

Утверждение 25: Если некоторая функция $\tilde{Q}: \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ удовлетворяет уравнению оптимальности Беллмана (3.16), то $\tilde{Q} \equiv Q^*$.

Утверждение 26: 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 7: Value Iteration

Вход: ε — критерий останова.

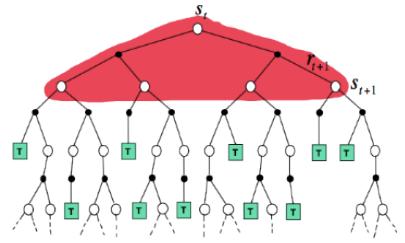
Инициализируем $V_0^*(s)$ произвольно для всех $s \in \mathcal{S}$.

На k -ом шаге:

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

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

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



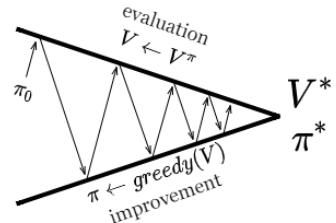
Пример 50: Решим задачу из примера 49, $\gamma = 0.9$; на каждой итерации отображается значение текущего приближения $V^*(s)$. В конце концов в силу детерминированности среды станет понятно, что можно избежать попадания в терминальное -1 и кратчайшим путём добираться до терминального +1.

3.3.4. Policy Iteration

Мы сейчас в некотором смысле «обобщим» Value Iteration и придумаем более общую схему алгоритма планирования для табличного случая.

Для очередной стратегии π_k посчитаем её оценочную функцию Q^{π_k} , а затем воспользуемся теоремой Policy Improvement 17 и построим стратегию лучше; например, жадно:

$$\pi_{k+1}(s) := \operatorname{argmax}_a Q^{\pi_k}(s, a)$$



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:** вычисление Q^π для текущей стратегии π ;
- **Policy Improvement:** улучшение стратегии $\pi(s) \leftarrow \operatorname{argmax}_a Q^\pi(s, a)$;

При этом у нас есть гарантии, что когда алгоритм «останавливается» (не может провести Policy Improvement), то он находит оптимальную стратегию. Будем считать⁴, что в такой момент остановки после проведения Policy Improvement наша стратегия не меняется: $\pi_{k+1} \equiv \pi_k$.

Теорема 24: В табличном сеттинге Policy Iteration завершает работу за конечное число итераций.

Доказательство. Алгоритм перебирает детерминированные стратегии, и, если остановка не происходит, каждая следующая лучше предыдущей:

$$\pi_k \succ \pi_{k-1} \succ \dots \succ \pi_0$$

Это означает, что все стратегии в этом ряду различны. Поскольку в табличном сеттинге число состояний и число действий конечны, детерминированных стратегий конечное число; значит, процесс должен закончиться. ■

⁴ считаем, что аргмакс берётся однозначно для любой Q-функции: в случае, если в **Argmax** содержится более одного элемента, множество действий как-то фиксированно упорядочено, и берётся действие с наибольшим приоритетом.

Алгоритм 8: Policy Iteration

Гиперпараметры: ε — критерий останова для процедуры PolicyEvaluation.

Инициализируем $\pi_0(s)$ произвольно для всех $s \in \mathcal{S}$.

На k -ом шаге:

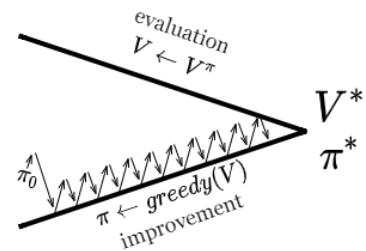
1. $V^{\pi_k} := \text{PolicyEvaluation}(\pi_k, \varepsilon)$
2. $Q^{\pi_k}(s, a) := r(s, a) + \gamma \mathbb{E}_{s'} V^{\pi_k}(s')$
3. $\pi_{k+1}(s) := \underset{a}{\operatorname{argmax}} Q^{\pi_k}(s, a)$
4. критерий останова: $\pi_k \equiv \pi_{k+1}$

Пример 51: Решим задачу из примера 49, $\gamma = 0.9$; на каждой итерации слева отображается $V^{\pi_k}(s)$; справа улучшенная π_{k+1} . За 4 шага алгоритм сходится к оптимальной стратегии.

3.3.5. Generalized Policy Iteration

Теперь поймём, почему Policy Iteration — более общая схема. Будем считать, что процедуру оценивания стратегии, этап Policy Evaluation, мы решаем методом простой итерации. 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 $\varepsilon > 0$.

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 V^{π_k} в качестве инициализации. Мы формально потеряем 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, аргмакс поменяется. Тогда наш алгоритм примет следующий вид:



Алгоритм 9: Generalized Policy Iteration

Гиперпараметры: N — количество шагов.

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

На k -ом шаге:

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

Утверждение 27: Generalized Policy Iteration (алг. 9) совпадает с Value Iteration (алг. 7) при $N = 1$ и с Policy Iteration (алг. 8) при $N = \infty$.

Доказательство. Второе очевидно; увидим первое. При $N = 1$ наше обновление V-функции имеет следующий вид:

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

Вспомним, по какому распределению берётся мат. ожидание \mathbb{E}_a : по π , которая имеет вид

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

Внутри аргмакса как раз стоит содержимое нашего мат.ожидания в обновлении V, поэтому это обновление выродится в

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

Это в точности обновление из алгоритма Value Iteration. ■

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^* можно рассматривать как приближение Q^π для $\pi(s) := \operatorname{argmax}_a Q_k^*(s, a)$; то есть в алгоритме хоть и не потребовалось в явном виде хранить «текущую» стратегию, она всё равно неявно в нём присутствует.

Давайте попробуем понять, что происходит в Generalized Policy Iteration при промежуточных N . Заметим, что повторение N раз метода простой итерации для решения уравнения $\mathfrak{B}V^\pi = V^\pi$ эквивалентно одной итерации метода простой итерации для решения уравнения $\mathfrak{B}^N V^\pi = V^\pi$ (где запись \mathfrak{B}^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}^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 25 — N -шаговое уравнение Беллмана:

$$V^\pi(s_0) = \mathbb{E}_{\mathcal{T}: N \sim \pi | s_0} \left[\sum_{t=0}^{N-1} \gamma^t r_t + \gamma^N \mathbb{E}_{s_N} V^\pi(s_N) \right] \quad (3.23)$$

Доказательство по индукции. Для получения уравнения на N шагов берём $N - 1$ -шаговое и подставляем в правую часть раскрутку на один шаг из уравнения (3.2). Это в точности соответствует применению оператора Беллмана N раз. ■

Доказательство без индукции. Для любых траекторий \mathcal{T} верно, что

$$R(\mathcal{T}) = \sum_{t=0}^{N-1} \gamma^t r_t + \gamma^N R_N$$

Возьмём мат.ожидание $\mathbb{E}_{\mathcal{T} \sim \pi | s_0}$ слева и справа:

$$\mathbb{E}_{\mathcal{T} \sim \pi | s_0} R(\mathcal{T}) = \mathbb{E}_{\mathcal{T} \sim \pi | s_0} \left[\sum_{t=0}^{N-1} \gamma^t r_t + \gamma^N R_N \right]$$

Слева видно определение V-функции. Справа достаточно разделить мат.ожидание на мат.ожидание по первым N шагам и хвост:

$$V^\pi(s_0) = \mathbb{E}_{\mathcal{T}: N \sim \pi | s_0} \left[\sum_{t=0}^{N-1} \gamma^t r_t + \gamma^N \mathbb{E}_{s_N} \mathbb{E}_{\mathcal{T}_{N:} \sim \pi | s_N} R_N \right]$$

Осталось выделить справа во втором слагаемом определение V-функции. ■

Утверждение 28: \mathfrak{B}^N — оператор с коэффициентом сжатия γ^N .

Доказательство.

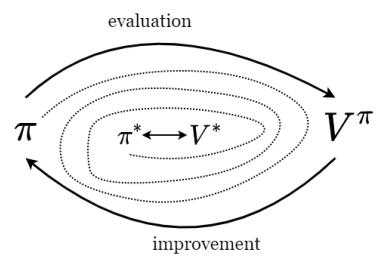
$$\rho(\mathfrak{B}^N V_1, \mathfrak{B}^N V_2) \leq \gamma \rho(\mathfrak{B}^{N-1} V_1, \mathfrak{B}^{N-1} V_2) \leq \dots \leq \gamma^N \rho(V_1, V_2)$$

■

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 одноподавленного 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 V^π , а коэффициент сжатия будет ноль, и метод простой итерации как бы сходится тут же за один шаг. Но для проведения этого шага нужно вычислить 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 Policy Improvement-ом. Интуитивно, алгоритм стабилизируется, если оценочная функция будет удовлетворять уравнению Беллмана для текущей π (иначе оператор \mathfrak{B}^N изменит значение функции), и если π выбирает $\operatorname{argmax}_a Q^\pi(s, a)$ из неё; то есть, при сходимости π удовлетворит критерию оптимальности; существуют доказанные гарантии сходимости.

Все алгоритмы, которые мы будем обсуждать далее, так или иначе подпадают под обобщённую парадигму «оценивание—улучшение». Оценивание внутри алгоритма будет явно или неявно происходить при подсчёте каких-то оценок V^π или Q^π для текущей стратегии; Policy Improvement, как мы увидим, тоже будет выступать в разных формах. Например, самый простой способ решать задачу RL — погенерировать несколько случайных стратегий и выбрать среди них лучшую — тоже подпадает под эту парадигму: мы считаем Монте-Карло оценки значения $J(\pi)$ для нескольких разных стратегий (evaluation) и выбираем наилучшую стратегию (improvement). Мы далее начнём строить model-free алгоритмы, взяв наши алгоритмы планирования — Policy Iteration и Value Iteration, — и попробовав превратить их в табличные алгоритмы решения задачи.



§3.4. Табличные алгоритмы

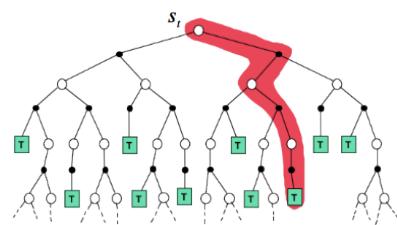
3.4.1. Монте-Карло алгоритм

Value iteration и Policy iteration 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 полечим нейронками, а сейчас будем лечить 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 RL-алгоритм: мы можем отправить в среду пособирать траектории какой-то стратегии, 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.

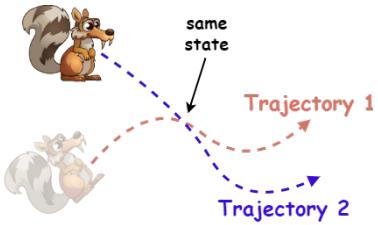
Рассмотрим самый простой способ превратить Policy Iteration в model-free метод. Давайте очередную стратегию π_k отправим в среду, сыграем несколько эпизодов, и будем оценивать Q^{π_k} по Монте-Карло:

$$Q^\pi(s, a) \approx \frac{1}{N} \sum_{i=0}^N R(\mathcal{T}_i), \quad \mathcal{T}_i \sim \pi \mid s_0 = s, a_0 = a$$

Теперь, доиграв эпизод до конца, мы для каждой встретившейся пары s, a в полученной траектории можем посчитать reward-to-go и использовать этот сэмпл для обновления нашей аппроксимации Q-функции — проведения **Монте-Карло бэкапа** (MC-backup). 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.



Формально, из-за петлей сэмплы являются скоррелированными. Если мы крутимся в петле, а потом в какой-то момент эпизода вышли и получили +1, то сэмплы будут выглядеть примерно так: $\gamma^5, \gamma^4, \gamma^3, \dots$. Причина в том, что мы взяли по несколько сэмплов для одной и той же Монте-Карло оценки (для одной и той же пары s, a) из одного и того же эпизода («*every-visit*»); для теоретической корректности следует гарантировать независимость сэмплов, например, взяв из каждого эпизода для каждой пары s, a сэмпл только для первого посещения («*first-visit*»).



$Q^\pi(s, a)$, то мы «забыли», какую часть из этой +100 мы получена в далёком будущем — не использовали разложение награды за эпизод в сумму наград за шаг. Также мы поселяли «информацию о соединениях состояниях»: пусть у нас было две траектории (см. рисунок), 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 для детерминированных стратегий.

А ещё в таком Монте-Карло алгоритме встаёт вопрос: когда заканчивать оценивание Q-функции и делать шаг Policy Improvement-а? Точное значение Q^{π_k} за конечное время мы Монте-Карло оценкой всё равно не получим, и в какой-то момент improvement проводить придётся, с потерей теоретических гарантий. Возникает вопрос: насколько разумно в таких условиях после очередного обновления стратегии начинать расчёт оценочной функции $Q^{\pi_{k+1}}$ «с нуля»? Может, имеет смысл проинициализировать как-то приближением Q^{π_k} , которое хоть и считалось для «предыдущей» стратегии и формально содержит сэмплы не из того распределения, но всё-таки их там было аккумулировано много, да и стратегия потенциально поменялась не сильно. Возникает желание усреднять сэмплы с приоритетом более свежих, приходящих из «правильной» стратегии; а «неправильные» сэмплы, из старой стратегии, всё-таки использовать, но с каким-то маленьким весом. Хочется это как-то делать онлайн, не храня всю историю Монте-Карло оценок.

3.4.2. 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 48: Экспоненциальным сглаживанием (exponential smoothing) для последовательности $x_1, x_2, x_3 \dots$ 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.

Можно ли оценивать мат.ожидание как-то по-другому? В принципе, любая выпуклая комбинация имеющихся сэмплов будет несмешённой оценкой. В частности, если $\alpha_k > \frac{1}{k}$, то мы «выдаём» более свежим сэмплам больший вес. Зададимся таким техническим вопросом: при каких других последовательностях α_k формула позволит оценивать среднее?

Определение 49: Будем говорить, что последовательность $\alpha_k \in [0, 1]$ удовлетворяет **условиям Роббинса-Монро** (Robbins-Monro conditions), если:

$$\sum_{k \geq 0} \alpha_k = +\infty, \quad \sum_{k \geq 0} \alpha_k^2 < +\infty. \tag{3.24}$$

Теорема 26: 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 (3.24). Тогда экспоненциальное сглаживание

$$m_k := (1 - \alpha_k)m_{k-1} + \alpha_k x_k \quad (3.25)$$

сходится к \mathbf{m} с вероятностью 1.

Доказательство. Без ограничения общности будем доказывать утверждение для $\mathbf{m} = \mathbf{0}$, поскольку для сведения к этому случаю достаточно вычесть \mathbf{m} из правой и левой части (3.25) и перейти к обозначениям $\hat{m}_k := m_k - \mathbf{m}$, $\hat{x}_k := x_k - \mathbf{m}$.

Итак, пусть $\mathbb{E}x_k = \mathbf{0}$. Будем доказывать, что $v_k := \mathbb{E}m_k^2 \xrightarrow{k \rightarrow \infty} 0$. Для начала возведём обе стороны уравнения (3.25) в квадрат:

$$m_k^2 = (1 - \alpha_k)^2 m_{k-1}^2 + \alpha_k^2 x_k^2 + 2\alpha_k(1 - \alpha_k)x_k m_{k-1}$$

Возьмём справа и слева матожидание:

$$\mathbb{E}m_k^2 = (1 - \alpha_k)^2 \mathbb{E}m_{k-1}^2 + \alpha_k^2 \mathbb{E}x_k^2 + 2\alpha_k(1 - \alpha_k)\mathbb{E}(x_k m_{k-1})$$

Последнее слагаемое зануляется, поскольку в силу независимости $\mathbb{E}(x_k m_{k-1}) = \mathbb{E}x_k \mathbb{E}m_{k-1}$, а $\mathbb{E}x_k$ равно нулю по условию. Используя введённое обозначение, получаем такой результат:

$$v_k = (1 - \alpha_k)^2 v_{k-1} + \alpha_k^2 \mathbb{E}x_k^2 \quad (3.26)$$

Сейчас мы уже можем доказать, что $v_k \leq C$. Действительно, сделаем это по индукции. База: $v_0 = \mathbf{0} \leq C$ по определению. Шаг: пусть $v_{k-1} \leq C$, тогда

$$v_k = (1 - \alpha_k)^2 v_{k-1} + \alpha_k^2 \mathbb{E}x_k^2 \leq (1 - \alpha_k)^2 C + \alpha_k^2 C \leq C$$

где последнее неравенство верно при любых $\alpha_k \in [0, 1]$.

Особенность дальнейшего доказательства в том, что последовательность v_k вовсе не обязана быть монотонной. Поэтому применим пару фокусов в стиле матана. Сначала раскроем скобки в рекурсивном выражении (3.26):

$$v_k - v_{k-1} = -2\alpha_k v_{k-1} + \alpha_k^2(v_{k-1} + \mathbb{E}x_k^2)$$

Мы получили счётное число равенств, проиндексированных k . Просуммируем первые n из них:

$$v_n - v_0 = -2 \sum_{k=0}^{n-1} \alpha_k v_k + \sum_{k=0}^{n-1} \alpha_k^2 (v_k + \mathbb{E}x_k^2) \quad (3.27)$$

Заметим, что $v_0 = \mathbf{0}$, а $v_n \geq \mathbf{0}$ по определению как дисперсия. Значит:

$$2 \sum_{k=0}^{n-1} \alpha_k v_k \leq \sum_{k=0}^{n-1} \alpha_k^2 (v_k + \mathbb{E}x_k^2)$$

Применяем ограниченность v_k и $\mathbb{E}x_k^2$:

$$2 \sum_{k=0}^{n-1} \alpha_k v_k \leq \sum_{k=0}^{n-1} \alpha_k^2 (C + C) = 2C \sum_{k=0}^{n-1} \alpha_k^2$$

Ряд справа сходится при $n \rightarrow +\infty$. Значит, сходится и ряд слева. Коли так, имеет предел правая часть (3.27). Значит, имеет предел и левая.

Было доказано, что последовательность v_k имеет предел. Понятно, что он неотрицателен. Допустим, он положителен и отделён от 0, равен некоторому $b > 0$. Возьмём какое-нибудь небольшое $\varepsilon > 0$, так что $b - \varepsilon > 0$. Тогда, начиная с некоторого номера i все элементы последовательности $v_k > b - \varepsilon$ при $k \geq i$. Получим:

$$\sum_{k=0}^{n-1} \alpha_k v_k \geq \sum_{k=i}^{n-1} \alpha_k v_k \geq (b - \varepsilon) \sum_{k=i}^{n-1} \alpha_k$$

Правая часть неравенства расходится, поскольку мы просили расходимость ряда из α_k ; но ранее мы доказали сходимость левой части. Значит, предел равен нулю. ■

3.4.3. Стохастическая аппроксимация

Мы научились, можно считать, решать уравнения такого вида:

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

где справа стоит матожидание по неизвестному распределению $\varepsilon \sim p(\varepsilon)$ от какой-то 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:

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

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

$$x = \mathbb{E}_\varepsilon f(x, \varepsilon), \quad (3.28)$$

где справа стоит матожидание по неизвестному распределению 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 («перевзвешанной» Монте-Карло оценки)? 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»:

$$x_k = (1 - \alpha_k)x_{k-1} + \alpha_k f(x_{k-1}, \varepsilon), \quad \varepsilon \sim p(\varepsilon) \quad (3.29)$$

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 α_k удовлетворяют условиям (3.24), то такая процедура будет в пределе сходиться.

Поймём, почему задача стохастической аппроксимации тесно связана со стохастической оптимизацией. Для этого перепишем формулу (3.29) в альтернативной очень интересной форме:

$$x_k = x_{k-1} + \alpha_k (f(x_{k-1}, \varepsilon) - x_{k-1}), \quad \varepsilon \sim p(\varepsilon) \quad (3.30)$$

Да это же формула стохастического градиентного спуска! И, видимо, выражение $f(x_{k-1}, \varepsilon) - x_{k-1}$ есть какой-то «стохастический градиент». Стохастический он потому, что это выражение случайно: мы сэмплируем $\varepsilon \sim p(\varepsilon)$; а градиент поскольку он обладает следующим свойством: в точке решения, то есть в точке x^* , являющейся решением уравнения (3.28), в среднем его значение равно нулю:

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

Это наблюдение будет иметь для нас ключевое значение. В 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

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

мы сталкиваемся в точности с задачей стохастической аппроксимации (3.28)! Справа стоит матожидание по недоступному нам распределению, содержимое которого мы, тем не менее, при данном сэмпле $s' \sim \sim p(s' | s, a)$ 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^* (3.17), имеющее вид «максимум от неизвестного распределения»

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

под данную форму не попадает. И с такими уравнениями мы ничего сделать не сможем. К счастью, нам и не понадобится.

3.4.4. Temporal Difference

Итак, 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 наше текущее приближение $Q^\pi(s, a)$ в сторону сэмпла

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

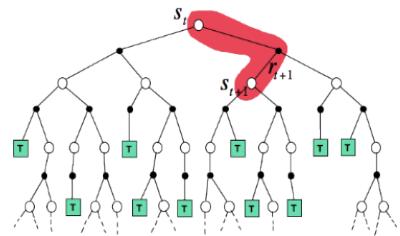
который также будем называть *таргетом*⁵ (Bellman target). Получаем следующую формулу обновления:

$$Q_{k+1}^\pi(s, a) \leftarrow Q_k^\pi(s, a) + \alpha_k \underbrace{\left(\overbrace{r + \gamma Q_k^\pi(s', a')}^{\text{таргет}} - Q_k^\pi(s, a) \right)}_{\text{временная разность}} \quad (3.31)$$

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

Мы таким образом придумали *TD-backup*: обновление, имеющее как ширину, так и длину один. Мы рассматриваем лишь одну версию будущего (один сэмпл) и заглядываем на один шаг вперёд, приближая всё дальнейшее будущее своей собственной текущей аппроксимацией. Этот ход позволяет нам 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') .

Формула (3.31) отличается от Монте-Карло обновлений Q-функции лишь способом построения таргета: в Монте-Карло мы бы взяли в качестве y reward-to-go, когда здесь же используем одношаговое приближение. Поэтому смотреть на эту формулу также можно через интуицию *бутстррапирования* (bootstrapping)⁶. 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.



Пример 52: Вы сидите в кафе (s) и хотите вернуться домой. Вы прикидываете, что в среднем этой займёт $-Q^\pi(s, a) = 30$ минут. Вы тратите одну минуту ($-r$) на выход из кафе и обнаруживаете пробку (s'). За счёт этой новой информации вы можете дать более точную оценку времени до возвращения до дома: $-Q^\pi(s', a') = 40$ минут. Как можно в будущем откорректировать наши прогнозы? Можно засечь время, сколько в итоге заняла поездка — доиграть эпизод до конца и посчитать Монте-Карло оценку — а можно уже сделать вывод о том, что случилась некоторая «временная разность», ошибка за один шаг, равная $41 - 30 = 11$ минут. Заменять исходное приближение расстояния от кафе до дома $-Q^\pi(s, a)$ на 41 минуту, конечно же, слишком грубо: но временная разность говорит, что 30 минут было заниженной оценкой и её надо чуть-чуть увеличить.

Обсудим следующий важный технический момент. Какие есть ограничения на переходы (s, a, r, s', a') , которые мы можем использовать для обновлений по формуле (3.31)? 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.

⁵ в англоязычной литературе встречается слово guess — «догадка»; этот таргет имеет смысл наших собственных предположений о том, какое будущее нас ждёт.

⁶бутстрэп — «ремешки на ботинках», происходит от выражения «потянуть самого себя за ремешки на ботинках и так перелезть через ограду». Русскоязычный аналог — «тащить самого себя за волосы из болота». Наши таргеты построены на принципе такого бутстррапирования, поскольку мы начинаем «из воздуха» делать себе сэмплы из уже имеющихся сэмплов.

Оба этих утверждения вытекают из того, что обновление (3.31) неявно ищет решение уравнения

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

как схема стохастической аппроксимации. Поэтому мы будем уделять много внимания тому, из правильных ли распределений приходят данные, которые мы «скармливаем» этой формуле обновления. Но и с другой стороны: схема не требует, например, чтобы после $Q^\pi(s, a)$ мы обязательно на следующем шаге обновили $Q^\pi(s', a')$, ровно как и не говорит ничего о требованиях на распределение, из которого приходят пары s, a . Как мы увидим позже, это наблюдение означает, что нам вовсе не обязательно обучаться с онлайн-опыта.

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

$$\begin{aligned} y &= r + \gamma Q_k^\pi(s', a') = \\ &= r + \gamma Q_k^\pi(s', \pi_k(s')) = \\ &= r + \gamma Q_k^\pi(s', \underset{a'}{\operatorname{argmax}} Q_k^\pi(s', a')) = \\ &= r + \gamma \max_{a'} Q_k^\pi(s', a') \end{aligned}$$

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:

$$Q_{k+1}^*(s, a) \leftarrow Q_k^*(s, a) + \alpha_k \left(r + \gamma \max_{a'} Q_k^*(s', a') - Q_k^*(s, a) \right) \quad (3.32)$$

Теорема 27 — 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

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

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

$$\sum_{k \geq 1} \alpha_k(s, a) = +\infty \quad \sum_{k \geq 1} \alpha_k(s, a)^2 < +\infty \quad (3.33)$$

Тогда Q_k^* сходится к Q^* с вероятностью один.

Доказательство вынесено в приложение A.3. ■

В частности, если агент некоторым образом взаимодействует со средой и на k -ом шаге обновляет своё текущее приближение Q^* только для одной пары s, a , то можно считать, что $\alpha_k(s, a) \neq 0$ только для неё. При этом ограничения (3.33) всё ещё могут оказаться выполненными, поэтому эта интересующая нас ситуация есть просто частный случай сформулированной теоремы.

Заметим, что для выполнения условий сходимости необходимо для каждой пары s, a проводить бесконечное число обновлений $Q^*(s, a)$: в противном случае, в ряду $\sum_{k \geq 1} \alpha_k(s, a)$ будет конечное число

ненулевых членов, и мы не попадём под теорему. Значит, наш сбор опыта должен удовлетворять условию *infinite visitation* — 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' :

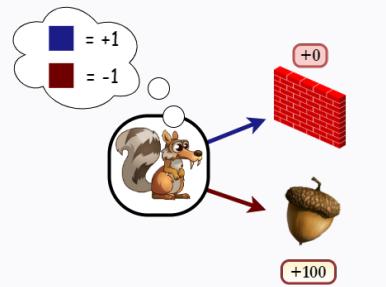
Утверждение 29: Пусть сбор опыта проводится так, что пары s, a встречаются бесконечное число раз. Пусть $n(s, a)$ — счётчик количества выполнений действия a в состоянии s во время взаимодействия агента со средой. Тогда можно положить $\alpha_k(s, a) := \frac{1}{n(s, a)}$, чтобы гарантировать сходимость алгоритма к Q^* .

3.4.5. Exploration-exploitation дилемма

Так какой же стратегией играть со средой, чтобы получать траектории? Коли мы учим Q^* , у нас на очередном шаге есть текущее приближение $\pi_k^*(s) := \underset{a}{\operatorname{argmax}} Q_k^*(s, a)$, и мы можем использовать (exploit) эти знания.

Теорема 28: Собирая траектории при помощи жадной стратегии, есть риск не сойтись к оптимальной.

Доказательство. Приведём простой пример. Есть два действия: получить приз (+100) и тупить в стену (+0), после выполнения любого игра заканчивается. Заинициализировали $Q_0^*(\text{приз}) = -1$, $Q_0^*(\text{стена}) = +1$. В первой игре, пользуясь текущей аппроксимацией π^* , выбираем тупить в стену. Получая 0, слаживаем 0 и текущее приближение +1, получая новое значение $Q_{k=1}^*(\text{стена}) \geq 0$. Оно превосходит $Q_{k=1}^*(\text{приз}) = -1$. Очевидно, и в дальнейших играх агент никогда не выберет приз, и оптимальная стратегия не будет найдена. ■



Действительно, детерминированные стратегии не позволяют получить свойство infinite visitation: многие пары s, a просто принципиально не встречаются в порождаемых ими траекториях. В частности, из-за неё нельзя ограничиваться рассмотрением только класса детерминированных стратегий, хоть и была доказана теорема ?? о существовании в нём оптимальной стратегии: мы показали, что для сбора данных — взаимодействия со средой — детерминированные стратегии не подходят. Какие подходят?

Теорема 29: Любая стратегия, для которой $\forall s, a: \pi(a | s) > 0$, удовлетворяет условию infinite visitation, то есть с отличной от нуля вероятностью в траектории, порождённой π , встретится любая пара s, a .

Доказательство. Для любого s существует набор действий $a_0, a_1 \dots a_N$, которые позволяют с некоторой ненулевой вероятностью добраться до него из s_0 : $p(s | s_0, a_0 \dots a_N) > 0$; если это не так, то ни одна стратегия никогда не попадёт s , и мы без ограничения общности можем считать, что таких «параллельных вселенных» в нашей среде не существует. Значит, π с некоторой ненулевой вероятностью выберет эту цепочку действий $a_0 \dots a_N$, после чего с ненулевой вероятностью окажется в s и с ненулевой вероятностью выберет заданное a . ■

Если мы воспользуемся любой такой стохастической стратегией, мы попадём под действие теоремы о сходимости 27. Совершая случайные действия, мы рискуем творить ерунду, но занимаемся **исследованием** (exploration) — поиском новых возможностей в среде. Означает ли это, что нам достаточно взять полностью случайную стратегию, которая равновероятно выбирает из множества действий, отправить её в среду порождать переходики s, a, r, s', a' , обучать на них Q-функцию и на выходе в пределе мы получим Q^* ? В пределе — да.

Пример 53: Представьте, что вы отправили случайную стратегию играть в Марио. Через экспоненциально много попыток она случайно пройдёт первый уровень и попадёт на второй; для состояний из второго уровня будет проведено первое обновление Q-функции. Через условно бесконечное число таких удач Q-функция для состояний из второго уровня действительно будет выучена...

Иначе говоря, исследование — корректный, но не эффективный способ генерации данных. Использование — куда более эффективный метод, позволяющий быстро добираться до «труднодоступных областей» в среде — такими областями обычно являются области с высокой наградой, иначе задача RL скорее всего не является особо интересной — и быстрее набирать сэмплы для обновлений пока ещё редко обновлённых ячеек $Q^\pi(s, a)$. Но это «некорректный» способ: детерминированная стратегия может застрять в локальном оптимуме и никогда не увидеть, что в каком-то месте другое действие даёт ещё большую награду.

Обсудим базовые варианты, как можно решать этот exploration-exploitation trade-off в контексте вычисления оптимальной Q-функции методом временных разностей. Нам нужно взять нашу стратегию использования $\pi(s) := \underset{a}{\operatorname{argmax}} Q^*(s, a)$ и что-нибудь с ней поделать так, чтобы она стала формально стохастичной.

Определение 51: ε -жадной (ε -greedy) называется стратегия

$$\pi(s) = \begin{cases} a \sim \text{Uniform}(\mathcal{A}) & \text{с вероятностью } \varepsilon; \\ \underset{a}{\operatorname{argmax}} Q^*(s, a) & \text{иначе.} \end{cases}$$

Определение 52: *Больцмановской* с температурой τ называется стратегия

$$\pi(a | s) = \text{softmax} \left(\frac{Q^*(s, a)}{\tau} \right)$$

Для Больцмановской стратегии мы увидим интересную интерпретацию в контексте обсуждения Maximum Entropy RL (раздел ??).

3.4.6. Q-learning

Итак, соберём классический алгоритм табличного RL под названием Q-learning. Это метод временных разностей для вычисления оптимальной Q-функции с ε -жадной стратегией исследования.

Алгоритм 10: Q-learning

Вход: α — параметр экспоненциального сглаживания, ε — параметр исследований.

Инициализируем $Q^*(s, a)$ произвольно для всех $s \in \mathcal{S}, a \in \mathcal{A}$.

Наблюдаем s_0 .

На k -ом шаге:

1. с вероятностью ε играем $a_k \sim \text{Uniform}(\mathcal{A})$, иначе $a_k = \underset{a_k}{\operatorname{argmax}} Q^*(s_k, a_k)$

2. наблюдаем r_k, s_{k+1}

3. обновляем $Q^*(s_k, a_k) \leftarrow Q^*(s_k, a_k) + \alpha \left(r_k + \gamma \max_{a_{k+1}} Q^*(s_{k+1}, a_{k+1}) - Q^*(s_k, a_k) \right)$

Q-learning является типичным представителем off-policy алгоритмов: нам нигде не требовались сэмплы взаимодействия со средой конкретной стратегии. Наша стратегия сбора данных могла быть, вообще говоря, произвольной. В том числе возможно обучение с буфера: допустим, некоторый «эксперт» π^{expert} провёл много-много сессий взаимодействия со средой и собрал для нас кучу траекторий. Рассмотрим их как набор переходов. Тогда мы можем, вообще не взаимодействуя больше со средой, провести обучение Q^* с буфера: сэмплируем равномерно переход (s, a, r, s') и делаем обновление ячейки $Q^*(s, a)$ по формуле (3.32). Что мы тогда выучим?

Определение 53: Для данного буфера — набора переходов (s, a, r, s') — будем называть *эмпирическим MDP* (empirical MDP) MDP с тем же пространством состояний, действий и функций награды, где функция переходов задана следующим образом:

$$\hat{p}(s' | s, a) := \frac{N(s, a, s')}{N(s, a)},$$

где $N(s, a, s')$ — число троек s, a, s' , входящих в буфер, $N(s, a)$ — число пар s, a , входящих в буфер.

Утверждение 30: При выполнении условий на learning rate, Q-learning, запущенный с буфера, выучит Q^* для эмпирического MDP.

Доказательство. Именно из эмпирического распределения $\hat{p}(s' | s, a)$ приходит s' в формуле обновления (при равномерном сэмплировании переходов из буфера). Следовательно, для такого MDP мы и выучим оптимальную Q-функцию. ■

Естественно, если буфер достаточно большой, то мы выучим очень близкую Q^* к настоящей. Ещё более интересно, что Q-learning как off-policy алгоритм может обучаться со своего собственного опыта — со своего же собственного буфера, составленного из порождённых очень разными стратегиями переходов.

Определение 54: *Реплей буфер* (replay buffer, experience replay) — это память со всеми собранными агентом переходами $(s, a, r, s', \text{done})$.

Если взаимодействие со средой продолжается, буфер расширяется, и распределение, из которого приходит s' , становится всё больше похожим на настоящее $p(s' | s, a)$; на факт сходимости это не влияет.

Алгоритм 11: Q-learning with experience replay

Вход: α — параметр экспоненциального сглаживания, ε — параметр исследований.

Инициализируем $Q^*(s, a)$ произвольно для всех $s \in \mathcal{S}, a \in \mathcal{A}$;

Наблюдаем s_0 ;

На k -ом шаге:

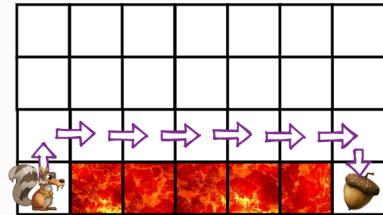
1. с вероятностью ε играем $a_k \sim \text{Uniform}(\mathcal{A})$, иначе $a_k = \underset{a_k}{\text{argmax}} Q^*(s_k, a_k)$;
2. наблюдаем r_k, s_{k+1} ;
3. кладём s_k, a_k, r_k, s_{k+1} в буфер;
4. сэмплируем случайный переход s, a, r, s' из буфера;
5. обновляем $Q^*(s, a) \leftarrow Q^*(s, a) + \alpha \left(r + \gamma \max_{a'} Q^*(s', a') - Q^*(s, a) \right)$

3.4.7. SARSA

Мы придумали off-policy алгоритм: мы умеем оценивать нашу текущую стратегию ($\underset{a}{\text{argmax}} Q(s, a)$, неявно сидящий внутри формулы обновления), используя сэмплы другой стратегии. Иными словами, у нас в алгоритме различаются понятия **целевой политики** (target policy) — стратегии, которую алгоритм выдаст по итогам обучения, она же оцениваемая политика, то есть та политика, для которой мы хотим посчитать оценочную функцию — и **политики взаимодействия** (behavior policy) — стратегии взаимодействия со средой, стратегии с подмешанным эксплорершном. Это различие было для нас принципиально: оптимальны детерминированные стратегии, а взаимодействовать со средой мы готовы лишь стохастичными стратегиями. У этого «несовпадения» есть следующий эффект.

Пример 54 — Cliff World: Рассмотрим MDP с рисунка с детерминированной функцией переходов, действиями вверх-вниз-вправо-влево и $\gamma < 1$; за попадание в лаву начисляется огромный штраф, а эпизод прерывается.

Q-learning, тем не менее, постепенно сойдётся к оптимальной стратегии: кратчайшим маршрутом агент может добраться до терминального состояния с положительной наградой. Однако даже после того, как оптимальная стратегия уже выучилась, Q-learning продолжает прыгать в лаву! Почему? Проходя прямо возле лавы, агент каждый шаг подбрасывает монетку и с вероятностью ε совершает случайное действие, которое при невезении может отправить его гореть! Если речь не идёт о симуляции, подобное поведение даже во время обучения может быть крайне нежелательно.



Возможны ситуации, когда небезопасное поведение во время обучения не является проблемой, но для, например, реальных роботов сбивать пешеходов из-за случайных действий — не самая лучшая идея. Что, если мы попробуем как-то «учесть» тот факт, что мы обязаны всегда заниматься исследованиями? То есть внутри оценочной функции должно закладываться, что «оптимальное» поведение в будущем невозможно, а возможно только около-оптимальное поведение с подмешиванием исследования. Для примера будем рассматривать ε -жадную стратегию, пока что с константным ε .

Рассмотрим очень похожий на Q-learning алгоритм. Будем использовать пятёрки s, a, r, s', a' (hence the name) прямо из траекторий нашего взаимодействия со средой и аппроксимировать текущее приближение Q-функции по формуле

$$Q(s, a) \leftarrow Q(s, a) + \alpha (r(s, a) + \gamma Q(s', a') - Q(s, a)) \quad (3.34)$$

Какую Q-функцию такой алгоритм будет учить? Поскольку $a' \sim \pi$, где π — стратегия взаимодействия со средой, то мы стохастически аппроксимируем Q^π , решая явно обычное уравнение Беллмана. Формула обновления не будет эквивалентна формуле Q-learning-a (3.32): a' мог быть тем самым «случайным» действием, случившимся из-за исследований. В большинстве случаев (с вероятностью $1 - \varepsilon$) обновление будет совпадать с Q-learning: тогда a' будет аргмаксимумом, но иногда наша Q-функция будет сдвигаться в сторону ценности случайных действий. Так в оценочную функцию попадает знание о том, что в будущем мы на каждом шаге с вероятностью ε будем обязаны выбрать случайное действие, и подобное «дёрганье» может мешать нам проходить по краю вдоль обрыва с лавой.

Алгоритм 12: SARSA

Вход: α — параметр экспоненциального сглаживания, ε — параметр исследований.

Инициализируем $Q(s, a)$ произвольно для всех $s \in \mathcal{S}, a \in \mathcal{A}$.

Наблюдаем s_0 .

$a_0 \sim \text{Uniform}(\mathcal{A})$.

На k -ом шаге:

1. наблюдаем r_k, s_{k+1} .
2. с вероятностью ε играем $a_{k+1} \sim \text{Uniform}(\mathcal{A})$, иначе $a_{k+1} = \underset{a_{k+1}}{\operatorname{argmax}} Q(s_{k+1}, a_{k+1})$
3. обновляем $Q(s_k, a_k) \leftarrow Q(s_k, a_k) + \alpha (r_k + \gamma Q(s_{k+1}, a_{k+1}) - Q(s_k, a_k))$

Попробуем понять формальнее, что происходит в таком алгоритме. Он чередует два шага: обновление (3.34), которое учит Q^π для текущей π , и, неявно, некий аналог policy improvement-а: замены π на ε -greedy(Q^π). Именно при помощи обновлённой стратегии мы будем взаимодействовать со средой на следующем шаге. Является ли такое обновление policy improvement-ом (допустим, для идеально посчитанной Q^π)? Вообще говоря, нет, но наши стратегии π , которые мы рассматриваем — не произвольные. Они все ε -жадные. Введём на минутку такое определение.

Определение 55: Будем говорить, что стратегия π — ε -мягкая (ε -soft), если $\forall s, a: \pi(a | s) \geq \frac{\varepsilon}{|\mathcal{A}|}$.

Утверждение 31: Если π_1 — ε -мягкая, то $\pi_2 := \varepsilon$ -greedy(Q^{π_1}) не хуже, чем π_1 .

Доказательство. Проверим выполнение теоремы 17; выглядит немного страшновато, но суть этих выкладок довольно лобовая:

$$\begin{aligned} V^{\pi_1}(s) &= \{\text{уравнение VQ (3.5)}\} = \sum_a \pi_1(a | s) Q^{\pi_1}(s, a) = \\ &= \sum_a \left(\pi_1(a | s) - \frac{\varepsilon}{|\mathcal{A}|} \right) Q^{\pi_1}(s, a) + \frac{\varepsilon}{|\mathcal{A}|} \sum_a Q^{\pi_1}(s, a) = \\ &= (1 - \varepsilon) \sum_a \frac{\pi_1(a | s) - \frac{\varepsilon}{|\mathcal{A}|}}{1 - \varepsilon} Q^{\pi_1}(s, a) + \frac{\varepsilon}{|\mathcal{A}|} \sum_a Q^{\pi_1}(s, a) \leq \\ &\leq (1 - \varepsilon) \max_a Q^{\pi_1}(s, a) \sum_a \frac{\pi_1(a | s) - \frac{\varepsilon}{|\mathcal{A}|}}{1 - \varepsilon} + \frac{\varepsilon}{|\mathcal{A}|} \sum_a Q^{\pi_1}(s, a) = (*) \end{aligned}$$

Заметим, что последний переход был возможен только потому, что $\pi_1(a | s) - \frac{\varepsilon}{|\mathcal{A}|} > 0$ по условию, так как π_1 — ε -мягкая по условию. Осталось заметить, что

$$\sum_a \frac{\pi_1(a | s) - \frac{\varepsilon}{|\mathcal{A}|}}{1 - \varepsilon} = \frac{\sum_a \pi_1(a | s) - \frac{\varepsilon}{|\mathcal{A}|}}{1 - \varepsilon} = 1,$$

и мы показали, что $V^{\pi_1}(s) \leq \mathbb{E}_{\pi_2(a|s)} Q^{\pi_1}(s, a)$. ■

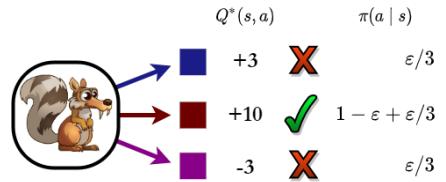
Давайте изменим постановку задачи: скажем, что мы запрещаем к рассмотрению стратегии, «похожие на детерминированные». Наложим ограничение в нашу задачу оптимизации: скажем, что стратегия обязательно должна быть ε -мягкой. В такой задаче будут свои оптимальные оценочные функции.

Определение 56: Для данного MDP оптимальными ε -мягкими оценочными функциями назовём:

$$V_{\varepsilon\text{-soft}}^*(s) := \max_{\pi \in \varepsilon\text{-soft}} V^\pi(s)$$

$$Q_{\varepsilon\text{-soft}}^*(s, a) := \max_{\pi \in \varepsilon\text{-soft}} Q^\pi(s, a)$$

Как тогда будет выглядеть принцип оптимальности? Раньше нужно было выбирать самое хорошее действие, но теперь так делать нельзя. Проводя аналогичные рассуждения, можно показать, что теперь оптимально выбирать самое хорошее действие с вероятностью $1 - \varepsilon + \frac{\varepsilon}{|\mathcal{A}|}$, а всем остальным действиям выдавать минимально разрешённую вероятность $\frac{\varepsilon}{|\mathcal{A}|}$. Как это понять? Мы уже поняли, что «взятие ε -жадной» стратегии есть местный Policy Improvement. Если мы не можем его провести ни в одном состоянии, а то есть $\pi \equiv \varepsilon$ -greedy(Q^π), то, видимо, придумать стратегию лучше в принципе невозможно:



Утверждение 32: Стратегия π оптимальна в классе ε -мягких стратегий тогда и только тогда, когда $\forall s, a$ таких, что $a \notin \text{Argmax } Q^\pi(s, a)$ верно $\pi(a | s) = \frac{\varepsilon}{|\mathcal{A}|}$.

Скетч доказательства. Притворимся, что в будущем мы сможем выбирать действия как угодно ε -мягко, то есть для данного состояния s для каждого действия a сможем в будущем набирать $Q_{\varepsilon\text{-soft}}^*(s, a)$. Как нужно выбрать действия сейчас? Нужно решить такую задачу оптимизации:

$$\begin{cases} \mathbb{E}_{\pi(a|s)} Q_{\varepsilon\text{-soft}}^*(s, a) \rightarrow \max \\ \int_{\mathcal{A}} \pi(a | s) da = 1; \quad \forall a \in \mathcal{A}: \pi(a | s) \geq \frac{\varepsilon}{|\mathcal{A}|} \end{cases}$$

Формально решая эту задачу условной оптимизации, получаем доказываемое. ■

Утверждение 33: Уравнения оптимальности, соответственно, теперь выглядят так:

$$Q_{\varepsilon\text{-soft}}^*(s, a) = r + \gamma \mathbb{E}_{s'} \left[(1 - \varepsilon) \max_{a'} Q_{\varepsilon\text{-soft}}^*(s', a') + \frac{\varepsilon}{|\mathcal{A}|} \sum_{a'} Q_{\varepsilon\text{-soft}}^*(s', a') \right]$$

Доказательство. Их можно получить, например, взяв обычное уравнение QQ (3.6) и подставив вид оптимальной стратегии. ■

Мы теперь понимаем, что наша формула обновления (3.34) — просто метод решения такого уравнения оптимальности: сэмплируя a' из ε -жадной стратегии, мы просто стохастически аппроксимируем по мат.ожиданию из ε -жадной стратегии. Мы могли бы, вообще говоря, взять это мат.ожидание полностью явно, сбив таким образом дисперсию:

$$Q(s, a) \leftarrow Q(s, a) + \alpha (r(s, a) + \gamma \mathbb{E}_{a' \sim \pi} Q(s', a') - Q(s, a)),$$

где мат.ожидание по a' по определению π равно

$$\mathbb{E}_{a' \sim \pi} Q(s', a') = (1 - \varepsilon) \max_{a'} Q(s', a') + \frac{\varepsilon}{|\mathcal{A}|} \sum_{a'} Q(s', a')$$

Такая схема называется Expected SARSA — «SARSA, в которой взяли мат.ожидание». Мы всё ещё учим Q-функцию текущей политики, но не используем сэмпл из текущей траектории, а вместо этого берём мат.ожидание по действиям из уравнения Беллмана (3.6) честно. Вообще говоря, в такой схеме мы работаем не с пятёрками s, a, r, s', a' , а с четвёрками s, a, r, s' (несмотря на название).

Алгоритм 13: Expected-SARSA

Вход: α — параметр экспоненциального сглаживания, ε — параметр исследований.

Инициализируем $Q(s, a)$ произвольно для всех $s \in \mathcal{S}, a \in \mathcal{A}$.

Наблюдаем s_0 .

π_0 случайная.

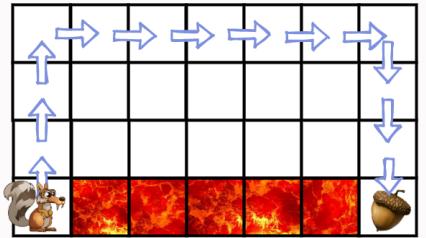
На k -ом шаге:

1. играем $a_k \sim \pi_k(a_k | s_k)$
2. наблюдаем r_k, s_{k+1}
3. π_{k+1} есть с вероятностью ε выбрать $a_{k+1} \sim \text{Uniform}(\mathcal{A})$, иначе $a_{k+1} = \arg\max_{a_{k+1}} Q(s_{k+1}, a_{k+1})$

$$4. \text{ обновляем } Q(s_k, a_k) \leftarrow Q(s_k, a_k) + \alpha (r_k + \gamma \mathbb{E}_{a_{k+1} \sim \pi_{k+1}} Q(s_{k+1}, a_{k+1}) - Q(s_k, a_k))$$

Соответственно, и SARSA, и Expected SARSA будут сходиться уже не к обычной оптимальной оценочной функции, а к $Q_{\epsilon\text{-soft}}^*(s, a)$ — оптимальной оценочной функции в семействе ϵ -мягких стратегий.

Пример 55 — Cliff World: Попробуем запустить в MDP из примера 54 SARSA. Мы сойдёмся вовсе не к оптимальной стратегии — а «к безопасной» оптимальной стратегии. Внутри нашей оценочной функции сидит вероятность сорваться в лаву при движении вдоль неё, и поэтому кратчайший маршрут перестанет давать нам наибольшую награду просто потому, что стратегии, для которых такой маршрут был оптимальен, мы перестали допускать к рассмотрению.



Принципиальное отличие схемы SARSA от Q-learning в том, что мы теперь учимся ровно на тех же сэмплах действий a' , которые отправляем в среду. Наши behavior и target policy теперь совпадают: для очередного шага алгоритма нужно сделать шаг в среде при помощи текущей π , и поэтому мы должны учиться онлайн, «в on-policy режиме».

Чтобы понять, что это влечёт, рассмотрим, что случится, если взять буфер некоторого эксперта π_{expert} , генерировать пятёрки s, a, r, s', a' из него и проводить обновления (3.34) по ним. Что мы выучим? Применим наш стандартный ход рассуждений: $a' \sim \pi_{\text{expert}}(a' | s')$, и, значит, мы учим $Q^{\text{expert}}!$ Если же мы попробуем запустить SARSA с experience replay, то есть обучаться на собственной же истории, то мы вообще творим полную фигню: на каждом шаге мы движемся в сторону Q-функции для той стратегии π , которая породила засэмплированный переход (например, если переход был засэмплирован откуда-то из начала обучения — скорее всего случайной или хуже). Такой алгоритм не просто будет расходиться, но и не будет иметь никакого смысла. Поэтому SARSA нельзя (в таком виде, по крайней мере) запустить с реплей буфера.

ГЛАВА 4

Value-based подход

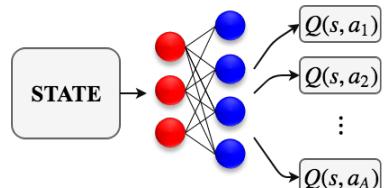
В данной главе будет рассмотрен второй, Value-based подход к решению задачи, в котором алгоритм ищет не саму стратегию, а оптимальную Q-функцию. Для этого табличный алгоритм Value Iteration будет обобщён на более сложные пространства состояний; требование конечности пространства действий $|\mathcal{A}|$ останется ограничением подхода.

§4.1. Deep Q-learning

4.1.1. Q-сетка

В сложных средах пространство состояний может быть непрерывно или конечно, но велико (например, пространство всех экранов видеоигры). В таких средах моделировать функции от состояний, будь то стратегии или оценочные функции, мы можем только приближённо при помощи параметрических семейств.

Далее мы будем предполагать, что Q-функция задана нейросеткой $Q^*(s, a, \theta)$ с параметрами θ . Заметим, что для дискретных пространств действий сетка может как принимать действия на входе, так и принимать на вход только состояние s , а выдавать $|\mathcal{A}|$ чисел $Q^*(s, a_1, \theta) \dots Q^*(s, a_{|\mathcal{A}|}, \theta)$. В последнем случае мы можем за константу находить жадное действие $\pi(s) = \underset{a}{\operatorname{argmax}} Q^*(s, a, \theta)$ и за один проход по сети считать $\max_a Q^*(s, a, \theta)$, необходимый для вычисления таргета (4.1).



В случае, если пространство действий непрерывно, выдать по числу для каждого варианта уже не получится. При этом, если непрерывное действие подаётся на вход вместе с состоянием, то оптимизировать по нему для поиска максимума или аргмаксимума придётся при помощи серии прямых и обратных проходов (для дискретного пространства — за $|\mathcal{A}|$ прямых проходов), что вычислительно ни в какие ворота. Поэтому такой вариант на практике не встречается, а алгоритм пригоден в таком виде только для дискретных пространств состояний (позже мы пофиксим это при обсуждении алгоритма DDPG, раздел ??).



Использование нейросеток позволяет обучаться для сред, используя в качестве состояний s , например, пиксельное представление экранов видеоигр. Стандартным вариантом архитектуры является несколько (не очень много) свёрточных слоёв, обычно без использования макспулингов (важно не убить информацию о расположении распознанных объектов на экране). Использование батч-нормализаций и дроп-аутов сопряжено с вопросами о том, нужно ли их включать-выключать на этапах генерации таргетов (который, как мы увидим позже, тоже распадается на два этапа), сбере опыта с возможностью исследования и так далее. Чаще их не используют, чем используют, так как можно нарваться на неожиданные эффекты. Важно помнить, что все эти блоки были придуманы для решения немного других задач, и стоит осторожно переносить их в контекст обучения с подкреплением.

4.1.2. Переход к параметрической Q-функции

Итак, мы хотим адаптировать алгоритм Q-learning для случая, когда оптимальная Q-функция представлена дифференцируемой по параметрам θ функцией $Q^*(s, a, \theta)$.

Вообще говоря, мы помним, что мы хотим решать уравнения оптимальности Беллмана (3.16), и можно было бы, например, оптимизировать невязку:

$$\left(Q^*(s, a, \theta) - r(s, a) - \gamma \mathbb{E}_{s'} \max_{a'} Q^*(s', a', \theta) \right)^2 \rightarrow \min_{\theta}$$

однако мат.ожидание $\mathbb{E}_{s'}$ в формуле берётся по неизвестному нам распределению (а даже если известному, то почти наверняка в сложных средах аналитически ничего не возьмётся) и никак не выносится (несмешённые оценки градиента нас бы устроили).

В Q-learning-e мы смогли с сохранением теоретических гарантий побороться с этим при помощи стохастической аппроксимации, получив формулу (3.32). Хочется сделать какой-то аналогичный трюк:

$$Q^*(s, a, \theta) \leftarrow Q^*(s, a, \theta) + (1 - \alpha) \left(r(s, a) + \gamma \max_{a'} Q^*(s', a', \theta) - Q^*(s, a, \theta) \right)$$

Мы уже отмечали ключевое наблюдение о том, что формула стохастической аппроксимации очень напоминает градиентный спуск, а α играет роль learning rate. Да даже условия сходимости 3.33, собственно, те же, что в стохастическом градиентном спуске¹! И, действительно, табличный Q-learning является градиентным спуском для решения некоторой задачи регрессии.

Поскольку это очень принципиальный момент, остановимся в этом месте подробнее. Пусть у нас есть текущая версия $Q^*(s, a, \theta_k)$, и мы хотим проделать шаг метода простой итерации для решения уравнения $Q^* = Q^*$ (3.16). Зададим следующую задачу регрессии:

- входом является пара s, a
- искомым (!) значением на паре s, a — правая часть уравнения оптимальности Беллмана (3.16), т.е.

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

- наблюдаемым («зашумлённым») значением целевой переменной или **таргетом** (target)

$$y(s, a) := r(s, a) + \gamma \max_{a'} Q^*(s', a', \theta_k) \quad (4.1)$$

где $s' \sim p(s' | s, a)$

- функцией потерь MSE: $\text{Loss}(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$

Заметим, что, как и в классической постановке задачи машинного обучения, значение целевой переменной — это её «зашумлённое» значение: по входу s, a генерируется s' , затем от s' считается детерминированная функция, и результат y является наблюдаемым значением. Мы можем для данной пары s, a засэмплировать себе таргет, взяв переход (s, a, r, s') и воспользовавшись сэмплом s' , но существенно, что такой таргет — случайная функция от входа. Принципиально: согласно уравнениям Беллмана, мы хотим выучить мат.ожидание такого таргета, а значит, нам нужна квадратичная функция потерь.

Теорема 30: Пусть $Q^*(s, a, \theta_{k+1})$ — достаточно ёмкая модель (model with enough capacity), выборка неограниченно большая, а оптимизатор идеальный. Тогда решением вышеописанной задачи регрессии будет шаг метода простой итерации для поиска оптимальной Q-функции:

$$Q^*(s, a, \theta_{k+1}) = r(s, a) + \gamma \mathbb{E}_{s'} \max_{a'} Q^*(s', a', \theta_k)$$

Доказательство. Под «достаточно ёмкой моделью» подразумевается, что модель может имитировать любую функцию, например для каждой пары s, a из выборки выдавать любое значение. Найдём оптимальное значение для пары s, a : для неё $Q^*(s, a, \theta_{k+1})$ выдаёт некоторое число, которое должно минимизировать MSE:

$$\frac{1}{2} \mathbb{E}_{s'} (y(s, a) - Q^*(s, a, \theta_{k+1}))^2 \rightarrow \min_{\theta_{k+1}}$$

Оптимальным константным решением регрессии с MSE как раз является среднее, получаем

$$Q^*(s, a, \theta_{k+1}) = \mathbb{E}_{s'} y(s, a, s') = r(s, a) + \gamma \mathbb{E}_{s'} \max_{a'} Q^*(s', a', \theta_k) \quad \blacksquare$$

В частности, формула Q-learning (3.32) — это частный случай решения указанной задачи регрессии стохастическим градиентным спуском для специального вида параметрических распределений:

¹Это не случайность: корни теории одни и те же.

Теорема 31: Пусть Q-функция задана табличным параметрическим семейством, а то есть табличкой $\mathcal{S} \times \mathcal{A}$, где в каждой ячейке записан параметр $\theta_{s,a}$:

$$Q^*(s, a, \theta) = \theta_{s,a}$$

Тогда формула (3.32) представляет собой градиентный спуск для решения обсуждаемой задачи регрессии.

Доказательство. Посчитаем градиент функции потерь на одном объекте s, a . Предсказанием модели будет $\hat{y} = Q^*(s, a, \theta) = \theta_{s,a}$, то есть градиент предсказания по параметрам модели равен

$$\nabla_{\theta} \hat{y} = e_{s,a}$$

где $e_{s,a}$ — вектор из $\mathbb{R}^{|\mathcal{S}| \times |\mathcal{A}|}$ из всех нулей с единственной единичкой в позиции, соответствующей паре s, a . Теперь посчитаем градиент функции потерь:

$$\nabla_{\theta} \text{Loss}(y, \hat{y}(\theta)) = (\hat{y} - y(s, a)) \nabla_{\theta} \hat{y} = (Q^*(s, a, \theta_t) - y(s, a)) e_{s,a}$$

Итак, градиентный спуск делает следующие апдейты параметров:

$$\theta_{k+1} = \theta_k - \alpha \nabla_{\theta} \text{Loss}(y, \hat{y}) = \theta_k + \alpha (y(s, a) - Q^*(s, a, \theta_t)) e_{s,a}$$

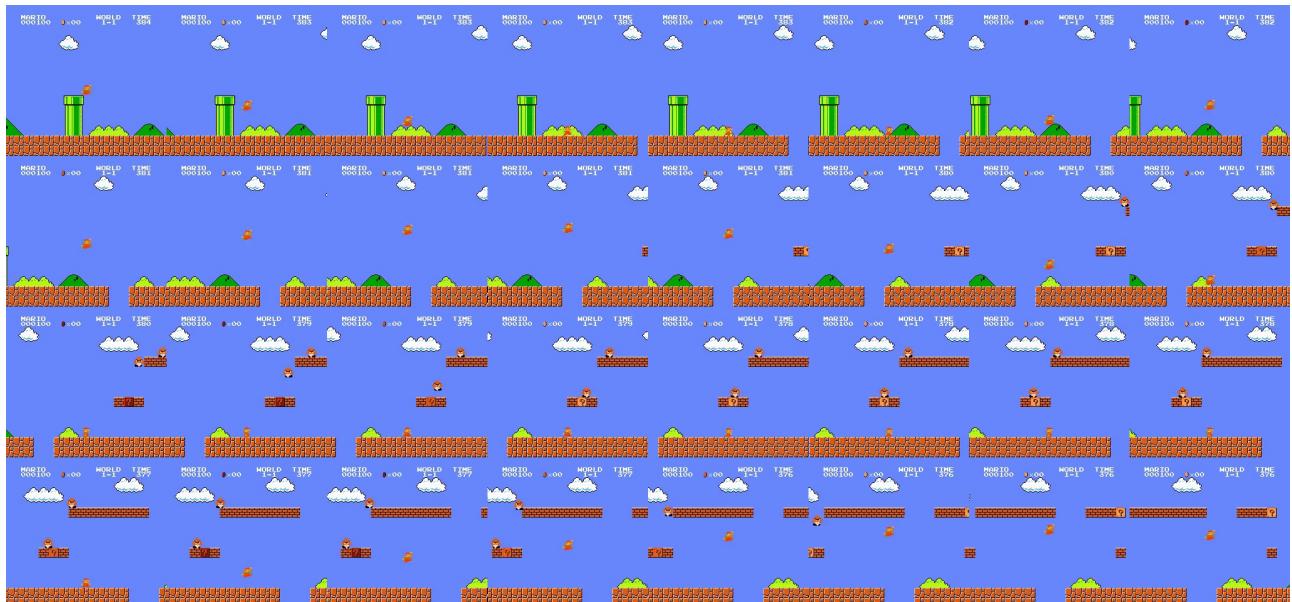
В этой формуле обновляется ровно одна ячейка таблички $\theta_{s,a}$, и это в точности совпадает с (3.32), поскольку . ■

Принципиально важно, что зависимость целевой переменной $y(s, a)$ (4.1) от параметров θ игнорируется. На одном шаге алгоритма мы хотим провести шаг метода простой итерации, но не можем взять матожидание, и решаем эту проблему, составляя задачу регрессии, где y — зафиксированная целевая переменная. Если вдруг в ней протекут градиенты, мы будем не только подстраивать прошлое под будущее, но и будущее под прошлое, что не будет являться корректной процедурой.

Важно, что как только мы от «табличных параметрических функций» переходим к произвольным параметрическим семействам, все теоретические гарантии сходимости Q-learning-a 3.33 теряются (теоремы сходимости использовали «табличность» нашего представления Q-функции). Как только мы используем ограниченные параметрические семейства вроде нейросеток, неидеальные оптимизаторы вроде Адама или не доводим каждый этап метода простой итерации до сходимости, гарантит нет.

4.1.3. Декорреляция сэмпллов

Q-learning после одного шага в среде делал апдейт одного значения таблицы $Q^*(s, a)$. Сейчас нас такой вариант, очевидно, не устраивает, потому что обучать нейросетки на мини-батчах размера 1 (особенно с уровнем доверия к нашим таргетам) — это явно плохая идея, но главное, сэмплы s, a, y сильно скоррелированы: в сложных средах последовательности состояний обычно очень сильно похожи. Нейросетки на скоррелированных данных обучаются очень плохо (чаще — не обучаются вовсе). Поэтому сбор мини-батча подряд с одной траектории здесь не сгодится.

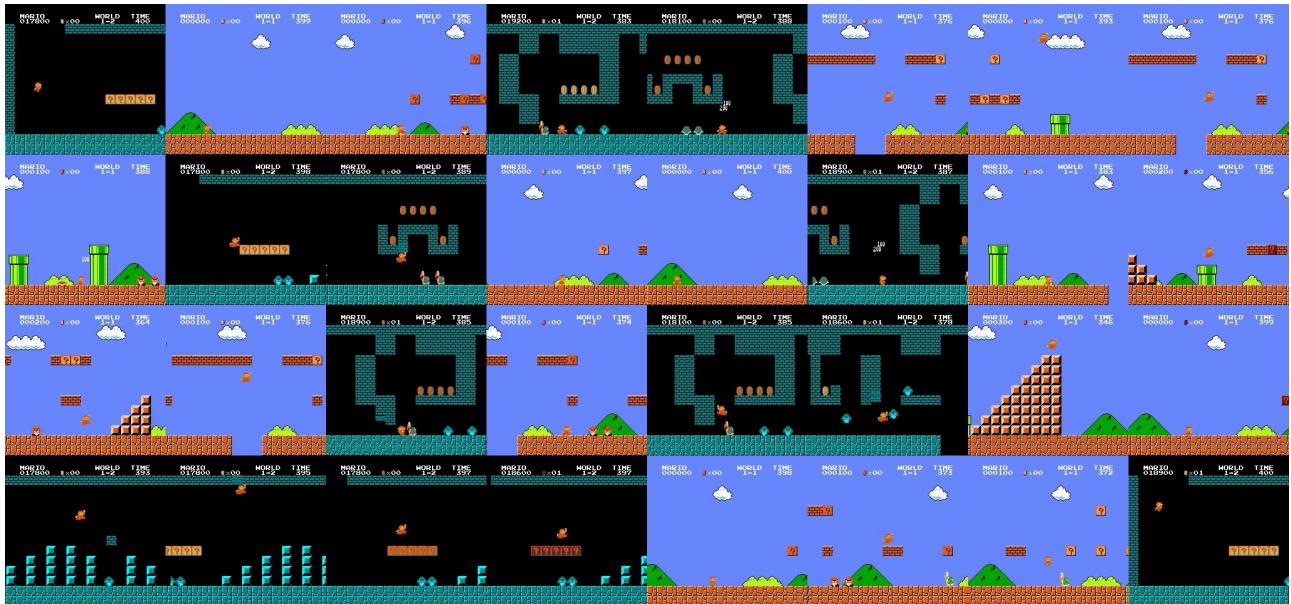


Есть два доступных на практике варианта **декорреляции сэмплов** (sample decorrelation). Первый — запуск параллельных агентов, то есть сбор данных сразу в нескольких параллельных средах. Этот вариант доступен всегда, по крайней мере, если среда виртуальна; иначе эта опция может быть дороговатой... Второй вариант — реплей буфер, который, как мы помним, является прерогативой исключительно off-policy алгоритмов.



На практике картинки в какой-то момент начинают переполнять оперативку и начинаются проблемы. Простое решение заключается в том, чтобы удалять самые старые переходы, то есть оставлять самый новый опыт, однако есть альтернативные варианты (см. приоритизированный реплей, раздел 4.2.6)

При наличии реплей буфера агент может решать задачи регрессии, сэмплируя мини-батчи переходов $\mathbb{T} = (s, a, r', s', \text{done})$ из буфера, затем делая для каждого перехода расчёт таргета $y(\mathbb{T}) := r + \gamma \max_{a'} Q^*(s', a', \theta)$, игнорируя зависимость таргета от параметров, и проводя шаг оптимизации по такому мини-батчу. Такой батч уже будет декоррелирован.



Почему мы можем использовать здесь реплей буфер? Если бы мы решали уравнения оптимальности Беллмана (3.16) минимизацией невязки (каким-то образом вычисляя точное мат.ожидание по динамике среды), мы бы минимизировали какую-нибудь сумму невязок по всем парам s, a . Поэтому в поставленной задаче регрессии мы можем брать тройки s, a, y для обучения из условно произвольного распределения до тех пор, пока оно достаточно разнообразно и нескоррелировано. Единственное ограничение — внутри y сидит s' , который должен приходить из и только из $p(s' | s, a)$. Но поскольку среда однородна, любые тройки s, a, s' из любых траекторий, сгенерированных любой стратегией, таковы, что $s' \sim p(s' | s, a)$, а значит, s' может быть использован для генерации таргета. Иными словами, в рамках текущего подхода мы, как и в табличном Q-learning-e, находимся в off-policy режиме, и наши условия на процесс порождения переходов s, a, r, s' точно такой же.

4.1.4. Таргет-сеть

Концептуально, мы поняли, что на очередном шаге алгоритма нам нужно зафиксировать целевую переменную $y(s, a)$ по формуле (4.1), провести несколько итераций градиентного спуска, и затем переходить к следующему шагу метода простой итерации, на котором задача регрессии изменится (поменяется целевая переменная). Учитывая, что целевая переменная, которая во время решения задачи регрессии считается ground truth-ом, строится с использованием текущего приближения Q-функции, возникает естественное желание менять целевую переменную каждый шаг, то есть после каждого градиентного шага использовать новую версию Q-функции для генерации таргета. Эмпирически легко убедиться, что такой подход нестабилен примерно от слова совсем.

Для стабилизации процесса одну задачу регрессии нужно решать более одной итерации градиентного спуска; необходимо сделать хотя бы условно 100-200 шагов. Проблема в том, что если таргет строится по формуле $r + \gamma \max_{a'} Q^*(s', a', \theta)$, то после первого же градиентного шага θ поменяется.

Вычисление таргета для всего реплей буфера, конечно же, непрактично (реплей буфер обычно огромен (порядка 10^6 переходов), а одна задача регрессии будет решаться суммарно 100-200 шагов на мини-батчах размера, там, 32 (итого таргет понадобится считать всего для порядка 3000 переходов). Поэтому хранится копия Q -сетки, называемая **таргет-сетью** (target network), единственная цель которой — генерировать таргеты текущей задачи регрессии для транзишнов из засэмплированных мини-батчей. Будем обозначать её параметры как θ^- . Итак, целевая переменная в таких обозначениях генерится по формуле

$$y(\mathbb{T}) = r + \gamma \max_{a'} Q^*(s', a', \theta^-)$$

а раз в K шагов веса θ^- просто копируются из текущей модели с весами θ для «задания» новой задачи регрессии.



При обновлении задачи регрессии график функции потерь типично подскакивает. Поэтому распространённой, но чуть более вычислительно дорогой альтернативой является на каждом шаге устраивать экспоненциальное сглаживание весов таргет сети. Тогда на каждом шаге:

$$\theta^- = (1 - \alpha)\theta^- + \alpha\theta,$$

где α — гиперпараметр. Такой вариант тоже увеличивает стабильность алгоритма хотя бы потому, что решаемая задача регрессии меняется «постепенно».

4.1.5. DQN

Собираем алгоритм целиком. ε -жадную стратегию исследования придётся оставить — с проблемой исследования-использования мы ничего пока не делали, и при стартовой инициализации есть риск отправиться тупить в ближайшую стену. Также лишний раз вспомним про то, что в терминальных состояниях обязательно нужно домножаться на $(1 - \text{done})$, поскольку шансов у приближённого динамического программирования сойтись куда-то без «отправной точки» не очень много.

Алгоритм 14: Deep Q-learning (DQN)

Гиперпараметры: B — размер мини-батчей, K — периодичность апдейта таргет-сети, $\varepsilon(t)$ — стратегия исследования, Q^* — нейросеть с параметрами θ , SGD-оптимизатор.

Инициализировать θ произвольно.

Положить $\theta^- := \theta$

Пронаблюдать s_0

На очередном шаге t :

1. выбрать a_t случайно с вероятностью $\varepsilon(t)$, иначе $a_t := \operatorname{argmax}_{a_t} Q^*(s_t, a_t, \theta)$
2. пронаблюдать $r_t, s_{t+1}, \text{done}_{t+1}$
3. добавить пятёрку $(s_t, a_t, r_t, s_{t+1}, \text{done}_{t+1})$ в реплей буфер
4. засэмплировать мини-батч размера B из буфера
5. для каждого перехода $\mathbb{T} = (s, a, r, s', \text{done})$ посчитать таргет:

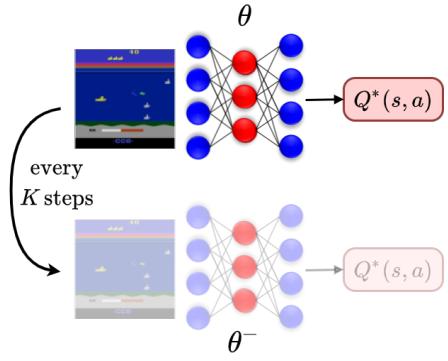
$$y(\mathbb{T}) := r + \gamma(1 - \text{done}) \max_{a'} Q^*(s', a', \theta^-)$$

6. посчитать лосс:

$$\text{Loss}(\theta) := \frac{1}{B} \sum_{\mathbb{T}} (Q^*(s, a, \theta) - y(\mathbb{T}))^2$$

7. делаем шаг градиентного спуска по θ , используя $\nabla_{\theta} \text{Loss}(\theta)$

8. если $t \bmod K = 0$: $\theta^- \leftarrow \theta$



§4.2. Модификации DQN

4.2.1. Twin DQN

Отмечалось, что без таргет-сетки (при обновлении задачи регрессии каждый шаг) можно наблюдать, как Q^* начинает неограниченно расти. Хотя таргет-сетка более-менее справляется с тем, чтобы стабилизировать процесс, предотвратить этот эффект полностью у неё не получается: сравнение обучающейся Q^* с Монте-Карло оценками и зачастую просто со здравым смыслом выдаёт присутствие в алгоритме заметного *смещения в сторону переоценки* (overestimation bias). Почему так происходит?

Очевидно, что источник проблемы — оператор максимума в формуле построения таргета:

$$y(\mathbb{T}) = r + \gamma \max_{a'} Q^*(s', a', \theta^-)$$

При построении таргета есть два источника ошибок: 1) ошибка аппроксимации 2) внешняя стохастика. Максимум здесь «выбирает» то действие, для которого из-за ошибки нейросети или из-за везения в прошлых играх $Q(s', a')$ больше правильного значения.

Пример 56: Вы выиграли в лотерею +100. Из-за того, что вы моделируете Q-сетку нейросетью, при увеличении $Q(s, a)$ для действия «купить билет в лотерею» случайно увеличилось значение $Q(s, a)$ для действия «играть с однорукими бандитами», поскольку оно опиралось на примерно те же признаки описания состояний. В результате, при построении таргета для состояния казино s' в задачу регрессии поступают завышенные значения, которые дальше распространяются на другие состояния: начинается «цепная реакция».

Одно из хороших решений проблемы заключается в разделении (decoupling) двух этапов подсчёта максимума: *выбор действия* (action selection) и *оценка действия* (action evaluation):

$$\max_{a'} Q^*(s', a') = \underbrace{Q^*(s', \operatorname{argmax}_{a'} Q^*(s', a'))}_{\text{выбор действия}} \underbrace{\overbrace{Q^*(s', \operatorname{argmax}_{a'} Q^*(s', a'))}^{\text{оценка действия}}}_{\text{выбор действия}}$$

В Twin DQN² предлагается обучать два приближения Q-функции параллельно и использовать аппроксимацию Q-функции «близнеца» для этапа оценивания действия:

$$y_1(\mathbb{T}) := r + \gamma Q_{\theta_1}^*(s', \operatorname{argmax}_{a'} Q_{\theta_2}^*(s', a'))$$

$$y_2(\mathbb{T}) := r + \gamma Q_{\theta_2}^*(s', \operatorname{argmax}_{a'} Q_{\theta_1}^*(s', a'))$$

Если обе аппроксимации Q-функции идеальны, то, понятное дело, мы всё равно получим честный максимум. Однако, если оба DQN честно запущены параллельно и даже собирают каждый свой опыт (что в реальности, конечно, дорого), их ошибки аппроксимации и везения будут в «разных местах». В итоге, Q-функция близнеца выступает в роли более пессимистичного критика действия, выбранного текущей Q-функцией. Таргет-сети в такой модели не нужны, то есть задача регрессии, условно говоря, меняется после каждого обновления весов.



Если сбор уникального опыта для каждого из близнецов не организуется, Q-функции всё равно получаются скоррелированными. Как минимум, нужно сэмплировать для обучения сеток разные мини-батчи из реплей буфера, если он общий.

Интуиция, почему «независимая» Q-функция близнеца используется именно для оценки действия, а не наоборот для выбора: если из-за неудачного градиентного шага наша сетка Q_{θ_1} пошла куда-то не туда, мы не хотим, чтобы плохие значения попадали в её же таргет. Плохие действия в таргет пусть попадают: пессимистичная оценка здесь предпочтительнее.

²когда эту идею предлагали в 2010-ом году в рамках классического RL (тогда нейронки ещё не вставили), то называли её Double Q-learning, но сейчас под Double DQN подразумевается алгоритм 2015-го года, а этот трюк иногда именуется «близнецами» (Twin). Правда, в последнее время из-за алгоритма Twin Delayed DDPG под словом Twin понимается снова не эта формула...

4.2.2. Clipped Twin DQN

Разовьём интуицию дальше. Вот мы строим таргет для \mathbf{Q}_{θ_1} . Мы готовы выбрать при помощи неё же действия, но не готовы оценить их ею же самой в силу потенциальной переоценки. Для этого мы и берём «независимого близнеца» \mathbf{Q}_{θ_2} . Но что, если он выдаёт ещё больше? Что, если его оценка выбранного действия, так случилась, потенциально ещё более завышенная? Давайте уж в таких ситуациях всё-таки брать то значение, которое поменьше! Получаем следующую интересную формулу:

$$y_1(\mathbb{T}) := r + \gamma \min_{i=1,2} Q_{\theta_i}^*(s', \operatorname{argmax}_{a'} \mathbf{Q}_{\theta_2}^*(s', a'))$$

$$y_2(\mathbb{T}) := r + \gamma \min_{i=1,2} Q_{\theta_i}^*(s', \operatorname{argmax}_{a'} \mathbf{Q}_{\theta_1}^*(s', a'))$$

Мы по сути придумали забавный метод «ансамблирования» Q-функций: только вместо интуитивного среднего берём минимум, для борьбы с максимумом.



Конечно, в отличие от обычного ансамблирования в машинном обучении, такой подход плохо масштабируется с увеличением числа обучаемых Q-функций (обычно учат всё-таки только две). В случае ансамбля имеет смысл брать не среднее, и не минимум, а, например, какой-то квантиль, более близкий к минимуму, чем к медиане — но возникает неудобный гиперпараметр, что же именно брать.

4.2.3. Double DQN

Запускать параллельно обучение двух сеток дорого, а при общем реплей буфере корреляция между ними всё равно будет. Поэтому предлагается простая идея: запускать лишь один DQN, а в формуле таргета для оценивания вместо «близнеца» использовать таргет-сеть. То есть: пусть θ — текущие веса, θ^- — веса таргет-сети, раз в K шагов копирующиеся из θ . Тогда таргет вычисляется по формуле:

$$y(\mathbb{T}) := r + \gamma Q^*(s', \operatorname{argmax}_{a'} Q^*(s', a', \theta), \theta^-)$$

4.2.4. Dueling DQN

Рассмотрим ещё одну очевидную беду Q-обучения на примере.

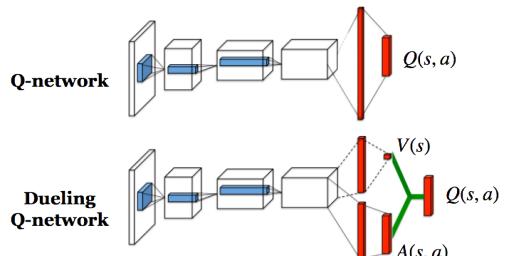
Пример 57: Вы в целях исследования попробовали кинуться в яму s . Сидя в яме, вы попробовали a — поднять правую руку. В яме холодно и грустно, поэтому вы получили -100. Какой вывод делает агент? Правильно: для этого состояния s оценку этого действия a нужно понизить. Остальные не трогать; оценка самого состояния (по формуле (3.14) — максимум Q-функции по действиям) скорее всего не изменится, и надо бы вернуться в это состояние и перепроверить ещё и все остальные действия: попробовать поднять левую руку, свернуться калачиком...

В сложных MDP ситуация зачастую такова, что получение негативной награды в некоторой области пространства состояний означает в целом, что попадание в эту область нежелательно. Верно, что с точки зрения теории остальные действия должны быть «исследованы», но неудачный опыт должен учитываться внутри оценки самого состояния; иначе агент будет возвращаться в плохие состояния с целью перепробовать все действия без понимания, что удача здесь маловероятна, вместо того, чтобы исследовать те ветви, где негативного опыта «не было». Понятно, что проблема тем серьёзнее, чем больше размерность пространства действий $|\mathcal{A}|$.

Формализуя идею, мы хотели бы в модели учить не $Q^*(s, a)$ напрямую, а получать их с учётом $V^*(s)$. Иными словами, модель должна знать ценность самих состояний и с её учётом выдавать ценности действий. *Дуэльная* (dueling) архитектура — это модификация вычислительного графа нашей модели с параметрами θ , в которой на выходе предлагаются иметь две головы, V-функцию V^* и Advantage-функцию A^* :

$$Q^*(s, a, \theta) := V^*(s, \theta) + A^*(s, a, \theta) \quad (4.2)$$

Здесь есть проблема: если $V^*(s, \theta)$ — это произвольный скаляр, то $A^*(s, a, \theta)$ — не произвольный. Advantage-функция не является произвольной функцией и обязана подчиняться (20). Для оптимальных стратегий в предположении жадности нашей стратегии это утверждение вырождается в следующее свойство:



Утверждение 34:

$$\forall s: \max_a A^*(s, a) = 0$$

Доказательство.

$$\max_a A^*(s, a) = \max_a Q^*(s, a) - V^*(s) = \{V^*Q^*\} (3.14) = 0 \quad \blacksquare$$

Это условие можно легко учесть, вычтя максимум в формуле (4.2):

$$Q^*(s, a, \theta) = V^*(s, \theta) + A^*(s, a, \theta) - \max_{\hat{a}} A^*(s, \hat{a}, \theta) \quad (4.3)$$

Таким образом мы гарантируем, что максимум по действиям последних двух слагаемых равен нулю, и они корректно моделируют Advantage-функцию.

Заметим, что V^* и A^* не учаются по отдельности (для V^* уравнение оптимальности Беллмана не сводится к регрессии, для A^* уравнения Беллмана не существует вообще); вместо этого минимизируется лосс для Q-функции точно так же, как и в обычном DQN.



В нейросетях формулу (4.3) реализовать очень просто при помощи двух голов: одна выдаёт скаляр, другая $|\mathcal{A}|$ чисел, из которых вычитается максимум. Дальше к каждой компоненте второй головы добавляется скаляр, выданный первой головой, и результат считается выданным моделью $Q^*(s, a, \theta)$. Преимущество модели остаётся в том, что при апдейте значения для одной пары s, a неизбежно поменяется значение $V^*(s, \theta)$ ценности всего состояния, и ценность «всех действий» сразу упадёт.

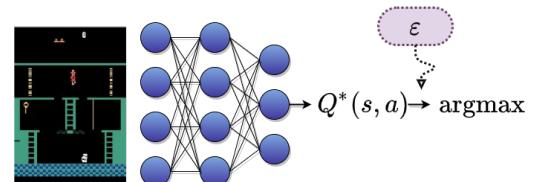
Нюанс: авторы статьи эмпирически обнаружили, что замена максимума на среднее даёт чуть лучшие результаты. В результате, на текущий день под дуэльной архитектурой понимают альтернативную формулу:

$$Q^*(s, a, \theta) = V^*(s, \theta) + A^*(s, a, \theta) - \frac{1}{|\mathcal{A}|} \sum_{\hat{a}} A^*(s, \hat{a}, \theta) \quad (4.4)$$

4.2.5. Шумные сети (Noisy Nets)

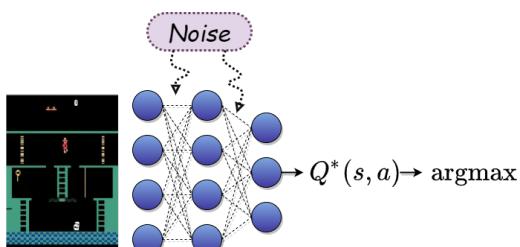
По дефолту, алгоритмы на основе DQN никак не пытаются решать дилемму исследования-использования и просто используют ϵ -жадную стратегию взаимодействия со средой. Этот бэйзлайн-подход плох примерно всем, в первую очередь тем, что крайне чувствителен к гиперпараметрам: раннее затухание приведёт к застrevанию алгоритма в локальном оптимуме (агент будет биться головой об стенкой, не пробуя её обойти), а слишком большие значения существенно замедляют обучение, заставляя агента вести себя случайно.

Ключевая причина, почему ϵ -жадная стратегия примитива, заключается в независимости добавляемого шума от текущего состояния. Мы выдаём оценки Q-функции и в зависимости только от времени принимаем решение, использовать ли эти знания или эксплорить. Интуитивно, правильнее было бы принимать это решение в зависимости от текущего состояния: если состояние исследовано, чаще принимать решение в пользу использования знаний, если ново — в пользу исследования. Открытие новой области пространства состояний скорее всего означает, что в ней стоит поделать разные действия, когда двигаться к ней нужно за счёт использования уже накопленных знаний.



Шумные сети (noisy nets) — добавление шума с обучаемой и, главное, зависимой от состояния (входа в модель) дисперсией. Как чисто инженерный: давайте каждый параметр в модели заменим на

$$\theta_i := w_i + \sigma_i \varepsilon_i \quad \varepsilon_i \sim \mathcal{N}(0, 1),$$



или, другими словами, заменим веса сети на сэмплы из $\mathcal{N}(w_i, \sigma_i^2)$, где $w, \sigma \in \mathbb{R}^h$ — параметры модели, обучаемые градиентным спуском. Очевидно, что выход сети становится случайной величиной, и, в зависимости от шума ε , будет меняться выбор действия $a = \arg\max_a Q^*(s, a, \theta, \varepsilon)$. При этом влияние шума на принятное решение зависит от состояния.



Если состояния — изображения, шум в свёрточные слои обычно не добавляется (зашумлять выделение объектов из изображения кажется бессмысленным).

Формально, коли наша модель стала стохастичной, мы поменяли оптимизируемый функционал: мы хотим минимизировать функцию потерь в среднем по шуму:

$$\mathbb{E}_\varepsilon \text{Loss}(\theta, \varepsilon) \rightarrow \min_{\theta}$$

Видно, что градиент такого функционала можно несмешённо оценивать по Монте-Карло:

$$\nabla_\theta \mathbb{E}_\varepsilon \text{Loss}(\theta, \varepsilon) = \mathbb{E}_\varepsilon \nabla_\theta \text{Loss}(\theta, \varepsilon) \approx \nabla_\theta \text{Loss}(\theta, \varepsilon) \quad \varepsilon \sim \mathcal{N}(0, I)$$

Гарантий, что магнитуда шума в среднем будет падать для исследованных состояний, вообще говоря, нет. Надежда этой идеи в том, что магнитуда будет подстраиваться в зависимости от текущих в модель градиентов, при этом гиперпараметры в подходе отсутствуют.



Кроме инициализации. Неудачная инициализация σ всё равно может замедлить процесс обучения; обычно дисперсию шума инициализируют какой-нибудь константой, и эта константа становится в некотором смысле важным гиперпараметром алгоритма.



Генерация сэмплов шума по числу параметров нейросети на видеокарте может сильно замедлить время прохода через сеть. Для оптимизации авторы предлагают для матриц полно связанных слоёв генерировать шум следующим образом. Пусть n — число входов, m — число выходов в слое. Сэмплируются $\varepsilon_1 \sim \mathcal{N}(0, I_{m \times m})$, $\varepsilon_2 \sim \mathcal{N}(0, I_{n \times n})$, после чего полагается шум для матрицы равным

$$\varepsilon := f(\varepsilon_1) f(\varepsilon_2)^T$$

где f — масштабирующая функция, например $f(x) = \text{sign}(x) \sqrt{|x|}$ (чтобы каждый сэмпл в среднем всё ещё имел дисперсию 1). Процедура требует всего $m + n$ сэмплов вместо mn , но жертвует независимостью сэмплов внутри слоя.



Альтернативно, можно зашумлять выходы слоёв (тогда сэмплов понадобится на порядок меньше) или просто добавлять шум на вход. В обоих случаях, «зашумлённость» выхода будет обучаемой, а степень влияния шума на выход сети будет зависеть от состояния.

Заметим, что таргет $y(\mathbb{T})$, который мы генерируем для каждого перехода \mathbb{T} из батча, формально теперь тоже должен вычисляться как матожидание по шуму:

$$y(\mathbb{T}) := \mathbb{E}_\varepsilon \left[r + \gamma \max_{a'} Q^*(s', a', \theta, \varepsilon) \right]$$

Опять же, матожидание несмешённо оценивается по Монте-Карло, однако с целью декорреляции полезно использовать в качестве ε другие сэмплы, нежели используемые при вычислении лосса.

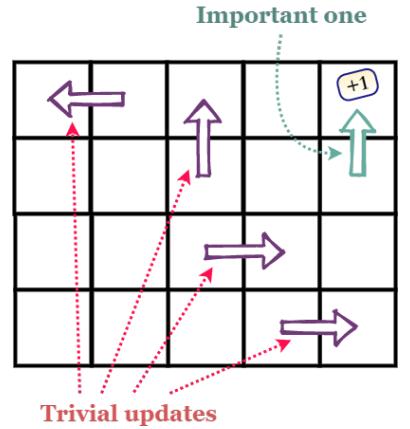
4.2.6. Приоритизированный реплей (Prioritized DQN)

Off-policy алгоритмы позволяют хранить и переиспользовать весь накопленный опыт. Однако, интуитивно ясно, что встречавшиеся переходы существенно различаются по важности. Зачастую большая часть буфера, особенно поначалу обучения, состоит из записей изучения агентом ближайшей стенки, а переходы, включавшие, например, получение ещё не выученной внутри аппроксимации Q-функции награды, встречаются в буфере сильно реже и при равномерном сэмплировании редко оказываются в мини-батчах.

Важно, что при обучении оценочных функций информация о награде распространяется от последних состояний к первым. Например, на первых итерациях довольно бессмысленно обновлять те состояния, где сигнала награды не было ($r(s, a) = 0$), а Q-функция для следующего состояния примерно случайна (а именно такие переходы чаще всего и попадаются алгоритму). Такие обновления лишь схлопывают выход аппроксимации к константе (которая ещё и имеет тенденцию к росту из-за оператора максимума). Ценной информацией поначалу являются терминальные состояния, где целевая переменная по определению равна $y(\mathbb{T}) = r(s, a)$ и является абсолютно точным значением $Q^*(s, a)$. Типично, что на таких переходах значения временной разницы (лосса DQN) довольно высоко. Аналогичная ситуация в принципе справедлива для любых наград, которые для агента новы и ещё не распространялись в аппроксимацию через уравнение Беллмана.

Очень хочется сэмплировать переходы из буфера не равномерно, а приоритизировано. Приоритет установим, например, следующим образом:

$$\rho(\mathbb{T}) := (y(\mathbb{T}) - Q^*(s, a, \theta))^2 = \text{Loss}(y(\mathbb{T}), Q^*(s, a, \theta)) \tag{4.5}$$



Сэмплирование переходов из буфера происходит по следующему правилу:

$$\mathbf{P}(\mathbb{T}) \propto \rho(\mathbb{T})^\alpha$$

где гиперпараметр $\alpha > 0$ контролирует масштаб приоритетов (в частности, $\alpha = 0$ соответствует равномерному сэмплированию, когда $\alpha \rightarrow +\infty$ соответствовало бы жадному сэмплированию самых «важных» переходов).



Добиться эффективного сэмплирования с приоритетами можно благодаря структуре данных SumTree: бинарному дереву, у которого в каждом узле хранится сумма значений в двух детях. Сам массив приоритетов для буфера хранится на нижнем уровне дерева; в корне, соответственно, лежит сумма всех приоритетов, нормировочная константа. Для сэмплирования достаточно взять случайную равномерную величину и спуститься по дереву. Таким образом, процедура сэмплирования имеет сложность $O(\log M)$, где M — размер буфера. За ту же сложность проходит обновление приоритета одного элемента, для чего достаточно обновить значения во всех его предках для поддержания структуры.

Техническая проблема идеи (4.5) заключается в том, что после каждого обновления весов сети θ приоритеты переходов меняются для всего буфера (состоящего обычно из миллионов переходов). Пересчитывать все приоритеты, конечно же, непрактично, и необходимо ввести некоторые упрощения. Например, можно обновлять приоритеты только у переходов из текущего батча, для которых значение лосса так и так считается. Это, вообще говоря, означает, что если у перехода был низкий приоритет, и до него дошла, условно, «волна распространения» награды, алгоритм не узнает об этом, пока не засэмплирует переход с тем приоритетом, который у него был.

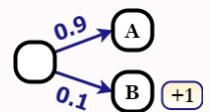


Новые переходы добавляются в буфер с наивысшим приоритетом $\max_{\mathbb{T}} \rho(\mathbb{T})$, который можно поддерживать за константу, или вычислять текущий приоритет, дополнительно рассчитывая (4.5) для онлайн-сэмплов.

В чём у такого подхода есть фундаментальная проблема? После неконтролируемой замены равномерного сэмплирования на какое-то другое, могло случиться так, что для наших переходов $s' \not\sim p(s' | s, a)$. Почему это так, проще понять на примере.

Пример 58: Пусть для данной пары s, a с вероятностью 0.1 мы попадаем в $s' = A$, а с вероятностью 0.9 мы попадаем в $s' = B$. В условно бесконечном буфере для этой пары s, a среди каждого 10 сэмплов будет 1 сэмпл с $s' = A$ и 9 сэмплов с $s' = B$, и равномерное сэмплирование давало бы переходы, удовлетворяющие $s' \sim p(s' | s, a)$.

Для приоритизированного реплея, веса у переходов с $s' = A$ могут отличаться от весов для переходов с $s' = B$. Например, если истинное значение $V^*(sA) = 1, V^*(sB) = 1$, и мы уже выучили правильное значение $Q^*(s, a) = 0.1$, то $\text{Loss}(s, a, s' = A)$ будет равен 0.1^2 , а для $\text{Loss}(s, a, s' = B) = 0.9^2$. Значит, s'_1 может появиться в переходах чаще или реже, чем с вероятностью 0.1, и это выбьет $Q(s, a)$ с её правильного значения.



Иными словами, приоритизированное сэмплирование приводит к **смещению** (bias). Этот эффект не так страшен поначалу обучения, когда распределение, из которого приходят состояния, всё равно скорее всего не сильно разнообразно. Более существенно нивелировать этот эффект по ходу обучения, в противном случае процесс обучения может полностью дестабилизироваться или где-нибудь застрять.

Заметим, что равномерное сэмплирование не является единственным «корректным» способом, но основным доступным. Мы не очень хотим «возвращаться» к нему постепенно с ходом обучения, но можем сделать похожую вещь: раз мы хотим подменить распределение, то можем при помощи `importance sampling` сохранить тот же оптимизируемый функционал:

Теорема 32: При сэмплировании с приоритетами $\mathbf{P}(\mathbb{T})$ использование весов $w(\mathbb{T}) := \frac{1}{\mathbf{P}(\mathbb{T})}$ позволит избежать эффекта смещения.

Доказательство. Пусть M — размер буфера.

$$\begin{aligned} \mathbb{E}_{\mathbb{T} \sim Uniform} \text{Loss}(\mathbb{T}) &= \sum_{i=1}^M \frac{1}{M} \text{Loss}(\mathbb{T}_i) = \\ &= \sum_{i=1}^M \mathbf{P}(\mathbb{T}_i) \frac{1}{M \mathbf{P}(\mathbb{T}_i)} \text{Loss}(\mathbb{T}_i) = \\ &= \mathbb{E}_{\mathbb{T} \sim \mathbf{P}(\mathbb{T})} \frac{1}{M \mathbf{P}(\mathbb{T})} \text{Loss}(\mathbb{T}), \end{aligned}$$

что с точностью до константы $\frac{1}{M}$ и есть перевзвешивание функции потерь. ■

Importance sampling подразумевает, что мы берём «интересные» переходы, но делаем по ним меньшие шаги (веса меньше именно для «приоритетных» переходов). Цена за такую корректировку, конечно, в том, что полезность приоритизированного сэмплирования понижается. Раз поначалу смещение нас не так беспокоит, предлагается вводить веса постепенно: а именно, использовать веса

$$w(\mathbb{T}) := \frac{1}{P(\mathbb{T})^{\beta(t)}}$$

где $\beta(t)$ — гиперпараметр, зависящий от итерации алгоритма t . Изначально $\beta(t=0) = 0$, что делает веса равномерными (корректировки не производится), но постепенно $\beta(t)$ растёт к 1 и полностью избавляет алгоритм от эффекта смещения.



На практике веса, посчитанные по такой формуле, могут оказаться очень маленькими или большими, и их следует нормировать. Вариации, как нормировать, различаются в реализациях: можно делить веса на $\max_{\mathbb{T}} w(\mathbb{T})$, где максимум берётся, например, только по текущему мини-батчу, чтобы гарантировать максимальный вес 1.

4.2.7. Multi-step DQN

DQN страдает от проблемы *накапливающейся ошибки* (compound error). Условно говоря, чтобы распространить награду, полученную в некоторый момент времени, на 100 шагов в прошлое, понадобится пройти 100 этапов метода простой итерации. Каждый этап мы решаем задачу регрессии в сильно неидеальных условиях, и ошибка аппроксимации накапливается.

Эта проблема, как мы увидим далее, фундаментальна для off-policy подхода. *Многошаговый* (Multi-step) — теоретически некорректная эвристика для небольшого занижения этого эффекта. Грубо говоря, нам очень хочется распространять за одну итерацию награду сразу на несколько шагов вперёд, то есть решать многошаговые уравнения Беллмана (3.23). Мы как бы и можем уравнение оптимальности многошаговое выписать...

Утверждение 35 — N -шаговое уравнение оптимальности Беллмана:

$$Q^*(s_0, a_0) = \mathbb{E}_{\mathcal{T}_{:N} \sim \pi^* | s_0, a_0} \left[\sum_{t=0}^{N-1} \gamma^t r_t + \gamma^N \mathbb{E}_{s_N} \max_{a_N^*} Q^*(s_N, a_N^*) \right] \quad (4.6)$$

Что мы можем сделать? Мы можем прикинуться, будто решаем многошаговые уравнения Беллмана, задав целевую переменную следующим образом:

$$y(s_0, a_0) := \sum_{t=0}^{N-1} \gamma^t r_t + \gamma^N \max_{a_N} Q^*(s_N, a_N, \theta) \quad (4.7)$$

где $s_1, a_1 \dots a_{N-1}, s_N$ взяты из буфера. Для этого в буфере вместо одношаговых переходов $\mathbb{T} := (s, a, r, s', \text{done})$ достаточно просто хранить другую пятёрку:

$$\mathbb{T} := \left(s, a, \sum_{n=0}^{N-1} \gamma^n r^{(n)}, s^{(N)}, \text{done} \right)$$

где $r^{(n)}$ — награда, полученная через n шагов после посещения рассматриваемого состояния s , $s^{(N)}$ — состояние, посещённое через N шагов, и, что важно, флаг `done` указывает на то, завершился ли эпизод в течение этого N -шагового роллаута³. Все остальные элементы алгоритма не изменяются, в частности, можно видеть, что случай $N = 1$ соответствует обычному DQN.

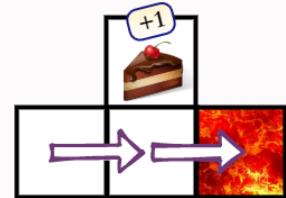
Видно, что теперь награда, полученная за один шаг, распространяется на N состояний в прошлое, и мы таким образом не только ускоряем обучение оценочной функции стартовых состояний, но и нивелируем проблему накапливающейся ошибки.

Почему теоретически это некорректно? Беря $s_1, a_1 \dots a_{N-1}, s_N$ из буфера, мы получаем состояния из функции переходов, которая стационарна и соответствует тому мат.ожиданию, которое стоит в уравнении (4.6). Но вот действия в этом мат.ожидании должны приходить из оптимальной стратегии! А в буфере $a_1, a_2 \dots a_{N-1}$ — действия нашей стратегии произвольной давности (то есть сколь угодно неоптимальные). Вместо того, чтобы оценивать оптимальное поведение за хвост траектории (по определению Q^*),

³естественно, алгоритм должен рассматривать все N -шаговые роллауты, включая те, которые завершились за $k < N$ шагов. Для них, естественно, $r^{(k')} = 0$ для $k' > k$, и $Q^*(s^{(N)}, a_N) \equiv 0$ для всех a_N .

мы $N - 1$ шагов ведём себя сколько угодно неоптимально, а затем в $s^{(N)}$ подставляем оценку оптимального поведения за хвост. Иными словами, мы недооцениваем истинное значение правой части N -шагового уравнения Беллмана при $N > 1$. Вместо уравнения оптимальности мы решаем такое уравнение: что, если я следующие N шагов веду себя как стратегия π , когда-то породившая данный роллаут, и только потом сберусь вести себя оптимально? Причём из-за нашего желания делать так в off-policy режиме π для каждого перехода своё, то есть схеме Generalized Policy Iteration это не соответствует.

Пример 59: Пример, когда многошаговая оценка приводит к некорректным алгоритмам. На втором шаге игры в s_1 агент может скушать тортик или прыгнуть в лаву, и в первом эпизоде обучения агент совершил ошибку и получил огромную негативную награду. В буфер при $N > 1$ запишется пример со стартовым состоянием s_0 и этой большой отрицательной наградой (в качестве $s^{(N)}$ будет записана лава). Пока этот пример живёт в реплей буфере, каждый раз, когда он сэмплируется в мини-батче, оценочная функция для s_0 обновляется этой отрицательной наградой, даже если агент уже научился больше не совершать эту ошибку и в s_1 наслаждается тортиками.



Эмпирически большое значение N действительно может полностью дестабилизировать процесс, как и подсказывает теория, поэтому рекомендуется выставлять небольшие значение 2-3, от силы 5. Большие значения могут быть работоспособны в средах, где сколько угодно неоптимальное поведение в течение N шагов не приводит к существенному изменению награды по сравнению с оптимальным поведением, то есть в средах, где нет моментов с «критическими решениями».

ГЛАВА А

Приложение

§A.1. Натуральный градиент

A.1.1. Проблема параметризации

Стандартная градиентная оптимизация страдает от зависимости от параметризации. Допустим, мы градиентно оптимизируем

$$f(x) \rightarrow \min_x,$$

и решили сменить параметризацию на $y = y(x)$ (допустим, даже, биективным преобразованием); задача

$$f(y) \rightarrow \min_y,$$

казалось бы, эквивалентна, но траектории градиентного спуска не только будут кардинально отличаться (при эквивалентной инициализации, x_0 для первой задачи и $y_0 = y_0(x_0)$ для второй), но и могут сильно различаться по свойствам (в одной параметризации оптимизация проходит намного успешнее другой).

Для нас обычно проблема закопана ещё чуть глубже. Оптимизируемые нами функционалы обычно имеют такой вид:

$$f(\phi) := f(q(x | \phi)) \rightarrow \min_{\phi} \quad (\text{A.1})$$

где $q(x | \phi)$ — некоторое параметрически заданное распределение, и функционал F зависит только непосредственно от самого распределения. В качестве такого функционала обычно выступает или логарифм правдоподобия, или функция потерь для задач машинного обучения, также такой вид имеет вспомогательная функция в эволюционных стратегиях (2.4) и, самое главное, так выглядит и наш главный оптимизируемый функционал в обучении с подкреплением (1.5).

Вспомним, как устроен градиентный спуск. Мы рассматриваем некоторую окрестность точки ϕ_0 и хотим, основываясь на локальных свойствах функции, выбрать направление для изменения текущих параметров. Для этого функция $f(\phi)$ раскладывается в ряд Тейлора до первого члена с центром в точке ϕ_0 , и это разложение используется как приближённая модель поведения функции:

$$\begin{cases} f(\phi) \approx f(\phi_0) + \langle \nabla_{\phi} f(\phi)|_{\phi=\phi_0}, \phi - \phi_0 \rangle \rightarrow \min_{\phi} \\ \|\phi - \phi_0\|_2^2 < \alpha \end{cases}$$

где α — learning rate, а условие $\|\phi - \phi_0\|_2^2 < \alpha$ задаёт «**регион доверия**» (trust-region) к построенному приближению. Понятие региона требует понятие расстояния: мы готовы отойти от текущей точки ϕ_0 не далее чем на α в смысле обычного Евклидова расстояния. Аналитическое решение задачи даёт стандартную формулу градиентного спуска:

$$\phi - \phi_0 \propto -\nabla_{\phi} f(\phi)|_{\phi=\phi_0}$$

Однако, в случае функционала вида (A.1) параметризации $q(x | \phi)$ могут быть таковы, что небольшие изменения параметров ϕ могут радикально сильно поменять само распределение $q(x | \phi)$, а значит, и значение $f(q(x | \phi))$; и наоборот, для внесения каких-то небольших изменений в $q(x | \phi)$ необходимо поменять ϕ достаточно сильно в смысле Евклидова расстояния.

Пример 60: $\mathcal{N}(0, 100)$ похож $\mathcal{N}(1, 100)$, когда $\mathcal{N}(0, 0.1)$ совсем не похоже на $\mathcal{N}(1, 0.1)$; при этом для обоих пар евклидово расстояние между значениями параметров равно единице.

A.1.2. Матрица Фишера

Оптимизация при помощи натурального градиента предлагает использовать другую метрику, которая учитёт структуру нашего функционала:

$$\begin{cases} f(\phi) \approx f(\phi_0) + \langle \nabla_\phi f(\phi)|_{\phi=\phi_0}, \phi - \phi_0 \rangle \rightarrow \min_\phi \\ \text{KL}(q(x | \phi_0) \| q(x | \phi)) < \alpha \end{cases}$$

Как решать такую задачу условной оптимизации? Если $\phi \approx \phi_0$, достаточно аппроксимировать дивергенцию $\text{KL}(q(x | \phi_0) \| q(x | \phi))$ при помощи разложения в ряд Тейлора до второго члена. До второго — потому что первое ноль.

Утверждение 36:

$$\nabla_\phi \text{KL}(q(x | \phi_0) \| q(x | \phi))|_{\phi=\phi_0} = 0$$

Доказательство. **KL**-дивергенция в точке $\phi = \phi_0$ равна 0 как дивергенция между одинаковыми распределениями, следовательно как функция от ϕ она достигает в этой точке глобального минимума \Rightarrow градиент равен нулю. ■

Определение 57: Для распределения $q(x | \phi)$ **матрицей Фишера** (Fisher matrix) называется

$$F_q(\phi) := -\mathbb{E}_{q(x|\phi)} \nabla_\phi^2 \log q(x | \phi)$$

Теорема 33: Матрица Фишера есть гессиан **KL**-дивергенции:

$$\nabla_\phi^2 \text{KL}(q(x | \phi_0) \| q(x | \phi))|_{\phi=\phi_0} = F_q(\phi_0)$$

Доказательство.

$$\nabla_\phi^2 \text{KL}(q(x | \phi_0) \| q(x | \phi))|_{\phi=\phi_0} = \nabla_\phi^2 [\text{const}(\phi) - \mathbb{E}_{q(x|\phi_0)} \log q(x | \phi)]|_{\phi=\phi_0} = F_q(\phi_0) \quad \blacksquare$$

Прежде чем двинуться дальше, остановимся на паре важных для нас свойств матрицы Фишера.

Теорема 34 — Эквивалентное определение матрицы Фишера:

$$F_q(\phi) := \mathbb{E}_{q(x|\phi)} \nabla_\phi \log q(x | \phi) (\nabla_\phi \log q(x | \phi))^T \quad (\text{A.2})$$

Доказательство.

$$\begin{aligned} F_q(\phi) &= -\mathbb{E}_{q(x|\phi)} \nabla_\phi \nabla_\phi \log q(x | \phi) = \\ &= \{\text{градиент логарифма}\} = -\mathbb{E}_{q(x|\phi)} \nabla_\phi \frac{\nabla_\phi q(x | \phi)}{q(x | \phi)} = \\ &= \{\text{градиент отношения}\} = -\mathbb{E}_{q(x|\phi)} \frac{\nabla_\phi^2 q(x | \phi)}{q(x | \phi)} + \mathbb{E}_{q(x|\phi)} \frac{\nabla_\phi q(x | \phi) \nabla_\phi q(x | \phi)^T}{q(x | \phi)^2} = (*) \end{aligned}$$

Заметим, что первое слагаемое равно нулю. Действительно:

$$\mathbb{E}_{q(x|\phi)} \frac{\nabla_\phi^2 q(x | \phi)}{q(x | \phi)} = \int_x \nabla_\phi^2 q(x | \phi) dx = \nabla_\phi^2 \int_x q(x | \phi) dx = \nabla_\phi^2 1 = 0$$

В оставшемся втором слагаемом перегруппируем множители (заметим, что в знаменатели дроби стоит скаляр):

$$\begin{aligned} (*) &= \mathbb{E}_{q(x|\phi)} \frac{\nabla_\phi q(x | \phi)}{q(x | \phi)} \left(\frac{\nabla_\phi q(x | \phi)}{q(x | \phi)} \right)^T = \\ &= \{\text{градиент логарифма}\} = \mathbb{E}_{q(x|\phi)} \nabla_\phi \log q(x | \phi) (\nabla_\phi \log q(x | \phi))^T \end{aligned} \quad \blacksquare$$

Утверждение 37: Любая матрица Фишера симметрична и положительно определена.

Теорема 35 — Репараметризация матрицы Фишера:

Пусть одно распределение параметризовано двумя способами, а то есть $q_\phi(x | \phi) \equiv q_\nu(x | \nu)$, где $\nu = \nu(\phi)$ — преобразование с якобианом¹ J . Тогда:

$$F_q(\phi) = J F_q(\nu) J^T \quad (\text{A.3})$$

Доказательство.

$$\begin{aligned} F_q(\phi) &= \mathbb{E}_{q_\phi(x|\phi)} (\nabla_\phi \log q(x|\phi)) (\nabla_\phi \log q(x|\phi))^T = \\ &= \{q_\phi(x|\phi) \equiv q_\nu(x|\nu)\} = \mathbb{E}_{q_\nu(x|\nu)} (\nabla_\phi \log q(x|\nu)) (\nabla_\phi \log q(x|\nu))^T = \\ &= \{\text{chain rule}\} = \mathbb{E}_{q_\phi(x|\phi)} J \nabla_\nu \log q(x|\nu) (\nabla_\nu \log q(x|\nu))^T J^T = \\ &= J F_q(\nu) J^T \end{aligned}$$

¹будем считать, что якобиан имеет размер $h_\phi \times h_\nu$, где h_ν, h_ϕ — размерности ν и ϕ соответственно. ■

A.1.3. Натуральный градиент

Итак, приближённая задача выглядит следующим образом:

$$\begin{cases} f(\phi) \approx f(\phi_0) + \langle \nabla_\phi f(\phi)|_{\phi=\phi_0}, \phi - \phi_0 \rangle \rightarrow \min_\phi \\ (\phi - \phi_0)^T F_q(\phi_0)(\phi - \phi_0) < \alpha \end{cases}$$

По сути, матрица Фишера задаёт нам приближённо метрику², индуцированную **KL**-дивергенцией. Такая задача также решается аналитично и получается приятный ответ:

$$\phi - \phi_0 \propto -F_q(\phi_0)^{-1} \nabla_\phi f(\phi)|_{\phi=\phi_0} \quad (\text{A.4})$$

Определение 58: *Натуральным градиентом* (natural gradient) функции $f(\phi) := f(q(x|\phi))$ называется

$$\tilde{\nabla}_\phi f(\phi) := F_q(\phi)^{-1} \nabla_\phi f(\phi)$$

Итак, натуральный градиент есть скорректированный матрицей Фишера обычный градиент. Он очень напоминает методы оптимизации второго порядка, так как происходит учёт вида оптимизируемой функции, в том числе масштабирование по осям.

Теорема 36 — Инвариантность натурального градиента относительно параметризации: Пусть одно распределение параметризовано двумя способами, а то есть $q_\phi(x | \phi) \equiv q_\nu(x | \nu)$, где $\nu = \nu(\phi)$ — преобразование с якобианом J .

Тогда натуральные градиенты указывают в одном и том же направлении:

$$\tilde{\nabla}_\phi f(\nu) = J^T \tilde{\nabla}_\phi f(\phi)$$

Доказательство.

$$\begin{aligned} J^T \tilde{\nabla}_\phi f(\phi) &= J^T F_q(\phi)^{-1} \nabla_\phi f(\phi) = \\ &= \{\text{репараметризация матрицы Фишера (A.3)}\} = J^T (J F_q(\nu) J^T)^{-1} \nabla_\phi f(\phi) \\ &= \{\text{свойства обратной матрицы}\} = J^T J^{-T} F_q(\nu)^{-1} J^{-1} \nabla_\phi f(\phi) \\ &= \{\text{chain rule}\} = F_q(\nu)^{-1} J^{-1} J \nabla_\nu f(\nu) = \\ &= F_q(\nu)^{-1} \nabla_\nu f(\nu) = \\ &= \tilde{\nabla}_\nu f(\nu) \end{aligned}$$

■

²А точнее, **Римановскую метрику**. В Евклидовом пространстве общей формой скалярного произведения является $\langle x, y \rangle := x^T G y$, где G — некоторая фиксированная положительно определённая матрица, и индуцированной метрикой соответственно является $d(x, y)^2 := (y - x)^T G (y - x)$. В Римановском пространстве G , называемая также **метрическим тензором** (metric tensor), зависит от x , и относительное расстояние между точками зависит от положения в пространстве. Римановские пространства используются для описания расстояний между точками на поверхностях, а метрические тензоры для них обладают рядом приятных свойств, которыми, в частности, обладает и матрица Фишера.

§A.2. Обоснование формул CMA-ES

В данном разделе мы вычислим формулу натурального градиента для оптимизации вспомогательного функционала

$$g(\lambda) := \mathbb{E}_{\theta \sim q(\theta|\lambda)} J(\theta) \rightarrow \max_{\lambda}, \quad (\text{A.5})$$

для эволюционной стратегии $q(\theta | \lambda) := \mathcal{N}(\mu, \Sigma)$ с параметрами $\lambda := (\mu, \Sigma)$.

A.2.1. Вычисление градиента

Сначала нам придётся напрячься и продифференцировать логарифм правдоподобия по параметрам μ и Σ :

$$\log q(\theta | \mu, \Sigma) = \text{const}(\mu, \Sigma) - \frac{1}{2} \log \det \Sigma - \frac{1}{2} (\theta - \mu)^T \Sigma^{-1} (\theta - \mu)$$

Поскольку нам придётся дифференцировать функционал по матрице, вычислять градиенты удобно в терминах дифференциала. Будем использовать следующую нотацию: будем обозначать за $D_x f(x)[h]$ дифференциал функции $f(x)$ по переменной x с приращением h . Дифференциал имеет ту же размерность, что и выход функции f , что и делает его использование таким удобным. В итоге мы должны получить линейную часть приращения $f(x)$; так, в итоге вычислений мы получим представление дифференциала в виде $D_x f(x)[h] = \langle \nabla_x f(x), h \rangle$ и сможем «вытащить» градиент.

Утверждение 38:

$$\nabla_{\mu} \log q(\theta | \mu, \Sigma) = \Sigma^{-1} (\theta - \mu)$$

Доказательство.

$$D_{\mu} \log q(\theta | \mu, \Sigma)[h] = \frac{1}{2} h^T \Sigma^{-1} (\theta - \mu) + \frac{1}{2} (\theta - \mu)^T \Sigma^{-1} h = \langle \Sigma^{-1} (\theta - \mu), h \rangle$$

Отсюда получаем градиент как вектор, скалярно перемножаемый на приращение $h: \Sigma^{-1} (\theta - \mu)$. ■

Для Σ нам понадобится пара табличных формул векторного дифференцирования (здесь Tr — оператор взятия [следа матрицы](#)):

$$D_{\Sigma} \log \det \Sigma[H] = \text{Tr}(\Sigma^{-1} H) \quad (\text{A.6})$$

$$D_{\Sigma} \Sigma^{-1}[H] = -\Sigma^{-1} H \Sigma^{-1} \quad (\text{A.7})$$

Утверждение 39:

$$\nabla_{\Sigma} \log q(\theta | \mu, \Sigma) = -\frac{1}{2} \Sigma^{-1} + \frac{1}{2} \Sigma^{-1} (\theta - \mu) (\theta - \mu)^T \Sigma^{-1}$$

Доказательство.

$$\begin{aligned} D_{\Sigma} \log q(\theta | \mu, \Sigma)[H] &= -\frac{1}{2} D_{\Sigma} \log \det \Sigma[H] - \frac{1}{2} (\theta - \mu)^T D_{\Sigma} \Sigma^{-1}[H] (\theta - \mu) \\ &= \{\text{подставляем формулы (A.6) и (A.7)}\} = -\frac{1}{2} \text{Tr}(\Sigma^{-1} H) + \frac{1}{2} (\theta - \mu)^T \Sigma^{-1} H \Sigma^{-1} (\theta - \mu) \\ &= \{\text{фокус: если } v \text{ — скаляр, } v = \text{Tr}(v)\} = -\frac{1}{2} \text{Tr}(\Sigma^{-1} H) + \frac{1}{2} \text{Tr}((\theta - \mu)^T \Sigma^{-1} H \Sigma^{-1} (\theta - \mu)) = \\ &= \{\text{тождество } \text{Tr}(ABC) = \text{Tr}(BCA)\} = -\frac{1}{2} \text{Tr}(\Sigma^{-1} H) + \frac{1}{2} \text{Tr}(\Sigma^{-1} (\theta - \mu) (\theta - \mu)^T \Sigma^{-1} H) \end{aligned}$$

Дифференциал — это скалярное произведение градиента на приращение H , которое в пространстве матриц задано оператором Tr . ■

Из этих формул можно увидеть, почему градиентный подъём для оптимизации (A.5) по μ, Σ не так хорош. Допустим, матрица ковариации адаптировалась так, что вдоль оси θ_0 разброс большой. Пусть где-то справа нашлись особи с огромным $J(\theta)$; тогда взвешенное среднее

$$\frac{1}{N} \sum_{\theta \in \mathcal{P}} J(\theta) \theta$$

будет указывать примерно в центр этого скопления, будет говорить сильно увеличить компоненту θ_0 . Однако, формула градиента говорит сделать поправку на матрицу ковариации: мол, мы генерировали вдоль θ_0 с большим разбросом, и поэтому вдоль этой оси градиент нужно пропорционально сбить. В

итоге, к центру скопления делается куда меньший шаг, чем по идее должен был бы по итогам генерации целого поколения особей. Примерно тоже самое наблюдается с матрицей ковариации: в формуле также делается поправка на текущее значение матрицы ковариации (умножение на Σ^{-1} с двух сторон) вместо движения к эмпирической (взвешанной на $\hat{J}(\theta)$) матрице.

A.2.2. Произведение Кронекера

Нам понадобится работать с матрицей Фишера, которая имеет размерность «количество параметров на количество параметров». Иными словами, нам понадобится сравнивать попарно между собой все элементы матрицы Σ , а это значит, нам придётся чуть-чуть залезть в алгебру Кронекера. Для этого мы введём операцию *векторизация матрицы vec*, вытягивающей все элементы матрицы в вектор.

Утверждение 40: Векторизация — линейная операция; поэтому, в частности:

$$\mathbb{E}_\theta \text{vec}(f(\theta)) = \text{vec}(\mathbb{E}_\theta f(\theta)) \quad (\text{A.8})$$

$$\nabla_{\text{vec}(\Sigma)} f(\Sigma) = \text{vec}(\nabla_\Sigma f(\Sigma)) \quad (\text{A.9})$$

Доказательство. Можно проверить непосредственно: мы просто переписываем все элементы матрицы в другой форме, сложению и умножению на скаляры всё равно. ■

Определение 59: Произведением Кронекера двух матриц $A \in \mathbb{R}^{n \times m}$, $B \in \mathbb{R}^{p \times q}$ называется

$$A \otimes B := \begin{pmatrix} a_{11}B & \dots & a_{1m}B \\ \vdots & \ddots & \vdots \\ a_{n1}B & \dots & a_{nm}B \end{pmatrix} \in \mathbb{R}^{np \times mq},$$

где a_{ij} — элементы матрицы A .

Пример 61:

$$\begin{aligned} \begin{pmatrix} 3 & 0 \\ 1 & -2 \end{pmatrix} \otimes \begin{pmatrix} 4 & 0 & -1 \\ -4 & 2 & 5 \end{pmatrix} &= \begin{pmatrix} 3 \begin{pmatrix} 4 & 0 & -1 \\ -4 & 2 & 5 \end{pmatrix} & 0 \begin{pmatrix} 4 & 0 & -1 \\ -4 & 2 & 5 \end{pmatrix} \\ 1 \begin{pmatrix} 4 & 0 & -1 \\ -4 & 2 & 5 \end{pmatrix} & -2 \begin{pmatrix} 4 & 0 & -1 \\ -4 & 2 & 5 \end{pmatrix} \end{pmatrix} = \\ &= \begin{pmatrix} 12 & 0 & -3 & 0 & 0 & 0 \\ -12 & 6 & 15 & 0 & 0 & 0 \\ 4 & 0 & -1 & -8 & 0 & 2 \\ -4 & 2 & 5 & 8 & -4 & -10 \end{pmatrix} \end{aligned}$$

Теорема 37: Для матриц A, B, C, D с размерностями, для которых существует произведения AC и BD , верно:

$$(A \otimes B)(C \otimes D) = AC \otimes BD \quad (\text{A.10})$$

Доказательство. Проверяется непосредственно: пусть a_{ij} — элементы матрицы $A \in \mathbb{R}^{n \times m}$, c_{ij} — элементы матрицы $C \in \mathbb{R}^{m \times q}$, тогда:

$$\begin{aligned} (A \otimes B)(C \otimes D) &= \begin{pmatrix} a_{11}B & \dots & a_{1m}B \\ \vdots & \ddots & \vdots \\ a_{n1}B & \dots & a_{nm}B \end{pmatrix} \begin{pmatrix} c_{11}D & \dots & c_{1q}D \\ \vdots & \ddots & \vdots \\ c_{m1}D & \dots & c_{mq}D \end{pmatrix} \\ &= \begin{pmatrix} \sum_{k=1}^m a_{1k}Bc_{k1}D & \dots & \sum_{k=1}^m a_{1k}Bc_{kq}D \\ \vdots & \ddots & \vdots \\ \sum_{k=1}^m a_{nk}Bc_{k1}D & \dots & \sum_{k=1}^m a_{nk}Bc_{kq}D \end{pmatrix} \end{aligned}$$

Вводя обозначение $e_{ij} = \sum_{k=0}^m a_{ik}c_{kj}$, получаем Кронекерово произведение между матрицей E , состоящей из этих элементов, и матрицей BD . Осталось заметить, что матрица $E = AC$ по определению. ■

Утверждение 41: Для обратимых матриц A, B :

$$(A \otimes B)^{-1} = A^{-1} \otimes B^{-1} \quad (\text{A.11})$$

Пояснение.

$$(A^{-1} \otimes B^{-1})(A \otimes B) = A^{-1}A \otimes B^{-1}B = I \otimes I = I \quad \blacksquare$$

Теорема 38: Для матриц A, B, C , для которых существует произведение ABC , верно:

$$\text{vec}(ABC) = (C^T \otimes A) \text{vec}(B) \quad (\text{A.12})$$

Доказательство. Проверяется непосредственно. Пусть c_{ij} — элементы матрицы C , b_i — строки матрицы B . Тогда $\text{vec}(B) = (b_1, b_2 \dots b_m)^T$, и:

$$(C^T \otimes A) \text{vec}(B) = \begin{pmatrix} c_{11}A & \dots & c_{m1}A \\ \vdots & \ddots & \vdots \\ c_{1n}A & \dots & c_{mn}A \end{pmatrix} \begin{pmatrix} b_1^T \\ \vdots \\ b_m^T \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^m c_{i1}Ab_i^T \\ \vdots \\ \sum_{i=1}^m c_{in}Ab_i^T \end{pmatrix}$$

Осталось заметить, что $\sum_{i=1}^m c_{ij}Ab_i^T$ есть транспонированная j -ая строчка матрицы ABC . \blacksquare

A.2.3. Вычисление матрицы Фишера

Получение матрицы Фишера для нормального распределения представляет собой напряжное техническое упражнение. Будем далее считать, что наш набор параметров есть $\lambda := (\mu, \text{vec}(\Sigma))$; тогда матрица Фишера для нас будет матрицей, размера $(h + h^2) \times (h + h^2)$. Представим её в блочно-диагональном виде:

$$F_q(\lambda) = \begin{pmatrix} F_{\mu\mu} & F_{\mu \text{vec } \Sigma} \\ F_{\mu \text{vec } \Sigma}^T & F_{\text{vec } \Sigma \text{ vec } \Sigma} \end{pmatrix}$$

где $F_{\mu\mu} \in \mathbb{R}^{h \times h}$, $F_{\text{vec } \Sigma \text{ vec } \Sigma} \in \mathbb{R}^{h^2 \times h^2}$. Мы воспользовались симметричностью матрицы Фишера (что было видно из её эквивалентного определения (A.2)).

Будем искать эти блоки по одному. Для удобства введём обозначение

$$S := (\theta - \mu)(\theta - \mu)^T$$

и продублируем формулы градиентов логарифма правдоподобия, полученные в разделе A.2.1:

$$\nabla_\mu \log q(\theta | \mu, \Sigma) = \Sigma^{-1}(\theta - \mu) \quad (\text{A.13})$$

$$\nabla_\Sigma \log q(\theta | \mu, \Sigma) = -\frac{1}{2}\Sigma^{-1} + \frac{1}{2}\Sigma^{-1}S\Sigma^{-1} \quad (\text{A.14})$$

Заметим, что по свойствам нормального распределения:

$$\mathbb{E}_\theta \theta = \mu \quad \mathbb{E}_\theta S = \Sigma$$

Теорема 39: Матрица Фишера для нормального распределения имеет вид:

$$F_q(\lambda) = \begin{pmatrix} \Sigma^{-1} & 0 \\ 0 & \frac{1}{2}(\Sigma^{-1} \otimes \Sigma^{-1}) \end{pmatrix}$$

Доказательство для $F_{\mu\mu}$.

$$F_{\mu\mu} = -\mathbb{E}_\theta \nabla_\mu^2 \log q(\theta | \mu, \Sigma) = \{(\text{A.13})\} = -\mathbb{E}_\theta \nabla_\mu \Sigma^{-1}(\theta - \mu) = \mathbb{E}_\theta \Sigma^{-1} = \Sigma^{-1} \quad \blacksquare$$

Доказательство для $F_{\mu \text{vec } \Sigma}$. Сначала посчитаем вторую производную.

$$D_\Sigma \nabla_\mu \log q(\theta | \mu, \Sigma)[H] = \{(\text{A.13})\} = -D_\Sigma \Sigma^{-1}(\theta - \mu)[H] = \{(\text{A.7})\} = \Sigma^{-1}H\Sigma^{-1}(\theta - \mu)$$

Тут дальше есть проблема: нас, вообще говоря, интересует градиент. Однако производная вектора $\nabla_\mu \log q(\theta | \mu, \Sigma)$ по матрице это трёхмерный тензор, и надо танцевать с векторизацией. Мы схитрим:

давайте посмотрим на мат.ожидание дифференциала по θ :

$$-\mathbb{E}_\theta \Sigma^{-1} H \Sigma^{-1} (\theta - \mu) = -\Sigma^{-1} H \Sigma^{-1} \mathbb{E}_\theta (\theta - \mu) = 0$$

Как следствие, это нулевой тензор, и матрица Фишера тоже ноль. ■

Доказательство для $F_{\text{vec } \Sigma \mu}$ (Sanity check). Проверим, что и симметричный блок тоже ноль (формально это уже доказано в силу симметрии). Для этого нужно дифференцировать по μ первую производную по Σ (A.14). Сразу же будем смотреть на мат.ожидание этого выражения по θ :

$$-\mathbb{E}_\theta D_\mu \nabla_\Sigma \log q(\theta | \mu, \Sigma)[h] = \frac{1}{2} \Sigma^{-1} \mathbb{E}_\theta D_\mu S[h] \Sigma^{-1}$$

При этом $\mathbb{E}_\theta D_\mu S[h] = 0$, поскольку:

$$\mathbb{E}_\theta D_\mu S[h] = -\mathbb{E}_\theta h(\theta - \mu)^T - \mathbb{E}_\theta (\theta - \mu)h^T = 0$$

Доказательство для $F_{\text{vec } \Sigma \text{ vec } \Sigma}$.

$$\begin{aligned} & -\mathbb{E}_\theta D_\Sigma \nabla_{\text{vec } \Sigma} \log q(\theta | \mu, \Sigma)[H] = \\ &= -\mathbb{E}_\theta D_\Sigma \text{vec}(\nabla_\Sigma \log q(\theta | \mu, \Sigma)) [H] = \\ &= \{\text{подставляем (A.14)}\} = \\ &= -\mathbb{E}_\theta D_\Sigma \text{vec}\left(-\frac{1}{2} \Sigma^{-1} + \frac{1}{2} \Sigma^{-1} S \Sigma^{-1}\right) [H] = \\ &= \{\text{вносим минус, } D \text{ и } \mathbb{E}_\theta\} = \\ &= \text{vec}\left(\frac{1}{2} D_\Sigma \Sigma^{-1}[H] - \frac{1}{2} \mathbb{E}_\theta D_\Sigma (\Sigma^{-1} S \Sigma^{-1}) [H]\right) = \\ &= \{\text{дифф. билинейной функции}\} = \\ &= \text{vec}\left(\frac{1}{2} D_\Sigma \Sigma^{-1}[H] - \frac{1}{2} D_\Sigma \Sigma^{-1}[H] \mathbb{E}_\theta S \Sigma^{-1} - \frac{1}{2} \Sigma^{-1} \mathbb{E}_\theta S D_\Sigma \Sigma^{-1}[H]\right) = \\ &= \{\text{дифф. обратной матрицы (A.7)}\} = \\ &= \text{vec}\left(-\frac{1}{2} \Sigma^{-1} H \Sigma^{-1} + \frac{1}{2} \Sigma^{-1} H \Sigma^{-1} \mathbb{E}_\theta S \Sigma^{-1} + \frac{1}{2} \Sigma^{-1} \mathbb{E}_\theta S \Sigma^{-1} H \Sigma^{-1}\right) = \\ &= \{\mathbb{E}_\theta S = \Sigma\} = \text{vec}\left(-\frac{1}{2} \Sigma^{-1} H \Sigma^{-1} + \frac{1}{2} \Sigma^{-1} H \Sigma^{-1} + \frac{1}{2} \Sigma^{-1} H \Sigma^{-1}\right) = \\ &= \frac{1}{2} \text{vec}(\Sigma^{-1} H \Sigma^{-1}) = \\ &= \{\text{свойство (A.12)}\} = \frac{1}{2} (\Sigma^{-1} \otimes \Sigma^{-1}) \text{vec}(H) \end{aligned}$$

Приращение вектора, полученное в виде умножения матрицы на векторизацию приращения, эквивалентно тому, что функция под дифференциалом рассматривалась бы как функция от $\text{vec } \Sigma$:

$$\mathbb{E}_\theta D_{\text{vec } \Sigma} \nabla_{\text{vec } \Sigma} \log q(\theta | \mu, \Sigma)[\text{vec } H] = \frac{1}{2} (\Sigma^{-1} \otimes \Sigma^{-1}) \text{vec}(H)$$

Отсюда $F_{\text{vec } \Sigma \text{ vec } \Sigma} = \frac{1}{2} (\Sigma^{-1} \otimes \Sigma^{-1})$. ■

Утверждение 42: Обратная матрица Фишера для нормального распределения имеет вид:

$$F_q^{-1}(\lambda) = \begin{pmatrix} \Sigma & \mathbf{0} \\ \mathbf{0} & 2(\Sigma \otimes \Sigma) \end{pmatrix} \quad (\text{A.15})$$

Пояснение. Для обращения нижнего блока применить формулу (A.11). ■

A.2.4. Covariance Matrix Adaptation Evolution Strategy (CMA-ES)

Вспомним общую формулу градиента для эволюционных стратегий:

$$\nabla_{\lambda} g(\lambda) = \mathbb{E}_{\theta \sim q(\theta | \lambda)} \nabla_{\lambda} \log q(\theta | \lambda) J(\theta) \quad (\text{A.16})$$

Достаточно домножить её на обратную матрицу Фишера, чтобы получить формулу натурального градиента для обновления $\mu = (\mu, \Sigma)$.

Теорема 40: Натуральный градиент для μ в (A.5) выглядит так:

$$\tilde{\nabla}_{\mu} g(\mu, \Sigma) = \frac{1}{N} \sum_{\theta \in \mathcal{P}} \hat{J}(\theta)(\theta - \mu)$$

Доказательство.

$$\begin{aligned} \tilde{\nabla}_{\mu} g(\mu, \Sigma) &= F_{\mu\mu}^{-1} \nabla_{\mu} g(\mu, \Sigma) = \\ &= \{\text{подставляем (A.16)}\} = \frac{1}{N} \sum_{\theta \in \mathcal{P}} \hat{J}(\theta) F_{\mu\mu}^{-1} \nabla_{\mu} \log q(\theta | \mu, \Sigma) = \\ &= \{\text{подставляем } F_{\mu\mu}^{-1} \text{ из (A.15) и градиент из (A.13)}\} = \\ &= \frac{1}{N} \sum_{\theta \in \mathcal{P}} \hat{J}(\theta) \Sigma \Sigma^{-1} (\theta - \mu) = \\ &= \frac{1}{N} \sum_{\theta \in \mathcal{P}} \hat{J}(\theta) (\theta - \mu) \end{aligned}$$

■

Заметим, что для OpenAI-ES 2.7 при константной матрице ковариации натуральный градиент и градиент отличаются на константу (она «сокращается с learning rate»); поэтому можно считать, что OpenAI-ES есть тоже алгоритм натуральной эволюционной стратегии.

Теорема 41: Натуральный градиент для Σ в (A.5) выглядит так:

$$\tilde{\nabla}_{\Sigma} g(\mu, \Sigma) = \frac{1}{N} \sum_{\theta \in \mathcal{P}} \hat{J}(\theta) (S - \Sigma)$$

Доказательство.

$$\begin{aligned} \tilde{\nabla}_{\text{vec}(\Sigma)} g(\mu, \Sigma) &= F_{\text{vec}(\Sigma) \text{ vec}(\Sigma)}^{-1} \nabla_{\text{vec}(\Sigma)} g(\mu, \Sigma) = \\ &= \{\text{подставляем (A.16)}\} = \frac{1}{N} \sum_{\theta \in \mathcal{P}} \hat{J}(\theta) F_{\text{vec}(\Sigma) \text{ vec}(\Sigma)}^{-1} \text{ vec} (\nabla_{\Sigma} \log q(\theta | \mu, \Sigma)) = \\ &= \{\text{подставляем } F_{\text{vec}(\Sigma) \text{ vec}(\Sigma)}^{-1} \text{ из (A.15) и градиент из (A.14)}\} = \\ &= \frac{1}{N} \sum_{\theta \in \mathcal{P}} \hat{J}(\theta) (\Sigma \otimes \Sigma) \text{ vec} (\Sigma^{-1} S \Sigma^{-1} - \Sigma^{-1}) = \\ &= \{\text{свойство (A.10)}\} = \frac{1}{N} \sum_{\theta \in \mathcal{P}} \hat{J}(\theta) \text{ vec} (\Sigma \Sigma^{-1} S \Sigma^{-1} \Sigma - \Sigma \Sigma^{-1} \Sigma) = \\ &= \text{vec} \left(\frac{1}{N} \sum_{\theta \in \mathcal{P}} \hat{J}(\theta) (S - \Sigma) \right) \end{aligned}$$

Отсюда, убирая векторизацию, получаем формулу.

■

§A.3. Сходимость Q-learning

В данном разделе приведено доказательство теоремы 27. Пусть дано MDP с конечным пространством состояний \mathcal{S} и действий \mathcal{A} . $Q_0^*(s, a)$ — начальное приближение, на k -ом шаге $Q_k^*(s, a)$ строится по правилу

$$Q_{k+1}^*(s, a) = (1 - \alpha_k(s, a))Q_k^*(s, a) + \alpha_k(s, a) \left(r(s, a) + \gamma \max_{a'} Q_k^*(s', a') \right) \quad (\text{A.17})$$

где $s' \sim p(s' | s, a)$, а $\alpha_k(s, a)$ — случайные величины, на которые накладывается единственное требование: для всех s, a с вероятностью 1 выполнено:

$$\sum_{k \geq 0} \alpha_k(s, a) = +\infty \quad \sum_{k \geq 0} \alpha_k(s, a)^2 < +\infty \quad (\text{A.18})$$

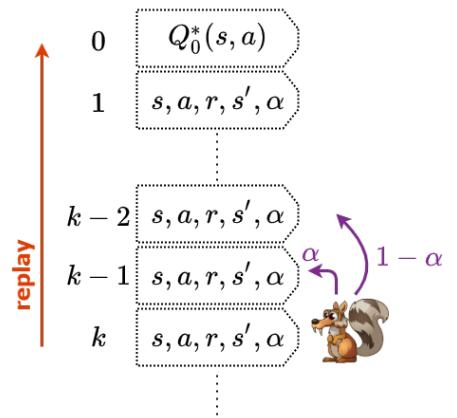
A.3.1. Action Replay Process

Для доказательства рассмотрим конструкцию под названием Action Replay Process. Давайте запустим алгоритм Q-learning (проведём, чисто теоретически, бесконечное число итераций) и запишем для каждого реализации $\alpha_k(s, a)$ и $s'_k(s, a)$ — те следующие состояния, которые использовались на k -ом шаге для обновления $Q_k^*(s, a)$. Будем проводить такую аналогию — запишем эту историю «на карточках»: на k -ой карточке записано для каждой пары s, a по одному сэмплу $s'_k(s, a) \sim p(s' | s, a)$, а также степень $\alpha_k(s, a)$, с которой этот сэмпл был использован для обновления Q-функции; можно считать, что для них в случившейся реализации выполнено (A.18), поскольку это происходит с вероятностью 1. Карточки, считаем, «сложены в стопку», начиная с нулевой карточки, на которой, условно, напишем наше исходное приближение Q-функции $Q_0(s, a)$ для всех s, a . Мы сейчас будем эту стопку карт «просматривать» от конца к началу.

Определение 60: Для данной реализации алгоритма Q-learning Action Replay Process (ARP) будем называть следующее MDP:

- Пространством состояний будем считать пару s, n , где $s \in \mathcal{S}$, n — номер карточки.
- Пространством действий будем считать \mathcal{A} .
- Процесс генерации следующего состояния по данному состоянию s, n и действию a , который мы будем обозначать как $p_{\text{ARP}}(s', n' | s, n, a)$, задаётся следующим образом. Если номер карточки n равен нулю, то «стопка карт закончилась», и следующее состояние — терминальное. Если $n > 0$, то бросаем нечестную монетку, которая с вероятностью $\alpha_{n-1}(s, a)$ выдаёт результат «остановиться», а с вероятностью $1 - \alpha_{n-1}(s, a)$ выдаёт результат «пойти дальше». «Остановиться» означает, что мы полагаем итоговым следующим состоянием пару $s'_{n-1}(s, a), n - 1$. «Пойти дальше» означает, что верхняя карточка колоды удаляется — n всё равно уменьшается на единицу, — и процедура повторяется уже для тройки $s, n - 1, a$: мы снова подбрасываем монетку и так далее, пока не выпадет «остановиться» или колода карт не закончится.
- Награда для тройки s, n, a есть $r(s, a)$, если $n > 0$, и $Q_0^*(s, a)$ иначе.

Данное определение внезапно содержит в себе всю основную идею доказательства. Что тут происходит? Давайте посмотрим на первые n шагов результата работы Q-learning-a. Попробуем построить MDP, которое было бы в некотором смысле «похоже» на исходное MDP, но использующее только собранную историю (т.к. доступа к $p(s' | s, a)$ у нас нет). Если $n = 0$, и истории нет, то мы считаем, что мы бы получили $Q_0^*(s, a)$ в качестве награды за всю оставшуюся игру — такого наше «исходное» приближение MDP. Для больших n для данной пары состояния-действие мы в качестве следующего состояния хотели бы засэмплировать $s' \sim p(s' | s, a)$, но вместо этого у нас есть лишь коллекция $s'_k(s, a)$. Давайте с вероятностью $\alpha_{n-1}(s, a)$ возьмём в качестве сэмпла $s'_n(s, a)$, с вероятностью $(1 - \alpha_{n-1}(s, a))\alpha_{n-2}(s, a)$ возьмём в качестве сэмпла $s'_{n-1}(s, a)$, и так далее. Мы знаем, что при стремлении n к бесконечности альфы, во-первых, уходят к нулю (это гарантирует сходимость ряда из квадратов альф), а во-вторых, сэмплов будет бесконечно много (бесконечно много альф обязано быть ненулевыми из-за расходимости ряда из альф). Однако, после каждого шага в ARP, n уменьшается (как минимум на единицу), и через некоторое число шагов такая игра гарантированно завершится — вся история из первых n шагов будет «проиграна» как на повторе от n -го шага до первого. Hence the name.



A.3.2. Ключевые свойства ARP

Принципиально по построению выполнен таковой фокус:

Теорема 42: В любом ARP оптимальная Q-функция $Q_{\text{ARP}}^*(s, n, a)$ в точности равна

$$Q_{\text{ARP}}^*(s, n, a) = Q_n^*(s, a) \quad (\text{A.19})$$

Доказательство. По индукции. Для $n = 0$ по определению следующее состояние всегда будет терминальным, а наградой для s, n, a является $Q_0^*(s, a)$. Значит, $Q_{\text{ARP}}^*(s, 0, a) = Q_0^*(s, a)$.

Пусть выполнено $Q_{\text{ARP}}^*(s, n, a) = Q_n^*(s, a)$ для любых s, a . Тогда для любых s, a величина $Q_{\text{ARP}}^*(s, n + 1, a)$ равна следующему: с вероятностью $\alpha_n(s, a)$ после выполнения действия a в состоянии $s, n + 1$ будет получена награда $r(s, a)$, а следующим состоянием будет $s'_n(s, a), n$; тогда дальнейшее оптимальное поведение даст награду $\max_{a'} Q_{\text{ARP}}^*(s'_n(s, a), n, a') = \max_{a'} Q_n^*(s'_n(s, a), a')$ по предположению индукции. А с вероятностью $1 - \alpha_n(s, a)$ мы не остановимся на n и повторим бросок монетки для s, n, a , после которого при дальнейшем оптимальном поведении мы получаем награду $Q_{\text{ARP}}^*(s, n, a) = Q_n^*(s, a)$. Собирая это вместе, получаем:

$$Q_{\text{ARP}}^*(s, n + 1, a) = \alpha_n(s, a) \left(r(s, a) + \gamma \max_{a'} Q_n^*(s'_n(s, a), a') \right) + (1 - \alpha_n(s, a)) Q_n^*(s, a)$$

Справа в точности стоит $Q_{n+1}^*(s, a)$! Иначе говоря, функция переходов в ARP специально построена так, что оценочные функции удовлетворяют формулам обновления Q-learning-a. ■

Доказательство сходимости Q-learning-a идеино сводится к тому, что при стремлении n к бесконечности ARP с начальным состоянием s_0, n (и коэф. дисконтирования γ) становится всё больше похож на исходное, настоящее MDP.

Покажем, что в ARP неявно содержится информация о $p(s' | s, a)$. Рассмотрим следующую величину:

$$p_{\text{ARP}}(s' | s, n, a) = \sum_{n'=1}^{n-1} p_{\text{ARP}}(s', n' | s, n, a),$$

то есть вероятность после выбора действия a из состояния s, n оказаться в s' , если неважно, сколько карточек n' у нас останется после одного шага.

Теорема 43:

$$p_{\text{ARP}}(s' | s, n, a) \xrightarrow{n \rightarrow +\infty} p(s' | s, a) \quad (\text{A.20})$$

Доказательство. Рассмотрим $p_{\text{ARP}}(s' | s, n + 1, a)$ — вероятность оказаться в s' после выполнения a из состояния $s, n + 1$ вне зависимости от n' . С вероятностью $\alpha_n(s, a)$ исходом будет $s'_n(s, a)$: если оно равно рассматриваемому s' , то это даёт $\alpha_n(s, a)$ вероятность для $p_{\text{ARP}}(s' | s, n + 1, a)$, иначе 0; с вероятностью $1 - \alpha_n(s, a)$ процесс генерации следующего состояния будет повторён из s, n, a , и тогда вероятность оказаться в состоянии s' равна $p_{\text{ARP}}(s' | s, n, a)$ по определению. Итого получаем:

$$p_{\text{ARP}}(s' | s, n + 1, a) = (1 - \alpha_n(s, a)) p_{\text{ARP}}(s' | s, n, a) + \alpha_n(s, a) \mathbb{I}[s'_n(s, a) = s']$$

Мы получили формулу экспоненциального сглаживания (3.25) для величин $\mathbb{I}[s'_n(s, a) = s']$. При этом альфы удовлетворяют условиям сходимости (3.24)! Значит, по теореме о сходимости экспоненциального сглаживания 26 эти величины в пределе сходятся к мат.ожиданию случайной величины $\mathbb{E}\mathbb{I}[s'_n(s, a) = s']$. Поскольку $s'_n(s, a)$ для любого n генерировался из $p(s' | s, a)$, то

$$\mathbb{E}\mathbb{I}[s'_n(s, a) = s'] = p(s' | s, a),$$

и, следовательно, именно к нему стремится $p_{\text{ARP}}(s' | s, n, a)$. ■

Мы показали, что процесс генерации s' в ARP «корректно» имитирует реальную $p(s' | s, a)$ при большом числе карточек. Значит, наше ARP с большим числом карточек всё больше похоже на настоящее MDP. Коли так, то и наверняка и оптимальная Q-функция для ARP при стремлении n к бесконечности всё больше похожа на $Q^*(s, a)$ исходного MDP:

$$\lim_{n \rightarrow +\infty} Q_{\text{ARP}}^*(s, n, a) = Q^*(s, a)$$

Если это так, то в совокупности с (A.19) мы получаем доказываемое: для любой реализации Q-learning-a

$$\lim_{n \rightarrow +\infty} Q_n^*(s, a) = \lim_{n \rightarrow +\infty} Q_{\text{ARP}}^*(s, n, a) = Q^*(s, a)$$

A.3.3. Схожесть ARP и настоящего MDP

Нам осталось формально показать, почему «похожесть MDP» влечёт похожесть оптимальных Q-функций. Техническим препятствием для этого является то, что ARP на каждом шаге «тратит карточки» — мы, идя с конца колоды к началу, теряем какое-то случайное число карточек, а, оставшись с маленьким числом карточек, уже не умеем «хорошо имитировать» настоящую функцию переходов.

Следующее утверждение является вспомогательным для основной теоремы: оно говорит, что можно запастись достаточным количеством карточек, чтобы можно было сделать шаг и остаться всё равно со сколь угодно большим числом карточек.

Теорема 44: Для любого ARP и любого целого числа карточек m можно выбрать число карточек n так, что для всех s, a, s' вероятность $p_{\text{ARP}}(n' < m \mid s, n, a, s')$ бесконечна мала.

Доказательство. Вероятность оказаться с числом карточек меньше m , стартуя из уровня n , не меньше чем $\prod_{i=m}^n (1 - \alpha_i(s, a))$ — это вероятность в принципе прокрутить историю от n -й карточки до m -й. Воспользуемся следующим фактом: при любых $\alpha \in [0, 1]$ верно

$$1 - \alpha \leq \exp(-\alpha).$$

Подставляя это неравенство в произведение, получаем:

$$\prod_{i=m}^n (1 - \alpha_i(s, a)) \leq \prod_{i=m}^n \exp(-\alpha_i(s, a)) = \exp\left(-\sum_{i=m}^n \alpha_i(s, a)\right) \xrightarrow{n \rightarrow +\infty} \exp(-\infty) = 0$$

В последнем переходе мы воспользовались тем, что ряд альф расходится, а значит и любой его хвост, начинающийся с любого конечного m , тоже расходится. ■

Теперь обсудим идею основного доказательства о том, что $Q_{\text{ARP}}^*(s, n, a) \xrightarrow{n \rightarrow +\infty} Q^*(s, a)$. Если мы в ARP сидим в состоянии с большим n , то у нас есть три причины, по которым $Q_{\text{ARP}}^*(s, n, a)$ отличается от $Q^*(s, a)$:

- с некоторой маленькой вероятностью мы после нескольких первых шагов окажемся в ARP в состоянии с маленьким значением n , где все наши приближения уже не работают. Мы выберем n достаточно большим, чтобы эта вероятность была очень маленькой.
- наше приближение функции переходов $p_{\text{ARP}}(s' \mid s, n, a)$ при больших n близка, но не точно совпадает с истинной $p(s' \mid s, a)$. Мы будем пользоваться тем, что как истинная оценочная функция, так и наша оценочная функция ограничены: $Q_{\text{ARP}}^*(s, n, a) < C, Q^*(s, a) < C$ для некоторой константы C (это следует из наших стандартных требований регулярности к MDP, которое справедливы и для ARP), поэтому достаточно выбрать n достаточно большим, чтобы сделать эту ошибку маленькой.
- наконец, основная ошибка заключается в том, что мы принимаем решения последовательно, а значит, ошибка из-за погрешности функции переходов будет накапливаться. Очень условно, это «ошибка внутри нашей аппроксимации Q-функции». С ней мы будем бороться наиболее хитрым образом: мы рассмотрим ошибку в награде, собираемой на протяжении первых k шагов. Остальная часть этой ошибки будет домножаться на γ^k , следовательно, достаточно будет выбрать k достаточно большим, чтобы ошибка была меньше наперёд заданного малого числа $\epsilon > 0$.

Введём следующее обозначение: «максимальная ошибка, если у нас на руках n карточек»:

$$\nu(n) := \max_{s, a} |Q_{\text{ARP}}^*(s, n, a) - Q^*(s, a)|$$

Надо доказать, что она стремится к нулю. Можно считать, что и $\nu(n)$ ограничено в силу ограниченности оценочных функций, чем мы будем пользоваться, когда карточек остаётся мало.

Теорема 45: Пусть n и m таковы, что для любых s, a, s' выполнено

$$|p_{\text{ARP}}(s' \mid s, n, a) - p(s' \mid s, a)| < \epsilon$$

$$p_{\text{ARP}}(n' < m \mid s, n, a, s') < \epsilon$$

для данного ϵ . Тогда для некоторой константы C справедливо следующее рекуррентное соотношение:

$$\nu(n) \leq \gamma \max_{n' \geq m} \nu(n') + C\epsilon$$

Доказательство.

$$\begin{aligned}
\nu(n) &= \max_{s,a} |Q_{\text{ARP}}^*(s, n, a) - Q^*(s, a)| = \\
&= \{\text{уравнение оптимальности Беллмана (3.16); слагаемые с наградой сокращаются}\} = \\
&= \gamma \max_{s,a} \left| \sum_{s',n'} p_{\text{ARP}}(s', n' | s, n, a) \max_{a'} Q_{\text{ARP}}^*(s', n', a') - \sum_{s'} p(s' | s, a) \max_{a'} Q^*(s', a') \right| = \\
&= \{\text{добавляем и вычитаем } \sum_{s',n'} p_{\text{ARP}}(s', n' | s, a) \max_{a'} Q^*(s', a')\} = \\
&= \gamma \max_{s,a} \left| \sum_{s',n'} p_{\text{ARP}}(s', n' | s, n, a) (\max_{a'} Q_{\text{ARP}}^*(s', n', a') - \max_{a'} Q^*(s', a')) - \right. \\
&\quad \left. + \sum_{s'} (p_{\text{ARP}}(s' | s, n, a) - p(s' | s, a)) \max_{a'} Q^*(s', a') \right| \leq \\
&\leq \{\text{используем свойство максимумов (3.22)} \max_x f(x) - \max_x g(x) \leq \max_x |f(x) - g(x)|\} \leq \\
&\leq \gamma \max_{s,a} \left[\sum_{s',n'} p_{\text{ARP}}(s', n' | s, n, a) \max_{a'} |Q_{\text{ARP}}^*(s', n', a') - Q^*(s', a')| + \right. \\
&\quad \left. + \sum_{s'} |p_{\text{ARP}}(s' | s, n, a) - p(s' | s, a)| \max_{a'} Q^*(s', a') \right] = \\
&= \{\text{правило произведения}\} = \\
&= \gamma \max_{s,a} \left[\sum_{s'} p_{\text{ARP}}(s' | s, n, a) \sum_{n'} p_{\text{ARP}}(n' | s, n, a, s') \max_{a'} |Q_{\text{ARP}}^*(s', n', a') - Q^*(s', a')| + \right. \\
&\quad \left. + \sum_{s'} |p_{\text{ARP}}(s' | s, n, a) - p(s' | s, a)| \max_{a'} Q^*(s', a') \right] \leq \\
&\leq \{\text{определение } \nu(n) \text{ и свойство } \mathbb{E}f(x) \leq \max_x f(x)\} \leq \\
&\leq \gamma \max_{s,a} \left[\sum_{n'} p_{\text{ARP}}(n' | s, n, a, s') \nu(n') + \right. \\
&\quad \left. + \sum_{s'} |p_{\text{ARP}}(s' | s, n, a) - p(s' | s, a)| \max_{a'} Q^*(s', a') \right] \leq \\
&\leq \{\text{ошибка аппроксимации функции переходов и ограниченность } Q^*(s, a)\} \leq \\
&\leq \gamma \max_{s,a} \left[\sum_{n'} p_{\text{ARP}}(n' | s, n, a, s') \nu(n') + C\epsilon \right] = \\
&= \{\text{рассматриваем два случая: } n' < m \text{ и } n' \geq m\} = \\
&\leq \gamma \max_{s,a} \left[\sum_{n' \geq m} p_{\text{ARP}}(n' | s, n, a, s') \nu(n') + \sum_{n' < m} p_{\text{ARP}}(n' | s, n, a, s') v(n') + C\epsilon \right] \leq \\
&\leq \{\text{пользуемся условием теоремы и тем, что } \nu(n) \text{ ограничено}\} \leq \\
&\leq \gamma \max_{s,a} \left[\sum_{n' \geq m} p_{\text{ARP}}(n' | s, n, a, s') \nu(n') + C'\epsilon \right] \leq \\
&\leq \{\text{свойство } \mathbb{E}f(x) \leq \max_x f(x)\} \leq \\
&\leq \gamma \max_{n' \geq m} \nu(n') + C'\epsilon
\end{aligned}$$

■

Теорема 46:

$$\nu(n) \xrightarrow{n \rightarrow +\infty} 0$$

Доказательство. Пусть дано $\epsilon > 0$. Покажем, что начиная с некоторого номера, $\nu(n) < 2C\epsilon$. Для этого выберем целое k так, чтобы $\gamma^k < \epsilon$, и применим предыдущую теорему k раз следующим образом. Выберем какое-нибудь m_0 и подберём m_1 так, чтобы для всех s, a, s'

$$p_{\text{ARP}}(n' < m_0 | s, m_1, a, s') < \frac{\epsilon}{k}.$$

Убедимся, что m_1 достаточно большое, что $|p_{\text{ARP}}(s' | s, m_1, a) - p(s' | s, a)| < \frac{\epsilon}{k}$ (если нет, то

заменим на достаточно большое). Затем подберём m_2 так, что для всех s, a, s'

$$p_{\text{ARP}}(n' < m_1 \mid s, m_2, a, s') < \frac{\epsilon}{k}$$

и так далее вплоть до m_k .

Тогда для всех $n > m_k$:

$$\nu(n) \leq \gamma \max_{n' \geq m_{k-1}} \nu(n') + C \frac{\epsilon}{k}$$

Аналогично, для всех $n > m_{k-1}$:

$$\nu(n) \leq \gamma \max_{n' \geq m_{k-2}} \nu(n') + C \frac{\epsilon}{k}$$

Последовательно раскручивая эту цепочку k раз получим, что для всех $n > m_k$:

$$\begin{aligned} \nu(n) &\leq \gamma \max_{n' \geq m_{k-1}} \nu(n') + C \frac{\epsilon}{k} \leq \gamma^2 \max_{n' \geq m_{k-2}} \nu(n') + 2C \frac{\epsilon}{k} \leq \dots \leq \\ &\leq \gamma^k \max_{n' \geq m_0} \nu(n') + kC \frac{\epsilon}{k} \leq C\epsilon + C\epsilon = 2C\epsilon \end{aligned}$$

Вот такие дела. ■

Материалы

Большая часть материалов взята из основных курсов по обучению с подкреплением:

Курс Сергея Левина;

Курс Practical RL;

Курс Дэвида Сильвера;

Курс DeepMind;

Курс GeorgiaTech;

В [главе 2](#) большинство материала взято из книги [Luke, 2013]. Хороший обзор эволюционных стратегий можно найти в блоге Lil'log [Weng, 2019]. Алгоритм NEAT предложен в [Stanley and Miikkulainen, 2002]. Алгоритм WANN развил его идею в [Gaier and Ha, 2019]. Кросс-энтропийный метод как метод оптимизации и метод вычисления вероятности маловероятных событий предложен в [Botev et al., 2013]; его применение к задаче RL обычно связывают с [Szita and Lörincz, 2006], где его применили к тетрису. OpenAI-ES описана в [Salimans et al., 2017]; упомянутый алгоритм ARS, действующий примерно также, предложен в [Mania et al., 2018]. Идея адаптировать ковариационную матрицу в эволюционных стратегиях восходит корнями к [Hansen and Ostermeier, 1996]; полный технический обзор всего набора эвристик, использующихся как алгоритм CMA-ES, можно прочитать [здесь](#) [Hansen, 2016]. Доказательство того, что адаптация матрицы ковариации по сути является натуральным градиентным спуском, было независимо получено в [Akimoto et al., 2010] и [Glasmachers et al., 2010].

Больше информации по [главе 3](#) и более подробную библиографию можно получить в классической книге Саттона-Барто [Sutton and Barto, 2018]; отмечу только некоторые дополнительные ссылки. Лемма RPI была представлена в [Kakade and Langford, 2002]. Алгоритм Q-learning, изначально придуманный в [Watkins, 1989], был придуман как эвристика, но позже авторам удалось доказать сходимость в [Watkins and Dayan, 1992]; это доказательство через ARP и приведено в приложении. Связь с теорией стохастической аппроксимации, начатой ещё в далёком 1951-ом году статьёй [Robbins and Monro, 1951], была обнаружена после этого в [Tsitsiklis, 1994], что позволило доказать более сильные утверждения вроде сходимости TD(λ).

[Глава 4](#) основана на алгоритме DQN [Mnih et al., 2013], продемонстрировавшем потенциал совмещения глубокого обучения с классической теорией. Идея борьбы с переоценкой при помощи двух аппроксимаций Q-функций была предложена в [Hasselt, 2010] для табличного алгоритма. Clipped Twin оценка предложена была позже (в рамках алгоритма TD3) в [Fujimoto et al., 2018]. Double DQN предложен в [Van Hasselt et al., 2016]; Dueling DQN в [Wang et al., 2015]; Noisy Nets в [Fortunato et al., 2017]. Приоритизированный реплей был использован в DQN в [Schaal et al., 2015]; идею высчитывать приоритеты онлайн и добавлять в буфер уже «с правильным» приоритетом реализовали в алгоритме R2D2 [Horgan et al., 2018]. Эвристика многошагового DQN была описана в составе Rainbow [Hessel et al., 2018].

Изображения с роботом взяты из [курса CS-188 Introduction to Artificial Intelligence, Berkeley](#). Прочие изображения взяты из исходных статей и из распространённых сред (OpenAI Gym [Brockman et al., 2016], Mario [Kauten, 2018], Unity ML Agents [Juliani et al., 2018]). Кастомные изображения для примеров и схем были нарисованы в [draw.io](#); изображение белки на них взято из [бота этого твита](#); судя по всему, это незаузанный концепт-арт для некой игры «Трагедия белок»... а вот, пригодился!



[Akimoto et al., 2010] Akimoto, Y., Nagata, Y., Ono, I., and Kobayashi, S. (2010). Bidirectional relation between cma evolution strategies and natural evolution strategies. In *International Conference on Parallel Problem Solving from Nature*, pages 154–163. Springer.

[Botev et al., 2013] Botev, Z. I., Kroese, D. P., Rubinstein, R. Y., and L'Ecuyer, P. (2013). The cross-entropy method for optimization. In *Handbook of statistics*, volume 31, pages 35–59. Elsevier.

[Brockman et al., 2016] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

[Fortunato et al., 2017] Fortunato, M., Azar, M. G., Piot, B., Menick, J., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., et al. (2017). Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*.

[Fujimoto et al., 2018] Fujimoto, S., Van Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*.

- [Gaier and Ha, 2019] Gaier, A. and Ha, D. (2019). Weight agnostic neural networks. In *Advances in Neural Information Processing Systems*, pages 5364–5378.
- [Glasmachers et al., 2010] Glasmachers, T., Schaul, T., Yi, S., Wierstra, D., and Schmidhuber, J. (2010). Exponential natural evolution strategies. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 393–400.
- [Hansen, 2016] Hansen, N. (2016). The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*.
- [Hansen and Ostermeier, 1996] Hansen, N. and Ostermeier, A. (1996). Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *Proceedings of IEEE international conference on evolutionary computation*, pages 312–317. IEEE.
- [Hasselt, 2010] Hasselt, H. V. (2010). Double q-learning. In *Advances in neural information processing systems*, pages 2613–2621.
- [Hessel et al., 2018] Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. (2018). Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.
- [Horgan et al., 2018] Horgan, D., Quan, J., Budden, D., Barth-Maron, G., Hessel, M., Van Hasselt, H., and Silver, D. (2018). Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933*.
- [Juliani et al., 2018] Juliani, A., Berges, V.-P., Vckay, E., Gao, Y., Henry, H., Mattar, M., and Lange, D. (2018). Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*.
- [Kakade and Langford, 2002] Kakade, S. and Langford, J. (2002). Approximately optimal approximate reinforcement learning. In *ICML*, volume 2, pages 267–274.
- [Kauten, 2018] Kauten, C. (2018). Super Mario Bros for OpenAI Gym. GitHub.
- [Luke, 2013] Luke, S. (2013). *Essentials of metaheuristics*, volume 2. Lulu Raleigh.
- [Mania et al., 2018] Mania, H., Guy, A., and Recht, B. (2018). Simple random search provides a competitive approach to reinforcement learning. *arXiv preprint arXiv:1803.07055*.
- [Mnih et al., 2013] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.
- [Robbins and Monro, 1951] Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407.
- [Salimans et al., 2017] Salimans, T., Ho, J., Chen, X., Sidor, S., and Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*.
- [Schaul et al., 2015] Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015). Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.
- [Stanley and Miikkulainen, 2002] Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127.
- [Sutton and Barto, 2018] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.
- [Szita and Lörincz, 2006] Szita, I. and Lörincz, A. (2006). Learning tetris using the noisy cross-entropy method. *Neural computation*, 18(12):2936–2941.
- [Tsitsiklis, 1994] Tsitsiklis, J. N. (1994). Asynchronous stochastic approximation and q-learning. *Machine learning*, 16(3):185–202.
- [Van Hasselt et al., 2016] Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Thirtieth AAAI Conference on Artificial Intelligence*.
- [Wang et al., 2015] Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., and De Freitas, N. (2015). Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*.
- [Watkins and Dayan, 1992] Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292.
- [Watkins, 1989] Watkins, C. J. C. H. (1989). Learning from delayed rewards.
- [Weng, 2019] Weng, L. (2019). Evolution strategies. lilianweng.github.io/lil-log.