



Конспект по обучению с подкреплением

qbrick@mail.ru

25 января 2022 г.



Аннотация

Современные алгоритмы глубокого обучения с подкреплением способны решать задачи искусственного интеллекта методом проб и ошибок без использования каких-либо априорных знаний о решаемой задаче. В этом конспекте собраны принципы работы основных алгоритмов, достигших прорывных результатов во многих задачах от игрового искусственного интеллекта до робототехники. Вся необходимая теория приводится с доказательствами, использующими единый ход рассуждений, унифицированные обозначения и определения. Основная задача этой работы — не только собрать информацию из разных источников в одном месте, но понять разницу между алгоритмами различного вида и объяснить, почему они выглядят именно так, а не иначе.

Предполагается знакомство читателя с основами машинного обучения и глубокого обучения. Об ошибках и опечатках в тексте можно сообщать в [репозитории проекта](#).

Оглавление

1 Задача обучения с подкреплением	6
1.1 Модель взаимодействия агента со средой	6
1.1.1 Связь с оптимальным управлением	6
1.1.2 Марковская цепь	7
1.1.3 Среда	8
1.1.4 Действия	9
1.1.5 Траектории	9
1.1.6 Марковский процесс принятия решений (MDP)	10
1.1.7 Эпизодичность	11
1.1.8 Дисконтирование	12
1.2 Алгоритмы обучения с подкреплением	14
1.2.1 Условия задачи RL	14
1.2.2 Сравнение с обучением с учителем	14
1.2.3 Концепция model-free алгоритмов	15
1.2.4 On-policy vs Off-policy	16
1.2.5 Классификация RL-алгоритмов	17
1.2.6 Критерии оценки RL-алгоритмов	17
1.2.7 Сложности задачи RL	18
1.2.8 Дизайн функции награды	20
1.2.9 Бенчмарки	21
2 Мета-эвристики	23
2.1 Бэйзлайны	23
2.1.1 Задача безградиентной оптимизации	23
2.1.2 Случайный поиск	24
2.1.3 Hill Climbing	25
2.1.4 Имитация отжига	26
2.1.5 Эволюционные алгоритмы	27
2.1.6 Weight Agnostic Neural Networks (WANN)	28
2.1.7 Видовая специализация	29
2.1.8 Генетические алгоритмы	30
2.2 Эволюционные стратегии	32
2.2.1 Идея эволюционных стратегий	32
2.2.2 Оценка вероятности редкого события	32
2.2.3 Метод Кросс-Энтропии для стохастической оптимизации	34
2.2.4 Метод Кросс-Энтропии для обучения с подкреплением (CEM)	35
2.2.5 Натуральные эволюционные стратегии (NES)	35
2.2.6 OpenAI-ES	36
2.2.7 Адаптация матрицы ковариации (CMA-ES)	37
3 Классическая теория	39
3.1 Оценочные функции	39
3.1.1 Свойства траекторий	39
3.1.2 V-функция	41
3.1.3 Уравнения Беллмана	42
3.1.4 Оптимальная стратегия	43
3.1.5 Q-функция	43
3.1.6 Принцип оптимальности Беллмана	44
3.1.7 Отказ от однородности	45
3.1.8 Вид оптимальной стратегии (доказательство через отказ от однородности)	46
3.1.9 Уравнения оптимальности Беллмана	47
3.1.10 Критерий оптимальности Беллмана	48

3.2	Улучшение политики	49
3.2.1	Advantage-функция	49
3.2.2	Relative Performance Identity (RPI)	50
3.2.3	Policy Improvement	52
3.2.4	Вид оптимальной стратегии (доказательство через PI)	53
3.3	Динамическое программирование	54
3.3.1	Метод простой итерации	54
3.3.2	Policy Evaluation	55
3.3.3	Value Iteration	57
3.3.4	Policy Iteration	58
3.3.5	Generalized Policy Iteration	59
3.4	Табличные алгоритмы	62
3.4.1	Монте-Карло алгоритм	62
3.4.2	Экспоненциальное слаживание	63
3.4.3	Стochastic аппроксимация	64
3.4.4	Temporal Difference	66
3.4.5	Q-learning	67
3.4.6	Exploration-exploitation дилемма	68
3.4.7	Реплей буфер	69
3.4.8	SARSA	70
3.5	Bias-Variance Trade-Off	73
3.5.1	Дилемма смещения-разброса	73
3.5.2	N-step Temporal Difference	74
3.5.3	Интерпретация через Credit Assignment	75
3.5.4	Backward View	76
3.5.5	Eligibility Trace	78
3.5.6	TD(λ)	78
3.5.7	Retrace(λ)	81
4	Value-based подход	85
4.1	Deep Q-learning	85
4.1.1	Q-сетка	85
4.1.2	Переход к параметрической Q-функции	85
4.1.3	Таргет-сеть	87
4.1.4	Декорреляция сэмплов	88
4.1.5	DQN	89
4.2	Модификации DQN	91
4.2.1	Overestimation Bias	91
4.2.2	Twin DQN	93
4.2.3	Double DQN	93
4.2.4	Dueling DQN	93
4.2.5	Шумные сети (Noisy Nets)	94
4.2.6	Приоритизированный реплей (Prioritized DQN)	96
4.2.7	Multi-step DQN	97
4.2.8	Retrace	99
4.3	Distributional RL	100
4.3.1	Идея Distributional подхода	100
4.3.2	Z-функция	101
4.3.3	Distributional-форма уравнения Беллмана	103
4.3.4	Distributional Policy Evaluation	103
4.3.5	Distributional Value Iteration	108
4.3.6	Категориальная аппроксимация Z-функций	110
4.3.7	Categorical DQN	111
4.3.8	Квантильная аппроксимация Z-функций	114
4.3.9	Quantile Regression DQN	115
4.3.10	Implicit Quantile Networks	117
4.3.11	Rainbow DQN	118
5	Policy Gradient подход	120
5.1	Policy Gradient Theorem	120
5.1.1	Выход первым способом	120
5.1.2	Выход вторым способом	121
5.1.3	Физический смысл	124

5.1.4	REINFORCE	125
5.1.5	State visitation frequency	125
5.1.6	Расцепление внешней и внутренней стохастики	126
5.1.7	Связь с policy improvement	128
5.1.8	Бэйзлайн	129
5.2	Схемы «Актёр-критик»	131
5.2.1	Введение критика	131
5.2.2	Bias-variance trade-off	132
5.2.3	Generalized Advantage Estimation (GAE)	134
5.2.4	Обучение критика	135
5.2.5	Advantage Actor-Critic (A2C)	136
5.3	Продвинутые Policy Gradient	138
5.3.1	Суррогатная функция	138
5.3.2	Нижняя оценка	140
5.3.3	Trust Region Policy Optimization (TRPO)	142
5.3.4	Proximal Policy Loss	145
5.3.5	Clipped Value Loss	147
5.3.6	Proximal Policy Optimization (PPO)	148
6	Continuous control	150
6.1	DDPG	150
6.1.1	Выход из Deep Q-learning	150
6.1.2	Выход из Policy Gradient	151
6.1.3	Связь между схемами	152
6.1.4	Ornstein–Uhlenbeck Noise	152
6.1.5	Deep Deterministic Policy Gradient (DDPG)	153
6.1.6	Twin Delayed DDPG (TD3)	154
6.1.7	Обучение стохастических политик	156
6.2	Soft Actor-Critic	158
6.2.1	Maximum Entropy RL	158
6.2.2	Soft Policy Evaluation	159
6.2.3	Soft Policy Improvement	160
6.2.4	Soft Actor-Critic (SAC)	162
6.2.5	Аналоги других алгоритмов	163
7	Model-based	166
7.1	Бандиты	166
7.1.1	Задача многоруких бандитов	166
7.1.2	Простое решение	167
7.1.3	Теорема Лаи-Роббина	168
7.1.4	Upper Confidence Bound (UCB)	169
7.1.5	Сэмплирование Томпсона	170
7.1.6	Обобщение на табличные MDP	172
7.2	Обучаемые модели окружения	173
7.2.1	Планирование	173
7.2.2	Модели мира (World Models)	174
7.2.3	Модель прямой динамики	175
7.2.4	Сновидения	176
7.3	Планирование для дискретного управления	176
7.3.1	Monte-Carlo Tree Search (MCTS)	176
7.3.2	Применение MCTS	178
7.3.3	Дистилляция MCTS	180
7.3.4	AlphaZero	180
7.3.5	μ -Zero	181
7.4	Планирование для непрерывного управления	183
7.4.1	Прямое дифференцирование	183
7.4.2	Linear Quadratic Regulator (LQR)	184
7.4.3	Случай шумной функции перехода	186
7.4.4	Iterative LQR (iLQR)	187
8	Next Stage	189
8.1	Имитационное обучение	189
8.1.1	Клонирование поведения	189

8.1.2	Обратное обучение с подкреплением (Inverse RL)	190
8.1.3	Guided Cost Learning	192
8.1.4	Generative Adversarial Imitation Learning (GAIL)	193
8.1.5	Generative Adversarial Imitation from Observation (GAIfO)	195
8.2	Внутренняя мотивация	195
8.2.1	Вспомогательные задачи	195
8.2.2	Совмещение мотиваций	196
8.2.3	Exploration Bonuses	198
8.2.4	Дистилляция случайной сетки (RND)	199
8.2.5	Любопытство	200
8.2.6	Модель обратной динамики	201
8.2.7	Внутренний модуль любопытства (ICM)	202
8.3	Multi-task RL	203
8.3.1	Многозадачность	203
8.3.2	Мета-контроллеры	204
8.3.3	Переразметка траекторий	205
8.3.4	Hindsight Experience Replay (HER)	205
8.3.5	Hindsight Relabeling	206
8.4	Иерархическое обучение с подкреплением	208
8.4.1	Опции	208
8.4.2	Semi-MDP	209
8.4.3	Оценочная функция по прибытию (U-функция)	210
8.4.4	Intra-option обучение	211
8.4.5	Обучение стратегий рабочих	211
8.4.6	Обучение функций терминальности	212
8.4.7	Феодальный RL	213
8.4.8	Feudal Networks (FuN)	214
8.4.9	HIRO	215
8.5	Частично наблюдаемые среды	216
8.5.1	Частично наблюдаемые MDP	216
8.5.2	Belief MDP	216
8.5.3	Рекуррентные сети в Policy Gradient	218
8.5.4	R2D2	218
8.5.5	Neural Episodic Control (NEC)	219
8.6	Мульти-агентное обучение с подкреплением	220
8.6.1	Связь с теорией игр	220
8.6.2	Централизация обучения	220
8.6.3	Self-play в антагонистических играх	221
8.6.4	QMix в кооперативных играх	221
8.6.5	Multi-Agent DDPG (MADDPG) в смешанных играх	223
8.6.6	Системы коммуникации (DIAL)	224

A Приложение	226	
A.1	Натуральный градиент	226
A.1.1	Проблема параметризации	226
A.1.2	Матрица Фишера	227
A.1.3	Натуральный градиент	228
A.2	Обоснование формул CMA-ES	229
A.2.1	Вычисление градиента	229
A.2.2	Произведение Кронекера	230
A.2.3	Вычисление матрицы Фишера	231
A.2.4	Covariance Matrix Adaptation Evolution Strategy (CMA-ES)	233
A.3	Сходимость Q-learning	234
A.3.1	Action Replay Process	234
A.3.2	Ключевые свойства ARP	235
A.3.3	Схожесть ARP и настоящего MDP	236

ГЛАВА 1

Задача обучения с подкреплением

Ясно, что обучение с учителем это не та модель «обучения», которая свойственна интеллектуальным существам. 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 зарезервированы 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 дифференциальным уравнениям:

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

Коли агент может как-то взаимодействовать со средой или влиять на неё, мы можем промоделировать взаимодействие следующим образом: скажем, что в каждый момент времени t агент в зависимости от текущего состояния среды s выбирает некоторое действие («управление») $a(s, t)$:

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

Для моделирования награды положим, что в каждый момент времени агент получает наказание или штраф (cost)¹ в объёме $L(s, a(s, t), t)$. Итоговой наградой агента полагаем суммарную награду, то есть интеграл по времени:

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

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 оптимальных $a(s, t)$ аналитически. Проводя аналогию с задачей максимизации некоторой, например, дифференцируемой функции $f(x) \rightarrow \max_x$, теория оптимального управления ищет необходимые условия решения: например, что в экстремуме функции обязательно $\nabla_x f(x) = 0$. Никакого «обучения методом проб и ошибок» здесь не предполагается.

В обучении с подкреплением вместо поиска решения аналитически мы будем строить итеративные методы оптимизации, искать аналоги, например, градиентного спуска. В такой процедуре неизбежно появится цепочка приближений решений: мы начнём с какой-то функции, выбирающей действия, возможно, не очень удачной и не способной набрать большую награду, но с ходом работы алгоритма награда будет оптимизироваться и способ выбора агентом действий будет улучшаться. Так будет выглядеть обучение.

Также RL 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

В обучении с подкреплением 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».²

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.

¹теория оптимального управления строилась в СССР в формализме минимизации штрафов (потерь, убытков и наказаний), когда построенная в США теория обучения с подкреплением — в формализме максимизации награды (прибыли, счастья, тортиков). Само собой, формализмы эквивалентны, и любой штраф можно переделать в награду домножением на минус, и наоборот.

²вопрос 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 $p(s' | s, a)$.

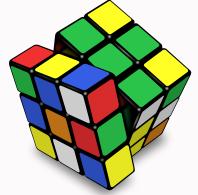
В таком определении среды заложена марковость (независимость переходов от истории) и стационарность (независимость от времени). Время при этом дискретно, в частности, нет понятия «времени принятия решения агентом»: среда, находясь в состоянии s , ожидает от агента действие $a \in \mathcal{A}$, после 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 8.5.

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 $\mathcal{A} = \{\text{■}, \text{■}\}$; 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 конфигурацией и действием, соответственно среди 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} :

- a) **конечное**, или **дискретное пространство действий** (discrete action space): $|\mathcal{A}| < +\infty$. Мы также будем предполагать, что число действий $|\mathcal{A}|$ достаточно мало.
- b) **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).



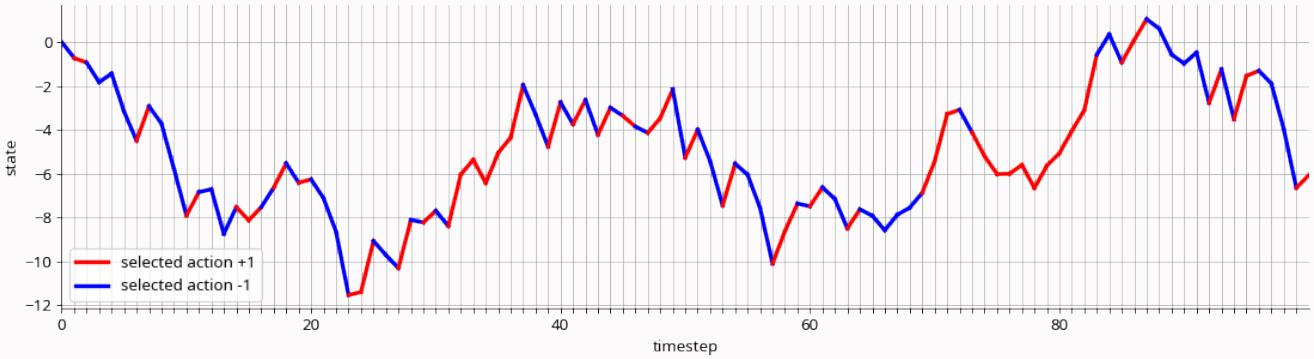
Заметим, что множество действий не меняется со временем и не зависит от состояния. Если в практической задаче предполагается, что множество допустимых действий разное в различных состояниях, в «законы физики» прописывается реакция на некорректное действие, например, случайный выбор корректного действия 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 (не зависящих от времени), вообще говоря, потребует обоснования.

1.1.5. Траектории

Определение 4: Набор $\mathcal{T} := (s_0, a_0, s_1, a_1, s_2, a_2, s_3, a_3 \dots)$ 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:

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

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 $\mathbb{E}_{\pi \sim \pi}$.

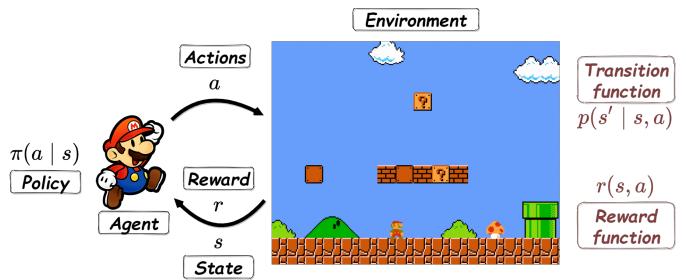
1.1.6. Марковский процесс принятия решений (MDP)

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:

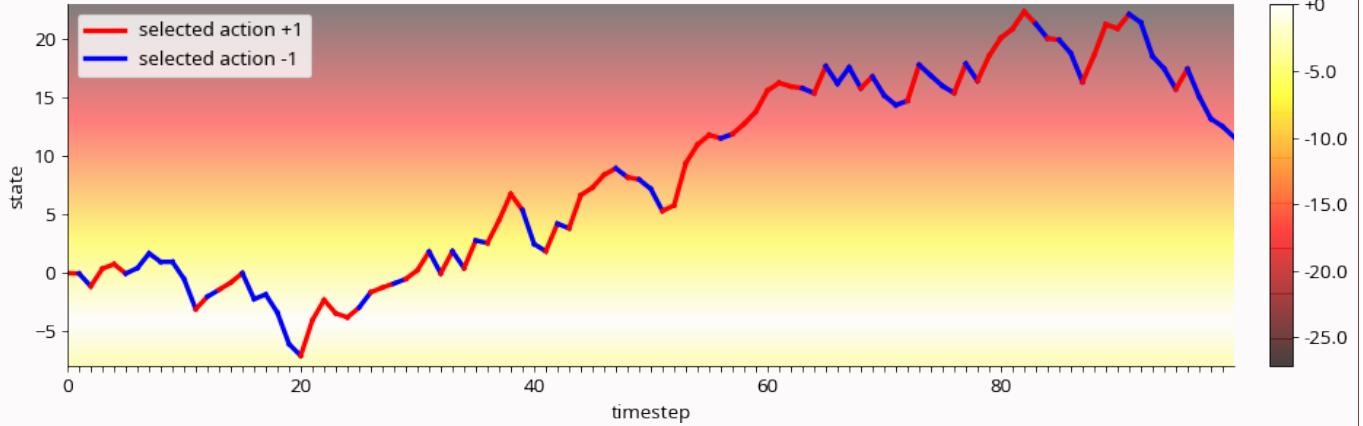
Определение 6: *Марковский процесс принятия решений* (Markov Decision Process, MDP) — это четвёрка $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r)$, где:

- $\mathcal{S}, \mathcal{A}, \mathcal{P}$ — 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 фиксировано³. 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.

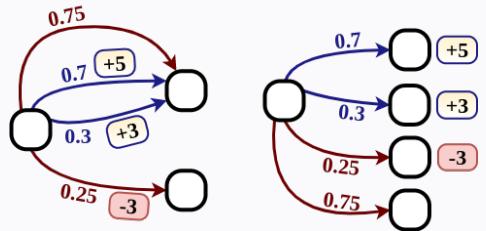
Теорема 1 — Эквивалентные определения MDP: Эквивалентно рассматривать MDP, где

- 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.

³формально его можно рассматривать как часть заданного MDP.

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



* которое 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) или **табличным**, 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 $\forall a \in \mathcal{A}$:

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

то есть с вероятностью 1 агент не сможет покинуть состояние.

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 $done(s) \in \{0, 1\}$, является ли данное состояние терминальным. По сути, после попадания в терминальное состояние дальнейшее взаимодействие бессмысленно (дальнейшие 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).

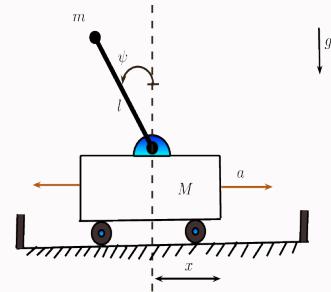
Продолжительности эпизодов (количество шагов взаимодействия) при этом, конечно, могут различаться от эпизода к эпизоду.

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 натурального n . Иначе говоря, эпизоды могут быть неограниченно долгими. ■

Пример 8 — Cartpole: К тележке на шарнире крепится стержень 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 RL будет 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:

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

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

Мы хотим исключить из рассмотрения MDP, где данный функционал может улететь в бесконечность⁴ 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 хватит⁵. Введём **дисконтирование** (discounting), коэффициент которого традиционно обозначают γ :

Определение 11: **Дисконтированной кумулятивной наградой** (discounted cumulative reward) или **total**

⁴если 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».

⁵могут возникнуть ситуации, где суммарной награды просто не существует (например, если агент в бесконечном процессе всегда получает $+1$ на чётных шагах и -1 на нечётных).

`return` для траектории \mathcal{T} с коэффициентом $\gamma \in (0, 1]$ 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.

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:

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

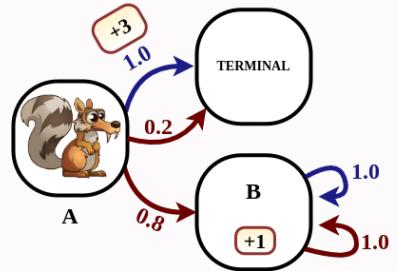
Пример 9: Посчитаем $J(\pi)$ для приведённого на рисунке MDP, $\gamma = \frac{10}{11}$ и начального состояния A. Ясно, что итоговая награда зависит только от стратегии агента в состоянии A, поэтому можно рассмотреть все стратегии, обозначив $\pi(a = \square | s = A)$ за параметр стратегии $\theta \in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

С вероятностью θ агент выберет действие \square , 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:

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



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 оптимальными были бы все стратегии.

Всюду далее подразумевается⁶ 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. ■

⁶в качестве акта педантичности оговоримся, что также всюду подразумевается измеримость всех функций, необходимая для существования всех рассматриваемых интегралов и мат.ожиданий.

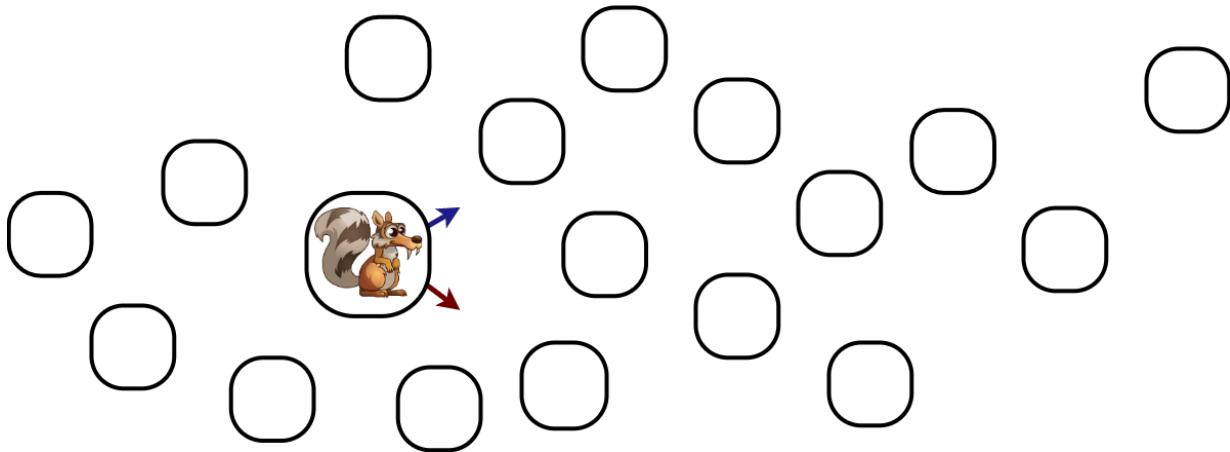
§1.2. Алгоритмы обучения с подкреплением

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 $T \sim \pi$; иными словами, принципиально доступны только сэмплы из trajectory distribution (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 собирать 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: Пятёрки $T := (s, a, r, s', \text{done})$, 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*), предназначеннной для использования в среде по итогам обучения.

1.2.2. Сравнение с обучением с учителем

Необходимость собирать данные внутри самого алгоритма — одно из ключевых отличий задачи RL от обучения с учителем, где выборка подаётся алгоритму на вход. Здесь же «сигнал» для обучения предоставляется дизайном функции награды; на практике, чтобы говорить о том, что встретилась задача RL, необходимо задать MDP (представить как среду в виде симулятора или интерфейс для взаимодействия настоящего робота в реальном мире, так и функцию награды). Зачастую если какую-то «репрезентативную» выборку собрать можно, всегда проще и лучше обратиться к обучению с учителем как менее общей постановке задачи.

Утверждение 2: Задача обучения с учителем (с заданной функцией потерь) является частным случаем задачи RL.

Доказательство. Пусть дана обучающая выборка в виде пар (x, y) , x — входной объект, y — целевая переменная. Скажем, что начальное состояние есть описание объекта x , случайно сэмплированного из обучающей выборки (начальное состояние будет случайно). Агент выбирает метку класса $\hat{y} \sim \pi(\hat{y} | x)$. После этого он получает награду за шаг в размере — $\text{Loss}(\hat{y}, y)$ и игра завершается. Оптимизация такой функции награды в среднем будет эквивалентна минимизации функции потерь в обучении с учителем. ■

Поскольку RL всё-таки предполагает, что в общем случае эпизоды длиннее одного шага, агент будет своими действиями влиять на дальнейшие состояния, которые он встречает. «Управление» марковской цепью куда более существенная часть оптимизации RL алгоритмов, нежели максимизация наград за шаг, именно из этого свойства возникает большинство специфических именно для RL особенностей. И зачастую именно в таких задачах появляется такая непреодолимая для обучения с учителем проблема, как то, что правильный ответ никакому человеку, в общем-то, неизвестен. Не знаем мы обычно оптимальный наилучший ход в шахматах или как правильно поворачивать конечности роботу, чтобы начать ходить.

Да, 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 как-либо на них предобучиться. В простейшем случае предобучение выглядит так: если в алгоритме присутствует параметрически заданная стратегия π_θ , можно **клонировать поведение** (behavior cloning), т.е. восстанавливать по парам s, a из всех собранных экспертом траекторий функцию $S \rightarrow A$ (это обычная задача обучения с учителем), учиться воспроизводить действия эксперта. Задача обучения по примерам её решения около-оптимальным экспертом называется **имитационным обучением** (imitation learning); её мы обсудим отдельно в разделе 8.1, однако, если эксперт не оптимален, обученная стратегия вряд ли будет действовать хоть сколько-то лучше. Здесь можно провести прямую аналогию с задачей обучения с учителем, где верхняя граница качества алгоритма определяется качеством разметки; если разметка запутлена и содержит ошибки, обучение вряд ли удастся.

В общем же случае мы считаем, что на вход в алгоритм никаких данных эксперта не поступает. Поэтому говорят, что 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 (получать примеры траекторий с всё большей суммарной наградой), за счёт неё выбирать всё более и более оптимальные действия, и так «вытягивать сам себя из болота».

Здесь важно задать следующий вопрос: а если у нас есть данные из очень плохой стратегии, что можно рассчитывать с них выучить? То есть если мы можем собирать лишь примеры траекторий, в которых агент совершенно случайно себя ведёт (и редко получает положительную награду), можно ли с таких данных выучить, например, оптимальную стратегию? Оказывается, как мы увидим дальше, за счёт структуры задачи RL и формализма MDP ответ положительный. Само собой, такое обучение только будет страшно неэффективным как по времени работы, так и по требуемым объёмам данных.

1.2.3. Концепция model-free алгоритмов

Ещё одним возможным существенным изменением сеттинга задачи является наличие у агента прямого доступа к функции переходов $p(s' | s, a)$ и функции награды.

Определение 14: 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 .

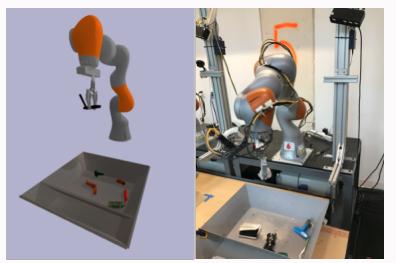
Симулятор — по сути копия среды, которую агент во время обучения может откатывать к произвольному состоянию. Симулятор позволяет агенту строить свою стратегию при помощи **планирования** (planning) — рассмотрения различных вероятных версий предстоящего будущего и использования их для текущего выбора действия. При использовании планирования в идеальном симуляторе никакого процесса непосредственного обучения в алгоритме может не быть, поскольку если на руках есть симулятор, то данные собирать не нужно: можно из симулятора сколько захотим данных получить.

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 напрашиваются какие-то алгоритмы разумного перебора — «планирование».

13	2	3	12
9	11	1	10
	6	4	14
15	8	7	5

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 предоставляется 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, да и обучение подобного симулятора сопряжено с рядом других нюансов. Например, в сложных средах в описании состояний может хранится колоссальное количество информации, и построение моделей, предсказывающих будущее, может оказаться вычислительно неподъемной 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 15: Алгоритм RL классифицируется как **model-free**, если он не использует и не пытается выучить модель динамики среды $p(s' | s, a)$.

Пример 13: Очень похоже, что когда мы учимся кататься на велосипеде, мы обучаемся как-то в model-free режиме. Мы, находясь в некотором состоянии, не планируем, в какой конфигурации окажется велосипед при разных возможных поворотах наших рук, и вместо этого учимся в точности методом проб и ошибок: мы просто запоминаем, при каком ощущении положения тела как нужно поворачивать руль.

Здесь очень любопытно пофилософствовать на тему того, почему в таких задачах мы зачастую умеем «запоминать» на всю жизнь полученные знания, не разучиваясь ездить на велосипеде после долгих лет без тренировок. Например, нейросетевые модели, которые мы дальше будем обучать для решения задач RL, таким свойством не обладают, и, обучаясь на одном опыте, они старый забывают; если агента RL обучать кататься на велосипеде, а потом долго учить кататься на самокате, стратегия для велосипеда крайне вероятно «сломается».

Оказывается, человек тоже может разучиться ездить на велосипеде при помощи очень хитрой процедуры: нужно придумать такой «самокат», обучение езде на котором требует противоположных навыков, чем велосипед. В качестве такого «самоката» можно взять «обратный велосипед» («The Backwards Bicycle»): велосипед, в котором поворот руля влево отклоняет колесо вправо, и наоборот. Подробнее про этот эксперимент можно посмотреть в [этом видео](#). Интересно, что обе стратегии — и для езды на велосипеде, и для езды на «обратном велосипеде» — восстанавливаются после некоторой тренировки (причём как-то подозрительно резко, с каким-то «фазовым переходом») и в конечном счёте уживаются вместе.

1.2.4. On-policy vs Off-policy

В model-free алгоритмах сбор данных становится важной составной частью: определяя политику взаимодействия со средой (behavior policy), мы влияем на то, для каких состояний s, a мы получим сэмпл s' из функции переходов. Собираемые данные — траектории — алгоритм может запоминать, например, в памяти. Но не каждый алгоритм RL сможет пользоваться такими сохранёнными данными, и поэтому возникает ещё одна важная классификация RL алгоритмов.

Определение 16: Алгоритм RL называется **off-policy**, если он может использовать для обучения опыт взаимодействия произвольной стратегии.

Определение 17: 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: обычно в нашем алгоритме явно или неявно будет присутствовать некоторая «текущая», «наилучшая» найденная стратегия π : та самая целевая политика (target policy), которую алгоритм выдаст в качестве ответа, если его работу прервать. Если алгоритм умеет улучшать её (проводить очередную итерацию обучения), используя сэмплы любой другой произвольной стратегии μ , то мы проводим «off-policy» обучение: обучаем политику «не по ней же самой». On-policy алгоритмам будет необходимо «отправлять в среду конкретную стратегию», поскольку они будут способны улучшать стратегию π лишь

по сэмплам из неё же самой, будут «привязаны к ней»; это существенное ограничение. Другими словами, если обучение с подкреплением — обучение на основе опыта, то on-policy алгоритмы обучаются на своих ошибках, когда off-policy алгоритмы могут учиться как на своих, так и на чужих ошибках.

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 лучших траекторий), то мы всё равно будем относить его к on-policy, поскольку для каждой новой итерации алгоритма нужно будет снова собирать сколько-то данных. 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 клонирования поведения). 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.

1.2.5. 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 $J(\pi)$ (1.5) 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 5).

Затем в главе 6 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 7.

1.2.6. 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.

Конечно, на практике мы хотим получить как можно больший performance при наименьших затратах ресурсов. Время работы алгоритма (wall-clock time) и эффективность по сэмплам связаны между собой, но их следует различать в силу того, что в разных средах сбор данных — проведение одного шага взаимодействия в среде — может быть как относительно дешёвым, так и очень дорогим. Например, если мы говорим о реальных роботах, то один шаг взаимодействия со средой — сверхдорогая операция, по сравнению с которыми любые вычисления на компьютере можно считать очень дешёвыми. Тогда нам выгодно использовать алгоритм, который максимально эффективен по сэмплам. Если же среда задана симулятором, а у нас ещё и есть куча серверов, чтобы параллельно запустить много таких симуляторов, то сбор данных для нас становится дешёвой процедурой. Тогда выгодно не гнаться за sample efficiency и выбирать вычислительно дешёвые алгоритмы.

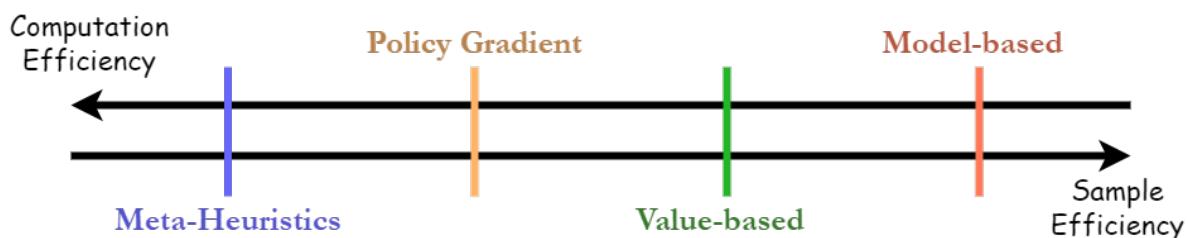
Очень условно классы RL алгоритмов располагаются по sample efficiency в следующем порядке: самыми неэффективными по требуемым объёмам данных алгоритмами являются мета-эвристики. Зато в них практически не будет никаких вычислений: очень условно, на каждое обновление весов модели будет приходиться сбор нескольких сотен переходов в среде. Их будет иметь смысл применять, если есть возможность параллельно собирать много данных.

Далее, в Policy Gradient подходе мы будем работать в on-policy режиме: текущая, целевая, политика будет отправляться в среду, сбирать сколько-то данных (например, мини-батч переходов, условно проводить порядка 32-64 шагов в среде) и затем делать одно обновление модели. Очень приближённо и с массой условностей говорят, что эффективность по сэмплам будет на порядок выше, чем у эволюционных алгоритмов, за счёт проведения на порядок большего количества вычислений: на одно обновление весов приходится сбор 30-60 переходов.

В value-based подходе за счёт off-policy режима работы можно будет контролировать соотношение числа собираемых переходов к количеству обновлений весов, но по умолчанию обычно полагают, что алгоритм собирает 1 переход и проводит 1 итерацию обучения. Таким образом, вычислений снова на порядок больше, как и (потенциально) sample efficiency.

Наконец, model-based вычислительно страшно тяжеловесные алгоритмы, но за счёт этого можно попытаться добиться максимальной эффективности по сэмплам.

Отразить четыре класса алгоритмов можно на условной картинке:



В зависимости от скорости работы среды, то есть времени, затрачиваемой на сбор данных, а также от особенностей среды (связанных непосредственно со спецификой этих четырёх классов алгоритмов), оптимальное соотношение ресурсы-качество может достигаться на разных классах алгоритмов. Но, к сожалению, на практике редко бывает очевидно, алгоритмы какого класса окажутся наиболее подходящими в конкретной ситуации.

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 встречавшаяся в ходе всего процесса.

1.2.7. Сложности задачи RL

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

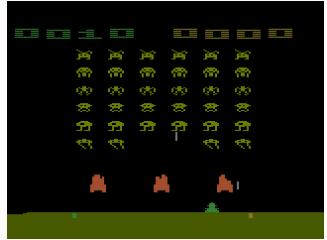
Проблема **застревания в локальных оптимумах** приходит напрямую из методов оптимизации. В оптимизируемом функционале (1.5) может существовать огромное количество стратегий π , для которых его значение далеко не максимально, но все в некотором смысле «соседние» стратегии дают в среднем ещё меньшую награду.

Пример 14: Часто агент может выучить тривиальное «пассивное» поведение, которое не приносит награды, но позволяет избегать каких-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 неходить 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 15: 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.



Если бы этого эффекта вдруг не было, и агент за каждое своё действие мгновенно получал бы всю награду $r(s, a)$, то решением задачи было бы банальное $\pi(s) = \operatorname{argmax}_a r(s, a)$ — жадная оптимизация функции награды. Очевидно, это никогда не решение: «часть» награды сидит в s' , в сэнпле следующего состояния, которого мы добились своим выбором действия, и в описании этого следующего состояния в силу свойства марковости обязана хранится информация о том, сколько времени осталось до получения награды за уже совершенные действия.

Смежная проблема — какое именно из многих совершенных 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 16: 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 17 — 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 (х-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 исследования-использования** (exploration-exploitation trade-off). Пока ограничимся лишь примером для иллюстрации.

Пример 18: 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 19: 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.

1.2.8. Дизайн функции награды

И есть ещё одна, вероятно, главная проблема⁷. **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.

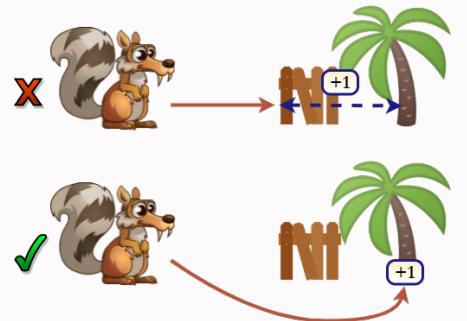
Пример 20 — «Взлом» функции награды: Классический пример того, как RL оптимизирует не то, что мы ожидаем, а ту награду, которую ему подсунули. Причина зацикливания агента видна в нижнем левом углу, где отображается счёт игры.

Пример 21: Попробуйте сформулировать функцию награды для следующих интеллектуальных задач:

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

Общее практическое правило звучит так: хорошая функция награды поощряет агента за достигнутые результаты, а не за то, каким способом агент этих результатов добивается. Это логично: если вдруг дизайнеру награды кажется, что он знает, как решать задачу, то вероятно, RL не особо и нужен. К сожалению, такая «хорошая» функция награды обычно разреженная.

Пример 22: Если вы хотите научиться доехать до дерева, то хорошая функция награды — выдать агенту +1 в момент, когда он доехал до дерева. Если попытаться поощрять агента каждый шаг в размере «минус расстояние до дерева», то может возникнуть непредвиденная ситуация: агент поедет прямо в забор, который стоит возле дерева, и это может оказаться с точки зрения такой награды выгоднее, чем делать длинный крюк по району вдали от дерева с целью этого самого забора объехать.



Есть, однако, один способ, как можно функцию награды модифицировать, не изменяя решаемую задачу, то есть эквивалентным образом. К сигналу «из среды» можно что-то добавлять при условии, что на следующем шаге это что-то обязательно будет вычтено. Другими словами, можно применять трюк, который называется **телеокопирующая сумма** (telescoping sums): для любой последовательности a_t , т. ч. $\lim_{t \rightarrow \infty} a_t = 0$, верно

$$\sum_{t \geq 0}^{\infty} (a_{t+1} - a_t) = -a_0 \quad (1.6)$$

Определение 19: Пусть дана некоторая функция $\Phi(s): \mathcal{S} \rightarrow \mathbb{R}$, которую назовём **потенциалом**, и которая удовлетворяет двум требованиям: она ограничена и равна нулю в терминальных состояниях. Будем говорить, что мы проводим **reward shaping** при помощи потенциала $\Phi(s)$, если мы заменим функцию награды по следующей формуле:

$$r^{\text{new}}(s, a, s') := r(s, a) + \gamma \Phi(s') - \Phi(s) \quad (1.7)$$

Теорема 3 — Reward Shaping: Проведение любого reward shaping по формуле (1.7) не меняет задачи.

Доказательство. Посчитаем суммарную награду для произвольной траектории. До reward shaping сум-

⁷ а точнее, в принципе главная проблема всей нашей жизни (what is your reward function?)

марная награда равнялась $\sum_{t \geq 0} \gamma^t r_t$. Теперь же суммарная награда равна

$$\sum_{t \geq 0} \gamma^t r_t + \sum_{t \geq 0} [\gamma^{t+1} \Phi(s_{t+1}) - \gamma^t \Phi(s_t)]$$

В силу свойства телескопирующей суммы (1.6), все слагаемые во второй сумме посокращаются, кроме первого слагаемого $-\Phi(s_0)$, который не зависит от стратегии взаимодействия и поэтому не влияет на итоговую задачу оптимизации, «последнего слагаемого», которое есть ноль: действительно, $\lim_{t \rightarrow \infty} \gamma^t \Phi(s_t) = 0$ в силу ограниченности функции потенциала. Если же в игре конечное число шагов T , не сокращающееся слагаемое $\Phi(s_T)$ есть значение потенциала в терминальном состоянии s_T , которая по условию также равно нулю. ■

Reward shaping позволяет поменять награду, «не ломая» задачу. Это может быть способом борьбы с разреженной функцией награды, если удаётся придумать какой-нибудь хороший потенциал. Конечно, для этого нужно что-то знать о самой задаче, и на практике reward shaping — это инструмент внесения каких-то априорных знаний, дизайна функции награды. Тем не менее, в будущем в главе 3.2 мы рассмотрим один хороший универсальный потенциал, который будет работать всегда.

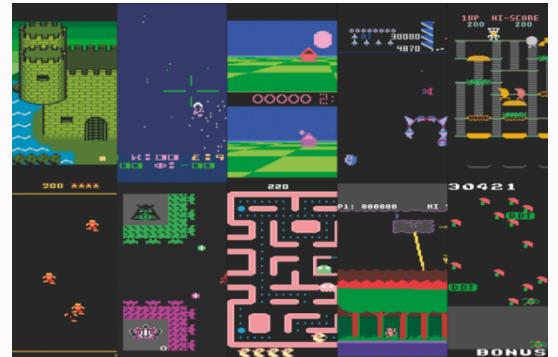
1.2.9. Бенчмарки

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 23 — 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.

При запуске алгоритмов на Atari обычно используется пре-процессинг. В общем случае MDP, заданный 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.

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. Если агент «видит» из-за этого только кратные кадры, могут встретиться неожиданные последствия: например, в Space Invaders каждые 4 кадра «исчезают» все выстрелы на экране, и агент может перестать их видеть.



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



В играх часто встречается понятие «жизней». Полезно считать, что потеря жизни означает конец эпизода, и выдавать в этот момент алгоритму флаг *done*.

Функция переходов в Atari — детерминированная; рандомизировано только начальное состояние. 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, действие повторяется только 2 раза, затем для очередного кадра подбрасывается монетка, и с вероятностью 0.5 агент повторяет действие снова; монетка подбрасывается снова, и так далее, пока не выпадет останов).

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. При этом по условию бенчмарка алгоритм должен быть запущен на всех 57 играх с одними и теми же настройками и гиперпараметрами.

Пример 24 — 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 обращаются 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 25 — Locomotion: 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 небольшим компактным векторочком.

Типичная задача заключается в том, чтобы в рамках представленной симуляции физики научиться добираться как можно дальше в двумерном или трёхмерном пространстве. Функция награды, описывающая такую задачу, не так проста: обычно она состоит из ряда слагаемых, дающих бонусы за продолжение движения, награду за коррелированность вектора скорости центра масс с желаемым направлением движения и штрафы за трату энергии.



Такая награда, хоть и является довольно плотной, обычно «плохо отнормирована»: суммарное значение за эпизод может быть довольно высоким. Нормировать награду «автоматически» могут в ходе самого обучения, считая, например, средний разброс встречаемых наград за шаг и деля награду на посчитанное стандартное отклонение. Распространено считать разброс не наград за шаг, а суммарных наград с начала эпизода, и делить награды за шаг на их посчитанное стандартное отклонение.

Несмотря на малую размерность пространства состояний и действий (случаи, когда нужно выдавать в качестве действия векторы размерности порядка 20, уже считаются достаточно тяжёлыми), а также информативную функцию награды, дающую постоянную обратную связь агенту, подобные задачи непрерывного управления обычно являются довольно сложными.



Пример 26: 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 объектах 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 20: 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 7.1. В теории мета-эвристик опция «вызвать 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 21: 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.

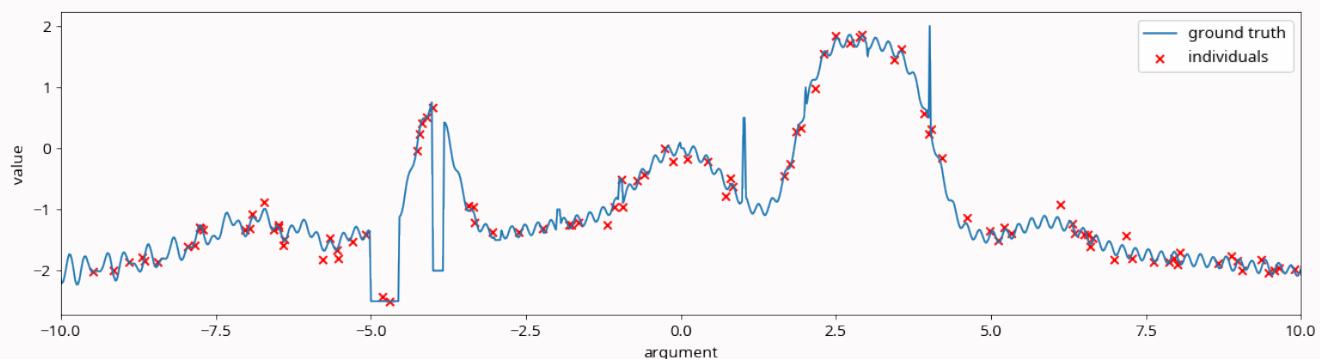
2.1.2. Случайный поиск

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

Определение 22: 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.

Пример 27 — Случайный поиск:



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

Определение 24: 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 25: 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 26: **Жадный** (greedy) отбор select_M^{top} — выбор топ- M самых приспособленных особей.

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.

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 неточны, и наилучшей на самом деле может оказаться особь с не самой высокой приспособленностью. В частности поэтому могут понадобиться альтернативы жадного отбора.

Пример 28 — 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 29 — 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: чем больше K , тем меньше у слабенькой особи шансов выжить.

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

Пример 30: 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, в исходном обходе мы проходили 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 максимальное. 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, случайнм перебором.

Пример 31 — Hill Climbing:

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.

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

Пример 32: В примере 30 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 с такой мутацией чревато скрым застrevанием 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 застrevать 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 33: 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 $\theta'_{k+1} \sim m(\theta'_{k+1} | \theta_k)$, 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 4 — 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 3: 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 маленькая, это распределение очень похоже на вырожденное в точке максимального значения оптимизируемой функции. Одновременно, правда, маленькая температура означает малую долю исследований в алгоритме, и тогда процедура 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 34 — Имитация отжига:



Если в операторе мутации есть параметр, отвечающий за «силу мутаций», связанный с балансом исследования-использования (например, дисперсия σ гауссовского шума в примере 33), его аналогично можно подвергнуть отжигу: в начале алгоритма его значение выставляется достаточно большим, после чего с ходом алгоритма оно постепенно по некоторому расписанию уменьшается.

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 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 гауссовых 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 28: Алгоритм называется **эволюционным** (evolutionary), если он строит последовательность популяций $\mathcal{P}_1, \mathcal{P}_2, \dots$, на k -ом шаге строя очередное *поколение* (generation) \mathcal{P}_k 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 отобранный 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 застревания 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 шансом 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

Вход: оракул $\hat{J}(\theta)$

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

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

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: $\mathcal{P}_{k+1} := (\hat{\theta}_i \sim m(\hat{\theta} | \theta) | \theta \in \mathcal{P}_k^+, i \in \{1, 2, \dots, K\})$

Пример 35 — $(4, 5)$ -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 пороговой функции Хевисайда ($f(x) := \mathbb{I}[x \geq 0]$).

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 мутироваться). Далее

³на заре нейросетевого подхода разумность использования полносвязных 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, а единственное ограничение на вычислительный граф — ацикличность.

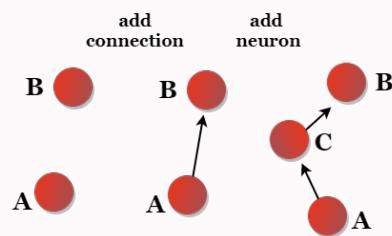
случайно выбирается, будет ли проходить мутация архитектуры (топологии) сети, и если да, то какого типа (обычно рассматривается несколько типов мутации).

Пример 36 — 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 добавляется ко входу в другой (вес инициализируется, например, 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 предопределённого набора.



Заметим, что в таком операторе мутации 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 36. 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 29): 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 исследуемых 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 слабые особи выживают в том числе там, где функция уже 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 стохастичным, в том числе чтобы быть вычислительно дешёвым.

Пример 37 — 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 $\tilde{\mathcal{P}} \subseteq \mathcal{P}$ — вид, а $\hat{J}_{mean}(\tilde{\mathcal{P}})$ — 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 $\tilde{\mathcal{P}}$ позволяет сгенерировать потомков пропорционально $\exp \hat{J}_{mean}(\tilde{\mathcal{P}})$ 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 применяется по несколько раз мутации.

Виды защищены мягким отбором, и поэтому заспавнившиеся 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 38 — Эволюция с видовой специализацией: В данном примере текущая популяция из 20 особей разделяется на виды согласно процедуре из примера 37 с метрикой $\rho(\theta_1, \theta_2) = |\theta_1 - \theta_2|$ и порогом 1. Для видов считается $\hat{J}_{mean}(\tilde{\mathcal{P}})$ (указан как fit в легенде), после чего при помощи сэмплирования из распределения $\propto \exp \hat{J}_{mean}(\tilde{\mathcal{P}})$ 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 «сочищая» 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 29: Кроссинговером (crossover) называется распределение $\mathbf{c}(\hat{\theta} | \theta_1, \theta_2)$, где θ_1, θ_2 называются *родителями* (parent), $\hat{\theta}$ — *потомком* (child).

Пример 39: 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. $\forall \hat{\theta}, \theta_1, \theta_2 \in \Theta: \mathbf{c}(\hat{\theta} | \theta_1, \theta_2) > 0$. 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: $c(\hat{\theta} \mid \theta_1, \theta_2)$ — кроссинговер, $m(\hat{\theta} \mid \theta)$ — мутация, $\text{select}(\mathcal{P}^+ \mid \mathcal{P}, \hat{J}(\mathcal{P}))$ — процедура отбора, $q(\theta)$ — стратегия перебора, N — размер популяции

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

На k -ом шаге:

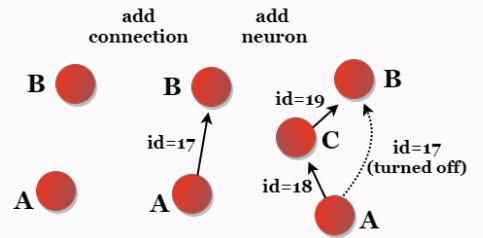
1. проводим отбор: $\mathcal{P}_k^+ \sim \text{select}(\mathcal{P}_k^+ \mid \mathcal{P}_k, \hat{J}(\mathcal{P}_k))$
2. проводим размножение: $\mathcal{P}_{k+1} := (\theta_i \sim c(\theta \mid \theta_l, \theta_r) \mid \theta_l, \theta_r \in \mathcal{P}_k^+)$
3. проводим мутацию: $\mathcal{P}_{k+1} \leftarrow (\hat{\theta} \sim m(\hat{\theta} \mid \theta) \mid \theta \in \mathcal{P}_{k+1})$



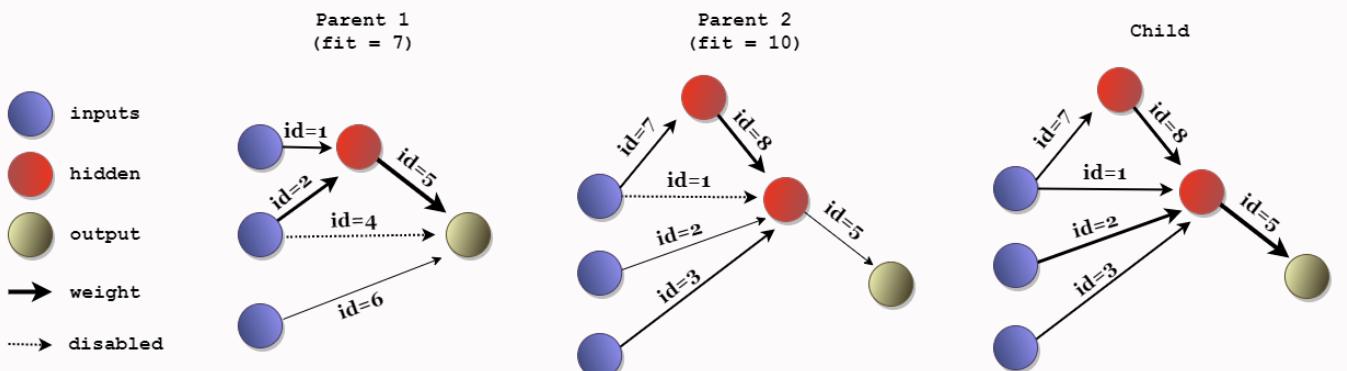
При эволюционном обучении нейросетей, в отличие от ряда других задач, кроссинговер во многом неудобен. Нельзя взять «половинку» одной хорошей нейросети и присоединить к «половинке» другой хорошей нейросети — каждый нейрон рассчитывает на тот набор входов, для которого он был обучен (неважно, эволюционно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

Пример 40 — 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 36); 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 разным задачам, с визуализацией получающейся сети (например, *Mario*).

§2.2. Эволюционные стратегии

2.2.1. Идея эволюционных стратегий

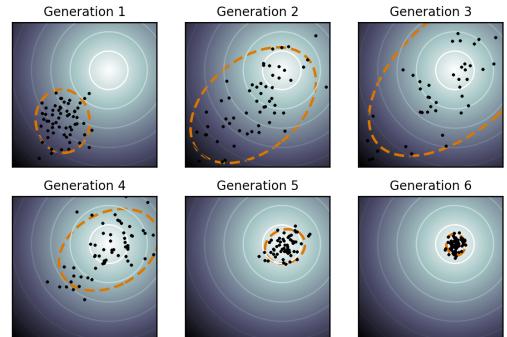
Когда мы пытаемся генерировать $k+1$ -ое поколение, используя особей k -го поколения в качестве исходного материала, единственная зависимость от всей истории заложена в составе k -ой популяции. Мы можем рассмотреть распределение, из которого появляются особи очередной популяции: весь смысл нашей процедуры 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 30: Распределение $q(\theta | \lambda_k)$, из которого генерируются особи k -ой популяции, называется **эволюционной стратегией** (evolutionary strategy, ES):

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

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 напушпать 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

Первую идею возьмём немного сбоку. Допустим, стоит задача оценки вероятности редкого события:

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

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

Под словами «редкое событие» подразумевается, что условие внутри индикатора $f(x) \geq \gamma$ выполняется с вероятностью, крайне близкой к нулю. Это означает, что Монте-Карло оценка с разумным на практике числом сэмпллов 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:

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. Можно считать, что здесь ключевая идея заключается в том, что мы явно ищем это распределение в некотором параметрическом семействе.

Утверждение 4: 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 распределения. Подставим данное $q(x)$ в подынтегральную функцию:

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

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 можем пытаться приблизить в параметрическом семействе $q(x | \lambda)$, минимизируя, например, такую KL-дивергенцию⁵:

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

Единственное зависящее от параметров λ слагаемое называется *кросс-энтропией* (cross entropy) и даёт название методу.

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 самим себе решать ту же задачу лучше. А 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:

$$\lambda_{k+1} \leftarrow \operatorname{argmax}_{\lambda} \frac{1}{N} \sum_{j=1}^N \mathbb{I}[f(x_j) \geq \gamma_k] \frac{p(x_j)}{q(x_j | \lambda_k)} \log q(x_j | \lambda) \quad (2.4)$$

⁵здесь и всюду далее под обозначением $\text{const}(\lambda)$ мы будем подразумевать функции, константные относительно λ ; такие слагаемые не влияют на оптимизацию по λ и могут быть опущены.

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 $q(x | \lambda)$ 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:

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

и перепишем алгоритм 3 в условиях, когда порог γ «не ограничен», ну или что тоже самое, $\gamma := \max_{\theta} J(\theta)$. Формально мы также можем выбирать любое $p(x)$; положим $p(x) := q(x | \lambda_k)$, просто чтобы в задаче (2.4) сократилась importance sampling коррекция. Мы получим очень простой на вид алгоритм, в котором фактически на очередном шаге минимизируется такое расстояние:

$$\text{KL}(\mathbb{I}[J(x) \geq \gamma_k] q(x | \lambda_{k-1}) \| q(x | \lambda_k)) \rightarrow \min_{\lambda_k},$$

где первое распределение задано с точностью до нормировочной константы.

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

Вход: оракул $\hat{J}(\theta)$

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

Инициализируем λ_0 произвольно.

На k -ом шаге:

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

2. проводим отбор $\mathcal{P}_k^+ := \text{select}_M^{\text{top}}(\mathcal{P}_k)$

3. решаем задачу оптимизации:

$$\lambda_{k+1} \leftarrow \underset{\lambda}{\operatorname{argmax}} \sum_{\theta \in \mathcal{P}_k^+} \log q(\theta | \lambda)$$

Видно, что мы по сути действуем эволюционно: хотим генерировать при помощи распределения q точки из области, где значение функции велико; берём и сэмплируем несколько точек из текущего приближения; из сгенерированных отбираем те, где значение функции было наибольшим и учим методом максимального правдоподобия повторять эти точки. Поскольку некоторая доля плохих точек была выкинута из выборки, распределение, которое учит очередное $q(x | \lambda_k)$, лучше предыдущего. Это первый способ обучения эволюционных стратегий.

Пример 41 — Кросс-энтропийный метод для black-box оптимизации:

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

В обучении с подкреплением в кросс-энтропийном методе можно сделать ещё один очень интересный 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 генерируем 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: $\pi(a | s, \theta)$ — стратегия с параметрами θ , N — число сэмплов, M — порог отбора

Инициализируем θ_0 произвольно.

На k -ом шаге:

1. сэмплируем N траекторий $\mathcal{T}_1 \dots \mathcal{T}_N$ игр при помощи стратегии $\pi(a | s, \theta_k)$
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:

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

2.2.5. Натуральные эволюционные стратегии (NES)

Рассмотрим альтернативный вариант⁶ подбора параметров эволюционной стратегии $q(\theta | \lambda)$. Будем подбирать λ , исходя из следующего функционала:

$$g(\lambda) := \mathbb{E}_{\theta \sim q(\theta | \lambda)} J(\theta) \rightarrow \max_{\lambda} \quad (2.5)$$

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 5:

$$\nabla_{\lambda} g(\lambda) = \mathbb{E}_{\theta \sim q(\theta | \lambda)} \nabla_{\lambda} \log q(\theta | \lambda) J(\theta) \quad (2.6)$$

Доказательство.

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

⁶ смысл названия «натуральные эволюционные стратегии» (natural evolution strategies) будет объяснён позже.

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

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:

$$\nabla_{\lambda} \log q(\theta | \lambda) = \frac{\nabla_{\lambda} q(\theta | \lambda)}{q(\theta | \lambda)} \quad (2.7)$$

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

$$\begin{aligned}
(*) &= \int_{\Theta} q(\theta | \lambda) \frac{\nabla_{\lambda} q(\theta | \lambda)}{q(\theta | \lambda)} J(\theta) d\theta = \\
&= \{\text{замечаем градиент логарифма (2.7)}\} = \int_{\Theta} q(\theta | \lambda) \nabla_{\lambda} \log q(\theta | \lambda) J(\theta) d\theta = \\
&= \{\text{выделяем мат.ожидание}\} = \mathbb{E}_{\theta \sim q(\theta | \lambda)} \nabla_{\lambda} \log q(\theta | \lambda) J(\theta) \quad \blacksquare
\end{aligned}$$

Итак, есть следующая идея: сгенерируем популяцию \mathcal{P}_k при помощи $q(\theta | \lambda_k)$, после чего воспользуемся особенноями как сэмплами для несмещённой оценки градиента функционала (2.5), 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

$$q(\theta | \lambda) := \mathcal{N}(\lambda, \sigma^2 I_{h \times h}) \quad (2.8)$$

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 6: Для эволюционной стратегии (2.8) оценка градиента (2.5) равна

$$\nabla_{\lambda} g(\lambda) = \frac{1}{N\sigma^2} \sum_{\theta \in \mathcal{P}} \hat{J}(\theta)(\theta - \lambda) \quad (2.9)$$

Доказательство.

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

Достаточно подставить выражение в общую формулу (2.6). ■

Полученная формула легко интерпретируется: 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 («скрещиваем 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) между серверами. За счёт параллелизации удается так «обучать» сетки играть в одну игру Атари за 10 минут. Если у вас есть 1440 процессоров. Естественно, ни про какой sample efficiency речь не идет.

Пример 42 — 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 идея⁷: давайте возьмём N случайных направлений $\nu_1 \dots \nu_N \sim \mathcal{N}(0, I)$ и сделаем шаг по всем этим направлениям:

$$\theta_{k+1} := \theta_k + \alpha \sum_{i=0}^N \underbrace{\frac{\hat{J}(\theta + \sigma\nu_i) - \hat{J}(\theta)}{\sigma}}_{\text{приближение градиента}} \nu_i \quad (2.10)$$

где α — learning rate.

Утверждение 5: Формулы (2.10) и (2.9) 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, (2.10) оценивает тот же градиент, что и формула

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

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:

$$q(\theta | \lambda) := \mathcal{N}(\mu, \Sigma) \quad (2.11)$$

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

⁷ в 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:

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

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 (2.9) (с точностью до learning rate):

$$\mu_{k+1} = \mu_k + \alpha \sum_{\theta \in \mathcal{P}_k} \underbrace{\hat{J}(\theta)}_{\text{вес}} \underbrace{(\theta - \mu_k)}_{\substack{\text{«предлагаемое»} \\ \text{особью изменение}}} \quad (2.12)$$

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:

$$\Sigma_{k+1} := \Sigma_k + \alpha \sum_{\theta \in \mathcal{P}_k} \hat{J}(\theta) ((\theta - \mu_k)(\theta - \mu_k)^T - \Sigma_k) \quad (2.13)$$

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)⁸. Исторически к этим формулам пришли эвристически, но позже у формулы появилось теоретическое обоснование.

Теорема 7: Формулы (2.12) и (2.13) для обновления $\lambda = (\mu, \Sigma)$ являются формулами натурального градиентного спуска для (2.5).

Доказательство требует обсуждения такой большой темы, как натуральный градиентный спуск (см. приложение A.1), а вывод формулы потребует небольшого введения в Кронекерову алгебру, поэтому доказательство вынесено в приложение A.2. ■

Именно поэтому данный вид алгоритмов для обучения эволюционных стратегий называется «**натуральными**» (natural): считается, что функционал (2.5) корректнее оптимизировать именно при помощи натурального градиентного спуска, инвариантного к параметризации $q(\theta | \lambda)$, а не обычного.

Пример 43 — CMA-ES (упрощ.):

Пример 44: Эволюционные стратегии и особенно CMA-ES весьма успешно применимы в задачах непрерывного управления вроде Locomotion (пример 25), в которых нужно научить разных существ ходить. Много интересных примеров можно найти в [этом видео](#).

Если мы попробуем проделать с данным подходом (оптимизацией (2.5)) 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 5.

⁸мы бы пришли к немного другим формулам, если бы использовали метод максимального правдоподобия при условии неизвестного среднего (μ_k заменилось бы на μ_{k+1}):

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

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 вносят в нашу задачу некоторую «структуру»; мы сможем придумать более эффективные алгоритмы решения за счёт эксплуатации этой структуры.

Что значит «структурой»? Представим, что мы решаем некоторую абстрактную задачу последовательного принятия решения, 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 45 — Парадокс обжора: 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 6 — 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:

$$\mathbb{E}_{\mathcal{T}|s_t=s} R(\mathcal{T}_{t:}) = \{\text{утв. 6}\} = \mathbb{E}_{\mathcal{T}_{t:}|s_t=s} \underbrace{\mathbb{E}_{\mathcal{T}_{t:}|s_t=s} R(\mathcal{T}_{t:})}_{\text{не зависит от } \mathcal{T}_{t:t}} = \mathbb{E}_{\mathcal{T}_{t:}|s_t=s} R(\mathcal{T}_{t:})$$

Определение 31: Для траектории \mathcal{T} величина

$$R_t := R(\mathcal{T}_{t:}) = \sum_{\hat{t} \geq t} \gamma^{\hat{t}-t} r_{\hat{t}} \quad (3.1)$$

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 7: Будущее определено текущим состоянием:

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

Доказательство. По определению:

$$p(\mathcal{T}_{t:} | s_t = s) = \prod_{\hat{t} \geq t} p(s_{\hat{t}+1} | s_{\hat{t}}, a_{\hat{t}}) \pi(a_{\hat{t}} | s_{\hat{t}}) = (*)$$

Воспользуемся однородностью MDP и однородностью стратегии, а именно:

$$\begin{aligned} \pi(a_{\hat{t}} | s_{\hat{t}} = s) &= \pi(a_0 | s_0 = s) \\ p(s_{\hat{t}+1} | s_{\hat{t}} = s, a_{\hat{t}}) &= p(s_1 | s_0 = s, a_0) \\ \pi(a_{\hat{t}+1} | s_{\hat{t}+1}) &= \pi(a_1 | s_1) \\ p(s_{\hat{t}+2} | s_{\hat{t}+1}, a_{\hat{t}+1}) &= p(s_2 | s_1, a_1) \end{aligned}$$

и так далее, получим:

$$(*) = \prod_{t \geq 0} p(s_{t+1} | s_t, a_t) \pi(a_t | s_t) = p(\mathcal{T} | s_0 = s) \quad \blacksquare$$

Утверждение 8: 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.

$$\mathbb{E}_{\mathcal{T}|s_0=s} f(\mathcal{T}) = \{\text{утв. 7}\} = \mathbb{E}_{\mathcal{T}_{t:}|s_t=s} f(\mathcal{T}_{t:}) = \{\text{утв. 6}\} = \mathbb{E}_{\mathcal{T}|s_t=s} f(\mathcal{T}_{t:}) \quad \blacksquare$$

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

$$V^\pi(s) := \mathbb{E}_{\mathcal{T} \sim \pi | s_0 = s} R(\mathcal{T}) \quad (3.2)$$

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 9: Для любого t верно:

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

Пояснение. Применить утверждение 8 для $R(\mathcal{T})$. ■

Утверждение 10: $V^\pi(s)$ ограничено.

Утверждение 11: Для терминальных состояний $V^\pi(s) = 0$.

Заметим, что 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 46: 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 C: там агент всегда выбирает действие ■, получает -1, и эпизод заканчивается: $V^\pi(s = C) = -1$.

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

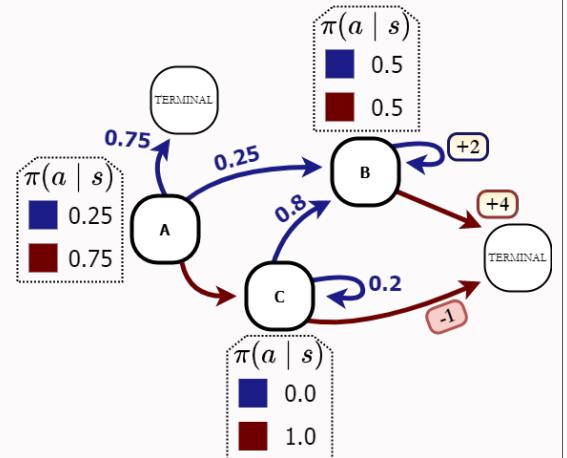
$$V^\pi(s = B) = \underbrace{0.5 \cdot 4 + 0.5 \cdot (2 + \gamma V^\pi(s = B))}_{■ ■}$$

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

Для состояния A достаточно аналогично рассмотреть все дальнейшие события:

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

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



3.1.3. Уравнения Беллмана

Если s_0 — стартовое состояние, то $V^\pi(s_0)$ по определению и 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 (листья соответствуют терминальным состояниям — там ноль), затем в узлах перед листьями, ну и так далее индуктивно добраться до корня.

Мы заметили, что в примере 46 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 8 — Уравнение Беллмана (Bellman expectation equation) для V^π :

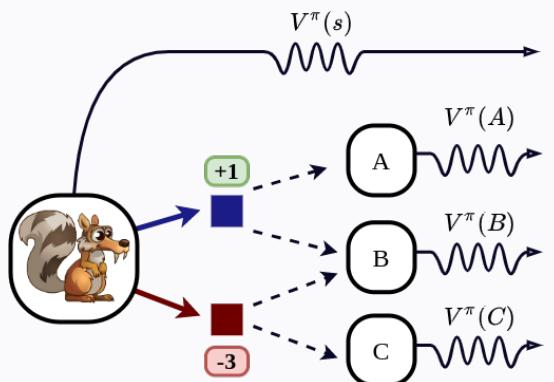
$$V^\pi(s) = \mathbb{E}_a [r(s, a) + \gamma \mathbb{E}_{s'} V^\pi(s')] \quad (3.3)$$

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 9 о независимости V -функции от времени:

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



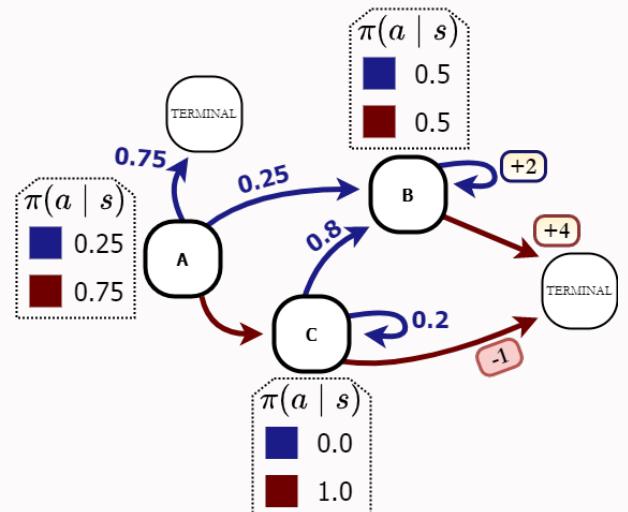
Пример 47: Выпишем уравнения Беллмана для MDP и стратегии π из примера 46. Число уравнений совпадает с числом состояний. Разберём подробно уравнение для состояния A :

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

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.

$$\begin{aligned} V^\pi(A) &= \frac{1}{16} \gamma V^\pi(B) + \frac{3}{4} \gamma V^\pi(C) \\ V^\pi(B) &= 0.5 (2 + \gamma V^\pi(B)) + 0.5 \cdot 4 \\ V^\pi(C) &= -1 \end{aligned}$$

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:

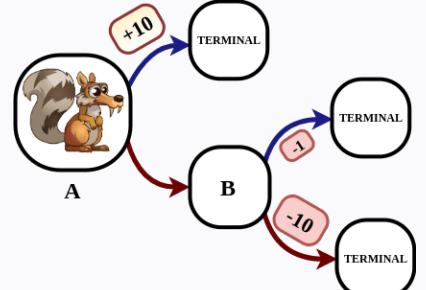
Определение 33: Политика π^* оптимальна, если $\forall \pi: V^{\pi^*}(s_0) \geq V^\pi(s_0)$.

Введём альтернативное определение:

Определение 34: Политика π^* оптимальна, если $\forall \pi, s: V^{\pi^*}(s) \geq V^\pi(s)$.

Теорема 9: 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 ■ и закончить игру. Поскольку оптимальный агент выберет ■ 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 ■ . ■



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

$$V^*(s) = \max_{\pi} V^\pi(s) \quad (3.4)$$

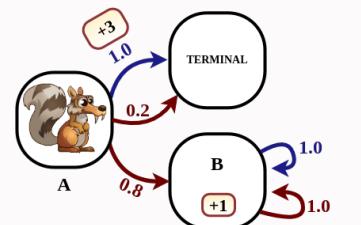
Пока нет никаких обоснований, что найдётся стратегия, которая максимизирует $V^\pi(s)$ сразу для всех состояний. Вдруг в одних s максимум (3.4) достигается на одной стратегии, а в другом — на другой? Тогда оптимальных стратегий в сильном смысле вообще не существует, хотя формальная величина (3.4) существует. Пока заметим лишь, что для ситуации, когда MDP — дерево, существование оптимальной стратегии в смысле второго определения можно опять показать «от листьев к корню».

Пример 48: Рассмотрим MDP из примера 9; $\gamma = \frac{10}{11}$, множество стратегий параметризуется единственным числом $\theta := \pi(a = \text{■} | s = A)$.

По определению оптимальная V -функция для состояния A равна

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

Оценочные функции для состояния B 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 36: Для данного MDP *Q-функцией* (state-action value function, action quality function) для данной стратегии π называется

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

Теорема 10 — Связь оценочных функций: V -функции и Q -функции взаимозависимы, а именно:

$$Q^\pi(s, a) = r(s, a) + \gamma \mathbb{E}_{s'} V^\pi(s') \quad (3.5)$$

$$V^\pi(s) = \mathbb{E}_{a \sim \pi(a|s)} Q^\pi(s, a) \quad (3.6)$$

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

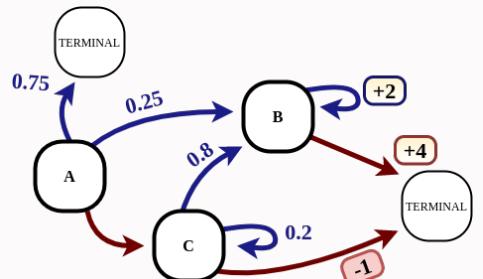
$$Q^\pi(s, a) = r(s, a) + \gamma \mathbb{E}_{s'} \mathbb{E}_a Q^\pi(s', a') \quad (3.7)$$

Доказательство. Можно воспользоваться (3.5) + (3.6), можно расписать как награду на следующем шаге плюс хвостик. ■

Пример 49: Q -функция получает на вход пару состояние-действие и ничего не говорит 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 ■. Мы тем не менее можем посчитать $Q^\pi(s, ■)$ для любых состояний (например, для терминальных это значение формально равно нулю). Сделаем это при помощи QV уравнения:

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



Внутри V^π сидит дальнейшее поведение при помощи стратегии π , то есть выбор исключительно действий ■: 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 37: Для данного MDP *оптимальной Q -функцией* (optimal Q-function) называется

$$Q^*(s, a) := \max_{\pi} Q^\pi(s, a) \quad (3.8)$$

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

Утверждение 12:

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

Доказательство.

$$\begin{aligned} Q^*(s, a) &= \{\text{определение } Q^* \text{ (3.8)}\} = \max_{\pi} Q^\pi(s, a) = \\ &= \{\text{связь QV (3.5)}\} = \max_{\pi} [r + \gamma \mathbb{E}_{s'} V^\pi(s')] \leq \\ &\leq \{\text{максимум среднего не превосходит среднее максимума}\} \leq r + \gamma \mathbb{E}_{s'} \max_{\pi} V^\pi(s') = \\ &= \{\text{определение } V^* \text{ (3.4)}\} = r + \gamma \mathbb{E}_{s'} V^*(s') \end{aligned}$$

■

Равенство в месте с неравенством случилось бы, если бы мы доказали следующий факт: что вообще существует такая стратегия π , которая максимизирует V^π сразу для всех состояний s одновременно (и которую мы определили как оптимальную). Другими словами, нужно показать, что максимизация $V^\pi(s)$ для одного состояния «помогает» максимизировать награду для других состояний. Для V -функции мы можем построить аналогичную оценку сверху:

Утверждение 13:

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

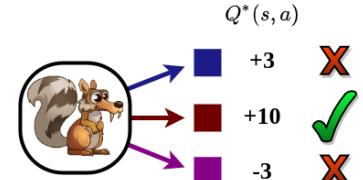
Доказательство.

$$\begin{aligned} V^*(s) &= \{\text{определение } V^* \text{ (3.4)}\} = \max_{\pi} V^{\pi}(s) = \\ &= \{\text{связь VQ (3.6)}\} = \max_{\pi} \mathbb{E}_{a \sim \pi(a|s)} Q^{\pi}(s, a) \leq \\ &\leq \{\text{по определению } Q^* \text{ (3.8)}\} \leq \max_{\pi} \mathbb{E}_{a \sim \pi(a|s)} Q^*(s, a) \leq \\ &\leq \{\text{свойство } \mathbb{E}_x f(x) \leq \max_x f(x)\} \leq \max_a Q^*(s, a) \end{aligned}$$

■

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 (3.8). 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 «дальнейшей оптимальности своего поведения», надеемся получить из текущего состояния $\max_a Q^*(s, a)$.



Определение 38: Для данного приближения Q-функции стратегия $\pi(s) := \operatorname{argmax}_a Q(s, a)$ называется **жадной** (greedy with respect to Q-function).

Определение 39: *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». Если в поддереве, соответствующем одному действию, можно получить больше, чем в другом поддереве, то понятно, что выбирать нужно его. 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), а затем в секции 3.2 про Policy Improvement мы поймём, что все желаемые утверждения можно получить и через него.

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 9 о независимости оценочной функции от времени тут перестаёт быть истинной, и, вообще говоря, оценочные функции теперь зависят от текущего момента времени t .

Определение 40: Для данного MDP и нестационарной стратегии $\pi = \{\pi_t(a | s) | t \geq 0\}$ обозначим её **оценочные функции** как

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

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

Пример 50: 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 .

¹мы 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.

Утверждение 14: Для нестационарных оценочных функций остаются справедливыми уравнения Беллмана:

$$V_t^\pi(s) = \mathbb{E}_{\pi_t(a|s)} Q_t^\pi(s, a) \quad (3.9)$$

$$Q_t^\pi(s, a) = r(s, a) + \gamma \mathbb{E}_{s'} V_{t+1}^\pi(s') \quad (3.10)$$

Доказательство. Всё ещё следует из определений. ■

Определение 41: Для данного MDP *оптимальными оценочными функциями среди нестационарных стратегий* назовём

$$V_t^*(s) := \max_{\substack{\pi_t \\ \pi_{t+1} \\ \dots}} V_t^\pi(s) \quad (3.11)$$

$$Q_t^*(s, a) := \max_{\substack{\pi_{t+1} \\ \pi_{t+2} \\ \dots}} Q_t^\pi(s, a) \quad (3.12)$$

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

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

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 12: Стратегия $\pi_t(s) := \operatorname{argmax}_a Q_t^*(s, a)$ оптимальна, то есть для всех состояний s верно $V_t^\pi(s) = V_t^*(s)$, и при этом справедливо:

$$V_t^*(s) = \max_a Q_t^*(s, a) \quad (3.13)$$

Доказательство. В силу VQ уравнения (3.9), 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 (3.12) 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. Сначала 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 частности*, 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 верхней оценки при этой π_t^* и том наборе $\pi_{t+1}^*, \pi_{t+2}^*, \dots$, на котором достигается значение $Q_t^*(s, \pi_t^*(s))$. ■

* здесь 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 $\operatorname{Argmax}_a Q_t^*(s, a)$.

Утверждение 16: Для нестационарных оценочных функций верно:

$$Q_t^*(s, a) = r(s, a) + \gamma \mathbb{E}_{s'} V_{t+1}^*(s') \quad (3.14)$$

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

$$\begin{aligned} Q_t^*(s, a) &= \max_{\substack{\pi_{t+1} \\ \pi_{t+2} \\ \dots}} Q_t^\pi(s, a) = \{\text{связь QV (3.10)}\} = \max_{\substack{\pi_{t+1} \\ \pi_{t+2} \\ \dots}} [r(s, a) + \gamma \mathbb{E}_{s'} V_{t+1}^\pi(s')] \leq \\ &\leq \{\text{определение } V^* \text{ (3.11)}\} \leq r(s, a) + \gamma \mathbb{E}_{s'} V_{t+1}^*(s') \end{aligned}$$

Эта верхняя оценка достигается на стратегии $\pi_t(s) = \operatorname{argmax}_a Q_t^*(s, a)$, на которой, как мы доказали в предыдущей теореме 12, 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 17: 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 12 максимум справа достигается на детерминированной $\pi_t(s) = \operatorname{argmax}_a Q_t^*(a, s)$. В силу утверждения 15, для всех моментов времени Q_t^* совпадают, следовательно такая π_t тоже совпадает для всех моментов времени и является стационарной стратегией. ■

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

3.1.9. Уравнения оптимальности Беллмана

Теорема 13 — Связь оптимальных оценочных функций:

$$V^*(s) = \max_a Q^*(s, a) \quad (3.15)$$

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

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):

$$Q^*(s, a) = r(s, a) + \gamma \mathbb{E}_{s'} \max_{a'} Q^*(s', a') \quad (3.17)$$

$$V^*(s) = \max_a [r(s, a) + \gamma \mathbb{E}_{s'} V^*(s')] \quad (3.18)$$

Доказательство. Подставили (3.15) в (3.16) и наоборот. ■

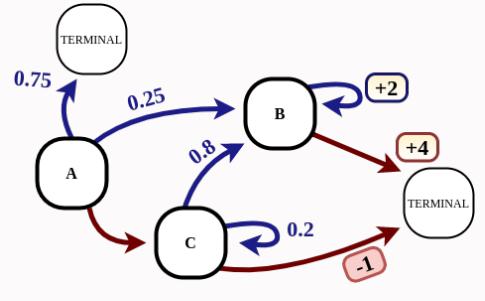
Хотя для строгого доказательства нам и пришлось поднапрячься и выписать относительно громоздкое рассуждение, уравнения оптимальности Беллмана крайне интуитивны. Для Q^* , например, можно рассудить так: что даст оптимальное поведение из состояния s после совершения действия a ? Что с нами случится дальше: 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 51: Сформулируем для MDP с рисунка уравнения оптимальности Беллмана для V^* . Мы получим систему из трёх уравнений с трёмя неизвестными.

$$V^*(s = A) = \max(0.25\gamma V^*(s = B), \gamma V^*(s = C))$$

$$V^*(s = B) = \max(2 + \gamma V^*(s = B), 4)$$

$$V^*(s = C) = \max(0.8\gamma V^*(s = B) + 0.2\gamma V^*(s = C), -1)$$



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 $S \times A \rightarrow \mathbb{R}$, 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 (3.15):

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

С другой стороны из связи VQ (3.6) 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 :

$$Q^\pi(s, a) = \{\text{связь } QQ \text{ (3.7)}\} = r(s, a) + \gamma \mathbb{E}_{s'} \mathbb{E}_{\pi(a'|s')} Q^\pi(s', a') = r(s, a) + \gamma \mathbb{E}_{s'} \max_{a'} Q^\pi(s', a')$$

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 позволяют 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 18: Если $|\mathcal{A}| < +\infty$, всегда существует оптимальная стратегия.

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

Утверждение 19: Оптимальной стратегии может не существовать.

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 $\underset{a}{\operatorname{Argmax}} Q^*(a)$ пуст. ■

Утверждение 20: Если существует хотя бы две различные оптимальные стратегии, то существует континуум оптимальных стратегий.

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

Утверждение 21: Если существует хотя бы одна оптимальная стратегия, то существует детерминированная оптимальная стратегия.

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

Пример 52: Найдём все оптимальные стратегии в MDP из примера 51 для $\gamma = 0.5$.

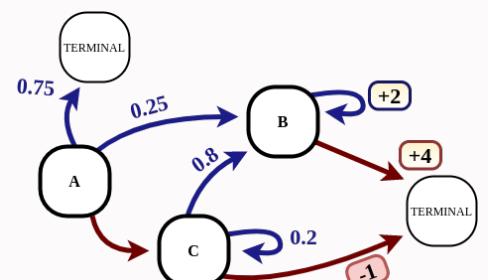
Мы могли бы составить уравнения оптимальности Беллмана для Q^* и решать их, но сделаем чуть умнее и воспользуемся критерием оптимальности Беллмана (теорема 15). Например, в состоянии В оптимально или выбирать какое-то одно из двух действий с вероятностью 1, или действия эквивалентны, и тогда оптимально любое поведение. Допустим, мы будем выбирать всегда $\boxed{\text{■}}$, тогда мы получим $\frac{2}{1-\gamma} = 4$; если же будем выбирать $\boxed{\text{■}}$, то получим $+4$. Значит, действия эквивалентны, оптимально любое поведение, и $V^*(s = B) = 4$.

Проведём аналогичное рассуждение для состояния С. Если оптимально действие $\boxed{\text{■}}$, то

$$Q^*(s = C, \boxed{\text{■}}) = 0.2\gamma Q^*(s = C, \boxed{\text{■}}) + 0.8\gamma V^*(s = B)$$

Решая это уравнение относительно неизвестного $Q^*(s = C, \boxed{\text{■}})$, получаем $\frac{16}{9} > Q^*(s = C, \boxed{\text{■}}) = -1$. Значит, в С оптимальная стратегия обязана выбирать $\boxed{\text{■}}$, и $V^*(C) = \frac{16}{9}$.

Для состояния А достаточно сравнить $Q^*(s = A, \boxed{\text{■}}) = 0.25\gamma V^*(s = B) = \frac{1}{4}$ и $Q^*(s = A, \boxed{\text{■}}) = -\gamma V^*(s = C) = \frac{8}{9}$, определив, что оптимальная стратегия должна выбирать $\boxed{\text{■}}$.



§3.2. Улучшение политики

3.2.1. Advantage-функция

Допустим, мы находились в некотором состоянии s , и засэмплировали $a \sim \pi(a | s)$ такое, что $Q^\pi(s, a) > V^\pi(s)$. Что можно сказать о таком действии? Мы знаем, что вообще в среднем политика π набирает из данного состояния $V^\pi(s)$, но какой-то выбор действий даст в итоге награду больше $V^\pi(s)$, а какой-то меньше. Если $Q^\pi(s, a) > V^\pi(s)$, то после того, как мы выбрали действие a , «приняли решение», наша средняя будущая награда вдруг увеличилась.

Мы ранее обсуждали в разделе 1.2.7 такую особую проблему обучения с подкреплением, как credit assignment, которая звучит примерно так: допустим, мы засэмплировали траекторию s, a, s', a', \dots до конца эпизода, и в конце в финальном состоянии через T шагов получили сигнал (награду) $+1$. Мы приняли T решений, но какое

из всех этих действий повлекло получение этого +1? «За что нас наградили?» Повлияло ли на получение +1 именно то действие a , которое мы засемплировали в стартовом s ? Вопрос нетривиальный, потому что в RL есть отложенный сигнал: возможно, именно действие a в состоянии s запустило какую-нибудь цепочку действий, которая дальше при любом выборе a', a'', \dots приводит к награде +1. Возможно, конечно, что первое действие и не имело никакого отношения к этой награде, и это поощрение именно за последний выбор. А ещё может быть такое, что имело место везение, и просто среда в какой-то момент перекинула нас в удачное состояние.

Но мы понимаем, что если какое-то действие «затриггило» получение награды через сто шагов, в промежуточных состояниях будет информация о том, сколько времени осталось до получения этой отложенной награды. Например, если мы выстрелили во вражеский инопланетный корабль, и через 100 шагов выстрел попадает во врага, давая агенту +1, мы будем видеть в состояниях расстояние от летящего выстрела до цели, и знать, что через такое-то время нас ждёт +1. Другими словами, вся необходимая информация лежит в идеальных оценочных функциях Q^π и V^π .

Так, если в некотором состоянии s засемплировалось такое a , что $Q^\pi(s, a) = V^\pi(s)$, то мы можем заключить, что выбор действия на этом шаге не привёл ни к какой «неожиданной» награде. Если же $Q^\pi(s, a) > V^\pi(s)$ — то мы приняли удачное решение, $Q^\pi(s, a) < V^\pi(s)$ — менее удачное, чем обычно. Если, например, $r(s, a) + V^\pi(s') > Q^\pi(s, a)$, то мы можем заключить, что имело место везение: среда засемплировала такое s' , что теперь мы получим больше награды, чем ожидали после выбора a в состоянии s . И так далее: мы сможем отследить, в какой конкретно момент случилось то событие (сэмплирование действия или ответ среды), за счёт которого получена награда.

Таким образом, идеальный «кредит» влияния действия a , выбранного в состоянии s , на будущую награду равен

$$Q^\pi(s, a) - V^\pi(s),$$

и именно эта величина на самом деле будет для нас ключевой. Поэтому из соображений удобства вводится ещё одно обозначение:

Определение 42: Для данного MDP *Advantage-функцией* политики π называется

$$A^\pi(s, a) := Q^\pi(s, a) - V^\pi(s) \quad (3.19)$$

Утверждение 22: Для любой политики π и любого состояния s :

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

Доказательство.

$$\begin{aligned} \mathbb{E}_{\pi(a|s)} A^\pi(s, a) &= \mathbb{E}_{\pi(a|s)} Q^\pi(s, a) - \mathbb{E}_{\pi(a|s)} V^\pi(s) = \\ \{V^\pi \text{ не зависит от } a\} &= \mathbb{E}_{\pi(a|s)} Q^\pi(s, a) - V^\pi(s) = \\ \{\text{связь } V \text{ через } Q \text{ (3.6)}\} &= V^\pi(s) - V^\pi(s) = 0 \end{aligned}$$

■

Утверждение 23: Для любой политики π и любого состояния s :

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

Advantage — это, если угодно, «центрированная» Q-функция. Если $A^\pi(s, a) > 0$ — действие a «лучше среднего» для нашей текущей политики в состоянии s , меньше нуля — хуже. И интуиция, что процесс обучения нужно строить на той простой идеи, что первые действия надо выбирать чаще, а вторые — реже, нас не обманывает.

Естественно, подвох в том, что на практике мы не будем знать точное значение оценочных функций, а значит, и истинное значение Advantage. Решая вопрос оценки значения Advantage для данной пары s, a , мы фактически будем проводить credit assigment — это одна и та же задача.

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 открывает глаза на мир. Для этого вспомним формулу reward shaping-a (1.7) и заметим, что мы можем выбрать в качестве потенциала V-функцию произвольной стратегии π_2 :

$$\Phi(s) := V^{\pi_2}(s)$$

Действительно, требований к потенциальному два: ограниченность (для V-функций это выполняется в силу наших ограничений на рассматриваемые MDP) и равенство нулю в терминальных состояниях (для V-функций это

верно по определению). Подставив такой потенциал, мы получим связь между performance-ом $J(\pi) = V^\pi(s_0)$ 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 :

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

Доказательство.

$$\begin{aligned} V^{\pi_2}(s) - V^{\pi_1}(s) &= \mathbb{E}_{\mathcal{T} \sim \pi_2 | s_0 = s} \sum_{t \geq 0} \gamma^t r_t - V^{\pi_1}(s) = \\ &= \mathbb{E}_{\mathcal{T} \sim \pi_2 | s_0 = s} \left[\sum_{t \geq 0} \gamma^t r_t - V^{\pi_1}(s_0) \right] = \\ \{\text{телескопирующая сумма (1.6)}\} &= \mathbb{E}_{\mathcal{T} \sim \pi_2 | s_0 = s} \left[\sum_{t \geq 0} \gamma^t r_t + \sum_{t \geq 0} [\gamma^{t+1} V^{\pi_1}(s_{t+1}) - \gamma^t V^{\pi_1}(s_t)] \right] = \\ \{\text{перегруппируем слагаемые}\} &= \mathbb{E}_{\mathcal{T} \sim \pi_2 | s_0 = s} \sum_{t \geq 0} \gamma^t (r_t + \gamma V^{\pi_1}(s_{t+1}) - V^{\pi_1}(s_t)) = \\ \{\text{фокус } \mathbb{E}_x f(x) = \mathbb{E}_x \mathbb{E}_x f(x)\} &= \mathbb{E}_{\mathcal{T} \sim \pi_2 | s_0 = s} \sum_{t \geq 0} \gamma^t (r_t + \gamma \mathbb{E}_{s_{t+1}} V^{\pi_1}(s_{t+1}) - V^{\pi_1}(s_t)) = \\ \{\text{выделяем Q-функцию (3.5)}\} &= \mathbb{E}_{\mathcal{T} \sim \pi_2 | s_0 = s} \sum_{t \geq 0} \gamma^t (Q^{\pi_1}(s_t, a_t) - V^{\pi_1}(s_t)) \\ \{\text{по определению (3.19)}\} &= \mathbb{E}_{\mathcal{T} \sim \pi_2 | s_0 = s} \sum_{t \geq 0} \gamma^t A^{\pi_1}(s_t, a_t) \quad \blacksquare \end{aligned}$$

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

Конечно, это теоретическое утверждение, поскольку на практике узнать точно оценочную функцию какой-то другой стратегии достаточно сложно (хотя ничто не мешает в качестве потенциала использовать произвольную функцию, приближающую $V^{\pi_1}(s)$). Однако в этой «новой» награде замешаны сигналы из будущего, награды, которые будут получены через много шагов, и эта «новая» награда априори информативнее исходной $r(s, a)$.

Представим, что мы оптимизировали исходный функционал

$$\mathbb{E}_{\mathcal{T} \sim \pi_2 | s_0 = s} \sum_{t \geq 0} \gamma^t r(s_t, a_t) \rightarrow \max_{\pi_2}$$

и сказали: слушайте, мы не знаем, как управлять марковской цепью, не очень понимаем, как выбор тех или иных действий в состоянии влияет на структуру траектории $p(\mathcal{T} | \pi_2)$. А давайте мы притворимся, что у нас нет в задаче отложенного сигнала (что очень существенное упрощение), и будем просто во всех состояниях s оптимизировать $r(s, a)$: выбирать «хорошие» действия a , где функция награды высокая. То есть будем просто выбирать $\pi_2(s) = \operatorname{argmax}_a r(s, a)$. Смысла в этом будет мало.

Теперь же мы преобразовали функционал, сменив функцию награды:

$$\mathbb{E}_{\mathcal{T} \sim \pi_2 | s_0 = s} \sum_{t \geq 0} \gamma^t A^{\pi_1}(s_t, a_t) \rightarrow \max_{\pi_2}$$

Что, если мы поступим также с новой наградой? Мы, например, знаем, что Advantage — не произвольная функция, и она обязана в среднем равняться нулю (утв. 22). Значит, если мы выберем

$$\pi_2(s) = \operatorname{argmax}_a A^{\pi_1}(s, a),$$

то все встречаемые пары (s, a) в траекториях из π_2 будут обязательно с неотрицательными наградами за шаг $A^{\pi_1}(s, a) \geq 0$. Значит и вся сумма наград будет положительна для любого стартового состояния:

$$\mathbb{E}_{\mathcal{T} \sim \pi_2 | s_0 = s} \sum_{t \geq 0} \gamma^t \underbrace{A^{\pi_1}(s_t, a_t)}_{\geq 0} \geq 0$$

И тогда из теоремы 16 об RPI мы можем заключить, что для любого s :

$$V^{\pi_2}(s) - V^{\pi_1}(s) \geq 0$$

Это наблюдение - ключ к оптимизации стратегии при известной оценочной функции другой стратегии.

3.2.3. Policy Improvement

Определение 43: Будем говорить, что стратегия π_2 «не хуже» π_1 (запись: $\pi_2 \succeq \pi_1$), если $\forall s$:

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

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

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

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 Q^{π_1} ; тогда мы знаем и V^{π_1} из VQ уравнения (3.6) и A^{π_1} по определению (3.19). Давайте попробуем построить $\pi_2 \succ \pi_1$. Для этого 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:

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

или, в эквивалентной форме:

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

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

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

$$\begin{aligned} V^{\pi_1}(s) &= \{\text{связь VQ (3.6)}\} = \mathbb{E}_{\pi_1(a|s)} Q^{\pi_1}(s, a) \leq \\ &= \{\text{по построению } \pi_2\} = \mathbb{E}_{\pi_2(a|s)} Q^{\pi_1}(s, a) = \\ &= \{\text{связь QV (3.5)}\} = \mathbb{E}_{\pi_2(a|s)} [r + \gamma \mathbb{E}_{s'} V^{\pi_1}(s')] \leq \\ &\leq \{\text{применяем это же неравенство рекурсивно}\} = \mathbb{E}_{\pi_2(a|s)} [r + \mathbb{E}_{s'} \mathbb{E}_{\pi_2(a'|s')} [\gamma r' + \gamma^2 \mathbb{E}_{s''} V^{\pi_1}(s'')]] \leq \\ &\leq \{\text{раскручиваем цепочку далее}\} \leq \dots \leq \mathbb{E}_{\pi_2(a|s)} \sum_{t \geq 0} \gamma^t r_t = \\ &= \{\text{по определению (3.2)}\} = V^{\pi_2}(s) \end{aligned}$$

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. Улучшать стратегию можно прямо в отдельных состояниях, например, выбрав некоторое состояние s и сказав: неважно, как это повлияет на частоты посещения состояний, но будем конкретно в этом состоянии s выбирать действия так, что значение

$$\mathbb{E}_{\pi_2(a|s)} Q^{\pi_1}(s, a) \tag{3.21}$$

как можно больше. Тогда, если в s действие выбирается «новой» стратегией π_2 , а в будущем агент будет вести себя *не хуже*, чем π_1 , то и наберёт он в будущем не меньше $Q^{\pi_1}(s, a)$. Доказательство теоремы 17 показывает, что выражение (3.21) является нижней оценкой на награду, которую собирает «новый» агент со стратегией π_2 .

Если эта нижняя оценка поднята выше $V^{\pi_1}(s)$, то стратегию удалось улучшить: и тогда какой бы ни была π_1 , мы точно имеем гарантию $\pi_2 \succeq \pi_1$. Важно, что такой policy improvement работает всегда: и для «тупых» стратегий, близких к случайному поведению, и для уже умеющих что-то разумное делать.

В частности, мы можем попробовать нижнюю оценку (3.21) максимально поднять, то есть провести **жадный** (greedy) policy improvement. Для этого мы формально решаем такую задачу оптимизации:

$$\mathbb{E}_{\pi_2(a|s)} Q^{\pi_1}(s, a) \rightarrow \max_{\pi_2},$$

и понятно, что решение находится в детерминированной π_2 :

$$\pi_2(s) = \operatorname{argmax}_a Q^{\pi_1}(s, a) = \operatorname{argmax}_a A^{\pi_1}(s, a)$$

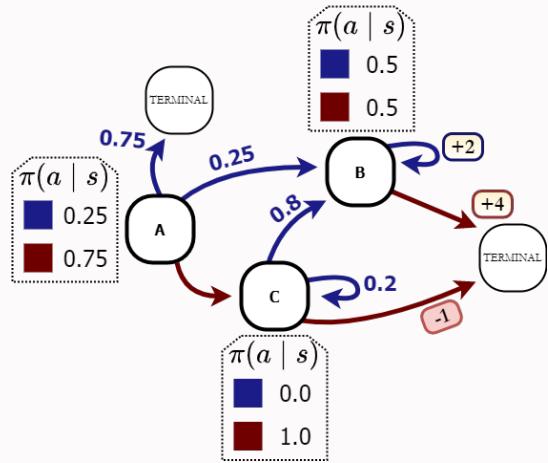
Конечно, мы так не получим «за один ход» сразу оптимальную стратегию, поскольку выбор $\pi_2(a | s)$ сколь угодно хитро может изменить распределение траекторий, но тем не менее.

Пример 53: Попробуем улучшить стратегию π из примера 46, $\gamma = 0.8$. 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, $Q^{\pi_2}(s = C, ■) \geq 3.04$, и, что важнее, из состояния С мы начали набирать больше награды: $V^{\pi_2}(C) > V^{\pi_1}(C)$ строго. Во-вторых, есть гарантии, 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, выполняются аналогичные неравенства.



3.2.4. Вид оптимальной стратегии (доказательство через RPI)

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 $\pi_1(\cdot | s)$). 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 и не требуя рассуждения про отказ от стационарности и обоснования 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)

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

однако все слагаемые в сумме неположительны. Противоречие. ■

Необходимость. Допустим, что π оптимальна, но для некоторого \hat{s} условие не выполнено, и $\max_a A^\pi(\hat{s}, a) > 0$ (меньше нуля он, ещё раз, быть не может в силу утв. 23). Рассмотрим детерминированную π_2 , которая в состоянии \hat{s} выбирает какое-нибудь \hat{a} , такое что $A^\pi(\hat{s}, \hat{a}) > 0$ (это можно сделать по условию утверждения — сам максимум может вдруг оказаться недостижим для сложных пространств действий, но какое-то действие с положительным advantage-ом 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 вероятностью^{*} 1 верно $s_0 = \hat{s}, a_0 = \hat{a}$, то есть первое слагаемое равно $A(\hat{s}, \hat{a}) > 0$. ■

² в доказательствах RPI и Policy Improvement мы не использовали понятия Q^* и V^* и их свойства; тем не менее, из этих теорем все свойства оптимальных оценочных функций следуют: например, пусть A^*, Q^*, V^* — оценочные функции оптимальных стратегий, тогда в силу выводимого из RPI критерия оптимальности (теорема 18) $\forall s: \max_a A^*(s, a) = 0$, или, что тоже самое, $\max_a [Q^*(s, a) - V^*(s)] = 0$; 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.

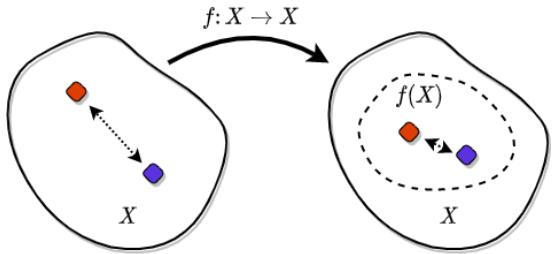
* мы специально стартовали из \hat{s} , чтобы пары \hat{s}, \hat{a} «встретились» в траекториях, иначе могло бы быть такое, что агент в это состояние \hat{s} «никогда не попадает», и отдельиться от нуля не получилось бы.

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

§3.3. Динамическое программирование

3.3.1. Метод простой итерации

Мы увидели, что знание оценочных функций открывает путь к улучшению стратегии. Напрямую по определению считать их затруднительно; попробуем научиться решать уравнения Беллмана. И хотя уравнения оптимальности Беллмана нелинейные, они, тем не менее, имеют весьма определённый вид и, как мы сейчас увидим, обладают очень приятными свойствами. Нам понадобится несколько понятий внезапно из фундамента о том, как решать системы нелинейных уравнений вида $x = f(x)$.



Определение 44: 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 45: Точка $x \in X$ для оператора $f: X \rightarrow X$ называется *неподвижной* (fixed point), если

$$x = f(x)$$

Определение 46: 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 полном* метрическом пространстве X у сжимающего оператора $f: X \rightarrow X$ существует и обязательно единственная 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 $k > 0$:

$$\begin{aligned} \rho(x_k, x_{k+1}) &= \{\text{определение } x_k\} = \rho(f(x_{k-1}), f(x_k)) \leq \\ &\leq \{\text{свойство сжатия}\} \leq \gamma \rho(x_{k-1}, x_k) \leq \dots \leq \\ &\leq \{\text{аналогичным образом}\} \leq \dots \leq \gamma^k \rho(x_0, x_1) \end{aligned} \tag{3.22}$$

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

$$\begin{aligned} \rho(x_k, x_{k+n}) &\leq \\ \{\text{неравенство треугольника}\} &\leq \rho(x_k, x_{k+1}) + \rho(x_{k+1}, x_{k+2}) + \dots + \rho(x_{k+n-1}, x_{k+n}) \leq \\ &\leq \{(3.22)\} \leq (\gamma^k + \gamma^{k+1} + \dots + \gamma^{k+n-1}) \rho(x_0, x_1) \leq \\ &\leq \{\text{геом. прогрессия}\} \leq \frac{\gamma^k}{1-\gamma} \rho(x_0, x_1) \xrightarrow{k \rightarrow \infty} 0 \end{aligned}$$

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

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

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

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

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

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

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

* любая фундаментальная последовательность имеет предел

3.3.2. Policy Evaluation

Вернёмся к RL. Известно, что V^π для данного MDP и фиксированной политики π удовлетворяет уравнению Беллмана (3.3). Для нас это система уравнений относительно значений $V^\pi(s)$. $V^\pi(s)$ — 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 $V: \mathcal{S} \subseteq \mathbb{R}^n$ и выдаёт некоторую другую функцию от состояний \mathfrak{BV} . Чтобы задать выход оператора, нужно задать значение выходной функции в каждом $s \in \mathcal{S}$; это значение мы будем обозначать $[\mathfrak{BV}](s)$ (квадратные скобки позволяют не путать применение оператора с вызовом самой функции) и определим его как правую часть решаемого уравнения (3.3). Итак:

Определение 47: Введём *оператор Беллмана* (Bellman operator) для заданного MDP и стратегии π как

$$[\mathfrak{BV}](s) := \mathbb{E}_{a \sim \pi(a|s)} [r(s, a) + \gamma \mathbb{E}_{s'} V(s')] \quad (3.23)$$

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.

$$\begin{aligned} d_\infty(\mathfrak{BV}_1, \mathfrak{BV}_2) &= \max_s |[\mathfrak{BV}_1](s) - [\mathfrak{BV}_2](s)| = \\ &= \{\text{подставляем значение операторов, т.е. правые части решаемого уравнения}\} = \\ &= \max_s |\mathbb{E}_a [r(s, a) + \gamma \mathbb{E}_{s'} V_1(s')] - \mathbb{E}_a [r(s, a) + \gamma \mathbb{E}_{s'} V_2(s')]| = \\ &= \{\text{слагаемые } r(s, a) \text{ сокращаются}\} = \\ &= \gamma \max_s |\mathbb{E}_a \mathbb{E}_{s'} [V_1(s') - V_2(s')]| \leq \\ &\leq \{\text{используем свойство } \mathbb{E}_x f(x) \leq \max_x f(x)\} \leq \\ &\leq \gamma \max_s \max_{s'} |V_1(s') - V_2(s')| = \gamma d_\infty(V_1, V_2) \end{aligned}$$

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

$$V_{k+1} := \mathfrak{BV}_k$$

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

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

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

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

³ вообще говоря, это система линейных уравнений относительно значений $V^\pi(s)$, 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 $\mathcal{S} \subseteq \mathbb{R}^n$.

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 преосходит $T - 1$.

Осталось увидеть, что на k -ой итерации метода простой итерации вычисляет точные значения $V_k(s) = V^\pi(s)$ 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 $V_{k+1}(s) := [\mathfrak{B}V_k](s)$ 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, $V^\pi(s) = \mathbf{0}$ для всех π, s) мы проинциализировали $V_0(s) = \mathbf{100}$ 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 24: Если некоторая функция $\tilde{V}: \mathcal{S} \rightarrow \mathbb{R}$ удовлетворяет уравнению Беллмана (3.3), 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:

$$1. \forall s: V_{k+1}(s) := \mathbb{E}_a [r(s, a) + \gamma \mathbb{E}_{s'} V_k(s')]$$

$$2. \text{критерий останова: } \max_s |V_k(s) - V_{k+1}(s)| < \varepsilon$$

Выход: $V_k(s)$

Пример 54: 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 $V_k(s) \approx V^\pi(s)$.

Итак, мы научились считать V^π в предположении известной динамики среды. Полностью аналогичное рассуждение верно и для уравнений QQ (3.7); то есть, расширив набор переменных, в табличных MDP можно

методом простой итерации находит \mathbf{Q}^* и «напрямую». Пока модель динамики среды считается известной, это не принципиально: мы можем посчитать и Q-функцию через V-функцию по формуле QV (3.5).

3.3.3. Value Iteration

Теорема Банаха позволяет аналогично Policy Evaluation (алг. 6) решать уравнения оптимальности Беллмана (3.17) через метод простой итерации. Действительно, проведём аналогичные рассуждения (мы сделаем это для \mathbf{Q}^* , но совершенно аналогично можно было бы сделать это и для \mathbf{V}^*):

Определение 48: Определим *оператор оптимальности Беллмана* (Bellman optimality operator, Bellman control operator) \mathfrak{B}^* :

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

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 25:

$$\left| \max_x f(x) - \max_x g(x) \right| \leq \max_x |f(x) - g(x)| \quad (3.24)$$

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

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

Второй случай рассматривается симметрично. ■

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

Доказательство.

$$\begin{aligned} d_\infty(\mathfrak{B}^* Q_1, \mathfrak{B}^* Q_2) &= \max_{s, a} |[\mathfrak{B}^* Q_1](s, a) - [\mathfrak{B}^* Q_2](s, a)| = \\ &= \{\text{подставляем значения операторов, т.е. правые части решаемой системы уравнений}\} = \\ &= \max_{s, a} \left| \left[r(s, a) + \gamma \mathbb{E}_{s'} \max_{a'} Q_1(s', a') \right] - \left[r(s, a) + \gamma \mathbb{E}_{s'} \max_{a'} Q_2(s', a') \right] \right| = \\ &= \{\text{слагаемые } r(s, a) \text{ сокращаются}\} = \\ &= \gamma \max_{s, a} \left| \mathbb{E}_{s'} \left[\max_{a'} Q_1(s', a') - \max_{a'} Q_2(s', a') \right] \right| \leq \\ &\leq \{\text{используем свойство } \mathbb{E}_x f(x) \leq \max_x f(x)\} \leq \\ &\leq \gamma \max_{s, a} \max_{s'} \left| \max_{a'} Q_1(s', a') - \max_{a'} Q_2(s', a') \right| = \\ &\leq \{\text{используем свойство максимумов (3.24)}\} \leq \\ &\leq \gamma \max_{s, a} \max_{s'} \max_{a'} |Q_1(s', a') - Q_2(s', a')| \leq \\ &= \{\text{внутри стоит определение } d_\infty(Q_1, Q_2), \text{ а от внешнего максимума ничего не зависит}\} = \\ &= \gamma d_\infty(Q_1, Q_2) \end{aligned}$$

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 26: Если некоторая функция $\tilde{Q}: \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ удовлетворяет уравнению оптимальности Беллмана (3.17), то $\tilde{Q} \equiv Q^*$.

Утверждение 27: Метод простой итерации сходится к Q^* из любого начального приближения.

Вообще, если известна динамика среды, то нам достаточно решить уравнения оптимальности для \mathbf{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

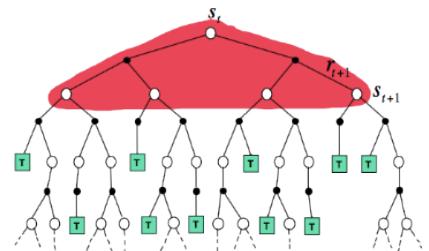
Инициализируем $V_0(s)$ произвольно для всех $s \in \mathcal{S}$

На k -ом шаге:

1. для всех s : $V_{k+1}(s) := \max_a [r(s, a) + \gamma \mathbb{E}_{s'} V_k(s')]$

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 («бэкампим») нашу текущую аппроксимацию V-функции на её **одношаговое приближение** (one-step approximation): смотрим на один шаг в будущее (a, r, s') и приближаем всё остальное будущее текущей же аппроксимацией. Такой «бэкамп 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 мат.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.

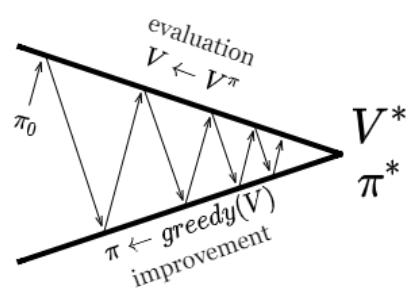
Пример 55: Решим задачу из примера 54, $\gamma = 0.9$; на каждой итерации отображается значение текущего приближения $V_k(s) \approx V^*(s)$. В конце концов в силу детерминированности среды станет понятно, что можно избежать попадания в терминальное -1 и кратчайшим путём добираться до терминального +1.

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. ■

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:

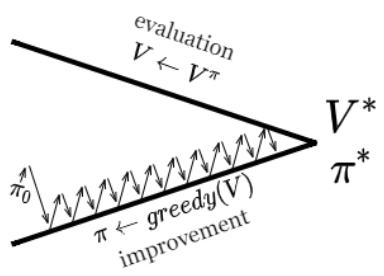
1. $V^{\pi_k} := \text{PolicyEvaluation}(\pi_k, \varepsilon)$
2. $Q^{\pi_k}(s, a) := r(s, a) + \gamma \mathbb{E}_{s'} V^{\pi_k}(s')$
3. $\pi_{k+1}(s) := \underset{a}{\operatorname{argmax}} Q^{\pi_k}(s, a)$
4. критерий останова: $\pi_k \equiv \pi_{k+1}$

Пример 56: Решим задачу из примера 54, $\gamma = 0.9$; 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

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

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



⁴считаем, 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 9: Generalized Policy Iteration

Гиперпараметры: N — количество шагов

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

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

На k -ом шаге:

1. Повторить N раз:

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

$$2. Q(s, a) \leftarrow r(s, a) + \gamma \mathbb{E}_{s'} V(s')$$

$$3. \pi(s) \leftarrow \underset{a}{\operatorname{argmax}} Q(s, a)$$

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

Утверждение 28: 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:

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

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

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

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

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

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 $Q_k(s, a) \approx \approx Q^*(s, a)$ можно также рассматривать как приближение $Q_k(s, a) \approx Q^\pi(s, a)$ для $\pi(s) := \underset{a}{\operatorname{argmax}} Q_k(s, a)$; 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 шага 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: Для любого состояния s_0 :

$$V^\pi(s_0) = \mathbb{E}_{T_{:N} \sim \pi | s_0} \left[\sum_{t=0}^{N-1} \gamma^t r_t + \gamma^N \mathbb{E}_{s_N} V^\pi(s_N) \right] \quad (3.25)$$

Доказательство по индукции. Для получения уравнения на N шагов берём $N-1$ -шаговое и подставляем в правую часть раскрутку на один шаг из уравнения (3.3). 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 29: \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 одноступенчатого уравнения; в этом смысле, мы ничего не выигрываем. В частности, если мы устремим N к бесконечности, то мы получим просто определение V-функции; формально, в правой части будет стоять выражение, вообще не зависящее от поданной на вход оператору $V(s)$, коэффициент сжатия будет ноль, и метод простой итерации как бы сходится тут же за один шаг. Но для проведения этого шага нужно 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.

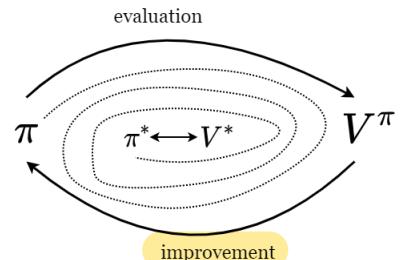
Теорема 26: Алгоритм Generalized Policy Iteration 9 при любом N сходится к оптимальной стратегии и оптимальной оценочной функции. ■

Без доказательства.

Интуитивно, такой алгоритм «стабилизируется», если оценочная функция будет удовлетворять уравнению Беллмана для текущей π (иначе оператор \mathfrak{B}^N изменит значение функции), и если π выбирает $\arg\max_a Q(s, a)$ из неё; а если аппроксимация V-функции удовлетворяет уравнению Беллмана, то она совпадает с V^π , и значит $Q(s, a) = Q^\pi(s, a)$. То есть, при сходимости стратегия π будет выбирать действие жадно по отношению к своей же Q^π , а мы помним, что это в точности критерий оптимальности.

Все алгоритмы, которые мы будем обсуждать далее, так или иначе подпадают под обобщённую парадигму «оценивание-улучшение». У нас будет два процесса оптимизации: обучение **актёра** (actor), политики π , и **критика** (critic), оценочной функции (Q или V). Критик обучается оценивать текущую стратегию, текущего актёра: сдвигаться в сторону решения какого-нибудь уравнения, для которого единственной неподвижной точкой является V^π или Q^π . Актёр же будет учиться при помощи policy improvement-а: вовсе не обязательно делать это жадно, возможно учиться выбирать те действия, где оценка критика «побольше», оптимизируя в каких-то состояниях (в каких - пока открытый вопрос) функционал (3.21):

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



Причём, возможно, в этом функционале нам не понадобится аппроксимация (модель) Q -функции в явном виде, и тогда мы можем обойтись лишь какими-то оценками Q^π ; в таких ситуациях нам достаточно будет на этапе оценивания политики обучать лишь модель V^π для текущей стратегии. А, например, в эволюционных методах мы обошлись вообще без обучения критика именно потому, что смогли обойтись лишь Монте-Карло оценками будущих наград. Этот самый простой способ решать задачу RL — погенерировать несколько случайных стратегий и выбрать среди них лучшую — тоже условно подпадает под эту парадигму: мы считаем Монте-Карло оценки значения $J(\pi)$ для нескольких разных стратегий (evaluation) и выбираем наилучшую

стратегию (improvement). Поэтому Policy Improvement, как мы увидим, тоже может выступать в разных формах: например, возможно, как в Value Iteration, у нас будет приближение Q-функции, и мы будем просто всегда полагать, что policy improvement проводится жадно, и текущей стратегией неявно будет $\arg\max_a Q(s, a)$.

Но главное, что эти два процесса, оценивание политики (обучение критика) и улучшение (обучение актёра) можно будет проводить стохастической оптимизацией. Достаточно, чтобы лишь в *среднем* модель оценочной функции сдвигалась в сторону V^π или Q^π , а актёр лишь в *среднем* двигался в сторону $\arg\max_a Q(s, a)$. И такой «рецепт» алгоритма всегда будет работать: пока оба этих процесса проводятся корректно, итоговый алгоритм запустится на практике. Это в целом фундаментальная идея всего RL. В зависимости от выбора того, как конкретно проводить эти процессы, получатся разные по свойствам алгоритмы, и, в частности, отдельно интересными будут алгоритмы, «схлопывающие» схему Generalized Policy Iteration в её предельную форму, в Value Iteration.

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

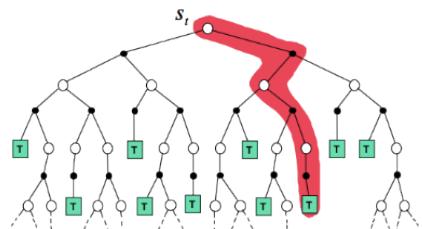
3.4.1. Монте-Карло алгоритм

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 какую-то стратегию, и дальше должны проводить итерации алгоритма, используя лишь эти сэмплы траекторий. Иначе говоря, для данного s мы можем выбрать a и получить ровно один сэмпл из очередного $p(s' | s, a)$, причём на следующем шаге нам придётся проводить сбор сэмплов именно из s' . Как в таких условиях «решать» уравнения Беллмана — неясно.

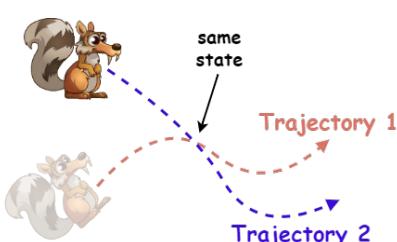
Рассмотрим самый простой способ превратить Policy Iteration в model-free метод. Давайте очередную стратегию π_k отправим в среду, сыграем несколько эпизодов, и будем оценивать Q^{π_k} по Монте-Карло:

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

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*»).



Монте-Карло алгоритм, на первый взгляд, плох примерно всем. Нужно доигрывать игры до конца, то есть алгоритм неприменим в неэпизодичных средах; Монте-Карло оценки также обладают огромной дисперсией, поскольку мы заменили на сэмплы все мат.ожидания, стоящие в мат.ожидании по траекториям; наконец, мы практически перестали использовать структуру задачи. Если мы получили сэмпл +100 в качестве очередного сэмпла для $Q^\pi(s, a)$, то мы «забыли», какую часть из этой +100 мы получили сразу же после действия a , а какая была получена в далёком будущем — не использовали разложение награды за эпизод в сумму наград за шаг. Также мы посеяли «информацию о соединениях состояний»: 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-a? 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 Q-функцию для новой стратегии π_{k+1} как-то при помощи текущего приближения $Q(s, a) \approx Q^{\pi_k}(s, a)$? Да, хоть оно и считалось для «предыдущей» стратегии и формально содержит сэмплы не из того распределения, но всё-таки этих сэмплов там было аккумулировано много, да и стратегия потенциально поменялась не сильно; и всяко лучше какой-нибудь нулевой инициализации. 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. Всё это хочется делать как-то 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 49: 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, называемый learning rate.

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

$$\sum_{k \geq 0}^{\infty} \alpha_k = +\infty, \quad \sum_{k \geq 0}^{\infty} \alpha_k^2 < +\infty \tag{3.26}$$

Теорема 27: 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 (3.26). Тогда экспоненциальное сглаживание

$$m_k := (1 - \alpha_k)m_{k-1} + \alpha_k x_k \tag{3.27}$$

сходится к m с вероятностью 1.

Доказательство. Без ограничения общности будем доказывать утверждение для $m = 0$, поскольку для сведения к этому случаю достаточно вычесть m из правой и левой части (3.27) и перейти к обозначениям $\hat{m}_k := m_k - m$, $\hat{x}_k := x_k - m$.

Итак, пусть $\mathbb{E}x_k = 0$. Будем доказывать, что $v_k := \mathbb{E}m_k^2 \xrightarrow{k \rightarrow \infty} 0$. Для начала возведём обе стороны уравнения (3.27) 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 мат.ожидание:

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

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:

$$v_k = (1 - \alpha_k)^2 v_{k-1} + \alpha_k^2 \mathbb{E}x_k^2 \quad (3.28)$$

Сейчас мы уже можем доказать, что $v_k \leq C$. Действительно, сделаем это по индукции. База: $v_0 = 0 \leq C$ 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 (3.28):

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

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

$$v_n - v_0 = -2 \sum_{k=0}^{n-1} \alpha_k v_k + \sum_{k=0}^{n-1} \alpha_k^2 (v_k + \mathbb{E}x_k^2) \quad (3.29)$$

Заметим, что $v_0 = 0$, а $v_n \geq 0$ 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 (3.29). 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 мат.ожидание 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 51: Стохастическая аппроксимация (Stochastic approximation) — задача решения уравнения вида

$$x = \mathbb{E}_{\varepsilon \sim p(\varepsilon)} f(x, \varepsilon), \quad (3.30)$$

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

Можно ли объединить идеи метода простой итерации и экспоненциального сглаживания («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 несмешённой оценкой правой части; вместо этого сгладим старое значение x_k и полученный новый «сэмпл»:

$$x_k = (1 - \alpha_k)x_{k-1} + \alpha_k f(x_{k-1}, \varepsilon), \quad \varepsilon \sim p(\varepsilon) \quad (3.31)$$

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 learning rate α_k удовлетворяют условиям (3.26), то такая процедура будет в пределе сходиться.

Поймём, почему задача стохастической аппроксимации тесно связана со стохастической оптимизацией. Для этого перепишем формулу (3.31) в альтернативной очень интересной форме:

$$x_k = x_{k-1} + \alpha_k (f(x_{k-1}, \varepsilon) - x_{k-1}), \quad \varepsilon \sim p(\varepsilon) \quad (3.32)$$

Да это же формула *стохастического градиентного спуска* (stochastic gradient descent, SGD)! Действительно, в нём мы, оптимизируя некоторую функцию $f(x)$, прибавляем к очередному приближению x_{k-1} с некоторым learning rate α_k несмешённую оценку градиента $\nabla f(x_{k-1})$, которую можно обозначить как $\nabla f(x_{k-1}, \varepsilon)$:

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

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

И, глядя на формулу (3.32), мы понимаем, что, видимо, выражение $f(x_{k-1}, \varepsilon) - x_{k-1}$ есть какой-то «стохастический градиент». Стохастический он потому, что это выражение случайно: мы сэмплируем $\varepsilon \sim p(\varepsilon)$; при этом это несмешённая оценка «градиента», поскольку она обладает следующим свойством: в точке решения, то есть в точке x^* , являющейся решением уравнения (3.30), в среднем его значение равно нулю:

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

И по аналогии можно предположить, что если стохастический градиентный спуск ищет решение $\mathbb{E}_\varepsilon \nabla f(x, \varepsilon) = 0$, то процесс (3.32) ищет решение уравнения $\mathbb{E}_\varepsilon f(x, \varepsilon) - x = 0$.

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 (3.30)! Справа стоит мат.ожидание по недоступному нам распределению, содержимое которого мы, тем не менее, при данном сэмпле $s' \sim p(s' | s, a)$ и текущем приближении Q-функции, умеем считать. Возникает идея проводить стохастическую аппроксимацию: заменять правые части решаемых уравнений на несмешённые оценки и сглаживать текущее приближение с получаемыми сэмплами.

Отметим интересный факт: уравнение V^*V^* (3.18), имеющее вид «максимум от мат.ожидания по неизвестному распределению»

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

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 с некоторым learning rate наше текущее приближение $Q(s, a)$ в сторону сэмпла

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

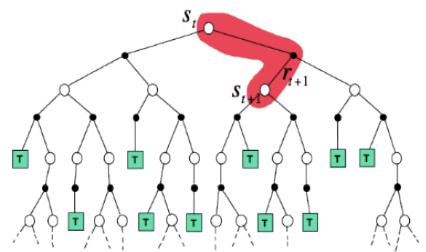
который также будем называть **таргетом**⁵ (Bellman target). Получаем следующую формулу обновления:

$$Q_{k+1}(s, a) \leftarrow Q_k(s, a) + \alpha_k \left(\underbrace{r + \gamma Q_k(s', a')}_{\text{таргет}} - \underbrace{Q_k(s, a)}_{\text{текущее приближение}} \right) \quad (3.33)$$

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

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 (3.33) 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 **бутстрэпирования** (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 57: Вы сидите в кафе (s) и хотите вернуться домой. Вы прикидываете, что в среднем этой займет $-Q(s, a) = 30$ 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: $-Q(s', a') = 40$ 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 $-Q(s, a)$ 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 (3.33)? Пока мы говорили, что мы хотим заниматься оцениванием стратегии π , и поэтому предлагается, видимо, ею и взаимодействовать со средой, собирая переходы. С точки же зрения стохастической аппроксимации, для корректности обновлений достаточно выполнения всего лишь двух требований:

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

Оба этих утверждения вытекают из того, что обновление (3.33) неявно ищет решение уравнения

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

⁵в 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 бутстрэпирования, поскольку мы начинаем «из воздуха» делать себе сэмплы из уже имеющихся сэмплов.

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 ячейки $Q(s, a)$ мы обязательно на следующем шаге обновили ячейку $Q(s', a')$, 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.

3.4.5. Q-learning

Поскольку довести $Q(s, a)$ до точного значения $Q^\pi(s, a)$ с гарантиями мы всё равно не сможем, однажды в алгоритме нам всё равно придётся сделать policy improvement. Что, если мы будем обновлять нашу стратегию $\pi_k(s) := \underset{a}{\operatorname{argmax}} Q_k(s, a)$ 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:

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

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: для перехода s, a, r, s' обновляется только одна ячейка нашего приближения Q-функции:

$$Q_{k+1}(s, a) := Q_k(s, a) + \alpha_k \left(r + \gamma \max_{a'} Q_k(s', a') - Q_k(s, a) \right) \quad (3.34)$$

Теорема 28 — 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:

$$\sum_{k \geq 0}^{\infty} \alpha_k(s, a) = +\infty \quad \sum_{k \geq 0}^{\infty} \alpha_k(s, a)^2 < +\infty \quad (3.35)$$

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 $Q \approx Q^*$ только для одной пары s, a , то можно считать, что $\alpha_k(s, a) \neq 0$ только для неё. При этом ограничения (3.35) 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' :

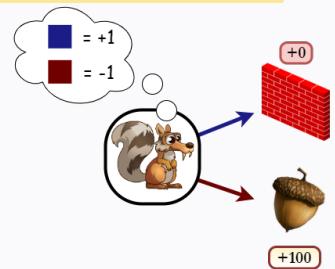
Утверждение 30: Пусть сбор опыта проводится так, что пары s, a встречаются бесконечное число раз. Пусть $n(s, a)$ — счётчик количества выполнений действия a в состоянии s во время взаимодействия агента со средой. Тогда можно положить $\alpha_k(s, a) := \frac{1}{n(s, a)}$, чтобы гарантировать сходимость алгоритма к Q^* .

3.4.6. 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 29: 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 $\pi_0(s) := \underset{a}{\operatorname{argmax}} Q_0(s, a)$, 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 21 о существовании в нём оптимальной стратегии: мы показали, что для сбора данных — взаимодействия со средой — детерминированные стратегии не подходят. Какие подходят?

Теорема 30: Любая стратегия, для которой $\forall s, a: \mu(a | s) > 0$, 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 в 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 28. 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 переходы s, a, r, s' , обучать на них Q-функцию по формуле (3.34), и на выходе в пределе мы получим Q^* ? В пределе — да.

Пример 58: 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 неэффективный 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 $Q(s, a)$. 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.

а

Определение 52: ε -жадной (ε -greedy) называется стратегия

$$\mu(s) := \begin{cases} a \sim \text{Uniform}(\mathcal{A}) & \text{с вероятностью } \varepsilon; \\ \underset{a}{\operatorname{argmax}} Q(s, a) & \text{иначе.} \end{cases} \quad (3.36)$$

Определение 53: *Больцмановской* с температурой τ называется стратегия

$$\mu(a | s) := \text{softmax}_a \left(\frac{Q(s, a)}{\tau} \right) \quad (3.37)$$

Первый вариант никак не учитывает, насколько кажутся хорошими другие действия помимо жадного, и, если решает «исследовать», выбирает среди них случайно. Второй же вариант будет редко выбирать очень плохие действия (это может быть крайне полезным свойством), но чувствителен к масштабу приближения Q-функции, что обычно менее удобно и требует настройки температуры τ . Для Больцмановской стратегии мы увидим интересную интерпретацию в контексте обсуждения Maximum Entropy RL (раздел 6.2).

3.4.7. Реплей буфер

Итак, мы теперь можем собрать 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

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



Естественно, в эпизодичных средах нужно всегда следить за флагом **done** и учитывать, что для терминальных состояний оценочные функции равны нулю. Мнемонически можно запомнить, что этот флаг является частью дисконтирования: домножение на **1 – done** происходит всюду, где мы домножаем приближение будущей награды на γ .

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. Это крайне существенное свойство, потому что в таких алгоритмах 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 (3.34). Что мы тогда выучим?

Определение 54: 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 31: При выполнении условий на learning rate, Q-learning, запущенный с фиксированного 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 в силу теоремы 28. ■

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 55: *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

Гиперпараметры: 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. кладём s_k, a_k, r_k, s_{k+1} в буфер
4. сэмплируем случайный переход s, a, r, s' из буфера
5. обновляем $Q(s, a) \leftarrow Q(s, a) + \alpha \left(r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right)$

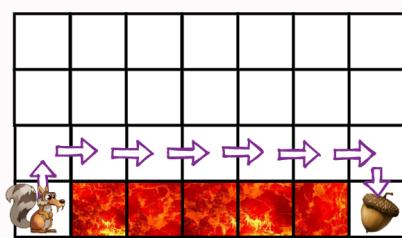
Реплей буфер — ключевое преимущество off-policy алгоритмов: мы сможем с каждого перехода потенциально обучаться бесконечное количество раз. Это развязывает нам руки в плане соотношения числа собираемых данных и количества итераций обновления нашей модели, поскольку здесь мы можем, в общем-то, сами решать, сколько переходов на один шаг обновления мы будем проводить. Проявляется это в том, что в Q-learning, как и в любом другом off-policy алгоритме, есть два независимых этапа: сбор данных (взаимодействие со средой при помощи ε -жадной стратегии и сохранение опыта в памяти — первые три шага), и непосредственно обучение (сэмплирование перехода из буфера и обновление ячейки — шаги 4-5). Можно проводить несколько шагов сбора данных на одно обновление, или наоборот: несколько обновлений на один шаг сбора данных, или даже проводить эти процессы независимо параллельно.

3.4.8. SARSA

Мы придумали off-policy алгоритм: мы умеем оценивать нашу текущую стратегию ($\underset{a}{\operatorname{argmax}} Q(s, a)$, 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 эксплорейшном. Это различие было для нас принципиально: оптимальны детерминированные стратегии, а взаимодействовать со средой мы готовы лишь стохастическими стратегиями. У этого «несовпадения» есть следующий эффект.

Пример 59 — 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. За попадание в целевое состояние агент получает $+1$, и эпизод также завершается; соответственно, задача агента — как можно быстрее добраться до цели, не угодив в лаву.

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

$$Q(s, a) \leftarrow Q(s, a) + \alpha (r(s, a) + \gamma Q(s', a') - Q(s, a)) \quad (3.38)$$

Какую Q-функцию такой алгоритм будет учить? Поскольку $a' \sim \mu$, где μ — стратегия взаимодействия со средой, то мы стохастически аппроксимируем Q^μ для этой самой μ , сдвигаясь в сторону решения обычного уравнения Беллмана. Формула обновления не будет эквивалентна формуле Q-learning-a (3.34), где мы сдвигались в сторону Q^π , где π было жадной стратегией по отношению к этой же Q-функции: да, в большинстве случаев (с вероятностью $1 - \epsilon$) a' будет аргмаксимумом, и обновление будет совпадать с Q-learning, но иногда a' будет оказываться тем самым «случайным» действием, случившимся из-за исследований, и наша Q-функция будет сдвигаться в сторону ценности случайных действий. Так в оценочную функцию будет попадать знание о том, что в будущем мы на каждом шаге с вероятностью ϵ будем обязаны выбрать случайное действие, и подобное «дёрганье» будет мешать нам проходить по краю вдоль обрыва с лавой.

Алгоритм 12: SARSA

Гиперпараметры: 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 , сэмплируем $a_0 \sim \text{Uniform}(\mathcal{A})$

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. Можно считать, что он чередует два шага: обновление (3.38), которое учит Q^π для текущей π , и, неявно, некий аналог policy improvement-а: замены π на ϵ -greedy(Q). Именно при помощи обновлённой стратегии мы будем взаимодействовать со средой на следующем шаге, то есть полагаем $\mu \equiv \pi$ («переходим в on-policy режим»). 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 56: Будем говорить, что стратегия π — ϵ -мягкая (ϵ -soft), если $\forall s, a: \pi(a | s) \geq \frac{\epsilon}{|\mathcal{A}|}$.

Утверждение 32: Если π_1 — ϵ -мягкая, то $\pi_2 := \epsilon$ -greedy(Q^{π_1}) не хуже, чем π_1 .

Доказательство. Проверим выполнение теоремы 17; выглядит немного страшновато, но суть этих выкладок довольно лобовая: если мы переложим всю вероятностную массу в самое «хорошее» с точки зрения $Q^{\pi_1}(s, a)$, оставив у остальных, не самых лучших, действий ровно $\frac{\epsilon}{|\mathcal{A}|}$, то среднее значение увеличится.

$$\begin{aligned} V^{\pi_1}(s) &= \{\text{уравнение VQ (3.6)}\} = \sum_a \pi_1(a | s) Q^{\pi_1}(s, a) = \\ &= \sum_a \left(\pi_1(a | s) - \frac{\epsilon}{|\mathcal{A}|} \right) Q^{\pi_1}(s, a) + \frac{\epsilon}{|\mathcal{A}|} \sum_a Q^{\pi_1}(s, a) = \\ &= (1 - \epsilon) \sum_a \frac{\pi_1(a | s) - \frac{\epsilon}{|\mathcal{A}|}}{1 - \epsilon} Q^{\pi_1}(s, a) + \frac{\epsilon}{|\mathcal{A}|} \sum_a Q^{\pi_1}(s, a) \leq \\ &\leq (1 - \epsilon) \max_a Q^{\pi_1}(s, a) \sum_a \frac{\pi_1(a | s) - \frac{\epsilon}{|\mathcal{A}|}}{1 - \epsilon} + \frac{\epsilon}{|\mathcal{A}|} \sum_a Q^{\pi_1}(s, a) \end{aligned}$$

Заметим, что последний переход был возможен только потому, что $\pi_1(a | s) - \frac{\epsilon}{|\mathcal{A}|} > 0$ по условию, так как

π_1 — ε -мягкая по условию. Осталось заметить, что

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

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

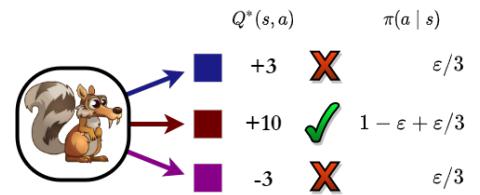
Давайте изменим постановку задачи: скажем, что мы запрещаем к рассмотрению стратегии, «похожие на детерминированные». Наложим ограничение в нашу задачу оптимизации: скажем, что стратегия обязательно должна быть ε -мягкой. В такой задаче будут свои оптимальные оценочные функции.

Определение 57: 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 привести ни в одном состоянии, а то есть $\pi \equiv \varepsilon\text{-greedy}(Q^\pi)$, то, видимо, придумать стратегию лучше в принципе невозможно:



Утверждение 33: 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:

$$\begin{cases} \mathbb{E}_{\pi(a|s)} Q_{\varepsilon\text{-soft}}^*(s, a) \rightarrow \max \\ \int_A \pi(a | s) da = 1; \quad \forall a \in \mathcal{A}: \pi(a | s) \geq \frac{\varepsilon}{|\mathcal{A}|} \end{cases}$$

Формально решая эту задачу условной оптимизации, получаем доказываемое. ■

Утверждение 34: 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 (3.7) и подставив вид оптимальной стратегии. ■

Мы теперь понимаем, что наша формула обновления (3.38) — 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:

$$Q(s, a) \leftarrow Q(s, a) + \alpha (r(s, a) + \gamma \mathbb{E}_{a' \sim \pi} Q(s', a') - Q(s, a)), \quad (3.39)$$

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 (3.7) 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

Гиперпараметры: α — параметр экспоненциального сглаживания, ε — параметр исследований

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

Инициализируем π_0 произвольно

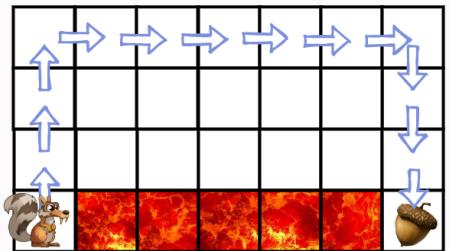
Наблюдаем s_0

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 $a_{k+1} := \underset{a_{k+1}}{\operatorname{argmax}} Q(s_{k+1}, a_{k+1})$
4. 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 $Q_{\varepsilon\text{-soft}}^*(s, a)$ — оптимальной оценочной функции в семействе ε -мягких стратегий.

Пример 60 — Cliff World: Попробуем запустить в MDP из примера 59 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, к чему это приводит, рассмотрим, что случится, если взять буфер некоторого эксперта π_{expert} , генерировать пятёрки s, a, r, s', a' из него и проводить обновления (3.38) по ним. Что мы выучим? Применим наш стандартный ход рассуждений: $a' \sim \pi_{\text{expert}}(a' | s')$, и, значит, мы учим $Q^{\pi_{\text{expert}}}$! Если же мы попробуем запустить SARSA с experience replay, то есть обучаться на собственной же истории, то мы вообще творим полную ахинею: 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.

§3.5. Bias-Variance Trade-Off

3.5.1. Дилемма смещения-разброса

Мы обсудили два вида бэкапов, доступных в model-free обучении: Монте-Карло бэкап и Temporal-Difference бэкап. На самом деле, они очень похожи, поскольку делают обновление вида

$$Q(s, a) \leftarrow Q(s, a) + \alpha (y_Q - Q(s, a)),$$

и отличаются лишь выбором y_Q : Монте-Карло берёт reward-to-go, а TD-backup — одношаговую будстратированную оценку с использованием уже имеющейся аппроксимации Q-функций:

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

Какой из этих двух вариантов лучше? Мы уже обсуждали недостатки Монте-Карло оценок: высокая дисперсия, необходимость играть до конца эпизодов, игнорирование структуры получаемой награды и потеря информации о соединениях состояний. Но не то, чтобы одношаговые оценки сильно лучше: на самом деле, они обладают полностью противоположными свойствами и проблемами.

Да, одношаговые оценки аппроксимируют решение одношаговых уравнений Беллмана и приближают алгоритм динамического программирования: поэтому они не теряют информации о том, на каком шаге какой сигнал от среды был получен, и сохраняют информацию о сэмплах s' из функции переходов; в том числе, как

мы видели, одношаговые алгоритмы могут использовать реплей буфер, по сути и хранящий собранную выборку таких сэмплов. Взамен в одношаговых алгоритмах возникает **проблема распространения сигнала**.

Пример 61: Представьте, что за 100 шагов вы можете добраться до сыра (+1). Пока вы не добьётесь успеха, сигнала нет, и ваша аппроксимация Q-функции остаётся всюду нулём. Допустим, вы учитесь с одношаговых оценок с онлайн опыта. После первого успеха +1 распространится лишь в пару s, a , непосредственно предшествующей получению сыра; после второго успеха +1 распространится из пары s, a в предыдущую и так далее. Итого, чтобы распространить сигнал на 100 шагов, понадобится сделать 100 обновлений. С другой стороны, если бы использовалась Монте-Карло оценка, после первого же успеха +1 распространялся бы во все пары s, a из успешной траектории.

Вместо высокой дисперсии Монте-Карло оценок в одношаговых оценках нас ждёт большое **смещение** (bias): если y_Q оценено через нашу же текущую аппроксимацию через бутстрапирование, то оно не является несмешённой оценкой искомой $Q^\pi(s, a)$ и может быть сколь угодно «неправильным». Как мы увидели, гарантии сходимости остаются, но естественно, что методы стохастической аппроксимации из-за смещения будут сходить сильно дальше экспоненциального слаживания, которому на вход поступают несмешённые оценки искомой величины. Но дисперсия y_Q в temporal difference обновлениях, например, в алгоритме Q-learning 10, конечно, сильно меньше дисперсии Монте-Карло оценок: внутри нашей аппроксимации Q-функции уже усреднены все будущие награды, то есть «взяты» все интегралы, относящиеся к будущим после первого шага наградам. Итого одношаговая оценка y_Q — случайная величина только от s' , а не от всего хвоста траектории $s', a', s'' \dots$. Выбор между оценками с высокой дисперсией и отсутствием смещения (Монте-Карло) и оценками с низкой дисперсией и большим смещением (одношаговые оценки) — особая задача в обучении с подкреплением, называемая **bias-variance trade-off**.

3.5.2. N-step Temporal Difference

Какие есть промежуточные варианты между одношаговыми оценками и Монте-Карло оценками? Давайте заглядывать в будущее не на один шаг и не до самого конца, а на N шагов. Итак, пусть у нас есть целый фрагмент траектории:

Определение 58: Фрагмент траектории $s, a, r, s', a', r', s'', a'', r'', \dots, s^{(N)}, a^{(N)}$, где $s^{(N)}, a^{(N)}$ — состояние и действие, которое агент встретил через N шагов, будем называть **роллаутом** (rollout) длины N .

Определение 59: N -шаговой оценкой (N-step estimation) для $Q^\pi(s, a)$ назовём следующий таргет:

$$y_Q := r + \gamma r' + \gamma^2 r'' + \dots + \gamma^{N-1} r^{(N-1)} + \gamma^N Q(s^{(N)}, a^{(N)}),$$

Такой таргет является стохастической аппроксимацией правой части N -шагового уравнения Беллмана (3.25), и формула (3.40) с таким таргетом позволяет эту систему уравнений решать в model-free режиме. По каким переменным мы заменили интегралы на Монте-Карло приближения в такой оценке? По переменным $s', a', s'', a'' \dots s^{(N)}, a^{(N)}$, которые, пусть и не все присутствуют явно в формуле, но неявно задают то уравнение, которое мы решаем. Соответственно, чтобы выучить $Q^\pi(s, a)$, нужно, чтобы состояния приходили из функции переходов, а действия — из оцениваемой стратегии π . Другими словами, роллаут, использованный для построения таргета, должен быть порождён оцениваемой стратегией.

Почему гиперпараметр N отвечает за bias-variance trade-off? Понятно, что при $N \rightarrow \infty$ оценка переходит в Монте-Карло оценку. С увеличением N всё больше интегралов заменяется на Монте-Карло оценки, и растёт дисперсия; наше же смешённое приближение будущих наград $Q^\pi(s^{(N)}, a^{(N)})$, которое может быть сколь угодно неверным, домножается на γ^N , и во столько же раз сбивается потенциальное смещение; с ростом N оно уходит в ноль. Замешивая в оценку слагаемое $r + \gamma r' + \gamma^2 r'' + \dots + \gamma^{N-1} r^{(N-1)}$ мы теряем информацию о том, что из этого в какой момент было получено, но и начинаем распространять сигнал в N раз «быстрее» одношаговой оценки.

Сразу заметим, что при $N > 1$ нам необходимо иметь в N -шаговой оценке сэмплы $a' \sim \pi(a | s), s'' \sim p(s'' | s', a')$. Это означает, что мы не можем получить такую оценку с буфера: действительно, в буфере для данной четвёрки s, a, r, s' не лежит сэмпл $a' \sim \pi(a | s)$, ведь в произвольном буфере a' генерируется стратегией сбора данных (старой версией стратегии или «экспертом»). Само по себе это, вообще говоря, не беда: мы могли бы взять из буфера четвёрку, прогнать стратегию π , которую хотим оценить, на s' и сгенерировать себе сэмпл a' ; но для него мы не сможем получить сэмпл s'' из функции переходов! Поэтому обучаться на многошаговые оценки с буфера не выйдет; по крайней мере, без какой-либо коррекции.

Также отметим, что все рассуждения одинаковы применимы как для обучения Q^π , так и V^π . Для V-функции общая формула обновления выглядит так:

$$V(s) \leftarrow V(s) + \alpha (y_V - V(s)), \quad (3.40)$$

где y_V — reward-to-go при использовании Монте-Карло оценки, и $y_V := r(s, a) + \gamma V(s')$ для одношагового метода временных разностей, где $a \sim \pi(a | s)$ (сэмплирован из текущей оцениваемой стратегии), $s' \sim p(s' |$

$| s, a \rangle$. Соответственно, для V-функции обучаться с буфера (без каких-либо коррекций) невозможно даже при $N = 1$, поскольку лежащий в буфере a сэмплирован из стратегии сбора данных, а не оцениваемой π .

Для простоты и наглядности будем обсуждать обучение V^π . Также введём следующие обозначения:

Определение 60: Введём такое обозначение *N-шаговой временной разности* (N-step temporal difference) для пары s, a :

$$\Psi_{(N)}(s, a) := \sum_{t=0}^{N-1} \gamma^t r^{(t)} + \gamma^N V(s^{(N)}) - V(s) \quad (3.41)$$

где V — текущая аппроксимация V-функции, $s, a, \dots, s^{(N)}$ — роллаут, порождённый π .

Ранее мы обсуждали temporal difference, в котором мы сдвигали нашу аппроксимацию на $\Psi_{(1)}(s, a)$:

$$\begin{aligned} V(s) &\leftarrow V(s) + \alpha(r + \gamma V(s') - V(s)) = \\ &= V(s) + \alpha \Psi_{(1)}(s, a) \end{aligned}$$

Теперь же мы можем обобщить наш метод, заменив оценку V-функции на многошаговую оценку:

$$V(s) \leftarrow V(s) + \alpha \Psi_{(N)}(s, a)$$

Но какое N выбрать?

3.5.3. Интерпретация через Credit Assingment

Вопрос, обучаться ли со смешённых оценок, или с тех, которые имеют большую дисперсию, имеет прямое отношение к одной из центральных проблем RL — credit assingment. На самом деле, это ровно та же самая проблема.

Рассмотрим проблему credit assingment-a: за какие будущие награды «в ответе» то действие, которое было выполнено в некотором состоянии s ? Как мы обсуждали в разделе 3.2.1, «идеальное» решение задачи — значение $A^\pi(s, a)$, но на практике у нас нет точных значений оценочных функций, а есть лишь аппроксимация, допустим, $V \approx V^\pi$. Положим, мы знаем сэмпл траектории $a, s', r, s'', a'', \dots$ до конца эпизода.

С одной стороны, мы можем выдавать кредит, полностью опираясь на аппроксимацию:

$$\begin{aligned} Q^\pi(s, a) &= r(s, a) + \gamma \mathbb{E}_{s'} V^\pi(s') \approx r(s, a) + \gamma V(s') \\ A^\pi(s, a) &= Q^\pi(s, a) - V^\pi(s) \approx r(s, a) + \gamma V(s') - V(s) \end{aligned} \quad (3.42)$$

Проблема в том, что наша аппроксимация может быть сколь угодно неверна, и выдавать полную ерунду. Более «безопасный» с этой точки зрения способ — в приближении Q-функции не опираться на аппроксимацию и использовать reward-to-go:

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s) \approx \sum_{t \geq 0} \gamma^t r^{(t)} - V(s) \quad (3.43)$$

У этого второго способа выдавать кредит есть важное свойство, которого нет у первого: в среднем такой кредит будет больше у тех действий, которые действительно приводят к более высокой награде. То есть, если a_1, a_2 таковы, что $Q^\pi(s, a_1) > Q^\pi(s, a_2)$, то при любой аппроксимации $V(s)$ среднее значение кредита для s, a_1 будет больше s, a_2 . Это и есть свойство несмешённости Монте-Карло оценок в контексте проблемы выдачи кредита.

Беда Монте-Карло в том, что в этот кредит закладывается награда не только за выбор a в s , но и за будущие решения. То есть: представьте, что через десять шагов агент выбрал действие, которое привело к +100. Эта награда +100 попадёт и в кредит всех предыдущих действий, хотя те не имели к нему отношения.

Пример 62: Допустим, мы ведём машину, и в состоянии s , где аппроксимация V-функции прогнозирует будущую награду ноль, решили выехать на встречку. Среда не сообщает нам никакого сигнала (пока не произошло никакой аварии), но аппроксимация V-функции резко упала; если мы проводим credit assingment одношаговым способом (3.42), то мы получаем сильно отрицательный кредит, который сообщает, что это действие было явно плохим.

Дальше агент следующими действиями исправил ситуацию, вернулся на правильную полосу и после решил поехать в магазин тортиков, где оказался юбилейным покупателем и получил бесплатный тортик +10. Если бы мы проводили credit assingment методом Монте-Карло (3.43), кредит получился бы сильно положительным: $+10 - 0 = +10$: мы положили, что выезд на встречку привёл к тортику.

Видно, что эти два крайних способа выдачи кредита есть в точности «градиент» $y_V - V(s)$, по которому учится V-функция в формуле (3.40). Фактически, занимаясь обучением V-функции и используя формулу

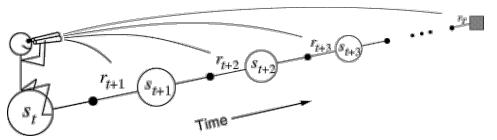
$$V(s) \leftarrow V(s) + \alpha \Psi(s, a),$$

мы выбором функции $\Psi(s, a)$ по-разному проводим credit assigment.

Таким образом видна новая интерпретации bias-variance trade-off-а: и «дисперсия» с этой точки зрения имеет смысл возложения на действие ответственности за те будущие награды, к которым это действие отношения на самом деле не имеет. Одношаговая оценка $\Psi_{(1)}$ говорит: действие влияет только на награду, которую агент получит тут же, и весь остальной сигнал будет учтён в аппроксимации V-функции. Монте-Карло оценка $\Psi_{(\infty)}$ говорит, что действие влияет на все будущие награды. А N -шаговая оценка $\Psi_{(N)}$ говорит странную вещь: действие влияет на события, который происходят с агентом в течение следующих N шагов.

Как и в любом trade-off, истина лежит где-то по середине. Однако подбирать на практике «хорошее» N , чтобы получить оценки с промежуточным смещением и дисперсией, затруднительно. Но что ещё важнее, все N -шаговые оценки на самом деле неудачные. Во-первых, они плохи тем, что не используют всю доступную информацию: если мы знаем будущее на M шагов вперед, то странно не использовать награды за шаг за все эти M шагов, и странно не учесть прогноз нашей аппроксимации оценочной функции $V(s^{(t)})$ для всех доступных t . Во-вторых, странно, что в стационарной задаче, где всё инвариантно относительно времени, у нас появляется гиперпараметр, имеющий смысл количества шагов — времени. Поэтому нас будет интересовать далее альтернативный способ «интерполировать» между Монте-Карло и одношаговыми оценками. Мы всё равно оттолкнёмся именно от N -шаговых оценок, поскольку понятно, что эти оценки «корректны»: они направляют нас к решению уравнений Беллмана, для которых искомая V^π является единственной неподвижной точкой.

Мы придумали эти N -шаговые оценки, посмотрев на задачу под следующим углом: мы знаем сэмпл будущего (хвост траектории) и хотим выдать кредит «настоящему»: самому первому действию. Такой взгляд на оценки называется «*forward view*»: мы после выполнения a из s знаем «вперед» своё будущее и можем обновить оценочную функцию для этой пары.

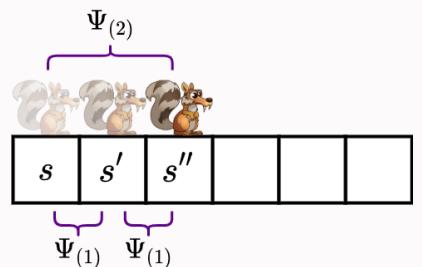


3.5.4. Backward View

Оказывается, мы можем посмотреть на задачу немного по-другому: можно, используя настоящее, обновлять кредит для прошлого. Рассмотрим эту идею, развив пример с кафе 57.

Пример 63: Ещё раз сядем в кафе (s) и захотим вернуться домой. Текущая аппроксимация даёт $-V(s) = 30$ минут. Делаем один шаг в среде: тратим одну минуту ($-r$) и обнаруживаем пробку (s'). Новая оценка времени возвращения даёт: $-V(s') = 40$ минут, соответственно с нами случилась одношаговая временная разность $\Psi_{(1)}(s, a) = 41 - 30 = 11$ минут, которая позволяет нам корректировать $V(s)$.

Давайте сделаем ещё один шаг в среде: тратим одну минуту $-r'$, видим пожар s'' и получаем новую оценку $-V(s'') = 60$ минут. Тогда мы можем посчитать как одношаговую временную разность для пары s', a' , равную $\Psi_{(1)}(s', a') = 61 - 40 = 21$ минуте, и уточнить свою аппроксимацию V-функции для состояния с пробкой; а можем посчитать и двухшаговую временную разность для кафе: мы потратили две минуты $r + r'$ на два шага и наша нынешнее приближение равно 60 минутам. Итого двухшаговая временная разность равна $\Psi_{(2)}(s, a) = 62 - 30 = 32$ минуты. Forward view говорит следующее: если мы хотим учиться на двухшаговые оценки вместо одношаговых, то нам не следовало на первом шаге использовать 11 минут для обновления $V(s)$, а нужно было дождаться второго шага, узнать двухшаговую ошибку в 32 минуты и воспользоваться ей.



Но понятно, что двухшаговая ошибка это сумма двух одношаговых ошибок! Напи 32 минуты ошибки — это 11 минут ошибки после выхода из кафе в пробку плюс 21 минута ошибки от выхода из пробки в пожар. Давайте после первого шага используем уже известную часть ошибки в 11 минут, а на втором шаге, если мы готовы обучаться с двухшаговых ошибок, возьмём и добавим недостающие 21 минуту.

Формализуем эту идею. Пусть мы взаимодействуем в среде при помощи стратегии π , которую хотим оценить; также будем считать learning rate α константным. После совершения первого шага «из кафе» мы можем, зная s, a, r, s' сразу же обновить нашу V-функцию:

$$V(s) \leftarrow V(s) + \alpha \Psi_{(1)}(s, a) \quad (3.44)$$

Затем мы делаем ещё один шаг в среде, узнавая a', r', s'' и временную разность за этот случившийся шаг $\Psi_{(1)}(s', a')$. Тогда мы можем просто ещё в нашу оценку V-функции добавить слагаемое:

$$V(s) \leftarrow V(s) + \alpha \gamma \Psi_{(1)}(s', a') \quad (3.45)$$

Непосредственной проверкой легко убедиться, что суммарное обновление V-функции получится эквивалентным двухшаговому обновлению:

Утверждение 35: Последовательное применение обновлений (3.44) и (3.45) эквивалентно двухшаговому обновлению

$$V(s) \leftarrow V(s) + \alpha \Psi_{(2)}(s, a)$$

Доказательство.

$$\begin{aligned} V(s) &\leftarrow V(s) + \alpha (\Psi_{(1)}(s, a) + \gamma \Psi_{(1)}(s', a')) = \\ &= V(s) + \alpha (r + \gamma V(s') - V(s) + \gamma r' + \gamma^2 V(s'') - \gamma V(s')) = \\ &= V(s) + \alpha (r + \gamma r' + \gamma^2 V(s'') - V(s)) = \\ &= V(s) + \alpha \Psi_{(2)}(s, a) \end{aligned}$$

■

Понятно, что можно обобщить эту идею с двухшаговых ошибок на N -шаговые: действительно, ошибка за N шагов равна сумме одношаговых ошибок за эти шаги.

Теорема 31:

$$\Psi_{(N)}(s, a) = \sum_{t=0}^{N-1} \gamma^t \Psi_{(1)}(s^{(t)}, a^{(t)}) \quad (3.46)$$

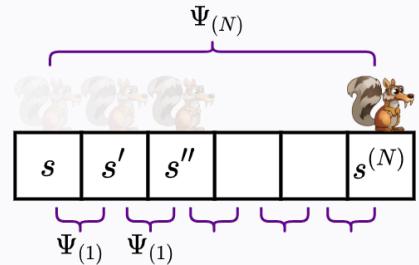
Доказательство. Докажем по индукции. Для $N = 1$ справа стоит только одно слагаемое, $\Psi_{(1)}(s, a)$, то есть одношаговая оценка.

Пусть утверждение верно для N , докажем для $N + 1$. В правой части при увеличении N на единицу появляется одно слагаемое, то есть для доказательства достаточно показать, что

$$\Psi_{(N+1)}(s, a) = \Psi_{(N)}(s, a) + \gamma^N \Psi_{(1)}(s^{(N)}, a^{(N)}) \quad (3.47)$$

Убедимся в этом, подставив определения:

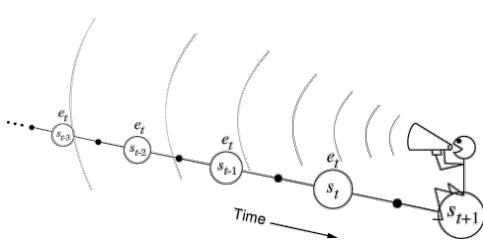
$$\begin{aligned} \Psi_{(N+1)}(s, a) &= \sum_{t=0}^N \gamma^t r^{(t)} + \gamma^{N+1} V(s^{(N+1)}) - V(s) = \\ &= \sum_{t=0}^{N-1} \gamma^t r^{(t)} + \gamma^N V(s^{(N)}) - V(s) + \gamma^N (r^{(N)} + \gamma V(s^{(N+1)}) - V(s^{(N)})) = \\ &= \Psi_{(N)}(s, a) + \gamma^N \Psi_{(1)}(s^{(N)}, a^{(N)}) \end{aligned}$$



Это наблюдение открывает, что все наши формулы обновления выражаются через одношаговые ошибки — $\Psi_{(1)}(s, a)$. Это интересный факт, поскольку одношаговая временная разность

$$\Psi_{(1)}(s, a) = r + \gamma V(s') - V(s)$$

очень похожа на reward shaping (1.7), где в качестве потенциала выбрана наша текущая аппроксимация $V(s)$. Поэтому эти **дельты**, как их ещё иногда называют, можно интерпретировать как некие «новые награды», центрированные — которые в среднем должны быть равны нулю, если аппроксимация V -функции точная.



Итого, оказывается, мы можем на каждом шаге добавлять к оценкам V -функции ранее встречавшихся в эпизоде пар s, a только что случившуюся одношаговую ошибку $\Psi_{(1)}(s, a)$ и таким образом получать N -шаговые обновления: достаточно пару s, a , посещённую K шагов назад, обновить с весом γ^K . То есть мы начинаем действовать по-другому: зная, что было в прошлом, мы правильным образом обновляем оценочную функцию посещённых состояний из прошлого, используя временную разность за один последний шаг («настоящее»). Такой подход к обновлению оценочной функции называется, соответственно, «**backward view**», и он позволяет взглянуть на обучение оценочных функций под другим углом.

3.5.5. Eligibility Trace

Рассмотрим случай $N = +\infty$, то есть допустим, что мы готовы обновлять V-функцию с reward-to-go. Наше рассуждение, можно сказать, позволяет теперь это делать, не доигрывая эпизоды до конца: мы сразу же в ходе эпизода можем уже «начинать» сдвигать V-функцию в правильном направлении. Это позволяет запустить «как бы Монте-Карло» алгоритм даже в неэпизодичных средах. Однако, на каждом шаге нам нужно перебирать все встретившиеся ранее в эпизоде состояния, чтобы обновить их, и, если эпизоды длинные (или среда неэпизодична), это хранение истории на самом деле становится избыточным.

Допустим, мы сделали шаг в среде и получили на этом одном шаге какую-то одношаговую ошибку $\Psi_{(1)}$. Рассмотрим какое-нибудь состояние s . С каким весом, помимо learning rate, нужно добавить эту ошибку к нашей текущей аппроксимации? Это состояние в течение прошлого эпизода было, возможно, посещено несколько раз, и за каждое посещение вес увеличивается на γ^K , где K — число шагов с момента посещения. Такой счётчик можно сохранить в памяти: заведём вектор $e(s)$ размером с число состояний, проинициализируем его нулём и далее на t -ом шаге будем обновлять следующим образом:

$$e_t(s) := \begin{cases} \gamma e_{t-1}(s) + 1 & \text{если } s = s_t \\ \gamma e_{t-1}(s) & \text{иначе} \end{cases} \quad (3.48)$$

После этого на каждом шаге мы будем добавлять текущую одношаговую ошибку, временную разность $\Psi_{(1)}(s_t, a_t) = r_t + \gamma V(s_{t+1}) - V(s_t)$, ко всем состояниям с коэффициентом $e_t(s)$.

Определение 61: Будем называть $e_t(s)$ **следом** (eligibility trace) для состояния s в момент времени t эпизода коэффициент, с которым алгоритм обновляет оценочную функцию $V(s)$ на t -ом шаге при помощи текущей одношаговой ошибки $\Psi_{(1)}(s_t, a_t)$:

$$V(s) \leftarrow V(s) + \alpha e_t(s) \Psi_{(1)}(s_t, a_t) \quad (3.49)$$

Допустим, эпизод доигран до конца, и мы в алгоритме используем формулу (3.49) со следом (3.48). Одношаговое обновление будет превращено в двухшаговое, двухшаговое — в трёхшаговое и так далее до N -шагового, где N — количество шагов до конца эпизода. Таким образом (если в ходе таких обновлений learning rate и оцениваемая стратегия не меняется) наши обновления в точности соответствуют Монте-Карло.

Важно, что eligibility trace имеет физический смысл «кредита», который выдан решениям, принятым в состоянии s : это та степень ответственности, с которой выбранное в этом состоянии действие влияют на события настоящего, на получаемые сейчас награды (временные разности). Действительно, обновление (3.49) говорит следующее: прогноз будущей награды в состоянии s нужно увеличить с некоторым learning rate-ом на получаемую награду ($\Psi_{(1)}(s_t, a_t)$), домноженную на степень ответственности решений в s за события настоящего ($e_t(s)$). И пока мы используем Монте-Карло обновления, кредит ведёт себя так: как только мы принимаем в s решение, он увеличивается на единичку и дальше не затухает. Домножение на γ можно не интерпретировать как затухание, поскольку это вызвано дисконтированием награды в нашей задаче, «затуханием» самой награды со временем. То есть мы рисуем мелом на стене «я здесь был», и выдаём постоянный кредит этому состоянию.

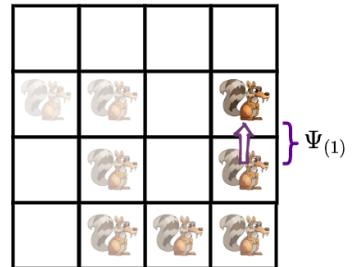
А что, если мы не хотим использовать бесконечношаговые (Монте-Карло) обновления? Мы помним, что это одна крайность, в которой обновления имеют большую дисперсию. Можно броситься в другую крайность: если мы хотим ограничиться лишь, допустим, одношаговыми, то мы можем использовать просто «другое» определение следа:

$$e_t(s) := \begin{cases} 1 & \text{если } s = s_t \\ 0 & \text{иначе} \end{cases}$$

То есть для одношаговых оценок нам нужно домножать след не на γ , а на ноль. Тогда вектор $e(s)$ на каждой итерации будет нулевым за исключением одного лишь только что встреченного состояния s_t , для которого он будет равен единице, и обновление 3.49 будет эквивалентно обычному одношаговому temporal difference. В таком кредите решение, принятое в s , влияет только на то, что произойдёт на непосредственно том же шаге; «мел на стене испаряется мгновенно».

3.5.6. TD(λ)

Как можно интерполировать между Монте-Карло обновлениями и одношаговыми с точки зрения backward view? Раз eligibility trace может затухать в γ раз, а может в 0, то, вероятно, можно тушить его и любым другим промежуточным способом. Так мы теперь можем придумать другой вид «промежуточных оценок» между Монте-Карло и одношаговыми. Пусть на очередном моменте времени след для состояния s затухает сильнее, чем в γ раз, но больше, чем в ноль; что тогда произойдёт?



После момента посещения состояния s след для него вырастет на единицу и оценочная функция обновится следующим образом:

$$V(s) \leftarrow V(s) + \alpha \Psi_{(1)}(s, a)$$

Допустим, на следующем шаге мы выбрали $\lambda_1 \in [0, 1]$ — «коэффициент затухания» — и потушили след не с коэффициентом γ , а с коэффициентом $\gamma\lambda_1$. Тогда дальше мы добавим новую текущую временную разность $\Psi_{(1)}(s', a')$ с коэффициентом $\lambda\gamma$, получая суммарно следующее обновление:

$$V(s) \leftarrow V(s) + \alpha (\Psi_{(1)}(s, a) + \lambda_1 \gamma \Psi_{(1)}(s', a')),$$

которое в силу (3.46) для $N = 2$ преобразуется в:

$$V(s) \leftarrow V(s) + \alpha ((1 - \lambda_1) \Psi_{(1)}(s, a) + \lambda_1 \Psi_{(2)}(s, a))$$

Таким образом, мы **заансамблировали** одношаговое и двухшаговое обновление. Из этого видно, что такая процедура корректна: V^π является неподвижной точкой как одношагового, так и двухшагового уравнения Беллмана, и значит неподвижной точкой любой их выпуклой комбинации.

С точки зрения кредита, мы сказали, что решение, принятое в s , влияет на то, что случится через 2 шага, но не так сильно, как на то, что случится через 1 шаг: степень ответственности за один шаг затухла в λ_1 раз. Мы пользуемся здесь следующим прайором из реальной жизни: решения более вероятно влияют на ближайшее будущее, нежели чем на далёкое.

Для понятности проведём ещё один шаг рассуждений. Допустим, мы сделали ещё один шаг в среде и увидели $\Psi_{(1)}(s'', a'')$; потушили eligibility trace $e(s)$, на этот раз, в $\gamma\lambda_2$ раз, где $\lambda_2 \in [0, 1]$. Тогда след стал равен $e(s) = \gamma^2\lambda_1\lambda_2$, и мы получим следующее обновление:

$$V(s) \leftarrow V(s) + \alpha ((1 - \lambda_1) \Psi_{(1)}(s, a) + \lambda_1 \Psi_{(2)}(s, a) + \gamma^2 \lambda_1 \lambda_2 \Psi_{(1)}(s'', a''))$$

Вспоминая формулу (3.47), последние два слагаемых преобразуются:

$$\Psi_{(2)}(s, a) + \gamma^2 \lambda_2 \Psi_{(1)}(s'', a'') = (1 - \lambda_2) \Psi_{(2)}(s, a) + \lambda_2 \Psi_{(3)}(s, a)$$

То есть долю λ_2 двухшаговой ошибки мы превращаем в трёхшаговое, а долю $1 - \lambda_2$ — нет. Суммарное обновление становится таким ансамблем:

$$V(s) \leftarrow V(s) + \alpha ((1 - \lambda_1) \Psi_{(1)}(s, a) + \lambda_1 (1 - \lambda_2) \Psi_{(2)}(s, a) + \lambda_1 \lambda_2 \Psi_{(3)}(s, a))$$

И так далее. Интерпретировать это можно так. Если мы тушим след в γ раз, то на очередном шаге обновления мы «превращаем» N -шаговое обновление в $N + 1$ -шаговое. Если мы тушим след полностью, зануляя его, то мы «отказываемся превращать» N -шаговое обновление в $N + 1$ -шаговое. Оба варианта корректны, и поэтому мы, выбирая $\lambda_t \in [0, 1]$, решаем заансамблировать их: мы возьмём и долю λ_t N -шагового обновления «превратим» в $N + 1$ -шаговое, а долю $1 - \lambda_t$ трогать не будем и оставим без изменения. Поэтому λ_t ещё называют «коэффициентом смешивания».

Мы уже обсуждали, что странно проводить credit assignment нестационарно, то есть чтобы в процедуре была какая-то зависимость от времени, прошедшего с момента посещения состояния, поэтому коэффициент затухания $\lambda \in [0, 1]$ обычно полагают не зависящим от момента времени, каким-то константным гиперпараметром, отвечающим за bias-variance trade-off. Естественно, $\lambda = 1$ соответствует Монте-Карло обновлениям, $\lambda = 0$ — одношаговым.

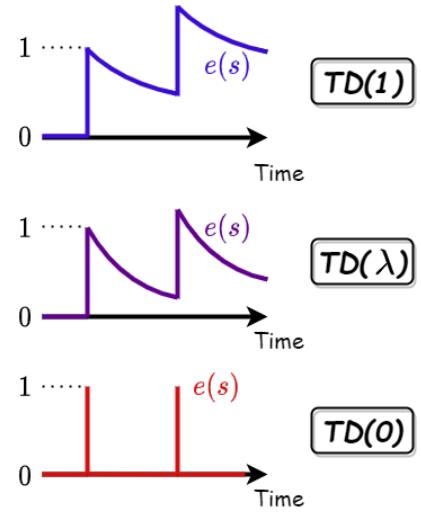
Определение 62: Будем называть *temporal difference обновлением* с параметром $\lambda \in [0, 1]$ («обновление TD(λ)») обновление (3.49) со следом $e_t(s)$, проинициализированным нулём и далее определённым следующим образом:

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

Формула следа задаёт алгоритм в парадигме backward view. Естественно, что любая оценка, придуманная в терминах backward view, то есть записанная в терминах следа (в явном виде хранящего «кредит» ответственности решений для каждого состояния), переделывается в парадигме forward view (как и наоборот), когда мы, используя сэмплы будущего, строим некоторую оценку $\Psi(s, a) \approx A^\pi(s, a)$ и просто сдвигаем по нему значение V-функции:

$$V(s) \leftarrow V(s) + \alpha \Psi(s, a) \tag{3.50}$$

Какому обновлению в парадигме forward view соответствует обновление TD(λ)?



Теорема 32 — Эквивалентные формы TD(λ): Обновление TD(λ) эквивалентно (3.50) с оценкой

$$\Psi(s, a) := \sum_{t \geq 0} \gamma^t \lambda^t \Psi_{(1)}(s^{(t)}, a^{(t)}) = (1 - \lambda) \sum_{t > 0} \lambda^{t-1} \Psi_{(t)}(s, a) \quad (3.51)$$

Доказательство. Составим такую табличку: какое суммарное обновление у нас получается в TD(λ) после t шагов в среде. Справа запишем, с какими весами входят разные N -шаговые оценки в получающийся ансамбль.

Step	Update	$\Psi_{(1)}(s, a)$	$\Psi_{(2)}(s, a)$	$\Psi_{(3)}(s, a)$...	$\Psi_{(N)}(s, a)$
1	$\Psi_{(1)}(s, a)$	1	0	0		0
2	$\Psi_{(1)}(s, a) + \gamma \lambda \Psi_{(1)}(s', a')$	$1 - \lambda$	λ	0		
3	$\Psi_{(1)}(s, a) + \gamma \lambda \Psi_{(1)}(s', a') + (\gamma \lambda)^2 \Psi_{(1)}(s'', a'')$	$1 - \lambda$	$(1 - \lambda)\lambda$	λ^2		0
\vdots					\ddots	
N	$\sum_{t \geq 0}^N (\gamma \lambda)^t \Psi_{(1)}(s^{(t)}, a^{(t)})$	$1 - \lambda$	$(1 - \lambda)\lambda$	$(1 - \lambda)\lambda^2$		λ^N

Продолжая строить такую табличку, можно по индукции увидеть, что после окончания эпизода суммарное обновление V-функции получится следующим:

$$V(s) \leftarrow V(s) + \alpha \sum_{t \geq 0} (\gamma \lambda)^t \Psi_{(1)}(s^{(t)}, a^{(t)})$$

Это и есть суммарное обновление V-функции по завершении эпизода. ■

Итак, полученная формула обновления имеет две интерпретации. Получается, что подобный ансамбль многошаговых оценок эквивалентен дисконтированной сумме будущих одношаговых ошибок («модифицированных наград» с потенциалом $V(s)$), где коэффициент дисконтирования равен $\gamma \lambda$; исходный коэффициент γ отвечает за затухание ценности наград со временем из исходной постановки задачи, а λ соответствует затуханию кредита ответственности действия за полученные в будущем награды.

А с другой стороны можно интерпретировать TD(λ) как ансамбль многошаговых оценок разной длины. Мы взяли одношаговую оценку с весом λ , двухшаговую с весом λ^2 , N -шаговую с весом λ^N и так далее. Сумма весов в ансамбле, как водится, должна равняться единице, отсюда в формуле домножение на $1 - \lambda$.

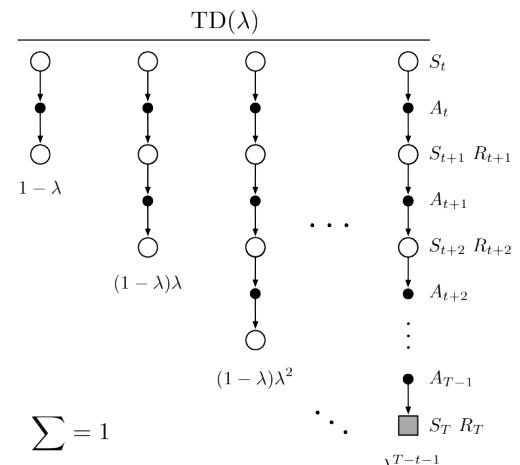
Если через N шагов после оцениваемого состояния s эпизод закончился, то все многошаговые оценки длины больше N , поганятно, совпадают с N -шаговой и равны reward-to-go. Следовательно, в такой ситуации reward-to-go по определению замещивается в оценку с весом $(1 - \lambda)(\lambda^{N-1} + \lambda^N + \dots) = \lambda^{N-1}$.

Мы привыкли, что любая формула обновления для нас — стохастическая аппроксимация решения какого-то уравнения. TD(λ) не исключение. Если N -шаговая оценка направляет нас в сторону решения N -шагового уравнения Беллмана $V^\pi = \mathcal{B}^N V^\pi$, то ансамбль из оценок направляет нас в сторону решения ансамбля N -шаговых уравнений Беллмана:

$$V^\pi = (1 - \lambda)(\mathcal{B}V^\pi + \lambda\mathcal{B}^2V^\pi + \lambda^2\mathcal{B}^3V^\pi + \dots) = (1 - \lambda) \sum_{N > 0} \lambda^{N-1} \mathcal{B}^N V^\pi \quad (3.52)$$

Поскольку V^π является неподвижной точкой для операторов \mathcal{B}^N для всех N , то и для их выпуклой комбинации, «ансамбля», он тоже будет неподвижной точкой. Итого TD(λ) дало нам прикольную идею: мы не могли выбрать одну из многошаговых оценок, и поэтому взяли их все сразу.

Итак, мы получили алгоритм TD(λ) оценивания стратегии, или temporal difference с параметром λ , который при $\lambda = 1$ эквивалентен Монте-Карло алгоритму (с постоянным обновлением V-функции «по ходу эпизода»), а при $\lambda = 0$ эквивалентен одношаговому temporal difference методу, который мы, в частности, применяли в Q-learning и SARSA для оценивания стратегии.



Алгоритм 14: TD(λ)

Вход: π — стратегия

Гиперпараметры: α — параметр экспоненциального сглаживания, $\lambda \in [0, 1]$ — степень затухания следа

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

Инициализируем $e(s)$ нулём для всех $s \in \mathcal{S}$

Наблюдаем s_0

На k -ом шаге:

1. выбираем $a_k \sim \pi(a_k | s_k)$
2. играем a_k и наблюдаем r_k, s_{k+1}
3. обновляем след $e(s_k) \leftarrow e(s_k) + 1$
4. считаем одношаговую ошибку $\Psi_{(1)} := r_k + \gamma V(s_{k+1}) - V(s_k)$
5. для всех s обновляем $V(s) \leftarrow V(s) + \alpha e(s) \Psi_{(1)}$
6. для всех s обновляем $e(s) \leftarrow \gamma \lambda e(s)$

Выход: $V(s)$

Мы обсуждали и выписали этот алгоритм для V-функции для задачи именно оценивания стратегии; естественно, мы могли бы сделать это для Q-функции или добавить policy improvement после, например, каждого шага в среде, получив табличный алгоритм обучения стратегии. Позже в разделе 3.5.7 мы рассмотрим формулировку теоремы о сходимости таких алгоритмов для ещё более общей ситуации.

Очевидно, TD(λ) обновление не эквивалентно никаким N -шаговым temporal difference формулам: в нём замешана как Монте-Карло оценка, то есть замешана вся дальнейшая награда (весь будущий сигнал), так и приближения V-функции во всех промежуточных состояниях (при любом $\lambda \in (0, 1)$). Гиперпараметр λ также не имеет смысла времени, и поэтому на практике его легче подбирать.



Полезность TD(λ) в том, что λ непрерывно и позволяет более гладкую настройку «длины следа». На практике алгоритмы будут чувствительны к выбору λ в намного меньшей степени, чем к выбору N . При этом даже если $\lambda < 1$, в оценку «поступает» информация о далёкой награде, и использование TD(λ) позволит бороться с проблемой распространения сигнала.

Всюду далее в ситуациях, когда нам понадобится разрешать bias-variance trade-off, мы будем обращаться к формуле forward view TD(λ) обновления (3.51). Однако как отмечалось ранее, для работы с многошаговыми оценками, а следовательно и с обновлением (3.51) TD(λ), необходимо работать в on-policy режиме, то есть иметь сэмплы взаимодействия со средой именно оцениваемой стратегии $\pi(a | s)$, и поэтому возможность разрешать bias-variance trade-off у нас будет только в on-policy алгоритмах.

3.5.7. Retrace(λ)

Что же тогда делать в off-policy режиме? Итак, пусть дан роллут $s, a, r, s', a', r', s'', \dots$, где действия сэмплированы из стратегии μ . Мы хотим при этом провести credit assignment для стратегии π , то есть понять, как обучать $V \approx V^\pi$ или $Q \approx Q^\pi$.

Проблема в том, что на самом деле это не всегда даже в принципе возможно. Представим, что a таково, что $\pi(a | s) = 0$. Тогда в роллайте хранится информация о событиях, которые произойдут с агентом, использующем стратегию π , с вероятностью 0. Понятно, что никакой полезной информации о том, как менять аппроксимацию V-функции, мы из таких данных не получим. Поэтому для детерминированных стратегий задача обучения с многошаговых оценок в off-policy режиме может запросто оказаться бессмысленной.

Пример 64: Допустим, стратегия μ с вероятностью один в первом же состоянии прыгает в лаву. Мы же хотим посчитать $V^\pi(s)$, будущую награду, для стратегии π , которая с вероятностью один не прыгает в лаву, а кушает тортики. Поскольку в задаче RL функция переходов $p(s' | s, a)$ для разных a может быть произвольно разной, мы ничего не можем сказать о ценности одного действия по информации о другом действии.

Возможность в принципе обучаться off-policy в Q-learning обеспечивалась тем, что, когда мы учим Q-функцию с одношаговых оценок, нам для любых s, a , которые можно взять из буфера, достаточно лишь сэмпла s' . При этом сэмпл $a' \sim \pi(a' | s')$ мы всегда сможем сгенерировать «онлайн», прогнав на взятом из буфера s' оцениваемую стратегию. Обычно в off-policy режиме учат именно Q-функцию, что важно в том числе тем, что по крайней мере одношаговая оценка будет доступна всегда. Если же a' из буфера такого, что $\pi(a' | s') = 0$,

то любые наши коррекции схлопнутся в одношаговую оценку (или же будут теоретически некорректны), но по крайней мере хоть что-то мы сможем сделать.

Итак, попробуем разрешить bias-variance trade-off-а для Q-функции. Для этого снова вернёмся к идеи следа. Допустим, для взятых из буфера s, a, r, s' мы составили одношаговое обновление:

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma Q(s', a'_\pi) - Q(s, a)),$$

где $a'_\pi \sim \pi(\cdot | s')$, положив неявно значение следа $e(s, a) = 1$ (след при обучении Q-функции зависит от пары s, a). Также лучше, если есть такая возможность, использовать не сэмпл a'_π , а усреднить по нему (аналогично тому, как было проделано в формуле (3.39) Expected SARSA):

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma \mathbb{E}_{a'_\pi \sim \pi} Q(s', a'_\pi) - Q(s, a)) \quad (3.53)$$

Затем возьмём из буфера a', r', s'' . Какое значение следа мы можем выбрать? Можно $e(s, a) = 0$, оставив одношаговое обновление и «не превращая» его в двухшаговое, это одна крайность. А как превратить обновление в двухшаговое целиком? Хотелось бы прибавить

$$\gamma(r' + \gamma \mathbb{E}_{a''_\pi \sim \pi} Q(s'', a''_\pi) - \mathbb{E}_{a'_\pi \sim \pi} Q(s', a'_\pi)),$$

где $s'' \sim p(s'' | s', a'_\pi)$. Однако в буфере у нас нет такого сэмпла, а вместо него есть сэмпл $s'' \sim p(s'' | s', a')$. Технически, ошибка за второй шаг является случайной величиной от a' , который должен приходить из π , когда у нас есть сэмпл лишь из μ . Поэтому здесь необходимо применить importance sampling коррекцию, после которой ошибка за второй шаг принимает следующий вид⁷:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \gamma \frac{\pi(a' | s')}{\mu(a' | s')} (r' + \gamma \mathbb{E}_{a''_\pi \sim \pi} Q(s'', a''_\pi) - Q(s', a')), \quad (3.54)$$

где a', r', s'' — взяты из буфера.

Утверждение 36: Последовательное применение обновлений (3.53) и (3.54) является корректным двухшаговым обновлением, то есть эквивалентно

$$Q(s, a) \leftarrow Q(s, a) + \alpha(y(Q) - Q(s, a)),$$

где среднее значение $y(Q)$ по всей заложенной в ней стохастике является правой частью двухшагового уравнения Беллмана для Q-функции:

$$\mathbb{E}y(Q) = \mathbb{E}_{s'} \mathbb{E}_{a' \sim \pi} \mathbb{E}_{s'' \sim \pi} \mathbb{E}_{a'' \sim \pi} [r(s, a) + \gamma r(s', a') + \gamma^2 Q(s'', a'')]$$

Доказательство. После двух рассматриваемых обновлений получается, что «целевая переменная», таргет, равен:

$$y(Q) = r + \gamma \mathbb{E}_{a'_\pi \sim \pi} Q(s', a'_\pi) + \gamma \frac{\pi(a' | s')}{\mu(a' | s')} (r' + \gamma \mathbb{E}_{a''_\pi \sim \pi} Q(s'', a''_\pi) - Q(s', a')),$$

где случайными величинами являются $s', a', s'',$ и $a' \sim \mu(a' | s')$. Возьмём среднее по этим величинам:

$$\mathbb{E}y(Q) = \mathbb{E}_{s'} \left[r + \gamma \mathbb{E}_{a'_\pi \sim \pi} Q(s', a'_\pi) + \gamma \mathbb{E}_{a' \sim \mu} \frac{\pi(a' | s')}{\mu(a' | s')} (r' + \gamma \mathbb{E}_{s'' \sim \pi} \mathbb{E}_{a''_\pi \sim \pi} Q(s'', a''_\pi) - Q(s', a')) \right]$$

Раскроем importance sampling коррекцию, воспользовавшись

$$\mathbb{E}_{a' \sim \mu} \frac{\pi(a' | s')}{\mu(a' | s')} f(a') = \mathbb{E}_{a' \sim \pi} f(a'),$$

⁷видно, что в отношении последнего слагаемого возможны вариации, например, ошибку за второй шаг можно оценить как

$$\gamma \frac{\pi(a' | s')}{\mu(a' | s')} (r' + \gamma \mathbb{E}_{a''_\pi \sim \pi} Q(s'', a''_\pi) - \mathbb{E}_{a'_\pi \sim \pi} Q(s', a'_\pi)),$$

или даже не корректировать последнее слагаемое importance sampling коррекцией, так как в нём не требуется брать a' обязательно из буфера:

$$\gamma \frac{\pi(a' | s')}{\mu(a' | s')} (r' + \gamma \mathbb{E}_{a''_\pi \sim \pi} Q(s'', a''_\pi)) - \gamma \mathbb{E}_{a'_\pi \sim \pi} Q(s', a'_\pi).$$

Далее в формулах предполагается вариант, рассматриваемый в статье про Retrace(λ).

получим:

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

Осталось заметить, что слагаемое $\gamma Q(s', a')$ по аналогии с on-policy режимом сокращается. ■

Таким образом, одношаговую ошибку за второй шаг для получения полного превращения одношагового обновления в двухшаговое необходимо добавить не с весом γ , как в on-policy режиме, а с весом $\gamma \frac{\pi(a'|s')}{\mu(a'|s')}$.

Продолжая рассуждение дальше, можно получить, что одношаговая ошибка через t шагов после выбора оцениваемого действия a в состоянии s в off-policy режиме зависит от случайных величин $s', a', s'', \dots, s^{(t+1)}$, и поэтому importance sampling коррекция для неё будет равна:

$$\prod_{i=1}^{t=t} \frac{\pi(a^{(i)} | s^{(i)}) p(s^{(i+1)} | s^{(i)}, a^{(i)})}{\mu(a^{(i)} | s^{(i)}) p(s^{(i+1)} | s^{(i)}, a^{(i)})} = \prod_{i=1}^{t=t} \frac{\pi(a^{(i)} | s^{(i)})}{\mu(a^{(i)} | s^{(i)})}$$

Здесь и далее считается, что при $t = 0$ подобные произведения равны единице.

Получается, что для того, чтобы строить оценку максимальной длины, нужно на t -ом шаге домножать след на $\gamma \frac{\pi(a^{(t)} | s^{(t)})}{\mu(a^{(t)} | s^{(t)})}$. Итоговую формулу обновления часто записывают в следующем виде:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \sum_{t \geq 0} \gamma^t \left(\prod_{i=1}^{t=t} \frac{\pi(a^{(i)} | s^{(i)})}{\mu(a^{(i)} | s^{(i)})} \right) \Psi_{(1)}(s^{(t)}, a^{(t)}), \quad (3.55)$$

где

$$\Psi_{(1)}(s^{(t)}, a^{(t)}) = r^{(t)} + \gamma \mathbb{E}_{a_\pi^{(t+1)} \sim \pi} Q(s^{(t+1)}, a_\pi^{(t+1)}) - Q(s^{(t)}, a^{(t)}) \quad (3.56)$$

Мы получили «off-policy» Монте-Карло оценку в терминах следа. Теперь по аналогии с TD(λ) проинтерполируем между одношаговыми (где след зануляется после первого шага) и бесконечношаговыми обновлениями (где след есть importance sampling дробь): оказывается, оценка будет корректна при любом промежуточном значении следа. Чтобы записать это формально, перепишем формулу (3.55) в следующем виде:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \sum_{t \geq 0} \gamma^t \left(\prod_{i=1}^{t=t} c_i \right) \Psi_{(1)}(s^{(t)}, a^{(t)}), \quad (3.57)$$

где c_i — коэффициенты затухания следа: в on-policy они могли быть в диапазоне $[0, 1]$ (и мы выбирали его равным гиперпараметру λ), а здесь, в off-policy режиме, он может быть в диапазоне

$$c_i \in \left[0, \frac{\pi(a^{(i)} | s^{(i)})}{\mu(a^{(i)} | s^{(i)})} \right]$$

То, что при любом выборе способа затухания следа табличные алгоритмы, использующие оценку (3.57), будут сходиться — один из ключевых и самых общих результатов табличного RL. Приведём несколько нестрогую формулировку этой теоремы:

Теорема 33 — Retrace: Пусть число состояний и действий конечно, таблица $Q_0(s, a)$ проинициализирована произвольно. Пусть на k -ом шаге алгоритма для каждой пары s, a ячейка таблицы обновляется по формуле

$$Q_{k+1}(s, a) \leftarrow Q_k(s, a) + \alpha_k(s, a) \sum_{t \geq 0} \gamma^t \left(\prod_{i=1}^{t=t} c_i \right) \Psi_{(1)}(s^{(t)}, a^{(t)}),$$

$$\Psi_{(1)}(s^{(t)}, a^{(t)}) = r^{(t)} + \gamma \mathbb{E}_{a_\pi^{(t+1)} \sim \pi_k} Q_k(s^{(t+1)}, a_\pi^{(t+1)}) - Q(s^{(t)}, a^{(t)})$$

где $\mathcal{T} \sim \mu_k | s_0 = s, a_0 = a$ сгенерирована произвольной стратегией сбора данных μ_k (причём не обязательно стационарной), learning rate $\alpha_k(s, a)$ — случайно, π_k произвольно, и коэффициенты следа — любые в диапазоне

$$c_i \in \left[0, \frac{\pi_k(a^{(i)} | s^{(i)})}{\mu_k(a^{(i)} | s^{(i)})} \right].$$

Тогда, если с вероятностью 1 learning rate удовлетворяет условиям Роббинса-Монро (3.26), а стратегия π_k с вероятностью 1 становится жадной по отношению к Q_k в пределе $k \rightarrow \infty$, то при некоторых технических ограничениях с вероятностью 1 Q_k сходится к оптимальной Q-функции Q^* , а π_k , соответственно, к оптимальной стратегии.

Без доказательства; интересующиеся могут обратиться к оригинальной статье по Retrace. ■

Пользуясь этой теоремой, мы можем в полной аналогии с $\text{TD}(\lambda)$ выбрать гиперпараметр $\lambda \in [0, 1]$, и считать след по формуле

$$c_i = \lambda \frac{\pi(a^{(i)} | s^{(i)})}{\mu(a^{(i)} | s^{(i)})}$$

В частности, в on-policy режиме $\pi \equiv \mu$ мы получим коэффициент затухания следа, равный просто λ . Это очень удобно, но на практике неприменимо.

Такая оценка страдает сразу от двух проблем. Первая проблема — типичная: **затухающий след** (vanishing trace), когда $\mu(a^{(t)} | s^{(t)}) \gg \pi(a^{(t)} | s^{(t)})$ для какого-то t , и соответствующий множитель близок к нулю. Такое случится, если, например, μ выбирает какие-то действия, которые π выбирает редко, что типично. В предельном случае для детерминированных стратегий может быть такое, что числитель коэффициента равен нулю (π не выбирает такого действия никогда), и коррекция скажет, что вся информация, начиная с этого шага, полностью неактуальна. Эта проблема неизбежна.

Вторая проблема — **взрывающийся след** (exploding trace): $\mu(a^{(t)} | s^{(t)}) \ll \pi(a^{(t)} | s^{(t)})$. Такое может случиться, если в роллауте попалось редкое для μ действие, которое тем не менее часто выполняется оцениваемой стратегией π . С одной стороны, кажется, что как раз такие роллауты наиболее ценные для обучения Q^π , ведь в них описывается шаг взаимодействия со средой, который для π как раз достаточно вероятен. Но на практике взрывающийся importance sampling коэффициент — источник большой дисперсии рассматриваемой оценки.

Название	Коэффициенты c_i	Проблема
$\text{TD}(\lambda)$	λ	только on-policy режим
Одношаговые	0	сильное смещение
Importance Sampling	$\lambda \frac{\pi(a^{(i)} s^{(i)})}{\mu(a^{(i)} s^{(i)})}$	легко взрываются

Идея борьбы со взрывающимся коэффициентом, предложенной в оценке Retrace(λ), заключается в следующем: если importance sampling коррекция для t -го шага взорвалась, давайте воспользуемся тем, что мы теоретически обоснованно можем выбрать любой коэффициент меньше («быстрее» потушить след), и выберем единицу:

$$c_i := \lambda \min \left(1, \frac{\pi(a^{(i)} | s^{(i)})}{\mu(a^{(i)} | s^{(i)})} \right) \quad (3.58)$$

Коррекция с такими коэффициентами корректна, стабильна, но, конечно, никак не помогает с затуханием следа. Важно помнить, что если π и μ сильно отличаются, то велика вероятность, что коэффициенты затухания будут очень близки к нулю, и мы получим что-то, практически всегда похожее на одношаговое обновление. С этим мы фундаментально не можем ничего сделать, хотя из-за этого итоговая формула обновления получается сильно смещённой. По этой причине смысла выбирать $\lambda < 1$ в off-policy режиме обычно мало, и его почти всегда полагают равным единице.

Также обсудим, что говорит формула Retrace в ситуации, когда оцениваемая политика π детерминирована. Если на шаге t политика сбора данных μ выбрала ровно то действие, которое выбирает политика π , то коэффициент $c_i = \lambda$, то есть ситуация совпадает с on-policy режимом (действительно, в дискретных пространствах действий в числителе единица, а в знаменателе что-то меньшее единицы, когда в непрерывных пространствах действий числитель технически равен бесконечности, поэтому мы обрежем дробь до единицы). В противном же случае дробь $\frac{\pi(a^{(i)} | s^{(i)})}{\mu(a^{(i)} | s^{(i)})}$ имеет ноль в числителе, и след занулится. Таким образом, пока в буфере в цепочке s', a', s'', a'', \dots записанные действия совпадают с теми, которые выбирает детерминированная π , мы «пользуемся $\text{TD}(\lambda)$ », но вынуждены оборвать след, как только очередное действие разойдётся. Есть некоторая польза в том, чтобы в алгоритме π и μ были стохастичны: тогда след по крайней мере полностью никогда не затухнет.

Мы дальше в off-policy будем обсуждать в основном одношаговые оценки, в том числе для простоты. Из одношаговости будут вытекать все ключевые недостатки таких алгоритмов, связанные со смещённостью подобных оценок и невозможностью полноценно разрешать bias-variance trade-off. Во всех этих алгоритмах с этой проблемой можно будет частично побороться при помощи идей Retrace(λ).

ГЛАВА 4

Value-based подход

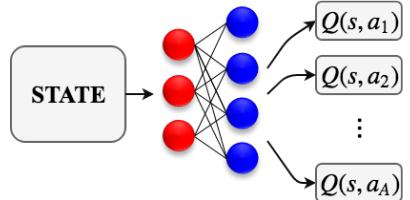
В данной главе будет рассмотрен второй, Value-based подход к решению задачи, в котором алгоритм ищет не саму стратегию, а оптимальную Q-функцию. Для этого табличный алгоритм Value Iteration будет обобщён на более сложные пространства состояний; требование конечности пространства действий $|\mathcal{A}|$ 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.

Попробуем промоделировать в сложных средах алгоритм 10 Q-learning. Для этого будем приближать оптимальную Q-функцию $Q^*(s, a)$ при помощи нейронной сети $Q_\theta(s, a)$ 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 $Q_\theta(s, a_1) \dots Q_\theta(s, a_{|\mathcal{A}|})$. В последнем случае мы можем за константу находить жадное действие $\pi(s) = \underset{a}{\operatorname{argmax}} Q_\theta(s, a)$.



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 в главе 6.1).



Использование нейросеток позволяет обучаться для сред, в которых состояния s заданы, например, пиксельным представлением 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 со пряжено с вопросами о том, нужно ли их включать-выключать на этапах генерации таргетов (который, как мы увидим позже, тоже распадается на два этапа), сборе 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

Как обучать параметры θ нейронной сети так, чтобы $Q_\theta(s, a) \approx Q^*(s, a)$? Вообще говоря, мы помним, что мы хотим решать уравнения оптимальности Беллмана (3.17), и можно было бы, например, оптимизировать невязку:

$$\left(Q_\theta(s, a) - r(s, a) - \gamma \mathbb{E}_{s'} \max_{a'} Q_\theta(s', a') \right)^2 \rightarrow \min_{\theta}$$

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 (3.34). Хочется сделать какой-то аналогичный трюк:

$$Q_\theta(s, a) \leftarrow Q_\theta(s, a) + \alpha \left(r(s, a) + \gamma \max_{a'} Q_\theta(s', a') - Q_\theta(s, a) \right)$$

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 3.35, 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 $Q_{\theta_k}(s, a)$, и мы хотим проделать шаг метода простой итерации для решения уравнения Q^*Q^* (3.17). Зададим следующую задачу регрессии:

- входом является пара s, a

- искомым (!) значением на паре s, a — правая часть уравнения оптимальности Беллмана (3.17), т.е.

$$f(s, a) := r(s, a) + \gamma \mathbb{E}_{s'} \max_{a'} Q_{\theta_k}(s', a')$$

- наблюдаемым («зашумлённым») значением целевой переменной или *таргетом* (target)

$$y(s, a) := r(s, a) + \gamma \max_{a'} Q_{\theta_k}(s', a') \quad (4.1)$$

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 34: Пусть $Q_{\theta_{k+1}}(s, a)$ — 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:

$$Q_{\theta_{k+1}}(s, a) = r(s, a) + \gamma \mathbb{E}_{s'} \max_{a'} Q_{\theta_k}(s', a')$$

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 $Q_{\theta_{k+1}}(s, a)$ выдаёт некоторое число, которое должно минимизировать MSE:

$$\frac{1}{2} \mathbb{E}_{s'} (y(s, a) - Q_{\theta_{k+1}}(s, a))^2 \rightarrow \min_{\theta_{k+1}}$$

Оптимальным константным решением регрессии с MSE как раз является среднее, получаем

$$Q_{\theta_{k+1}}(s, a) = \mathbb{E}_{s'} y(s, a) = r(s, a) + \gamma \mathbb{E}_{s'} \max_{a'} Q_{\theta_k}(s', a')$$

Другими словами, когда мы решаем задачу регрессии с целевой переменной $y(s, a)$ по формуле (4.1), мы в среднем сдвигаем нашу аппроксимацию в сторону правой части уравнения оптимальности Беллмана. Полезно наглядно увидеть это в формуле самого градиента. Представим на секунду, что мы решаем задачу регрессии с «идеальной», незашумлённой целевой переменной $f(s, a) := r(s, a) + \gamma \mathbb{E}_{s'} \max_{a'} Q_{\theta_k}(s', a')$: для одного примера s, a градиент MSE тогда будет равен:

$$\nabla_\theta \frac{1}{2} (f(s, a) - Q_\theta(s, a))^2 = \overbrace{(f(s, a) - Q_\theta(s, a))}^{\text{скаляр}} \underbrace{\nabla_\theta Q_\theta(s, a)}_{\substack{\text{направление} \\ \text{увеличения} \\ \text{значения } Q_\theta(s, a)}}$$

¹это не случайность: корни теории одни и те же.

И наша «зашумлённая» целевая переменная $y(s, a)$ есть несмешённая оценка $f(s, a)$, поскольку специально построена так, что $\mathbb{E}_{s'} y(s, a) = f(s, a)$. Значит, мы можем несмешённо оценить этот градиент как

$$(y(s, a) - Q_\theta(s, a)) \nabla_\theta Q_\theta(s, a) = \nabla_\theta \frac{1}{2} (y(s, a) - Q_\theta(s, a))^2 \quad (4.2)$$

Мы всегда оптимизируем нейронные сети стохастической градиентной оптимизации, поэтому несмешённая оценка градиента нам подойдёт.

В частности, формула Q-learning (3.34) — это частный случай решения указанной задачи регрессии стохастическим градиентным спуском для специального вида параметрических распределений:

Теорема 35: Пусть Q-функция задана «табличным параметрическим семейством», то есть табличкой размера $|\mathcal{S}|$ на $|\mathcal{A}|$, где в каждой ячейке записан параметр $\theta_{s,a}$:

$$Q_\theta(s, a) = \theta_{s,a}$$

Тогда формула (3.34) представляет собой градиентный спуск для решения обсуждаемой задачи регрессии.

Доказательство. Посчитаем градиент функции потерь на одном объекте s, a . Предсказанием модели будет $Q_\theta(s, a) = \theta_{s,a}$, то есть градиент предсказания по параметрам модели равен

$$\nabla_\theta Q_\theta(s, a) = e_{s,a}$$

где $e_{s,a}$ — вектор из $\mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$ из всех нулей с единственной единичкой в позиции, соответствующей паре s, a . Теперь посчитаем градиент функции потерь:

$$\nabla_\theta \frac{1}{2} (y - Q_\theta(s, a))^2 = (Q_\theta(s, a) - y(s, a)) \nabla_\theta Q_\theta(s, a) = (Q_\theta(s, a) - y(s, a)) e_{s,a}$$

Итак, градиентный спуск делает следующие апдейты параметров:

$$\theta_{k+1} = \theta_k - \alpha \nabla_\theta \text{Loss}(y, Q_\theta(s, a)) = \theta_k + \alpha (y(s, a) - Q_{\theta_k}(s, a)) e_{s,a}$$

В этой формуле на каждом шаге обновляется ровно одна ячейка таблички $\theta_{s,a}$, и это в точности совпадает с (3.34). ■

Само собой, аналогичный приём мы в будущем будем применять не только для обучения модели Q^* , но и любых других оценочных функций. Для этого мы будем составлять задачу регрессии, выбирая в качестве целевой переменной несмешённую оценку правой части какого-нибудь уравнения Беллмана, неподвижной точкой которого является искомая оценочная функция. Этот приём является полным аналогом методов временной разности, поэтому и свойства получаемых алгоритмов будут следовать из классической теории табличных алгоритмов.

4.1.3. Таргет-сеть

Рассмотренное теоретическое объяснение перехода от табличных методов к нейросетевым, конечно, предполагает, что мы решаем задачу регрессии «полностью», обучая θ при фиксированных θ_k . «Замороженные» θ_k соответствуют фиксированию формулы целевой переменной $y(s, a)$ (4.1), то есть фиксированию задачи регрессии. Так мы моделируем один шаг метода простой итерации, и только после этого объявляем выученные параметры модели $\theta_{k+1} := \theta$. В этот момент задача регрессии изменится (поменяется целевая переменная), и мы перейдём к следующему шагу метода простой итерации.

Однако в Q-learning же, как и во всех табличных методах, теория стохастической аппроксимации позволяла «сменять» задачу регрессии каждый шаг, используя свежие параметры модели при построении целевой переменной. Конечно, как только мы от «табличных параметрических функций» переходим к произвольным параметрическим семействам, все теоретические гарантии сходимости Q-learning-a 3.35 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, гарантый нет.

Но возникает вопрос: сколько шагов градиентного спуска тратить на решение фиксированной задачи регрессии? Возникает естественное желание по аналогии с табличными методами использовать для построения таргета свежую модель, то есть менять целевую переменную в задаче регрессии каждый шаг после каждого градиентного шага:

$$y(s, a) := r + \gamma \max_{a'} Q_\theta(s', a')$$

Принципиально важно, что зависимость целевой переменной $y(s, a)$ (4.1) от параметров текущей модели θ игнорировалась. 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.

Эмпирически легко убедиться, что такой подход нестабилен примерно от слова совсем. Стохастическая оптимизация чревата тем, что после очередного шага модель может стать немножко «сломанной» и некоторое время выдавать неудачные значения на ряде примеров. В обычном обучении с учителем этот эффект сглаживается большим количеством итераций: обучение на последующих мини-батчах «исправляют» предыдущие ошибки, движение идёт в среднем в правильную сторону, но нет гарантий удачности каждого конкретного шага (на то это и стохастическая оптимизация). Здесь же при неудачном шаге сломанная модель может начать портить целевую переменную, на которую она же и обучается. Это приводит к цепной реакции: плохая целевая переменная начнёт портить модель, которая начнёт портить целевую переменную... Этот эффект особенно ярко проявляется ещё и потому, что мы используем одношаговые целевые переменные, которые, как мы обсуждали в главе 3.5, сильно смещены (слишком сильно опираются на текущую же аппроксимацию).

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 $r + \gamma \max_{a'} Q_\theta(s, a)$, то после первого же градиентного шага θ поменяется.

Поэтому хранится копия Q -сетки, называемая *таргет-сетью* (target network), единственная цель которой — генерировать таргеты текущей задачи регрессии для транзишнов из заэмплированных мини-батчей. Традиционно её параметры обозначаются θ^- . Итак, целевая переменная в таких обозначениях генерится по формуле

$$y(\mathbb{T}) := r + \gamma \max_{a'} Q_{\theta^-}(s', a')$$

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:

$$\theta^- \leftarrow (1 - \alpha)\theta^- + \alpha\theta,$$

где α — гиперпараметр. Такой вариант тоже увеличивает стабильность алгоритма хотя бы потому, что решаемая задача регрессии меняется «постепенно».

4.1.4. Декорреляция сэмплов

Q -learning после одного шага в среде делал апдейт одного значения таблицы $Q(s, a) \approx Q^*(s, a)$. 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.

При наличии реплей буфера агент может решать задачи регрессии, сэмплируя мини-батчи переходов $\mathbb{T} = (s, a, r', s')$ из буфера, затем делая для каждого перехода расчёт таргета² $y(\mathbb{T}) := r + \gamma \max_{a'} Q_{\theta}(s', a')$, 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? Мы хотим решать уравнения оптимальности Беллмана для всех пар 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 такие же.



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).

4.1.5. DQN

Собираем алгоритм целиком. Нам придётся оставить ϵ -жадную стратегию исследования — 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 - done)**, поскольку шансов у приближённого динамического программирования сойтись куда-то без «отправной точки» не очень много.

Алгоритм 15: Deep Q-learning (DQN)

Гиперпараметры: B — размер мини-батчей, K — периодичность апдейта таргет-сети, $\epsilon(t)$ — стратегия исследования, Q — нейросеть с параметрами θ , SGD-оптимизатор

Инициализировать θ произвольно

²наличие реплей буфера не избавляет от необходимости использовать таргет-сеть, поскольку 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).

Положить $\theta^- := \theta$

Пронаблюдать s_0

На очередном шаге t :

1. выбрать a_t случайно с вероятностью $\varepsilon(t)$, иначе $a_t := \operatorname{argmax}_{a_t} Q_\theta(s_t, a_t)$
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:

$$y(\mathbb{T}) := r + \gamma(1 - \text{done}) \max_{a'} Q_{\theta^-}(s', a')$$

6. посчитать лосс:

$$\text{Loss}(\theta) := \frac{1}{B} \sum_{\mathbb{T}} (Q_\theta(s, a) - y(\mathbb{T}))^2$$

7. сделать шаг градиентного спуска по θ , используя $\nabla_\theta \text{Loss}(\theta)$

8. если $t \bmod K = 0$: $\theta^- \leftarrow \theta$

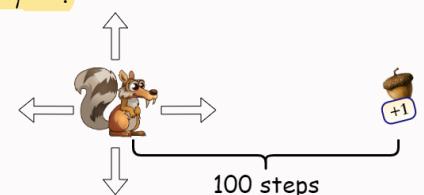
Все алгоритмы, которые относят к Value-based подходу в RL, будут основаны на DQN: само название «value-based» обозначает, что мы учим только оценочную функцию, а такое возможно только если мы учим модель Q-функции и полагаем, что policy improvement проводится на каждом шаге жадно. Неявно в DQN, конечно же, присутствует текущая политика, «целевая политика», которую мы оцениваем — $\operatorname{argmax}_a Q_\theta(s, a)$. Тем не менее ключевое свойство алгоритма, которое стоит помнить — это то, что он работает в off-policy режиме, и потому потенциально является достаточно sample efficient.

Есть, однако, много причин, почему алгоритм может не раскрыть этот потенциал и «плохо» заработать на той или иной среде: либо совсем не обучиться, либо обучаться очень медленно. Со многими из этих недостатков можно пытаться вполне успешно бороться, что и пытаются делать модификации этого алгоритма, которые мы обсудим далее в главе 4.2.

Выделим отдельно одну особую фундаментальную причину, почему алгоритмы на основе DQN могут не справиться с оптимизацией награды и застрять на какой-то асимптоте, с которой мало что можно сделать, и которая вытекает непосредственно из off-policy режима работы DQN. DQN, как и любые алгоритмы, основанные на одношаговых целевых переменных, страдает от проблемы **накапливающейся ошибки** (compound error). Условно говоря, чтобы распространить награду, полученную в некоторый момент времени, на 100 шагов в прошлое, понадобится провести 100 этапов метода простой итерации. Каждый этап мы решаем задачу регрессии в сильно неидеальных условиях, и ошибка аппроксимации накапливается. На это накладывается необходимость обучать именно Q-функцию, которая должна дифференцировать между действиями.

Пример 65: Рассмотрим типичную задачу: вы можете перемещаться в пространстве в разные стороны. Если вы отправитесь вправо, через 100 шагов вы получите +1. $Q^*(s, \text{вправо}) = \gamma^{100}$.

Посмотрим на значение функции в других действиях: например, $Q^*(s, \text{влево}) = \gamma^{102}$. Вот с такой точностью наша Q^* должна обучиться в этом состоянии, чтобы жадная стратегия выбирала правильные действия. Именно поэтому в подобных ситуациях DQN-подобные алгоритмы не срабатывают: из-за проблемы накапливающейся ошибки сигнал на 100 шагов просто не распространяется с такой точностью.



Если же этих проблем «с точностью» не возникает, если в среде нет сильно отложенного сигнала или награда за каждый шаг очень информативна, то value-based подход может раскрыть свой потенциал и оказаться эффективнее альтернатив за счёт использования реплей буфера.

§4.2. Модификации DQN

4.2.1. Overestimation Bias

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:

$$y(\mathbb{T}) = r + \gamma \max_{a'} Q_{\theta^-}(s', a')$$

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 66: Вы выиграли в лотерею +100 в силу везения. В вашем опыте нет примеров того, как вы купили билет и проиграли, и поэтому алгоритм будет учить завышенное значение $Q(s, a)$ для действия «купить билет в лотерею» (напомним, в утверждении 31 мы отмечали, что запущенный с реплей буфера Q-learning, как и DQN, учит оптимальную Q-функцию для эмпирического MDP). Это завышение из-за везения.

Из-за того, что вы моделируете Q-сетку нейросетью, при увеличении $Q(s, a)$ случайно увеличилось значение ценности действия «играть в казино», поскольку оно опиралось на примерно те же признаки описания состояний. Это завышение из-за ошибки аппроксимации.

А дальше начинается «цепная реакция»: при построении таргета для состояния «казино» s' в задачу регрессии поступают завышенные значения, предвещавшие кучу награды. Завышенное значение начинает распространяться дальше на другие состояния: вы начинаете верить, что вы всегда можете пойти в казино и получить кучу награды.

Утверждение 37: Рассмотрим одно s' . Пусть $Q^*(s', a')$ — истинные значения, и для каждого действия a' есть модель $Q(s', a') \approx Q^*(s', a')$, которая оценивает это действие с некоторой погрешностью $\varepsilon(a')$:

$$Q(s', a') = Q^*(s', a') + \varepsilon(a')$$

Пусть эти погрешности $\varepsilon(a')$ независимы по действиям, а завышение и занижение оценки равновероятны:

$$\mathbf{P}(\varepsilon(a') > 0) = \mathbf{P}(\varepsilon(a') < 0) = 0.5$$

Тогда оценка максимума скорее завышена, чем занижена:

$$\mathbf{P}\left(\max_{a'} Q(s', a') > \max_{a'} Q^*(s', a')\right) > 0.5$$

Доказательство. С вероятностью 0.5 модель переоценит самое хорошее действие, на котором достигается максимум истинной $Q^*(s', a')$; и ещё к тому же с некоторой ненулевой вероятностью возникнет настолько завышенная оценка какого-нибудь из других действий, что $Q(s', a') > \max_{a'} Q^*(s', a')$. ■

Пример 67: Предположим, для s' у нас есть три действия, и на самом деле $Q^*(s', a') = 0$ для всех трёх возможных действий. Мы оцениваем каждое действие с некоторой погрешностью. Допустим даже, эта погрешность в среднем равна нулю и распределена по Гауссу: $Q(s', a') \sim \mathcal{N}(0, \sigma^2)$. Тогда случайная величина $\max_{a'} Q(s', a')$ — взятие максимума из трёх сэмплов из гауссианы — очевидно такова, что

$$\mathbb{E} \max_{a'} Q(s', a') > 0$$

Хотя истинный максимум $\max_{a'} Q^*(s', a') = 0$. Получается, что, несмотря на то, что каждый элемент оценён несмещённо, максимум по аппроксимациям будет смещён в сторону завышения.

Одно из хороших решений проблемы заключается в разделении (decoupling) двух этапов подсчёта максимума:

выбор действия (action selection) и **оценка действия** (action evaluation):

$$\max_{a'} Q(s', a') = \underbrace{Q(s', \operatorname{argmax}_{a'} Q(s', a'))}_{\text{выбор действия}} \quad \underbrace{\overbrace{Q(s', \operatorname{argmax} Q(s', a'))}^{\text{оценка действия}}}_{\text{выбор действия}}$$

Действительно: мы словно дважды используем одну и ту же погрешность при выборе действия и оценке действия. Из-за скоррелированности ошибки на этих двух этапах и возникает эффект завышения.

Основная идея борьбы с этой проблемой заключается в следующем: предлагается³ обучать два приближения Q-функции параллельно и использовать аппроксимацию Q-функции «независимого близнец» для этапа оценивания действия:

$$y_1(T) := r + \gamma \underbrace{Q_{\theta_2}(s', \operatorname{argmax}_{a'} Q_{\theta_1}(s', a'))}_{a'}$$

$$y_2(T) := r + \gamma Q_{\theta_1}(s', \operatorname{argmax}_{a'} Q_{\theta_2}(s', a'))$$

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, дорогостоящее), их ошибки аппроксимации и везения будут в «разных местах». В итоге, Q-функция близнецка выступает в роли более пессимистичного критика действия, выбираемого текущей Q-функцией.



Если сбор уникального опыта для каждого из близнецов не организуется, Q-функции всё равно получаются скоррелированными. Как минимум, можно сэмплировать для обучения сеток разные мини-батчи из реплей-буфера, если он общий.

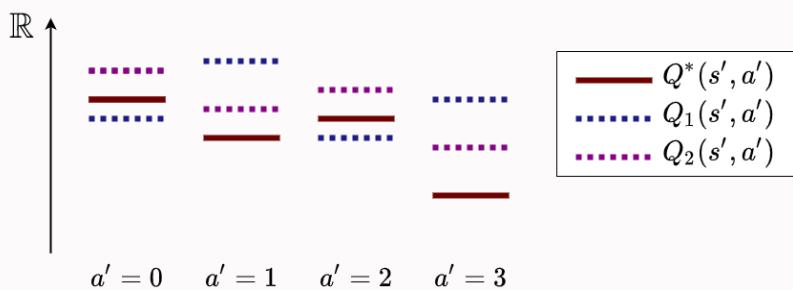
Интуиция, почему «независимая» Q-функция близнецка используется именно для оценки действия, а не наоборот для выбора: если из-за неудачного градиентного шага наша сетка Q_{θ_1} пошла куда-то не туда, мы не хотим, чтобы плохие значения попадали в её же таргет y_1 . Плохие действия в таргет пусть попадают: пессимистичная оценка здесь предпочтительнее.



На практике таргет-сети в такой модели всё равно используют, и в формулах целевой переменной всюду используются именно «замороженные» $Q_{\theta_1^-}$ и $Q_{\theta_2^-}$. Это ещё больше противодействует цепным реакциям.

Без них всё равно может быть такое, что сломанная Q_{θ_1} поломает y_2 , та поломает Q_{θ_2} , та поломает y_1 , а та в свою очередь продолжит ломать Q_{θ_1} , хотя, конечно, такая «цепь» менее вероятна, чем в обычном DQN.

Пример 68: На картинке для каждого из четырёх действий указаны значения идеальной $Q^*(s', a')$ и две аппроксимации Q_1, Q_2 . Каждая аппроксимация условно выдаёт значение с погрешностью, которая в среднем равна нулю (вероятности завышенной оценки или заниженной оценки равны).



Видно, что обе аппроксимации оценивают максимум по действиям завыщенно. Но если одна из аппроксимаций выберет действие (то a' , на котором достигается её максимум), а другая аппроксимация выдаст значение в выбранном индексе, то получится более близкое к адекватной оценки истинного максимума значение.

³Когда эту идею предлагали в 2010-ом году в рамках классического RL (тогда нейронки ещё не вставили), то назвали её Double Q-learning, но сейчас под Double DQN подразумевается алгоритм 2015-го года (см. раздел 4.2.3), а этот трюк иногда называется «близнецами» (Twin). Правда, в последнее время из-за алгоритма Twin Delayed DDPG (см. раздел 6.1.6) под словом Twin понимается снова не эта формула, и с названиями есть небольшая путаница...

4.2.2. 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, которую называют twin-оценкой (или ещё clipped double оценкой):

$$y_1(\mathbb{T}) := r + \gamma \min_{i=1,2} Q_{\theta_i}(s', \operatorname{argmax}_{a'} Q_{\theta_2}(s', a'))$$

$$y_2(\mathbb{T}) := r + \gamma \min_{i=1,2} Q_{\theta_i}(s', \operatorname{argmax}_{a'} Q_{\theta_1}(s', a'))$$

Это очень забавная формула, поскольку она говорит бороться с проблемой «клип клином»: зная, что взятие максимума по аппроксимациям приводит к завышению, мы вводим в оценку искусственное занижение, добавляя в формулу минимум по аппроксимациям! По сути, это формула «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 «блзинеца» 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:

$$y(\mathbb{T}) := r + \gamma Q_{\theta^-}(s', \operatorname{argmax}_{a'} Q_{\theta}(s', a'))$$

Хотя понятно, что таргет-сеть и текущая сетка очень похожи, такое изменение формулы целевой переменной всё равно «избавляет» нас от взятия оператора максимума; эмпирически оказывается, что такая декорреляция действительно помогает стабилизации процесса. При этом, в отличие от предыдущих вариантов, такое изменение бесплатно: не требует обучения второй нейросети.



Если обучение второй Q-сети тяжеловесно, то рекомендуется использовать формулу Double DQN. Если же речь идёт об обучении маленьких полно связанных нейросеток, то вероятно, обучение «ансамбля» хотя бы из двух Q-функций по крайней мере с общего буфера не должно быть особо дорогим, и тогда имеет смысл пользоваться Twin-оценкой из раздела 4.2.2.

4.2.4. Dueling DQN

Рассмотрим ещё одну очевидную беду Q-обучения на примере.

Пример 69: 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 (3.15) — 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. При этом при получении, скажем, негативной информации о ценности одного из действий, должна понижаться в том числе и оценка V-функции. **Дуэльная** (dueling) архитектура — это модификация вычислительного графа нашей модели с параметрами θ , в которой на выходе предлагается иметь две головы, V-функцию $V(s) \approx V^*(s)$ и Advantage-функцию $A(s, a) \approx A^*(s, a)$:

$$Q_\theta(s, a) := V_\theta(s) + A_\theta(s, a) \quad (4.3)$$

Это интересная идея решить проблему просто сменой архитектуры вычислительного графа: при апдейте значения для одной пары s, a неизбежно поменяется значение $V_\theta(s)$ ценности всего состояния, которое общее для всех действий. Мы таким образом вводим следующий «прайор»: ценности действий в одном состоянии всё-таки связаны между собой, и если одно действие в состоянии плохое, то вероятно, что и остальные тоже плохие («само состояние плохое»).

Здесь есть подвох: если $V^*(s)$ — это произвольный скаляр, то $A^*(s, a)$ — не произвольный. Действительно: мы моделируем $|\mathcal{A}|$ чисел, выдавая почему-то $|\mathcal{A}| + 1$ число: то есть почему-то вводим «лишнюю» степень свободы». Вообще-то, Advantage-функция не является произвольной функцией и обязана подчиняться (22). Для оптимальных стратегий в предположении жадности нашей стратегии это утверждение вырождается в следующее свойство:

Утверждение 38:

$$\forall s: \max_a A^*(s, a) = 0$$

Доказательство.

$$\max_a A^*(s, a) = \max_a Q^*(s, a) - V^*(s) = \{\text{связь } V^*Q^* \text{ (3.15)}\} = 0 \quad \blacksquare$$

Это условие можно легко учесть, вычитя максимум в формуле (4.3):

$$Q_\theta(s, a) := V_\theta(s) + A_\theta(s, a) - \max_{\hat{a}} A_\theta(s, \hat{a}) \quad (4.4)$$

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 учатся 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 (4.4) 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 $Q_\theta(s, a)$.

Нюанс: авторы статьи эмпирически обнаружили, что замена максимума на среднее даёт чуть лучшие результаты. Вероятно, в реплей буфере очень часто встречаются пары s, a такие, что a — наилучшее действие в этом состоянии по мнению текущей модели (а то есть $a = \operatorname{argmax}_a A_\theta(s, a)$), и поэтому градиент $A_\theta(s, a) - \max_{\hat{a}} A_\theta(s, \hat{a})$ часто зануляется. В результате, на текущий день под дуэльнной архитектурой понимают альтернативную формулу:

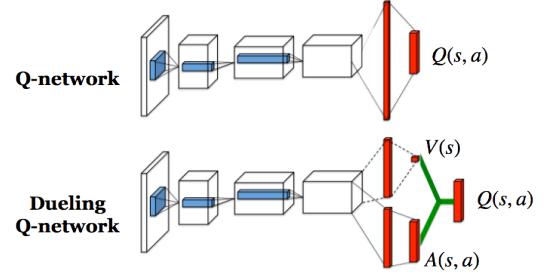
$$Q_\theta(s, a) := V_\theta(s) + A_\theta(s, a) - \frac{1}{|\mathcal{A}|} \sum_{\hat{a}} A_\theta(s, \hat{a}) \quad (4.5)$$

4.2.5. Шумные сети (Noisy Nets)

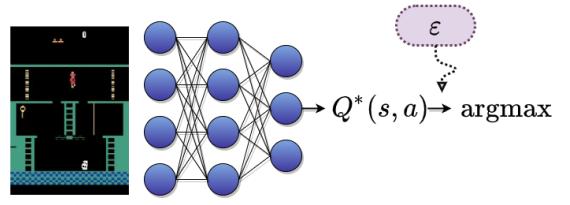
По дефолту, алгоритмы на основе DQN решают дилемму исследования-использования при помощи прimitивной ϵ -жадной стратегии взаимодействия со средой. Этот бэйзлайн-подход плох примерно всем, в первую очередь тем, что крайне чувствителен к гиперпараметрам: для ϵ обязательно нужно составлять какое-нибудь расписание, чтобы в начале обучения он был побольше, а потом постепенно затухал, и откуда брать это расписание — непонятно. При этом слишком большие значения шума существенно замедляют обучение, заставляя агента вести себя случайно, а раннее затухание приведёт к застреванию алгоритма в каком-нибудь локальном оптимуме (агент будет биться головой об стенку, не пробуя её обойти).



Чтобы понять, что случилось именно это, можно посмотреть игры агента: если, например, Марио всё время доходит до середины первого уровня и прыгает в одну и ту же яму, то у него просто нет положительного опыта перепрыгивания этой ямы, и поэтому он не знает, что можно набрать больше награды.



Ключевая причина, почему ϵ -жадная стратегия примитивна, 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, 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 $a = \text{argmax } Q_\theta(s, a, \varepsilon)$. При этом влияние шума на принятное решение зависит от поданного в модель входного 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 несмещённо 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: если модель часто видит какое-то s и функция потерь говорит, что на этом состоянии нужно выдавать, скажем, низкое значение, модель будет учиться при любых сэмплах ε выдавать указанное низкое значение. Для этого модели будет удобно уменьшать дисперсию впрыскиваемого шума и больше опираться на те нейронные связи, которые мало зашумлены. Если же в сеть поступают противоречивые сигналы о паре s, a , или это какое-то новое s , которого модель ещё не видела, выходное значение модели будет, интуитивно, сильно зашумлено, и часто аргмаксум будет достигаться именно на нём.

Ещё одно ключевое преимущество идеи в том, что в этом подходе отсутствуют гиперпараметры.

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.

Заметим, что таргет $y(T)$, который мы генерируем для каждого перехода T из батча, формально теперь тоже должен вычисляться как мат.ожидание по шуму:

$$y(T) := \mathbb{E}_\varepsilon \left[r + \gamma \max_{a'} Q_\theta(s', a', \varepsilon) \right]$$

Опять же, мат.ожидание несмещённо оценивается по Монте-Карло, однако с целью декорреляции полезно использовать в качестве ε другие сэмплы, нежели используемые при вычислении лосса. Считается, что подобное «зашумление» целевой переменной в DQN может даже пойти на пользу.

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. К сожалению, даже при таком ходе время работы алгоритма заметно увеличивается, поскольку проходов через нейросеть в DQN нужно делать очень много.



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.

4.2.6. Приоритизированный реплей (Prioritized DQN)

Off-policy алгоритмы позволяют хранить и переиспользовать весь накопленный опыт. Однако, интуитивно ясно, что встречающиеся переходы существенно различаются по важности. Зачастую большая часть буфера, особенно **поначалу обучения**, состоит из записей изучения агентом ближайшей стены, 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 **при мерно 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:

$$\rho(\mathbb{T}) := (y(\mathbb{T}) - Q_\theta(s, a))^2 = \text{Loss}(y(\mathbb{T}), Q_\theta(s, a)) \quad (4.6)$$

Сэмплирование переходов из буфера происходит по следующему правилу:

$$P(\mathbb{T}) \propto \rho(\mathbb{T})^\alpha$$

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.

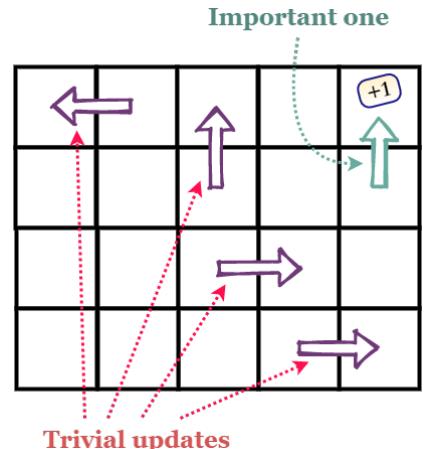
Техническая проблема идеи (4.6) заключается в том, что после каждого обновления весов сети θ приоритеты переходов меняются для всего буфера (состоящего обычно из миллионов переходов). Пересчитывать все приоритеты, конечно же, непрактично, и необходимо ввести некоторые упрощения. Например, можно обновлять 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.



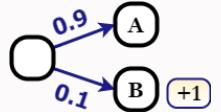
Новые переходы добавляются в буфер с наивысшим приоритетом $\max_{\mathbb{T}} \rho(\mathbb{T})$, который можно поддерживать за константу, или вычислять текущий приоритет, дополнительно рассчитывая (4.6) для онлайн-сэмплов.

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 70: Пусть для данной пары s, a с вероятностью 0.9 мы попадаем в $s' = A$, а с вероятностью 0.1 мы попадаем в $s' = B$. В условно бесконечном буфере для этой пары s, a среди **каждых 10 сэмплов** будет 1 сэмпл с $s' = B$ и 9 сэмплов с $s' = A$, 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 мы оцениваем $V(A) = 0, V(B) = 1$, и уже даже правильно выучили среднее 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, $s' = B$ будет появляться в засэмплированных переходах чаще, 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 36: 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.

$$\begin{aligned}\mathbb{E}_{\mathbb{T} \sim \text{Uniform}} \text{Loss}(\mathbb{T}) &= \sum_{i=1}^M \frac{1}{M} \text{Loss}(\mathbb{T}_i) = \\ &= \sum_{i=1}^M \mathbf{P}(\mathbb{T}_i) \frac{1}{M \mathbf{P}(\mathbb{T}_i)} \text{Loss}(\mathbb{T}_i) = \\ &= \mathbb{E}_{\mathbb{T} \sim \mathbf{P}(\mathbb{T})} \frac{1}{M \mathbf{P}(\mathbb{T})} \text{Loss}(\mathbb{T}),\end{aligned}$$

что с точностью до константы $\frac{1}{M}$ есть перевзвешивание функции потерь. ■

Importance sampling подразумевает, что мы берём «интересные» переходы, но делаем по ним меньшие шаги (вес 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

$$w(\mathbb{T}) := \frac{1}{\mathbf{P}(\mathbb{T})^{\beta(t)}}$$

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

Уже упоминалось, что DQN из-за одношаговых целевых переменных страдает от проблемы отложенного сигнала и сопряжённой с ней в контексте нейросетевой аппроксимации проблемы накапливающейся ошибки. Эта проблема фундаментальна для off-policy подхода: в разделе 3.5 про bias-variance trade-off упоминалось, что разрешать дилемму смещения-разброса (то есть «проводить умный credit assignment») мы можем только в on-policy режиме.

Многошаговый (multi-step) DQN — теоретически некорректная эвристика для занижения этого эффекта. Грубо говоря, нам очень хочется распространять за одну итерацию награду сразу на несколько шагов вперёд, то есть решать многошаговые уравнения Беллмана (3.25). Мы как бы и можем уравнение оптимальности многошаговое выписать...

Утверждение 39 — N -шаговое уравнение оптимальности Беллмана:

$$Q^*(s_0, a_0) = \mathbb{E}_{\pi_{:N} \sim \pi^* | s_0, a_0} \left[\sum_{t=0}^{N-1} \gamma^t r_t + \gamma^N \mathbb{E}_{s_N} \max_{a_N} Q^*(s_N, a_N) \right] \quad (4.7)$$

Что мы можем сделать? Мы можем прикинуться, будто решаем многошаговые уравнения Беллмана, задав целевую переменную следующим образом:

$$y(s_0, a_0) := \sum_{t=0}^{N-1} \gamma^t r_t + \gamma^N \max_{a_N} Q_\theta(s_N, a_N) \quad (4.8)$$

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:

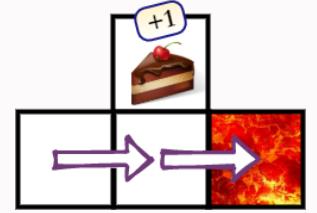
$$\mathbb{T} := \left(s, a, \sum_{n=0}^{N-1} \gamma^n r^{(n)}, s^{(N)}, \text{done} \right)$$

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 роллаута⁴. 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 (4.7). 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^*), 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 μ , когда-то породившая данный роллаут, и только потом сберусь вести себя оптимально? Причём из-за нашего желания делать так в off-policy режиме μ для каждого перехода своё, то есть схеме Generalized Policy Iteration (алг. 9) это не соответствует: в ней мы всегда должны оценивать именно текущую стратегию π , а текущей стратегией в DQN является $\pi(s) = \operatorname{argmax}_a Q_\theta(s, a)$.

Пример 71: Пример, когда многошаговая оценка приводит к некорректным действиям. 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 сколь 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» (когда $\max_a Q^*(s, a) - \min_a Q^*(s, a)$ мало, то есть неоптимальное поведение в течение одного шага не приводит к сильно меньшей награде, чем оптимальное).

Пример 72: Пример среди без «критических решений»: вы робот, который хочет добраться до соседней комнаты. Действия вверх-вниз-вправо-влево чуть-чуть сдвигают робота в пространстве. Тогда «вести себя как угодно» в течение $N - 1$ шагов и потом отправиться кратчайшим маршрутом до соседней комнаты приносит

⁴естественно, алгоритм должен рассматривать все N -шаговые роллауты, включая те, которые привели к завершению эпизода 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 .

практически столько же награды, сколько и сразу отправиться кратчайшим маршрутом до соседней комнаты. Поэтому в таких ситуациях использование относительно большого N (5-10) может помочь, хоть алгоритм и может полностью дестабилизироваться (процедура некорректна).

4.2.8. Retrace

Как мы обсуждали в разделе 3.5.7, теоретически корректным способом обучаться в off-policy с многошаговых оценок является использование Retrace оценки. Конечно, она может на практике схлопываться в одношаговые обновления, но по крайней мере гарантирует, что алгоритм не ломается; и важно, что если записанные в засемплированном из буфера роллауте действия достаточно вероятны для оцениваемой политики, то оценка получается достаточно длинной.

Конечно, сложно говорить про «достаточно вероятны», когда оцениваемая политика детерминирована. Поэтому в практическом алгоритме Retrace предлагается перейти от моделирования Q-learning к моделированию SARSA (см. раздел 3.4.8): то есть, считать целевой политикой $\pi(a | s)$ ε -жадную стратегию по отношению к текущей модели Q-функции. Преимущество в том, что это делает стратегию стохастичной, и любые действия в буфере не приведут к занулению следа и полному схлопыванию в одношаговую оценку.

В буфере также нужно сохранять вероятности выбора сохранённых действий $\mu(a | s)$ в момент сбора данных (для ε -жадных стратегий эти значения всё время будут или $\frac{\varepsilon}{|\mathcal{A}|}$, или $1 - \varepsilon + \frac{\varepsilon}{|\mathcal{A}|}$, где ε — параметр эксплорейшна на момент сбора перехода). Вместо отдельных переходов теперь хранятся роллауты — фрагменты траекторий некоторой длины N .



Если в DQN обучение проводилось на мини-батчах из, скажем, 64 переходов, то в Retrace (при том же масштабе задачи) нужно засемплировать для одного мини-батча 4 роллаута длины $N = 16$. Такой выбор позволит надеяться на получение оценок длины вплоть до 16-шаговой (что в Retrace будет достигнуто, если политика сбора данных совпадёт с оцениваемой политикой на оцениваемом роллауте). Важно помнить про проблему декорреляции: нужно, чтобы в мини-батче должны оказаться разнообразные примеры, поэтому нельзя, например, взять только один роллаут длины 64.

Далее для каждого засемплированного из буфера роллаута $s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_N$ мы сначала для каждой пары s_t, a_t считаем, используя формулы из теории Retrace, следующие вспомогательные величины: значение коэффициента затухания следа c_t по формуле (3.58) (коэффициент λ обычно полагают равным единице) и значение одношаговой ошибки $\Psi_{(1)}(s_t, a_t)$ по формуле (3.56):

$$c_t := \min \left(1, \frac{\pi(a_t | s_t)}{\mu(a_t | s_t)} \right)$$

$$\Psi_{(1)}(s_t, a_t) = r_t + \gamma \mathbb{E}_{\hat{a}_{t+1} \sim \pi} Q_{\theta^-}(s_{t+1}, \hat{a}_{t+1}) - Q_{\theta^-}(s_t, a_t)$$

Всюду, где используется π , используется ε -жадная стратегия по отношению к таргет-сети (хотя, если использовать идею Double DQN из раздела 4.2.3, то как раз во всех местах, где используется π — оцениваемая стратегия — имеет смысл использовать свежую версию Q-функции). В частности, матожидание по π можно посчитать явно:

$$\mathbb{E}_{\hat{a} \sim \pi} Q_{\theta^-}(s, \hat{a}) = (1 - \varepsilon) \max_{\hat{a}} Q_{\theta^-}(s, \hat{a}) + \frac{\varepsilon}{|\mathcal{A}|} \sum_{\hat{a}} Q_{\theta^-}(s, \hat{a})$$

После этого в Retrace для одной пары s_t, a_t все будущие одношаговые ошибки нужно просуммировать (воспользуемся индексом \hat{t} для обозначения этого перебора), но заглядывание на каждый следующий i -ый шаг в будущее обязывает нас потушить след в c_i раз:

$$\Psi^{\text{retrace}}(s_t, a_t) := \sum_{\hat{t} \geq t}^N \gamma^{\hat{t}-t} \left(\prod_{i=t+1}^{\hat{t}} c_i \right) \Psi_{(1)}(s_{\hat{t}}, a_{\hat{t}})$$

Заметим, что в этой формуле внешняя сумма по \hat{t} идёт не до бесконечности, как в теории Retrace, а до N , до конца роллаута. После этого считаем, что след зануляется: это корректно, хотя иногда можно и потерять возможность получить более длинную оценку.

Далее в табличном методе мы бы провели обновление по формуле

$$Q(s, a) \leftarrow Q(s, a) + \alpha \Psi^{\text{retrace}}(s, a),$$

то есть воспользовались бы $\Psi^{\text{retrace}}(s, a)$ как градиентом. Другими словами, оценка указывает, нужно ли увеличивать выход модели для рассматриваемой пары s, a или уменьшать. Чтобы получить задачу регрессии, целевая переменная строится по формуле

$$y(s_t, a_t) := \Psi^{\text{retrace}}(s_t, a_t) + Q_{\theta}(s_t, a_t)$$

и дальше оптимизируется MSE, игнорируя зависимость y от θ :

$$(y(s_t, a_t) - Q_\theta(s_t, a_t))^2 \rightarrow \min_{\theta}$$

Тогда градиент функции потерь по θ для одного примера равен:

$$\nabla_{\theta} \frac{1}{2} (y(s, a) - Q_{\theta}(s, a))^2 = (y(s, a) - Q_{\theta}(s, a)) \nabla_{\theta} Q_{\theta}(s, a) = \Psi^{\text{retrace}}(s, a) \nabla_{\theta} Q_{\theta}(s, a)$$

Это полностью аналогично градиенту обычного DQN (4.2), только там оценка $\Psi(s, a) = r + \gamma \max_{a'} Q_{\theta-}(s, a) - Q_{\theta}(s, a)$ была одношаговой, а здесь мы заглядываем настолько максимально далеко вперёд, насколько возможно (в силу использования $\lambda = 1$).

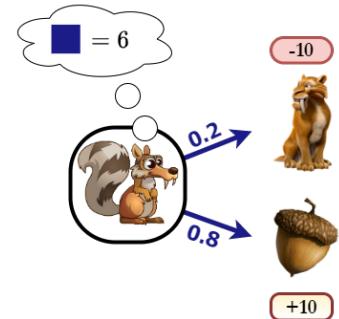


В большинстве последних алгоритмов на основе DQN, таких как Agent57, используются формулы Retrace. Они позволяют максимально возможным образом побороться с ключевыми фундаментальными проблемами off-policy подхода, вытекающими из одношаговых целевых переменных, когда из недостатков можно выделить, пожалуй, лишь громоздкость формул.

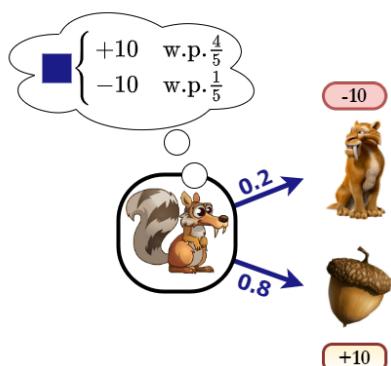
§4.3. Distributional RL

4.3.1. Идея Distributional подхода

Задача RL такова, что в среде содержится в том числе неподконтрольная агента стохастика: **алеаторическая неопределенность** (aleatoric uncertainty)⁵. Агент, предсказывающий, что он получит в будущем в среднем суммарную награду 6, на самом деле может получить, например, только -10 или 10, просто последний исход случится с вероятностью 0.8, а первый — 0.2. Помимо прочего, это означает, что часто агенту приходится рисковать: например, теоретически возможна ситуация, когда агент с малой вероятностью получает гигантскую награду, и тогда оптимальный агент на практике будет постоянно получать, например, какой-то штраф, компенсирующийся редко выпадающими мегаудачами. Вся эта информация заложена в распределении награды $R(\mathcal{T})$ (1.4) как случайной величины.



В Distributional-подходе предлагается учить не среднее будущей награды, а всё распределение будущей награды как случайной величины. Складывается эта неопределенность как из подконтрольной агенту стохастики — его собственных будущих выборов действий — так и неподконтрольной, переходов (и награды, если рассматривается формализм со случайной функцией награды). Среднее есть лишь одна из статистик этого распределения.



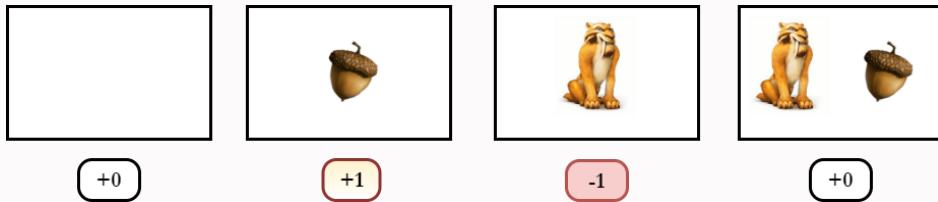
Здесь стоит заранее оговориться о противоречиях, связанных с этой идеей. Обсуждение этой темы в первую очередь мотивировано эмпирическим превосходством Distributional-подхода по сравнению с алгоритмами, учащими только среднее, однако с теоретической точки зрения ясного обоснования этого эффекта нет. Даже наоборот: мы далее встретим теоретические результаты, показывающие эквивалентность Distributional-алгоритмов с обычными в рамках табличного сеттинга.

Одно гипотетическое объяснение преимущества distributional-подхода в нейросетевом сеттинге, когда будущая награда предсказывается сложной параметрической моделью, может быть следующая: обучая модель предсказывать не только среднее награды, но и другие величины (другие статистики), сильно связанные по смыслу со средней наградой, в модель отправляются более информативные градиенты. Например, если с вероятностью 0.01 во входном изображении появляется грибочек, намекающий на приближение вкусного +100, обновление среднего будет, абстрактно говоря, проходить с учётом малой вероятности явления; когда обучение 1%-го квантиля наоборот крайне интересуется именно хвостом распределения и поможет быстрее выучить, например, фильтр для детекции грибочка в первых свёрточных слоях, который, в свою очередь, ускорит и более точное вычисление среднего, для которого распознавание грибочка на самом деле было существенным. Вот ещё один пример, почему это может быть хорошо:

Пример 73: Допустим, состояние описывается картинкой. Если вы видите на картинке орешек, то получите +1. Если тигра — то -1. Если и тигра, и орешек, то вам может повезти, а может не повезти, и вы с вероятностью 0.5

⁵ которую не стоит путать с **эпистемической неопределенностью** (epistemic uncertainty), или байесовской неопределенностью, связанной с нашим субъективным незнанием о том, как устроена, например, функция переходов и награды на самом деле, связанной с нашим осознанием неточности прогнозов.

получите +1, а с вероятностью 0.5 получите -1. В такой ситуации модель, предсказывающая среднюю будущую награду, должна выдавать 0 — тоже самое значение, что и для пустой картинки, где никаких объектов нет.



Было бы здорово в таких ситуациях в последнем слое модели увидеть что-то вроде признаков, отвечающих на вопрос «есть ли на картинке орешек?» или «есть ли на картинке тигр?». Конечно, такой разметки нам никто не предоставит, и обучать такой слой напрямую не получится. Так вот, забавное наблюдение заключается в том, что распределение награды в таких ситуациях содержит информацию о том, «какие объекты» есть на картинке: если целевая переменная для пустой картинке будет «0 с вероятностью 1», а для картинки с тигром и орешком « ± 1 с вероятностями 0.5», то модель научится различать такие ситуации, и скорее всего будет быстрее обучаться распознавать тигров и орешки.



Ещё одним способом придумать для модели целевую переменную, предсказание которой сильно связано со средней наградой, является использование нескольких дискаунт-факторов $0 < \gamma_1 < \gamma_2 < \dots < \gamma_G < \gamma$, где γ — коэффициент, для которого решается задача. Иначе говоря, нейросетевая модель выдаёт для каждой пары состояния-действие не одно, а $G+1$ число, соответствующее Q -функции для соответствующего коэффициента дисконтирования:

$$Q^\pi(s, a, g) := \mathbb{E}_{\mathcal{T}|s_0=s, a_0=a} \sum_{t \geq 0} \gamma_g^t r_t, \quad g \in \{1 \dots G\}$$

Иными словами, мы дополнительно учим, какую награду мы получим в самое ближайшее время, более близкое, чем реальный рассматриваемый горизонт. Эти дополнительные величины самой стратегией использоваться не будут (взаимодействие со средой использует только значения Q для настоящего коэффициента γ), однако их обучение поможет «ускорить» обучение для настоящей Q с самым большим горизонтом. Мы вернёмся к обобщению этой идеи, когда будем обсуждать multi-task RL в разделе 8.3.

4.3.2. Z-функция

Определение 63: Для данного MDP *оценочной функцией в distributional форме* (distributional state-action value function) для стратегии π называется случайная величина, обусловленная на пару состояния-действие s, a и определяющаяся как reward-to-go для такого старта:

$$\mathcal{Z}^\pi(s, a) \stackrel{\text{c.d.f.}}{=} R(\mathcal{T}), \quad \mathcal{T} \sim \pi \mid s_0 = s, a_0 = a$$

Здесь читателю предлагается заварить себе кофе на том факте, что введённая так называемая Z-функция⁶ является для каждой пары s, a случайной величиной. Во-первых, это скалярная случайная величина, соответственно, она задаётся некоторым распределением на \mathbb{R} , во-вторых, как и для любой случайной величины, существенно, на что она обуславливается. Запись $\mathcal{Z}^\pi(s, a)$ предполагает, что мы сидим в состоянии s и выполнили действие a , после чего «бросаем кости» для сэмплирования случайной величины; нас, вообще говоря, будет интересовать её *функция распределения* (cumulative distribution function, c. d. f.):

$$F_{\mathcal{Z}^\pi(s, a)}(x) := \mathbf{P}(\mathcal{Z}^\pi(s, a) \leq x)$$

Нам будет удобнее работать с ними, а не с плотностями, поскольку зачастую распределение $\mathcal{Z}^\pi(s, a)$ — дискретное или вообще вырожденное (принимающее с вероятностью 1 только какое-то одно значение). Таким образом, пространство всевозможных Z-функций имеет такой вид:

$$\mathcal{Z}^\pi(s, a) \in \mathcal{S} \times \mathcal{A} \rightarrow \mathbf{P}(\mathbb{R}), \tag{4.9}$$

где $\mathbf{P}(\mathbb{R})$ — пространство скалярных случайных величин.

Надпись c.d.f. над равенством здесь и далее означает, что слева и справа стоят случайные величины. Очень важно, что случайные величины справа и слева в подобных равенствах обусловлены на одну и ту же информацию: справа, как и слева, стоит случайная величина, обусловленная на s, a . Случайная величина здесь задана процессом генерации: сначала генерируется случайная траектория \mathcal{T} при заданных $s_0 = s, a_0 = a$ (это по определению MDP эквивалентно последовательному сэмплированию $s_1, a_1, s_2 \dots$), затем от этой случайной величины считается детерминированная функция $R(\mathcal{T})$. Запись $\stackrel{\text{c.d.f.}}{=}$ означает, что $\mathcal{Z}^\pi(s, a)$ имеет в точности то же распределение, что и случайная величина, генерируемая процессом, описанном справа.

По определению:

⁶слово «функция» здесь, конечно, не очень удачно, однако автор данного текста не справился с нахождением альтернатив. «Величина» ещё можно было бы.

Утверждение 40:

$$Q^\pi(s, a) = \mathbb{E} Z^\pi(s, a)$$

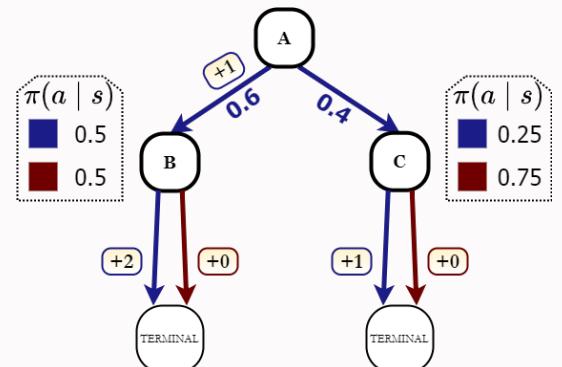
(4.10)

Утверждение 41: В терминальных состояниях для всех действий $Z^\pi(s, a)$ есть вырожденная случайная величина, с вероятностью 1 принимающая значение ноль.

Пример 74: Допустим, мы сидим в состоянии и выполнили действие $\boxed{\text{■}}$. Как будет выглядеть $Z^\pi(s, \boxed{\text{■}})$ для MDP и стратегии π с рисунка, $\gamma = 0.5$?

Нас ждёт два источника случайности: сначала среда кинет нас или в состояние B, или в состояние C, затем мы случайно будем определять своё следующее действие. Всего нас ждёт 4 возможных исхода. Для каждого мы можем посчитать его вероятность и получаемый reward-to-go. Итого $Z^\pi(s, \boxed{\text{■}})$ – дискретное распределение с 4 исходами:

Исход s'	Исход a'	Вероятность	reward-to-go
B	■	0.3	$1 + 2\gamma$
B	■	0.3	1
C	■	0.1	γ
C	■	0.3	0



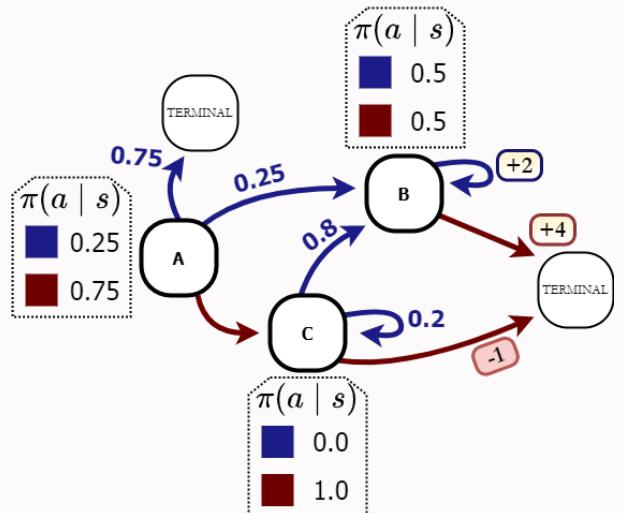
Пример 75: Посчитаем Z-функцию для MDP и стратегии π с рисунка, $\gamma = 0.8$.

Начнём с состояния B: если агент выбирает действие $\boxed{\text{■}}$, то он получает $+4$, и эпизод заканчивается. Значит, $Z^\pi(s = B, \boxed{\text{■}})$ всегда принимает значение 4.

Что произойдёт, если он выберет $\boxed{\text{■}}$? Агент точно получит $+2$ и вернётся в состояние B. Вся дальнейшая награда будет дисконтирована на $\gamma = 0.8$. После этого начинается первая стохастика: агент снова будет выбирать действие! С вероятностью 0.5 он выберет $\boxed{\text{■}}$ и получит итоговый reward-to-go $2 + \gamma \cdot 4 = 5.2$ за эпизод. Вот мы нашли часть нашей Z-функции для $s = B, a = \boxed{\text{■}}$: с вероятностью 0.5 исход будет 5.2. Ищем, что соответствует оставшейся вероятностной массе 0.5: мы выберем снова $\boxed{\text{■}}$, получим уже суммарно $2 + \gamma \cdot 2 = 3.6$, снова вернёмся в B, снова случится дисконтирование и снова за нами будет выбор. Мы можем найти, что соответствует ещё 0.25 нашей вероятностной массы: $2 + \gamma \cdot 2 + \gamma^2 \cdot 4 = 6.16$. Дальше в этом распределении будет ещё исход с вероятностью $\frac{1}{8}$, затем $\frac{1}{16}$ и так далее: $Z^\pi(s = B, \boxed{\text{■}})$ есть распределение со счётым множеством исходов!

Очевидно, что $Z^\pi(s = A, \boxed{\text{■}})$ и $Z^\pi(s = C, \boxed{\text{■}})$ тоже будут вырожденными: reward-to-go для таких стартов однозначно определён и равен -0.8 и -1 соответственно.

$Z^\pi(s = A, \boxed{\text{■}})$ содержит компоненту «с вероятностью 0.75 мы попадём в терминальное состояние и получим 0». Оставшиеся 0.25 соответствуют попаданию в B и распределяются пополам между выбором следующего действия $\boxed{\text{■}}$ (соответствует награде 3.2) и $\boxed{\text{■}}$ (тогда начнётся та же цепочка исходов, домноженных на $\gamma = 0.8$, что и для $Z^\pi(s = B, \boxed{\text{■}})$). Аналогично можно расписать $Z^\pi(s = C, \boxed{\text{■}})$; как видно, такая «оценочная функция» содержит в себе намного больше информации о будущих наградах.



4.3.3. Distributional-форма уравнения Беллмана

Заметим, что в доказательстве уравнений Беллмана, например, (3.3), мы ссылаемся на то, что для reward-to-go любых траекторий верно рекурсивное соотношение. После этого мы берём по траекториям мат.ожидание слева и справа, получая традиционное уравнение Беллмана. Ясно, что мы могли бы вместо среднего взять любую другую статистику от случайной величины (дисперсию, медианы, другие квантили...), а, вообще говоря, верно совпадение левой и правой части по распределению. Иначе говоря, можно зачеркнуть символ мат.ожидания из уравнения Беллмана для получения более общего утверждения.

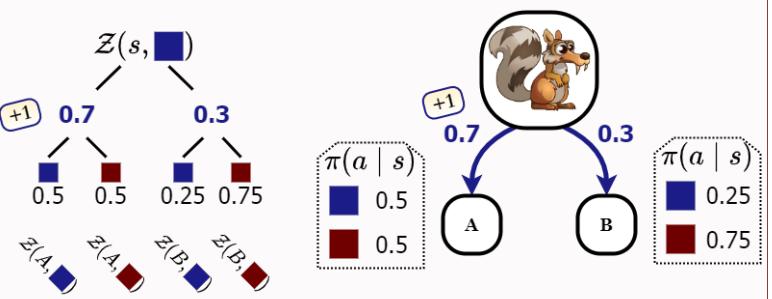
Теорема 37 — Уравнение Беллмана в Distributional-форме:

$$\mathcal{Z}^\pi(s, a) \stackrel{\text{c.d.f.}}{=} r(s, a) + \gamma \mathcal{Z}^\pi(s', a'), \quad s' \sim p(s' | s, a), a' \sim \pi(a' | s') \quad (4.11)$$

Доказательство. Следует из тождества $R(\mathcal{T}) = r + \gamma R(\mathcal{T}_{1:})$ для любых траекторий \mathcal{T} . ■

Читатель подозревается в недоумении от происходящего; остановимся на этом уравнении и обсудим, что тут написано. Во-первых, необходимо пояснить, что данное уравнение есть переформулировка (другая нотация) используемых определений. Reward-to-go $R(\mathcal{T})$ — детерминированная функция от заданной траектории \mathcal{T} , \mathcal{Z}^π — по сути тоже самое, только траектория рассматривается как случайная величина (а параметры s, a указывают на начальные условия генерации траекторий). И слева, и справа в уравнении (4.11) стоят случайные величины, зависящие от s, a ; равенство означает, что они имеют одинаковые распределения. Иными словами, слева и справа записаны два процесса генерации одной и той же случайной величины. Мы можем бросить кость $\mathcal{Z}^\pi(s, a)$ (случайная величина слева), а можем — сначала s' , потом a' , затем $\mathcal{Z}^\pi(s', a')$ и выдать исход⁷ $r(s, a) + \gamma \mathcal{Z}^\pi(s', a')$ (случайная величина справа), и эти две процедуры порождения эквивалентны.

Пример 76: Уравнение Беллмана всё ещё связывает «содержимое» Z-функции через неё же саму, раскрывая дерево на один шаг. Эти уравнения теперь затруднительно выписать аналитически, поскольку теперь «компоненты» распределения $\mathcal{Z}(s, a)$ есть перевзвешанные на вероятности переходов и выборов действий (и подправленные по значению на дисконт фактор и смещённые на награду за шаг) $\mathcal{Z}(s', a')$ для всевозможных s', a' .



Подобные уравнения называются *recursive distributional equations* и рассматриваются математикой в одном из разделов теории вероятности. Нам далее не понадобится какая-то особая теория оттуда, однако для вдохновения рассмотрим каноничный местный пример.

Пример 77: Пусть

$$X_1 \stackrel{\text{c.d.f.}}{=} \frac{X_2}{\sqrt{2}} + \frac{X_3}{\sqrt{2}}$$

где X_1, X_2, X_3 — независимые случайные величины из одного распределения $p(x)$. То есть заданы две процедуры порождения: мы можем взять сэмпл из распределения, а может взять два сэмпла из распределения (независимо), отмасштабировать на корень из двух и сложить. Уравнение заявляет, что эти две процедуры эквивалентны. Вопрос для математики такой: каким могло бы быть $p(x)$? Несколько ответов можно угадать: например, ответом является, детерминированный ноль или $\mathcal{N}(0, \sigma^2)$.

4.3.4. Distributional Policy Evaluation

Будем строить аналог Policy Evaluation для distributional-формы оценочной функции. Иными словами, мы хотим чисто теоретический алгоритм, позволяющий для данного MDP и данной стратегии π посчитать распределение $\mathcal{Z}^\pi(s, a)$. MDP пока считаем полностью известным (распределение $p(s' | s, a)$ считаем данным). Действуем в полной аналогии с обычными уравнениями: начнём с ввода оператора Беллмана.

Определение 64: Для данного MDP и стратегии π будем называть *оператором Беллмана в distributional форме* оператор \mathfrak{B}_D , действующий из пространства Z-функций (4.9) в пространство Z-функций, задающей случайную величину для s, a на выходе оператора как правую часть distributional уравнения Беллмана

⁷если в формализме функция награды также случайна, вместо сдвига на константу здесь было бы сложение двух случайных величин; для расчёта распределения тогда теоретически рассматривалась бы операция свёртки.

(4.11):

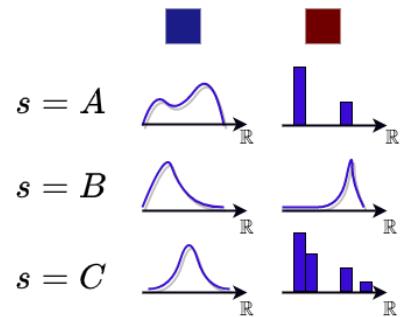
$$[\mathfrak{B}_D \mathcal{Z}] (s, a) \stackrel{\text{c.d.f.}}{=} r(s, a) + \gamma \mathcal{Z}(s', a'), \quad s' \sim p(s' | s, a), a' \sim \pi(a' | s')$$

По определению, истинное \mathcal{Z}^π будет неподвижной точкой такого оператора:

$$\mathcal{Z}^\pi = \mathfrak{B}_D \mathcal{Z}^\pi$$

Нас интересует вопрос о сходимости метода простой итерации. Что это означает? Если на k -ой итерации мы храним большую табличку, где для каждой пары s, a хранится целиком и в точности всё распределение $\mathcal{Z}_k(s, a)$, то на очередном шаге для всех пар s, a происходит обновление

$$\mathcal{Z}_{k+1}(s, a) \stackrel{\text{c.d.f.}}{=} r(s, a) + \gamma \mathcal{Z}_k(s', a'), \quad (4.12)$$



где вероятности случайных величин $s' \sim p(s' | s, a), a' \sim \pi(a' | s')$ мы знаем и потому можем полностью посчитать свёртку распределений $\mathcal{Z}_k(s', a')$ для всевозможных пар следующих s', a' .

Чтобы показать сходимость такой процедуры, хочется в аналогии с традиционным случаем доказать сжимаемость оператора \mathfrak{B}_D . Однако, обсуждение сжимаемости имеет смысл только при заданной метрике, а в данном случае даже для конечных пространств состояний и пространств действий пространство Z-функций бесконечномерно, поскольку бесконечномерно $\mathbf{P}(\mathbb{R})$. Нам нужна метрика в таком пространстве, и, внезапно, от её выбора будет зависеть ответ на вопрос о сжимаемости.

Определение 65: Пусть \mathcal{D} — метрика в пространстве $\mathbf{P}(\mathbb{R})$. Тогда её *максимальной формой* (maximal form) будем называть следующую метрику в пространстве Z-функций:

$$\mathcal{D}^{\max}(\mathcal{Z}_1, \mathcal{Z}_2) := \sup_{s \in \mathcal{S}, a \in \mathcal{A}} \mathcal{D}(\mathcal{Z}_1(s, a), \mathcal{Z}_2(s, a))$$

Теорема 38: Для любой метрики \mathcal{D} в пространстве $\mathbf{P}(\mathbb{R})$ её максимальная форма \mathcal{D}^{\max} есть метрика в пространстве Z-функций.

Доказательство. Проверим неравенство треугольника. Для любых трёх Z-функций $\mathcal{Z}_1, \mathcal{Z}_2, \mathcal{Z}_3$:

$$\begin{aligned} \mathcal{D}^{\max}(\mathcal{Z}_1, \mathcal{Z}_2) &= \sup_{s, a} \mathcal{D}(\mathcal{Z}_1(s, a), \mathcal{Z}_2(s, a)) \leq \\ &\leq \{\text{неравенство треугольника для } \mathcal{D}\} \leq \sup_{s, a} [\mathcal{D}(\mathcal{Z}_1(s, a), \mathcal{Z}_3(s, a)) + \mathcal{D}(\mathcal{Z}_3(s, a), \mathcal{Z}_2(s, a))] \leq \\ &\leq \{\text{свойство максимума}\} \leq \sup_{s, a} \mathcal{D}(\mathcal{Z}_1(s, a), \mathcal{Z}_3(s, a)) + \sup_{s, a} \mathcal{D}(\mathcal{Z}_3(s, a), \mathcal{Z}_2(s, a)) = \\ &= \mathcal{D}^{\max}(\mathcal{Z}_1, \mathcal{Z}_3) + \mathcal{D}^{\max}(\mathcal{Z}_3, \mathcal{Z}_2) \end{aligned}$$

Симметричность, неотрицательность и равенство нулю только при совпадении аргументов проверяется непосредственно. ■

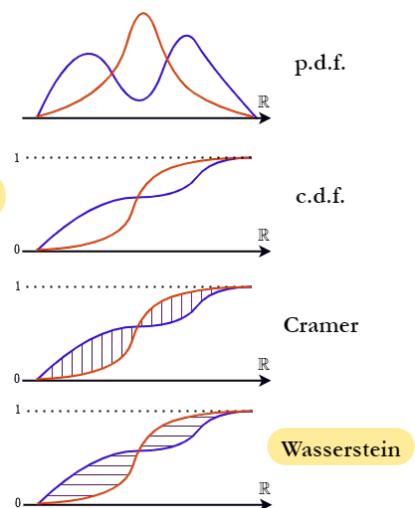
Соответственно, вопрос о выборе метрики в пространстве Z-функций сводится к вопросу о метрике в пространстве скалярных случайных величин. Вопрос, вообще, довольно богатый. Можно пытаться посчитать расстояние между функциями распределения (такова, например, метрика Крамера), а можно — между их обратными функциями:

Определение 66: Для скалярной случайной величины X с функцией распределения $F_X(x)$: $\mathbb{R} \rightarrow [0, 1]$ *квантильной функцией* (inverse distribution function (quantile function)) называется*

$$F_X^{-1}(\omega) := \inf\{x \in \mathbb{R} \mid F_X(x) \geq \omega\}$$

Значение этой функции $F_X^{-1}(\omega)$ в точке $\tau \in (0, 1)$ будем называть τ -квантилем.

* инфинум берётся для однозначности определения в ситуациях, когда в $F_X(x)$ есть плато.



Определение 67: Для $1 \leq p \leq +\infty$ для двух случайных скалярных величин* X, Y с функциями распределения

ления F_X and F_Y соответственно **расстоянием Вассерштайн** (Wasserstein distance) называется

$$\mathcal{W}_p(X, Y) := \left(\int_0^1 \left| F_X^{-1}(\omega) - F_Y^{-1}(\omega) \right|^p d\omega \right)^{\frac{1}{p}} \quad (4.13)$$

$$\mathcal{W}_\infty(X, Y) := \sup_{\omega \in [0, 1]} \left| F_X^{-1}(\omega) - F_Y^{-1}(\omega) \right|$$

* с ограниченными p -ми моментами, что в нашем контексте гарантируется ограниченностью награды (1.3) и ограниченностью домена.

Расстояние Вассерштайн — это довольно глубокая тема в математике; особый интерес представляет случай $p = 1$ (см., например, [википедию](#)).

Теорема 39 — Эквивалентная форма \mathcal{W}_1 :

$$\mathcal{W}_1(X, Y) = \int_{\mathbb{R}} |F_X(x) - F_Y(x)| dx \quad (4.14)$$

Доказательство. При $p = 1$ расстояние Вассерштайн \mathcal{W}_1 есть просто площадь между графиками с.d.f. F_X, F_Y ; тоже самое записано и в этой форме, только интегрирование («суммирование») проводится по оси значений, а не оси квантилей, но как график не повернен, площадь остается той же.

Формальное обоснование даёт теорема из матана о том, что в двойном интеграле можно менять местами интегралы:

$$\int_{\mathbb{R}} |F_X(x) - F_Y(x)| dx = \int_{\mathbb{R}} dx \int_{\min(F_X(x), F_Y(x))}^{\max(F_X(x), F_Y(x))} d\omega$$

Это площадь, если просуммировать вдоль оси x ; давайте попробуем поменять местами интегралы.

Рассмотрим какое-нибудь ω ; есть два симметричных случая. В первом $F_X(x) \leq \omega \leq F_Y(x)$. Ну тогда из второго неравенства $F_Y^{-1}(\omega) \leq x$. Теперь заметим, что $F_X^{-1}(\omega) \geq F_X^{-1}(\hat{\omega})$ для любого $\hat{\omega} \leq \omega$ в силу неубывания F_X^{-1} ; поэтому возьмём в качестве $\hat{\omega} := F_X(x)$, который по первому неравенству не больше ω , и получим

$$F_X^{-1}(\omega) \geq F_X^{-1}(\hat{\omega}) = F_X^{-1}(F_X(x)) = x$$

Мы получили, что $F_X^{-1}(\omega) \leq x \leq F_Y^{-1}(\omega)$. В симметричном случае, когда $F_Y(x) \leq \omega \leq F_X(x)$, мы получили бы аналогично $F_Y^{-1}(\omega) \leq x \leq F_X^{-1}(\omega)$. Таким образом, при данном ω для подсчёта площади переменной x нужно пробежать от $\min(F_X^{-1}(\omega), F_Y^{-1}(\omega))$ до $\max(F_X^{-1}(\omega), F_Y^{-1}(\omega))$; мы получаем равенство

$$\left(\int_{\mathbb{R}} dx \int_{\min(F_X(x), F_Y(x))}^{\max(F_X(x), F_Y(x))} d\omega \right) = \left(\int_0^1 d\omega \int_{\min(F_X^{-1}(\omega), F_Y^{-1}(\omega))}^{\max(F_X^{-1}(\omega), F_Y^{-1}(\omega))} dx \right)$$

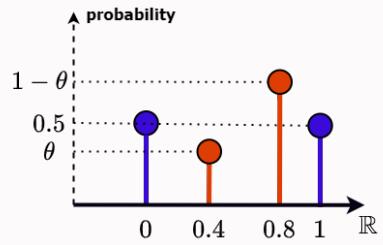
Ну а это в точности равно \mathcal{W}_1 , то есть выинтегрированию модуля разности между F_X^{-1} и F_Y^{-1} . ■

Замечание. Утверждение верно только для $p = 1$: иначе существенно, вдоль какой оси мы растягиваем разность функций возведением в p -ую степень (физический смысл «площади между графиками» пропадает). Поэтому метрика Крамера, которая есть L2-расстояние между c.d.f., не равна \mathcal{W}_2 , которая есть L2-расстояние между квантильными функциями.

Пример 78: Расстояние \mathcal{W}_1 между двумя распределениями неспроста имеет второе название **Earth Moving Distance**. Аналогия такая: нам даны две кучи песка. Объём песка в кучах одинаков, но у них разные конфигурации, они «насыпаны» по-разному. Чтобы перенести каждую песчинку массы m на расстояние x , нам нужно затратить «работы» объёмом mx . Расстояние Вассерштайн \mathcal{W}_1 замеряет, какое минимальное количество работы нужно совершить, чтобы перевести конфигурацию первой кучи песка во вторую кучу; объём песка в каждой куче одинаков. Для дискретных распределений, когда функции распределения (и, соответственно, квантильные функции) — «ступеньки», минимальная работа полностью соответствует площади между функциями распределений.

Посчитаем \mathcal{W}_1 между двумя следующими распределениями. Первое распределение (синие на картинке) — честная монетка с исходами 0 и 1. Вторая случайная величина (красная на картинке) принимает значение 0.4 с вероятностью $\theta < 0.5$ и 0.8 с вероятностью $1 - \theta$. Можно нарисовать функции распределения и посчитать площадь между ними. А можно рассуждать так: давайте «превратим» вторую кучу песка в первую. Посмотрим на песок объёма θ в точке 0.4. Куда его переносить? Наверное, в точку 0, куда его тащить ближе. Перенесли; совершили работы объёмом 0.4θ . Посмотрим на песок объёма $1 - \theta$ в точке 0.8. Его удобно тащить в точку 1, но там для получения первой конфигурации нужно только 0.5 песка. Поэтому 0.5 песка из точки 0.8 мы можем перевести в точку 1, совершив работу $0.2 \cdot 0.5$, а оставшийся объём $1 - \theta - 0.5$ придётся переводить в точку 0, совершая работу $0.8(0.5 - \theta)$. Итого расстояние Вассерштайна равно:

$$\mathcal{W}_1 = 0.4\theta + 0.8(0.5 - \theta) + 0.1$$



Утверждение 42: Максимальная форма метрики Вассерштайна \mathcal{W}_p^{\max} есть метрика в пространстве Z -функций.

Теорема 40: По метрике $\mathcal{W}_p^{\max}(\mathcal{Z}_1, \mathcal{Z}_2)$ оператор \mathfrak{B}_D является сжимающим.

Доказательство для \mathcal{W}_1 . Воспользуемся формой расстояния через c.d.f. (4.14), тогда

$$\mathcal{W}_1^{\max}(\mathfrak{B}_D \mathcal{Z}_1, \mathfrak{B}_D \mathcal{Z}_2) = \sup_{s,a} \int_{\mathbb{R}} |\mathbf{P}(r + \gamma \mathcal{Z}_1(s', a') \leq x) - \mathbf{P}(r + \gamma \mathcal{Z}_2(s', a') \leq x)| dx = (*)$$

где внутри вероятностей s', a' — тоже случайные величины! Ну, сначала заметим, что добавление награды не изменяет значение интеграла. Давайте сделаем такую замену: $\hat{x} = \frac{x-r}{\gamma}$. Получаем:

$$(*) = \sup_{s,a} \gamma \int_{\mathbb{R}} |\mathbf{P}(\mathcal{Z}_1(s', a') \leq \hat{x}) - \mathbf{P}(\mathcal{Z}_2(s', a') \leq \hat{x})| d\hat{x} = (**)$$

Осталось справиться со случайностью s', a' ; к счастью, для функций распределений это несложно. Пусть s, a — фиксированы, тогда просто по формуле полной вероятности:

$$\mathbf{P}(\mathcal{Z}(s', a') \leq \hat{x}) = \int_s \int_{\mathcal{A}} p(s' | s, a) \pi(a' | s') F_{\mathcal{Z}(s', a')}(\hat{x}) ds' da',$$

где $F_{\mathcal{Z}(s', a')}(\hat{x})$ — вероятность, что $\mathcal{Z}(s', a')$ не превзойдёт \hat{x} при фиксированных s', a' . Подставляем:

$$\begin{aligned} (***) &= \gamma \sup_{s,a} \int_{\mathbb{R}} \left| \int_s \int_{\mathcal{A}} p(s' | s, a) \pi(a' | s') (F_{\mathcal{Z}_1(s', a')} - F_{\mathcal{Z}_2(s', a')}) ds' da' \right| d\hat{x} \leq \\ &\leq \{ \text{наше любимое } \mathbb{E}_x f(x) \leq \max_x f(x) \} \leq \gamma \sup_{s,a} \mathcal{W}_1^{\max}(\mathcal{Z}_1, \mathcal{Z}_2) \end{aligned}$$

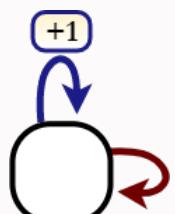
Это значит, что для систем уравнений (4.11) выполняется теорема Банаха 3.3.1!

Утверждение 43: Существует единственная функция $\mathcal{S} \times \mathcal{A} \rightarrow \mathbf{P}(\mathbb{R})$, являющаяся решением уравнений (4.11), и метод простой итерации сходится к ней из любого начального приближения по метрике Вассерштайна. ■

Пример 79: Попробуем найти \mathcal{Z}^π для случайной π (выбирающей из двух действий всегда равновероятно) для MDP с рисунка и $\gamma = 0.5$.

Сначала попробуем понять, в каких границах может лежать наша награда за весь эпизод. Если, например, мы всё время выбираем $\boxed{\text{■}}$, то получим в итоге ноль; меньше, понятно, получить нельзя. Если же мы всё время выбираем $\boxed{\text{□}}$, то получим в итоге $1 + \gamma + \gamma^2 + \dots + \frac{1}{1-\gamma} = 2$. Значит, вероятные исходы размазаны на отрезке $[0, 2]$.

Попробуем посмотреть на $\mathcal{Z}^\pi(\boxed{\text{■}})$. По определению, мы предполагаем, что на первом шаге выбирается действие $\boxed{\text{■}}$, и значит, на первом шаге мы гарантированно получим $+0$. Тогда, проводя аналогичные рассуждения, можно заключить, что дальнейшая возможная



награда лежит в отрезке $[0, 1]$. Но что именно это за распределение? Можно рассмотреть распределение случайной величины $\sum_{t=0}^T \gamma^t r_t | a_0 = \square$ не при $T = +\infty$, а при меньших T . Например, при $T = 1$ мы получим $+0$, затем в качестве r_1 с равными вероятностями получим $+0$ или $+\frac{1}{2}$. Получится равновероятное распределение с исходами $0, +\frac{1}{2}$. При $T = 2$ мы получим уже равновероятное распределение с исходами $0, +\frac{1}{4}, +\frac{1}{2}, +\frac{3}{4}$. Продолжая рассуждение дальше, можно увидеть, что при $T \rightarrow +\infty$ распределение продолжает равномерно размазывать вероятности по $[0, 1]$. Видимо, в пределе получится просто равномерное распределение на $[0, 1]$. Как можно строго доказать, что это правильный ответ?

Попробуем подставить в уравнения Беллмана (4.11) в качестве $\mathcal{Z}^\pi(\square)$ равномерное распределение на отрезке $[0, 1]$, а в качестве $\mathcal{Z}^\pi(\blacksquare)$ равномерное распределение на отрезке $[1, 2]$ (так как тут мы гарантированно получим $+1$ на первом же шаге). Что мы получим? Рассмотрим первое уравнение:

$$\mathcal{Z}^\pi(\blacksquare) \stackrel{\text{c.d.f.}}{=} \underbrace{+1}_r + \underbrace{0.5}_{\gamma} \mathcal{Z}^\pi(a'),$$

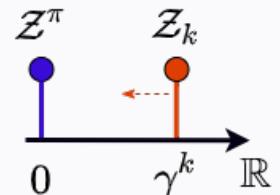
где a' — случайная величина, с равной вероятностью принимающая оба возможных значения.

Вот мы выбрали \blacksquare : с одной стороны левая часть уравнения говорит, что мы получим равномерное распределение на $[1, 2]$. С другой стороны правая часть уравнения рассматривает «одношаговое приближение»: мы точно получим $+1$, затем кинем кубик; с вероятностью 0.5 выберем на следующем шаге \blacksquare и получим равномерное из $[1, 2]$, а с вероятностью 0.5 выберем \square и получим равномерное из $[0, 1]$. Значит, начиная со второго шага мы получим смесь из равномерного на $[0, 2]$; он будет дисконтируется на гамму и получится смесь из $[0, 1]$; дальше мы его сдвинем на $+1$, который мы получили на первом шаге, и в итоге как раз получится равномерное из $[1, 2]$! Соплось; в левой и правой стороне уравнения получается одно и то же распределение! Аналогично проверяется, что сходится второе distributional-уравнение. Из доказанного нами свойства сжатия следует, что это решение — единственное, и, значит, является истинной \mathcal{Z}^π .

Итак, мы с каждым шагом алгоритма становимся всё ближе к \mathcal{Z}^π , однако только если мы понимаем близость в терминах Вассерштейна. Это не единственная метрика в $\mathbf{P}(\mathbb{R})$, по максимальной форме которой была доказана сжимаемость \mathfrak{B} ; например, ещё она доказана для максимальной формы метрики Крамера. Важно, что есть примеры метрик, для максимальных форм которых свойства сжатия нет (например, Total Variation). Для нас важен более практический пример: в реальности нам с Вассерштейном обычно неудобно работать, и мы предпочитаем более удобные **дивергенции**⁸, например, KL-дивергенцию. Чисто теоретически мы можем задаться вопросом, как ведёт себя расстояние от $\mathcal{Z}_k := \mathfrak{B}_D^k \mathcal{Z}_0$ до истинной \mathcal{Z}^π в терминах KL-дивергенции, то есть стремится ли оно хотя бы к нулю? Оказывается, не просто не стремится, а вообще полное безобразие происходит: KL-дивергенция не умеет адекватно мерить расстояние между распределениями с **несовпадающим доменом** (disjoint support).

Теорема 41: Расстояние между $\mathfrak{B}_D^k \mathcal{Z}_0$ и истинным \mathcal{Z}^π по максимальной форме KL-дивергенции может быть равно бесконечности для всех k .

Пример. Пусть в MDP с одним состоянием, одним действием и нулевой функцией награды мы проинициализировали \mathcal{Z}_0 вырожденной случайной величиной, всегда принимающей значение 1. Тогда на первом шаге метода мы получим случайную величину, с вероятностью 1 равную γ ; на k -ом шаге, по индукции, случайную величину, с вероятностью 1 равную γ^k . При этом KL-дивергенция между ней и истинным распределением \mathcal{Z}^π — вырожденной в нуле — равна бесконечности!



Так, ну ладно: оно же сходится по Вассерштейну, который лишён этой проблемы. Мы показали, что мы чисто теоретически умеем конструктивно находить \mathcal{Z}^π , запустив метод простой итерации из произвольного начального приближения (в том числе, кстати, можем стартовать из вырожденных распределений). От практического алгоритма нас пока отделяет тот факт, что даже в табличном сетинге мы не умеем в ячейках таблицы хранить «полностью» распределения на \mathbb{R} ; мы займёмся этой проблемой чуть позже.

Пока что ответим на такой вопрос: а вот когда мы учим так \mathcal{Z}^π , что там происходит с их мат.ожиданиями, то есть, по сути, с нашими представлениями о Q^π ? Может, они там как-то быстрее сходятся за счёт того, что мы начали дополнительную информацию о распределении учить? Нет: их поведение в точности совпадает с тем, что получилось бы в методе простой итерации для обучения Q-функции непосредственно.

Пусть \mathfrak{B} — обычный оператор Беллмана из пространства Q-функций в пространство Q-функций, а \mathfrak{B}_D , как и раньше, оператор Беллмана из пространства Z-функций в пространство Z-функций.

Теорема 42: Пусть инициализации \mathcal{Z}_0 и Q_0 удовлетворяют $\mathbb{E} \mathcal{Z}_0 = Q_0$, и рассматривается два метода простой

⁸снимается требование симметричности и неравенства треугольника.

итерации:

$$Q_k := \mathcal{B}^k Q_0$$

$$\mathcal{Z}_k := \mathcal{B}_D^k \mathcal{Z}_0$$

Тогда:

$$Q_k = \mathbb{E} \mathcal{Z}_k$$

Доказательство. По индукции. Пусть это верно для k -ой итерации, покажем для $k+1$ -ой:

$$\mathbb{E} \mathcal{Z}_{k+1} = \mathbb{E} [\mathcal{B}_D \mathcal{Z}_k] (s, a) = \mathbb{E} r(s, a) + \gamma \mathcal{Z}_k(s', a') = (*)$$

Заметим, что мат.ожидание в последнем выражении берётся по s', a' и случайности в $\mathcal{Z}_k(s', a')$ (случайности по хвосту траектории). По предположению индукции:

$$\mathbb{E} \mathcal{Z}_k(s', a') = Q_k(s', a')$$

Подставляем:

$$(*) = \mathbb{E}_{s'} \mathbb{E}_{a'} r(s, a) + \gamma \mathbb{E} \mathcal{Z}_k(s', a') = \mathbb{E}_{s'} \mathbb{E}_{a'} [r(s, a) + \gamma Q_k(s', a')] = \mathcal{B} Q_k = Q_{k+1}$$

■

4.3.5. Distributional Value Iteration

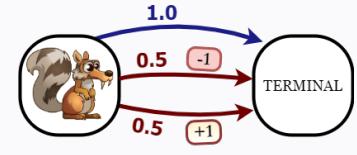
По аналогии с традиционным случаем, очень хочется ввести оптимальную оценочную функцию в distributional-форме как Z-функцию оптимальных стратегий:

$$\mathcal{Z}^*(s, a) \stackrel{\text{c.d.f.}}{=} Z^{\pi^*}(s, a) \quad (4.15)$$

Мы начинаем спотыкаться уже на этом моменте, и дальше будет только хуже.

Теорема 43: Определение (4.15) неоднозначно.

Доказательство. Рассмотрим MDP, где агент может выбрать действие $a = \square$ и получить нулевую награду с вероятностью 1, или $a = \blacksquare$ и получить $+1$ или -1 с вероятностями 0.5 (эпизод в обоих случаях заканчивается). Все стратегии будут оптимальными, хотя все Z-функции различны. ■



С уравнением оптимальности Беллмана для \mathcal{Z}^* тоже внезапно есть тонкости. Для любой оптимальной стратегии π^* вследствие (4.10) верно, что

$$Q^*(s, a) = \mathbb{E} \mathcal{Z}^*(s, a),$$

и мы знаем, что, в частности, среди оптимальных есть стратегия

$$\pi^*(s) = \operatorname{argmax}_a Q^*(s, a) = \operatorname{argmax}_a \mathbb{E} \mathcal{Z}^*(s, a).$$

В принципе, можно взять (4.11) для этой $\pi^*(s)$ и использовать её вид.

$$\mathcal{Z}^*(s, a) \stackrel{\text{c.d.f.}}{=} r(s, a) + \gamma \mathcal{Z}^*(s', \pi^*(s')), \quad s' \sim p(s' | s, a) \quad (4.16)$$

Здесь справа мы для данных s, a описываем следующий процесс генерации случайной величины: генерируем s' из функции переходов, определяем однозначно⁹ $a' = \operatorname{argmax}_{a'} \mathbb{E} \mathcal{Z}^*(s', a')$, после чего генерируем сэмпл $\mathcal{Z}^*(s', a')$ и выдаём $r(s, a) + \gamma \mathcal{Z}^*(s', a')$ в качестве результата.

Заметим, что взятие мат.ожидания справа и слева в уравнении (4.16) приведёт к традиционному уравнению оптимальности Беллмана для Q (3.17).

Определение 68: Введём оператор оптимальности Беллмана в distributional-форме \mathcal{B}_D^* :

$$[\mathcal{B}_D^* \mathcal{Z}] (s, a) \stackrel{\text{c.d.f.}}{=} r(s, a) + \gamma \mathcal{Z}(s', \operatorname{argmax}_{a'} \mathbb{E} \mathcal{Z}(s', a')), \quad s' \sim p(s' | s, a)$$

Пусть также \mathcal{B}^* — обычный оператор оптимальности Беллмана из пространства Q-функций в пространство Q-функций.

⁹здесь есть нюанс с выбором жадного действия в случае равного среднего для нескольких вариантов: для полной корректности, множество действий должно быть упорядочено, и в «спорных» ситуациях следует выбирать действие с наименьшим порядком. Это существенно, поскольку выбор разных оптимальных действий приводит к одному и тому же среднему, но другие статистики могут быть различны.

Теорема 44: Пусть инициализации \mathcal{Z}_0 и Q_0 удовлетворяют $\mathbb{E}\mathcal{Z}_0 = Q_0$, и рассматривается два метода простой итерации:

$$\begin{aligned} Q_k &:= (\mathfrak{B}^*)^k Q_0 \\ \mathcal{Z}_k &:= (\mathfrak{B}_D^*)^k \mathcal{Z}_0 \end{aligned}$$

Тогда:

$$Q_k = \mathbb{E}\mathcal{Z}_k$$

Доказательство. По индукции. Пусть это верно для k -ой итерации, покажем для $k+1$ -ой:

$$\mathbb{E}\mathcal{Z}_{k+1} = \mathbb{E}[\mathfrak{B}_D^*\mathcal{Z}_k](s, a) = \mathbb{E}\left[r(s, a) + \gamma\mathcal{Z}_k(s', \operatorname{argmax}_{a'} \mathbb{E}\mathcal{Z}_k(s', a'))\right] = (*)$$

Заметим, что внутренний аргмакс эквивалентен аргмаксу по Q -функции, так что здесь мы тоже можем воспользоваться предположением индукции:

$$\mathbb{E}\mathcal{Z}_k(s', a') = Q_k(s', a')$$

Подставляем:

$$\begin{aligned} (*) &= \mathbb{E}_{s'} \mathbb{E}_{a'} \left[r(s, a) + \gamma \mathbb{E}\mathcal{Z}_k(s', \operatorname{argmax}_{a'} Q_k(s', a')) \right] = \\ &= r(s, a) + \gamma Q_k(s', \operatorname{argmax}_{a'} Q_k(s', a')) = \\ &= r(s, a) + \gamma \max_{a'} Q_k(s', a') = \mathfrak{B}^* Q_k = Q_{k+1} \end{aligned}$$

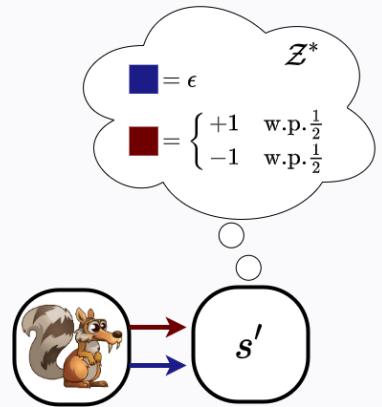
■

Итак, мы показали, что в методе простой итерации с оператором \mathfrak{B}_D^* средние движутся точно также, как и Q^* в обычном подходе. Однако, хвосты распределений при этом могут вести себя довольно нестабильно, и понятно, почему.

Теорема 45: Оператор \mathfrak{B}_D^* может не являться непрерывным.

Доказательство. Пусть после выполнения s, a мы точно попадаем в некоторое s' , для которого наше приближение \mathcal{Z} указано как на рисунке; варьируя ϵ , мы можем получать близкие (по любой непрерывной метрике) Z -функции. Рассмотрим $\epsilon \rightarrow 0$ и поймём, что оператор \mathfrak{B}_D^* выдаёт совершенно разные Z -функции в зависимости от того, приближаемся ли мы к нулю с положительной полуоси или отрицательной.

Смотрим на $[\mathfrak{B}_D^*\mathcal{Z}](s, a) \stackrel{\text{c.d.f.}}{=} \gamma\mathcal{Z}(s', \pi^*(s'))$, где $\pi^*(s')$ — жадная. Если $\epsilon > 0$, жадная политика выдаст $a' = \square$, и результат оператора будет вырожденным: $[\mathfrak{B}_D^*\mathcal{Z}](s, a)$ скажет, что с вероятностью 1 будет получена награда $\gamma\epsilon$. Если $\epsilon < 0$, то результат оператора будет дискретным распределением с двумя атомами γ и $-\gamma$ (с вероятностями $\frac{1}{2}$). Расстояние между этими двумя вариантами по любой метрике не будет нулевым. Иными словами, при переходе ϵ через ноль при непрерывном изменении аргумента оператора значение оператора может сколько угодно сильно измениться.



■

Утверждение 44: Оператор \mathfrak{B}_D^* может не являться сжимающим.

Пояснение. По определению, любой сжимающий оператор Липшицев, и, значит, обязан быть непрерывным.

■

Мы, тем не менее, показали, что средние сходятся к Q^* , поэтому не совсем ясно, насколько страшно, что хвосты распределений могут начать вести как-то нестабильно.

4.3.6. Категориальная аппроксимация Z-функций

Пока что мы не получили даже табличный алгоритм в рамках distributional-подхода: даже если пространства \mathcal{S} и \mathcal{A} конечны, а функция переходов известна, хранить в памяти точное распределение $\mathcal{Z}(s, a)$ мы не умеем. Нам придётся выбрать некоторую параметрическую аппроксимацию. Хорошая новость заключается в том, что награда — скаляр, и распределения, с которыми мы хотим работать, одномерны. Более того, в силу (1.3) домен распределений ограничен. Мы можем этим воспользоваться и придумать какую-нибудь «сеточную» аппроксимацию.

Определение 69: Зададимся набором *атомов* (atoms) $r^{\min} = z_0 < z_1 < z_2 \dots < z_A = r^{\max}$, где $A + 1$ — число атомов. Обозначим *семейство категориальных распределений* $\mathcal{C} \subset \mathbf{P}(\mathbb{R})$ как множество дискретных распределений на домене $\{z_0, z_1 \dots z_A\}$: если $\mathcal{Z}(s, a) \in \mathcal{C}$, то

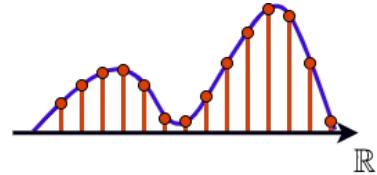
$$\mathbf{P}(\mathcal{Z}(s, a) = z_i) = p_i,$$

где p — набор из $A + 1$ чисел, суммирующихся в единицу.

Пример 80 — c51: Типично атомы образуют просто равномерную сетку, для задания которой требуется три гиперпараметра: число атомов, минимальное и максимальное значение награды. Распространённый дефолтный вариант для Atari игр — 51 атом на отрезке $[-10, 10]$. В честь такой параметризации (categorical with 51 atoms) иногда алгоритм Categorical DQN, к построению которого мы приближаемся, называют c51.

Итак, для каждой пары s, a мы будем хранить в табличке $A + 1$ неотрицательное число $p_0, p_1 \dots p_A$, суммирующиеся в единицу, и полагать, что $A + 1$ узлов нашей сетки $z_0, z_1 \dots z_A$ являются единственными возможными исходами будущей награды. Такова наша аппроксимация.

Возникает следующая проблема: мы, в принципе, можем посчитать распределения $\mathfrak{B}_D^* \mathcal{Z}$, но что, если оно «не попадёт» в рассматриваемое семейство аппроксимаций? То есть что, если для какой-то пары s, a $[\mathfrak{B}_D^* \mathcal{Z}](s, a) \notin \mathcal{C}$, то есть что, если оно не является категориальным распределением на домене $\{z_0, z_1 \dots z_A\}$? Нам придётся как-то проецировать полученный результат на нашу сетку...



Утверждение 45: В табличном сеттинге если $\mathcal{Z}(s, a) \in \mathcal{C}$ для всех s, a , то $[\mathfrak{B}_D^* \mathcal{Z}](s, a)$ — дискретное распределение с конечным множеством исходов.

Доказательство. Распределение является смесью не более чем $|\mathcal{S}| |\mathcal{A}|$ категориальных распределений с A исходами, поэтому у него не может быть более $|\mathcal{S}| |\mathcal{A}| A$ различных исходов. ■

Значит, нам нужно научиться проецировать лишь дискретные распределения.

Определение 70: Введём *оператор проекции* Π , действующий из пространства произвольных дискретных распределений в \mathcal{C} следующим образом. Пусть $\tilde{\mathcal{Z}}(s, a)$ — произвольное дискретное распределение с исходами \tilde{z}_i с соответствующими вероятностями \tilde{p}_i (суммирующимися в единицу). Изначально инициализируем все p_i для результата работы оператора нулями.

Дальше перебираем исходы \tilde{z}_i ; если очередной исход меньше $r^{\min} = z_0$, всю его вероятностную массу отправляем в p_0 , то есть увеличиваем p_0 на \tilde{p}_i . Аналогично поступаем если $\tilde{z}_i > r^{\max} = z_A$. В остальных случаях найдётся два соседних атома нашей сетки, такие что $z_j \leq \tilde{z}_i \leq z_{j+1}$. Распределим вероятностную массу между ними обратно пропорционально расстоянию до них, а то есть:

$$\begin{aligned} p_j &\leftarrow p_j + \frac{z_{j+1} - \tilde{z}_i}{z_{j+1} - z_j} \tilde{p}_i \\ p_{j+1} &\leftarrow p_{j+1} + \frac{\tilde{z}_i - z_j}{z_{j+1} - z_j} \tilde{p}_i \end{aligned} \tag{4.17}$$

Почему именно так мы ввели оператор проекции? Наш метод простой итерации теперь «подкорректированный», после каждого шага мы применяем проекцию:

$$\mathcal{Z}_{k+1} \stackrel{\text{c.d.f.}}{=} \Pi \mathfrak{B}_D^* \mathcal{Z}_k,$$

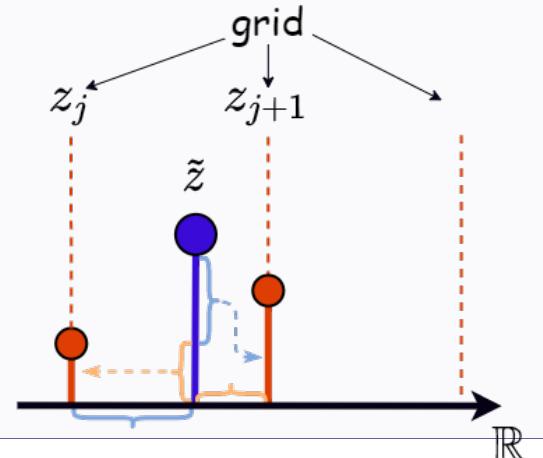
где применение Π к Z-функции означает проецирование всех распределений $\mathcal{Z}(s, a)$ для всех s, a . Мы очень хотели бы сохранить гарантии теоремы 44 о том, что средние в таком подправленном процессе продолжают вести себя как аппроксимации Q-функции в Value Iteration!

Теорема 46: Пусть $\mathcal{Z}(s, a)$ дискретно и выдаёт исходам вне отрезка $[r_{\min}, r_{\max}]$ нулевую вероятность. Тогда оператор проекции (4.17) сохраняет мат.ожидание, $\forall \mathcal{Z}, s, a$:

$$\mathbb{E} \Pi \mathcal{Z}(s, a) = \mathbb{E} \mathcal{Z}(s, a)$$

Доказательство. Рассмотрим один исход $\mathcal{Z}(s, a)$; он вносил в итоговое среднее вклад $\tilde{z}\tilde{p}$, где \tilde{p} — его вероятность, \tilde{z} — значение исхода. По условию, вся вероятностная масса размазывалась между двумя соседними узлами $z_j \leq \tilde{z} \leq z_{j+1}$, и в $[\Pi \mathcal{Z}](s, a)$ соответственно появляются два слагаемых:

$$\begin{aligned} & \text{левый узел} \quad \text{правый узел} \\ & \frac{z_{j+1} - \tilde{z}}{z_{j+1} - z_j} \tilde{p} z_j + \frac{\tilde{z} - z_j}{z_{j+1} - z_j} \tilde{p} z_{j+1} = \\ & = \frac{z_{j+1} z_j - \tilde{z} z_j + \tilde{z} z_{j+1} - z_j z_{j+1}}{z_{j+1} - z_j} \tilde{p} = \\ & = \tilde{z} \tilde{p} \end{aligned}$$



4.3.7. Categorical DQN

Попробуем составить уже полностью практический алгоритм. Впервые, обобщим алгоритм на случай произвольных пространств состояний, моделируя $\mathcal{Z}_\theta \approx \mathcal{Z}^*$ (а точнее — её распределение) при помощи нейросети с параметрами θ . Для каждой пары s, a такая нейросеть выдаёт $A + 1$ неотрицательное число $p_0(s, a, \theta), p_1(s, a, \theta) \dots p_A(s, a, \theta)$, суммирующиеся в единицу, и мы предполагаем категориальную аппроксимацию

$$\mathbf{P}(\mathcal{Z}_\theta(s, a) = z_i) := p_i(s, a, \theta). \quad (4.18)$$

Как и в DQN, считаем, что у нас есть таргет-сеть с параметрами θ^- — Z-функция \mathcal{Z}_{θ^-} с предыдущего (условно, k -го) шага метода простой итерации, а мы хотим обучать θ так, чтобы получить Z-функцию на $k + 1$ -ом шаге: наша цель — выучить $\mathfrak{B}_D^* \mathcal{Z}_{\theta^-}$.

В model-free режиме, без доступа к функции переходов, мы не то чтобы посчитать $\mathfrak{B}_D^* \mathcal{Z}_{\theta^-}$ не можем, нам даже недоступна большая часть информации о нём. Для данной пары s, a из реплей буфера мы можем получить только один сэмпл $s' \sim p(s' | s, a)$, и нам нужен какой-то «аналог» метода временных разностей.

Первое соображение: мы умеем сэмплировать из $[\mathfrak{B}_D^* \mathcal{Z}_{\theta^-}](s, a)$. Действительно: пусть дано s, a ; берём сэмпл s' из, например, буфера; смотрим на нашу таргет сеть $\mathcal{Z}_{\theta^-}(s', a')$ для всех действий a' , считаем для каждого действия a' мат.ожидание (для ситуации $\mathcal{Z}_{\theta^-}(s', a') \in \mathcal{C}$ это, очевидно, не проблема) и выбираем «наилучшее» действие $a' = \underset{a'}{\operatorname{argmax}} \mathbb{E} \mathcal{Z}_{\theta^-}(s', a')$. Выбираем такое a' , и дальше у нас есть даже не сэмпл, а целая компонента искомого распределения $[\mathfrak{B}_D^* \mathcal{Z}_{\theta^-}](s, a)$ в виде распределения $r(s, a) + \gamma \mathcal{Z}_{\theta^-}(s', a')$, которое мы и будем использовать в качестве таргета.

Итак, пусть $\mathbb{T} := (s, a, r, s')$ — четвёрка из буфера. Введём целевую переменную (таргет) следующим образом:

$$y(\mathbb{T}) \stackrel{\text{c.d.f.}}{=} r + \gamma \mathcal{Z}_{\theta^-}(s', \underset{a'}{\operatorname{argmax}} \mathbb{E} \mathcal{Z}_{\theta^-}(s', a')) \quad (4.19)$$

где s' в формуле берётся из \mathbb{T} , то есть взято из буфера. Такой таргет является дискретным распределением с, очевидно, A атомами, но из-за того, что мы взяли лишь один сэмпл s' , он является лишь компонентой из $[\mathfrak{B}_D^* \mathcal{Z}_{\theta^-}](s, a)$.

Второе соображение: допустим, для данной пары s, a мы сможем оптимизировать следующий функционал для некоторой дивергенции \mathcal{D} , используя лишь сэмплы из первого распределения:

$$\mathcal{D}([\mathfrak{B}_D^* \mathcal{Z}_{\theta^-}](s, a) \| \mathcal{Z}_\theta(s, a)) \rightarrow \min_\theta$$

Если бы мы могли моделировать произвольные Z-функции, минимум достигался бы в нуле на совпадающих распределениях, и наша цель была бы достигнута. Однако мы ограничены нашим аппроксимирующим категориальным семейством \mathcal{C} , и при оптимизации такого функционала даже чисто теоретически мы получим лишь проекцию на \mathcal{C} ; здесь возникает вопрос, а не потеряем ли мы свойство сохранения мат.ожидания. Мы знаем, что наш оператор проекции (4.17) обладает этим свойством: мы могли бы приближать наше распределение сразу к «хорошей» проекции:

$$\mathcal{D}([\Pi \mathfrak{B}_D^* \mathcal{Z}_{\theta^-}](s, a) \| \mathcal{Z}_\theta(s, a)) \rightarrow \min_\theta \quad (4.20)$$

Тогда мы будем учить категориальное распределение с сохранённым мат.ожиданием.

Вопрос: не потеряли ли мы возможность сэмплировать из целевого распределения? То есть можем ли мы получить сэмпл из $[\Pi \mathcal{B}_D^* \mathcal{Z}_{\theta-}](s, a)$?

Теорема 47:

$$[\Pi \mathcal{B}_D^* \mathcal{Z}_{\theta-}](s, a) \stackrel{\text{c.d.f.}}{=} \Pi y(\mathbb{T}), \quad s' \sim p(s' | s, a) \quad (4.21)$$

Пояснение. Сначала разберёмся, что здесь написано. Мы можем (теоретически) посчитать полностью одношаговую аппроксимацию $\mathcal{B}_D^* \mathcal{Z}_{\theta-}$ и спроектировать полученное распределение (случ. величина слева); а можем взять случайный s' , посмотреть на распределение $r + \gamma \mathcal{Z}_{\theta-}(s', a')$ для жадного a' и спроектировать его (случ. величина справа). Утверждается эквивалентность этих процедур: мы можем проецировать лишь компоненты $[\mathcal{B}_D^* \mathcal{Z}_{\theta-}](s, a)$. Таким образом, сэмплы из $\Pi y(\mathbb{T})$ при случайных $s' \sim p(s' | s, a)$ есть сэмплы $[\Pi \mathcal{B}_D^* \mathcal{Z}_{\theta-}](s, a)$.

Доказательство. Следует, в общем-то, из определения нашего оператора проекции (4.17), который с каждым возможным исходом работает «независимо» от всех остальных. Пусть для некоторого $s' p$ — вероятность исхода z , тогда в правой части эта вероятность будет размазана между соседними узлами $z_i \leq z \leq z_{i+1}$ с некоторыми весами w_i, w_{i+1} . Тогда в силу того, что s' случайно, эта вероятностная масса в итоговом распределении в правой части уравнения будет домножена на $p(s' | s, a)$. В распределении в левой части вероятностная масса будет сначала домножена на $p(s' | s, a)$, а только потом размазана между теми же z_i, z_{i+1} с теми же весами w_i, w_{i+1} (которые по определению зависят только от значения исхода z); очевидно, эти процедуры эквивалентны. ■

Итак, $\Pi y(\mathbb{T})$ есть компонента $[\Pi \mathcal{B}_D^* \mathcal{Z}_{\theta-}](s, a)$, то есть у нас, условно, есть сэмплы из этого распределения. Для каких метрик D мы умеем получать несмешённую оценку градиента (4.20) по сэмплам? Теория подсказывает, что в пространстве Z-функций осмыслиленной метрикой является Вассерштайн (4.13). И тут случается облом.

Теорема 48: Градиенты расстояния Вассерштайна до сэмплов не являются несмешёнными оценками градиента расстояния Вассерштайна до полного распределения.

Контрпример. Контрпримером будет являться практически любая ситуация, где s' недетерминировано, а наше параметрическое семейство \mathcal{Z}_{θ} может моделировать невырожденные случайные величины; так в принципе устроено расстояние Вассерштайна.

Разберём какой-нибудь пример, для простоты для \mathcal{W}_1 . Пусть для данных s, a с вероятностью 0.5 s' такого, что $y_1(\mathbb{T})$ — вырожденная в нуле, а с вероятностью 0.5 s' такого, что $y_2(\mathbb{T})$ — вырожденная в единице. Тогда понятно, что целиком распределение $[\mathcal{B}_D^* \mathcal{Z}_{\theta-}](s, a)$ есть распределение с двумя равновероятными исходами, 0 и 1.

Пусть \mathcal{Z}_{θ} равна 0.4 с вероятностью θ и 0.8 с вероятностью $1 - \theta$, других значений не принимает. Будем смотреть на точку $\theta < 0.5$. Мы как раз считали подобные расстояния Вассерштайна в примере 78. Тогда:

$$\mathcal{W}_1([\mathcal{B}_D^* \mathcal{Z}_{\theta-}](s, a) \| \mathcal{Z}_{\theta}) = 0.4\theta + 0.8(0.5 - \theta) + 0.1$$

$$\mathcal{W}_1(y_1(\mathbb{T}) \| \mathcal{Z}_{\theta}) = 0.4\theta + 0.8(1 - \theta)$$

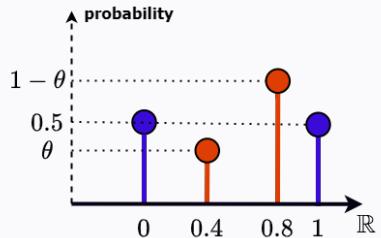
$$\mathcal{W}_1(y_2(\mathbb{T}) \| \mathcal{Z}_{\theta}) = 0.6\theta + 0.2(1 - \theta)$$

Получаем:

$$\nabla_{\theta} \mathcal{W}_1([\mathcal{B}_D^* \mathcal{Z}_{\theta-}]) \| \mathcal{Z}_{\theta}) = -0.4$$

$$\nabla_{\theta} \mathbb{E}_{s'} \mathcal{W}_p(y(\mathbb{T}) \| \mathcal{Z}_{\theta}) = \frac{1}{2}(-0.4) + \frac{1}{2}(+0.4) = 0$$

Не сходится. ■



Как мы сейчас покажем, псевдометрикой, которую можно оптимизировать по сэмплам, является наша любимая KL-дивергенция. Мы понимаем, что, с одной стороны, теория подсказывает нам, что в пространстве Z-функций KL-дивергенция потенциально не приближает нас к истинной оптимальной Z-функции, но зато мы сможем оптимизировать её в model-free режиме.

Итак, рассмотрим в (4.20) в качестве D KL-дивергенцию (значит, будет важен порядок аргументов). Для неё вылезает ещё одна проблема: домен сравниваемых распределений должен совпадать, иначе KL-дивергенция по определению бесконечность и не оптимизируется. К счастью, мы уже решили, что мы будем в качестве целевого распределения использовать $\Pi y(\mathbb{T})$, которое имеет тот же домен — сетку $z_0 < z_1 < \dots < z_A$.

Теорема 49: Градиент KL-дивергенции до целевой переменной $\Pi y(\mathbb{T})$ есть несмешённая оценка градиента (4.20):

$$\nabla_{\theta} \text{KL}([\Pi \mathfrak{B}_D^* \mathcal{Z}_{\theta-}] (s, a) \| \mathcal{Z}_{\theta}(s, a)) = \mathbb{E}_{s'} \nabla_{\theta} \text{KL}(\Pi y(\mathbb{T}) \| \mathcal{Z}_{\theta}(s, a)) \quad (4.22)$$

Доказательство.

$$\begin{aligned} \nabla_{\theta} \text{KL}([\Pi \mathfrak{B}_D^* \mathcal{Z}_{\theta-}] (s, a) \| \mathcal{Z}_{\theta}(s, a)) &= \nabla_{\theta} \text{const}(\theta) - \\ &\quad - \nabla_{\theta} \mathbb{E}_{z \sim [\Pi \mathfrak{B}_D^* \mathcal{Z}_{\theta-}] (s, a)} \log \mathbf{P}(\mathcal{Z}_{\theta}(s, a) = z) = \\ &= \{(4.21)\} = -\nabla_{\theta} \mathbb{E}_{s'} \mathbb{E}_{z \sim \Pi y(\mathbb{T})} \log \mathbf{P}(\mathcal{Z}_{\theta}(s, a) = z) = \\ &= \mathbb{E}_{s'} \nabla_{\theta} \text{KL}(\Pi y(\mathbb{T}) \| \mathcal{Z}_{\theta}(s, a)) \end{aligned}$$

■

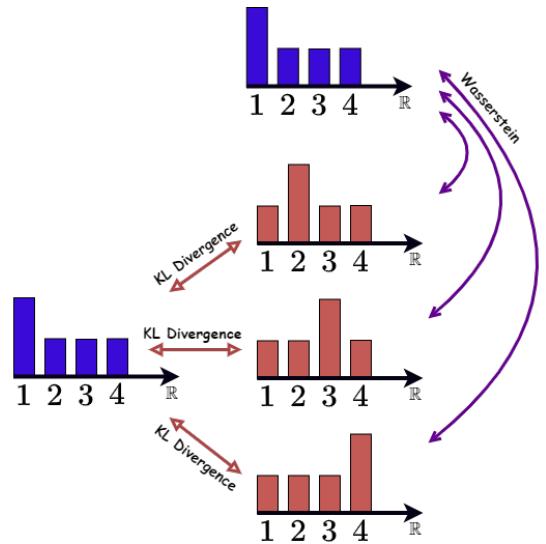
Итак, градиент KL-дивергенции — мат.ожидание по целевому распределению, и значит, мы можем вместо мат.ожидания по $[\Pi \mathfrak{B}_D^* \mathcal{Z}_{\theta-}] (s, a)$ использовать Монте-Карло оценку по сэмплам. При этом поскольку у нас есть даже не просто сэмплы, а целая компонента $\Pi y(\mathbb{T})$ целевого распределения, то по ней интеграл мы можем взять просто целиком (он состоит всего из A слагаемых, как видно, поскольку $\Pi y(\mathbb{T}) \in \mathcal{C}$).

Получается следующее: для данного перехода мы в качестве функции потерь возьмём $\text{KL}(\Pi y(\mathbb{T}) \| \mathcal{Z}_{\theta}(s, a))$, где таргет $y(\mathbb{T})$ вычисляется по формуле (4.19). Раз мы используем категориальную аппроксимацию (4.18), и $\Pi y(\mathbb{T})$ — категориальное распределение на той же сетке, то эта KL-дивергенция считается явно и (с точностью до константы, не зависящей от θ) равна

$$-\sum_{i=0}^A \mathbf{P}(\Pi y(\mathbb{T}) = z_i) \log p_i(s_t, a_t, \theta).$$

Как видно из этой формулы, мы по сути начинаем решать задачу классификации, где у нас есть для данного входа s, a сразу целая компонента «целевого» распределения. Минимизация KL-дивергенции, хоть и является стандартной функцией потерь в таких ситуациях, сейчас имеет для нас побочный эффект: мы отчасти потеряли «физический смысл» наших «классов». KL-дивергенция смотрит на каждый узел z_i нашей сетки отдельно и сравнивает вероятность, которую мы выдаём сейчас, с вероятностью z_i в таргете. Она не учитывает, находится ли разница в вероятностной массе на соседнем узле, например, z_{i+1} (в «соседнем» исходе) или на противоположном конце сетки в условном z_0 ; в обоих случаях KL-дивергенция будет выдавать одно и то же значение. Адекватные метрики в пространстве распределений, например, Вассерштайн, продифференцировали бы эти случаи. Причём заметим, что мы, вообще говоря, могли бы посчитать того же Вассерштайна между $y(\mathbb{T})$ и $\mathcal{Z}_{\theta}(s, a)$ (эти распределения дискретны, и мы разбирали, как считать расстояние Вассерштайна в таком случае в примере 78), но градиенты такой функции потерь в силу теоремы 48 не были бы несмешёнными оценками градиента для минимизации (4.20), и такой алгоритм был бы некорректен.

Тем не менее, мы получили первый полноценный Distributional алгоритм. Соберём с51, он же Categorical DQN, целиком.



Алгоритм 16: Categorical DQN (c51)

Гиперпараметры: B — размер мини-батчей, V_{\max}, V_{\min}, A — параметры категориальной аппроксимации, K — периодичность обновления таргет-сети, $\varepsilon(t)$ — стратегия исследования, $p_i(s, a, \theta)$ — нейросеть с параметрами θ , SGD-оптимизатор

Предпосчитать узлы сетки $z_i := V_{\min} + \frac{i}{A} (V_{\max} - V_{\min})$

Инициализировать θ произвольно

Положить $\theta^- := \theta$

Пронаблюдать s_0

На очередном шаге t :

1. выбрать a_t случайно с вероятностью $\varepsilon(t)$, иначе $a_t := \underset{a_t}{\operatorname{argmax}} \sum_{i=0}^A z_i p_i(s_t, a_t, \theta)$
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:

$$\mathbf{P}(y(\mathbb{T}) = r + \gamma(1 - \text{done})z_i) := p_i \left(s', \underset{a'}{\operatorname{argmax}} \sum_{i=0}^A z_i p_i(s', a', \theta^-), \theta^- \right)$$

6. спроектировать таргет на сетку $\{z_0, z_1 \dots z_A\}$: $y(\mathbb{T}) \leftarrow \Pi y(\mathbb{T})$

7. посчитать лосс:

$$\text{Loss}(\theta) := -\frac{1}{B} \sum_{\mathbb{T}} \sum_{i=0}^A \mathbf{P}(y(\mathbb{T}) = z_i) \log p_i(s_t, a_t, \theta)$$

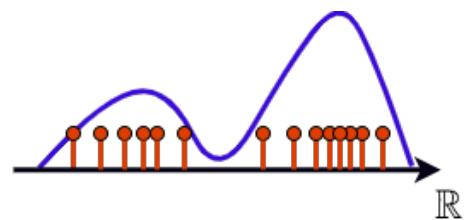
8. сделать шаг градиентного спуска по θ , используя $\nabla_{\theta} \text{Loss}(\theta)$

9. если $t \bmod K = 0$: $\theta^- \leftarrow \theta$

4.3.8. Квантильная аппроксимация Z-функций

В с51 мы воспользовались тем, что **KL**-дивергенция — это мат.ожидание по одному из сравниваемых распределений. Только это позволило нам несмешенно оценивать градиенты, используя лишь один сэмпл s' . Иначе говоря, у нас не ложатся карты: по адекватным метрикам такой фокус не прокатывает — их нельзя так просто «оптимизировать по сэмплам» — и к тому же у нас есть сложности с доменом распределения, нам необходим оператор проекции и аккуратный подбор неудобных гиперпараметров V_{\max}, V_{\min} , которые критично подобрать более-менее правильно.

Оказывается, карты сложатся, если мы выберем другую аппроксимацию распределений в $\mathbf{P}(\mathbb{R})$. Если раньше мы зафиксировали домен (узлы сетки) и подбирали вероятности, то теперь мы зафиксируем вероятности и будем подбирать узлы сетки. На первый взгляд это может показаться странно (как можно отказываться от предсказания вероятностей?), однако на самом деле это весьма гибкое семейство распределений с интересными свойствами. Итак:



Определение 71: Обозначим *семейство квантильных распределений* $\mathcal{Q} \subset \mathbf{P}(\mathbb{R})$ с A атомами как множество равномерных дискретных распределений с A произвольными исходами: если $\mathcal{Z}(s, a) \in \mathcal{Q}$, то для некоторых A чисел $z_0, z_1 \dots z_{A-1}$:

$$\mathbf{P}(\mathcal{Z}(s, a) = z_i) = \frac{1}{A}$$

Сразу хорошо то, что нам понадобится всего один гиперпараметр — число атомов A — и не понадобится подбирать верхнюю-нижнюю границу ручками. Также заметим, что вырожденное распределение принадлежит \mathcal{Q} : просто все z_i в этом случае совпадают.

$\mathfrak{B}_D^* \mathcal{Z}_{\theta^-}$, тем не менее, может снова выпадать из такого семейства представлений, и нам всё равно понадобится какая-то проекция. Но на этот раз мы сможем сделать куда более естественную проекцию. На очередном шаге для заданной пары s, a мы будем оптимизировать расстояние Вассерштейна \mathcal{W}_1 между правой частью уравнения Беллмана и тем, что мы выдаём:

$$\mathcal{W}_1([\mathfrak{B}_D^* \mathcal{Z}_{\theta^-}](s, a) \| \mathcal{Z}) \rightarrow \min_{\mathcal{Z} \in \mathcal{Q}} \quad (4.23)$$

Если бы умели выдавать произвольные распределения, мы бы искали \mathcal{Z} , условно, среди всех распределений и тогда выдали бы $[\mathfrak{B}_D^* \mathcal{Z}_{\theta^-}](s, a)$, получив точный шаг метода простой итерации. Но поскольку мы ограничены семейством квантильных распределений, то лучшее, что мы можем сделать, это спроектировать шаг метода простой итерации в него.

Ключевой момент: оказывается, задача (4.23) имеет аналитическое решение. Введём следующее обозначение («середины отрезков сетки»):

$$\tau_i := \frac{\frac{i}{A} + \frac{i+1}{A}}{2}$$

Теорема 50: Пусть F — функция распределения $[\mathfrak{B}_D^* \mathcal{Z}_{\theta^-}](s, a)$. Тогда решение $\mathcal{Z} \in \mathcal{Q}$ задачи (4.23) имеет домен $z_0, z_1 \dots z_{A-1}$:

$$z_i = F^{-1}(\tau_i)$$

Доказательство. Задача минимизации выглядит так:

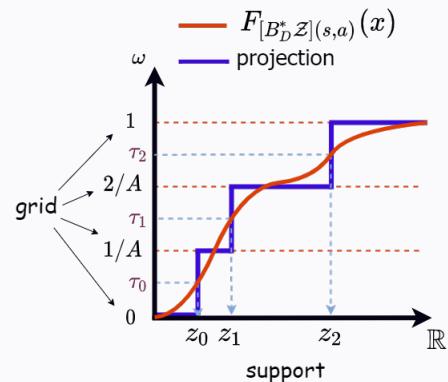
$$\int_0^1 |F^{-1}(\omega) - U_{z_0, z_1 \dots z_{A-1}}^{-1}(\omega)| d\omega \rightarrow \min_{z_0, z_1 \dots z_{A-1}} \quad (4.24)$$

где $U_{z_0, z_1 \dots z_{A-1}}^{-1}$ — функция распределения равномерного дискретного распределения на домене $\{z_0, z_1 \dots z_{A-1}\}$. Чему она равна? Ну, понятно, что это такая «лесенка»:

$$U_{z_0, z_1 \dots z_{A-1}}^{-1}(\omega) = \begin{cases} z_0 & 0 \leq \omega < \frac{1}{A} \\ z_1 & \frac{1}{A} \leq \omega < \frac{2}{A} \\ \vdots & \vdots \\ z_{A-1} & \frac{A-1}{A} \leq \omega < 1 \end{cases}$$

Подставляем это в (4.24):

$$\sum_{i=0}^{A-1} \int_{\frac{i}{A}}^{\frac{i+1}{A}} |F^{-1}(\omega) - z_i| d\omega \rightarrow \min_{z_0, z_1 \dots z_{A-1}}$$



Видим, что задача распадается на A отдельных задач оптимизации:

$$\int_{\frac{i}{A}}^{\frac{i+1}{A}} |F^{-1}(\omega) - z_i| d\omega \rightarrow \min_{z_i} \quad (4.25)$$

Продифференцируем по z_i и приравняем к нулю. Функция F монотонна, поэтому сначала будет кусок интеграла, где градиент равен -1 , затем кусок, где $+1$:

$$\int_{\frac{i}{A}}^{F(z_i)} -1 d\omega + \int_{F(z_i)}^{\frac{i+1}{A}} 1 d\omega = 0$$

Берём интегралы от константы:

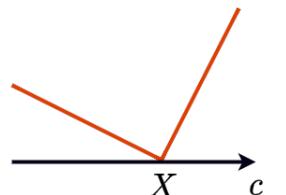
$$-\left(F(z_i) - \frac{i}{A}\right) + \frac{i+1}{A} - F(z_i) = 0$$

Откуда видим $F(z_i) = \tau_i$ и получаем доказываемое. ■

4.3.9. Quantile Regression DQN

Мы получили, что нам достаточно уметь искать лишь A определённых квантилей распределения $[B_D^* Z_{\theta-}] (s, a)$ для вычисления аппроксимации правой части уравнения Беллмана. Можем ли мы это сделать, используя только сэмплы? Конечно.

Квантильная регрессия (quantile regression) — способ получить τ -ый квантиль некоторого распределения, из которого доступна только выборка. В частном случае, мы получим известный факт о том, что для получения медианы ($\frac{1}{2}$ -го квантиля) нужно минимизировать МАЕ между константным прогнозом и сэмплами из распределения.



Определение 72: Для заданного $\tau \in (0, 1)$ квантильной функцией потерь (quantile loss) называется:

$$\text{Loss}_\tau(c, X) := \begin{cases} \tau(c - X) & c \geq X \\ (1 - \tau)(X - c) & c < X \end{cases} \quad (4.26)$$

Теорема 51 — Квантильная регрессия: Решением задачи

$$\mathbb{E}_X \text{Loss}_\tau(c, X) \rightarrow \min_{c \in \mathbb{R}} \quad (4.27)$$

будет τ -ый квантиль распределения случайной величины X .

Доказательство. Пусть F — функция распределения X . Распишем (4.27): для заданной точки c интеграл будет состоять из двух слагаемых, где в первом $c < X$, а во втором $c \geq X$:

$$\int_0^{F^{-1}(c)} (1 - \tau)(X - c) d\omega + \int_{F^{-1}(c)}^1 \tau(c - X) d\omega = 0$$

Дифференцируем по c и приравниваем к нулю:

$$\int_0^{F^{-1}(c)} (\tau - 1) d\omega + \int_{F^{-1}(c)}^1 \tau d\omega = 0$$

Берём константные интегралы:

$$F^{-1}(c)(\tau - 1) + (1 - F^{-1}(c))\tau = -F^{-1}(c) + \tau = 0$$

Отсюда получаем, что $F^{-1}(c) = \tau$, то есть c — τ -ый квантиль. ■

Утверждение 46: Формулу (4.26) можно переписать «в одну строчку» в следующем виде:

$$\text{Loss}_\tau(c, X) = (\tau - \mathbb{I}[c < X])(c - X)$$

Итак, соберём всё вместе. У нас есть нейросеть $z_i(s, a, \theta)$ с параметрами θ , которая для данного состояния-действия выплёвывает A произвольных вещественных чисел, которые мы интерпретируем как A равновероятных возможных исходов случайной величины $\mathcal{Z}_\theta(s, a)$. Обозначим за θ^- веса таргет-сети, как обычно. Для очередного перехода $T := (s, a, r, s')$ из буфера мы хотим провести оптимизацию

$$\mathcal{W}_1([\mathfrak{B}_D^* \mathcal{Z}_{\theta^-}](s, a) \parallel \mathcal{Z}_\theta(s, a)) \rightarrow \min_{\theta} \quad (4.28)$$

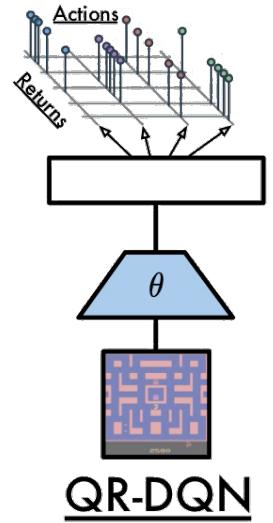
и мы поняли, что это эквивалентно поиску квантилей распределения $[\mathfrak{B}_D^* \mathcal{Z}_{\theta^-}](s, a)$, поэтому для оптимизации i -го выхода нейросетки будем оптимизировать квантильный лосс (4.26) (по i просто просуммируем — хотим учить все A интересующих нас квантилей):

$$\sum_{i=0}^{A-1} \mathbb{E}_{x \sim [\mathfrak{B}_D^* \mathcal{Z}_{\theta^-}](s, a)} \text{Loss}_{\tau_i}(z_i(s, a, \theta), x) \rightarrow \min_{\theta}$$

Конечно же, мы не будем доводить этот процесс оптимизации до конца, а сделаем всего один шаг обновления весов θ по градиенту, а затем сразу же возьмём из буфера другой мини-батч. Мы лишь получили способ получать несмещённые оценки градиентов, указывающих в сторону минимизации (4.28).

Опять же заметим, что $\mathbb{E}_{x \sim [\mathfrak{B}_D^* \mathcal{Z}_{\theta^-}](s, a)}$ распадается в сэмплирование s' и интегрирование по возможным исходам $\mathcal{Z}_{\theta^-}(s', \pi^*(s'))$, где $\pi^*(s')$ выбирает действие жадно. Мат.ожидание по $\mathcal{Z}_{\theta^-}(s', a')$ при данных s', a' есть просто усреднение по A равновероятным исходам, поэтому его мы посчитаем явно. Итого:

$$\begin{aligned} &\text{вероятности} \\ &\text{сэмплов} \\ &\sum_{i=0}^{A-1} \underbrace{\mathbb{E}_{s'} \frac{1}{A} \sum_{j=0}^{A-1}}_{\text{учим } A \text{ квантилей}} \underbrace{\text{Loss}_{\tau_i}(z_i(s, a, \theta), r + \gamma z_j(s', a', \theta^-))}_{\text{прогноз}} \rightarrow \min_{\theta} \end{aligned}$$



QR-DQN

Занося внешнюю сумму под мат.ожидание по s' , получаем функцию потерь, градиент которой можно оценивать по Монте-Карло, используя лишь сэмплы s' из функции переходов.

Алгоритм 17: Quantile Regression DQN (QR-DQN)

Гиперпараметры: B — размер мини-батчей, A — число атомов, K — периодичность обновления таргет-сети, $\varepsilon(t)$ — стратегия исследования, $z_i(s, a, \theta)$ — нейросетка с параметрами θ , SGD-оптимизатор

Предпосчитать середины отрезков квантильной сетки $\tau_i := \frac{i+A}{2}$

Инициализировать θ произвольно

Положить $\theta^- := \theta$

Пронаблюдать s_0

На **очередном** шаге t :

1. выбрать a_t случайно с вероятностью $\varepsilon(t)$, иначе $a_t := \operatorname{argmax}_{a_t} \sum_{i=0}^{A-1} z_i(s_t, a_t, \theta)$
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:

$$y(\mathbb{T})_j := r + (1 - \text{done})\gamma z_j \left(s', \underset{a'}{\operatorname{argmax}} \sum_i z_i(s', a', \theta^-), \theta^- \right)$$

6. посчитать лосс:

$$\text{Loss}(\theta) := \frac{1}{BA} \sum_{\mathbb{T}} \sum_{i=0}^{A-1} \sum_{j=0}^{A-1} (\tau_i - \mathbb{I}[z_i(s, a, \theta) < y(\mathbb{T})_j]) (z_i(s, a, \theta) - y(\mathbb{T})_j)$$

7. сделать шаг градиентного спуска по θ , используя $\nabla_{\theta} \text{Loss}(\theta)$

8. если $t \bmod K = 0$: $\theta^- \leftarrow \theta$

4.3.10. Implicit Quantile Networks

В QR-DQN мы фиксировали «равномерную сетку» на оси квантилей: говорили, что наше аппроксимирующее распределение есть равномерное на домене из A атомов. Идея: давайте будем уметь в нашей нейросети выдавать произвольные квантили, каким-то образом задавая $\tau \in (0, 1)$ дополнительно на вход. Тогда наша модель $z(s, a, \tau, \theta)$ будет неявно (implicit) задавать, вообще говоря, произвольное распределение на \mathbb{R} . По сути, мы моделируем квантильную функцию «целиком»; очень удобно:

$$F_{z_{\theta}(s, a)}^{-1}(\tau) := z(s, a, \tau, \theta)$$

Поймём, как тогда считать мат.ожидание (или Q-функцию) в такой модели.

Теорема 52: Пусть F — функция распределения случайной величины X . Тогда, если $\tau \sim U[0, 1]$, случайная величина $F^{-1}(\tau)$ имеет то же распределение, что и X .

Доказательство. Заметим, что функция распределения равномерной случайной величины при $x \in [0, 1]$ равна $P(\tau < x) = x$. Теперь посмотрим на функцию распределения случайной величины $F^{-1}(\tau)$:

$$P(F^{-1}(\tau) < x) = P(\tau < F(x)) = F(x)$$

Итак, мы можем аппроксимировать жадную стратегию примерно так:

$$\pi^*(s) := \underset{a}{\operatorname{argmax}} \sum_{i=0}^N z(s, a, \tau_i, \theta), \quad \tau_i \sim U[0, 1]$$

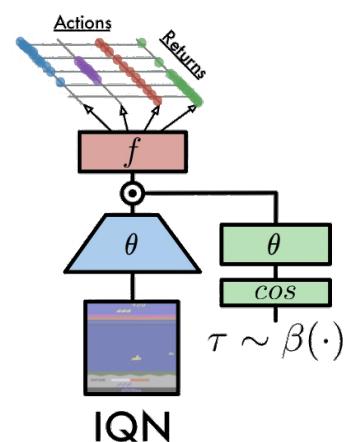
В качестве функции потерь предлагается использовать тот же квантильный лосс, что и в QR-DQN, только если в QR-DQN нам были нужны определённые A квантилей, то теперь предлагается засэмплировать N' каких-то квантилей и посчитать лосс для них. Для подсчёта лосса нам было нужно брать мат.ожидание по $\mathcal{Z}_{\theta^-}(s', a')$, для чего в формуле мы пользовались тем, что это распределение в нашей модели равномерно. Теперь же этот интеграл мы заменяем на Монте-Карло оценку с произвольным числом сэмплов N'' , а для сэмплирования опять же используем теорему 52:

$$\text{Loss}(\mathbb{T}, \theta) := \sum_{i=0}^{N'} \frac{1}{N''} \sum_{j=0}^{N''} \text{Loss}_{\tau_i}(z(s, a, \tau_i, \theta), r + \gamma z(s', \pi^*(s'), \tau_j, \theta^-)),$$

где $\tau_i, \tau_j \sim U[0, 1]$.



Возможно так обучать стратегию, которая в меньшей или большей степени предпочитает рисковать. Если при расчёте π сэмплировать квантили τ не из равномерного распределения, а чаще брать квантили, близкие к 1, агент будет в большей степени смотреть на награду, которую он может получить при везении. Если же чаще сэмплировать квантили, близкие к 0, агент будет избегать рискованных ситуаций, когда есть вероятность получить низкую награду, и это может быть полезно в Safe RL. Поэтому в общем случае можно считать, что квантили генерируются в этой схеме из некоторого распределения $\beta(\tau)$.





Архитектура сети предлагается такая. Входное состояние сжимается в некоторый эмбеддинг при помощи основной части сети $f_\theta(s)$. Параллельно строится некоторый эмбеддинг $g_\theta(\tau)$, описывающий квантиль $\tau \in (0, 1)$: тут могут быть разные варианты, но авторы остановились на следующем: число τ «описывается» несколькими признаками, где i -й признак равен $\cos(\pi i \tau)$, и преобразуется одним полно связанным слоем (с обучаемыми параметрами) для получения $g_\theta(\tau)$. Дальше финальное преобразование h должно взять $f_\theta(s)$ и $g_\theta(\tau)$ и выдать по одному числу для каждого действия a ; чтобы не городить ещё слои, взаимодействие этих двух эмбеддингов предлагается организовать при помощи поэлементного перемножения:

$$z(s, a, \tau, \theta) := h_\theta(f_\theta(s) \odot g_\theta(\tau))$$

4.3.11. Rainbow DQN

В разделе про модификации 4.2 были рассмотрены весьма разные улучшения DQN, нацеленные на решения очень разных проблем. Хорошо видно, что все эти модификации «ортогональны» и могут включаться-выключаться, так сказать, независимо в алгоритм. Distributional-подход, вообще говоря, не решает какую-то проблему внутри DQN, но может рассматриваться как ещё одна модификация базового алгоритма DQN.

Rainbow DQN совмещает 6 модификаций алгоритма DQN:

- Double DQN (раздел 4.2.3)
- Dueling DQN (раздел 4.2.4)
- Noisy DQN (раздел 4.2.5)
- Prioritized Experience Replay (раздел 4.2.6)
- Multi-step DQN (раздел 4.2.7)
- Distributional RL

В последнем пункте исторически в Rainbow используется Categorical DQN (раздел 4.3.7), однако понятно, что можно использовать любой другой алгоритм; в частности, QR-DQN (раздел 4.3.9) или IQN (раздел 4.3.10). Обсудим только пару нюансов, которые возникают при совмещении некоторых пар из этих идей, а дальше приведём полный-полный алгоритм.

Совмещение приоритизированного реплея и Distributional-подхода требует введения приоритета перехода \mathbb{T} : в его качестве, аналогично обычному случаю (4.6), берётся значение функции потерь (4.22):

$$\rho(\mathbb{T}) := \text{KL}(y(\mathbb{T}) \parallel \mathcal{Z}_\theta(s, a))$$

При совмещении шумных сетей с эвристикой Double DQN, шум сэмплируется заново на каждом проходе через сеть или таргет-сеть (то есть генерируется отдельный сэмпл шума для выбора действия a' , отдельный для оценивания при построении таргета и отдельный для прохода через сеть для подсчёта градиентов).

Забивать костыльми приходится совмещение Categorical DQN с Dueling DQN. Здесь остаётся только идея о том, что при обновлении модели для пары s, a мы должны «легче обобщаться» на все остальные действия $\hat{a} \in \mathcal{A}$, для чего вычисление $p_i(s, a, \theta)$ проходит в два потока: «типа» V-поток $V_\theta(s)$, который выдаёт \mathbf{A} атомов как бы общих для всех действий, и «типа» A-поток $A_\theta(s, a)$, который выдаёт \mathbf{A} атомов для каждого из $|\mathcal{A}|$ действий. Дальнейшая формула «взята» из обычного Dueling DQN (4.4); softmax, необходимый, чтобы получить на выходе валидное категориальное распределение, применяется в самом конце:

$$p_i(s, a, \theta) := \text{softmax}_i \left(V_\theta(s)_i + A_\theta(s, a)_i - \frac{1}{|\mathcal{A}|} \sum_a A_\theta(s, a)_i \right) \quad (4.29)$$

Последнее слагаемое — «типа» централизация A-потока к нулю, снова со средним вместо максимума (хотя эта логика к вероятностям исходов \mathcal{Z} уже плохо применима).



Ablation study там показывал, что убиение Dueling DQN, в отличие от остальных 5 модификаций, к особому падению качества итогового алгоритма не приводит. Вероятно, это связано с тем, что «семантика» потоков ещё больше теряется. Стоит отметить, что если использовать QR-DQN вместо c51, то softmax становится не нужен, и формула становится «более логичной».

Алгоритм 18: Rainbow DQN

Гиперпараметры: B — размер мини-батчей, $V_{\max}, V_{\min}, \mathbf{A}$ — параметры категориальной аппроксимации, K — периодичность обновления таргет-сети, N — количество шагов в оценке, α — степень приоритизации, $\beta(t)$ — параметр importance sampling коррекции для приоритизированного реплея, $p_i(s, a, \theta, \varepsilon)$ — нейросеть с параметрами θ , SGD-оптимизатор

Предпосчитать узлы сетки $z_i := V_{\min} + \frac{i}{A}(V_{\max} - V_{\min})$

Инициализировать θ произвольно

Положить $\theta^- := \theta$

Пронаблюдать s_0

На очередном шаге t :

1. выбрать $a_t := \operatorname{argmax}_{a_t} \sum_{i=0}^A z_i p_i(s_t, a_t, \theta, \varepsilon)$, $\varepsilon \sim \mathcal{N}(0, I)$

2. пронаблюдать r_t , s_{t+1} , done_{t+1}

3. построить N -шаговый переход $\mathbb{T} := (s, a, \sum_{n=0}^N \gamma^n r^{(n)}, s^{(N)}, \text{done})$, используя последние N наблюдений, и добавить его в реплей буфер с максимальным приоритетом

4. засэмплировать мини-батч размера B из буфера, используя вероятности $\mathbf{P}(\mathbb{T}) \propto \rho(\mathbb{T})^\alpha$

5. посчитать веса для каждого перехода:

$$w(\mathbb{T}) := \frac{1}{\rho(\mathbb{T})^{\beta(t)}}$$

6. для каждого перехода $\mathbb{T} := (s, a, \bar{r}, \bar{s}, \text{done})$ посчитать таргет:

$$\varepsilon_1, \varepsilon_2 \sim \mathcal{N}(0, I)$$

$$\mathbf{P}(y(\mathbb{T}) = \bar{r} + \gamma^N (1 - \text{done}) z_i) := p_i \left(\bar{s}, \operatorname{argmax}_{\bar{a}} \sum_{i=0}^A z_i p_i(\bar{s}, \bar{a}, \theta, \varepsilon_1), \theta^-, \varepsilon_2 \right)$$

7. спроектировать таргет на сетку $\{z_0, z_1 \dots z_A\}$: $y(\mathbb{T}) \leftarrow \Pi y(\mathbb{T})$

8. посчитать для каждого перехода лосс:

$$L(\mathbb{T}, \theta) := - \sum_{i=0}^A \mathbf{P}(y(\mathbb{T}) = z_i) \log p_i(s_t, a_t, \theta, \varepsilon) \quad \varepsilon \sim \mathcal{N}(0, I)$$

9. обновить приоритеты всех переходов из буфера: $\rho(\mathbb{T}) \leftarrow L(\mathbb{T}, \theta)$

10. посчитать суммарный лосс:

$$\text{Loss}(\theta) := \frac{1}{B} \sum_{\mathbb{T}} w(\mathbb{T}) L(\mathbb{T}, \theta)$$

11. сделать шаг градиентного спуска по θ , используя $\nabla_\theta \text{Loss}(\theta)$

12. если $t \bmod K = 0$: $\theta^- \leftarrow \theta$



При решении задач дискретного управления в off-policy режиме имеет смысл выбирать Rainbow DQN, но задумываться о том, какие модули нужны, а какие можно отключить. Дело в том, что каждая модификация DQN несколько увеличивает вычислительную сложность каждой итерации. Использование тех модулей, которые несущественны для решаемой задачи, может ускорить работу алгоритма; однако, на практике часто сложно сказать, какие модули важны в том или ином случае. Полезно помнить, какую проблему решает каждая модификация, и пытаться отслеживать, возникает ли она.



Несмотря на увесистую теорию, на практике код Distributional RL алгоритмов отличается от кода DQN буквально несколькими строчками: нужно лишь поменять размерность выхода нейронной сети и поменять функцию потерь. Поэтому их имеет смысл обязательно попробовать при экспериментировании с value-based подходом; обучение всего распределения будущей награды вместо среднего может значительно повысить sample efficiency алгоритма.

ГЛАВА 5

Policy Gradient подход

В данной главе будет рассмотрен третий, Policy Gradient подход к решению задачи, в котором целевой функционал будет оптимизироваться градиентными методами. Для этого мы выведем формулу градиента средней награды по параметрам стратегии и обсудим различные способы получения его стохастических оценок. В итоге мы сможем получить общие алгоритмы, основным ограничением которых будет жёсткий on-policy режим.

§5.1. Policy Gradient Theorem

5.1.1. Вывод первым способом

Часто говорят, что функционал в задаче обучения с подкреплением не дифференцируем. Имеется в виду, что функция награды $r(s, a)$ не дифференцируема по действиям a ; например, просто потому что пространство действий дискретно, например, в состоянии s агент выбрал действие $a = \mathbf{0}$ и значение полученной награды можно лишь сравнивать со значениями для других действий. Однако, мы уже видели в примере 9, что это не мешает дифференцируемости по параметрам стратегии в ситуации, когда стратегия ищется в семействе стохастических стратегий. Фактически, оптимизация в пространстве стохастических стратегий является этакой «релаксацией» нашей задачи.

Пусть политика $\pi_\theta(a | s)$ параметризована θ и дифференцируема по параметрам. Тогда наш оптимизируемый функционал $J(\pi_\theta) = V^{\pi_\theta}(s_0)$ тоже дифференцируем по θ , и далее мы будем выводить формулу этого градиента. Для этого нам понадобится стандартная техника вычисления градиента мат.ожидания по распределению, зависящего от параметров; мы уже встречались с ней при обсуждении эволюционных стратегий в главе 2.2.5. Сейчас в оптимизируемом функционале у нас стоит целая цепочка вложенных мат.ожиданий, и наш вывод будет заключаться просто в последовательном применении той же техники к каждому стоящему там мат.ожиданию $\mathbb{E}_{a \sim \pi(a|s)}(\cdot)$.

Заранее оговоримся, что при минимальных технических условиях регулярности¹ мы имеем право менять местами знаки градиента, мат.ожиданий, сумм и интегралов.

Теорема 53:

$$\nabla_\theta V^{\pi_\theta}(s) = \mathbb{E}_a [\nabla_\theta \log \pi_\theta(a | s) Q^\pi(s, a) + \nabla_\theta Q^{\pi_\theta}(s, a)] \quad (5.1)$$

Доказательство.

$$\begin{aligned} \nabla_\theta V^{\pi_\theta}(s) &= \{(3.6)\} = \nabla_\theta \mathbb{E}_{a \sim \pi_\theta(a|s)} Q^{\pi_\theta}(s, a) = \\ &= \{\text{мат.ожидание — это интеграл}\} = \\ &= \nabla_\theta \int_A \pi_\theta(a | s) Q^{\pi_\theta}(s, a) da = \\ &= \{\text{проносим градиент внутрь интеграла}\} = \\ &= \int_A \nabla_\theta [\pi_\theta(a | s) Q^{\pi_\theta}(s, a)] da = \\ &= \{\text{правило градиента произведения}\} = \end{aligned}$$

¹Это следует из наших условий регулярности на MDP и предположения интегрируемости всех функций; тогда все оценочные функции и награды ограничены, следовательно, все интегралы и ряды в рассуждении сходятся абсолютно и равномерно по параметрам θ ; мы просто не «связываемся» в контексте нашей задачи с бесконечностью, на которой в теории математического анализа и возникают ситуации, когда так делать нельзя.

$$\begin{aligned}
&= \int_{\mathcal{A}} \nabla_{\theta} \pi_{\theta}(a | s) Q^{\pi}(s, a) da + \int_{\mathcal{A}} \pi(a | s) \nabla_{\theta} Q^{\pi_{\theta}}(s, a) da = \\
&= \{\text{второе слагаемое — это мат.ожидание}\} = \\
&= \int_{\mathcal{A}} \nabla_{\theta} \pi_{\theta}(a | s) Q^{\pi}(s, a) da + \mathbb{E}_a \nabla_{\theta} Q^{\pi_{\theta}}(s, a) = \\
&= \{\text{log-derivative trick (2.7)}\} = \\
&= \int_{\mathcal{A}} \pi_{\theta}(a | s) \nabla_{\theta} \log \pi_{\theta}(a | s) Q^{\pi}(s, a) da + \mathbb{E}_a \nabla_{\theta} Q^{\pi_{\theta}}(s, a) = \\
&= \{\text{первое слагаемое тоже стало мат.ожиданием}\} = \\
&= \mathbb{E}_a [\nabla_{\theta} \log \pi_{\theta}(a | s) Q^{\pi}(s, a) + \nabla_{\theta} Q^{\pi_{\theta}}(s, a)]
\end{aligned}$$

■

Эта техника вычисления градиента через «стохастичный узел нашего вычислительного графа», когда мы сэмплируем $a \sim \pi(a | s)$, носит название REINFORCE. Как видно, эта техника универсальна: применима всегда для любых пространств действий, а также в ситуации, когда функция $Q^{\pi}(s, a)$ не дифференцируема по действиям. Заметим, что в глубоком обучении при некоторых дополнительных условиях может быть также применим другой способ; его мы обсудим позже в главе 6, когда пространство действий будет непрерывно, а $Q^{\pi}(s, a)$ — дифференцируема по действиям, и альтернативный метод будет применен.

Мы смогли выразить градиент V-функции через градиент Q-функции, попробуем сделать наоборот. Для этого нам нужно посчитать градиент от мат.ожидания по функции переходов, не зависящей от параметров нашей стратегии, поэтому здесь всё тривиально.

Утверждение 47:

$$\nabla_{\theta} Q^{\pi_{\theta}}(s, a) = \gamma \mathbb{E}_{s'} \nabla_{\theta} V^{\pi_{\theta}}(s') \quad (5.2)$$

Доказательство.

$$\begin{aligned}
\nabla_{\theta} Q^{\pi_{\theta}}(s, a) &= \{(3.5)\} = \nabla_{\theta} [r(s, a) + \gamma \mathbb{E}_{s'} V^{\pi_{\theta}}(s')] = \\
&= \{r(s, a) \text{ не зависит от } \theta, p(s' | s, a) \text{ тоже}\} = \\
&= \gamma \mathbb{E}_{s'} \nabla_{\theta} V^{\pi_{\theta}}(s')
\end{aligned}$$

■

Подставляя (5.2) в (5.1), получаем:

Утверждение 48:

$$\nabla_{\theta} V^{\pi_{\theta}}(s) = \mathbb{E}_a \mathbb{E}_{s'} [\nabla_{\theta} \log \pi_{\theta}(a | s) Q^{\pi}(s, a) + \gamma \nabla_{\theta} V^{\pi_{\theta}}(s')] \quad (5.3)$$

Следовательно, мы получили рекурсивное выражение градиента $V^{\pi_{\theta}}(s)$ через него же само. Очень похоже на уравнение Беллмана, кстати: в правой части стоит мат.ожидание по выбранному в s действию a и следующему состоянию.

Осталось раскрутить эту рекурсивную цепочку, продолжая раскрывать $\nabla_{\theta} V^{\pi_{\theta}}(s')$ в будущее до бесконечности. Аккуратно собирая слагаемые, а также собирая из мат.ожиданий мат.ожидание по траектории, получаем следующее:

Утверждение 49 — Policy Gradient Theorem: Выражение для градиента оптимизируемого функционала можно записать следующим образом:

$$\nabla_{\theta} J(\pi_{\theta}) = \mathbb{E}_{\mathcal{T} \sim \pi} \sum_{t \geq 0} \gamma^t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) Q^{\pi}(s_t, a_t) \quad (5.4)$$

5.1.2. Вывод вторым способом

Прежде, чем мы обсудим физический смысл полученной формулы, выведем её альтернативным способом. Применим REINFORCE не к мат.ожиданиям по отдельным действиям, а напрямую к распределению всей траектории следующим образом:

Теорема 54:

$$\nabla_{\theta} V^{\pi_{\theta}}(s) = \mathbb{E}_{\mathcal{T} \sim \pi | s_0 = s} \sum_{t \geq 0} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) R(\mathcal{T})$$

Доказательство.

$$\begin{aligned}
 \nabla_{\theta} V^{\pi_{\theta}}(s) &= \nabla_{\theta} \mathbb{E}_{\mathcal{T} \sim \pi | s_0 = s} R(\mathcal{T}) = \\
 &= \{\text{рассмотрим матожидание как интеграл по всевозможным траекториям}\} = \\
 &= \nabla_{\theta} \iint_{\mathcal{T}} p(\mathcal{T} | \pi, s_0 = s) R(\mathcal{T}) d\mathcal{T} = \\
 &= \{\text{проносим градиент внутрь интеграла}\} = \\
 &= \iint_{\mathcal{T}} \nabla_{\theta} p(\mathcal{T} | \pi, s_0 = s) R(\mathcal{T}) d\mathcal{T} = \\
 &= \{\text{log-derivative trick (2.7)}\} = \\
 &= \mathbb{E}_{\mathcal{T} \sim \pi | s_0 = s} \nabla_{\theta} \log p(\mathcal{T} | \pi, s_0 = s) R(\mathcal{T}) = \\
 &= \{\text{вспоминаем, что вероятность траектории есть произведение вероятностей}\} = \\
 &= \mathbb{E}_{\mathcal{T} \sim \pi | s_0 = s} \sum_{t \geq 0} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) R(\mathcal{T})
 \end{aligned}$$

В последнем переходе логарифмы вероятностей переходов $\sum_{t \geq 0} \nabla_{\theta} \log p(s_{t+1} | s_t, a_t)$ сократились как не зависящие от параметров стратегии. ■

Видим, что мы более простым способом получили очень похожую формулу, но с суммарной наградой за игры вместо Q-функции из первого доказательства (5.4). В силу корректности всех вычислений, уже можно утверждать равенство между этими формулами, что наводит на мысль, что градиент можно записывать в нескольких математически эквивалентных формах. Математически эти формы будут эквивалентны, то есть равны, как интегралы, но их Монте-Карло оценки могут начать вести себя совершенно по-разному. Может быть, мы можем как-то «хорошую форму» выбрать для наших алгоритмов.

Рассмотрим, как можно этим вторым способом рассуждений дойти формально до формы градиента из первого способа. Раскрывая $R(\mathcal{T})$ по определению, мы сейчас имеем произведение двух сумм под интегралом:

$$\nabla_{\theta} V^{\pi_{\theta}}(s) = \mathbb{E}_{\mathcal{T} \sim \pi | s_0 = s} \left(\sum_{t \geq 0} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right) \left(\sum_{\hat{t} \geq 0} \gamma^{\hat{t}} r_{\hat{t}} \right)$$

Давайте перемножим эти два ряда:

$$\nabla_{\theta} V^{\pi_{\theta}}(s) = \mathbb{E}_{\mathcal{T} \sim \pi | s_0 = s} \sum_{t \geq 0} \sum_{\hat{t} \geq 0} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \gamma^{\hat{t}} r_{\hat{t}} \quad (5.5)$$

Видим странную вещь: на градиент по параметрам за решение выбрать a_t в момент времени t влияет награда, собранная при $\hat{t} < t$, то есть величина, на которую наше решение точно не могло повлиять, поскольку принималось после её получения. Но почему формально это так?

Теорема 55: Для произвольного распределения $\pi_{\theta}(a)$ с параметрами θ , верно:

$$\mathbb{E}_{a \sim \pi_{\theta}(a)} \nabla_{\theta} \log \pi_{\theta}(a) = 0 \quad (5.6)$$

Доказательство.

$$\begin{aligned}
 \mathbb{E}_{a \sim \pi_{\theta}(a)} \nabla_{\theta} \log \pi_{\theta}(a) &= \{\text{производная логарифма}\} = \mathbb{E}_{a \sim \pi_{\theta}(a)} \frac{\nabla_{\theta} \pi_{\theta}(a)}{\pi_{\theta}(a)} = \\
 &= \int_{\mathcal{A}} \nabla_{\theta} \pi_{\theta}(a) da = \nabla_{\theta} \int_{\mathcal{A}} \pi_{\theta}(a) da = \nabla_{\theta} 1 = 0
 \end{aligned}$$

■

Следующее утверждение формализует этот тезис о том, что «будущее не влияет на прошлое»: выбор действий в некоторый момент времени никак не влияет на те слагаемые из награды, которые были получены в прошлом.

Теорема 56 — Принцип причинности (causality): При $t > \hat{t}$:

$$\mathbb{E}_{\mathcal{T} \sim \pi} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \gamma^{\hat{t}} r_{\hat{t}} = 0$$

Доказательство.

$$\begin{aligned}
 & \mathbb{E}_{\tau \sim \pi} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \gamma^{\hat{t}} r_{\hat{t}} = \\
 &= \{\text{представляем мат.ожидание по траектории как вложенные мат.ожидания}\} = \\
 &= \mathbb{E}_{a_1, s_1 \dots s_{\hat{t}}, a_{\hat{t}}} \mathbb{E}_{s_{\hat{t}+1}, a_{\hat{t}+1} \dots s_t, a_t \dots} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \gamma^{\hat{t}} r_{\hat{t}} = \\
 &= \{\text{выносим константу из мат.ожидания}\} = \\
 &= \mathbb{E}_{a_1, s_1 \dots s_{\hat{t}}, a_{\hat{t}}} \gamma^{\hat{t}} r_{\hat{t}} \mathbb{E}_{s_{\hat{t}+1}, a_{\hat{t}+1} \dots s_t, a_t \dots} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) = \\
 &= \{\text{мат.ожидание градиента логарифма вероятности есть ноль (5.6)}\} = \\
 &= \mathbb{E}_{a_1, s_1 \dots s_{\hat{t}}, a_{\hat{t}}} \gamma^{\hat{t}} r_{\hat{t}} \cdot \mathbf{0} = \mathbf{0}
 \end{aligned}$$

■

Значит, вместо полной награды за игру можно в весах оставить только reward-to-go (3.1), поскольку из всех слагаемых в (5.5) слагаемые для $\hat{t} < t$ занулятся. Понятно, что дисперсия Монте-Карло оценки такого интеграла будет меньше: мы убрали некоторую запутывающую часть нашей стохастической оценки, которая, как мы теперь поняли, в среднем равна нулю.

При этом дисконтирование в сумме наград шло с самого начала игры, поэтому для того, чтобы записать формулу в терминах reward-to-go, нужно вынести γ^t :

$$\sum_{\hat{t} \geq t} \gamma^{\hat{t}} r_{\hat{t}} = \gamma^t \sum_{\hat{t} \geq t} \gamma^{\hat{t}-t} r_{\hat{t}} = \gamma^t R_t$$

Утверждение 50:

$$\nabla_{\theta} V^{\pi_{\theta}}(s) = \mathbb{E}_{\tau \sim \pi | s_0 = s} \sum_{t \geq 0} \gamma^t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) R_t \quad (5.7)$$

Reward-to-go очень похож на Q-функцию, так как является Монте-Карло оценкой Q-функции, а мат.ожидание по распределениям всё равно берётся. Формальное обоснование эквивалентности выглядит так:

Утверждение 51: Формула (5.7) эквивалентна

$$\nabla_{\theta} V^{\pi_{\theta}}(s) = \mathbb{E}_{\tau \sim \pi | s_0 = s} \sum_{t \geq 0} \gamma^t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) Q^{\pi}(s_t, a_t)$$

Доказательство.

$$\begin{aligned}
 \nabla_{\theta} V^{\pi_{\theta}}(s) &= \mathbb{E}_{\tau \sim \pi | s_0 = s} \sum_{t \geq 0} \gamma^t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) R_t = \\
 &= \{\text{меняем местами сумму по } t \text{ и мат.ожидание по траекториям}\} = \\
 &= \sum_{t \geq 0} \mathbb{E}_{\tau \sim \pi | s_0 = s} \gamma^t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) R_t = \\
 &= \{\text{представляем мат.ожидание по траектории как вложенные мат.ожидания}\} = \\
 &= \sum_{t \geq 0} \mathbb{E}_{a_0, s_1 \dots s_t, a_t} \mathbb{E}_{s_{t+1}, a_{t+1} \dots} \gamma^t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) R_t = \\
 &= \{\text{выносим константу из мат.ожидания}\} = \\
 &= \sum_{t \geq 0} \mathbb{E}_{a_0, s_1 \dots s_t, a_t} \gamma^t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \mathbb{E}_{s_{t+1}, a_{t+1} \dots} R_t = \\
 &= \{\text{видим определение Q-функции}\} = \\
 &= \sum_{t \geq 0} \mathbb{E}_{a_0, s_1 \dots s_t, a_t} \gamma^t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) Q^{\pi}(s_t, a_t) = \\
 &= \{\text{формально, берём фиктивное мат.ожидание по } s_{t+1}, a_{t+1}, \dots\} = \\
 &= \sum_{t \geq 0} \mathbb{E}_{\tau \sim \pi | s_0 = s} \gamma^t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) Q^{\pi}(s_t, a_t) = \\
 &= \{\text{снова меняем местами сумму и мат.ожидание}\} =
 \end{aligned}$$

$$= \mathbb{E}_{\mathcal{T} \sim \pi | s_0 = s} \sum_{t \geq 0} \gamma^t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) Q^{\pi}(s_t, a_t)$$

■

Итак, мы получили формулу (5.4) вторым способом.

5.1.3. Физический смысл

Обсудим, а в каком направлении, собственно, указывает полученная формула градиента (5.4). Оказывается, градиент нашего функционала имеет вид градиента взвешенных логарифмов правдоподобий. Чтобы ещё лучше увидеть это, рассмотрим **суррогатную функцию** (surrogate objective) — другой функционал, который будет иметь в точке текущих значений параметров стратегии π такой же градиент, как и $J(\theta)$:

$$\mathcal{L}_{\tilde{\pi}}(\theta) := \mathbb{E}_{\mathcal{T} \sim \tilde{\pi}(s)} \sum_{t \geq 0} \gamma^t \log \pi_{\theta}(a_t | s_t) Q^{\tilde{\pi}}(s_t, a_t) \quad (5.8)$$

Это суррогатная функция от двух стратегий: стратегии π_{θ} , которую мы оптимизируем, и ещё одной стратегии $\tilde{\pi}$. Давайте рассмотрим эту суррогатную функцию в точке θ такой, что эти две стратегии совпадают: $\tilde{\pi} \equiv \pi_{\theta}$, и посмотрим на градиент при изменении θ , только одной из них. То есть что мы сказали: давайте «заморозим» оценочную Q -функцию, и «заморозим» распределение, из которого приходят пары s, a . Тогда:

Утверждение 52:

$$\nabla_{\theta} \mathcal{L}_{\tilde{\pi}}(\theta)|_{\tilde{\pi}=\pi_{\theta}} = \nabla_{\theta} J(\theta)$$

Доказательство. Поскольку матожидание по траекториям не зависит в этой суррогатной функции от θ , то градиент просто можно пронести внутрь:

$$\nabla_{\theta} \mathcal{L}_{\tilde{\pi}}(\theta)|_{\tilde{\pi}=\pi_{\theta}} = \mathbb{E}_{\mathcal{T} \sim \tilde{\pi}(s)} \sum_{t \geq 0} \gamma^t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)|_{\tilde{\pi}=\pi_{\theta}} Q^{\tilde{\pi}}(s_t, a_t)$$

В точке θ : $\tilde{\pi} = \pi_{\theta}$ верно, что $p(\mathcal{T} | \tilde{\pi}) \equiv p(\mathcal{T} | \pi)$ и $Q^{\tilde{\pi}}(s, a) = Q^{\pi}(s, a)$; следовательно, значение градиента в этой точке совпадает со значением формулы (5.4). ■

Значит, направление максимизации $J(\theta)$ в текущей точке θ просто совпадает с направлением максимизации этой суррогатной функции! Это принципиально единственное свойство введённой суррогатной функции. Таким образом, можно считать, что в текущей точке мы на самом деле «как бы» максимизируем (5.8), а это уже в чистом виде логарифм правдоподобия каких-то пар (s, a) , для каждой из которых дополнительно выдан «вес» в виде значения $Q^{\pi}(s, a)$.



Эта суррогатная функция очень удобна для подсчёта градиента $\nabla_{\theta} J(\pi)$, поскольку она представляет его «в терминах лосса»:

$$\nabla_{\theta} J(\pi) = \nabla_{\theta} \text{Loss}^{\text{actor}}(\theta)$$

Это полезно в средствах автоматического дифференцирования, где нужно задать некоторый вычислительный граф для получения градиентов; также её можно считать некой «функцией потерь» для актёра, хотя это название очень условно хотя бы потому, что значение этой функции вовсе не должно убывать и может вести себя довольно хаотично.

Проведём такую аналогию с задачей обучения с учителем: если в машинном обучении в задачах регрессии и классификации мы для данной выборки (x, y) максимизировали правдоподобие

$$\sum_{(x, y)} \log p(y | x, \theta) \rightarrow \max_{\theta},$$

то теперь в RL, когда выборки нет, мы действуем по-другому: мы сэмплируем сами себе входные данные s и примеры выходных данных a , выдаём каждой паре какой-то «кредит доверия» (credit), некую скалярную оценку хорошести, выраженную в виде $Q^{\pi}(s, a)$, и идём в направлении максимизации

$$\sum_{(s, a)} \log \pi(a | s, \theta) Q^{\pi}(s, a) \rightarrow \max_{\theta}.$$

5.1.4. REINFORCE

Попробуем сразу построить какой-нибудь практический RL алгоритм при помощи формулы (5.4). Нам достаточно лишь несмешённой оценки на градиент, чтобы воспользоваться методами стохастической градиентной оптимизации, и поэтому мы просто попробуем всё неизвестное в формуле заменить на Монте-Карло оценки.

Во-первых, для оценки мат.ожидания по траекториям просто сыграем несколько полных игр при помощи текущей стратегии π . Сразу заметим, что мы тогда требуем эпизодичности среды и сразу ограничиваем себя **on-policy режимом**: для каждого следующего шага нам требуется сыграть эпизоды при помощи именно текущей стратегии. Во-вторых, воспользуемся Монте-Карло оценкой для приближения $Q^\pi(s_t, a_t)$, заменив его просто на reward-to-go:

$$Q^\pi(s, a) \approx R(\mathcal{T}), \quad \mathcal{T} \sim \pi \mid s_t = s, a_t = a$$

Можно сказать, что мы воспользовались формулой градиента в форме (5.7).

Алгоритм 19: REINFORCE

Гиперпараметры: N — количество игр, $\pi(a \mid s, \theta)$ — стратегия с параметрами θ , SGD-оптимизатор.

Инициализировать θ произвольно

На очередном шаге t :

1. играем N игр $\mathcal{T}_1, \mathcal{T}_2 \dots \mathcal{T}_N \sim \pi$

2. для каждого t в каждой игре \mathcal{T} считаем reward-to-go: $R_t(\mathcal{T}) := \sum_{\hat{t}=t}^T \gamma^{\hat{t}-t} r_{\hat{t}}$

3. считаем оценку градиента:

$$\nabla_\theta J(\pi) := \frac{1}{N} \sum_{\mathcal{T}} \sum_{t \geq 0} \gamma^t \nabla_\theta \log \pi(a_t \mid s_t, \theta) R_t(\mathcal{T})$$

4. делаем шаг градиентного подъёма по θ , используя $\nabla_\theta J(\pi)$

Первая беда такого алгоритма очевидна: для одного шага градиентного подъёма нам необходимо играть несколько игр целиком (!) при помощи текущей стратегии. Такой алгоритм просто неэффективен в плане сэмплов, и это негативная сторона on-policy режима. Чтобы как-то снизить этот эффект, хотелось бы научиться как-то делать шаги обучения, не доигрывая эпизоды до конца.

Вторая проблема алгоритма — колоссальная дисперсия нашей оценки градиента. На одном шаге направление оптимизации указывает в одну сторону, на следующем — совсем в другую. В силу корректности нашей оценки все гарантии стохастической оптимизации лежат у нас в кармане, но на практике дождаться каких-то результатов от такого алгоритма в сколько-то сложных задачах не получится.

Чтобы разобраться с этими двумя проблемами, нам понадобится чуть подробнее познакомиться с конструкцией (5.4).

5.1.5. State visitation frequency

Мы получили, что градиент оптимизируемого функционала по параметрам стратегии (5.4) имеет вид мат.ожиданий по траекториям стратегии π . Казалось бы, для его оценки нам придётся играть полные эпизоды. Однако видно, что внутри интеграла по траекториям и суммы по времени стоят нечто, зависящее только от пар состояния-действие:

$$f(s, a) := \log \pi_\theta(a \mid s) Q^\pi(s, a)$$

Нельзя ли как-то сказать, что если мы делаем стратегией π лишь несколько шагов в среде, то собранные пары s, a приходят из того самого распределения, которое мы хотим оценить?

Допустим, мы взаимодействуем со средой при помощи стратегии π . Из какого распределения нам приходят состояния, которые мы встречаем?

Утверждение 53: Состояния, которые встречает агент со стратегией π , приходят из некоторой стационарной марковской цепи.

Доказательство. Выпишем вероятность оказаться на очередном шаге в состоянии s' , если мы используем стратегию π :

$$p(s' \mid s) = \int_{\mathcal{A}} \pi(a \mid s) p(s' \mid s, a) da$$

Эта вероятность не зависит от времени и от истории, следовательно, цепочка состояний образует марковскую цепь. ■

Допустим, начальное состояние s_0 фиксировано. Обозначим вероятность оказаться в состоянии s в момент времени t при использовании стратегии π как $p(s_t = s | \pi)$. Мы могли бы попробовать посчитать, сколько раз мы в среднем оказываемся в некотором состоянии s , просто просуммировав по времени:

$$\sum_{t \geq 0} p(s_t = s | \pi),$$

однако, как легко видеть, такой ряд может оказаться равен бесконечности (например, если в MDP всего одно состояние). Поскольку мы хотели получить распределение, мы можем попробовать отнормировать этот «счётчик»:

Определение 73: Для данного MDP и политики π *state visitation frequency* называется

$$\mu_\pi(s) := \lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t \geq 0}^T p(s_t = s | \pi) \quad (5.9)$$

Вообще говоря, мы мало что можем сказать про это распределение. При некоторых технических условиях у марковской цепи встречающихся состояний может существовать некоторое стационарное распределение $\lim_{t \rightarrow \infty} p(s_t = s | \pi)$, из которого будут приходить встречающиеся состояния, условно, через бесконечное количество шагов взаимодействия; если так, то (5.9) совпадает с ним, поскольку, интуитивно, начиная с некоторого достаточно большого момента времени t все слагаемые в ряде будут очень похожи на стационарное распределение.

Можно примерно считать, что во время обучения при взаимодействии со средой состояния приходят из $\mu_\pi(s)$, где π — стратегия взаимодействия. Конечно, это верно лишь в предположении, что марковская цепь уже «разогрелась» и распределение действительно похоже на стационарное; то есть, в предположении, что обучение продолжается достаточно долго (обычно это так), и предыдущие стратегии, использовавшиеся для взаимодействия, менялись достаточно плавно.

Естественно, надо помнить, что сэмплы из марковской цепи скоррелированы, и соседние состояния будут очень похожи — то есть независимости в цепочке встречающихся состояний, конечно, нет. При необходимости набрать мини-батч независимых сэмплов из $\mu_\pi(s)$ для декорреляции необходимо воспользоваться параллельными средами: запустить много сред параллельно и для очередного мини-батча собирать состояния из разных симуляций взаимодействия. Вообще, если в батч попадает целая цепочка состояний из одной и той же среды, то это нарушает условие независимости сэмплов и мешает обучению.

5.1.6. Расцепление внешней и внутренней стохастики

Итак, давайте попробуем формально понять, из какого распределения приходят состояния в формуле градиента (5.4), и отличается ли оно от $\mu_\pi(s)$. Для этого мы сейчас придумаем, как можно записывать функционалы вида

$$\mathbb{E}_{\tau \sim \pi} \sum_{t \geq 0} \gamma^t f(s_t, a_t),$$

где f — какая-то функция от пар состояния-действие, немного по-другому.

В MDP есть два вида стохастики:

- **внешняя** (extrinsic), связанная со случайностью в самой среде и неподконтрольная агенту; она заложена в функции переходов $p(s' | s, a)$.
- **внутренняя** (intrinsic), связанная со случайностью в стратегии самого агента; она заложена в $\pi(a | s)$. Это стохастика нам подконтрольна при обучении.

Мат. ожидание по траектории (1.1) плохо тем, что мат.ожидания по внешней и внутренней стохастике чередуются. При этом во время обучения из внешней стохастики мы можем только получать сэмплы, поэтому было бы здорово переписать наш функционал как-то так, чтобы он имел вид мат.ожидания по всей внешней стохастике.

Введём ещё один, «дисконтированный счётчик посещения состояний» для стратегии взаимодействия π . При дисконтировании отпадают проблемы с нормировкой.

Определение 74: Для данного MDP и политики π *discounted state visitation distribution* называется

$$d_\pi(s) := (1 - \gamma) \sum_{t \geq 0} \gamma^t p(s_t = s | \pi) \quad (5.10)$$

Утверждение 54: State visitation distribution есть распределение на множестве состояний, то есть:

$$\int_s d_\pi(s) ds = 1$$

Доказательство.

$$\int_s d_\pi(s) ds = \int_s (1 - \gamma) \sum_{t \geq 0} \gamma^t p(s_t = s) ds = (1 - \gamma) \sum_{t \geq 0} \gamma^t \int_s p(s_t = s) ds = (1 - \gamma) \sum_{t \geq 0} \gamma^t = 1 \quad \blacksquare$$

State visitation distribution (5.10) является важным понятием ввиду следующей теоремы, благодаря которой мы можем чисто теоретически (!) расцепить (decouple) внешнюю и внутреннюю стохастику:

Теорема 57: Для произвольной функции $f(s, a)$:

$$\mathbb{E}_{\tau \sim \pi} \sum_{t \geq 0} \gamma^t f(s_t, a_t) = \frac{1}{1 - \gamma} \mathbb{E}_{s \sim d_\pi(s)} \mathbb{E}_{a \sim \pi(a|s)} f(s, a) \quad (5.11)$$

Доказательство.

$$\begin{aligned} \mathbb{E}_{\tau \sim \pi} \sum_{t \geq 0} \gamma^t f(s_t, a_t) &= \\ \{\text{меняем местами сумму и интеграл}\} &= \sum_{t \geq 0} \gamma^t \mathbb{E}_{\tau \sim \pi} f(s_t, a_t) = \\ \{\text{расписываем мат.ожидание по траектории}\} &= \sum_{t \geq 0} \gamma^t \int_s \int_{\mathcal{A}} p(s_t = s, a_t = a | \pi) f(s, a) da ds = \\ \{\text{по определению процесса}\} &= \sum_{t \geq 0} \gamma^t \int_s \int_{\mathcal{A}} p(s_t = s | \pi) \pi(a | s) f(s, a) da ds = \\ \{\text{выделяем мат.ожидание по стратегии}\} &= \sum_{t \geq 0} \gamma^t \int_s p(s_t = s | \pi) \mathbb{E}_{\pi(a|s)} f(s, a) ds = \\ \{\text{заносим сумму по времени обратно}\} &= \int_s \sum_{t \geq 0} \gamma^t p(s_t = s | \pi) \mathbb{E}_{\pi(a|s)} f(s, a) ds = \\ \{\text{выделяем state visitation distribution (5.10)}\} &= \int_s \frac{d_\pi(s)}{1 - \gamma} \mathbb{E}_{\pi(a|s)} f(s, a) ds = \\ \{\text{выделяем мат.ожидание по состояниям}\} &= \frac{1}{1 - \gamma} \mathbb{E}_{s \sim d_\pi(s)} \mathbb{E}_{\pi(a|s)} f(s, a) \end{aligned} \quad \blacksquare$$

Итак, мы научились переписывать мат.ожидание по траекториям в другом виде. В будущем мы будем постоянно пользоваться формулой (5.11) для разных $f(s, a)$, поэтому полезно запомнить эту альтернативную форму записи. Например, мы можем получить применить этот результат к нашему функционалу:

Утверждение 55: Оптимизируемый функционал (1.5) можно записать в таком виде:

$$J(\pi) = \frac{1}{1 - \gamma} \mathbb{E}_{s \sim d_\pi(s)} \mathbb{E}_{\pi(a|s)} r(s, a)$$

Интерпретация у полученного результата может быть такая: нам не столько существенна последовательность принимаемых решений, сколько частоты посещений «хороших» состояний с высокой наградой.

Теорема 57 позволяет переписать и формулу градиента:

Утверждение 56: Выражение для градиента оптимизируемого функционала можно записать следующим образом:

$$\nabla_\theta J(\pi) = \frac{1}{1 - \gamma} \mathbb{E}_{d_\pi(s)} \mathbb{E}_{\pi(a|s)} \nabla_\theta \log \pi_\theta(a | s) Q^\pi(s, a) \quad (5.12)$$

Доказательство. Применить теорему 57 для $f(s, a) := \nabla_\theta \log \pi_\theta(a | s) Q^\pi(s, a)$. ■

Итак, множитель γ^t в формуле (5.4) имеет смысл «дисконтирования частот посещения состояний». Для нас это представляет собой, мягко говоря, очень странную проблему. Если мы рассмотрим формулу градиента $J(\theta)$, то есть зафиксируем начальное состояние в наших траекториях, то все слагаемые, соответствующие состояниям, встречающимся в эпизодах только после, условно, 100-го шага, будут домножены на γ^{100} . То есть одни слагаемые в нашем оптимизируемом функционале имеют один масштаб, а другие — домножаются на близкий к нулю γ^{100} , совершенно иной. Градиентная оптимизация с такими функционалами просто не справится: в градиентах будет доминировать информация об оптимизации наших решений в состояниях около начального, имеющих большой вес.

Откуда это лезет? Давайте посмотрим на то, что мы оптимизируем: $J(\theta)$ (1.5). Ну действительно: награда, которую мы получим после сотого шага, дисконтируется на γ^{100} . Мы уже поняли, что для того, чтобы промаксимизировать её, нам всё равно придётся промаксимизировать $V^\pi(s)$ для всех состояний, то есть эти слагаемые «в микро-масштабе» для достижения глобального оптимума тоже должны оптимизироваться, и задача оптимизации поставлена «корректно». Но для градиентной оптимизации такая «форма» просто не подходит.

Сейчас случится неожиданный поворот событий. На практике во всех Policy Gradient методах от дисконтирования частот посещения отказываются. Это означает, что мы заменяем $d_\pi(s)$ на $\mu_\pi(s)$ (5.9):

$$\nabla_\theta J(\pi) \approx \frac{1}{1-\gamma} \mathbb{E}_{\mu_\pi(s)} \mathbb{E}_{\pi(a|s)} \nabla_\theta \log \pi_\theta(a | s) Q^\pi(s, a)$$

В формуле (5.4) это эквивалентно удалению множителя γ^t :

$$\nabla_\theta J(\pi) \approx \mathbb{E}_{\tau \sim \pi} \sum_{t \geq 0} \nabla_\theta \log \pi_\theta(a_t | s_t) Q^\pi(s_t, a_t)$$

Мы вовсе не отказались от дисконтирования вовсе: оно всё ещё сидит внутри оценочной функции и даёт приоритет ближайшей награде перед получением той же награды в будущем, как мы и задумывали изначально.

Итак, при таком соглашении мы можем получить как бы оценку Монте-Карло на градиент:

$$\nabla_\theta J(\pi) \approx \frac{1}{1-\gamma} \mathbb{E}_{a \sim \pi(a|s)} \nabla_\theta \log \pi_\theta(a | s) Q^\pi(s, a), \quad s \sim \mu_\pi(s)$$

Такую оценку можно считать условно несмещённой (с учётом нашего забивания на множитель γ^t и приближённого сэмплирования из $\mu_\pi(s)$), если у нас на руках есть точная (или хотя бы несмещённое) оценка «кредита» $Q^\pi(s, a)$.

5.1.7. Связь с policy improvement

Формула градиента в форме (5.12) даёт ещё одну интересную интерпретацию того, в каком направлении указывает полученная формула градиента. Давайте введём ещё одну *суррогатную функцию* (surrogate objective). Как и суррогатная функция (5.8), это будет ещё один функционал, который имеет в точке текущих значений параметров стратегии π такой же градиент, как и $J(\theta)$:

$$\mathcal{L}_{\tilde{\pi}}(\theta) := \frac{1}{1-\gamma} \mathbb{E}_{d_{\tilde{\pi}}(s)} \mathbb{E}_{a \sim \pi_\theta(a|s)} Q^{\tilde{\pi}}(s, a) \tag{5.13}$$

Эта суррогатная функция, опять же, от двух стратегий: стратегии π_θ , которую мы оптимизируем, и ещё одной стратегии $\tilde{\pi}$. Снова смотрим на эту суррогатную функцию в такой точке θ , что две стратегии совпадают: $\tilde{\pi} \equiv \pi_\theta$; будем «шевелить» θ и смотреть на градиент. То есть теперь мы «заморозили» распределение частот посещений состояний и, как и в прошлый раз, оценочную функцию. Тогда:

Утверждение 57:

$$\nabla_\theta \mathcal{L}_{\tilde{\pi}}(\theta)|_{\tilde{\pi}=\pi_\theta} = \nabla_\theta J(\theta)$$

Доказательство. Поскольку $d_{\tilde{\pi}}(s)$ и $Q^{\tilde{\pi}}(s, a)$ не зависят от θ , то для дифференцирования суррогатной функции достаточно лишь пронести градиент через матожидание по выбору стратегии при помощи REINFORCE:

$$\begin{aligned} \nabla_\theta \frac{1}{1-\gamma} \mathbb{E}_{d_{\tilde{\pi}}(s)} \mathbb{E}_{a \sim \pi_\theta(a|s)} Q^{\tilde{\pi}}(s, a) &= \frac{1}{1-\gamma} \mathbb{E}_{d_{\tilde{\pi}}(s)} \int_{\mathcal{A}} \nabla_\theta \pi_\theta(a | s) Q^{\tilde{\pi}}(s, a) da = \\ &= \{\text{log-derivative trick (2.7)}\} = \frac{1}{1-\gamma} \mathbb{E}_{d_{\tilde{\pi}}(s)} \mathbb{E}_{a \sim \pi_\theta(a|s)} \nabla_\theta \log \pi_\theta(a | s) Q^{\tilde{\pi}}(s, a) \end{aligned}$$

В точке $\theta: \tilde{\pi} = \pi_\theta$ верно, что $d_{\tilde{\pi}}(s) = d_\pi(s)$ и $Q^{\tilde{\pi}}(s, a) = Q^\pi(s, a)$; следовательно, значение градиента в этой точке совпадает со значением формулы (5.12). ■

Это значит, что в текущей точке градиенты указывают туда же, куда и при максимизации $\mathcal{L}_{\tilde{\pi}}(\theta)$ по θ . А куда указывает направление градиентов для неё? Выражение $\mathbb{E}_{a \sim \pi_\theta(a|s)} Q^{\tilde{\pi}}(s, a)$ говорит максимизировать Q-функцию по выбираемым нами действиям: это в чистом виде направление policy improvement-а из теоремы 17!

Но если в табличном сеттинге policy improvement мы делали, так сказать, «жёстко», заменяя целиком стратегию на жадную по действиям, то формула градиента теперь говорит нам, что это лишь направление максимального увеличения $J(\theta)$; после любого шага в этом направлении наша Q-функция тут же, формально, меняется, и в новой точке направление уже должно быть скорректировано.

Второе уточнение, которое дарит нам эта формула, это распределение, из которого должны приходить состояния. В табличном случае мы не знали, в каких состояниях проводить improvement «важнее», теперь же мы видим, что s должно приходить из $d_{\pi}(s)$. Если какое-то состояние посещается текущей стратегией часто, то улучшать стратегию в нём важнее, чем в состояниях, которые мы посещаем редко.

Но ещё это наблюдение, что градиент будущей награды и policy improvement связаны, в частности, даёт одно из оправданий тому, что мы отказались от дисконтирования частот посещения состояний. Мы используем формулу градиента для максимизации $V^\pi(s_0)$. Но, в общем-то, мы хотим максимизировать $V^\pi(s)$ сразу для всех s , и мы можем делать это с некоторыми весами (и эти веса, в целом, наш произвол). Тогда, выходит, частоты появления s в оптимизируемом функционале определяются в том числе этими весами. Другими словами, $d_{\pi}(s)$ указывает на «самое правильное» распределение, из которого должны приходить состояния, которые дадут направление именно максимального увеличения функционала, но теория policy improvement-а подсказывает нам, что теоретически корректно выбрать и любое другое. Даже если мы совсем другое какое-то распределение выберем для сэмплирования состояний s , то функционал

$$\mathbb{E}_s \mathbb{E}_{a \sim \pi_\theta(a|s)} Q^{\tilde{\pi}}(s, a) \rightarrow \max_{\theta}$$

будет давать какое-то направление подъёма, какое-то годное направление оптимизации.

Получается, исходя из этого рассуждения, что нам для использования формулы градиента даже не нужны on-policy сэмплы, и мы можем брать состояния просто из буфера! Можем ли мы так делать? Допустим, мы готовы заменить распределение состояний на произвольное. Всё равно остаётся существенным сэмплировать действия a именно из текущей версии $\pi(a | s)$; значит, брать действие a из буфера нельзя. Следовательно, мы не можем получить $s' \sim p(s' | s, a)$; но если у нас как-то есть на руках для любой пары s, a какое-то значение «кредита» $Q^\pi(s, a)$, то для градиента актёра нам, согласно формуле, больше ничего и не нужно.

Итак, если у нас есть модель Q-функции, мы даже с оговорками можем обучать стратегию с буфера, если будем брать из него только состояния, а матожидание по a (или его Монте-Карло оценку) считать, используя текущую стратегию π . Однако, коли уж мы хотим off-policy алгоритм, то и эту Q-функцию тогда нужно учить с буфера. В итоге, мы получим алгоритм, очень похожий на DQN, со схожими недостатками и преимуществами. Мы обсудим такой подход позже в главе 6, а пока что мы не будем гоняться за возможностью обучаться с буфера и за счёт этого сможем получить некоторые другие полезные преимущества, которые открываются в on-policy режиме.

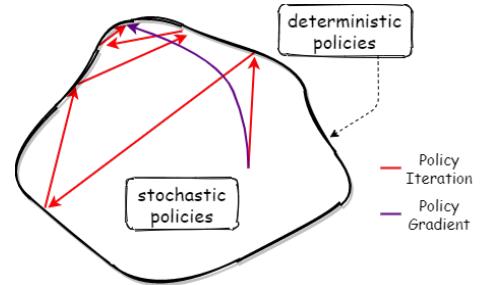
В частности, раз уж формула градиента подсказывает сэмплировать s из частот посещения состояний, то мы будем стараться так делать, стараться моделировать именно оценку градиента: на практике мы возьмём состояния из недисконтированных частот $\mu_{\pi}(s)$, но тем не менее.

5.1.8. Бэйзлайн

При стохастической оптимизации ключевым фактором является дисперсия оценки градиента. Когда мы заменяем матожидания на Монте-Карло оценки, дисперсия увеличивается. Понятно, что замена Q-функции — выинтегрированных будущих наград — на её Монте-Карло оценку в REINFORCE (5.7) повышало дисперсию. Однако, в текущем виде основной источник дисперсии заключается в другом.

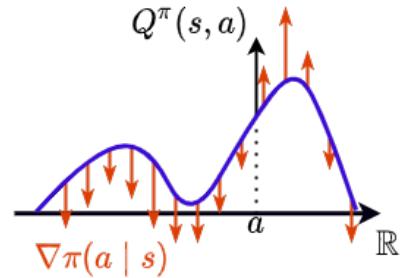
Пример 81: Допустим, у нас два действия $\mathcal{A} = \{0, 1\}$. Мы породили траекторию, в котором выбрали $a = 0$. Выданное значение «кредита доверия» — пусть это точное значение $Q^\pi(s, a = 0)$ — допустим, равно 100. Это значит, что градиент для данной пары s, a указывает на направление изменения параметров, которые увеличивают вероятность $\pi(a = 0 | s)$, и это направление войдёт в итоговую оценку градиентов с весом 100. Но 100 — это много или мало?

Рассмотрим один вес $\theta_i \in \mathbb{R}$ в нашей параметризации стратегии. Без ограничения общности будем считать, что для повышения $\pi(a = 0 | s)$ вес θ_i нужно повышать. Тогда если на следующем шаге оптимизации мы в том же s засэмплировали $a = 1$, то для повышения $\pi(a = 1 | s)$ вес θ_i , очевидно, нужно уменьшать (так как сумма $\pi(a = 0 | s) + \pi(a = 1 | s)$ обязана равняться 1). С каким весом мы будем уменьшать θ_i ? С весом $Q^\pi(s, a = 1)$. Что, если оно равно 1000? На прошлом шаге мы шли в одну сторону с весом 100, на текущем — в другую с весом 1000. За счёт разницы весов мы в среднем движемся в правильную сторону, но



нас постоянно дёргает то в одном направлении, то в другом.

Обобщим описанную в примере ситуацию. Для этого вспомним утверждение (5.6): градиент логарифма правдоподобия в среднем равен нулю. Это значит, что если для данного s мы выдаём некоторое распределение $\pi(a | s)$, для увеличения вероятностей в одной области A нужно данный вес θ_i параметризации увеличивать, а в другой области — уменьшать. В среднем «мagnитуда изменения» равна нулю. Но у нас в Монте-Карло оценке только один сэмпл $a \sim \pi(a | s)$, и для него направление изменения домножится на кредит, на нашу оценку $Q^\pi(s, a)$. Если эта оценка в одной области 100, а в другой 1000 — дисперсия получаемых значений $\nabla_\theta \log \pi_\theta(a | s) Q^\pi(s, a)$ становится колоссальной. Было бы сильно лучше, если бы «кредит» — вес примеров — был в среднем центрирован, и тоже колебался возле нуля. Тогда для «плохих действий» мы правдоподобие этих действий уменьшаем, а для «хороших действий» — увеличиваем, что даже чисто интуитивно логичнее. И для центрирования весов правдоподобия в Policy Gradient методах всегда вводится **бэйзлайн** (baseline), без которого алгоритмы обычно не заведутся.



Утверждение 58: Для произвольной функции $b(s) : \mathcal{S} \rightarrow \mathbb{R}$, называемой бэйзлайном, верно:

$$\nabla_\theta J(\pi) = \frac{1}{1-\gamma} \mathbb{E}_{d_\pi(s)} \mathbb{E}_{\pi(a|s)} \nabla_\theta \log \pi_\theta(a | s) (Q^\pi(s, a) - b(s)) \quad (5.14)$$

Доказательство. Добавленное слагаемое есть ноль в силу формулы (5.6). ■

Это верно для произвольной функции от состояний и становится неверно, если вдруг бэйзлайн начинает зависеть от a . Мы вольны выбрать бэйзлайн произвольно; он не меняет среднего значения оценок градиента, но изменяет дисперсию.

Теорема 58: Бэйзлайном, максимально снижающим дисперсию Монте-Карло оценок формулой градиентов (5.14), является

$$b^*(s) := \frac{\mathbb{E}_a \|\nabla_\theta \log \pi_\theta(a | s)\|_2^2 Q^\pi(s, a)}{\mathbb{E}_a \|\nabla_\theta \log \pi_\theta(a | s)\|_2^2}$$

Доказательство. Рассмотрим одно состояние s и попробуем вычислить оптимальное значение $b(s)$. Пусть для данного состояния m — среднее значение оценки градиента, которое, как мы поняли ранее, не зависит от значения b :

$$m := \mathbb{E}_{a \sim \pi(a|s)} \nabla_\theta \log \pi_\theta(a | s) Q^\pi(s, a)$$

Для нас будет важно, что m — просто какой-то фиксированный вектор той же размерности, что и θ . Распишем дисперсию и будем минимизировать её по b :

$$\mathbb{E}_a \|\nabla_\theta \log \pi_\theta(a | s) (Q^\pi(s, a) - b) - m\|_2^2 \rightarrow \min_b$$

Дифференцируем по b и приравниваем к нулю:

$$2\mathbb{E}_a (\nabla_\theta \log \pi_\theta(a | s) (Q^\pi(s, a) - b) - m)^T (-\nabla_\theta \log \pi_\theta(a | s)) = 0$$

Выделяем норму градиента логарифма правдоподобия:

$$-\mathbb{E}_a \|\nabla_\theta \log \pi_\theta(a | s)\|_2^2 Q^\pi(s, a) + \mathbb{E}_a \|\nabla_\theta \log \pi_\theta(a | s)\|_2^2 b + \mathbb{E}_a m^T (\nabla_\theta \log \pi_\theta(a | s)) = 0 \quad (5.15)$$

Осталось заметить, что третье слагаемое есть ноль. Это обобщение нашей теоремы о бэйзлайне (формулы (5.6)): условно, бэйзлайн может быть свой для каждой компоненты вектора θ , опять же, до тех пор, пока он не зависит от действий. В данном случае m — некоторый фиксированный вектор, одинаковый для всех a ; поэтому, если d — размерность вектора параметров θ , то:

$$\mathbb{E}_a m^T (\nabla_\theta \log \pi_\theta(a | s)) = \mathbb{E}_a \sum_{i=0}^d m_i \nabla_{\theta_i} \log \pi_\theta(a | s) = \sum_{i=0}^d m_i \underbrace{\mathbb{E}_a \nabla_{\theta_i} \log \pi_\theta(a | s)}_{0 \text{ по формуле (5.6)}} = 0$$

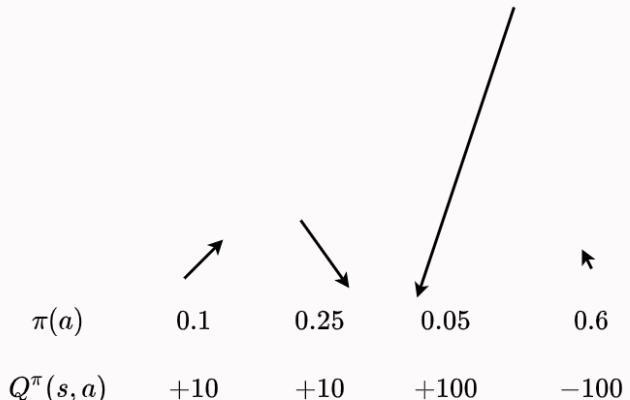
Убирая это нулевое третье слагаемое из (5.15), получаем равенство между первыми двумя:

$$b \mathbb{E}_a \|\nabla_\theta \log \pi_\theta(a | s)\|_2^2 = \mathbb{E}_a \|\nabla_\theta \log \pi_\theta(a | s)\|_2^2 Q^\pi(s, a)$$

Выражая из него b , получаем доказываемое.

Пример 82: Представить себе интуицию этой формулы можно следующим образом. Рассмотрим одно s . В нём мы сэмплируем одно какое-то действие a с вероятностями $\pi(a | s)$. Допустим, действий четыре: засэмплировалось какое-то одно. Для каждого действия есть свой градиент, показывающий, как повысить вероятность выбора этого действия $\nabla_\theta \log \pi_\theta(a | s)$. У него есть некоторая норма $\|\nabla_\theta \log \pi_\theta(a | s)\|_2^2$ — его длина. Мы в качестве градиента берём такой вектор, отмасштабированный на $Q^\pi(s, a)$ — произвольное, в общем-то, число. Как снизить дисперсию получающихся градиентов?

Мы можем вычесть из значения «кредитов» $Q^\pi(s, a)$ какое-то общее для всех действий число b . И формула говорит: возьмите среднее значение $Q^\pi(s, a)$ с учётом двух вещей: что, во-первых, какие-то вектора мелькают чаще других, и поэтому нужно брать среднее с учётом вероятностей $\pi(a | s)$, а во-вторых, какие-то вектора длиннее по норме, чем другие, и поэтому также нужно перевзвесить значения $Q^\pi(s, a)$ пропорционально норме градиента $\|\nabla_\theta \log \pi_\theta(a | s)\|_2^2$. Второе указывает сделать бэйзлайн таким, чтобы отцентрированный «кредит» для самых «длинных» векторов был поближе к нулю.



Практическая ценность результата невысока. Знать норму градиента для всех действий a вычислитель но будет труднозатратно даже в дискретных пространствах действий. Поэтому мы воспользуемся небольшим предположением: мы предположим, что норма градиента примерно равна для всех действий. Тогда:

$$b^*(s) = \frac{\mathbb{E}_a \|\nabla_\theta \log \pi_\theta(a | s)\|_2^2 Q^\pi(s, a)}{\mathbb{E}_a \|\nabla_\theta \log \pi_\theta(a | s)\|_2^2} = \{\|\nabla_\theta \log \pi_\theta(a | s)\|_2^2 \approx \text{const}(a)\} \approx \mathbb{E}_a Q^\pi(s, a) = V^\pi(s)$$

Эта аппроксимация довольно интуитивная: по логике, для дисперсии хорошо, если значения функции $Q^\pi(s, a) - b(s)$ вертятся вокруг нуля, то есть в среднем дают ноль, и поэтому хорошим (но неоптимальным) бэйзлайном будет

$$\mathbb{E}_{a \sim \pi(a|s)} (Q^\pi(s, a) - b(s)) := 0 \Rightarrow b(s) := \mathbb{E}_{a \sim \pi(a|s)} Q^\pi(s, a) = V^\pi(s)$$

Итак, всюду далее будем в качестве бэйзлайна использовать $b(s) := V^\pi(s)$. Подставляя и вспоминая определение (3.19) Advantage-функции, получаем:

Утверждение 59:

$$\nabla_\theta J(\pi) = \frac{1}{1-\gamma} \mathbb{E}_{d_\pi(s)} \mathbb{E}_{\pi(a|s)} \nabla_\theta \log \pi_\theta(a | s) A^\pi(s, a) \quad (5.16)$$

Таким образом, «кредит», который мы выдаём каждой паре s, a , будет являться оценкой Advantage, и состоять из двух слагаемых: оценки Q-функции и бэйзлайна. Именно поэтому этот вес и называется «кредитом»: задача оценки Advantage и есть в точности тот самый credit assigment, который мы обсуждали в главе 3.5!

§5.2. Схемы «Актёр-критик»

5.2.1. Введение критика

Мы хотели научиться оптимизировать параметры стратегии при помощи формулы градиента, не доигрывая эпизоды до конца. Мы уже поняли, что ожидание по траекториям не представляет для нас проблемы. Тогда осталось лишь придумать, как, не доигрывая эпизоды до конца, проводить credit assignment, то есть определять для каждой пары s, a оценку Advantage-функции.

Раз знание функции Q^π позволит обучаться, не доигрывая эпизоды до конца, а функцию в готовом виде нам никто не даст, то возникает логичная идея — аппроксимировать её. Итак, введём вторую сетку, которая будет «оценивать» наши собственные решения — **критика** (critic). Нейросеть, моделирующую стратегию, соответственно будем называть **актёром** (actor), и такие алгоритмы, в которых обучается как модель критика, так и модель актёра, называются **Actor-Critic**.

Здесь возникает принципиальный момент: мы не умеем обучать модели оценочных функций Q^π или V^π , так чтобы они оценивали будущую награду несмешённо («выдавали в среднем правильный ответ»). Что это влечёт? При смешённой оценке Q-функции любые гарантии на несмешённость градиентов мгновенно теряются. Единственный способ оценивать Q-функцию несмешённо — Монте-Карло, но она требует полных эпизодов и имеет более высокую дисперсию. Любое замешивание нейросетевой аппроксимации в оценку — и мы сразу теряем несмешённость оценок на градиент, и вместе с ними — любые гарантии на сходимость стохастической оптимизации к локальному оптимуму или даже вообще хоть куда-нибудь!

В машинном обучении в любых задачах оптимизации всегда необходимо оценивать градиент оптимизируемого функционала несмешённо, и важной необычной особенностью Policy Gradient методов в RL является тот факт, что в них внезапно используются именно смешённые оценки градиента. Надежда на работоспособность алгоритма после замены значения $Q^\pi(s, a)$ на смешённую оценку связана с тем, что формула градиента говорит проводить policy improvement во встречаемых состояниях (см. физический смысл суррогатной функции (5.13)). Коли градиент есть просто policy improvement, то мы знаем из общей идеи алгоритма 9 Generalized Policy Iteration, что для улучшения политики вовсе не обязательно использовать идеальную Q^π , достаточно любой аппроксимации критика $Q(s, a) \approx Q^\pi(s, a)$, которая параллельным процессом движется в сторону идеальной Q^π для текущей стратегии π . Фактически, все Actor-Critic методы моделируют именно эту идею.

Следующий существенный момент заключается в том, что в качестве критика обычно учат именно V-функцию. Во-первых, если бы мы учили модель Q-функции и подставили бы её, то градиент

$$\nabla_\theta \log \pi_\theta(a | s) Q(s, a)$$

направил бы нашу стратегию в $\underset{a}{\operatorname{argmax}} Q(s, a)$. Помимо того, что это выродило бы стратегию, это бы привело к тому, что качество обучения актёра выродилось бы в качество обучения критика: пока модель $Q(s, a)$ не похожа на истинную $Q^\pi(s, a)$, у нас нет надежды, что актёр выучит хорошую стратегию.



Можем ли мы в качестве критика напрямую учить сетку, выдающую аппроксимацию $A_\phi(s, a) \approx A^\pi(s, a)$, и использовать её выход в качестве нашей оценки $\Psi(s, a)$? Это не очень удобно хотя бы потому, что в отличие от Q-функций и V-функций, advantage — не абстрактная произвольная функция: она должна подчиняться теореме 22. При этом восстановить по advantage-у Q-функцию без знания V-функции нельзя, а значит, для advantage не получится записать аналога уравнения Беллмана, использующего только advantage-функции.

Возможность не обучать сложную Q^* является одним из преимуществ подхода прямой оптимизации $J(\theta)$. В DQN было обязательным учить именно Q-функцию, так как, во-первых, мы хотели выводить из неё оптимальную стратегию, во-вторых, для уравнения оптимальности Беллмана для оптимальной V-функции фокус с регрессией не прокатил бы — там матожидание стоит внутри оператора взятия максимума. Сейчас же мы из соображений эффективности алгоритма (желания не играть полные эпизоды и задачи снижения дисперсии оценок) не сможем обойтись совсем без обучения каких-либо оценочных функций, но нам хватит лишь $V_\phi(s) \approx V^\pi$, поскольку оценить Q-функцию мы можем хотя бы так:

$$Q^\pi(s, a) = r(s, a) + \gamma \mathbb{E}_{s'} V^\pi(s') \approx r(s, a) + \gamma V_\phi(s'), \quad s' \sim p(s' | s, a) \quad (5.17)$$

Иначе говоря, если у нас есть приближение V-функции, то мы можем использовать её как для бэйзлайна, так и для оценки Q-функции. Важно, что V-функция намного проще, чем Q-функция: ей не нужно дифференцировать между действиями, достаточно лишь понимать, какие области пространства состояний — хорошие, а какие плохие. И поэтому раз можно обойтись ей, то почему бы так не сделать и не упростить критику задачу.

5.2.2. Bias-variance trade-off

Обсудим сначала, как критик $V_\phi(s) \approx V^\pi(s)$ с параметрами ϕ будет использоваться в формуле градиента по параметрам стратегии (5.16). Мы собираемся вместо честного advantage подставить некоторую его оценку (advantage estimator) и провести таким образом credit assigment:

$$\nabla_\theta J(\pi) \approx \frac{1}{1 - \gamma} \mathbb{E}_{d_\pi(s)} \mathbb{E}_{a \sim \pi(a|s)} \nabla_\theta \log \pi_\theta(a | s) \underbrace{\Psi(s, a)}_{\approx A^\pi(s, a)} \quad (5.18)$$

Мы столкнулись ровно с тем же самым bias-variance trade-off, который мы обсуждали 3.5, который как раз и сводится к оцениванию Advantage и определению того, какие действия хорошие или плохие. Только теперь, в контексте policy gradient, речь напрямую идёт о дисперсии и смещении оценок градиента. Мы уже встречались с двумя самыми «крайними» вариантами выбора функции $\Psi(s, a)$: Монте-Карло и одношаговая оценка через V-функцию (5.17):

$\Psi(s, a)$	Дисперсия	Смещение
$R_t - V_\phi(s)$	высокая	нету
$r(s, a) + \gamma V_\phi(s') - V_\phi(s)$	низкая	большое

В этой таблице речь идёт именно о дисперсии и смещении оценки градиента $J(\pi)$ при использованной аппроксимации! А то есть, в первом случае оценка Q-функции несмешённая, оценка V-функции смешённая, но поскольку в качестве бэйзлайна в силу (5.6) может использоваться совершенно произвольная функция от состояний, совершенно несущественно, насколько наша аппроксимация $V^\pi(s)$ вообще похожа на истинную оценочную

функцию. Да, если наша аппроксимация V-функции будет неточной, мы, вероятно, не так сильно съём дисперсию оценок градиента, как могли бы, но ровно на этом недостатки использования смещённой аппроксимации V-функции в качестве бэйзлайна заканчиваются.

Во втором же случае, аппроксимация $V^\pi(s')$ используется для оценки Q-функции и вызывает смещение в том числе в оценке градиента. Дисперсия же снижается, поскольку в Q-функции аккумулированы мат.ожидания по всему хвосту траектории; она в обоих случаях существенно ниже, чем до введения бэйзлайна, но замена Монте-Карло оценки на бутстррапированную оценку снижает её ещё сильнее.

Вспомним, как можно интерполировать между этими крайними вариантами. Для начала, у нас есть ещё целое семейство промежуточных вариантов — многошаговых (multi-step) оценок Q-функции, использующих N-шаговое уравнение Беллмана (3.25):

$$Q^\pi(s, a) \approx \sum_{t=0}^{N-1} \gamma^t r^{(t)} + \gamma^N V_\phi(s^{(N)})$$

Тогда мы пользуемся для credit assigment-a N-шаговой оценкой Advantage, или N-шаговой временной разностью (3.41).

$$\Psi_{(N)}(s, a) := \sum_{t=0}^{N-1} \gamma^t r^{(t)} + \gamma^N V_\phi(s^{(N)}) - V_\phi(s)$$

С ростом N дисперсия такой оценки увеличивается: всё больший фрагмент траектории мы оцениваем по Монте-Карло, нам становятся нужны сэмплы $a_{t+1} \sim \pi(a_{t+1} | s_{t=1}), s_{t+2} \sim \pi(s_{t+2} | s_{t+1}, a_{t+1}), \dots, s_{t+N} \sim \pi(s_{t+N} | s_{t+N-1}, a_{t+N-1})$, а при $N \rightarrow \infty$ оценка в пределе переходит в полную Монте-Карло оценку оставшейся награды, где дисперсия большая, но зато исчезает смещение в силу отсутствия смещённой аппроксимации награды за хвост траектории.

Также с ростом N мы начинаем всё меньше опираться на модель критика. Используя сэмплы наград, мы ценой увеличения дисперсии напрямую учим связь между хорошими действиями и высоким откликом от среды, меньше опираясь на промежуточный этап в виде выучивания оценочной функции. Это значит, что нам не нужно будет дожидаться, пока наш критик идеально обучиться: в оценку Advantage попадает сигнал в том числе из далёкого будущего при больших N , и актёр поймёт, что удачно совершённое действие надо совершать чаще. Математически это можно объяснить тем, что, увеличивая N , мы снижаем смещение наших оценок градиента. Значит, нам в целом даже и не потребуется, чтобы критик оценивал состояния с высокой точностью, поскольку главное, чтобы в итоге получалась более-менее адекватная оценка совершённых нашей стратегией действий.

Trade-off заключается в том, что чем дальше в будущее мы заглядываем, тем выше дисперсия этих оценок; помимо этого, для заглядывания в будущее на N шагов нужно же иметь это самое будущее, то есть из каждой параллельно запущенной среды понадобится собрать для очередного мини-батча не по одному сэмплу, а собрать целый длинный фрагмент траектории. Поэтому, регулируя длину оценок, мы «размениваем» смещение на дисперсию, истина, как всегда, где-то посередине.

Возможность разрешать bias-variance trade-off и выбирать какую-то оценку с промежуточным смещением и дисперсией — чуть ли не главное преимущество on-policy режима обучения. Напомним, что сэмплы фрагментов траекторий из буфера получить не удастся (действия должны генерироваться из текущей стратегии), и использование многошаговых оценок в off-policy режиме было невозможно.

Пока что в нашем алгоритме роллауты непрактично делать сильно длинными. Дело в том, что пары s, a из длинных роллаутов будут сильно коррелированы, что потребуется перебивать числом параллельно запущенных сред, а тогда при увеличении длины роллаута начинает раздуваться размер мини-батча. Потом собранные переходы будут использованы всего для одного шага градиентного подъёма, и переиспользовать их будет нельзя; это расточительно и поэтому большой размер мини-батча невыгоден. Поэтому пока можно условно сказать, что самая «выгодная» оценка Advantage-а для не очень длинных роллаутов — оценка максимальной длины: для s_t мы можем построить N -шаговую оценку, её и возьмём; для s_{t+1} уже не более чем $N - 1$ -шаговую; наконец, для s_{t+N-1} нам доступна лишь одношаговая оценка, просто потому что никаких сэмплов после s_{t+N} мы ещё не получали.

Определение 75: Для пар s_t, a_t из роллаута $s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_N$ длины N будем называть *оценкой максимальной длины* (max trace estimation) оценку с максимальным заглядыванием в будущее: для Q-функции

$$y^{\text{MaxTrace}}(s_t, a_t) := \sum_{\hat{t}=t}^{N-1} \gamma^{\hat{t}-t} r_{\hat{t}} + \gamma^{N-t} V^\pi(s_N), \quad (5.19)$$

для Advantage функции, соответственно:

$$\Psi^{\text{MaxTrace}}(s_t, a_t) := y^{\text{MaxTrace}}(s_t, a_t) - V^\pi(s_t)$$

Заметим, что мы вовсе не обязаны использовать для всех пар s, a оценку одной и той же длины N . То есть мы не должны брать для s_t, a_t из N -шагового роллаута N -шаговую оценку, а остальные пары s, a из роллаута не использовать в обучении лишь потому, что для них N -шаговая оценка невозможна; вместо этого для них следует

просто использовать ту многошаговую оценку, которая доступна. Это полностью корректно, поскольку любая N -шаговая оценка является оценкой Advantage-а для данного слагаемого в нашей формуле градиента. Именно это и говорит оценка максимальной длины: если мы собрали роллаут $s_0, a_0, s_1, a_1 \dots s_5$ длины 5, то одну пару s_4, a_4 , самую последнюю, нам придётся оценить одншаговой оценкой (поскольку для неё известен сэмпл лишь следующего состояния s_5 и только, никаких альтернатив здесь придумать не получится), предпоследнюю пару s_3, a_3 — двухшаговой, и так далее. То есть «в среднем» длина оценок будет очень маленькой, такая оценка с точки зрения bias-variance скорее смещена, чем имеет большую дисперсию.

5.2.3. Generalized Advantage Estimation (GAE)

Пока N не так велико, чтобы дисперсия раздувалась, оценка максимальной длины — самое разумное решение trade-off-а, и о более умном решении думать не нужно. Однако, впоследствии мы столкнёмся с ситуацией, что оп-policy алгоритмы будут собирать достаточно длинные роллауты (порядка тысячи шагов), и тогда брать оценку наибольшей длины уже будет неразумно; с этой же проблемой можно столкнуться и в контексте обсуждаемой Actor-Critic схемы, если длина собираемых роллаутов достаточно большая.

Решение дилеммы bias-variance trade-off подсказывает теория TD(λ) оценки из главы 3.5. Нужно применить формулу (3.51) и просто заансамблировать N -шаговые оценки разной длины:

Определение 76: *GAE-оценкой* Advantage-функции называется ансамбль многошаговых оценок, где оценка длины N (3.41) берётся с весом λ^{N-1} , где $\lambda \in (0, 1)$ — гиперпараметр:

$$\Psi_{\text{GAE}}(s, a) := (1 - \lambda) \sum_{N>0} \lambda^{N-1} \Psi_{(N)}(s, a) \quad (5.20)$$

Как мы помним, при $\lambda \rightarrow 0$ такая GAE-оценка соответствует одношаговой оценке; при $\lambda = 1$ GAE-оценка соответствует Монте-Карло оценке Q-функции, которой мы фактически воспользовались, например, в REINFORCE.



Как и при дисконтировании, геометрическая прогрессия затухает очень быстро; это означает, что $\lambda \approx 0.9$ больше предпочитает короткие оценки длинным. Поэтому часто λ всё-таки близка к 1, типичное значение — 0.95.

В текущем виде в формуле суммируются все N -шаговые оценки вплоть до конца эпизода. В реальности собранные роллауты могут прерваться в середине эпизода: допустим, для данной пары s, a через M шагов роллаут «обрывается». Тогда на практике используется чуть-чуть другим определением GAE-оценки: если мы знаем $s^{(M)}$, но после этого эпизод ещё не доигран до конца, мы пользуемся формулой (3.51) и оставляем от суммы только «доступные» слагаемые:

$$\Psi_{\text{GAE}}(s, a) := \sum_{t \geq 0}^{M-1} \gamma^t \lambda^t \Psi_{(1)}(s^{(t)}, a^{(t)}) \quad (5.21)$$

Напомним, что это корректно, поскольку соответствует просто занулению следа на M -ом шаге, или, что тоже самое, ансамблированию (взятию выпуклой комбинации) первых M многошаговых оценок, где веса «пропавших» слишком длинных оценок просто перекладываются в вес самой длинной доступной M -шаговой оценки:

Утверждение 60: Формула (5.21) эквивалентна следующему ансамблю N -шаговых оценок:

$$\Psi_{\text{GAE}}(s, a) = (1 - \lambda) \sum_{N>0}^{M-1} \lambda^{N-1} \Psi_{(N)}(s, a) + \lambda^{M-1} \Psi_{(M)}(s, a)$$

Доказательство. Следует из доказательства теоремы 32. ■

В такой «обрезанной» оценке $\lambda = 1$ соответствует оценке максимальной длины (75), а $\lambda = 0$ всё ещё даст одношаговую оценку.

В коде формула (5.21) очень удобна для рекурсивного подсчёта оценки; также для практического алгоритма осталось учесть флаги `donet`. Формулы подсчёта GAE-оценки для всех пар (s, a) из роллаута $s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_N$ приобретают такой вид:

$$\begin{aligned} \Psi_{\text{GAE}}(s_{N-1}, a_{N-1}) &:= \Psi_{(1)}(s_{N-1}, a_{N-1}) \\ \Psi_{\text{GAE}}(s_{N-2}, a_{N-2}) &:= \Psi_{(1)}(s_{N-2}, a_{N-2}) + \gamma \lambda (1 - \text{done}_{N-2}) \Psi_{\text{GAE}}(s_{N-1}, a_{N-1}) \\ &\vdots \\ \Psi_{\text{GAE}}(s_0, a_0) &:= \Psi_{(1)}(s_0, a_0) + \gamma \lambda (1 - \text{done}_0) \Psi_{\text{GAE}}(s_1, a_1) \end{aligned}$$

Заметим, что эти формулы очень похожи на расчёт кумулятивной награды за эпизод, где «наградой за шаг» выступает $\Psi_{(1)}(s, a)$. В среднем наши оценки Advantage должны быть равны нулю, и, если наша аппроксимация этому не удовлетворяет, мы получаем направление корректировки стратегии.



Теория говорит, что любые off-policy алгоритмы должны быть более эффективны в плане числа используемых сэмплов, чем on-policy, но на практике может оказаться так, что продвинутые policy gradient алгоритмы, которые мы обсудим позднее, обойдут DQN-подобные алгоритмы из главы 4 и по sample efficiency. Вероятно, это происходит в ситуациях, когда DQN страдает от проблемы распространения сигнала. При использовании GAE замешивание далёких будущих наград в таргеты позволяет справляться с этой проблемой; если награда разреженная, то имеет смысл выставлять λ ближе к единице, если плотная — ближе к, допустим, 0.9. Распространённый дефолтный вариант 0.95 можно рассматривать как отчасти «универсальный». Если же награда информативная, на каждом шаге поступает какой-то информативный сигнал, то осмыслены и одношаговые обновления; и в таких случаях off-policy алгоритмы на основе value-based подхода скорее всего окажутся более эффективными.

5.2.4. Обучение критика

Как обучать критика $V_\phi(s) \approx V^\pi$? Воспользуемся идеей перехода к регрессии, которую мы обсуждали раньше в контексте DQN (раздел 4.1.2). Нам нужно просто решать методом простой итерации уравнение Беллмана (3.3):

$$V_{\phi_{k+1}}(s) \leftarrow \mathbb{E}_a [r + \gamma \mathbb{E}_{s'} V_{\phi_k}(s')]$$

Мы можем получить несмешённую оценку правой части $y := r + \gamma V_{\phi_k}(s')$ и с таким таргетом минимизировать MSE. Однако, давайте воспользуемся преимуществами on-policy режима и поймём, что мы можем поступить точно также, как с оценкой Q-функции в формуле градиента: решать многошаговое уравнение Беллмана (3.25) вместо одношагового. Например, можно выбрать любое N -шаговое уравнение и строить целевую переменную как

$$y := r + \gamma r' + \gamma^2 r'' + \cdots + \gamma^N V_{\phi_k}(s^{(N)}) \quad (5.22)$$

Конечно же, определение того, насколько далеко в будущее заглядывать при построении таргета — это снова всё тот же самый bias-variance trade-off, и очередное ключевое преимущество on-policy подхода — разрешать его в том числе при обучении критика.

В чём заключается bias-variance trade-off при обучении критика? С ростом N таргет (5.22) в задаче регрессии становится всё более и более шумным, зато мы быстрее распространяем сигнал и меньше опираемся в таргете на свою же собственную аппроксимацию. Это позволяет бороться с проблемой накапливающейся ошибки, от которой страдают off-policy алгоритмы вроде DQN. В пределе — при $N = +\infty$ — мы решаем задачу регрессии, где целевая переменная есть reward-to-go, и начинаем учить V-функцию просто по определению (3.2). Такая задача регрессии уже является самой обычной задачей регрессии из машинного обучения, целевая переменная будет являться «ground truth»: именно теми значениями, среднее которых мы и хотим выучить. Но такая задача регрессии будет обладать очень шумными целевыми переменными, плюс для сбора таких данных понадобится, опять же, полные эпизоды играть.

Мы можем для оценки Q-функции для обучения политики и для построения целевой переменной для критика использовать разные подходы (скажем, оценки разной длины), но особого смысла в этом немного: хороший вариант для одного будет хорошим вариантом и для другого. Соответственно, можно считать решение trade-off одинаковым для актёра и критика. Тогда если мы оцениваем Advantage как

$$\Psi(s, a) = y - V_\phi(s),$$

где y — некоторая оценка Q-функции, то y же является и таргетом для V-функции, и наоборот. Используя функцию потерь MSE с таким таргетом, мы как раз и учим среднее значение наших оценок Q-функции, то есть бэйзлайн.

Конечно же, мы можем использовать и GAE-оценку (5.21) Advantage, достаточно «убрать бэйзлайн»:

$$Q^\pi(s, a) = A^\pi(s, a) + V^\pi(s) \approx \Psi_{\text{GAE}}(s, a) + V_\phi(s)$$

При этом мы как бы будем решать «заансамблированные» N-шаговые уравнения Беллмана для V-функции:

Утверждение 61: Таргет $\Psi_{\text{GAE}}(s, a) + V_\phi(s)$ является несмешённой оценкой правой части «ансамбля» уравнений Беллмана:

$$V_\phi(s) = (1 - \lambda) \sum_{N>0} \lambda^{N-1} [\mathfrak{B}^N V_\phi](s),$$

где \mathfrak{B} — оператор Беллмана для V-функции (3.23).

Доказательство. По определению, поскольку $\Psi_{(N)}(s, a) + V(s)$ является несмешённой оценкой правой

части N -шагового уравнения Беллмана (т. е. несменённой оценкой $[\mathfrak{B}^N V^\pi](s)$), а

$$(1 - \lambda) \sum_{N>0} \lambda^{N-1} (\Psi_{(N)}(s, a) + V(s)) = \Psi_{\text{GAE}}(s, a) + V(s)$$

по определению (5.20) GAE-оценки. ■

Брать для обучения критика набор выходных состояний s мы можем откуда угодно. Поэтому для удобства будем брать $s \sim \mu_\pi(s)$, чтобы можно было использовать для обучения критика и актёра один и тот же мини-батч. Итого получаем следующее: делаем несколько шагов взаимодействия со средой, собирая таким образом роллаут некоторой длины N ; считаем для каждой пары s, a некоторую оценку Q-функции $y(s, a)$, например, оценку максимальной длины (5.19); оцениваем Advantage каждой пары как $\Psi(s, a) := y(s, a) - V_\phi(s)$; далее по Монте-Карло оцениваем градиент по параметрам стратегии

$$\nabla_\theta J(\pi) \approx \frac{1}{N} \sum_{s,a} \nabla_\theta \log \pi_\theta(a | s) \Psi(s, a)$$

и градиент для оптимизации критика (допустим, критик — Q-функция):

$$\text{Loss}^{\text{critic}}(\phi) = \frac{1}{N} \sum_{s,a} (y(s, a) - V_\phi(s))^2$$

Естественно, для декорреляции нужно собрать несколько независимых роллаутов из параллельно запущенных сред.



Когда состояния представлены в виде картинок, понятно, что и критику, и актёру нужны примерно одни и те же фичи с изображений (положение одних и тех же распознанных объектов). Поэтому кажется, что если критик и актёр будут двумя разными сетками, их первые слои будут обучаться одному и тому же. Логично для ускорения обучения объединить **экстрактор фич** (feature extractor) для критика и актёра и сделать им просто свои индивидуальные головы. Конечно, тогда нужно обучать всю сеть синхронно, но мы специально для этого считаем градиенты для актёра и критика по одному и тому же мини-батчу. Есть у такого **общего позвоночника** (shared backbone) и свои минусы: лоссы придётся отмасштабировать при помощи скалярного гиперпараметра α так, чтобы одна из голов не забивала градиентами другую:

$$\text{Loss}^{\text{ActorCritic}}(\theta) = \text{Loss}^{\text{actor}}(\theta) + \alpha \text{Loss}^{\text{critic}}(\theta)$$

5.2.5. Advantage Actor-Critic (A2C)

Мы собрали стандартную схему Advantage Actor Critic (A2C): алгоритма, который работает в чистом виде on-policy режиме. Из-за того, что роллауты в этом алгоритме не очень длинные, используются оценки максимальной длины (или, что тоже самое, GAE с $\lambda = 1$).

Алгоритм 20: Advantage Actor-Critic (A2C)

Гиперпараметры: M — количество параллельных сред, N — длина роллаутов, $V_\phi(s)$ — нейросеть с параметрами ϕ , $\pi_\theta(a | s)$ — нейросеть для стратегии с параметрами θ , α — коэф. масштабирования лосса критика, SGD оптимизатор.

Инициализировать θ, ϕ

На каждом шаге:

1. в каждой параллельной среде собрать роллаут длины N , используя стратегию π_θ :

$$s_0, a_0, r_0, s_1, \dots, s_N$$

2. для каждой пары s_t, a_t из каждого роллаута посчитать оценку Q-функции максимальной длины, игнорируя зависимость оценки от ϕ :

$$Q(s_t, a_t) := \sum_{\hat{t}=t}^{N-1} \gamma^{\hat{t}-t} r_{\hat{t}} + \gamma^{N-t} V_\phi(s_N)$$

3. вычислить лосс критика:

$$\text{Loss}^{\text{critic}}(\phi) := \frac{1}{MN} \sum_{s_t, a_t} (Q(s_t, a_t) - V_\phi(s_t))^2$$

4. делаем шаг градиентного спуска по ϕ , используя $\nabla_\phi \text{Loss}^{\text{critic}}(\phi)$

5. вычислить градиент для актёра:

$$\nabla_\theta^{\text{actor}} := \frac{1}{MN} \sum_{s_t, a_t} \nabla_\theta \log \pi_\theta(a_t | s_t) (Q(s_t, a_t) - V_\phi(s_t))$$

6. сделать шаг градиентного подъёма по θ , используя $\nabla_\theta^{\text{actor}}$



Для того, чтобы поощрить исследование среды, к суррогатной функции потерь актёра добавляют ещё одно слагаемое, «регуляризатор», поощряющий высокую энтропию. Обычно, энтропию распределения $\pi_\theta(\cdot | s_t)$ для выбранной параметризации распределения можно посчитать аналитически и дифференцировать по θ . Естественно, этот «дополнительный лосс» тоже нужно грамотно взвесить на скалярный коэффициент. Позже в разделе 6.2.1 мы встретимся с постановкой задачи Maximum Entropy RL, в рамках которой появление этого слагаемого можно объяснить небольшим изменением в самом оптимизируемом функционале.



Известно, что Policy Gradient алгоритмы особенно чувствительны к инициализации нейросетей. Рекомендуют ортогональную инициализацию слоёв. Также крайне существенна обрезка градиентов, которая защищает от взрывов в градиентной оптимизации. При обучении policy gradient алгоритмов важно логгировать норму градиентов; большое значение нормы сигнализирует о неудачном подборе параметров.

Пример 83: Перерыв на [чтение комиксов.](#)

Сравним A2C с value-based алгоритмами на основе DQN. Когда мы моделировали Value Iteration, мы целиком и полностью «полагались» на этап оценивания стратегии, то есть на обучение критика: модели актёра в явном виде даже не было в алгоритме, поскольку текущая стратегия всё время считалась жаждой по отношению к текущему критику. Это означало, что качество стратегии упиралось в качество критика: пока Q-функция не научится адекватно приближать истинную Q^* , стратегия хорошей не будет.

Достаточно интересно, что идея model-free алгоритмов, отказывающихся обучать модель функции переходов в том числе из соображений end-to-end схемы, пришла к тому, что мы иногда сводимся к обучению оценочных функций — промежуточных величин, вместо того, чтобы напрямую матчить состояния и действия, понимать, какие действия стоит выбирать чаще других. Вообще, интуиция подсказывает, что обучать актёра проще, чем оценочную функцию²: в реальных задачах человек редко когда думает в терминах будущей награды.

Пример 84: Представьте, что вы стоите перед кофеваркой и у вас есть два действия: получить кофе и не получить. Вы прекрасно знаете, какое действие лучше другого, но при этом не оцениваете, сколько кофе вы сможете выпить в будущем при условии, например, что сейчас вы выберете первый или второй вариант: то есть не строите в явном виде прогноз значения оценочной функции.

И формула градиента, идея policy gradient методов, как раз предоставляет возможность обучать актёра напрямую, минуя промежуточный этап в виде критика: так, в алгоритме 19 REINFORCE мы могли использовать Монте-Карло оценки Q-функции и обходиться без обучения в явном виде модели критика, то есть полностью опираться на этап policy improvement-a. Поэтому policy gradient алгоритмы ещё иногда называют *policy-based*. Таким образом, DQN и REINFORCE — это два «крайних случая», первый алгоритм полностью опирается на критика, а второй алгоритм — на актёра. Только у первого недостатки компенсируются возможностью обучаться в off-policy режиме и использовать реплей буфер, а вот недостатки алгоритма REINFORCE — высокая дисперсия и необходимость играть целые эпизоды — не имеют аналогичного противовеса.

Важно, что в Policy Gradient алгоритмах мы за счёт использования метода REINFORCE при расчёте градиента можем заменять значение Q-функции (необходимую для улучшения политики) на какую-то заглядывающую в будущее оценку. И насколько сильно при обучении актёра опираться на критика («насколько далеко в будущее заглядывать») — это и есть bias-variance trade-off, который в on-policy алгоритмах возможно разрешать. За счёт

²есть, однако, и ряд теоретических наблюдений, которые говорят, что эти задачи скорее эквивалентны по сложности. На это указывают формулы градиентов для обучения критика и актёра: так, для актёра мы идём по градиенту

$$\nabla_\theta \log \pi_\theta(a | s) \Psi(s, a),$$

а для критика мы идём по градиенту (продифференцируйте MSE, чтобы убедиться в этом):

$$\nabla_\phi V_\phi(s) \Psi(s, a)$$

Как видно, это как будто бы один и тот же градиент: он просто говорит увеличивать соответствующий выход нейронной сети, а кредит предоставляет скаляр, сообщающий масштаб изменения и, главное, знак.

этого A2C в отличие от DQN может использовать и в качестве таргета для обучения критика, и в качестве оценки для обучения Q-функции GAE-оценку (5.21). Да, пока что в A2C роллауты (зачастую) получаются слишком короткими, и GAE ансамбль «бедный», приходится $\lambda = 1$ использовать, но главное, что есть такая возможность технически, и даже короткие роллауты уже позволяют существенно справиться с проблемой распространения сигнала: это помогает и критику лучше учиться, и актёр может не полностью на критика опираться. В частности, нам не нужна модель Q-функции, достаточно более простой V-функции. Всё это делает Policy Gradient алгоритмы куда более эффективными в средах с сильно отложенным сигналом.

Наконец, ещё одно небольшое, но важное преимущество Policy Gradient — обучение стохастичной политики. Хотя мы знаем, что оптимальная политика детерминирована, обучение стохастичной политики позволяет использовать её для сбора данных, и стохастичность — ненулевая вероятность засэмплировать любое действие — частично решает проблему исследования. Да, это, конечно, далёкое от идеала решение, но намного лучшее, чем, например, ϵ -жадная стратегия: можно рассчитывать на то, что если градиент в какой-то области пространства состояний постоянно указывает на то, что одни действия хорошие, а другие плохие, актёр выучит с вероятностью, близкой к единице, выбирать хорошие действия; если же о действиях в каких-то состояниях приходит противоречивая информация, или же такие состояния являются для агента новыми, то можно надеяться, что стратегия будет близка к равномерной, и агент будет пробовать все действия.

Все эти преимущества неразрывно связаны с on-policy режимом. За счёт «свежести» данных, можно делать, так сказать, наилучшие возможные шаги обучения стратегии, практически идти по градиенту оптимизируемого функционала. Но из него проистекает и главный недостаток подхода: неэффективность по количеству затрачиваемых сэмплов. Нам всё время нужны роллауты, сгенерированные при помощи текущей политики с параметрами θ , чтобы посчитать оценку градиента в точке θ и сделать шаг оптимизации. После этого будут нужны уже сэмплы из новой стратегии, поэтому единственное, что мы можем сделать с уже собранными данными — почистить оперативную память.

Понятно, что это полное безобразие: допустим, агент долго вёл себя псевдослучайно и наконец наткнулся на какой-то хороший исход с существенным откликом среды. Схема сделает по ценному роллауту всего один градиентный шаг, что скорее всего не поможет модели выучить удачное поведение или значение отклика. После этого ценная информация никак не может быть использована и придётся дожидаться следующей удачи, что может случиться нескоро. Это очень sample inefficient и основная причина, почему от любого on-policy обучения в чистом виде нужно пытаться отойти.

Пример 85: Допустим, вы играете в видео-игру, и в начале обучения мало что умеете, всё время действуя примерно случайно и падая в первую же яму. Среда выдаёт вам всё время ноль, и вы продолжаете вести себя случайно. Вдруг в силу стохастичности стратегии вы перепрыгиваете первую яму и получаете монетку +1. В DQN этот ценнейший опыт будет сохранён в буфере, и постепенно критик выучит, какие действия привели к награде. В A2C же агент делает один малоносительный шаг изменения весов моделей и тут же выкинет все собранные данные в мусорку, потому что на следующих итерациях он никак не может переиспользовать их. Агенту придётся ждать ещё много-много сессий в самой игре, пока он не перепрыгнет яму снова, чтобы сделать следующий шаг обучения перепрыгиванию ям.

Частично с этой проблемой нам удастся побороться в следующей главе, что позволит построить алгоритмы лучше A2C. Но важно, что полностью решить эту проблему, полностью перейти в off-policy режим, нам не удастся точно также, как в off-policy мы не смогли адекватно разрешать bias-variance trade-off (и там максимум наших возможностей была Retrace-оценка). Поэтому построить алгоритм, берущий «лучшее от двух миров», нельзя, и поэтому среди model-free алгоритмов выделяют два класса алгоритмов: off-policy, работающие с реплей буфером и потому потенциально более sample efficient, и on-policy алгоритмы, где есть ряд вышеуказанных преимуществ.

§5.3. Продвинутые Policy Gradient

5.3.1. Суррогатная функция

Как можно хотя бы частично побороться с ключевой проблемой Policy Gradient подхода — с on-policy режимом? Есть два соображения, как это можно попробовать сделать.

Во-первых, можно попробовать вместо градиентного подъёма использовать какой-то более тяжеловесный метод оптимизации, например, методы оптимизации второго порядка. Такие методы обычно требуют меньше обращений к оракулу, делая меньше итераций, за счёт более высокой вычислительной сложности каждого шага. В нашем случае это может быть ровно то, что нам нужно — вызов оракула для нас это сбор данных, и какие-то дополнительные вычисления «на компьютере» можно считать дешёвой процедурой.

Вторая идея — можно как-то попробовать посчитать оценку градиента в точке текущих параметров, при этом используя сэмплы, полученные при помощи другой стратегии. Полностью перейти в off-policy режим с сохранением всех преимуществ on-policy у нас не получится, поэтому придётся наложить ограничение на эту другую стратегию сбора данных: она должна быть очень похожей версией оцениваемой стратегии, то есть, можно считать, недавней версии актуальной стратегии. То, что стратегия сбора данных «похожа» на текущую,

означает, что её траектории тоже в некотором смысле «похожи»: понятно, что это важно, ведь много информации о градиенте политики по около-оптимальным параметрам с траекторий случайного агента собрать не получится, просто потому что такие траектории у около-оптимальной стратегии почти не встречаются.

На самом деле эти две идеи примерно об одном и том же, что мы и увидим далее.

Итак, допустим мы хотим оптимизировать стратегию π_θ по параметрам θ , используя только сэмплы из другой стратегии π^{old} (в итоговой схеме это будет недавняя версия стратегии). Что нам тогда мешает оценить значение (5.16)?

$$\nabla_\theta J(\pi) = \frac{1}{1-\gamma} \mathbb{E}_{d_\pi(s)} \mathbb{E}_{\pi(a|s)} \nabla_\theta \log \pi_\theta(a | s) A^\pi(s, a)$$

Во-первых, у нас нет сэмплов из $d_\pi(s)$, ведь в данных есть только³ состояния из $d_{\pi^{\text{old}}}(s)$. Мы уже обсуждали, что поскольку формула говорит проводить policy improvement, мы можем подменить это распределение на любое другое, и схема останется рабочей. Однако ключевой является вторая проблема. У нас нет оценки $A^\pi(s, a)$. Если мы попробуем оценить Advantage для одной стратегии по данным из другой, то нам придётся вводить все те коррекции, которые мы обсуждали в разделе 3.5.7 про Retrace оценку, которая скорее всего склоняется в смешённую. Очень хотелось бы всё-таки оставить «чистую» GAE-оценку, то есть как-то использовать «несвежего» критика $A^{\pi^{\text{old}}}(s, a)$ в формуле градиента.

Как сделать так, чтобы у нас где-то в формулах образовался «несвежий» критик? На помощь приходит RPI, формула (3.20): мы можем сменить функцию награды на Advantage функцию любой другой стратегии! Давайте запишем это в следующем виде:

Утверждение 62:

$$J(\pi_\theta) = J(\pi^{\text{old}}) + \frac{1}{1-\gamma} \mathbb{E}_{s \sim d_{\pi_\theta}(s)} \mathbb{E}_{a \sim \pi_\theta(a|s)} A^{\pi^{\text{old}}}(s_t, a_t) \quad (5.23)$$

Доказательство. Из RPI (3.20) следует, что для любых двух стратегий верно

$$J(\pi_\theta) = J(\pi^{\text{old}}) + \mathbb{E}_{\pi \sim \pi_\theta} \sum_{t \geq 0} \gamma^t A^{\pi^{\text{old}}}(s_t, a_t)$$

Для получения формулы достаточно переписать матожидание по траекториям из π_θ через state visitation distribution, подставив в теореме 57 в качестве $f(s, a) = A^{\pi^{\text{old}}}(s, a)$. ■

Тогда градиент исходного функционала $\nabla_\theta J(\pi_\theta)$ есть градиент правой части, и в ней уже как-то «замешана» оценка Advantage для стратегии π^{old} , которое мы сможем посчитать при помощи GAE-оценки:

$$\nabla_\theta J(\pi_\theta) = \nabla_\theta \frac{1}{1-\gamma} \mathbb{E}_{s \sim d_{\pi_\theta}(s)} \mathbb{E}_{a \sim \pi_\theta(a|s)} A^{\pi^{\text{old}}}(s_t, a_t)$$

Мы также можем справиться с матожиданием $\mathbb{E}_{a \sim \pi_\theta(a|s)}$ при помощи importance sampling. Да, этот коэффициент может быть ужасен (сильно большим единицы или близким к нулю), но это коррекция всего лишь за один шаг, и такая дробь будет терпимой.

$$\nabla_\theta J(\pi_\theta) = \nabla_\theta \frac{1}{1-\gamma} \mathbb{E}_{s \sim d_{\pi_\theta}(s)} \mathbb{E}_{a \sim \pi^{\text{old}}(a|s)} \frac{\pi_\theta(a | s)}{\pi^{\text{old}}(a | s)} A^{\pi^{\text{old}}}(s_t, a_t) \quad (5.24)$$

Осталась последняя проблема: $d_{\pi_\theta}(s)$. В роллаутах, сгенерированных при помощи π^{old} , состояния всё-таки будут приходить из $d_{\pi^{\text{old}}}(s)$, и тут мы importance sampling не сделаем даже при большом желании, так как просто не можем внятно оценить эти величины: из частот посещения состояний мы можем только сэмплировать.

Рассмотрим аппроксимацию: что, если мы в формуле (5.24) заменим $d_{\pi_\theta}(s)$ на $d_{\pi^{\text{old}}}(s)$? Это, вообще говоря, будет какая-то другая функция.

Определение 77:

Введём суррогатную функцию:

$$L_{\pi^{\text{old}}}(\theta) := \frac{1}{1-\gamma} \mathbb{E}_{s \sim d_{\pi^{\text{old}}}(s)} \mathbb{E}_{a \sim \pi^{\text{old}}(a|s)} \frac{\pi_\theta(a | s)}{\pi^{\text{old}}(a | s)} A^{\pi^{\text{old}}}(s, a) \quad (5.25)$$

Утверждение 63:

Сэмплы из матожиданий в суррогатной функции — это сэмплы из роллаутов, сгенериро-

³с учётом забивания на γ^t и наших прочих оговорок; но для корректности выкладок будем в этой главе писать везде дисконтированные частоты посещения состояний d

ванных при помощи π^{old} :

$$L_{\pi^{\text{old}}}(\theta) = \mathbb{E}_{\tau \sim \pi^{\text{old}}} \sum_{t \geq 0} \gamma^t \frac{\pi_\theta(a | s)}{\pi^{\text{old}}(a | s)} A^{\pi^{\text{old}}}(s, a)$$

Пояснение. Применить теорему 57 в обратную сторону для $f(s, a) = \frac{\pi_\theta(a | s)}{\pi^{\text{old}}(a | s)} A^{\pi^{\text{old}}}(s, a)$ ■

Какой у этой суррогатной функции $L_{\pi^{\text{old}}}(\theta)$ физический смысл? Мы сказали, что наши настраиваемые параметры θ не влияют на частоты посещения состояний, то есть выбор действий не влияет на то, какие состояния мы будем посещать в будущем. Это очень сильное допущение, поэтому аппроксимация не самая удачная. Давайте поймём, что будет происходить, если мы будем оптимизировать вместо честного, исходного функционала, такую суррогатную функцию при фиксированной π^{old} :

$$L_{\pi^{\text{old}}}(\theta) \rightarrow \max_{\theta} \quad (5.26)$$

Утверждение 64: Решением оптимационной задачи (5.26) является жадная стратегия по отношению к $Q^{\pi^{\text{old}}}(s, a)$ (или, что тоже самое, к Advantage-функции стратегии π^{old}):

$$\pi_\theta(s) = \operatorname{argmax}_a A^{\pi^{\text{old}}}(s, a)$$

Доказательство. Эта оптимационная задача распадается на оптимационную задачу для каждого s , а решением задачи

$$\mathbb{E}_{a \sim \pi^{\text{old}}(a | s)} \frac{\pi_\theta(a | s)}{\pi^{\text{old}}(a | s)} A^{\pi^{\text{old}}}(s, a) = \mathbb{E}_{a \sim \pi_\theta(a | s)} A^{\pi^{\text{old}}}(s, a) \rightarrow \max_{\theta}$$

является «жадная» стратегия. ■

Другими словами, такая суррогатная функция просто говорит проводить policy improvement стратегии π^{old} , но в состояниях, приходящих из частот посещения состояний $d_{\pi^{\text{old}}}(s)$. Такая очередная форма нам сейчас будет удобна, поскольку такая суррогатная функция является локальной аппроксимацией нашего оптимизируемого функционала, причём эта аппроксимация — не просто, скажем, линейное приближение оптимизируемой функции, а какое-то более умное, учитывающее особенности задачи.

5.3.2. Нижняя оценка

Итак, у нас есть аппроксимация нашего оптимизируемого функционала через суррогатную функцию, с которой мы можем работать:

$$J(\pi) \approx J(\pi^{\text{old}}) + L_{\pi^{\text{old}}}(\theta) = J(\pi^{\text{old}}) + \frac{1}{1 - \gamma} \mathbb{E}_{s \sim d_{\pi^{\text{old}}}(s)} \mathbb{E}_{a \sim \pi^{\text{old}}(a | s)} \frac{\pi_\theta(a | s)}{\pi^{\text{old}}(a | s)} A^{\pi^{\text{old}}}(s, a)$$

Насколько эта аппроксимация хороша? Наша интуиция была в том, что если π «похожа» на π^{old} , то частоты посещения состояний у них тоже наверняка будут похожи. Формализовать «похожесть» стратегий можно, например, так:

Определение 78: Введём расстояние между стратегиями $\pi^{\text{old}}, \pi_\theta$ как среднюю KL-дивергенцию между ними по состояниям из частот посещения первой стратегии:

$$\text{KL}(\pi^{\text{old}} \| \pi_\theta) := \mathbb{E}_{s \sim d_{\pi^{\text{old}}}(s)} \text{KL}(\pi^{\text{old}}(a | s) \| \pi_\theta(a | s))$$

Теорема 59:

$$|J(\pi_\theta) - J(\pi^{\text{old}}) - L_{\pi^{\text{old}}}(\theta)| \leq C \sqrt{\text{KL}(\pi^{\text{old}} \| \pi_\theta)}$$

где C — константа, равная $C = \frac{\sqrt{2}\gamma}{(1-\gamma)^2} \max_{s,a} |A^{\pi^{\text{old}}}(s, a)|$

Без доказательства; интересующиеся могут обратиться к статье Constrained Policy Optimization. ■

Конечно, это очень грубая оценка, хотя бы потому, что она верна для произвольных MDP и произвольных двух стратегий. Но ключевой момент в том, что мы теперь формально можем вывести **нижнюю оценку** (lower

bound) на оптимизируемый функционал⁴:

Теорема 60 — Performance Lower Bound:

$$J(\pi_\theta) - J(\pi^{\text{old}}) \geq L_{\pi^{\text{old}}}(\theta) - C \sqrt{\text{KL}(\pi^{\text{old}} \parallel \pi_\theta)} \quad (5.27)$$

Возникает любопытнейшая идея: возможно, мы можем работать не с исходным функционалом, а с нижней оценкой.

Теорема 61: Процедура оптимизации

$$\theta_{k+1} := \underset{\theta}{\operatorname{argmax}} \left[L_{\pi_{\theta_k}}(\theta) - C \sqrt{\text{KL}(\pi_{\theta_k} \parallel \pi_\theta)} \right] \quad (5.28)$$

гарантирует монотонное неубывание функционала: $J(\pi_{\theta_{k+1}}) \geq J(\pi_{\theta_k})$

Доказательство. В точке $\theta = \theta_k$ суррогатная функция $L_{\pi_{\theta_k}}(\theta)$ равна нулю, поскольку

$$\begin{aligned} L_{\pi_{\theta_k}}(\theta_k) &= \\ &= \{\text{по определению (5.25)}\} = \frac{1}{1-\gamma} \mathbb{E}_{s \sim d_{\pi_{\theta_k}}(s)} \mathbb{E}_{a \sim \pi_{\theta_k}(a|s)} \frac{\pi_{\theta_k}(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a) = \\ &= \frac{1}{1-\gamma} \mathbb{E}_{s \sim d_{\pi_{\theta_k}}(s)} \mathbb{E}_{a \sim \pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a) = \\ &\{\text{следствие RPI (5.23)}\} = J(\pi_{\theta_k}) - J(\pi_{\theta_k}) = 0 \end{aligned}$$

Понятно, что $\text{KL}(\pi_{\theta_k} \parallel \pi_\theta)$ в точке $\theta = \theta_k$ тоже равна 0, так как во всех состояниях KL между одинаковыми стратегиями равна нулю. Значит, максимум нижней оценки не меньше нуля, а она есть нижняя оценка на $J(\pi_{\theta_{k+1}}) - J(\pi_{\theta_k})$. ■

Итак, в воздухе витает идея заняться типичным *minorization-maximization* алгоритмом. Сначала мы подтягиваем нашу нижнюю оценку так, чтобы в текущей точке $\theta = \theta^{\text{old}}$ она в точности совпадала с оптимизируемой функцией (minorization). Затем мы начинаем при фиксированном θ^{old} оптимизировать по θ не наш функционал (который мы не сможем оптимизировать без новых сэмплов из новой стратегии π_θ с текущими значениями параметров), а нижнюю оценку, с которой умеем работать (maximization).

То есть, что мы получили: мы «можем» оптимизировать не $J(\pi_\theta) - J(\pi^{\text{old}})$ по формуле (5.24), а нашу суррогатную функцию $L_{\pi^{\text{old}}}(\theta)$, то есть проводить policy improvement стратегии π^{old} , но добавив штраф — регуляризатор — за различие между π_θ и стратегией, из которой приходят данные π^{old} :

$$L_{\pi^{\text{old}}}(\theta) - C \sqrt{\text{KL}(\pi^{\text{old}} \parallel \pi_\theta)} \rightarrow \max_{\theta} \quad (5.29)$$

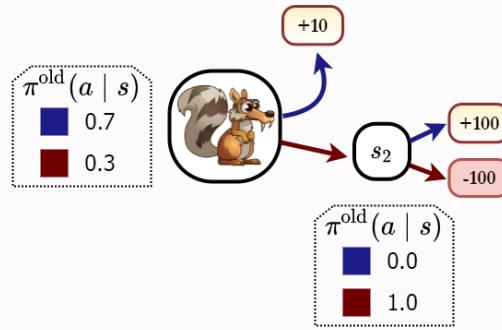
Мы получили процедуру, гарантирующую улучшение стратегии, что звучит подозрительно хорошо. Очень похожие гарантии улучшения стратегии у нас были в policy improvement. Интересно порассуждать, в чём отличие. Мы уже увидели в утверждении 64, что оптимизация суррогатной функции без регуляризатора соответствует policy improvement-у стратегии π^{old} . Однако как мы помним, жадный policy improvement вовсе не является наилучшим: если мы в некотором состоянии s перекладываем вероятностную массу в какое-то действие, чтобы увеличить среднее значение $A^{\pi^{\text{old}}}(s, a)$, мы попадаем чаще в те состояния, где стратегия π^{old} набирает больше, да, но на самом деле мы таким изменением стратегии можем помешать себе добираться чаще до тех состояний, где $A^{\pi^{\text{old}}}(s, a)$ принимает большие положительные значения!

⁴исторически в статьях по TRPO и PPO использовалась чуть более грубая нижняя оценка, в которой ошибка между суррогатной функцией и честным функционалом оценивалась сверху при помощи KL-дивергенции в максимальной форме:

$$\text{KL}^{\max}(\pi^{\text{old}} \parallel \pi) := \max_s \text{KL}(\pi^{\text{old}}(a|s) \parallel \pi(a|s))$$

Однако такое выражение посчитать в практических алгоритмах нельзя, поэтому далее её приходилось эвристически заменять на среднее по $s \sim d_{\pi^{\text{old}}}(s)$. Уточнённая нижняя оценка обосновывает этот переход и указывает, что из KL-дивергенции также нужно взять корень; в остальном на ход дальнейших рассуждений это не влияет.

Пример 86: Что говорит policy improvement для MDP с картинки и заданной стратегией π^{old} ? В начальном состоянии $Q^{\pi^{\text{old}}}(s, \square) = +10$, $Q^{\pi^{\text{old}}}(s, \blacksquare) = -100$, и поэтому улучшение будет заключаться в том, что новая стратегия должна чаще выбирать именно действие \square , выбирать $+10$. Это связано с тем, что policy improvement игнорирует влияние действий на частоты посещения состояний, и оптимизирует «новую награду» $A^{\pi^{\text{old}}}(s, a)$ независимо в каждом состоянии жадным образом. Он не видит, что выбор в начальном состоянии действия \blacksquare позволит ему попасть в такое состояние, где для одного из действий $A^{\pi^{\text{old}}}(s_2, \square) = +200$!



Поэтому формула policy gradient — «наилучшего policy improvement-а» — говорит, что как только параметры θ хоть чуть-чуть изменяются, сразу же стоит улучшать новую стратегию, то есть использовать свежего критика. Мы же свежего критика получить не можем, хотим пользоваться лишь «несвежим» $A^{\pi^{\text{old}}}(s, a)$, и нижняя оценка (5.29) даёт промежуточную альтернативу, что тогда можно делать: добавить регуляризатор, запрещающий сильное изменение исходной стратегии π^{old} .

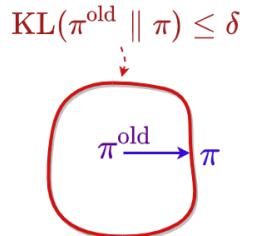
Однако чтобы оптимизировать (5.29), нужно как-то определить значение константы C : мы не умеем считать выражение для неё из теоремы 59, поскольку там присутствует максимум Advantage-функции по всем парам состояние-действие⁵. Мы можем заменить его на гиперпараметр, но, чтобы не потерять теоретические гарантии, он должен быть достаточно большим, чтобы превосходить значение из теоремы. Вообще, скорее всего, даже если бы мы знали эту константу, она была бы колоссальной: чего стоит только $(1 - \gamma)^2$ в знаменателе формулы из теоремы 59. Поэтому практической пользы от такой низкой оценки всё равно много не было: её оптимизация делала бы слишком консервативные шаги обновления политики.

5.3.3. Trust Region Policy Optimization (TRPO)

В TRPO предлагается воспользоваться полученной теорией, чтобы построить на основе идеи суррогатной функции более «мощный» метод оптимизации, чем обычный градиентный спуск. Как обычно устроены методы оптимизации? В текущей точке для рассматриваемой функции строится какая-то модель, какая-то локально верная аппроксимация (например, линейная или квадратичная). Далее эта модель либо оптимизируется, и алгоритм сдвигает текущее решение в сторону оптимума модели (такие методы относят к *line search* подходу), либо модель оптимизируется в некотором «*регионе доверия*» (trust region), где эта модель, считается, более-менее похожа на исходный функционал. Подход на основе регионов доверия считается более тяжеловесным, поскольку требует решать на каждом шаге работы алгоритма условную задачу оптимизации, зато более эффективным по числу итераций.

Основная идея TRPO заключается в переходе от оптимизации (5.29) без ограничений к задаче оптимизации с ограничением в trust region форме:

$$\begin{cases} L_{\pi^{\text{old}}}(\theta) \rightarrow \max_{\theta} \\ \text{KL}(\pi^{\text{old}} \parallel \pi_{\theta}) \leq \delta \end{cases} \quad (5.30)$$



То есть, суррогатная функция является локальной аппроксимацией нашего функционала, поэтому на каждом шаге работы алгоритма мы будем работать с ней. При этом второе слагаемое оптимизируемой нижней оценки (5.29) подсказывает, что с ростом KL-дивергенции «нечестный» функционал всё меньше похож на настоящий, и к нему «всё меньше доверия». Условная задача оптимизации говорит, что оптимизировать суррогатную функцию вместо настоящего функционала можно, но с жёстким ограничением на длину шага.

Вообще говоря, мы получили задачу для оптимизации $L_{\pi^{\text{old}}}(\theta)$ методом *натуральных градиентов* (natural gradient): мы оптимизируем функцию, разрешая не шаг некоторой длины по градиенту в пространстве параметров, а шаг некоторой длины в пространстве параметрически заданных распределений. Подробнее о натуральном градиенте можно прочитать в приложении A.1. Если раньше градиентный шаг мог сильно поменять стратегию (как распределение в пространстве действий), при том что для других небольших изменений распределения было бы необходимо сильно менять параметры θ , то здесь δ ограничивает изменение самой стратегии в терминах KL-дивергенции. Ограничение δ нам при этом всё равно нужно будет выбирать, это аналог learning rate в «trust region формах» методов оптимизации.

Как будем задачу (5.30) решать? В контексте нашей задачи мы собрали при помощи текущей стратегии некоторое количество данных для Монте-Карло оценок всех мат.ожиданий (в этот момент $\pi^{\text{old}} = \pi_{\theta}$) и хотим решить задачу (5.30), зафиксировав π^{old} и оптимизируя параметры θ . Обозначим параметры π^{old} как θ^{old} . Аппроксимируем оптимизируемый функционал $L_{\pi^{\text{old}}}(\theta)$ разложением Тейлора до первого порядка с центром в точке $\theta = \theta^{\text{old}}$, а ограничение — до второго. До второго — потому что слагаемое первого порядка ноль.

⁵мы могли бы оценить его сверху как $2R^{\text{max}}$, где R^{max} — максимальная награда, но это было бы непрактично грубою оценкой.

Утверждение 65: В точке $\theta = \theta^{\text{old}}$ первый член разложения ограничения в ряд Тейлора равен нулю:

$$\forall s: \nabla_{\theta} \text{KL}(\pi^{\text{old}} \| \pi_{\theta})|_{\theta=\theta^{\text{old}}} = 0$$

Доказательство. **KL**-дивергенция в этой точке равна 0 как среднее по состояниям дивергенций между одинаковыми распределениями, следовательно как функция от θ она достигает в этой точке глобального минимума \Rightarrow градиент равен нулю. ■

Итак, введём обозначения для предложенного разложения. Пусть \mathbf{g} — градиент $L_{\pi^{\text{old}}}(\theta)$ по параметрам θ в точке $\theta = \theta^{\text{old}}$, а \mathbf{F} — гессиан ограничения в точке $\theta = \theta^{\text{old}}$:

$$\mathbf{g} := \nabla_{\theta} L_{\pi^{\text{old}}}(\theta)|_{\theta=\theta^{\text{old}}} \quad (5.31)$$

$$\mathbf{F} := \nabla_{\theta}^2 \text{KL}(\pi^{\text{old}} \| \pi_{\theta})|_{\theta=\theta^{\text{old}}} \quad (5.32)$$

В этих обозначениях аппроксимация задачи получается следующая:

$$\begin{cases} \langle \mathbf{g}, \theta - \theta^{\text{old}} \rangle \rightarrow \max_{\theta} \\ \frac{1}{2} (\theta - \theta^{\text{old}})^T \mathbf{F} (\theta - \theta^{\text{old}}) \leq \delta \end{cases} \quad (5.33)$$

Градиент \mathbf{g} , на самом деле, весьма любопытен:

Утверждение 66: Градиент \mathbf{g} совпадает с градиентом из алгоритма Advantage Actor Critic (5.16).

Доказательство.

$$\begin{aligned} \mathbf{g} &= \nabla_{\theta} L_{\pi^{\text{old}}}(\theta)|_{\theta=\theta^{\text{old}}} = \frac{1}{1-\gamma} \mathbb{E}_{s \sim d_{\pi^{\text{old}}}(s)} \mathbb{E}_{a \sim \pi^{\text{old}}(a|s)} \frac{\nabla_{\theta} \pi_{\theta}(a | s)|_{\theta=\theta^{\text{old}}}}{\pi^{\text{old}}(a | s)} \mathbf{A}^{\pi^{\text{old}}}(s, a) = \\ &= \left\{ \begin{array}{l} \text{замечаем определение} \\ \text{производной логарифма} \end{array} \right\} = \frac{1}{1-\gamma} \mathbb{E}_{s \sim d_{\pi^{\text{old}}}(s)} \mathbb{E}_{a \sim \pi^{\text{old}}(a|s)} \nabla_{\theta} \log \pi_{\theta}(a | s)|_{\theta=\theta^{\text{old}}} \mathbf{A}^{\pi^{\text{old}}}(s, a) \end{aligned}$$

что в точности есть градиент обычного ActorCritic с бэйзлайном в точке $\theta = \theta^{\text{old}}$. ■

Теорема 62: Решение аппроксимированной задачи (5.33) есть

$$\theta - \theta^{\text{old}} = \mathbf{k} \mathbf{F}^{-1} \mathbf{g}$$

где скалярный коэффициент пропорциональности \mathbf{k} можно посчитать по формуле $\mathbf{k} = \sqrt{\frac{2\delta}{\mathbf{g}^T \mathbf{F}^{-1} \mathbf{g}}}$

Доказательство. Составляем лагранжиан (оптимизируемый функционал входит с минусом, т.к. максимум поменяем на минимум):

$$\mathcal{L}(\lambda, \theta) = -\mathbf{g}^T (\theta - \theta^{\text{old}}) + \lambda \left(\frac{1}{2} (\theta - \theta^{\text{old}})^T \mathbf{F} (\theta - \theta^{\text{old}}) - \delta \right)$$

Дифференцируем лагранжиан и приравниваем к нулю:

$$\nabla_{\theta} \mathcal{L}(\lambda, \theta) = -\mathbf{g} + \lambda \mathbf{F} (\theta - \theta^{\text{old}}) = 0$$

Отсюда получаем:

$$\theta - \theta^{\text{old}} = \frac{1}{\lambda} \mathbf{F}^{-1} \mathbf{g} \quad (5.34)$$

Осталось найти значение λ . Поскольку решение должно быть допустимой точкой, а оптимизируемый функционал линеен, понятно, что ограничение из задачи превратится в равенство и будет достигнуто на границе. То есть:

$$\frac{1}{2} (\theta - \theta^{\text{old}})^T \mathbf{F} (\theta - \theta^{\text{old}}) = \delta$$

Подставляем найденное решение (5.34):

$$\frac{1}{2} \left(\frac{1}{\lambda} \mathbf{F}^{-1} \mathbf{g} \right)^T \mathbf{F} \left(\frac{1}{\lambda} \mathbf{F}^{-1} \mathbf{g} \right) = \frac{1}{2\lambda^2} \mathbf{g}^T \mathbf{F}^{-1} \mathbf{g} = \delta$$

Отсюда находим коэффициент^{*}:

$$\lambda = \sqrt{\frac{\mathbf{g}^T \mathbf{F}^{-1} \mathbf{g}}{2\delta}}$$

Обратная дробь $\frac{1}{\lambda}$, соответственно, является коэффициентом пропорциональности. ■

* однозначно в силу положительности коэф. Лагранжа; число $\mathbf{g}^T \mathbf{F}^{-1} \mathbf{g}$ положительно в силу того, что гессиан KL-дивергенции является матрицей Фишера и поэтому является положительно определённой матрицей (см. приложение A.1.2). Естественно, усреднение по состояниям не нарушает этого факта.

Итак, что мы делаем на практике. Во-первых, собираем большой-большой роллаут (порядка 1024 переходов суммарно по средам), чтобы возможно было хоть сколько-то адекватно оценивать гессиан \mathbf{F} (5.32). Оцениваем Advantage собранных пар s, a при помощи GAE (5.21). Затем считаем градиент \mathbf{g} (5.31) аналогично Actor-Critic методу. Далее мы хотим решить систему линейных уравнений:

$$\mathbf{F}(\boldsymbol{\theta} - \boldsymbol{\theta}^{\text{old}}) = \mathbf{g}$$

Хранить в памяти гессиан и тем более обращать для нейросеток мы, конечно, не будем и воспользуемся **методом сопряжённых градиентов** (conjugate gradient method), который позволяет решать систему линейных уравнений итеративно, на каждой итерации требуя лишь вычислять $\mathbf{F}\mathbf{h}$ для некоторых векторов \mathbf{h} (прочитать про него можно, например, в [википедии](#)). Нам придётся для каждой итерации метода сопряжённых градиентов сделать два обратных прохода по вычислительному графу (дважды вычислять все производные: сначала по функции, для которой мы хотим посчитать гессиан, чтобы получить градиент $f := \nabla_{\boldsymbol{\theta}} \text{KL}(\boldsymbol{\pi}^{\text{old}} \parallel \boldsymbol{\pi}_{\boldsymbol{\theta}})|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{\text{old}}}$, затем для градиента функции $\langle f, h \rangle$), но до сходимости сводить алгоритм не будем и остановим после порядка 10 итераций. Процедура получается дороговатой всё равно, но поскольку в деле замешан гессиан, то что поделать — мы практически выходим в методы оптимизации второго порядка.

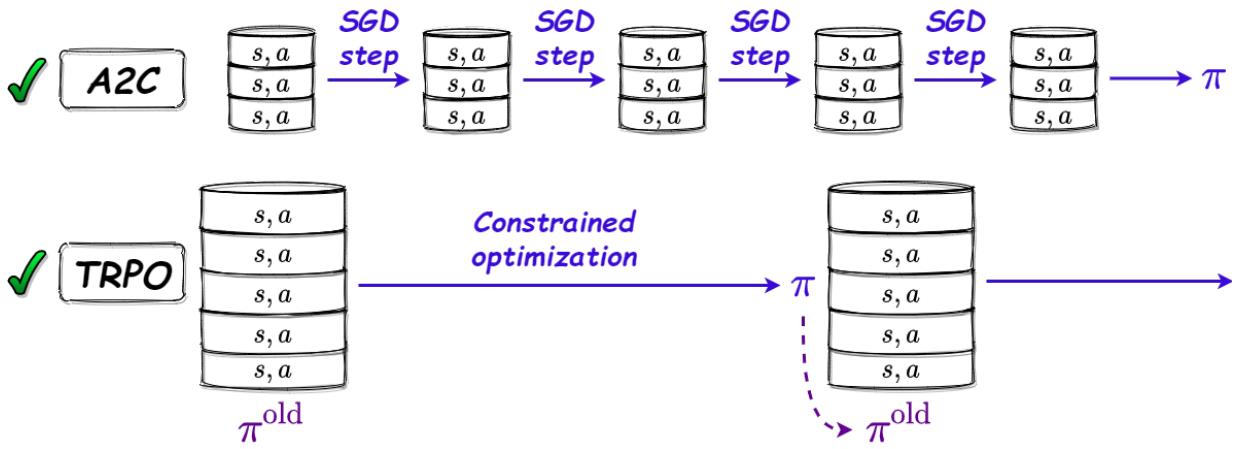
Посчитали приближение $\mathbf{F}^{-1}\mathbf{g}$, дальше возникает проблема: нам нужно домножить вектор на коэффициент k , который напрямую зависит от δ . Мы могли бы поставить δ гиперпараметром как learning rate в обычном градиентном спуске, но всё-таки мы боремся за то, чтобы делать шаги «правильного» размера. Мы уже знаем направление $\mathbf{F}^{-1}\mathbf{g}$, в котором будем менять параметры политики, но не знаем, насколько. И вот тут становится важно, что когда мы приблизили локально $J(\boldsymbol{\theta})$ суррогатной функцией, мы для суррогатной функции можем посчитать значение в любой точке.

Когда мы делаем шаг градиентного спуска, мы рискуем сделать слишком длинный шаг, даже если он делается в верном направлении. В Policy Gradient методах это особенно критично: если сделать неудачное обновление политики, и $\boldsymbol{\pi}$ сломается, то на следующем шаге собранные данные будут ужасными, поскольку они собираются on-policy, при помощи стратегии $\boldsymbol{\pi}$! Поэтому важно не сломать стратегию ни на одном шаге, и для этого learning rate приходится выбирать маленьким. Естественно, это приводит к тому, что модель будет обучаться очень долго (а у нас тут сэмплы на каждый шаг сжигаются!). При этом, как это обычно бывает при обучении нейронных сетей, запускать какую-нибудь процедуру автоматического подбора learning rate, дорогоевато — для этого нужно уметь вычислять значение оптимизируемой функции в разных точках. Тем более в случае с RL-ем, где, чтобы оценить $J(\boldsymbol{\theta})$ и проверить, не сломалось ли чего, нужно отправлять на каждом шаге несколько раз какие-то стратегии в среду и играть целые эпизоды, это совершенно не вариант.

Но здесь, в TRPO, мы применяли аппроксимацию дважды: сначала приблизили функционал на суррогатную функцию, а затем суррогатную функцию приблизили линейной моделью. Можно было бы сказать, что мы просто исходный функционал приблизили линейно (получился бы тот же самый градиент \mathbf{g} , как мы разобрали), но для промежуточной задачи (5.30), в которой мы работаем с суррогатной функцией, значение которой мы можем посчитать в любой точке, мы на самом деле можем проводить **бэктрэкинг** для адаптивного подсчёта δ — аналога learning rate в trust region подходе. Его можно проводить немного по-разному, рассмотрим конкретный вариант из стандартных реализаций TRPO.

Выставляем какое-нибудь большое значение δ (это начальное значение — гиперпараметр) и считаем коэффициент пропорциональности $k = \sqrt{\frac{2\delta}{\mathbf{g}^T \mathbf{F}^{-1} \mathbf{g}}}$ (заметим, что $\mathbf{F}^{-1}\mathbf{g}$ приближённо мы уже посчитали в методе сопряжённых градиентов). Дальше, во-первых, проверяется, а правда ли мы остались внутри trust region-а, то есть соблюли ли условие $\text{KL}(\boldsymbol{\pi}^{\text{old}} \parallel \boldsymbol{\pi}_{\boldsymbol{\theta}}) \leq \delta$. Мы могли нарушить это условие как из-за перехода к приближённой задаче (5.33), так и из-за приближённого её решения через метод сопряжённых градиентов; если таки нарушили, то уменьшаем δ в, допустим, два раза и перепроверяем. Если же условие соблюдено, то проверяем, что значение $L_{\boldsymbol{\pi}^{\text{old}}}(\boldsymbol{\theta})$, оценённое по имеющимся данным, больше нуля (что хотя бы для эмпирических данных удалось увеличить значение суррогатной функции): если нет, то продолжаем уменьшать регион доверия, а если да, то мы можем заканчивать процедуру бэктрэкинга и делать таким образом «максимально длинный» шаг.

После такой процедуры мы уже шагнули на границу нашего региона доверия, максимально отступив от $\boldsymbol{\pi}^{\text{old}}$, использовавшихся для сбора данных. Соответственно, смысла «продолжать» оптимизировать нижнюю оценку дальше, разложив функционал в $\boldsymbol{\theta} \neq \boldsymbol{\theta}^{\text{old}}$, нет; нужно перестраивать нижнюю оценку, то есть собирать новые данные при помощи обновлённой стратегии. Можно сказать, что переиспользовать данные мы не научились, но мы научились делать тяжёлые и дорогие, но хорошие шаги оптимизации.



В текущем виде в TRPO критика всё ещё нужно оптимизировать также, как в A2C, то есть собирая маленькие роллауты и делая небольшие шаги. Не очень удобно делать это параллельно со сбором больших роллаутов для актёра, да и оптимизировать актёра и критика в этой схеме, получается, придётся раздельно (иметь для них две раздельные сетки). Поскольку роллауты собирают большие, в имплементациях на критика иногда вообще забивают и используют Монте-Карло оценку как в REINFORCE, просто доигрывая игры до конца (если игры относительно короткие, по 20-100 шагов, то это может быть разумно: всё равно нам длинные роллауты нужны шагов по 100). Конечно, если всё-таки обучать критика, то можно существенно выиграть от использования GAE-оценок.

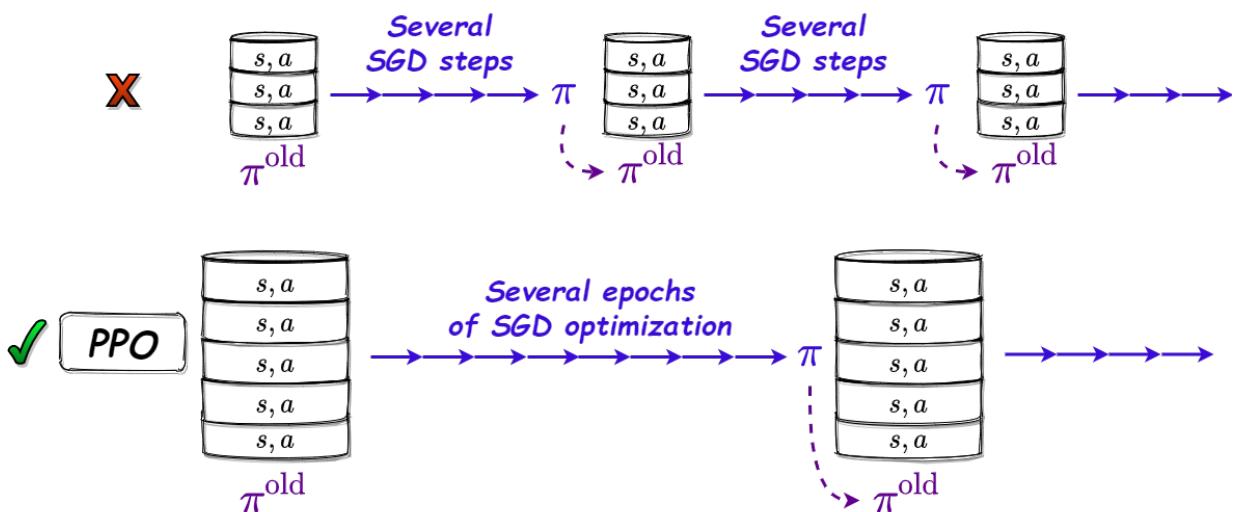
5.3.4. Proximal Policy Loss

Считается, что TRPO практически всегда работает лучше A2C, с единственным основным недостатком — сложностью самого алгоритма. Всё-таки в глубоком обучении привычнее работать с какими-то датасетами, из которых сэмплируются батчи, вычисляется какая-то функция потерь или оптимизируемый функционал, считается градиент и отправляется в условный Adam. Настраивать такой алгоритм тяжело и неприятно.

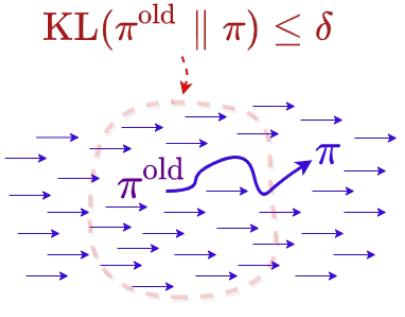
Proximal Policy Optimization (PPO) — альтернативный способ рассуждения после вывода нижней оценки (5.29). Давайте не будем прибегать к методам оптимизации, требующим какие-либо гессианы, и будем оптимизировать нижнюю оценку напрямую обычным градиентным спуском, где \mathbf{C} — гиперпараметр:

$$\mathbb{E}_{s \sim d_{\pi^{\text{old}}}(s)} \mathbb{E}_{a \sim \pi^{\text{old}}(a|s)} \left[\frac{\pi_{\theta}(a | s)}{\pi^{\text{old}}(a | s)} A^{\pi^{\text{old}}}(s, a) - C \sqrt{\text{KL}(\pi^{\text{old}} \| \pi_{\theta})} \right] \rightarrow \max_{\theta} \quad (5.35)$$

Сделать так напрямую в лоб не получится; давайте поймём, почему. Какого размера датасет нам понадобится для такого пайплайна? Собрать данных на мини-батч или несколько и «воспользоваться ими несколько раз» не получится: Монте-Карло оценки мат.ожиданий в наших функциях потерь просто будут скоррелированы. Чтобы по данным из π^{old} прооптимизировать θ несколькими шагами градиентного спуска, нужно, чтобы мини-батчи были достаточно разными. Это значит, что при помощи π^{old} придётся собрать датасет достаточно большого размера.



Чтобы что-то выиграть от такого процесса, нужно пройтись по датасету несколькими эпохами (иначе тот же A2C был бы выгоднее за счёт свежести данных). Это значит, что шагов градиентной оптимизации понадобится сделать достаточно много. Следовательно, стратегия потенциально обновляется за время оптимизации на одном датасете относительно сильно: вместе с тем, что небольшое изменение в пространстве параметров может приводить к большому изменению стратегии π (в пространстве распределений) без введения жёсткого региона доверия стратегия может начать сколь угодно сильно подстраиваться под те оценки критика, которые мы выдадим параметрам в датасете.



Действительно, на практике наш критик $A^{\pi^{\text{old}}}(s, a)$ неидеален и его оценка $\Psi(s, a) \approx A^{\pi^{\text{old}}}(s, a)$ будет смешённой. Но важно, что в оценке Ψ будут заложены сэмплы из собранных при помощи π^{old} данных: награды и состояния за будущие шаги. Для каждой пары s, a мы увидим лишь по одному сэмплу будущего, и как бы мы оценку $\Psi(s, a)$ ни строили, эти сэмплы s', a', \dots у нас ровно в одном экземпляре. Мы не хотим, ест-но, несколько раз π^{old} в среду отправлять — это бессмысленно, в on-policy режиме всегда выгоднее если и отправлять в среду, то самую свежую стратегию, переходя таким образом к очередному шагу алгоритма. А значит, что при фиксированных данных при оптимизации суррогатной функции мы не жаждый policy improvement стратегии π^{old} проведём, а полную ерунду: будем искать $\underset{a}{\operatorname{argmax}} \Psi(s, a)$, то есть перекладывать всю вероятность в те действия, где оценка (!) Advantage случилась положительная, и убирать вероятность оттуда, где оценка Advantage случилась отрицательная. Этот эффект можно охарактеризовать как «переобучение под сэмплы».

Пример 87: Утрированный пример. Текущая стратегия сделала два шага в среде, собрала s_1, a_1, s_2, a_2 . Оценки Advantage получились следующими: $\Psi(s_1, a_1) = 1$, $\Psi(s_2, a_2) = -1$. Теперь если мы по сэмплам оценили суррогатную функцию (5.25) и начали её оптимизировать по параметрам стратегии, то получится:

$$\frac{\pi_\theta(a_1 | s_1)}{\pi^{\text{old}}(a_1 | s_1)}(+1) + \frac{\pi_\theta(a_2 | s_2)}{\pi^{\text{old}}(a_2 | s_2)}(-1) \rightarrow \max_{\theta}$$

Знаменатели дробей можно считать какими-то положительными числами, поэтому в итоге $\pi_\theta(a_2 | s_2)$ полетит в ноль, $\pi_\theta(a_1 | s_1)$ — или в единицу, если пространство действий дискретно, или вообще в бесконечность, если пространство действий непрерывно.

То есть на тех парах s, a , где оценка Advantage положительна, стратегия начнёт улетать к вырожденной, а вероятности на парах s, a с отрицательной оценкой Advantage начнут уплывать в ноль. В итоге начинают регулярно встречаться взрывающиеся или затухающие importance sampling коэффициенты $\frac{\pi_\theta(a|s)}{\pi^{\text{old}}(a|s)}$, мини-батч становится несбалансированным и в плане «весов» объектов. И регуляризатор в виде KL-дивергенции из нижней оценки, поставленный в виде жёсткого ограничения («trust region»-а) в задачу оптимизации (5.30), защищал нас от этого эффекта в TRPO.

Предлагается полечить костылём: обрезать. Обозначим importance sampling коэффициент как

$$\rho(\theta) := \frac{\pi_\theta(a | s)}{\pi^{\text{old}}(a | s)}$$

и обрежем его как

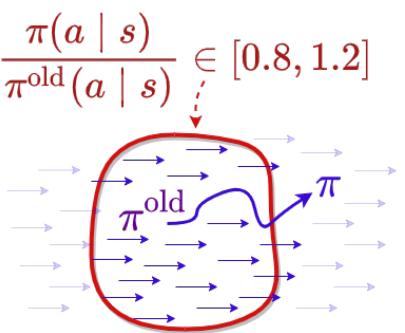
$$\rho^{\text{clip}}(\theta) := \text{clip}(\rho(\theta), 1 - \epsilon, 1 + \epsilon),$$

где $\epsilon \in (0, 1)$ — гиперпараметр (типичный выбор — 0.1 или 0.2). Рассмотрим альтернативную функцию потерь:

$$\mathbb{E}_{s \sim d_{\pi^{\text{old}}}(s)} \mathbb{E}_{a \sim \pi^{\text{old}}(a | s)} \left[\rho^{\text{clip}}(\theta) A^{\pi^{\text{old}}} (s, a) - C \sqrt{\text{KL}(\pi^{\text{old}} \| \pi_\theta)} \right] \rightarrow \max_{\theta}$$

Что случилось с градиентами при таком изменении? Если происходит обрезка, то есть если $\rho(\theta)$ не попадает в указанный диапазон $[1 - \epsilon, 1 + \epsilon]$, то градиент основного слагаемого, как легко видеть, зануляется. Иначе он остаётся без изменений:

$$\nabla_{\theta} \rho^{\text{clip}}(\theta) = \begin{cases} 0 & \rho(\theta) \notin [1 - \epsilon, 1 + \epsilon] \\ \nabla_{\theta} \rho(\theta) & \rho(\theta) \in [1 - \epsilon, 1 + \epsilon] \end{cases}$$



Таким образом, подобный клиппинг — «мягкий trust region»: как только стратегия слишком отдаляется от π^{old} на данной паре s, a , градиенты «пестрят» обновлять эту пару. Это не означает, что стратегия в этой паре не продолжит меняться: с ней потенциально в градиентном спуске может

происходить всё что угодно, она может продолжать изменяться за счёт «схожих» пар s, a , например, которые ещё остаются «внутри trust region-a» или, в конце концов, из-за моментаума в алгоритме стохастической оптимизации⁶.

Замена функции потерь лишила нас в очередной раз «гарантий нижней оценки»: хотелось бы, чтобы при достаточно большой константе C эти гарантии оставались. Поэтому мы не просто заменим функцию потерь на версию с обрезкой, а возьмём минимум между ними: так мы сохраним свойство нижней оценки и гарантируем, что функционал (5.35) не увеличился:

$$\mathbb{E}_{s \sim d_{\pi^{\text{old}}}(s)} \mathbb{E}_{a \sim \pi^{\text{old}}(a|s)} \left[\min \left(\rho(\theta) A^{\pi^{\text{old}}}(s, a), \rho^{\text{clip}}(\theta) A^{\pi^{\text{old}}}(s, a) \right) - C \sqrt{\text{KL}(\pi^{\text{old}} \parallel \pi_\theta)} \right] \rightarrow \max_\theta \quad (5.36)$$

Интуиция нижней оценки здесь, на самом деле, под очень большим вопросом. Авторы алгоритма на этом этапе в ablation studies внезапно обнаружили, что на слагаемое с KL-дивергенцией можно внезапно забить (выставить $C = 0$), и эмпирические результаты не изменятся. Обычно в имплементациях PPO KL-дивергенции в функционале по дефолту нет⁷, и итого оптимизируемый функционал выглядит просто вот так:

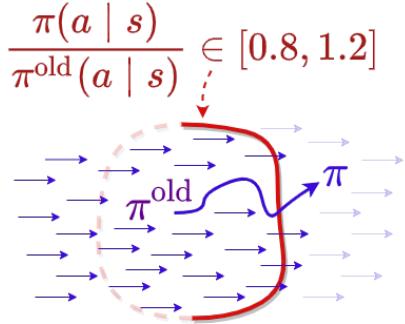
$$\mathbb{E}_{s \sim d_{\pi^{\text{old}}}(s)} \mathbb{E}_{a \sim \pi^{\text{old}}(a|s)} \min \left(\rho(\theta) A^{\pi^{\text{old}}}(s, a), \rho^{\text{clip}}(\theta) A^{\pi^{\text{old}}}(s, a) \right) \rightarrow \max_\theta \quad (5.37)$$

На этом месте гробик связи между алгоритмом и теорией нижней оценки засыпается землёй. Но что же произошло? Мы ввели обрезку (имеющую некоторый смысл trust region-a) и взятие минимума между двумя функциями потерь; если обрезка так хорошо «смоделировала» trust region, что регуляризатор (слагаемое с KL-дивергенцией) и не нужен, то какой физический смысл имеет минимум?

Итак, давайте поймём, какую роль в интуиции trust region-a играет взятие минимума, посмотрев на градиенты для одной пары s, a . Введение минимума всё ещё «выкидывает» из градиентов функционала пары s, a , на которых коэффициент $r(\theta)$ не близок к 1; или же оставляет градиент без изменений. Зануление градиента происходит в случае, если происходит сразу два события: минимум достигается на «обрезанной» версии градиента, и importance sampling вес вышел за границы. Рассмотрим всевозможные случаи:

$$\nabla_\theta \min \left(\rho(\theta) A^{\pi^{\text{old}}}(s, a), \rho^{\text{clip}}(\theta) A^{\pi^{\text{old}}}(s, a) \right) = \begin{cases} 0 & \rho(\theta) > 1 + \epsilon, \quad A^{\pi^{\text{old}}}(s, a) > 0 \\ \nabla_\theta \rho(\theta) & \rho(\theta) < 1 + \epsilon, \quad A^{\pi^{\text{old}}}(s, a) > 0 \\ 0 & \rho(\theta) < 1 - \epsilon, \quad A^{\pi^{\text{old}}}(s, a) < 0 \\ \nabla_\theta \rho(\theta) & \rho(\theta) > 1 - \epsilon, \quad A^{\pi^{\text{old}}}(s, a) < 0 \end{cases}$$

Внимательно взглянувшись в эту «таблицу», становится понятно, что происходит. Если оценка критика положительна, градиенты говорят увеличивать $\pi_\theta(a | s)$; importance sampling вес повышается, и в какой-то момент случится обрезка по $1 + \epsilon$. Но если оценка критика положительна, и градиенты указывают увеличивать вероятность, а она по какой-то причине уменьшилась (такое вполне возможно в процессе оптимизации) и даже «вылетела за границы trust region», то градиенты не зануляются: она вылетела «не с той стороны»! Симметричная ситуация случится при отрицательной оценке критика: вероятность должна уменьшаться, но сильно за счёт обрезки уменьшиться она не может, а за счёт оператора минимума при случайном увеличении градиенты продолжат тянуть вероятности «к барьеру». Мы получили этакий «полуоткрытый trust region».



5.3.5. Clipped Value Loss

Применим аналогичную идею «полуоткрытого trust region-a» для лосса критика. Пусть для данного состояния s наша аппроксимация V -функции выдаёт $V_\phi(s)$, а таргет равен y . Обычно мы бы минимизировали MSE:

$$\text{Loss}_1(\phi) := (V_\phi(s) - y)^2$$

Однако если таргет y содержит в себе в том числе какие-то Монте-Карло оценки и переиспользуется «несколько раз» из небольшого датасета, то мы не хотим на него переобучаться. Пусть на момент сбора датасета критик выдавал для данного состояния $V^{\text{old}}(s)$; таргет указывает лишь направление, в котором мы хотим изменить значение выхода критика, но, как и в любой стохастической аппроксимации, мы не хотим «заменять жёстко» выход $V_\phi(s)$ на y , а лишь сдвинуть в его сторону. Для этого добавим и вычтем в MSE $V^{\text{old}}(s)$:

$$\text{Loss}_1(\phi) = (V_\phi(s) - V^{\text{old}}(s) - (y - V^{\text{old}}(s)))^2$$

⁶в RL, как и в глубоком обучении, обычный стохастический градиентный спуск не используется, а вместо этого используется, например, Adam.

⁷если же его всё-таки оставляют, то обычно без квадратного корня.

Теперь мы сравниваем не выход сети с таргетом, а изменение значения выхода критика с «желаемым изменением» (на самом деле, просто оценкой Advantage; $y - V^{\text{old}}(s)$ используется в качестве аппроксимации $\Psi(s, a)$). Введём другую функцию потерь, с «обрезкой», которая построит «мягкий trust-region» и будет занулять градиенты, как только $V_\phi(s)$ станет непохожим на $V^{\text{old}}(s)$:

$$\text{Loss}_2(\phi) := (\text{clip}(V_\phi(s) - V^{\text{old}}(s), \epsilon, -\epsilon) - (y - V^{\text{old}}(s)))^2,$$

где ϵ — гиперпараметр. Аналогично лоссу актёра, градиенты в такой функции потерь просто будут зануляться в ситуациях, когда происходит обрезка.

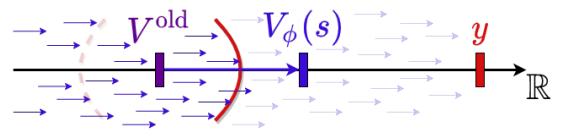
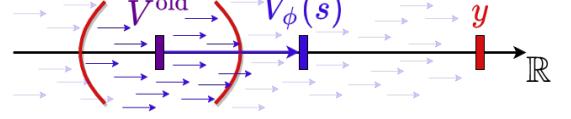
Наконец, чтобы «открыть» trust-region с одной стороны, нужно просто, по аналогии с лоссом актёра, взять максимум от этих двух функций потерь (для актёра мы брали минимум, поскольку там велась максимизация, а здесь лосс минимизируется):

$$\text{Loss}(\phi) := \max(\text{Loss}_1(\phi), \text{Loss}_2(\phi))$$

Непосредственной проверкой легко убедиться, что градиенты будут зануляться, только если наш критик вышел из trust-region-a «с правильной стороны»; а так, градиенты будут тянуть нашего критика к допустимому барьеру.



Считается, что обрезка функционала для критика менее существенна, чем для актёра. Если мы не обрежем функционал для актёра, и тот «сломается», переобучившись под сэмплы, это может сломать весь оптимизационный процесс, поскольку стратегия используется для сбора данных. Но если немного сломался критик, то это не так сильно отразится на процессе обучения, потому что актёр не опирается целиком на оценки критика. Тем не менее, раз такая возможность «зашщищаться» от переобучения под сэмплы в критике есть, то стоит ей пользоваться. Недостаток — в необходимости подобрать соответствующее масштабу V-функции значение гиперпараметра обрезки ϵ .



5.3.6. Proximal Policy Optimization (PPO)

Итоговая процедура работы алгоритма следующая. Собираем большой роллаут (порядка хотя бы 1000 шагов). Версия стратегии, использовавшаяся для сбора, обозначается как π^{old} , и вероятности, с которыми она выбирала действия, сохраняются. Для всех пар s, a высчитываются оценки Q-функции и Advantage методом GAE (5.21). К собранным данным относимся как к датасету, из которого можно брать пары $s \sim d_{\pi^{\text{old}}}(s), a \sim \pi^{\text{old}}(a | s)$ и считать Монте-Карло оценку градиента (5.36). Размер батча при сэмплировании из датасета при этом обычный для градиентных методов первого порядка, условно такой же, как был бы в A2C. По датасету нужно пройтись при этом несколько раз, теоретически — хочется как можно больше, но понятно, что чем больше расходится π_θ и π^{old} , тем менее эффективна «нижняя оценка» и тем больше данных будет резать наш клиппинг. Соответственно, количество эпох — сколько раз пройтись по датасету — является ключевым гиперпараметром. Существенно, сколько именно шагов градиентного спуска будет сделано по одному датасету; это более важно, чем размер мини-батчей, поскольку именно первое больше влияет на то, когда мы начнём «вываливаться» из trust region-a, то есть когда отступим от стратегии сбора данных достаточно далеко.

Алгоритм 21: Proximal Policy Optimization (PPO)

Гиперпараметры: M — количество параллельных сред, N — длина роллаутов, B — размер мини-батчей, n_epochs — количество эпох, λ — параметр GAE-оценки, ϵ — параметр обрезки для актёра, $\hat{\epsilon}$ — параметр обрезки для критика, $V_\phi(s)$ — нейросеть с параметрами ϕ , $\pi_\theta(a | s)$ — нейросеть для стратегии с параметрами θ , α — коэф. масштабирования лосса критика, SGD оптимизатор.

Инициализировать θ, ϕ

На каждом шаге:

1. в каждой параллельной среде собрать роллаут длины N , используя стратегию π_θ , сохраняя вероятности выбора действий как $\pi^{\text{old}}(a | s)$, а выход критика на встречающихся состояниях как $V^{\text{old}}(s) \leftarrow V_\phi(s)$
2. для каждой пары s, a из роллаутов посчитать одношаговую оценку Advantage:

$$\Psi_{(1)}(s, a) := r + \gamma(1 - \text{done}')V_\phi(s') - V_\phi(s)$$

3. посчитать GAE-оценку:

$$\Psi_{\text{GAE}}(s_{N-1}, a_{N-1}) := \Psi_{(1)}(s_{N-1}, a_{N-1})$$

4. для t от $N - 2$ до 0:

$$\bullet \quad \Psi_{\text{GAE}}(s_t, a_t) := \Psi_{(1)}(s_t, a_t) + \gamma \lambda (1 - \text{done}_t) \Psi_{\text{GAE}}(s_{t+1}, a_{t+1})$$

5. посчитать таргет для критика:

$$y(s) := \Psi_{\text{GAE}}(s, a) + V_\phi(s)$$

6. составить датасет из шестёрок $(s, a, \Psi_{\text{GAE}}(s, a), y(s), \pi^{\text{old}}(a | s), V^{\text{old}}(s))$

7. выполнить n_{epochs} проходов по роллауту, генерируя мини-батчи пятёрок $\mathbb{T} := (s, a, \Psi_{\text{GAE}}(s, a), y(s), \pi^{\text{old}}(a | s), V^{\text{old}}(s))$ размером B ; для каждого мини-батча:

- вычислить лосс критика:

$$\text{Loss}_1(\mathbb{T}, \phi) := (y(s) - V_\phi(s))^2$$

$$\text{Loss}_2(\mathbb{T}, \phi) := (y(s) - V^{\text{old}}(s) - \text{clip}(V_\phi(s) - V^{\text{old}}(s), -\hat{\epsilon}, \hat{\epsilon}))^2$$

$$\text{Loss}^{\text{critic}}(\phi) := \frac{1}{B} \sum_{\mathbb{T}} \max(\text{Loss}_1(\mathbb{T}, \phi), \text{Loss}_2(\mathbb{T}, \phi))$$

- сделать шаг градиентного спуска по ϕ , используя $\nabla_\phi \text{Loss}^{\text{critic}}(\phi)$
- нормализовать $\Psi_{\text{GAE}}(s, a)$ по батчу, чтобы в среднем значения равнялись 0, а дисперсия — 1.
- посчитать коэффициенты в importance sampling:

$$r_\theta(\mathbb{T}) := \frac{\pi_\theta(a | s)}{\pi^{\text{old}}(a | s)}$$

- посчитать обрезанную версию градиентов:

$$r_\theta^{\text{clip}}(\mathbb{T}) := \text{clip}(r_\theta(\mathbb{T}), 1 - \epsilon, 1 + \epsilon)$$

- вычислить градиент для актёра:

$$\nabla_\theta^{\text{actor}} := \frac{1}{B} \sum_{\mathbb{T}} \nabla_\theta \min \left(r_\theta(\mathbb{T}) \Psi_{\text{GAE}}(s, a), r_\theta^{\text{clip}}(\mathbb{T}) \Psi_{\text{GAE}}(s, a) \right)$$

- сделать шаг градиентного подъёма по θ , используя $\nabla_\theta^{\text{actor}}$



В Atari играх хорошим значением гиперпараметра «число эпох» считается 3, а для задач непрерывного управления — 10. Возможность несколько раз «увидеть» переход (пусть даже всего 3) вместо одного даёт существенный прирост эффективности алгоритма PPO по сравнению с A2C, поэтому на практике осмысленно использовать именно его, если нужен on-policy алгоритм.



PPO использовался для великих достижений в Dota и считается более-менее устоявшейся SOTA в RL: если нет проблем с симулятором и можно использовать on-policy алгоритмы — стоит начать с PPO, но причины, по которым он так круто работает, полностью не ясны. В частности, последовавшие исследования указывают на то, что клиппинг — не основная причина успеха, а кроется она в целом наборе удачных инженерных хаков в официальной реализации алгоритма от OpenAI. Ключевая из них приведена в описании алгоритма — нормализация $\Psi_{\text{GAE}}(s, a)$ по батчу; мы знаем, что в среднем Advantage должен быть равен нулю, и поэтому можем его центрировать. Деление же на стандартное отклонение Advantage-оценок позволяет «отнормировать» масштаб функции награды внутри самого алгоритма.

ГЛАВА 6

Continuous control

В этой главе будут рассмотрены алгоритмы для задачи непрерывного управления ($|\mathcal{A}| \subseteq [-1, 1]^A$), разыскивающие идеи Value-based и Policy gradient подходов. Мы увидим, что эти два подхода имеют много общего и отчасти даже эквивалентны, в обоих случаях получив схемы, моделирующие алгоритм Policy Iteration.

§6.1. DDPG

6.1.1. Вывод из Deep Q-learning

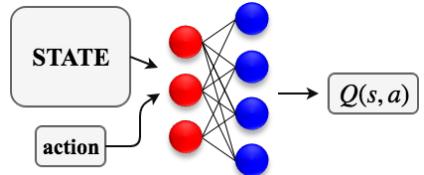
Схема DQN 15 имела принципиальный недостаток: мы не могли работать с непрерывными пространствами действий в силу необходимости постоянно считать операторы максимума и аргмаксимума

$$\max_a Q_\theta(s, a)$$

как для жадного выбора действия, так и для построения таргета в задаче регрессии.

Единственная возможная архитектура модели — приём действий на вход вместе с состояниями, тогда поиск аргмаксимума проводить, в целом, можно, но дорого: инициализируем a^0 случайно, и устраиваем градиентный подъём по входу в модель:

$$a^{k+1} = a^k + \alpha \nabla_a Q_\theta(s, a)|_{a=a^k}$$



Понятно, что такую процедуру устраивать по несколько раз за шаг дороговато. Однако, в глубинном обучении для таких проблем есть мега-универсальное решение: давайте задачу поиска аргмаксимума тоже аппроксимируем другой нейросетью! Максимум тогда можно будет считать, просто подставив в Q_θ вместо действия приближение этого аргмаксимума.

Итак, пусть $\pi_\omega(s)$ принимает на вход состояние s и выдаёт аргмаксимум текущей аппроксимации Q-функции, то есть будем добиваться $\pi_\omega(s) \approx \operatorname{argmax}_a Q_\theta(s, a)$. Понятно, как обучать такую сеть:

$$Q_\theta(s, \pi_\omega(s)) \rightarrow \max_\omega$$

Весь алгоритм DQN оставляем неизменным с единственной модификацией, что на каждом батче также нужно сделать шаг оптимизации ω . При этом каждый раз, когда в схеме необходимо считать максимум или аргмаксимум Q_θ , используется $\pi_\omega(s)$.

В стандартном алгоритме DQN нам было необходимо считать $\max_{a'} Q_{\theta-}(s', a')$, и в дефолтной версии алгоритма использовалась таргет-сеть. Технически это означает, что для таргет-сети $Q_{\theta-}(s', a')$ нам тоже нужно знать аргмаксимум, поэтому можно хранить старую версию вспомогательной функции $\pi_{\omega-}(s)$.



Считается, что это не так принципиально, поскольку использование «свежей» $\pi_\omega(s)$ при подсчёте таргета соответствует моделированию Double оценки (см. раздел 4.2.3). Все остальные модификации DQN также применимы.

Итого мы получили, что для жадного выбора действия используется $\pi_\omega(s)$ (отсюда такое обозначение этой «вспомогательной» функции — это фактически стратегия); а таргет для перехода $\mathbb{T} := (s, a, r, s')$ вычисляется по формуле

$$y(\mathbb{T}) := r + \gamma Q_{\theta-}(s', \operatorname{argmax}_{a'} Q_{\theta-}(s', a')) \approx r + \gamma Q_{\theta-}(s', \pi_{\omega-}(s))$$

Такой алгоритм называется Deep Deterministic Policy Gradient (DDPG), и название может сбить с толку: а причём здесь policy gradient?

6.1.2. Вывод из Policy Gradient

В Policy Gradient алгоритмах мы получили формулу градиента нашего функционала, «релаксировав» задачу и перейдя к оптимизации в пространстве стохастических стратегий. Если пространство действий непрерывно, то такая релаксация на самом деле не обязательна. Предположим¹ дифференцируемость любых Q-функций $Q^\pi(s, a)$ по действиям a . Попробуем посчитать градиент по параметрам стратегии в случае детерминированной стратегии $\pi_\theta(s)$:

Теорема 63 — Deterministic Policy Gradient: В непрерывных пространствах действий в предположении дифференцируемости Q-функций по действиям:

$$\nabla_\theta J(\pi_\theta) = \frac{1}{1-\gamma} \mathbb{E}_{d_{\pi_\theta}(s)} \nabla_\theta \pi_\theta(s) \nabla_a Q^\pi(s, a)|_{a=\pi_\theta(s)} \quad (6.1)$$

Доказательство.

$$\nabla_\theta V^{\pi_\theta}(s) = \{VQ \text{ уравнение (3.6)}\} = \nabla_\theta \mathbb{E}_{a \sim \pi_\theta(s)} Q^{\pi_\theta}(s, a) = \nabla_\theta Q^{\pi_\theta}(s, \pi_\theta(s)) = (*)$$

Заметим, что в последнем выражении при малом изменении θ поменяется не только $\pi_\theta(s)$, но и сама оценочная функция Q^{π_θ} . Считая, что якобиан функции $\mathbb{R}^n \rightarrow \mathbb{R}^m$ имеет размерность $n \times m$, и обозначая размерность действий как A , а размерность параметров θ буквой d , получаем следующие размерности градиентов:

$$\nabla_\theta Q^{\pi_\theta}(s, a) \in \mathbb{R}^{d \times 1} \quad \nabla_a Q^{\pi_\theta}(s, a) \in \mathbb{R}^{A \times 1} \quad \nabla_\theta \pi_\theta(s) \in \mathbb{R}^{d \times A}$$

Тогда продолжение вычисления выглядит так:

$$(*) = \nabla_\theta \pi_\theta(s) \nabla_a Q^\pi(s, a)|_{a=\pi_\theta(s)} + \nabla_\theta Q^{\pi_\theta}(s, a)|_{a=\pi_\theta(s)},$$

где последнее слагаемое — якобиан Q^{π_θ} при фиксированном a по параметрам стратегии π_θ , которую он оценивает. Отдельно это слагаемое имеет вид:

$$\nabla_\theta Q^{\pi_\theta}(s, a) = \{QV \text{ уравнение (3.5)}\} = \gamma \mathbb{E}_{s'} \nabla_\theta V^{\pi_\theta}(s')$$

Получаем рекурсивную формулу, аккуратно собирая которую, получим:

$$\nabla_\theta V^{\pi_\theta}(s) = \mathbb{E}_{\tau \sim \pi|s_0=s} \sum_{t \geq 0} \gamma^t \nabla_\theta \pi_\theta(s_t) \nabla_a Q^\pi(s_t, a)|_{a=\pi_\theta(s_t)}$$

Осталось только применить теорему 57 об эквивалентной форме мат.ожидания по траекториям для

$$f(s, a) = \nabla_\theta \pi_\theta(s) \nabla_a Q^\pi(s, a)|_{a=\pi_\theta(s)} \quad \blacksquare$$

Сразу построим суррогатную функцию для такой формулы градиента:

$$\mathcal{L}_\pi(\theta) := \frac{1}{1-\gamma} \mathbb{E}_{d_\pi(s)} Q^\pi(s, \pi_\theta(s))$$

Действительно, если мы посчитаем градиент этой функции по θ , то мы просто получим формулу chain rule для оптимизации параметров стратегии через градиент Q-функции по действиям. Иными словами, градиент по параметрам детерминированной стратегии указывает просто проводить policy improvement: выбирать те действия, для которых Q-функция больше, используя её градиент по действиям.

Если мы хотим построить Actor-Critic схему, воспользовавшись такой формулой, нам придётся аппроксимировать Q-функцию $Q_\omega(s, a) \approx Q^\pi(s, a)$ и явно использовать её градиент по действиям, надеясь на то, что $\nabla_a Q_\omega(s, a) \approx \nabla_a Q^\pi(s, a)$. Таким образом, обойтись обучением лишь V-функции или пользоваться многошаговыми оценками не получится, а качество обучения стратегии будет упираться в качество обучения критика, поэтому всеми преимуществами off-policy подхода в такой формуле воспользоваться не удастся.

Но можем ли мы воспользоваться формулой в off-policy режиме? Вообще говоря, нет, поскольку состояния должны приходить согласно формуле из распределения $d_{\pi_\theta}(s)$. Однако, как мы обсуждали в разделе 5.1.7, мы понимаем, что если мы будем оптимизировать Q-функцию по действиям, то как бы мы это не делали (из какого бы распределения не брали состояния, в которых мы изменяем стратегию), мы всё равно сможем увеличить значение нашего функционала, поскольку мы проводим policy improvement. Это означает, что если мы в формуле (6.1) заменим $d_{\pi_\theta}(s)$ на что-либо другое, полученная формула «градиента» будет всё равно направлением улучшения стратегии, пусть и не направлением локально максимального увеличения функционала, что верно для

¹ даже если это не так, в будущем мы всё равно будем приближать эти Q-функции нейросетями, для которых всегда справедлива дифференцируемость по входу.

честного градиента. Итого, будем сэмплировать батч состояний из реплей буфера и делать шаг градиентного подъёма:

$$\theta \leftarrow \theta + \alpha \mathbb{E}_s \nabla_\theta \pi_\theta(s) \nabla_a Q^\pi(s, a)|_{a=\pi_\theta(s)},$$

где состояния s приходят из произвольного распределения (например, из реплей буфера). Это эквивалентно одному шагу градиентной оптимизации суррогатной функции:

$$\mathbb{E}_s Q^\pi(s, \pi_\theta(s)) \rightarrow \max_\theta \quad (6.2)$$

Q-функцию $Q_\omega(s, a) \approx Q^\pi(s, a)$, необходимую для такой оптимизации, будем тоже учитывать в off-policy режиме с одношаговых оценок: ему для данной пары s, a требуется лишь сэмпл s' , поэтому такого критика можно обучать по переходам $\mathbb{T} := (s, a, r, s')$ из буфера на таргеты

$$y(\mathbb{T}) := r + \gamma Q_{\omega^-}(s', \pi(s'))$$

Одношаговые таргеты имеют сильное смещение (сильно опираются на выход нашей же нейросети), и поэтому для стабилизации процесса требуется использование таргет-сетей. Тут-то и можно, заметить, что...

6.1.3. Связь между схемами

Теорема 64: Предыдущие две схемы (вывод через DQN и через Policy Gradient) эквивалентны полностью.

Доказательство. Методом пристального взгляда. ■

Итак, DQN для непрерывных действий и Policy Gradient для детерминированных стратегий — это одно и то же. Поймём, как так случилось.

Мы двумя путями² пришли к Policy Iteration схеме 8. Действительно: мы параллельно ведём два оптимизационных процесса: оцениваем $Q \approx Q^\pi$ для текущей стратегии π и учим $\pi(s) \leftarrow \underset{a}{\operatorname{argmax}} Q(s, a)$, то есть проводим Policy Evaluation и Policy Improvement. При этом на этапе Policy Improvement мы делаем апдейт стратегии сразу для всех состояний, и никаких требований на «распределение» состояний мы можем не накладывать в силу теоремы 17 о Policy Improvement.

Таким образом, обоснование, почему в выводе через Policy Gradient мы можем забить на $d_\pi(s)$ и брать состояния из буфера, можно сформулировать так: мы отказываемся от Policy Gradient подхода, в котором мы оптимизируем функционал (1.5) напрямую, и переходим к Policy Iteration схеме 8.



А ещё подметим, что схема шибко похожа на GAN: критик в этом алгоритме «учит» функцию потерь для стратегии. Так что вполне естественно, что мы принципиально используем непрерывность пространства действий. Эта аналогия также объясняет, почему схема DDPG нестабильна; как только что-то ломается в одной из двух оптимизируемых функций (критике или актёре), другому тут же становится плохо. Поэтому схема жутко чувствительна к гиперпараметрам; пожалуй, это один из самых нестабильных алгоритмов RL.

6.1.4. Ornstein–Uhlenbeck Noise

В рассмотренной схеме из-за использования детерминированной стратегии, как и в DQN, возникает проблема exploration-exploitation-a. В непрерывных пространствах действий вместо ϵ -жадной стратегии возможно добавлять к выбранным стратегией действиям шум из нормального распределения:

$$a_t := \pi(s_t) + \varepsilon_t, \quad \varepsilon_t \sim \mathcal{N}(0, \sigma^2 I)$$

Гиперпараметр σ , контролирующий магнитуду вспрыкиваемого шума, нужно подбирать, его также можно, например, постепенно затухать к нулю с ходом обучения. Однако, такое вспрыкивание шума предполагает, что исследование в соседние моменты времени независимо.

Пример 88: Если действия робота — это направление движения (например, поворот руля управляемой машины), а один шаг в среде это доля секунды, странно проводить исследования, рандомно «подёргиваясь» пару раз в секунду. Хочется целенаправленно смещать траекторию: если мы решили в целях исследования повернуть руль чуть правее, чем говорит наша детерминированная стратегия, следует сохранить это смещение руля вправо и в дальнейшем. Для моделирования этого шум должен быть скоррелированным: поэтому вместо независимого шума имеет смысл добавлять случайный процесс, колеблящийся вокруг нуля.

²что в очередной раз означает, что между Value-based подходом и Policy Gradient подходом есть тесная связь.

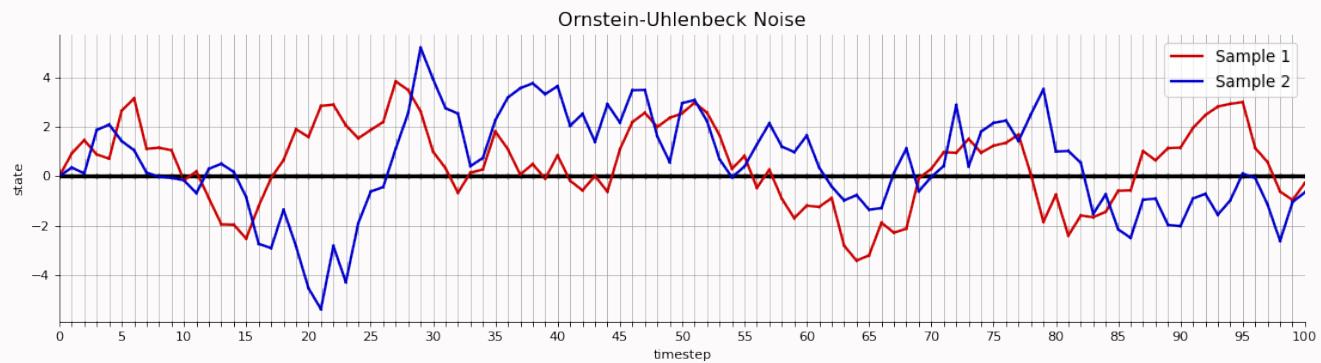
Определение 79: *Шум Орнштейна — Уленбека* (Ornstein–Uhlenbeck noise), в начале эпизода инициализированный нулём, задаётся рекурсивно как:

$$\varepsilon_{t+1} := \alpha \varepsilon_t + \mathcal{N}(0, \sigma^2 I),$$

где $\alpha \leq 1$ и σ — гиперпараметры.

По сути, это просто кумулятивный шум, который с коэффициентом α прибивается к нулю. Если $\alpha = 0$, получаем обычный независимый шум из нормального распределения. Одно из преимуществ такого эксплорейшна — считается, что его параметры можно со временем не менять, то есть даже при околооптимальном поведении такой шум будет исследовать разумные альтернативы вместо рандомных подёргиваний.

Пример 89: На рисунке приведён пример поведения процесса Орнштейна-Уленбека для $\alpha = 0.9, \sigma = 1$.



6.1.5. Deep Deterministic Policy Gradient (DDPG)

Мы собрали алгоритм DDPG — off-policy алгоритм для непрерывных пространств действий. Несмотря на название, по свойствам этот алгоритм похож именно на DQN, и обладает аналогичными недостатками и преимуществами.

Алгоритм 22: Deep Deterministic Policy Gradient (DDPG)

Гиперпараметры: B — размер мини-батчей, β — коэф. экспоненциального сглаживания для таргет-сеток, α, σ — параметры шума, $Q_\theta(s, a)$ — нейросеть с параметрами θ , $\pi_\omega(s)$ — детерминированная стратегия с параметрами ω , SGD-оптимизаторы.

Инициализировать θ, ω произвольно

Положить $\theta^- := \theta$

Положить $\omega^- := \omega$

Инициализировать шум $\varepsilon := 0$

Пронаблюдать s_0

На **очередном шаге t :**

1. обновить шум $\varepsilon \leftarrow \alpha \varepsilon + \hat{\varepsilon}$, где $\hat{\varepsilon} \sim \mathcal{N}(0, \sigma^2 I)$
2. выбрать $a_t := \pi_\omega(s_t) + \varepsilon$
3. пронаблюдать $r_t, s_{t+1}, \text{done}_{t+1}$
4. добавить пятёрку $(s_t, a_t, r_t, s_{t+1}, \text{done}_{t+1})$ в реплей буфер
5. засэмплировать мини-батч размера B из буфера
6. сделать один шаг градиентного подъёма по ω :

$$\frac{1}{B} \sum_{s \in B} Q_\theta(s, \pi_\omega(s)) \rightarrow \max_{\omega}$$

7. для каждого перехода $\mathbb{T} := (s, a, r, s', \text{done})$ посчитать таргет:

$$y(\mathbb{T}) := r + \gamma(1 - \text{done})Q_{\theta^-}(s', \pi_{\omega^-}(s'))$$

8. сделать один шаг градиентного спуска по θ :

$$\frac{1}{B} \sum_{\mathbb{T}} (Q_\theta(s, a) - y(\mathbb{T}))^2 \rightarrow \min_{\theta}$$

9. если $t \bmod K = 0$:

$$\begin{aligned}\theta^- &\leftarrow (1 - \beta)\theta^- + \beta\theta \\ \omega^- &\leftarrow (1 - \beta)\omega^- + \beta\omega\end{aligned}$$



DDPG за счёт off-policy режима может оказаться эффективнее PPO в задаче локомоции 25. В этой задаче нет тех проблем, из-за которых off-policy обучение может сломаться или плохо работать: там нет сильно отложенного сигнала (плохое действие — и существо сразу падает, хорошее действие — и существо продвигается вперёд и получит положительное подкрепление), Q-функция как функция от действий не похожа на плато (интуитивно, в большинстве состояний есть «хорошие» действия с высоким значением $Q(s, a)$, которые позволяют существу продолжать бежать, и «плохие» с низким значением, которые нарушают баланс устойчивости существа и приводят к его падению), а функция награды очень плотная и информативная, из-за чего полезно иметь возможность все переходы много раз из буфера вспоминать и обучаться восстанавливать хотя бы $r(s, a)$, содержащуюся в одношаговых таргетах, из разных пар s, a .

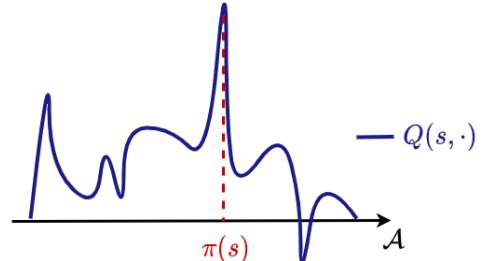
6.1.6. Twin Delayed DDPG (TD3)

TD3 — набор из трёх эвристических костылей, которые можно навесить над DDPG для существенного повышения стабильности происходящих процессов.

Одна из главных проблем DDPG — унаследованная от DQN проблема переоценивания (см. раздел 4.2.1). Хотя сейчас в формулах в явном виде не присутствует оператор максимума, на самом деле он всё равно есть: наш актёр учится «взламывать» критика и находить те действия, где погрешность нашей аппроксимации истинной Q-функции положительна. Поэтому оценка $Q_\theta(s, \pi_\omega(s))$ практически всегда завышенно оценивает $\max_a Q^\pi(s, a)$, и если это завышение попадает в целевую переменную для обучения модели Q-функции, начинается цепная реакция.

Более того, в пространстве действий могут обнаружиться узкие области, в которых наша неидеальная нейросетевая аппроксимация Q-функции имеет всплеск; актёр может научиться «взламывать» критика, используя эти всплески. Поэтому первая идея борьбы с этим эффектом заключается в том, чтобы при построении таргета зашумить выход нашей стратегии при помощи некоторого шума. Шум здесь не должен быть очень большим по модулю (важно, что этот шум не имеет смысла «исследований»): обычно сэмплируют зашумление из гауссианы и обрезают, чтобы получить что-то не очень большое по значению:

$$\varepsilon' \sim \text{clip}(\mathcal{N}(0, \hat{\sigma} I), -c, c)$$



Во-вторых, воспользуемся Twin-трюком, который мы обсуждали в разделе 4.2.2, а то есть будем обучать две Q-функции по общему буферу и использовать в таргетах минимум из двух оценок критиков. Таргет-сеть обычно заводят для каждой из двух копий; а вот стратегию предлагается оставить для них общую, то есть использовать одно и то же «приближение аргмакса» для обоих Q-функций, поскольку в такой схеме она моделируется отдельной нейросетью. Итого формула таргета получается общая для двух критиков:

$$y(\mathbb{T}) := r + \gamma \min_{i \in \{1, 2\}} Q_{\theta_i}(s', \pi_{\omega^-}(s') + \varepsilon')$$

При обучении актёра можно как оставить оптимизацию при помощи первого из двух критиков

$$Q_{\theta_1}(s, \pi_\omega(s)) \rightarrow \max_{\omega},$$

и тогда актёр не будет видеть одну из Q-функций (и там, где у первого критика будет взломанное актёром завышение, у второго критика, мы надеемся, завышение или занижение будет условно равновероятно), так и использовать снова минимум из двух критиков:

$$\min_{i \in \{1, 2\}} Q_{\theta_i}(s, \pi_\omega(s)) \rightarrow \max_{\omega}$$

Наконец, в-третьих, обновление весов стратегии будем делать реже, чем обновление весов Q-функции: это позволит «поучить» Q-функцию приближать Q^π именно для текущей, свежей версии π , прежде чем использовать её градиент для оптимизации стратегии; здесь наблюдается полная аналогия с GAN-ами, где тоже иногда помогают подобные фокусы.



Третья эвристика кажется наименее существенной, поскольку авторы предлагали делать $N = 2$ шагов обучения критика на один шаг обучения актёра.

Алгоритм 23: Twin Delayed DDPG (TD3)

Гиперпараметры: B — размер мини-батчей, N — периодичность обновления весов стратегии, α, σ — параметры шума, $\hat{\sigma}, c$ — параметры шума для добавки к действиям для таргета, β — коэф. экспоненциального сглаживания для таргет-сеток, $Q_{\theta_1}(s, a), Q_{\theta_2}(s, a)$ — нейросетки с параметрами θ_1, θ_2 , $\pi_\omega(s)$ — детерминированная стратегия с параметрами ω , SGD-оптимизаторы.

Инициализировать $\theta_1, \theta_2, \omega$ произвольно

Инициализировать таргет-сетки $\theta_1^- := \theta_1, \theta_2^- := \theta_2, \omega^- := \omega$

Инициализировать шум $\varepsilon_0 := 0$

Пронаблюдать s_0

На **очередном шаге t :**

1. посчитать шум $\varepsilon_t := \alpha \varepsilon_{t-1} + \varepsilon$, где $\varepsilon \sim \mathcal{N}(0, \sigma^2 I)$
2. выбрать $a_t := \pi_\omega(s_t) + \varepsilon_t$
3. пронаблюдать $r_t, s_{t+1}, \text{done}_{t+1}$
4. добавить пятёрку $(s_t, a_t, r_t, s_{t+1}, \text{done}_{t+1})$ в реплей буфер
5. засэмплировать мини-батч размера B из буфера
6. для каждого перехода $\mathbb{T} := (s, a, r, s', \text{done})$ посчитать таргет:

$$\varepsilon' \sim \text{clip}(\mathcal{N}(0, \hat{\sigma} I), -c, c)$$

$$y(\mathbb{T}) := r + \gamma(1 - \text{done}) \min_{i \in \{1, 2\}} Q_{\theta_i^-}(s', \pi_{\omega^-}(s') + \varepsilon')$$

7. сделать один шаг градиентного спуска по θ_1 и θ_2 :

$$\frac{1}{B} \sum_{\mathbb{T}} (Q_{\theta_1}(s, a) - y(\mathbb{T}))^2 \rightarrow \min_{\theta_1}$$

$$\frac{1}{B} \sum_{\mathbb{T}} (Q_{\theta_2}(s, a) - y(\mathbb{T}))^2 \rightarrow \min_{\theta_2}$$

8. если $t \bmod N = 0$:

- сделать один шаг градиентного подъёма по ω :

$$\frac{1}{B} \sum_{s \in B} Q_{\theta_1}(s, \pi_\omega(s)) \rightarrow \max_{\omega}$$

- обновить таргет-сети:

$$\theta_1^- \leftarrow (1 - \beta)\theta_1^- + \beta\theta_1$$

$$\theta_2^- \leftarrow (1 - \beta)\theta_2^- + \beta\theta_2$$

$$\omega^- \leftarrow (1 - \beta)\omega^- + \beta\omega$$

6.1.7. Обучение стохастических политик

Среди преимуществ on-policy режима, которые мы потеряли в off-policy алгоритмах, мы упоминали возможность обучать стохастические политики. Действительно, ряд наших проблем с нестабильностью в DDPG был связан с тем, что мы учим детерминированную стратегию: нужны костыли для решения проблемы exploration-a, актёр может «взламывать» нашу аппроксимацию критика и приводить к overestimation-y, и всё это приходится лечить очередной порцией эвристик. Когда же в on-policy подходе мы можем получить, если так можно сказать, «естественный» exploration за счёт обучения стохастической стратегии. Обязательно ли в off-policy режиме учить именно детерминированную стратегию?

На самом деле нет. В теории есть возможность обучать и стохастического актёра, хотя точные формулы градиента будут немного отличаться в зависимости от выбранной параметризации политики. Итак, допустим мы моделируем актёра $\pi_\theta(a | s)$ в классе стохастических стратегий.

Определение 80: Скажем, что для параметризации $\pi_\theta(a | s)$ применим *репараметризационный трюк* (reparameterization trick), если сэмплирование $a \sim \pi_\theta(a | s)$ эквивалентно сэмплированию шума из некоторого не зависящего от параметров распределения $\varepsilon \sim p(\varepsilon)$ и его дальнейшего детерминированного преобразования $a = f_\theta(s, \varepsilon)$.

Пример 90: Пусть наша стратегия параметризована нормальным распределением:

$$\pi_\theta(a | s) := \mathcal{N}(\mu_\theta(s), \sigma_\theta(s)^2 I)$$

Тогда для неё применим репараметризационный трюк: сэмплирование действий эквивалентно $a := \mu_\theta(s) + \varepsilon \odot \sigma_\theta(s)$, где $\varepsilon \sim \mathcal{N}(0, I)$, \odot — поэлементное перемножение.

Пример 91: Семейство детерминированных стратегий $\pi_\theta(s)$ тоже можно считать таким «вырожденным» примером параметризаций, для которой можно проворачивать репараметризационный трюк: просто шум ε считаем «пустым».

Можно заметить, что если для семейства стратегий применим репараметризационный трюк, то в выводе формулы Policy Gradient можно пользоваться им вместо REINFORCE: фактически, в теореме 63 мы этим воспользовались.

Теорема 65: Если для $\pi_\theta(a | s)$ применим репараметризационный трюк, то:

$$\nabla_\theta J(\pi_\theta) = \frac{1}{1 - \gamma} \mathbb{E}_{d_{\pi_\theta}(s)} \mathbb{E}_{\varepsilon \sim p(\varepsilon)} \nabla_\theta f_\theta(s, \varepsilon) \nabla_a Q^\pi(s, a)|_{a=f_\theta(s, \varepsilon)} \quad (6.3)$$

Доказательство. Полностью полностью повторяет вывод теоремы 63. ■

Таким образом, все идеи DDPG расширяются на этот случай. Убирая из формулы градиента частоты посещения состояний и переходя к policy iteration схеме, получаем следующий функционал:

$$\mathbb{E}_s \mathbb{E}_{a \sim \pi_\theta(a | s)} Q_\omega(s, a) \rightarrow \max_\theta, \quad (6.4)$$

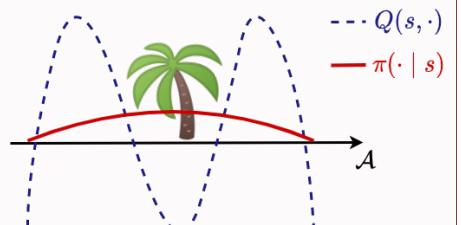
где состояния берутся из буфера. В силу репараметризационного трюка мы легко справимся со взятием градиента на практике:

$$\nabla_\theta \mathbb{E}_{a \sim \pi_\theta(a | s)} Q_\omega(s, a) = \mathbb{E}_{\varepsilon \sim p(\varepsilon)} \nabla_\theta Q_\omega(s, f_\theta(s, \varepsilon))$$

Теперь стохастическая оценка градиента по θ получается напрямую. Понятно, что мы получили обобщение формулы (6.2); мы оптимизируем актёра при помощи градиентов из критика, но дополнительно вприскиваем шум в действия для получения стохастической стратегии.

Недостатком гауссианы является то, что она унимодальная, что может приводить к бедам.

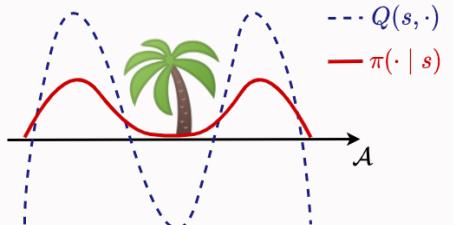
Пример 92: Представьте, что вы хотите обехать дерево. Вы можете обехать его справа, можете слева. Критик сообщает вам высокие значения слева, и справа, и оптимизируя (6.4) в классе гауссиан, можно получить стратегию, которая с наибольшей вероятностью выбирает действие «врезаться в дерево».



Поэтому мы можем захотеть выбрать более сложную параметризацию стратегии, для которой не применим репараметризационный трюк.

Пример 93: Например, можно использовать смесь гауссиан. Тогда при использовании K компонент смеси актёр для данного состояния s выдаёт следующие величины: K суммирующихся в единицу чисел $w_i(s, \theta)$, а также K векторов $\mu_i(s, \theta), \sigma_i(s, \theta)$, где $i \in \{1, 2, \dots, K\}$. Итоговое распределение полагается

$$\pi_\theta(a | s) := \sum_{i=1}^K w_i(s, \theta) \mathcal{N}(\mu_i(s, \theta), \sigma_i(s, \theta)^2 I)$$



Тогда для оптимизации (6.4) нам придётся использовать REINFORCE, обладающий более высокой дисперсией оценок, для сбивания которой необходимо использование бэйзлайна:

Утверждение 67: Градиент (6.4) по параметрам актёра θ равен

$$\nabla_\theta = \mathbb{E}_{a \sim \pi_\theta(a|s)} \nabla_\theta \log \pi_\theta(a | s) (Q_\omega(s, a)(s, a) - b(s)),$$

где $b(s)$ — бэйзлайн, произвольная функция от состояний.

Естественно, эту формулу можно интерпретировать как то, что в обычной формуле policy gradient (5.14) мы аналогично DDPG «отказались» от сэмплирования состояний из частот посещения текущей стратегии ради off-policy режима работы. Ну и хорошим бэйзлайном будет, соответственно, $V_\omega(s) := \mathbb{E}_{a \sim \pi_\theta(a|s)} Q_\omega(s, a)$. Чтобы посчитать такое значение, можно либо использовать Монте-Карло оценку, либо учить отдельную нейросеть, аппроксимирующую V-функцию (целевой переменной для входа s тогда будет $Q_\omega(s, a)$, где $a \sim \pi_\theta(a | s)$).

Обсудим ещё одну тонкость. В задачах непрерывного управления мы обычно работаем с пространством действий $|\mathcal{A}| \subseteq [-1, 1]^A$, и хотелось бы, чтобы наши семейства стратегий тоже выдавали действия именно из этого диапазона. Когда мы работали с детерминированными стратегиями, можно было учесть это в параметризации простым навешиванием гиперболического тангенса \tanh на последний слой сети. А что, если мы используем гауссиану или смесь гауссиан?



Как ни странно, игнорирование этого нюанса может всё равно сработать на практике. То есть, актёр моделирует гауссиану, и считает, что он отправил сэмплы a в среду, которые могут выходить за требуемый диапазон. Неожиданное преимущество такого подхода в том, что иногда в задачах «оптимальные» действия находятся на границе и равны +1 или -1 в каких-то компонентах, и тогда такая необрезанная гауссиана может довольно удобно «часто» сэмплировать подобные действия. Тем не менее, чаще имеет смысл всё же учесть домен в параметризации.

Тогда можно применять гиперболический тангенс к сэмплам из «необрезанной» модели. То есть, стратегия объявляется следующей: $\pi_\theta(a | s) := \tanh(u)$, где $u \sim \mu_\theta(u | s)$, и $u \in \mathbb{R}^A$. Понятно, как тогда применять ре параметризационный трюк, но если мы пользуемся этой идеей вместе с REINFORCE (например, для обучения смеси гауссиан или в on-policy алгоритмах), то нам нужно уметь считать $\log \pi_\theta(a | s)$. Для этого нужно вспоминать правило замены переменных в плотностях (см., например, [википедию](#)).

Теорема 66 — Формула замены переменной в плотности: Пусть $\pi_\theta(a | s) := g(u)$, где $g: \mathbb{R}^A \rightarrow \mathbb{R}^A$, и $u \sim \mu_\theta(u | s)$. Тогда:

$$\log \pi_\theta(a | s) = \log \mu_\theta(a | s) - \log |\det \nabla_u g| \quad (6.5)$$

Без доказательства. ■

Пример 94: Например, для функции $g(u) := \tanh(u)$ якобиан $\nabla_u g$ есть диагональная матрица (поскольку преобразование поэлементное), и его определитель равен покомпонентному произведению:

$$\begin{aligned} \det \nabla_u g(u) &= \prod_{i=1}^A \nabla_{u_i} \tanh(u_i) = \\ &= \{\text{производная гиперболического тангенса}\} = \prod_{i=1}^A (1 - \tanh^2(u_i)) \end{aligned}$$

Заметим, что все компоненты положительные ($\tanh(u_i) \leq 1$), поэтому модуль из формулы (6.5) брать не нужно. Подставляя в (6.5), получаем окончательно:

$$\log \pi_\theta(a | s) = \log \mu_\theta(a | s) - \sum_{i=1}^A \log(1 - \tanh^2(u_i))$$

Итак, теоретически возможность обучать стохастичные стратегии в off-policy режиме есть. Что тогда мешает в DDPG воспользоваться этим и использовать гауссиану или смесь гауссиан? Дело в том, что мы помним, что направление оптимизации актёра — детерминированная стратегия: мы идём в сторону жадной стратегии $\pi_\theta(s) = \underset{a}{\operatorname{argmax}} Q_\omega(s, a)$. Схлопывание стохастичной стратегии в детерминированную чревато численными проблемами: например, для гауссианы дисперсия начнёт уходить в ноль, градиенты могут начать взрываться.

Можно, конечно, попытаться как-то побороться с этим эффектом, например, при помощи эвристики, которой часто пользуются в on-policy методах — добавкой энтропийного лосса. Напомним: чем больше значение энтропии для распределения, тем оно «менее вырожденное»:

Определение 81: Энтропией распределения $\pi(a)$ называется

$$\mathcal{H}(\pi(a)) := -\mathbb{E}_{\pi(a)} \log \pi(a) \quad (6.6)$$

И в policy gradient алгоритмах часто в формулу градиента добавляют слагаемое $\nabla_\theta \mathcal{H}(\pi_\theta(\cdot | s))$, которое поощряет высокую энтропию стратегии. Однако в on-policy режиме Q-функция заменилась на оценку и была стохастичной. В off-policy же актёр имеет куда больше шансов «переобучиться» под критика, и энтропийный лосс придётся тогда выставлять с большим коэффициентом.

Вместо подобных плясок с бубном хотелось бы, чтобы подобных проблем в принципе не возникало. Конечно, детерминированность оптимальной стратегии — особенность постановки задачи RL, и поэтому если мы хотим, чтобы таких эффектов в оптимизационных процессах не было, нам придётся найти какую-то альтернативную постановку задачи. Оказывается, такая альтернативная формулировка есть, и она бывает крайне удобна. В ней оптимальные стратегии уже будут стохастичны, и ряд численных проблем, а также проблем с exploration-ом, отпадёт; в частности, она «обоснует» добавку градиента энтропии в формулу градиента.

§6.2. Soft Actor-Critic

6.2.1. Maximum Entropy RL

Определение 82: Задачей *Maximum Entropy RL* является максимизация функционала

$$J_{\text{soft}}(\pi) := \mathbb{E}_{\tau \sim \pi} \sum_{t \geq 0} \gamma^t [r_t + \alpha \mathcal{H}(\pi(\cdot | s))] \rightarrow \max_{\pi} \quad (6.7)$$

где α — гиперпараметр, называемый *температурой* (temperature).

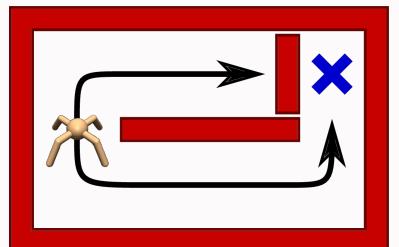
Утверждение 68: Задача (6.7) эквивалентна

$$J_{\text{soft}}(\pi) := \mathbb{E}_{\tau \sim \pi} \sum_{t \geq 0} \gamma^t [r_t - \alpha \log \pi(a_t | s_t)] \rightarrow \max_{\pi},$$

Доказательство. Следует из определения энтропии (6.6); ведь мат.ожидание по действиям присутствует в мат.ожидании по траекториям. ■

Интуиция такого функционала проста: мы говорим, что хотим не просто оптимальную стратегию, а самую стохастичную около-оптимальную стратегию (коэффициент α масштабирует добавку и определяет важность добавленного слагаемого). Цель — избегать локальных оптимумов в среде.

Пример 95: Представим, что у агента есть два пути, и по мере углубления награда на каждом пути одинаково растёт. Пусть первый путь заканчивается тупиком и суммарно позволяет набрать не более +100, а на втором тупик стоит чуть дальше и даёт +110. Во время обучения агент может уловить награду вдоль первого пути и учиться углубляться в него, игнорируя исследование второго пути, даже если агент умеет набирать там награду как на первом; за счёт бонуса за наиболее стохастичную стратегию агент мотивирован в течение обучения в начале эпизодов случайно выбирать между двумя путями. То есть, энтропийный бонус помогает избегать подобных «локальных максимумов» в среде.



Можно считать, что в данном фреймворке мы на самом деле лишь чуть-чуть модифицировали награду в среде:

$$r_{\text{soft}}(s, a) := r(s, a) - \alpha \log \pi(a | s) \quad (6.8)$$

Построенную теорию теперь нужно перепроверять, поскольку награда r стала зависеть напрямую от вероятностей, выдаваемых стратегией (такого в формализме MDP не предполагалось), да и очевидно, что не все утверждения переносятся на новую постановку:

Утверждение 69: Оптимальной детерминированной стратегии может не существовать.

Доказательство. В MDP, где награда всегда 0, оптимальна стратегия с максимальной энтропией. ■

Однако принцип Policy Iteration — чередование этапов оценивания и улучшения стратегии — остаётся для этой альтернативной постановки задачи неизменным. Чтобы подчеркнуть, что речь идёт именно о постановке Maximum Entropy RL, ко всем терминам из обычной теории добавляют слово *soft* («мягкие»), подразумевая, что энтропийный бонус «сглаживает» стратегию, предупреждая её вырождение в детерминированную. Поэтому будем обозначать оценочные функции в рамках данного фреймворка как $Q_{\text{soft}}^{\pi}, V_{\text{soft}}^{\pi}$. Также без ограничения общности далее будем считать $\alpha = 1$, так как всегда можно перемасштабировать награду $r(s, a)$ без изменения задачи.



Гиперпараметр температуры является основным недостатком идеи Maximum Entropy RL — на практике его обычно очень сложно подбирать, а он существенно влияет на то, какая стратегия получится в итоге обучения.

Мы также должны договориться о том, в какой момент во время взаимодействия приходит энтропийный бонус. Формула (6.8) предполагает, что после выбора действия a из состояния s помимо награды $r(s, a)$ агент ещё поощряется $\log \pi(a | s)$; однако мы договоримся по-другому³. Будем считать, что агент, приходя в некоторое состояние s , получает бонус в размере $\mathcal{H}(\pi(\cdot | s)) := -\mathbb{E}_{a \sim \pi(a|s)} \log \pi(a | s)$ (если состояние не терминальное), затем сэмплирует действие a , получает бонус $r(s, a)$ и потом наблюдает s' . В такой договорённости уравнения для мягких оценочных функций выглядят так:

Теорема 67 — Мягкие уравнения связи:

$$Q_{\text{soft}}^{\pi}(s, a) = r(s, a) + \gamma \mathbb{E}_{s'} V_{\text{soft}}^{\pi}(s') \quad (6.9)$$

$$V_{\text{soft}}^{\pi}(s) = \mathbb{E}_{\pi(a|s)} [Q_{\text{soft}}^{\pi}(s, a) - \log \pi(a | s)] \quad (6.10)$$

Доказательство. По определению с учётом договорённости. ■

Теорема 68 — Мягкие уравнения Беллмана (soft Bellman equations):

$$Q_{\text{soft}}^{\pi}(s, a) = r(s, a) + \gamma \mathbb{E}_{s'} \mathbb{E}_{a'} [Q_{\text{soft}}^{\pi}(s', a') - \log \pi(a' | s')] \quad (6.11)$$

$$V_{\text{soft}}^{\pi}(s) = \mathbb{E}_a [r(s, a) - \log \pi(a | s) + \gamma \mathbb{E}_{s'} V_{\text{soft}}^{\pi}(s')] \quad (6.12)$$

6.2.2. Soft Policy Evaluation

Имея на руках мягкие уравнения Беллмана, можно сразу же построить алгоритм оценивания стратегии (*Soft Policy Evaluation*), обучения мягкой Q- или V-функции по заданной стратегии π . Технически, необходимо только проверить, что правые части мягких уравнений Беллмана являются сжатиями:

Теорема 69: Операторы, стоящие в правой части мягких уравнений Беллмана, являются сжимающими с коэффициентом γ по метрике

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

Доказательство. Покажем для мягкой V-функции. Пусть $\mathfrak{B}_{\text{soft}}$ — оператор, стоящий в правой части (6.12), и пусть даны две V-функции V_1, V_2 . Тогда:

$$|[\mathfrak{B}_{\text{soft}} V_1](s) - [\mathfrak{B}_{\text{soft}} V_2](s)| = \gamma |\mathbb{E}_a \mathbb{E}_{s'} [V_1(s') - V_2(s')]|,$$

поскольку энтропия стратегии π вместе с наградой за шаг одинакова для V_1 и V_2 и потому сокращается. Дальше, как и для обычных V-функций, можно просто оценить это выражение сверху $\gamma \rho(V_1, V_2)$, заканчивая доказательство. ■

³такая договорённость удобнее, поскольку после сэмплирования действия a в состоянии s неудобно учитывать зависимость награды от распределения $\pi(\cdot | s)$.

Утверждение 70: Метод простой итерации сходится к единственному решению мягких уравнений Беллмана из любого начального приближения.

Итак, мы уже можем сразу построить процедуру обучения критика. Рассмотрим обучение $Q_\omega(s, a) \approx Q_{\text{soft}}^\pi(s, a)$ с одношаговых оценок в off-policy режиме, то есть будем просто решать мягкое уравнение Беллмана (6.11). Тогда для заданного перехода $\mathbb{T} = (s, a, r, s')$ целевая переменная строится как

$$y(\mathbb{T}) := r + \gamma \mathbb{E}_{a' \sim \pi(a'|s')} [Q_\omega(s', a') - \log \pi(a' | s')]$$

В непрерывных пространствах действий взять матожидание по a' аналитически может не удастся (например, если π моделируется гауссианой). Можно, как всегда, тоже заменить его на Монте-Карло оценку (по возможности, взяв аналитически энтропию). Но при желании есть возможность ещё немножко сбить дисперсию, заведя вспомогательную нейросеть для аппроксимации $V_\psi(s) \approx V_{\text{soft}}^\pi(s)$: то есть, фактически, просто учить матожидание $\mathbb{E}_{a'}$.

В таком варианте таргеты для критика (Q-функция с параметрами ω) и для вспомогательной V-функции выглядят следующим образом: для заданного перехода $\mathbb{T} = (s, a, r, s')$, взятого из буфера, генерируем a_π из текущей версии стратегии π и запоминаем вероятность $\pi(a_\pi | s)$, после чего вычисляем несмешённые оценки правых частей уравнений связи (6.10) и (6.9):

$$\begin{aligned} y_Q(\mathbb{T}) &:= r + \gamma V_\psi(s') \\ y_V(\mathbb{T}) &:= Q_\omega(s, a_\pi) - \log \pi(a_\pi | s) \end{aligned}$$

И с такими таргетами, как обычно, решаем задачу регрессии с функцией потерь MSE. Самое главное — соответствующее действие a_π для таргета y_V брать не из буфера, а генерировать из текущей версии стратегии, поскольку иначе мы некорректно оценим среднее $\mathbb{E}_{a \sim \pi}$.

Естественно, в любых одношаговых таргетах оценка очень смешённая, и поэтому для стабилизации полезны две стандартные эвристики — таргет-сети и clipped double оценки (см. раздел 4.2.2). В реализациях места, где используются таргет-сети, могут различаться, поскольку однозначно сложно сказать, где их лучше всего использовать, но они точно должны быть; без таргет-сетей эти алгоритмы не заработают из-за сильной смешённости всех оценок.

6.2.3. Soft Policy Improvement

Теперь обсудим, как обучать актёра. Напомним, что вся теория оптимизации стратегии сводится к policy improvement и нам нужен аналог теоремы 17. Построить этот аналог можно из простых соображений: вот нам дана стратегия π_1 и её мягкая оценочная функция $Q_{\text{soft}}^{\pi_1}(s, a)$. Как можно улучшить эту стратегию из состояния s ? Сейчас π_1 набирает из s в среднем $V_{\text{soft}}^{\pi_1}(s)$. А сколько мы будем в среднем набирать, если сейчас в состоянии s сменим стратегию на π_2 (теперь с учётом энтропийного бонуса, который мы тогда получим в s за эту новую стратегию π_2), а в будущем будем вести себя ну по крайней мере не хуже, чем стратегия π_1 ?

Теорема 70 — Soft Policy Improvement: Пусть стратегии π_1 и π_2 таковы, что для всех состояний s выполняется:

$$\mathbb{E}_{\pi_2(a|s)} Q_{\text{soft}}^{\pi_1}(s, a) + \mathcal{H}(\pi_2(\cdot | s)) \geq V_{\text{soft}}^{\pi_1}(s),$$

тогда $\pi_2 \succeq \pi_1$ с учётом энтропийного бонуса; если хотя бы для одного s неравенство выполнено строго, то $\pi_2 \succ \pi_1$.

Доказательство. Полностью аналогично доказательству в обычном случае (теорема 17). Покажем, что $V_{\text{soft}}^{\pi_2}(s) \geq V_{\text{soft}}^{\pi_1}(s)$ для любого s :

$$\begin{aligned} V_{\text{soft}}^{\pi_1}(s) &\leq \{\text{по построению } \pi_2\} \leq \mathbb{E}_{\pi_2(a|s)} Q_{\text{soft}}^{\pi_1}(s, a) + \mathcal{H}(\pi_2(\cdot | s)) = \\ &= \{\text{связь QV (6.9)}\} = \mathbb{E}_{\pi_2(a|s)} [r + \mathcal{H}(\pi_2(\cdot | s)) + \gamma \mathbb{E}_{s'} V_{\text{soft}}^{\pi_1}(s')] \leq \\ &\leq \{\text{применяем это же неравенство рекурсивно}\} = \mathbb{E}_{\pi_2(a|s)} \left[r + \mathcal{H}(\pi_2(\cdot | s)) + \right. \\ &\quad \left. + \mathbb{E}_{s'} \mathbb{E}_{\pi_2(a'|s')} [\gamma r' + \gamma \mathcal{H}(\pi_2(\cdot | s')) + \gamma^2 \mathbb{E}_{s''} V_{\text{soft}}^{\pi_1}(s'')] \right] \leq \\ &\leq \{\text{раскручиваем цепочку далее}\} \leq \dots \leq \mathbb{E}_{\pi_2(a|s)} \sum_{t \geq 0} \gamma^t r_t + \gamma^t \mathcal{H}(\pi_2(\cdot | s_t)) = \\ &= \{\text{по определению мягкой V-функции}\} = V_{\text{soft}}^{\pi_2}(s) \end{aligned}$$

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, $V_{\text{soft}}^{\pi_2}(s) > V_{\text{soft}}^{\pi_1}(s)$. ■

Таким образом, аналог общей схемы Generalized Policy Iteration в задаче Maximum Entropy RL выглядит так:

- **Soft Policy Evaluation** заключается в обучении аппроксимации мягкой оценочной функции $Q_\omega \approx Q_{\text{soft}}^\pi$ для текущей стратегии π ;
- **Soft Policy Improvement** заключается в оптимизации следующего функционала (для разных состояний s):

$$\mathbb{E}_{\pi(a|s)} Q_\omega(s, a) + \mathcal{H}(\pi(\cdot | s)) \rightarrow \max_{\pi} \quad (6.13)$$

Соответственно, для обучения актёра, заданного параметрической стратегией $\pi_\theta(a | s)$, нужно просто оптимизировать не среднее значение Q-функции, а учесть дополнительно добавку энтропийного бонуса в рассматриваемом состоянии. У задачи оптимизации (6.13) есть один важный альтернативный вид:

Теорема 71: Задача (6.13) эквивалентна задаче

$$\text{KL}(\pi_\theta(\cdot | s) \| \exp Q_\omega(s, \cdot)) \rightarrow \min_{\theta},$$

где $\exp Q_\omega(s, \cdot)$ — ненормированное распределение над действиями.

Доказательство. Обозначим нормировочную константу распределения $\exp Q_\omega(s, \cdot)$ как $Z_\omega(s) := \int_A \exp Q_\omega(s, a) da$. Тогда:

$$\text{KL}(\pi_\theta(\cdot | s) \| \exp Q_\omega(s, \cdot)) = \mathbb{E}_{a \sim \pi_\theta(a|s)} \log \pi_\theta(a | s) - Q_\omega(s, a) + \log Z_\omega(s)$$

Осталось заметить, что при домножении на минус получим (6.13): первое слагаемое есть энтропия, а третье слагаемое не зависит от оптимизируемых параметров θ :

$$\mathbb{E}_{a \sim \pi_\theta(a|s)} \log Z_\omega(s) = \log Z_\omega(s) = \text{const}(\theta) \quad \blacksquare$$

Теорема 72 — Вид жадной стратегии: Максимальное значение (6.13) достигается на стратегии

$$\pi(a | s) \propto \exp Q_\omega(s, a) \quad (6.14)$$

Доказательство. Следует из теоремы 71, поскольку минимум KL-дивергенции достигается в нуле на совпадающих распределениях. \blacksquare

Таким образом мы показали, что актёр просто пытается выучить распределение $\pi_\theta(a | s) \propto \exp Q_\omega(s, \cdot)$ — большевскую стратегию по отношению к Q-функции (3.37). Мы сталкивались с ней, когда обсуждали способы исследования. Соответственно, в Maximum Entropy RL понятие жадной стратегии немножко другое: нужно не аргмакс по действиям брать, а софтмакс.

На практике второе слагаемое функционала (6.13) — энтропию $\pi_\theta(a | s)$ — обычно можно рассчитать аналитически для выбранной параметризации $\pi_\theta(a | s)$. Первое же слагаемое берётся так, как мы обсуждали в разделе 6.1.7: при помощи репараметризационного трюка, если такая возможность есть, и при помощи REINFORCE иначе.

Пример 96: Пусть наша стратегия параметризована гауссианой (см. пример 90). Для неё можно проводить репараметризационный трюк и также можно аналитически посчитать энтропию:

$$\mathcal{H}(\mathcal{N}(\mu, \sigma^2 I)) = \sum_{i=1}^A \log \sigma_i$$

Итого, формула (6.13) в таком случае получается следующей:

$$\mathbb{E}_{\varepsilon \sim \mathcal{N}(0, I)} Q_\omega(s, \mu_\theta(s) + \sigma_\theta(s) \odot \varepsilon) + \sum_{i=1}^A \log \sigma_i(s, \theta) \rightarrow \max_{\theta}$$

Пример 97: Пусть наша стратегия параметризована смесью гауссиан (см. пример 93). Тогда для неё не применим репараметризационный трюк, и сложно аналитически посчитать энтропию. Тогда придётся применять REINFORCE, и формула градиента (6.13) получается следующей:

$$\nabla_{\theta} = \mathbb{E}_{a \sim \pi_{\theta}(a|s)} \nabla_{\theta} \log \pi_{\theta}(a | s) [Q_{\omega}(s, a) - \log \pi_{\theta}(a | s) - b(s)],$$

где $b(s)$ — бэйзлайн, произвольная функция от состояний. Имеет смысл делать её близкой к среднему значению $Q_{\omega}(s, a) - \log \pi_{\theta}(a | s)$, то есть хорошо выбирать $b(s) := V_{\omega}(s) = \mathbb{E}_a [Q_{\omega}(s, a) - \log \pi_{\theta}(a | s)]$.

6.2.4. Soft Actor-Critic (SAC)

Мы уже можем собрать алгоритм-аналог DDPG для задачи Maximum Entropy RL. В этом алгоритме по аналогии с DDPG есть актёр и критик, критик учится оценивать актёра по одностадийным таргетам, а актёр обучается моделированием soft policy improvement-a. Таким образом, схема работает в off-policy режиме.

Перечислим ряд деталей, с которыми в целом можно играться в этом алгоритме⁴. При обучении критика используется вспомогательная V-функция, для стабилизации можно ограничиться таргет-сетью (с параметрами ψ^-) только для V-функции:

$$y_Q(\mathbb{T}) := r + \gamma V_{\psi^-}(s'),$$

а для борьбы с overestimation bias в таргете для V-функции используется twin оценка аналогично алгоритму TD3 (см. раздел 6.1.6), то есть обучается две Q-функции с параметрами ω_1, ω_2 , и таргет y_V строится по следующей формуле:

$$y_V(\mathbb{T}) := \min_{i=1,2} Q_{\omega_i}(s, a_\pi) - \log \pi(a | s),$$

где $a_\pi \sim \pi_\theta(a | s)$.

Наконец, в функционале (6.13) актёру предлагается «взламывать» минимум из двух Q-функций, то есть использовать twin-оценку и в этой формуле:

$$\mathbb{E}_{\pi_\theta(a | s)} \min_{i=1,2} Q_{\omega_i}(s, a) + \mathcal{H}(\pi(\cdot | s)) \rightarrow \max_{\pi}$$

Далее схема приведена для простоты для случая, если актёр моделирует гауссиану (см. пример 96). Конечно, возможно использование и для смеси гауссиан (см. пример 93) при замене в функционале актёра репараметризационного трюка на REINFORCE.

Алгоритм 24: Soft Actor-Critic (SAC)

Гиперпараметры: B — размер мини-батчей, β — параметр экспоненциального сглаживания таргет-сети, α — температура, $\pi_\theta(a | s) := \mathcal{N}(\mu_\theta(s), \sigma_\theta(s)^2 I)$ — гауссова стратегия с параметрами θ , $Q_{\omega_1}(s, a), Q_{\omega_2}(s, a)$ — две нейросетки с параметрами ω_1 и ω_2 , $V_\psi(s)$ — нейросетка с параметрами ψ , SGD-оптимизаторы.

Инициализировать $\theta, \omega_1, \omega_2, \psi$ произвольно

Инициализировать таргет-сеть $\psi^- := \psi$

Пронаблюдать s_0

На **очередном шаге t :**

1. выбрать $a_t \sim \pi_\theta(a_t | s_t)$
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 s из батча засэмплировать шума $\varepsilon(s) \sim \mathcal{N}(0, I)$ и посчитать $\mu(s, \theta), \sigma(s, \theta)$ стратегии π_θ
6. посчитать оценку градиента по параметрам стратегии:

$$\nabla_\theta := \frac{1}{B} \sum_{s \in B} \nabla_\theta \left[\alpha \sum_{i=1}^A \log \sigma_i(s, \theta) + \min_{i=1,2} Q_{\omega_i}(s, \mu_\theta(s) + \sigma_\theta(s) \odot \varepsilon(s)) \right]$$

7. делаем шаг градиентного подъёма по θ , используя ∇_θ
8. для каждого перехода $\mathbb{T} := (s, a, r, s', \text{done})$ засэмплировать $a_\pi \sim \pi_\theta(a_\pi | s)$ и сохранить вероятности $\pi_\theta(a_\pi | s)$
9. посчитать таргеты:

$$y_V(\mathbb{T}) := \min_{i=1,2} Q_{\omega_i}(s, a_\pi) - \alpha \log \pi_\theta(a_\pi | s)$$

$$y_Q(\mathbb{T}) := r + \gamma V_{\psi^-}(s')$$

⁴существует несколько версий этого алгоритма; в частности, есть версия, где Q-функции учатся через Q-функции, и вспомогательная V-сеть не используется. Тогда нужно заводить таргет-сети для двух Q-сетей.

10. посчитать лоссы:

$$\text{Loss}_V(\psi) := \frac{1}{B} \sum_{\mathbb{T}} (V_\psi(s') - y_V(\mathbb{T}))^2$$

$$\text{Loss}_{Q1}(\omega_1) := \frac{1}{B} \sum_{\mathbb{T}} (Q_{\omega_1}(s, a) - y_Q(\mathbb{T}))^2$$

$$\text{Loss}_{Q2}(\omega_2) := \frac{1}{B} \sum_{\mathbb{T}} (Q_{\omega_2}(s, a) - y_Q(\mathbb{T}))^2$$

11. делаем шаг градиентного спуска по ψ , ω_1 и ω_2 , используя $\nabla_\psi \text{Loss}_V(\psi)$, $\nabla_\omega \text{Loss}_{Q1}(\omega_1)$ и $\nabla_{\omega_2} \text{Loss}_{Q2}(\omega_2)$ соответственно

12. обновляем таргет-сеть: $\psi^- \leftarrow (1 - \beta)\psi^- + \beta\psi$

Можно провести sanity check, убедившись, что при $\alpha \rightarrow 0$, мы получаем обычный DDPG (с twin-трюком и для стохастической стратегии).



Консенсуса по поводу того, кто круче — TD3 (алгоритм 23) или SAC, нету. Считается, что они оба хорошо работают, по крайней мере лучше DDPG, и на разных задачах может победить как первый, так и второй. Недостатком TD3 является некоторая «костыльность» предложенных эвристик, а недостатком SAC — неудобный гиперпараметр температуры.

6.2.5. Аналоги других алгоритмов

В обычном RL оптимальной стратегией была жадная по отношению к оптимальной Q-функции, то есть та, для которой нельзя провести policy improvement ни в одном состоянии. Аналогичные рассуждения верны и для Maximum Entropy RL, разве что форма жадной стратегии (6.14) теперь другая.

Сформулируем критерий оптимальности Беллмана в задаче Maximum Entropy RL. По аналогии с обычным случаем, введём оптимальные оценочные функции:

$$V_{\text{soft}}^*(s) := \max_{\pi} V_{\text{soft}}^\pi(s)$$

$$Q_{\text{soft}}^*(s, a) := \max_{\pi} Q_{\text{soft}}^\pi(s, a)$$

Теорема 73 — Вид оптимальной стратегии для Maximum Entropy RL: Оптимальной является единственная стратегия

$$\pi(a | s) \propto \exp Q_{\text{soft}}^*(s, a)$$

Доказательство. Проводим рассуждения аналогичные теореме 12. Откажемся от стационарности и будем рассматривать задачу поиска оптимальной стратегии $\pi_t(a | s_0)$ в предположении, что в будущем мы сможем набрать максимально возможную награду $Q_{\text{soft}}^*(s, a, t)$:

$$\mathbb{E}_{\pi_t(a|s)} [Q_{\text{soft}}^*(s, a, t) - \log \pi_t(a | s)] \rightarrow \max_{\pi_t(a|s)}$$

Аналогично теореме 71 про soft policy improvement, можно заметить, что с точностью до константы, не зависящей от π_t , оптимизируемое выражение есть:

$$-\text{KL}\left(\pi_t(a | s) \parallel \frac{\exp Q_{\text{soft}}^*(s, a, t)}{Z(s, t)}\right) \rightarrow \max_{\pi_t(a|s)},$$

где $Z(s, t)$ — нормировочная константа $\exp Q_{\text{soft}}^*(s, a, t)$. Понятно, что оптимум достигается в нуле на $\pi_t(a | s)$, совпадающей с этим распределением.

Дальнейшее рассуждение строится как раньше: Q_{soft}^* от времени не зависит по определению (аналогично утв. 15), поэтому оптимальная стратегия получается стационарной, следовательно

$$\pi(a | s) \propto \exp Q_{\text{soft}}^*(s, a)$$

Заметим, что в силу однозначного определения $Q_{\text{soft}}^*(s, a)$, такая стратегия в принципе единственна в отличие от обычной задачи RL. ■

Имея на руках вид оптимальной стратегии, мы можем получить уравнения оптимальности:

Теорема 74:

$$V_{\text{soft}}^*(s) = \log \int_{\mathcal{A}} \exp Q_{\text{soft}}^*(s, a) da \quad (6.15)$$

Доказательство. Мы знаем, что оптимальная стратегия имеет вид $\pi^*(a | s) = \frac{\exp Q_{\text{soft}}^*(s, a)}{Z(s)}$, где

$$Z(s) := \int_{\mathcal{A}} \exp Q_{\text{soft}}^*(s, a) da$$

является нормировочной константой. Посчитаем энтропию такого распределения:

$$-\mathbb{E}_{\pi^*(a|s)} \log \pi^*(a | s) = \int_{\mathcal{A}} \frac{\exp Q_{\text{soft}}^*(s, a)}{Z(s)} (\log Z(s) - Q_{\text{soft}}^*(s, a)) da = \log Z(s) - \mathbb{E}_{\pi^*(a|s)} Q_{\text{soft}}^*(s, a)$$

Подставим оптимальную стратегию в мягкое VQ уравнение (6.10), которое справедливо в том числе и для оптимальной стратегии:

$$\begin{aligned} V_{\text{soft}}^*(s) &= \mathbb{E}_{\pi^*(a|s)} [Q_{\text{soft}}^*(s, a) - \log \pi^*(a | s)] = \\ &= \mathbb{E}_{\pi^*(a|s)} Q_{\text{soft}}^*(s, a) - \mathbb{E}_{\pi^*(a|s)} \log \pi^*(a | s) + \log Z(s) = \\ &= \log Z(s) \end{aligned}$$

Вспоминая определение $Z(s)$, получаем доказываемое. ■

Утверждение 71:

$$Q_{\text{soft}}^*(s, a) = r(s, a) + \gamma \mathbb{E}_{s'} V_{\text{soft}}^*(s')$$

Утверждение 72 — Мягкое уравнение оптимальности Беллмана (soft Bellman optimality equations):

$$Q_{\text{soft}}^*(s, a) = r(s, a) + \gamma \mathbb{E}_{s'} \log \int_{\mathcal{A}} \exp Q_{\text{soft}}^*(s', a') da' \quad (6.16)$$

Теорема 75: Оператор, стоящий в правой части уравнения (6.16), является сжимающим с коэффициентом γ , и, следовательно, метод простой итерации решения этой системы уравнений сходится из любого начального приближения к единственной неподвижной точке.

Доказательство. Пусть даны две Q-функции, Q_1, Q_2 , и пусть

$$\rho(Q_1, Q_2) := \max_{s, a} |Q_1(s, a) - Q_2(s, a)| < \varepsilon$$

Тогда:

$$\log \int_{\mathcal{A}} \exp Q_1(s, a) da \leq \log \int_{\mathcal{A}} \exp(Q_2(s, a) + \varepsilon) da = \varepsilon + \log \int_{\mathcal{A}} \exp Q_2(s, a) da$$

Аналогично можно показать, что

$$\log \int_{\mathcal{A}} \exp Q_1(s, a) da \geq -\varepsilon + \log \int_{\mathcal{A}} \exp Q_2(s, a) da$$

Пусть $\mathfrak{B}_{\text{soft}}$ — оператор, стоящий в правой части (6.16). Тогда:

$$\begin{aligned} |[\mathfrak{B}_{\text{soft}} Q_1](s, a) - [\mathfrak{B}_{\text{soft}} Q_2](s, a)| &= \gamma \mathbb{E}_{s'} \left(\log \int_{\mathcal{A}} \exp Q_1(s', a') da' - \log \int_{\mathcal{A}} \exp Q_2(s', a') da' \right) \leq \\ &\leq \gamma \varepsilon \end{aligned}$$

Таким образом, $\rho(\mathfrak{B}_{\text{soft}} Q_1, \mathfrak{B}_{\text{soft}} Q_2)$ уменьшилось по крайней мере в γ раз по сравнению с $\rho(Q_1, Q_2)$. ■

Утверждение 73: Если $\pi(a | s) \propto \exp Q_{\text{soft}}^\pi(s, a)$, то она оптимальна.

Доказательство. Q-функция такой стратегии удовлетворяет мягкому уравнению оптимальности Беллмана (6.16) и в силу единственности его решения совпадает с $Q_{\text{soft}}^*(s, a)$. ■

Теперь можно построить аналог DQN для задачи Maximum Entropy RL, называемым *Soft Q-learning*. В нём нет отдельной модели актёра, и текущая стратегия просто полагается жадной по отношению к текущему критику. Это также можно интерпретировать как моделирование *Soft Value Iteration* — решение мягкого уравнения оптимальности (6.16). Тогда таргет для перехода $\mathbb{T} := (s, a, r, s')$ строится как

$$y(\mathbb{T}) := r + \gamma \log \int_{\mathcal{A}} \exp Q_{\theta^-}(s', a') da',$$

где θ^- — параметры таргет-сети. Интересно, что это соответствует «учёту» Больцмановской стратегии исследования внутри оценочной функции (нечто похожее мы делали в табличном алгоритме SARSA в разделе 3.4.8). Однако, такой подход применим, как и в DQN, только для дискретных пространств действий, поскольку иначе стоящий в формуле интеграл мы аналитически не возьмём. Если мы попробуем завести отдельную нейросеть, чтобы приближать больцмановскую стратегию — получим SAC.

И, наконец, обсудим аналоги Policy Gradient алгоритмов для Maximum Entropy RL сеттинга. Многошаговые уравнения Беллмана выводятся тривиально — достаточно учесть в наградах энтропийные бонусы. Ну а аналог формулы policy gradient тоже можно легко угадать; мы же знаем, что если из формулы градиента удалить матожидание по частотам посещения состояний текущей стратегии, мы получаем формулу policy improvement-a. Это свойство в Maximum Entropy RL сохраняется, то есть можно просто к формуле soft policy improvement (6.13) добавить матожидание по частотам посещения состояний. Выпишем, что получится для метода REINFORCE:

Теорема 76 — Soft Policy Gradient:

$$\nabla_\theta J_{\text{soft}}(\theta) = \frac{1}{1 - \gamma} \mathbb{E}_{s \sim d_{\pi_\theta}(s)} \mathbb{E}_{a \sim \pi_\theta(a|s)} \nabla_\theta \log \pi_\theta(a | s) Q_{\text{soft}}^\pi(s, a) + \alpha \nabla_\theta \mathcal{H}(\pi_\theta(\cdot | s))$$

Доказательство. Аналогично доказательству в обычном случае. ■

Другими словами, сеттинг Maximum Entropy RL «объясняет» добавку энтропийного лосса при обучении актёра, но здесь важно помнить, что поменялось определение Q-функции: мягкая оценочная функция учитывает получение в будущем энтропийных бонусов (поэтому добавка энтропийного регуляризатора в обычных policy gradient алгоритмах остаётся эвристикой).

ГЛАВА 7

Model-based

В этой главе мы рассмотрим ситуацию, когда нам дана или мы готовы учить модель динамики среды.

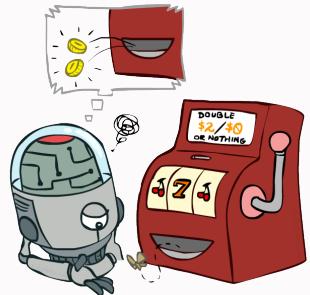
§7.1. Бандиты

7.1.1. Задача многоруких бандитов

Рассмотрим сильно упрощённую задачу RL, где эпизод заканчивается после первого шага.

Пример 98: В игровом зале стоят в ряд $|\mathcal{A}|$ игровых автоматов («одноруких бандитов»): в каждый можно отправить монетку, дёрнуть за ручку, и тогда автомат с некоторой вероятностью выдаст вам приз. Вероятность приза фиксирована для каждого автомата (зависит от настроек, выставленных владельцами зала...), но может различаться между автоматами.

Игроку (агенту) доступна только информация о том, получил ли он приз или нет, и всё, что он может — это пробовать дёргать за ручки разных автоматов. Задача — как можно быстрее выучить, какую ручку дёргать наиболее выгодно на основе накапливаемого опыта. Однако, без знания истинных вероятностей у игрока возникает вопрос: действительно ли тот автомат, который согласно его опыту выдавал приз чаще, является самым хорошим, или ему просто везло? Или с автоматом, который редко выдавал приз, просто было сыграно слишком мало игр? Налицо проблема exploration-exploitation дилеммы, которую в таком упрощённом виде легче теоретически анализировать.



Формально мы рассмотрим задачу RL для MDP, в котором нет состояний («есть только одно состояние»). Агент выбирает действие при помощи стратегии $\pi(a)$ и получает какой-то приз из автомата с выбранным номером, после чего эпизод заканчивается. Формально мы будем интерпретировать это как стохастическую функцию награды, которая зависит только от выбранного игроком действия.

Определение 83: *Многорукий бандит* (multi-armed bandit) — это \mathcal{A} распределений на вещественной оси $p(r | a)$, $a \in \{1, 2, 3 \dots A\}$.

Понятно, что в данной задаче Q-функция не зависит от стратегии и всегда равна

$$Q(a) = \mathbb{E}_{p(r|a)} r,$$

а оптимальная V-функция — скаляр, равный Q-функции оптимального действия, «наилучшего автомата»:

$$V^* = \max_a Q(a)$$

После каждого эпизода (т.е. после каждого шага) мы вольны улучшить свою стратегию, поэтому с каждым эпизодом наша стратегия меняется. Полный процесс, соответственно, задан следующим образом: на k -ом эпизоде мы сэмплируем $a_k \sim \pi_k(a)$ и наблюдаем награду $r_k \sim p(r | a_k)$, затем как-то меняем свою стратегию и получаем π_{k+1} .

Допустим, всего проводится T итераций обучения (играется T эпизодов). Если бы мы знали оптимальную ручку, мы бы сыграли все T эпизодов именно с ней, получив TV^* награды. Но нам придётся сколько-то итераций потратить на поиск этой самой ручки — на «исследования» — и за счёт этого мы проиграем за каждое неоптимальное действие a_k в среднем объёме $V^* - Q(a_k)$.

Определение 84: Сожалением (regret) за T эпизодов называется величина

$$\text{Regret}_T := TV^* - \sum_{k=0}^T Q(a_k)$$

Задача многорукого бандита — поиск такой процедуры обучения π , которая минимизировала бы наши средние сожаления. В первую очередь здесь интересны асимптотически оптимальные стратегии с точки зрения сожалений: что $\lim_{T \rightarrow \infty} \mathbb{E} \text{Regret}_T$, где математическое ожидание берётся по $\pi_1, \pi_2 \dots$, ведёт себя в некотором смысле наилучшим возможным образом.

7.1.2. Простое решение

Единственный способ оценивать $Q(a)$ — по Монте-Карло. На k -ом шаге мы можем оценить каждую Q -функцию как

$$Q_k(a) := \frac{\sum_k r_k[a_k = a]}{\sum_k [a_k = a]}$$

Как мы обсуждали в разделе про экспоненциальное слаживание, обновление такой Монте-Карло оценки $Q_k(a)$ через счётчики по сути и есть обучение: пусть $n_k(a)$ — счётчик, сколько раз мы выбирали действие a за все эпизоды, тогда:

$$\begin{aligned} Q_k(a_k) &= \left(1 - \frac{1}{n_k(a_k)}\right) Q_{k-1}(a_k) + \frac{1}{n_k(a_k)} r_k = \\ &= Q_{k-1}(a_k) + \frac{1}{n_k(a_k)} (r_k - Q_{k-1}(a_k)) \end{aligned}$$

Весь вопрос заключается в том, как на основе этих оценок и знания, сколько раз какое действие было попробовано — величины $n_k(a_k)$ — выбирать действия. Понятно, что жадный выбор может завести нас в ситуацию, когда мы будем всё время играть с не самой оптимальной ручкой.

Утверждение 74: При любом алгоритме скорость роста сожалений не более чем линейная: для некоторой константы C

$$\mathbb{E} \text{Regret}_T \leq CT$$

Доказательство. Рассмотрим худшую стратегию, которая всегда выбирает худшее действие с наибольшими сожалениями $\max_a (V^* - Q(a))$. Тогда сожаления за T эпизодов будут равны

$$\text{Regret}_T = T \max_a (V^* - Q(a))$$

Понятно, что это верхняя оценка на сожаления любого алгоритма, и $\max_a (V^* - Q(a))$ — константа C в линейной скорости роста. ■

Утверждение 75: При жадном выборе сожаления растут с линейной скоростью: для некоторой константы C

$$\mathbb{E} \text{Regret}_T \geq CT$$

Доказательство. С некоторой вероятностью α жадный алгоритм застрянет на постоянном выборе автомата a с ненулевым регретом $V^* - Q(a)$ и продолжит выбирать его до бесконечности. Тогда в среднем регрете появится слагаемое $T\alpha(V^* - Q(a))$, следовательно как минимум можно в качестве константы C выбрать $\alpha(V^* - Q(a))$. ■

Наивное решение — решать проблему эксплорейшна при помощи ε -жадной стратегии 3.36.

Алгоритм 25: Наивный бандит

Гиперпараметры: $\varepsilon(k)$ — стратегия исследования

Инициализировать $Q_0(a)$ произвольно

Обнулить счётчики $n_0(a) := 0$

На очередном шаге k :

1. выбрать a_k случайно с вероятностью $\varepsilon(k)$, иначе $a_k := \operatorname{argmax}_{a_k} Q_k(a_k)$
2. увеличить счётчик $n_k(a) := n_{k-1}(a) + [a_k = a]$
3. пронаблюдать r_k
4. обновить $Q_k(a) := Q_{k-1}(a) + [a_k = a] \frac{1}{n_k(a)} (r_k - Q_{k-1}(a))$

Утверждение 76: При ε -жадном выборе сожаления всё равно растут с линейной скоростью: для некоторой константы C

$$\mathbb{E} \text{Regret}_T \geq CT$$

Доказательство. Поскольку на каждом шаге с вероятностью $\frac{\varepsilon}{|\mathcal{A}|}$ мы выбираем некоторое неоптимальное действие a с ненулевым регретом $V^* - Q(a)$, в среднем регрете появится слагаемое $T \frac{\varepsilon}{|\mathcal{A}|} (V^* - Q(a))$, следовательно как минимум можно в качестве константы C выбрать $\frac{\varepsilon}{|\mathcal{A}|} (V^* - Q(a))$. ■



В случае нестационарных бандитов распределения $p(r | a)$ тоже меняются со временем и куда-то плывут. Наивное решение в такой ситуации можно модифицировать костыльми. Во-первых, ε нельзя уменьшать к нулю, необходимо постоянно пробовать различные действия, поэтому можно выставить ε в константу. Во-вторых, вместо счётчиков будем обновлять информацию о Q-функции через экспоненциальное слаживание:

$$Q_k(a_k) = Q_{k-1}(a_k) + \alpha (r_k - Q_{k-1}(a_k))$$

где $\alpha < 1$ — константный гиперпараметр.

7.1.3. Теорема Лай-Роббинса

Можно ли придумать что-то лучшее, чем линейная скорость роста регрета? Перепишем регрет чуть-чуть по-другому:

Утверждение 77:

$$\text{Regret}_T = \sum_a n_T(a) (V^* - Q(a)) \quad (7.1)$$

Понятно, что действия с большим регретом $V^* - Q(a_k)$ нужно выбирать как можно меньше, то есть асимптотически уменьшать счётчик $n_T(a)$. С другой стороны, ясно, что если распределения $p(r | a^*)$, где a^* — оптимальная ручка, и $p(r | a)$ для некоторого другого действия a очень похожи, то различить эти два автомата будет тяжело (просто потому, что если распределения похожи, то и сэмплы из них будут очень похожи). Также ясно, что если распределения похожи, то есть надежда, что у них будут очень похожи средние, то есть регрет для такого действия a будет маленьким. И наоборот: если регрет за действие большой, у автомата наверняка сильно другое распределение, нежели чем у оптимального автомата, и тогда их наверняка можно как-то просто различить по выборкам сэмплов. Оказывается, можно получить следующую нижнюю оценку на асимптотическое поведение любого алгоритма:

Теорема 77 — Теорема Лай-Роббинса (Lai-Robbins theorem):

$$\mathbb{E} \text{Regret}_T \geq \log T \sum_{a \neq a^*} \frac{V^* - Q(a)}{\text{KL}(p(r | a) \| p(r | a^*))}$$

Без доказательства. ■

Весьма сильное утверждение: оно говорит, что теоретически нельзя построить алгоритм с лучшим асимптотическим поведением регрета, чем $\log T$. Константа имеет понятный физический смысл: каждый автомат вносит свой вклад в эту константу, пропорциональный неоптимальности действия (числитель) и обратно пропорциональный похожести распределения награды в этом автомате с оптимальным автоматом (знаменатель).

Определение 85: Будем говорить, что алгоритм *асимптотически оптимальен*, если он имеет логарифмическую скорость роста среднего регрета.

7.1.4. Upper Confidence Bound (UCB)

Попробуем поискать хорошую эвристику исследования среди алгоритмов следующего вида: на очередном шаге k будем выбирать действие по следующей формуле:

$$a_k := \underset{a}{\operatorname{argmax}} [Q_k(a) + U_k(a)], \quad (7.2)$$

где $U_k(a)$ — некоторая положительная добавка, имеющая смысл **бонуса за исследование** (exploration bonus). То, что добавка должна быть положительна, следует из принципа **оптимизма перед неопределенностью** (optimism in the face of uncertainty).

Пример 99: Представьте, что вы идёте мимо пещеры, в которую вы никогда не заходили, и ваша оценка Q-функции для действия «зайти в пещеру» ниже, чем оценка других действий. Если алгоритм исследования таков, что ваше значение Q-функции занижается, то может возникнуть ситуация, что вы никогда не зайдёте в пещеру и не узнаете, что там. Если бы вы были уверены в идеальности ваших оценок, вы бы имели гарантии неоптимальности действия «зайти в пещеру»; но в реальности оценки никогда не бывают точны, и поэтому алгоритму исследований следует «заныть» те значения Q-функции, которые потенциально посчитаны ошибочно. При этом, чем больше неопределенность в их значениях, тем больше должно быть завышение.

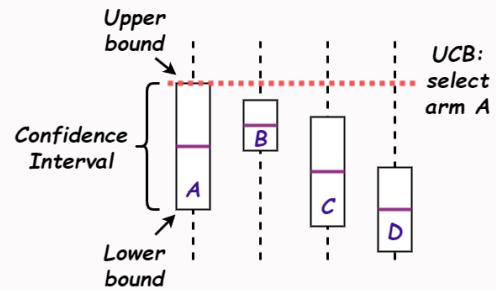
Строить добавку нужно из соображений, вытекающих из формы регрета (7.1). Добавка должна быть маленькая, если данное действие было выбрано уже много раз, и наша неопределенность в знаниях о среднем значении $Q(a)$ достаточно точные, или же если нам кажется, что регрет для этого действия близок к нулю.

Идея *upper confidence bounds* (UCB) алгоритмов следующая: давайте выбором $U_k(a)$ прогарантируем, что

$$Q(a) \leq Q_k(a) + U_k(a)$$

с очень высокой вероятностью, близкой к единице, то есть, другими словами, построим **доверительный интервал** (confidence interval) и возьмём его верхнюю границу. Такой $U_k(a)$ будет обратно пропорционален $n_k(a)$, ведь граница будет сжиматься к эмпирическому среднему. Жадный выбор $\underset{a}{\operatorname{argmax}} Q_k(a)$, интуитивно, будет выбираться часто; его счётчик будет увеличиваться, и exploration bonus для него будет уменьшаться; тогда с достаточно маленькой вероятностью мы выберем при помощи формулы (7.2) действительно неоптимальное действие, для которого добавка $U_k(a)$ в силу его редкого выбора большая, и эта вероятность будет тем меньше, чем меньше эмпирическое среднее для этого неоптимального действия.

Пример 100: На картинке справа изображены «свечки»: для каждого из четырёх автоматов указаны средние (оценки $Q_k(a)$), а также верхние и нижние границы доверительного интервала. Хотя автомат В кажется наиболее выгодным, мы уже достаточно часто играли с ним, поэтому его верхняя граница интервала доверия не так далека от среднего; а вот с автоматом А мы играли редко и поэтому на очередном шаге решим выбрать именно его.



Доверительный интервал можно построить при помощи следующего неравенства:

Теорема 78 — Неравенство Хёффнинга: Пусть $X_1 \dots X_n$ — i.i.d выборка из распределения на домене^a $[0, 1]$ с истинным средним μ , $\hat{\mu} := \frac{1}{N} \sum_i^N X_i$ — выборочная оценка среднего. Тогда $\forall u > 0$:

$$\mathbb{P}(\mu \geq \hat{\mu} + u) \leq e^{-2nu^2}$$

Без доказательства. ■

^aмы всегда предполагаем ограниченность наград; для удобства записи будем считать, что диапазон награды — $[0, 1]$.

Мы можем переформулировать эту теорему в терминах доверительного интервала:

Утверждение 78: Для любого δ с вероятностью $1 - \delta$ истинное значение $Q(a)$ не превосходит $Q_k(a) + U_k(a)$, где:

$$U_k(a) := \sqrt{\frac{-\ln \delta}{2n_k(a)}}$$

Доказательство. В силу неравенства Хёфдинга:

$$\mathbf{P}(Q(a) \geq Q_k(a) + U_k(a)) \leq \exp^{-2n_k(a)U_k(a)^2}$$

Возьмём заданное δ и прогарантируем, что $\exp^{-2n_k(a)U_k(a)^2} = \delta$. Разрешая это равенство относительно $U_k(a)$, получаем доказываемое. ■

В качестве δ на k -ом шаге будем выбирать, например, $\frac{1}{k^c}$, где c — гиперпараметр. Таким образом, истинное значение будет всё с большей вероятностью оказываться внутри доверительного интервала. Итого в алгоритме UCB предлагается следующая формула выбора действия на очередном k -ом шаге:

$$a_k = \operatorname{argmax}_a \left[Q_k(a) + c \sqrt{\frac{\log k}{n_k(a)}} \right] \quad (7.3)$$

Получается интерпретируемая формула: числитель $\sqrt{\log k}$ гарантирует, что если некоторое действие давно не выбиралось (счётчик $n_k(a)$ не меняется с увеличением числа итераций k), то добавленное слагаемое будет неограниченно расти и в какой-то момент промотивирует попробовать данное действие ещё раз.

Теорема 79: UCB-алгоритм (7.3) асимптотически оптимален.

Без доказательства. ■

7.1.5. Сэмплирование Томпсона

Мы далее попробуем учить степень нашей неопределённости в знаниях об $Q(a)$, моделируя всё неизвестное распределение $p(r | a)$, а не только среднюю награду. Таким образом, мы перейдём к model-based подходу в RL, в котором динамику среды предлагаются пытаться учить. В случае с бандитами под динамикой среды подразумеваются распределения награды для каждого автомата $p(r | a)$, для чего мы воспользуемся стандартным байесовским подходом.

Введём предположение, что распределения наград $p(r | a)$ принадлежат некоторому параметрическому семейству; для каждого автомата заданы значения параметров этого семейства $\theta_a \in \Theta$, и награда генерируется из распределения $p(r | \theta_a)$. Однако, истинные значения θ_a нам неизвестны.

Вместо оценки максимального правдоподобия на θ_a будем делать **байесовский вывод**. Зададимся некоторым **априорным распределением** («прайором») $p(\theta_a)$ для каждого действия a (сюда мы можем в том числе положить какую-то дополнительную априорную информацию о том, какие автоматы заведомо лучше), и после очередного эпизода игры с автоматом a с исходом r_k будем обновлять распределение над θ_a по формуле Байеса на **апостериорное распределение**:

$$p(\theta_a) \leftarrow p(\theta_a | r_k) \propto p(r_k | \theta_a)p(\theta_a)$$

Таким образом мы аккумулируем всю информацию о значениях θ_a , полученную из всех имеющихся сэмплов награды и исходного априорного распределения. Средний выигрыш из автомата a , соответственно равный $\mathbb{E}p(r | \theta_a)$, теперь для нас будет являться случайной величиной, поскольку θ_a — случайное, с распределением $p(\theta_a)$.

Пример 101 — Bernoulli-бандиты: Положим, что автоматы выдают только награду 0 или 1, то есть что истинное распределение есть распределение Бернулли с вероятностью θ_a :

$$p(r | a) := \text{Bernoulli}(r | \theta_a) = \theta_a^{\mathbb{I}[r=1]}(1 - \theta_a)^{\mathbb{I}[r=0]} = \theta_a^r(1 - \theta_a)^{1-r}$$

Априорное распределение зададим при помощи **Бета-распределения**¹ распределения, а то есть:

$$p(\theta_a) := \text{Beta}(\theta_a | \alpha, \beta) \propto \theta_a^{\alpha-1}(1 - \theta_a)^{\beta-1}$$

где α и β — некоторые параметры (свои для каждого автомата a). Мы можем обновлять эти знания при помощи новых сэмплов $r_k \sim p(r | a)$, проводя байесовский вывод. Применяем формулу Байеса:

$$p(\theta_a | r) \propto \underbrace{p(r_k | \theta_a)}_{p(r_k | \theta_a)} \underbrace{\theta_a^{r_k}(1 - \theta_a)^{1-r_k}}_{\theta_a^{\alpha-1}(1 - \theta_a)^{\beta-1}} = \text{Beta}(\theta_a | \alpha + r_k, \beta + 1 - r_k)$$

Таким образом, для обновления параметров α, β достаточно увеличить α на r_k , а β — на $1 - r_k$.

При заданном θ_a мы можем посчитать среднее значение выигрыша, ценность автомата a : для случайной Бернуллиевской величины с параметром θ_a среднее, собственно, совпадает с θ_a :

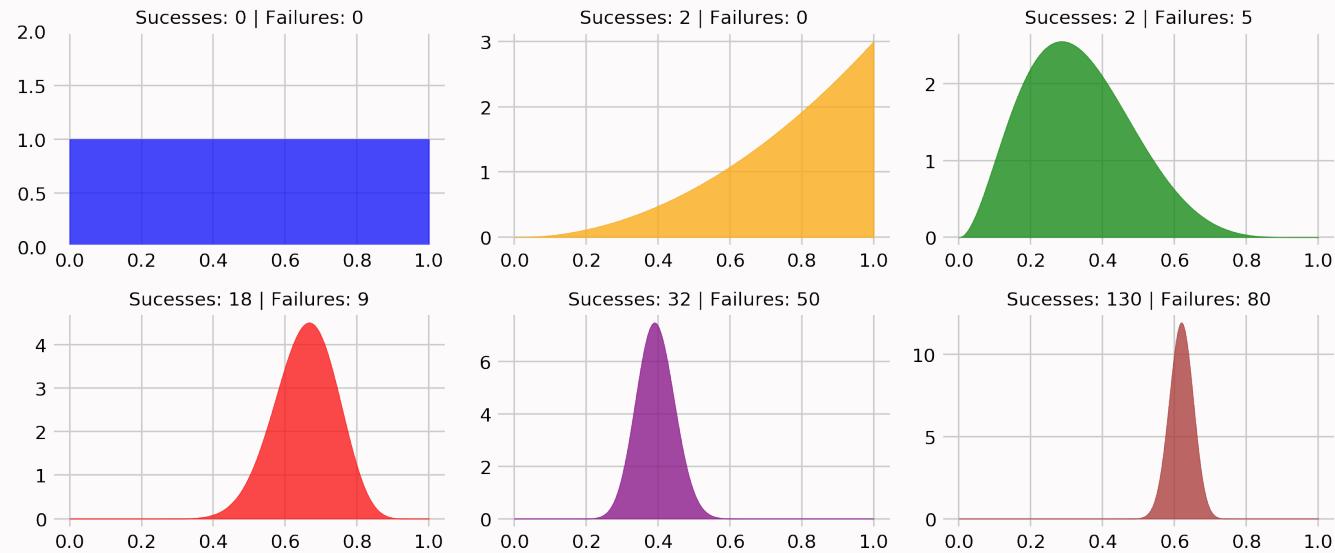
$$\hat{Q}(a) = \mathbb{E}p(r | \theta_a) = \theta_a$$

Тогда, имея α и β , мы всегда можем посчитать мат.ожидание выигрыша в нашей модели:

$$\mathbb{E}_{\theta_a} \hat{Q}(a) = \mathbb{E}_{\text{Beta}(\theta_a | \alpha, \beta)} \theta_a = \frac{\alpha}{\alpha + \beta}$$

Как видно, оно не отличается от Монте-Карло оценки Q-функции: α имеет смысл «успехов», когда Бернуллиевская величина выкинула нам единичку, а β — число неуспехов, нулей, и формула обновления этих параметров в точности соответствует подсчёту этих счётчиков; при выборе прайора $\alpha = \beta = 0$ мы получим в формуле мат.ожидания отношение числа успехов к общему числу попыток.

Однако теперь у нас есть не только оценка среднего значения θ_a , но и вероятности для каждого его возможного значения. На рисунке далее приведены $p(\theta_a)$ для разных значений α («успехов») и β («неудач») при прайоре $\alpha = \beta = 1$:



¹это распределение является **сопряжённым** (conjugate) к Бернули, что означает, что после применения формулы Байеса наше распределение останется в этом же семействе распределений.

Как пользоваться полученным апостериорным распределением? Мы можем реализовать идею, которая называется **probability matching**. Согласно текущим $p(\theta_a)$ жадный выбор действия имеет вид

$$a := \underset{a}{\operatorname{argmax}} \mathbb{E}_{\theta_a \sim p(\theta_a)} \mathbb{E} p(r | \theta_a),$$

но теперь в нашей модели есть некоторая вероятность и того, что такое действие на самом деле неоптимально. Мы можем посчитать эту вероятность, то есть с учётом распределений θ_a посчитать вероятность, что хотя бы для одного другого автомата $\hat{a} \neq a$

$$\mathbb{E} p(r | \theta_{\hat{a}}) > \mathbb{E} p(r | \theta_a),$$

и посмотреть, с какой вероятностью мы ошибёмся. Аналогично можно для любого действия посчитать вероятность того, что на самом деле оптимальным является оно. Предлагается ровно с такими вероятностями, собственно, и выбирать автомат на очередном шаге.

Определение 86: *Сэмплированием Томпсона* (Thompson Sampling) называется процедура, при котором на k -ом шаге при решении задачи многоруких бандитов действие a выбирается с вероятностью того, что оно оптимально в рамках выученных моделей $p(\theta_a)$:

$$\pi(a) := \mathbf{P} \left(\mathbb{E} p(r | \theta_a) = \max_{\hat{a}} \mathbb{E} p(r | \theta_{\hat{a}}) \right)$$

Посчитать такие вероятности может быть нетривиально, зато легко из такого распределения сэмплировать (отсюда название²). Действительно: просто для каждого действия a засэмплируем θ_a из текущего $p(\theta_a)$, и, соответственно, выберем автомат с наибольшим мат.ожиданием согласно выпавшим θ_a .

Пример 102: Допустим, мы храним информацию о распределениях $p(r | a)$ в Бета-распределениях, как в предыдущем примере. Для всех действий сэмплируем $\theta_a \sim \text{Beta}(\theta_a | \alpha_a, \beta_a)$ при выученных параметрах α_a, β_a , и считаем мат.ожидание соответствующего распределения Бернулли при выпавших параметрах θ_a —

²забавный факт: Томпсон вообще первый рассмотрел задачу многоруких бандитов, и в качестве эвристического решения предложил сэмплировать из моделей и брать аргмаксимум из сэмплов. То есть, можно сказать, сэмплирование Томпсона было первым придуманным способом решения задачи. А позже через почти век выяснилось, что это решение асимптотически оптимально.

для распределения Бернулли среднее совпадает с θ_a в точности. Таким образом, мы выберем автомат, для которого выпало наибольшее значение сэмпла θ_a .

Идея в том, что для автоматов, которые мы пробовали часто, дисперсия апостериорного распределения будет маленькой и сэмпл почти всегда будет в точности равен текущей оценке Q-функции. Для автоматов, которые мы пробовали редко в силу маленького реварда, дисперсия будет большая и иногда сэмплы будут оказываться лучше текущего самого хорошего автомата, что заставит нас поэксплорить данный автомат ещё.

Алгоритм 26: Beta-Bernoulli Бандит с сэмплированием Томпсона

Гиперпараметры: α, β — параметры прайора.

Обнулить счётчики $\alpha_0(a) := \alpha, \beta_0(a) := \beta$

На очередном шаге k :

1. сгенерировать $\theta_a \sim \text{Beta}(\alpha_k(a), \beta_k(a))$
2. выбрать $a_k := \underset{a}{\operatorname{argmax}} \theta_a$
3. пронаблюдать $r_k \in \{0, 1\}$
4. обновить счётчик $\alpha_k(a) := \alpha_{k-1}(a) + [a_k = a][r_k = 1]$
5. обновить счётчик $\beta_k(a) := \beta_{k-1}(a) + [a_k = a][r_k = 0]$

Теорема 80: Сэмплирование Томпсона асимптотически оптимально.

Без доказательства. ■

Пример 103: Поиграться с визуализацией этого алгоритма можно [здесь](#).

7.1.6. Обобщение на табличные MDP

Теория бандитов подсказывает, как можно «правильно» разрешать exploration-exploitation дилемму. К сожалению, эти идеи плохо масштабируются на произвольные MDP. Подход на основе UCB-счётчиков (7.3) можно эвристически моделировать в сложных MDP, что мы обсудим позже в главе 8.2. Сэмплирование Томпсона же может оказаться удобным способом справиться с некоторыми практическими задачами табличного RL.

Итак, пусть задано табличное MDP с конечным числом состояний и действий. Будем в байесовском смысле учитывать функцию переходов и функцию награду, то есть моделировать вероятности $p(r | \theta_{s,a}) \approx p(r | s, a)$ и $p(s' | \phi_{s,a}) \approx p(s' | s, a)$ и обновлять θ и ϕ по правилу Байеса. Для этого для каждой пары s, a мы будем хранить «текущее» распределение $p(\theta_{s,a})$ и $p(\phi_{s,a})$, обусловленные на всю встретившуюся историю. Фактически это будут вероятности, с которыми истинное MDP живёт по тем или иным законам. После очередного перехода (s, a, r, s') будем обновлять распределения над параметрами по формулам

$$p(\theta_{s,a}) \leftarrow p(\theta_{s,a} | r) \propto p(r | \theta_{s,a})p(\theta_{s,a})$$

$$p(\phi_{s,a}) \leftarrow p(\phi_{s,a} | s') \propto p(s' | \phi_{s,a})p(\phi_{s,a})$$

Далее на очередном шаге мы будем из такого «распределения в пространстве MDP» сэмплировать MDP: $\forall s, a$:

$$\theta_{s,a} \sim p(\theta_{s,a}), \quad \phi_{s,a} \sim p(\phi_{s,a})$$

Конкретные значения θ и ϕ теперь задают какое-то одно конкретное MDP, в котором мы можем любым алгоритмом динамического программирования (например, Value Iteration) найти оптимальную стратегию $\pi^*(a | s, \theta, \phi)$. Ей и будем пользоваться, чтобы сыграть, например, очередной эпизод игры и собирать новые данные для уточнения модели функции переходов и функции наград.

Такая процедура называется *posterior sampling*, и является аналогом сэмплирования Томпсона для табличных MDP: мы будем выбирать действие с той вероятностью, с которой оно является оптимальным, обусловленной на всю историю нашего взаимодействия с настоящим MDP. Понятнее идея может стать на следующем примере.

Пример 104 — Online Shortest Path: Каждый день с утра агент едет на машине на работу. Карта представляет собой граф из нескольких вершин, некоторые из которых соединены ребрами. Агент хочет кратчайшим маршрутом добираться из вершины «дом» в вершину «работа». Такая задача решалась бы алгоритмом поиска

кратчайшего пути в графе, если бы агенту были известны точные значения, сколько времени нужно на проезд по каждому ребру. Но это какие-то случайные величины, и каждый раз, когда агент проезжает по тому или иному ребру e , он тратит случайное время $r \sim p(r | e)$. Как ездить на работу так, чтобы постепенно, с течением процесса суммарное время («регрет»), которое агент тратил на поездки, было всё меньше и меньше?

В этой задаче вершины можно считать состояниями s , а рёбра можно считать парами s, a . Агент, допустим, знает карту, то есть знает, что «функция переходов» — детерминированная, и знает $p(s' | s, a)$, но не знает распределение наград. Поэтому в этой задаче достаточно учить лишь второе.

Итак, что говорит сэмплирование Томпсона: а давайте мы для каждого ребра e заведём модель $p(r | \theta_e)$, и будем в байесовском смысле учить θ_e . Здесь r — вещественное, поэтому понадобится проводить байесовский вывод для непрерывных распределений*. Допустим, мы выбрали какое-то семейство $p(r | \theta_e)$, завели какой-то прайор $p(\theta_e)$ и научились обновлять апостериорное распределение по формуле Байеса.

Тогда перед очередным эпизодом мы сэмплируем $\theta_e \sim p(\theta_e)$, и теперь у нас есть конкретное MDP с конкретной моделью наград. Мы можем посчитать среднее $E_p(r | \theta_e)$ и считать, что в среднем ровно столько времени потребуется для проезда по ребру e . Затем мы в графе можем найти кратчайший маршрут (поиск кратчайшего маршрута в общем-то является частным случаем общего алгоритма динамического программирования Value Iteration), и алгоритм предлагает отправиться именно по нему. Собранные награды за шаг — затраченное время для проезда по каждому ребру — используются для уточнения распределений $p(\theta_e)$, и модель всё улучшается.

*например, если моделировать $p(r | \theta_e)$ в виде нормального распределения с настраиваемыми средним и дисперсией, $p(r | \theta_e) := \mathcal{N}(\mu_e, \sigma_e^2)$, то тогда в качестве сопряжённого распределения, как подсказывает [википедия](#), понадобится выбрать [гамма-нормальное распределение](#). Формулы вывода тогда можно посмотреть, например, [здесь](#).

§7.2. Обучаемые модели окружения

7.2.1. Планирование

Понятно, что model-free алгоритмы, рассмотренные ранее, действуют не совсем так, как размышляет человек. Принимая решения, мы принимаем в учёт наши предсказания о том, в какое будущее выльется то или иное действие, и для построения предсказаний используем наши представления о том, как работает окружающий нас мир.

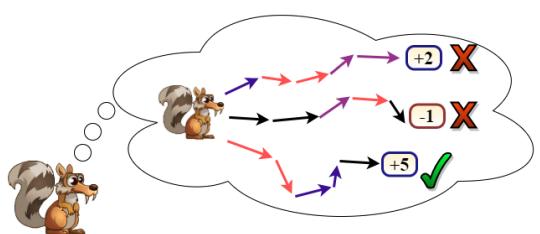
Итак, вернёмся к общей постановке задачи RL, и сейчас допустим, что нам доступен симулятор $p(s', r | s, a)$ для произвольных s, a в произвольный момент времени.

В табличном сеттинге знание $p(s', r | s, a)$ позволяет применять алгоритмы динамического программирования Value Iteration и Policy Iteration. Однако, в табличном сеттинге мы умеем считать все интегралы — мат. ожидания по функции переходах; а в сложных средах знание функции переходов эти интегралы брать не позволяют.

Понятное дело, что, имея доступ к точному симулятору, мы можем, например, запустить какой-нибудь model-free алгоритм для обучения π , но нас интересует, можем ли мы непосредственно «воспользоваться» симулятором внутри самой стратегии, или даже придумать стратегию, использующую только симулятор. Раз мы полностью знаем правила игры, то, выбирая действие в некотором состоянии s_0 , мы можем играть «виртуальные» игры, заглядывая в своё будущее.

Определение 87: Для данного состояния s_0 набор действий a_0, a_1, a_2, \dots будем называть *планом* (plan).

Итак, допустим мы сидим в некотором s_0 и хотим найти хорошее действие a_0 . Мы можем выбрать какое-нибудь a_0 , засэмплировать s_1 , выбрать как-то a_1 , и так построить какой-нибудь план, для которого у нас будет сэмпл будущей награды. Можем так сделать, скажем, много раз. Нас ждёт две проблемы: у нас не получится рассмотреть всевозможные варианты будущего, и мы не сможем посчитать интеграл средней награды для одного плана; мы можем получить лишь сэмпл или Монте-Карло оценку по нескольким сэмплам.



Пример 105: Какой типичный способ построения ИИ для ботов в играх: для текущего состояния s , в котором нужно выбрать действие, рассматривается несколько вариантов дальнейших действий на ближайшие шаги сто (при этом желательно заранее неоптимальные варианты отсеять, чтобы сократить перебор); для каждого плана проводится симуляция (или несколько, если правила игры стохастичны и время позволяет). Играть до конца эпизода обычно дорого, поэтому состояние \hat{s} , на котором симуляцию игры прервали, придётся оценивать при помощи какой-то эвристической оценки $V^*(\hat{s})$.

Подобные алгоритмы могут «спланировать» свои будущие действия, а не выучить непосредственно зависимость $\pi(a | s)$ («обучить стратегию»).

Определение 88: Задача

$$\underset{a_0, a_1, a_2 \dots}{\operatorname{argmax}} \mathbb{E}_{\mathcal{T} | s_0, a_0, a_1, a_2 \dots} R(\mathcal{T}) \quad (7.4)$$

при доступной функции переходов $p(s' | s, a)$ и функции награды называется *планированием* (planning).

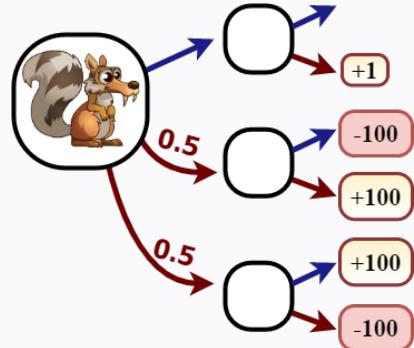
В такой концепции понятия обучения нет. Для детерминированных MDP задача планирования по определению даёт оптимальный план, если задача (7.4) решена точно, но для стохастичных MDP это неверно:

Теорема 81 — Неоптимальность планирования: Пусть MDP стохастично и $a_0, a_1, a_2 \dots$ — точное решение задачи (7.4) для s_0 . Тогда может быть такое, что действие a_0 неоптимально ($Q^*(s_0, a_0) < Q^*(s_0, a)$ для некоторого $a \neq a_0$).

Доказательство. Посмотрим, чему равна средняя награда для каждого плана для MDP с рисунка.

План	Ценность плана
(■, ■)	+1
(■, □)	0
(□, ■)	$\frac{1}{2}(+100 - 100) = 0$
(□, □)	$\frac{1}{2}(+100 - 100) = 0$

Значит, будет выбран «безопасный» вариант в виде плана (■, ■). Конечно же, оптимально выбрать в начале действие ■ и далее в зависимости от состояния выбрать то действие, которое даст +100, таким образом на самом деле $V^*(s_0) = +100$. ■



Другими словами, планирование — разумный подход в детерминированных средах, когда будущее предопределено. Поэтому большинство планировщиков, которые мы будем обсуждать далее, заточены именно под детерминированный случай. Важно, что даже в такой ситуации решать задачу (7.4) точно, конечно же, не получится, и наше решение будет лишь приближённым.

По этим двум причинам планировщик иметь смысл перезапускать на каждом шаге. Обычно алгоритм решения задачи планирования используется как просто «очень вычислительно дорогая» стратегия: сидя в s_0 , алгоритм планирует совершить действия $a_0, a_1, a_2 \dots$; в реальной среде совершается лишь действие a_0 , после чего для $s_1 \sim p(s_1 | s_0, a_0)$ алгоритм планирования запускается ещё раз (важно, что тут он может использовать какую-то информацию с прошлого процесса планирования) для определения a_1 и так далее. Такая процедура называется *планированием с обратной связью*. Планировщик, в частности, может понимать, что он не рассмотрел всевозможные исходы будущего, и из этих соображений выдать вероятностное распределение $\pi(a_0 | s_0)$; тогда стратегия, полученная при помощи такого планировщика, будет стохастична.

Пример 106 — Мета-эвристики как планировщики: Для решения (7.4) можно использовать, например, случайный поиск. Сидя в s_0 , засэмплируем 100 случайных планов и при помощи симулятора посчитаем для каждого из них сэмплы $R(\mathcal{T})$; выберем то a_0 , для которого нашлась траектория с максимальным сэмплом reward-to-go. Если среда детерминирована, то a_0 гарантировано позволяет получить такую награду; что не означает, что мы нашли наилучший план, и оптимальным может оказаться другое действие. Если среда стохастична, то даже гарантий получить «найденный» reward-to-go нет, поскольку это лишь сэмпл и нам могло в симуляции просто повезти. Выбранное a_0 отправляется в среду; для сэмпла s_1 из реальной среды процесс планирования запускается заново.

То есть, обучения в такой концепции нет, но каждое принятие решения алгоритмом (каждый «запуск стратегии») — это целый процесс оптимизации, что обычно очень дорого.

7.2.2. Модели мира (World Models)

В реальной жизни в рамках нашей общей постановки задачи функция переходов и награды $p(s', r | s, a)$ нам неизвестна. Тем не менее, у нас есть опция как-то попытаться её выучить и как-то использовать для улучшения наших алгоритмов. Открываются два вопроса: как учить и как использовать.

Определение 89: *Моделью мира* (world model) называется любая модель, явно или неявно обучающаяся модели динамики среды.

Под «явно или неявно» подразумевается, что это не обязательно должна быть функция, по состоянию и действию возвращающая следующее состояние, хотя, конечно, это самая стандартная опция.



Модель мира также может сжимать описание состояний в некоторое латентное пространство, которое далее может использовать стратегия. Ещё можно нарушить end-to-end парадигму и в целом отдельно выучить автокодировщик, который сожмёт описание состояний в некоторый компактный векторочек; RL

же применять только для обучения самой стратегии, которая будет принимать на вход такое компактное описание состояний. Пример подобного сжатия.

Простейший способ «использовать» выученную модель — взять в качестве стратегии какой-нибудь планировщик, который будет работать с обученной аппроксимацией модели мира будто это истинная модель. Нюанс такого подхода заключается в том, что планировщик по определению «хорош» только если модель мира точна. Если модель мира обучается только на тех примерах, которые мы встречаем в нашем опыте, то мы рискуем познать только тот мир, который обозревает наша текущая стратегия. До «генеральной совокупности» подобному опыту может быть далеко, и поэтому необходимо постоянно дообучать модель мира после каждого улучшения стратегии.

Алгоритм 27: Общая схема Model-based подхода

Гиперпараметры: Планировщик, модель мира.

Проинициализировать стратегию $\pi_0(a | s)$ случайно.

Проинициализировать модель мира случайно.

Проинициализировать датасет пустым множеством.

На k -ом шаге:

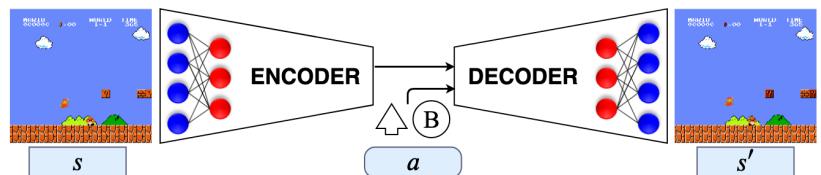
- Повзаимодействовать со средой при помощи π_k , добавив встреченные переходы $\{s, a, r, s'\}$ в датасет.
- Провести дообучение модели мира на собранном датасете:
- Получить π_k из планировщика, используя текущую модель мира.

Под словами «получить π_k из планировщика» здесь понимается, что далее в качестве стратегии π_k используется запуск планировщика, который кушает текущее состояние, рассматривает какие-то планы и выдаёт первое действие наилучшего плана a_0 или какое-то распределение в пространстве действий, «вероятности того, что a_0 соответствует оптимальному плану».

7.2.3. Модель прямой динамики

Определение 90: Модель функции переходов $p_\theta(s' | s, a)$ называется *моделью прямой динамики* (forward dynamics model).

Учить генеративную модель может быть дорого, и простым удешевлением является обучение детерминированного приближения $s' \approx f_\theta(s, a)$. Это можно делать по любым доступным траекториям, собранным любым способом:



$$\sum_{s,a,s'} \|f_\theta(s, a) - s'\|^2 \rightarrow \min_{\theta}$$

$$\sum_{s,a,r} (r_\psi(s, a) - r)^2 \rightarrow \min_{\psi}$$

Этот лобовой подход многим плох. В сложных средах описание состояний обычно содержит огромное количество никак не связанной с задачами агента информацией; например, декоративные элементы в видеоиграх. Обучение подобной f_θ сопряжено с бессмысленным изучением этой информации. Наконец, если состояния s — картинки, то в текущей формуле сгенерированное изображение сравнивается с реальным s' по l2-метрике; для изображений это не сильно осмысленно, конечно, и нужно придумывать что-то другое.



Иногда в задачах робототехники можно в модель прямой динамики внести «inductive bias», используя знания о том, что модель динамики представляет собой какое-то дифференциальное уравнение. Тогда могут искаститься параметры функций, стоящих внутри уравнений связи, а в качестве итоговой модели использовать какая-то дискретизация «обученного» диффура.

7.2.4. Сновидения

Наличие модели прямой динамики $p_\theta(s', r | s, a)$ позволяет нам при помощи текущей стратегии π генерировать траектории, используя полностью наши «внутренние модели» и никак не используя реальную внешнюю среду. Это означает, что в случае обучаемой модели динамики мы теоретически можем начать симулировать опыт при помощи, условно, нашей генеративной модели, и обучать на нём model-free алгоритм.

Определение 91: *Сновидениями* (dreaming) называется обучение агента на опыте, собранном при помощи приближения динамики среды $p_\theta(s', r | s, a)$.

Далеко мы так, конечно, не уедем, поскольку наша генеративная модель вряд ли будет идеальна: регулярно нужно «просыпаться» и отправляться в реальную среду собирать сэмплы для улучшения как минимум модели мира. Тем не менее, использование снов может существенно повысить sample efficiency алгоритмов: потребуется меньше реальных взаимодействий со средой, и большую часть процесса обучения самой стратегии можно переложить на сны. Цена — в вычислительной неэффективности: траекторий во сне из-за отличий от реальных потребуется генерировать довольно много. То есть, хотя шагов взаимодействия с настоящей средой станет меньше, время обучения алгоритма (wall-clock time) наоборот увеличится.

§7.3. Планирование для дискретного управления

7.3.1. Monte-Carlo Tree Search (MCTS)

Попробуем придумать алгоритм планирования (7.4) в предположении идеального симулятора для MDP с дискретным пространством действий.

Понятно, что для детерминированных MDP с функцией переходов $s' = f(s, a)$, чисто теоретически, мы можем просто построить полное дерево игры. Заведём дерево, в котором каждому узлу будет соответствовать состояние, дугам — действия, и скажем, что узел, соответствующий s' — потомок s по дуге a , если $s' = f(s, a)$.

У нас есть две проблемы. Во-первых, в сложных MDP такое дерево экспоненциально большое и целиком мы его не построим. Во-вторых, наши MDP, вообще говоря, стохастичны. Мы могли бы ввести много рёбер из s , соответствующих одному и тому же действию a и «подписать» над каждым вероятности перехода $p(s' | s, a)$, однако наша функция переходов $p(s' | s, a)$ может быть произвольным распределением, да и вообще пространства состояний \mathcal{S} могут быть в том числе континуальные или очень большие. Поэтому давайте введём понятие дерева чуть-чуть по-другому³:

Определение 92: Деревом MDP с корнем в состоянии s_0 будем называть дерево, где каждой дуге соответствует действие a ; узел на t -ом уровне дереве соответствуют плану $a_0, a_1 \dots a_{t-1}$, соответствующему пути от корня.

Нам нужно продумать эвристики, как сокращать перебор поиска по такому дереву, и как его строить в ситуациях, когда эта задача экспоненциально сложная и у нас ограничены вычислительные ресурсы. Мы сейчас рассмотрим некоторую очень общую схему, как можно такую эвристику строить.

Будем строить дерево MDP с корнем в s_0 постепенно. Изначально, на первой итерации, наше дерево пусто — есть только корень. Узлы дерева будем обозначать красивым символом \aleph . Дуги тогда однозначно задаются парой (\aleph, a) , где \aleph — узел дерева, откуда исходит дуга, a — действие, которому дуга соответствует. Для каждой дуги (\aleph, a) мы будем также хранить некоторую вспомогательную информацию; самый типичный вариант — это счётчики прохождения по данной дуге $n(\aleph, a)$ и ценность⁴ $Q(\aleph, a)$, скаляр.

Один шаг процедуры *Monte-Carlo Tree Search* (MCTS) заключается в проведении четырёх этапов: Selection, Expansion, Simulation и Update. Введём их по порядку.

Определение 93: На шаге *Selection* стартуем в корне, которому соответствует текущее состояние в реальной игре s_0 . При помощи некоторой стратегии, которую будем называть *TreePolicy*, и которая использует данные на дугах, выбираем действие a_0 ; спускаемся по дереву на уровень глубже по дуге, соответствующей a_0 , и сэмплируем s_1, r_1 из $p(s_1, r_1 | s_0, a_0)$. Повторяем процедуру до тех пор, пока не попадём в некоторый лист дерева на уровне t . Мы запоминаем (знаем) всю траекторию от корня до листа, то есть фактически выбираем таким образом начало некоторого плана $a_0, a_1 \dots a_{t-1}$ для рассмотрения, и заодно по Монте-Карло генерируем начало возможной траектории, соответствующей этому плану; в результате этой процедуры, мы попадаем в некоторый лист нашего текущего дерева и заодно симулируем для него состояние s_t .

³распространены объяснения MCTS в контексте детерминированных MDP, и поэтому про узлы постоянно говорят как про состояния; однако на самом деле ход рассуждений в принципе обобщается на стохастические MDP с тем замечанием, что в силу утверждения 81 в стохастических средах планирование (7.4) неоптимально.

⁴хотя принято обозначать ценности через V или Q , это не ценности каких-то состояний: в контексте стохастических MDP это ценность плана, который привёл нас в это состояние; если угодно, можно обозначить этот счётчик как $Q(s_0, a_0, a_1, a_2 \dots a_t) = \mathbb{E}_{\mathcal{T}|s_0, a_0, a_1 \dots a_t} R(\mathcal{T})$. Но в детерминированных средах они, конечно же, будут соответствовать оценочным функциям в стандартном понимании.

На этом шаге задачей TreePolicy является выбор того плана, для которого мы будем дальше строить дерево. То есть, допустим, мы сидим в некотором узле \mathbf{x} , не являющемся листом, из которого исходит $|\mathcal{A}|$ дуг, и для каждого возможного пути знаем приблизительную оценку качества $Q(\mathbf{x}, a)$ и то, сколько раз мы уже пробовали ходить по данной ветке — $n(\mathbf{x}, a)$. Задача этой части эвристики — найти самый перспективный путь в дереве, используя эти статистики. С одной стороны, нужно искать оптимальный план в той ветке, где оценка качества велика; однако, если по каким-то веткам мы ходили редко, то высока вероятность, что там нам не повезло с сэмплом награды или мы просто недостаточно раскрутили там дерево. Налицо проблема многорукого бандита (см. раздел 7.1); поэтому типичные TreePolicy вдохновлены⁵ решениями этой задачи.

Пример 107 — Пример TreePolicy: Воспользуемся UCB-бандитом (7.3), формула которого принимает следующий вид:

$$a := \underset{a}{\operatorname{argmax}} \left[Q(\mathbf{x}, a) + C \sqrt{\frac{\log n(\mathbf{x})}{n(\mathbf{x}, a)}} \right]$$

где $n(\mathbf{x}) := \sum_a n(\mathbf{x}, a)$ — это счётчик посещений узла, C — гиперпараметр. Действительно: $n(\mathbf{x})$ — сколько «эпизодов игры» мы провели с этим многоруким бандитом, а $n(\mathbf{x}, a)$ — сколько раз выбирали в этом месте действие a .

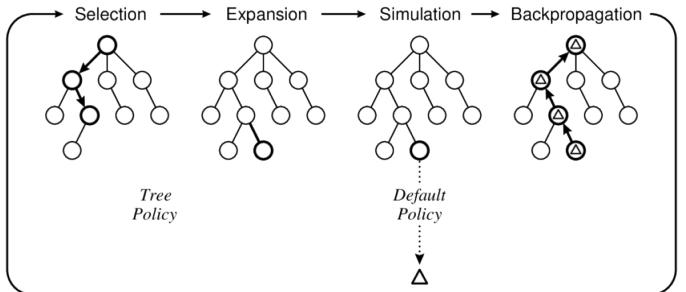
Определение 94: На шаге *Expansion* в выбранном на предыдущем этапе листе создаём для каждого действия $a_t \in \mathcal{A}$ по новому листу, соответствующему выбору этого действия: таким образом, мы расширяем дерево вдоль выбранной ветки «на один шаг вперёд».



Если число действий $|\mathcal{A}|$ велико, то имеет смысл определять поднабор действий случайным образом и создавать листы только для него. Тогда этап Selection заканчивается как только, например, TreePolicy сэмплирует действие, для которого ещё не существует дуги.

Определение 95: Шаг *Simulation* или *Evaluation* заключается в построении некоторой эвристической оценки reward-to-go для каждого нового построенного на предыдущем шаге листа.

В этом месте, конечно, можно использовать какие-нибудь handcrafted-эвристики, оценивающие очень примерно reward-to-go до конца эпизода (поэтому у этого шага есть второе название Evaluation), но универсальный вариант получить её — это для каждого a_t просимулировать игру (или несколько) из s_t, a_t до конца эпизода, выбирая действия при помощи некоторой другой стратегии, которую назовём *DefaultPolicy*, и которая уже не может использовать никакой информации в узлах дерева (эти узлы мы ещё просто-напросто не построили). Итого, мы получаем сэмплы reward-to-go для $|\mathcal{A}|$ планов, которые начинаются с $a_0, a_1 \dots a_{t-1}$, на t -ом месте действие варьируется, и для каждого варианта у нас есть одна (или несколько) новая Монте-Карло оценка награды.



Пример 108: Пример типичной DefaultPolicy — это банально случайная стратегия.

Определение 96: Шаг *Update* или *Backpropagation*: обновление счётчиков и оценок во всех дугах дерева, по которым мы проходили на данном шаге при помощи полученных на шаге симуляции новых Монте-Карло оценок.

Пример 109: Процесс обновления счётчиков прост: в свежераскрытом листе для новых дуг инициализируем $n(\mathbf{x}, a) := 1, Q(\mathbf{x}, a) := \hat{V}_a$, где \hat{V}_a — reward-to-go, полученный на шаге Simulation для действия a . Далее, рассмотрим пару d, a где-то в рассмотренной ветке дерева; во-первых, увеличиваем счётчик посещения этой дуги на единицу. Для обновления оценки Q просто учём новый сэмпл Монте-Карло оценки: мы симулировали награды за шаг, пока шли по дереву, и поэтому можем посчитать reward-to-go до момента прихода в лист. В листе же можно считать, что мы выбирали действие случайным образом, и поэтому усредним все \hat{V}_a по

⁵тем не менее, применение бандитов здесь — тоже эвристика, хотя бы потому, что из-за постепенного раскрытия дерева вдоль каждой ветки «распределение $p(r | a)$ », из которого приходит дальнейший reward-to-go, меняется.

действиям. Таким образом, если \hat{V} — полученный новый сэмпл reward-to-go, $Q(\mathbf{x}, a)$ обновляется как:

$$Q(\mathbf{x}, a) \leftarrow Q(\mathbf{x}, a) + \frac{1}{n(\mathbf{x}, a)} (\hat{V} - Q(\mathbf{x}, a))$$

Интуиция, почему это работает: чем больше мы строим дерево, тем меньше зависим от эвристики для этапа Simulation; в пределе после бесконечного числа итераций мы условно построим дерево игры целиком. Чем меньше эта зависимость, тем чаще TreePolicy выбирает хорошие действия. Эвристика из Simulation же лишь указывает на те узлы, куда MCTS быстрее отправится детальнее строить подробное дерево разбора игры, и чем эта эвристика лучше, тем удачнее пройдёт ограниченный перебор.

Вполне можно считать, что расчёт $Q(\mathbf{x}, a)$ при помощи Монте-Карло оценок — это Policy Evaluation, а поиск по дереву при помощи многоруких бандитов — Policy Improvement.

7.3.2. Применение MCTS

MCTS можно «предобучить» перед использованием: проводим много шагов MCTS-процедуры, считая, что корню соответствует s_0 (если распределение на начальное состояние стохастично, придётся сэмплировать s_0 : но тогда, в силу теоремы 81, мы будем искать план, хороший в среднем для возможных исходов). На каждом шаге MCTS-процедуры ровно один лист в дереве «раскрывается» и для каждого пути симулируется одна или несколько игр до конца эпизодов. Полностью дерево для всего MDP мы скорее всего не построим, и однажды придётся остановиться.

Дальше мы хотим нашу стратегию отправить «играть» с реальной средой, и тут возможны варианты. Во-первых, мы можем дерево больше не трогать и просто использовать, например, TreePolicy (возможно, с «выключенным» эксплорейшном): а то есть, изначально нам дали s_0 , которое, условно, соответствует корню. Мы выбираем действие при помощи TreePolicy и отправляем его в реальную среду; после этого мы можем в нашем дереве спустится по дуге, соответствующей выбранному действию, и на следующем шаге воспользоваться TreePolicy в этом узле. Однако, во время «использования» подобной стратегии нам могут встретиться состояния, для которых в дереве ещё нет узлов; тогда либо придётся выбирать действия случайно, либо проводить новые шаги MCTS-процедуры.

По этой причине, так обычно не делают; считается, что в общем случае, MCTS не строит дерево игры один раз и потом использует его во всех играх, а сама по себе является стратегией выбора действия в данном состоянии: просто долгой и тяжёлой в связи с переборной природой. То есть, перед каждой отправкой итогового действия в среду, нужно провести «ещё парочку» итераций MCTS-процедуры. Это означает, что, во-первых, можно вообще предобучения не проводить, а перед каждым выбором действия для реальной среды делать, там, 1000 шагов MCTS-процедуры, а во-вторых, после очередного реального шага, спускаясь на один узел вниз в дереве, мы можем оставлять только поддерево того узла, в который пришли, для оптимизации (наверх по дереву мы никогда не идём — прошлое уже не изменить).

Итоговое действие для реальной среды не обязательно выбирать при помощи TreePolicy (в нём есть учёт эксплорейшна, который при использовании стратегии нам не нужен). Распространённый вариант следующий: выбирать с большей вероятностью те действия, которые MCTS исследовал чаще всего в ходе всей процедуры:

$$\pi(a_0 | s_0) \propto n(\mathbf{x}_0, a_0)^T$$

где T — температура, очередной гиперпараметр.



При «использовании» стратегии в бою обычно используют жёсткий вариант: $a := \underset{a}{\operatorname{argmax}} n(\mathbf{x}_0, a)$.

Алгоритм 28: Стратегия MCTS (одна из вариаций)

Вход: s_0 — текущее состояние реальной среды, $p(s', r | s, a)$ — симулятор, C — гиперпараметр UCB-бандита, \mathbf{x}_0 — корень текущего дерева в памяти с хранением счётов $n(\mathbf{x}, a)$ и оценок $Q(\mathbf{x}, a)$ в дугах, K — число шагов, T — температура.

На k -ом шаге из K :

1. садимся в корень: $\mathbf{x} := \mathbf{x}_0, s := s_0$.
2. инициализируем траекторию симуляции: $\mathcal{T} := (s_0)$
3. пока \mathbf{x} — не лист:
 - выбираем ветку, куда пойти:

$$n(\mathbf{x}) := \sum_a n(\mathbf{x}, a)$$

$$a := \operatorname{argmax}_a \left[Q(\aleph, a) + C \sqrt{\frac{\log n(\aleph)}{n(\aleph, a)}} \right]$$

- генерируем $s', r \sim p(s', r | s, a)$
- сохраняем a, r, s' в \mathcal{T}
- спускаемся по дереву: $\aleph \leftarrow \operatorname{child}(\aleph, a), s \leftarrow s'$

4. для каждого $a \in \mathcal{A}$:

- создаём узел $\hat{\aleph}$ — ребёнка \aleph с дугой для действия a
- симулируем $\mathcal{T}_a \sim \pi^{\text{random}} | s, a$, где π^{random} — случайная стратегия
- инициализируем $n(\hat{\aleph}, a) := 1, Q(\hat{\aleph}, a) := R(\mathcal{T}_a)$

5. для каждой посещённой дуги \aleph, a :

- считаем \hat{V} — суммарный reward-to-go в траектории \mathcal{T} , полученный после посещения данной дуги, где награда после посещения листа оценена как $\frac{1}{|\mathcal{A}|} \sum_a R(\mathcal{T}_a)$.
- $Q(\aleph, a) \leftarrow Q(\aleph, a) + \frac{1}{n(\aleph, a)} (\hat{V} - Q(\aleph, a))$
- $n(\aleph, a) \leftarrow n(\aleph, a) + 1$

Выход: стратегия $\pi(a_0 | s_0) \propto n(\aleph_0, a_0)^T$

Итого, мы получили очень дорогостоящую по времени, разумную по памяти, но работающую процедуру поиска хороших действий в игре.

Пример 110: Попробуем провизуализировать шаг игры при помощи MCTS стратегии. Перед выбором действия будем делать 4 шага MCTS процедуры; выбирать действие для реальной среды будем при помощи $\pi^{\text{MCTS}}(a | s) \propto n(\aleph, a)$, где \aleph — корень дерева.

7.3.3. Дистилляция MCTS

Появляется прекрасная идея: *дистиллировать* MCTS в нейронку. Практически, мы займёмся имитационным обучением с MCTS в качестве эксперта. Для этого мы уже будем проводить этап обучения. Для обучения мы сыграем при помощи вышеописанной стратегии MCTS много игр, и сохраним их траектории⁶ \mathcal{T} . Далее решаем задачу классификации: пытаемся по состоянию s нейросеткой выдавать действия, которые выбрала бы MCTS. В идеале мы получим стратегию, работающую не хуже MCTS, но при этом скорость прямого прохода по сети будет намного, намного выше.

Схожая альтернатива — пытаться выдавать нейросеткой-стратегией $p_\theta(a | s)$ то же распределение, которое выдаёт долгий MCTS перебор; назовём это «целевое распределение» $\pi^{\text{MCTS}}(a | s)$. Тогда функция потерь выглядит так:

$$\mathbb{E}_s \text{KL}(\pi^{\text{MCTS}}(a | s) \| p_\theta(a | s)) = \mathbb{E}_s \sum_a \pi^{\text{MCTS}}(a | s) \log p_\theta(a | s) + \text{const}(\theta) \rightarrow \min_{\theta}, \quad (7.5)$$

где мат.ожидание по s берётся из буфера, из кучи сыгранных при помощи MCTS игр.

И тут возникает желание сделать следующий шаг: если у нас есть нейросеть, которая знает, какие действия в каком состоянии хорошие за счёт моделирования результата MCTS-перебора, может быть мы можем в новой игре воспользоваться ею внутри самого MCTS, использовать эту нейросеть $p_\theta(a | s)$ для ускорения перебора? Тогда мы наверняка сможем построить ещё более хорошую стратегию, дистиллировать её в нейронку, построить ещё более хорошую стратегию, дистиллировать её в нейронку, и так далее. Итак, появляется идея объединить MCTS с нейросетками.

7.3.4. AlphaZero

В алгоритме AlphaZero⁷ вводится нейросеть с параметрами θ , принимающая на вход состояние s , с двумя головами. Одна выдаёт распределение на действиях $p_\theta(a | s)$ («вспомогательную стратегию» или «дистиллированную MCTS»), другая выдаёт скалярную оценку текущего состояния $V_\theta(s)$. Сразу скажем, что p — лишь вспомогательная стратегия, и хотя ею вполне можно будет пользоваться для игры по итогам обучения, но наша итоговая стратегия всё-таки будет перебирать ходы при помощи MCTS и поэтому будет потенциально лучше. Цель $p_\theta(a | s)$ — запоминать с прошлых игр, какие действия были хорошими, ускоряя перебор, а цель $V_\theta(s)$ — запоминать reward-to-go, чтобы не было необходимости проводить симуляции при помощи случайной стратегии на этапе Simulation; вместо симуляции мы теперь просто будем вызывать эту нейросетку.

Итак, за основу алгоритма берётся схема MCTS-стратегии, описанная в алгоритме 28. Как и раньше, в каждом узле \aleph для каждого действия хранится оценка ценности $Q(\aleph, a)$ и счётчик, сколько раз какое действие было попробовано $n(\aleph, a)$; при выборе очередного действия для состояния s повторяется, там, 1600 раз наши четыре этапа MCTS схемы, а каждый эпизод дерево начинает строиться с нуля.

Изменений в схеме ровно два: на этапе Simulation мы вместо того, чтобы играть случайной стратегией до конца игры, просто воспользуемся нашей моделью $\hat{V}_a := V_\theta(\hat{s})$ как оценкой дальнейшего reward-to-go (здесь \hat{s} — симулированное состояние после выбора оцениваемого действия a в листе). Счётчики и скаляры $Q(\aleph, a)$ обновляются как раньше.

Второе изменение — в формуле TreePolicy, которая принимает следующий вид:

$$a := \underset{a}{\operatorname{argmax}} \left[Q(\aleph, a) + C p_\theta(a | s) \frac{\sqrt{n(\aleph)}}{n(\aleph, a) + 1} \right]$$

Здесь C — гиперпараметр; бонус за исследования немножко не похож на UCB-бандита, но имеет примерно тот же смысл (он обратно пропорционален числу выборов действия, а числитель гарантирует, что это слагаемое неограниченно растёт при фиксированном знаменателе и промотивирует выбор любого сколь угодно неоптимального действия рано или поздно). Ключевое изменение — домножение бонуса на $p_\theta(a | s)$. Если действие в прошлых играх никогда не выбиралось по итогам перебора, дистилляция в нейросетку приведёт к низкой вероятности $p_\theta(a | s)$, и такое плохое действие будет выбираться редко. Это позволяет существенно сократить перебор для сред с большим числом⁸ действий $|\mathcal{A}|$: в новых состояниях дерева, где все оценки, условно, $Q(\aleph, a) = 0, n(\aleph, a) = 0$, мы вместо случайного тыканья и исследования всех вариантов будем больше опираться именно на те действия, которые показались хорошими нейросетке. Запоминание информации из предыдущих эпизодов обучения направляет раскрытие дерева «в хорошую область».

⁶замечание: траектории именно реальных игр, в которых каждое действие выбиралось в результате, там, 1000 шагов MCTS процедуры, на каждом из которых MCTS делало кучу симуляций при помощи своей DefaultPolicy; эти симуляции внутри MCTS мы нигде не записываем, поскольку нас интересуют только хорошие действия по результатам перебора.

⁷AlphaZero исторически был обобщением алгоритма AlphaGo для игры в го как общей процедуры обучения стратегии для произвольной игры с доступным симулятором. Для случая игры двух игроков вроде шахмат и го, для которого она изначально и строилась, предполагается, что в симуляторе за противника играет просто недавняя версия текущей MCTS-стратегии; он также строился для детерминированных сред с ненулевой наградой лишь в конце эпизода по типу «выиграл-проиграл», но мы далее опишем сразу чуть-чуть более общую схему.

⁸например, в том же го число доступных действий достигает 19².

Как и раньше, результатом работы MCTS является стратегия $\pi^{\text{MCTS}}(a | s) \propto n(\mathbf{x}_0, a)^T$, где T — гиперпараметр температуры, \mathbf{x}_0 — текущий корень. В ответ на наше действие a , сэмплированное из этой стратегии, получаем истинный сэмпл s' из реальной среды и оставляем от нашего дерева лишь поддерево, соответствующее $\text{child}(\mathbf{x}_0, a)$.

При помощи такой MCTS-стратегии с текущими параметрами нейросети θ играем много-много игр и сохраняем записи реальных траекторий; для каждого состояния s запомним $\pi^{\text{MCTS}}(a | s)$ и считаем реальный reward-to-go, который обозначим за z . Дальше батчами учим V_θ воспроизводить среднее z , минимизируя MSE, а $p_\theta(a | s)$ — воспроизводить π^{MCTS} аналогично (7.5). Ещё добавлен регуляризатор на параметры, итого:

$$\text{Loss}(\theta) := \mathbb{E}_s \left[(z - V_\theta(s))^2 - \sum_a \pi(a | s) \log p_\theta(a | s) + \alpha \|\theta\|^2 \right] \rightarrow \min_{\theta},$$

где в мат.ожидании s берутся из буфера, α — гиперпараметр для регуляризации. После многих итераций дообучения нейросети снова играем много игр при помощи MCTS с новыми параметрами нейросетки θ , снова сохраняем записи реальных игр и докидываем их в буфер, и так далее.

В буфере можно хранить в том числе довольно старые игры, поскольку $\pi^{\text{MCTS}}(a | s)$ можно считать в некотором смысле «хорошой» стратегией даже если $V_\theta(s)$ выдаёт ерунду, а $p_\theta(a | s)$ условно случайная. Действительно, по мере раскрытия дерева первое «перестаёт» использоваться, а второе влияет лишь на исследования внутри самого дерева. Но чем лучше будут эти нейросетки, тем лучше пройдёт MCTS- поиск. Понятно, что $V_\theta(s)$, обучаясь на reward-to-go, учит V-функцию для MCTS-стратегии, породившей сэмпл z , но ей в таком алгоритме не существенно быть критиком именно самой свежей MCTS-стратегии: ведь в исходной схеме MCTS мы на этапе симуляции так вообще доигрывали игру случайной стратегией.

Пример 111 — AlphaGo Zero: Основные детали применения данного алгоритма для го можно посмотреть на [этой картинке](#).

7.3.5. μ -Zero

Обобщим⁹ AlphaZero на случай, когда симулятор нам недоступен. Параметры всех нейросетевых моделей будем обозначать θ , в конечном счёте все части модели будут обучаться end-to-end единой функцией потерь.

На вход стратегии будет приходить реальное состояние среды s_0 . Оно отправляется в кодировщик: нейросеть $h_\theta(s_0)$, которая вернёт латентное представление состояния \mathbf{x}_0 — некоторый компактный вещественный вектор. Далее мы для определения стратегии запускаем MCTS-процедуру, однако ей нужен симулятор. В качестве симулятора мы заведём нейросеть $g_\theta(\mathbf{x}, a)$, которая по данному латентному представлению состояния \mathbf{x} и действию a детерминировано выдаст латентное представление следующего состояния \mathbf{x}' ; а также нейросеть $\hat{r}_\theta(\mathbf{x}, a)$ для моделирования награды за один данный шаг. Поскольку функция g_θ детерминирована, пересчитывать её значение каждый раз, когда MCTS идёт вдоль этой ветки, не понадобится. Нейросети V_θ и p_θ теперь будут тоже принимать на вход компактное латентное представление \mathbf{x} ; это позволит нам внутри дерева запускать их в произвольных узлах.

Осталось понять, как обучать нововведённые функции $g_\theta, \hat{r}_\theta, h_\theta$. Как и в AlphaZero, соберём датасет из нескольких игр текущей версии модели, и дальше проведём несколько итераций улучшения всех параметров θ . Мы сможем сделать это end-to-end. Возьмём некоторый момент реальной игры с состоянием s , reward-to-go с этого момента z и выданной MCTS «хорошой стратегией» π^{MCTS} . Тогда функция потерь из AlphaZero будет в том числе зависеть от параметров функции h_θ :

$$(z - V_\theta(h_\theta(s)))^2 - \sum_a \pi^{\text{MCTS}}(a | s) \log p_\theta(a | h_\theta(s)) + \alpha \|\theta\|^2 \rightarrow \min_{\theta}$$

Добавить слагаемое для обучения модели функции награды, которая должна воспроизводить награду $r(s, a)$ за один шаг, несложно:

$$(r(s, a) - \hat{r}_\theta(h_\theta(s), a))^2 \rightarrow \min_{\theta}$$

Наконец, осталось разобраться с обучением приближения динамики g_θ . Тут есть первое соображение: давайте заглянем на один момент в будущее и посмотрим на реальное $s' \sim p(s' | s, a)$. Мы могли бы воспользоваться этим сэмплом и явно попросить нашу динамику предсказывать латентное представление будущего:

$$\| \underbrace{g_\theta(h_\theta(s), a)}_{\substack{\text{предсказанное} \\ \text{будущее}}} - \underbrace{h_\theta(s')}_{\substack{\text{реальное} \\ \text{будущее}}} \|_2^2 \rightarrow \min_{\theta}$$

⁹для полноты описания алгоритма, в оригинальной статье введён ещё ряд непринципиальных изменений в AlphaZero: используется ещё более длинная формула для TreePolicy сразу с несколькими гиперпараметрами, а также вводится обобщение алгоритма на случай частично наблюдаемых сред (см. раздел 8.5).

Мы так делать не будем (хотя могли бы). Такое слагаемое функции потерь мотивирует латентное представление быть в некотором смысле «простым» для предсказания, и эта простота достигается за счёт ухудшения семантического смысла: какие-то важные фичи для V_θ, p_θ «теряются», поскольку потеря в лоссе для них меньше, чем потеря в лоссе предсказания усложнённого представления $h_\theta(s)$. Нам же важно другое: чтобы латентное представление было «достаточным» для предсказания V_θ, p_θ , будущих наград и хорошей стратегии. Если это достигается, то мы спокойно согласимся как с «несогласованностью» латентного описания (когда результат применения модели динамики g_θ не соответствует латентному описанию реального s' , хотя всё равно позволяет предсказывать хороший reward-to-go и стратегию для этого s'), так и с потерей некоторой информации, несущественной для агента.

Поэтому мы будем обучать g_θ по-другому. Для s' мы знаем целевые переменные для функций $V_\theta, p_\theta, \hat{r}_\theta$. Давайте добавим их лоссы для s' , но входное латентное представление для s' получим не при помощи $h_\theta(s')$, а при помощи $g_\theta(h_\theta(s), a)$. То есть пусть z' — reward-to-go со следующего состояния в реальной игре, $\pi^{\text{MCTS}}(a' | s')$ — результат работы MCTS на следующем шаге s' , тогда добавим такое слагаемое в функцию потерь:

$$(z' - V_\theta(\aleph'_\theta))^2 - \sum_{\hat{a}} \pi^{\text{MCTS}}(\hat{a} | s') \log p_\theta(\hat{a} | \aleph'_\theta) + (r' - \hat{r}_\theta(\aleph'_\theta, a'))^2 + \alpha \|\theta\|^2 \rightarrow \min_{\theta},$$

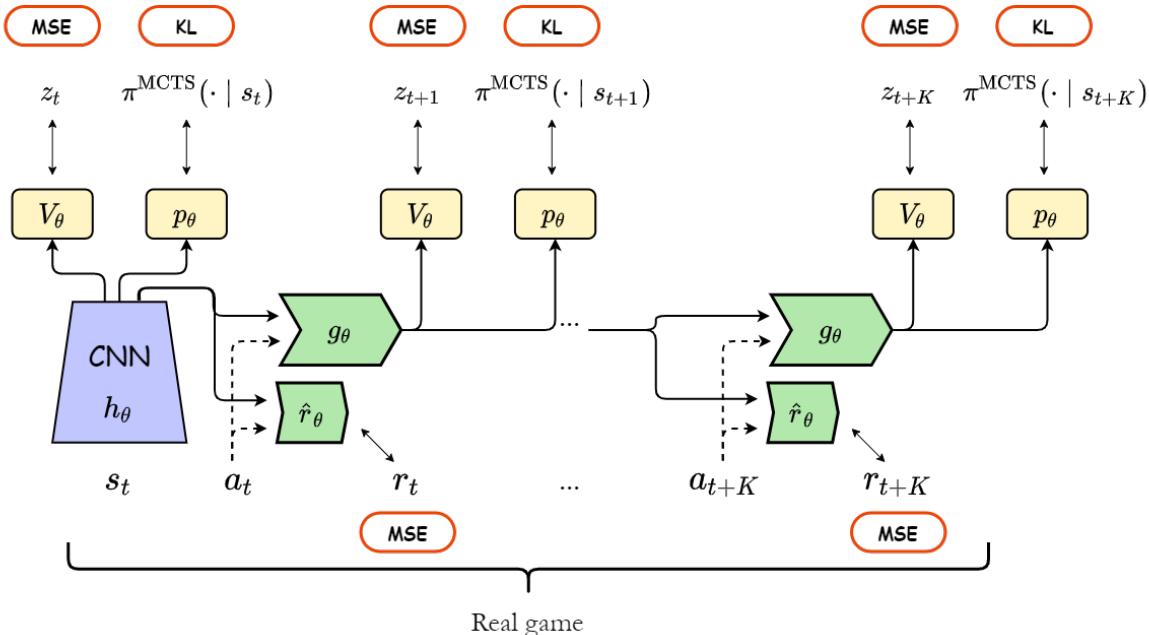
где $\aleph'_\theta := g_\theta(h_\theta(s), a)$.

Заметим, что мы можем взять из буфера целый роллаут длины K , и представление для $s^{(k)}$ получать как k преобразований нашей динамикой g представления первого состояния $h_\theta(s_0)$; аналогично, для него считать лоссы обучения V_θ, p_θ (а также \hat{r}_θ). Все выполненные при этом действия следует брать из роллаута, реальной записи игры. Полученные лоссы все суммируются; формальная функция потерь получается следующая:

$$\sum_{k=0}^K \left[(z^{(k)} - V_\theta(\aleph_\theta^{(k)}))^2 - \sum_{\hat{a}} \pi^{\text{MCTS}}(\hat{a} | s^{(k)}) \log p_\theta(\hat{a} | \aleph_\theta^{(k)}) + (r^{(k)} - \hat{r}_\theta(\aleph_\theta^{(k)}, a^{(k)}))^2 \right] + \alpha \|\theta\|^2 \rightarrow \min_{\theta}, \quad (7.6)$$

где $z^{(k)}$ — reward-to-go, начиная с k -го шага после рассматриваемого момента времени s_0 , латентное представление начального состояния определяется кодировщиком $\aleph_\theta^{(0)} \equiv \aleph := h_\theta(s)$, а последующие — динамикой: $\aleph_\theta^{(k)} := g_\theta(\aleph_\theta^{(k-1)}, a^{(k-1)})$.

Итоговую схему проще представить на картинке:



Подумаем, как полученный алгоритм μ -Zero будет обучаться, то есть «в каком порядке» будут улучшаться нейросетки. Понятно, что MCTS, в котором динамика среды заменена на необученную нейросетку, ничего разумного выдавать не будет. Однако для обучения модели награды за шаг r_θ в любом датасете у нас всегда будет ground truth, поэтому эта сеть постепенно будет обучаться, и с ходом этого обучения h_θ будет постепенно строить осмысленное латентное представление состояний. Наше приближение V_θ , обучающееся на reward-to-go, тоже учится воспроизводить V-функцию «текущей MCTS-стратегии», сколь бы плохой она ни была, и мы помним, что оценочная функция любой стратегии — это путь к её улучшению. Поскольку мы учимся предсказывать будущие reward-to-go в том числе по преобразованному моделью динамики латентному представлению в слагаемом (7.6), модель g_θ научится выдавать такое представление, по которому мы умеем хорошо предсказывать будущие награды. Именно их мы и используем на этапе Simulation в листьях дерева; дерево начнётходить ветви, где reward-to-go больше, и проводить таким образом policy improvement. Дальше эта более хорошая

стратегия будет дистиллироваться в нейронку p_θ , а для этого ещё более информативным будет становиться как латентное представление, так и его преобразование моделью динамики.



Число итераций на этапе планирования для каждого выбора действий для реальной среды здесь имеет поставить сильно меньше, чем в AlphaZero, поскольку понятно, что перебор с неидеальным симулятором менее осмысленный, чем с идеальным. Тем не менее, универсальность алгоритма μ -Zero противостоит его огромной вычислительной сложности: необходимо делать огромное количество итераций MCTS процедуры, требующей постоянных вызовов нейросетей.

§7.4. Планирование для непрерывного управления

7.4.1. Прямое дифференцирование

Построим планировщик для непрерывных пространств действий. Предположим, что модель динамики среды и награды (то есть все функции из постановки задачи) нам известны, причём даны не просто как симулятор, а как дифференцируемые и, главное, детерминированные функции. Для упрощения повествования будем считать, что эпизод состоит из T шагов, $\gamma = 1$. Дополнительно рассматривая случай непрерывного пространства действий $\mathcal{A} \equiv \mathbb{R}^d$, мы получим ситуацию, в которой мы можем просто дифференцировать вдоль оси времени и оптимизировать награду по действиям «напрямую».

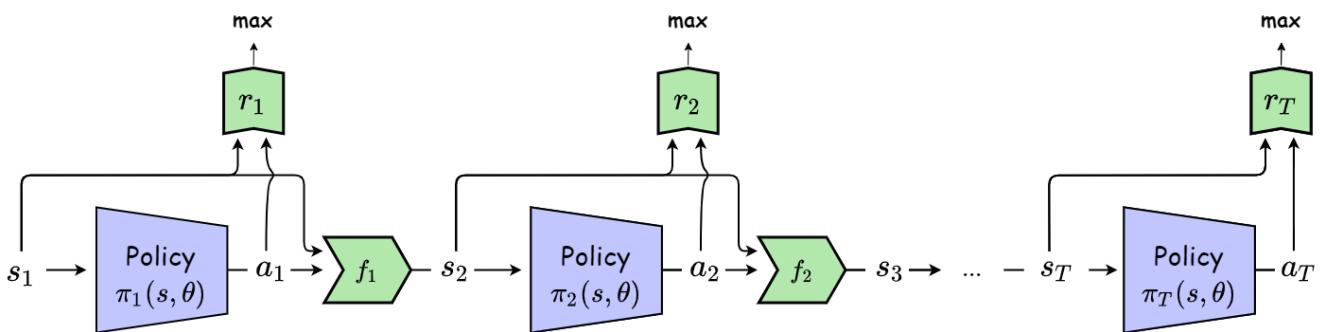
В силу детерминированности функции переходов наш выбор действий полностью определяет траекторию. То есть искать будем даже не оптимальную детерминированную стратегию, а **оптимальное управление** — набор действий $a_1, a_2 \dots a_T$, которые приведут к наилучшей траектории.

Поскольку мы здесь немного уходим в мир оптимального управления, для соблюдения каноничности не будем предполагать однородность: функции награды и динамики среды дополнительно зависят от дискретной переменной времени $t \in \{1 \dots T\}$.

Итого, рассматриваем следующую задачу:

$$\begin{cases} \sum_t^T r_t(s_t, a_t) \rightarrow \max \\ s_t = f_t(s_{t-1}, a_{t-1}) \end{cases} \quad (7.7)$$

Сразу заметим, что при сделанных предположениях можно попытаться решать задачу «в лоб». Промоделируем детерминированную стратегию нейросеткой $\pi_t(s, \theta)$ с параметрами θ , и рассмотрим такой вычислительный граф:



Здесь суммарная награда — дифференцируемая функция от θ , поэтому параметры стратегии можно оптимизировать напрямую. Таким образом, у нас получилась «дифференцируемая» задача динамического программирования. Однако, если T велико, то градиент вдоль такого вычислительного графа будет подвержен затуханию, а сам такой поиск в пространстве траектории будет содержать множество локальных минимумов.

Нам хочется построить, если угодно, более специализированный метод оптимизации для задачи вида 7.7. Как мы обычно действуем при построении методов оптимизации: вот у нас есть некоторая сложная функция, которую мы хотим оптимизировать, возможно, с ограничениями (а аналитически ничего не считается). Заводим некоторое приближение решения, в данном случае — какой-то план $a_1, a_2 \dots a_T$. Затем приблизим в окрестности этой точки оптимизируемую функцию какой-то простой моделью, с которой мы умеем работать, за счёт простоты модели делаем какой-то шаг и попадаем в новую точку, новый план $a_1, a_2 \dots a_T$. Оказывается, что в контексте задачи 7.7 мы можем поступить умнее, чем просто подставить ограничения внутрь оптимизируемого функционала и приблизить всё линейной моделью, как мы делаем при градиентном спуске. Оказывается, если мы заменим функции наград на квадратичное приближение, а ограничение — функцию динамики — на линейное приближение, то мы сможем за счёт структуры задачи просто методом динамического программирования аналитически решить задачу! Тогда «проведение шага» для нас будет просто сдвиг в точку нового оптимального плана для подобной локальной аппроксимации.

7.4.2. Linear Quadratic Regulator (LQR)

Обсудим, как можно аналитически найти решение задачи в ситуации, когда f, r — «простые» функции, а именно линейная и квадратичная соответственно. Пусть функция f — линейна, функция r — квадратична (hence the name). Введём соответствующие обозначения:

$$f_t(s, a) = F_t \begin{bmatrix} s \\ a \end{bmatrix} + f_t \quad (7.8)$$

$$r_t(s, a) = \frac{1}{2} \begin{bmatrix} s \\ a \end{bmatrix}^T R_t \begin{bmatrix} s \\ a \end{bmatrix} + \begin{bmatrix} s \\ a \end{bmatrix}^T r_t$$

Напоминаем, что выход функции f — следующее состояние (т.е. вектор), поэтому F_t — матрица, f_t — вектор. Выход функции r — скаляр, поэтому R_t — матрица, r_t — вектор. Свободный член квадратичной формы не пишем, потому что он не зависит от состояний и действий (траектории), поэтому при оптимизации это константная неоптимизируемая награда, и она несущественна. Матрицы F_t, R_t и вектора r_t, f_t считаем известными.

Нам понадобится ещё чуть-чуть обозначений. Будем считать, что блоки матрицы R_t и блоки вектора r_t выглядят следующим образом:

$$R_t := \begin{bmatrix} R_{t,s,s} & R_{t,s,a} \\ R_{t,a,s} & R_{t,a,a} \end{bmatrix}; \quad r_t := \begin{bmatrix} r_{t,s} \\ r_{t,a} \end{bmatrix}$$

Для упрощения выкладок также будем полагать, что $R_{t,s,a} = R_{t,a,s}^T$.

Поймём, что мы можем решить задачу обычным динамическим программированием «с конца».

Теорема 82: Оптимальное действие в последний момент времени a_T^* — линейная форма от состояния s_T .

Доказательство. Рассмотрим последний момент времени T и распишем оптимальную Q-функцию (в силу отказа от однородности оценочные функции также зависят от времени):

$$Q_T^*(s_T, a_T) = r_T(s_T, a_T) = \frac{1}{2} \begin{bmatrix} s_T \\ a_T \end{bmatrix}^T R_T \begin{bmatrix} s_T \\ a_T \end{bmatrix} + \begin{bmatrix} s_T \\ a_T \end{bmatrix}^T r_T$$

И всё, потому что на сим эпизод заканчивается и награды больше не будет. Мы легко можем найти оптимальное действие, если на последнем шаге оказались в состоянии s_T , промаксимизировав Q_T^* по действию a_T :

$$a_T^* = \underset{a_T}{\operatorname{argmax}} Q_T^*(s_T, a_T)$$

Ищем оптимум квадратичной формы*, приравняв градиент Q_T^* по действиям к нулю:

$$\nabla_{a_T} Q_T^*(s_T, a_T) = R_{T,a,a} a_T + R_{T,a,s} s_T + r_{T,a} = 0$$

$$a_T^* = -R_{T,a,a}^{-1} (R_{T,a,s} s_T + r_{T,a})$$

Видно, что оптимальное действие — линейная форма от последнего состояния. ■

* формально здесь нужно оговариваться, что на матрицу R_t и вектор r_t надо накладывать некоторые ограничения, например, что эта квадратичная форма отрицательно определена и у неё есть искомый глобальный максимум. Здесь и далее мы на условиях регулярности останавливаются не будем.



Видно, что придётся обращать матрицу $R_{T,a,a}$, но размерность пространства действий в задачах такой постановки имеет обычно разумные размеры (обычно не более 100), поэтому мы не боимся.

Введём обозначения, чтобы привести форму a_T^* к стандартному виду:

$$K_T := -R_{T,a,a}^{-1} R_{T,a,s}, \quad k_T := -R_{T,a,a}^{-1} r_{T,a}$$

$$a_T^* = K_T s_T + k_T$$

Теорема 83: Ценность последнего состояния $V^*(s_T)$ — квадратичная форма от состояния s_T .

Доказательство. По определению, в силу детерминированности среды, $V^*(s_T) = Q^*(s_T, a_T^*)$; осталось

заметить, что подстановка линейной формы в линейную даст квадратичную:

$$\begin{aligned} V^*(s_T) &= Q^*(s_T, a_T^*) = \frac{1}{2} \begin{bmatrix} s_T \\ a_T^* \end{bmatrix}^T R_T \begin{bmatrix} s_T \\ a_T^* \end{bmatrix} + \begin{bmatrix} s_T \\ a_T^* \end{bmatrix}^T r_T = \\ &= \frac{1}{2} \begin{bmatrix} s_T \\ K_T s_T + k_T \end{bmatrix}^T R_T \begin{bmatrix} s_T \\ K_T s_T + k_T \end{bmatrix} + \begin{bmatrix} s_T \\ K_T s_T + k_T \end{bmatrix}^T r_T = (*) \end{aligned}$$

Просто перегруппируя слагаемые, видим квадратичную форму от состояний s_T . Сразу запишем её в канонических обозначениях:

$$\begin{aligned} V_T &:= R_{T,s,s} + K_T^T R_{T,a,a} K_T + K_T^T R_{T,a,s} + R_{T,s,a} K_T \\ v_T &:= R_{T,s,a} k_T + r_T^T K_T^T r_T \\ (*) &= \frac{1}{2} s_T^T V_T s_T + v_T^T s_T, \quad \blacksquare \end{aligned}$$

Осталось понять, что мы можем раскручивать это к началу времён, не выходя из квадратичных форм. Так и есть.

Теорема 84: Оптимальные оценочные функции Q_t^* — квадратичные формы от $\begin{bmatrix} s_t \\ a_t \end{bmatrix}$, оптимальные действия a_t^* — линейные формы от s_t , а оптимальные оценочные функции V_t^* — квадратичные формы от s_t .

Доказательство. Из определений и детерминированности:

$$Q_{T-1}^*(s_{T-1}, a_{T-1}) = r_{T-1}(s_{T-1}, a_{T-1}) + V_T^*(s_T) = r_{T-1}(s_{T-1}, a_{T-1}) + V_T^* \left(F_T \begin{bmatrix} s_{T-1} \\ a_{T-1} \end{bmatrix} + f_T \right) = (*)$$

В последнее слагаемое подставили линейную форму динамики среды (7.8). При этом подстановка линейной формы в квадратичную оставит её квадратичной.

$$\begin{aligned} (*) &= r_{T-1}(s_{T-1}, a_{T-1}) + \frac{1}{2} \left[F_T \begin{bmatrix} s_{T-1} \\ a_{T-1} \end{bmatrix} + f_T \right]^T V_T \left[F_T \begin{bmatrix} s_{T-1} \\ a_{T-1} \end{bmatrix} + f_T \right] + v_T^T \left[F_T \begin{bmatrix} s_{T-1} \\ a_{T-1} \end{bmatrix} + f_T \right] = \\ &= r_{T-1}(s_{T-1}, a_{T-1}) + \frac{1}{2} \begin{bmatrix} s_{T-1} \\ a_{T-1} \end{bmatrix}^T F_T^T V_T F_T \begin{bmatrix} s_{T-1} \\ a_{T-1} \end{bmatrix} + \begin{bmatrix} s_{T-1} \\ a_{T-1} \end{bmatrix}^T (F_T^T V_T f_T + F_T^T v_T) + \text{const}(a_{T-1}) \end{aligned}$$

Переписываем в виде квадратичной формы:

$$\begin{aligned} Q_{T-1}^*(s_{T-1}, a_{T-1}) &= \frac{1}{2} \begin{bmatrix} s_{T-1} \\ a_{T-1} \end{bmatrix}^T Q_{T-1} \begin{bmatrix} s_{T-1} \\ a_{T-1} \end{bmatrix} + \begin{bmatrix} s_{T-1} \\ a_{T-1} \end{bmatrix}^T q_{T-1} + \text{const}(a_{T-1}) \\ Q_{T-1} &:= R_{T-1} + F_T^T V_T F_T \\ q_{T-1} &:= r_{T-1} + F_T^T V_T f_T + F_T^T v_T \end{aligned}$$

Коли это снова квадратичная формула, мы можем посчитать оптимальное действие, которое будет линейной формой:

$$a_{T-1} = K_{T-1} s_{T-1} + k_{T-1}$$

$$K_{T-1} := -Q_{T-1,a,a}^{-1} Q_{T-1,a,s}, \quad k_{T-1} := -Q_{T-1,a,a}^{-1} q_{T-1,a}$$

Подставляем её в V_{T-1}^* и получаем квадратичную и так далее. ■

Собираем всё вместе в единую схему.

Алгоритм 29: Обратный проход в LQR

Вход: F_t, f_t — функция динамики, R_t, r_t — функция награды.

Инициализировать $V_{T+1} = 0_{|\mathcal{S}| \times |\mathcal{S}|}, v_{T+1} = 0_{|\mathcal{S}|}$
for t от T до 1:

1. считаем Q-функцию:

$$Q_t := R_t + F_{t+1}^T V_{t+1} F_{t+1}$$

$$q_t := r_t + F_{t+1}^T V_{t+1} f_{t+1} + F_{t+1}^T v_{t+1}$$

2. считаем оптимальную стратегию:

$$K_t := -Q_{t,a,a}^{-1} Q_{t,a,s}$$

$$k_t := -Q_{t,a,a}^{-1} q_{t,a}$$

3. считаем V-функцию:

$$V_t := Q_{t,s,s} + K_t^T Q_{t,a,a} K_t + K_t^T Q_{t,a,s} + Q_{t,s,a} K_t$$

$$v_t := Q_{t,s,a} k_t + q_t^T K_t^T q_t$$

Выход: $\pi_t(s) := K_t s + k_t$

Поскольку в рамках сделанных предположений мы на самом деле детерминированно знаем, в каких состояниях окажемся (считаем стартовое состояние s_1 также известным), мы можем просто вывести оптимальное управление:

Алгоритм 30: Прямой проход в LQR

Вход: K_t, k_t — стратегия с прямого прохода, f — функция динамики.

for t от 1 до T :

1. $a_t = K_t s_t + k_t$

2. $s_{t+1} = f(s_t, a_t)$

Выход: a_1, a_2, \dots, a_T — план.

7.4.3. Случай шумной функции перехода

LQR обобщается на случай недетерминированных сред, если предположить, что динамика среды — нормальное распределение с линейной функцией от предыдущих состояний-действий и какой-то фиксированной (зависящей только от момента времени t , но не состояний-действий) матрицей ковариации:

$$p(s_t | s_{t-1}, a_{t-1}) := \mathcal{N}(f_t(s_{t-1}, a_{t-1}), \Sigma_t) \quad (7.9)$$

Формулу (7.9) можно интерпретировать как «зашумление сенсоров», причём зашумление не зависит от того, в какой области пространства состояний мы оказались.

Теорема 85: В предположении (7.9) схема LQR остаётся неизменной.

Доказательство. Единственное, что поменялось — это зависимость V-функции от Q-функции:

$$Q_{t-1}^*(s_{t-1}, a_{t-1}) = r_{t-1}(s_{t-1}, a_{t-1}) + \mathbb{E}_{s_t} V_t^*(s_t)$$

Покажем, что формулы для матрицы Q_{t-1} и вектора q_{t-1} не поменялись, а изменилась только константа (которая на вывод оптимальной стратегии не влияет).

$$\mathbb{E}_{s_t} V_t^*(s_t) = \mathbb{E}_{s_t} \left[\frac{1}{2} s_t^T V_t s_t + v_t^T s_t + \text{const}(s_t) \right] = (*)$$

Итак, нужно взять мат.ожидание квадратичной формы по гауссиане. Лезем^{*} в matrix cookbook (ур. (318)) и видим:

$$\mathbb{E}_{s_t \sim \mathcal{N}(\mu_t, \Sigma_t)} s_t^T V_t s_t = \text{Tr}(V_t \Sigma_t) + \mu_t^T V_t \mu_t$$

где $\mu_t := f_t(s_{t-1}, a_{t-1})$ — среднее гауссианы, Tr — операция взятия следа.

Итого получим:

$$(*) = \frac{1}{2} \text{Tr}(V_t \Sigma_t) + \frac{1}{2} \mu_t^T V_t \mu_t + v_t^T \mu_t + \text{const}(s_t) = V_t^*(\mu_t) + \text{const}(s_t)$$

то есть поменялась исключительно константа, которая на оптимальное управление не влияет. ■

* вывод не так сложен, нужно лишь применить трюк, что след скаляра равен скаляру:

$$\mathbb{E}_{s_t \sim \mathcal{N}(\mu_t, \Sigma_t)} s_t^T V_t s_t = \mathbb{E}_{s_t \sim \mathcal{N}(\mu_t, \Sigma_t)} \text{Tr}(s_t^T V_t s_t) = (*),$$

затем применить свойство следа $\text{Tr}(ABC) = \text{Tr}(BCA)$ и получить

$$(*) = \mathbb{E}_{s_t \sim \mathcal{N}(\mu_t, \Sigma_t)} \text{Tr}(V_t s_t s_t^T).$$

Осталось занести мат.ожидание внутрь следа (след — линейная операция), и вспомнить, что мат.ожидание $s_t s_t^T$ по гауссиане есть $\Sigma + \mu \mu^T$.

7.4.4. Iterative LQR (iLQR)

Вернёмся к детерминированному случаю. Что делать, если функции f, r нам известны, но не являются линейными и квадратичными соответственно? Мы хотели воспользоваться локальным приближением этих функций в окрестности некоторого текущего плана, и в качестве приближений будем использовать разложение в ряд Тейлора до первого и до второго члена соответственно.

Итак, возьмём какой-нибудь план и просчитаем честным прямым проходом точные значения состояний, в которых мы окажемся. Для полученной траектории $\hat{s}_1, \hat{a}_1, \hat{s}_2 \dots \hat{s}_T, \hat{a}_T$ разложим функции переходов и награды в Тейлора следующим образом:

$$f_t(s_{t-1}, a_{t-1}) \approx f_t(\hat{s}_{t-1}, \hat{a}_{t-1}) + \nabla_{s,a} f(\hat{s}_{t-1}, \hat{a}_{t-1})^T \begin{bmatrix} s_{t-1} - \hat{s}_{t-1} \\ a_{t-1} - \hat{a}_{t-1} \end{bmatrix}$$

$$r_t(s_t, a_t) \approx \frac{1}{2} \begin{bmatrix} s_t - \hat{s}_t \\ a_t - \hat{a}_t \end{bmatrix}^T \nabla_{s,a}^2 r_t(\hat{s}_t, \hat{a}_t) \begin{bmatrix} s_t - \hat{s}_t \\ a_t - \hat{a}_t \end{bmatrix} + \nabla_{s,a} r(\hat{s}_t, \hat{a}_t)^T \begin{bmatrix} s_t - \hat{s}_t \\ a_t - \hat{a}_t \end{bmatrix}$$

Вводим обозначения аналогично LQR:

$$F_t := \nabla_{s,a} f(\hat{s}_{t-1}, \hat{a}_{t-1}) \quad f_t := f_t(\hat{s}_{t-1}, \hat{a}_{t-1}) \tag{7.10}$$

$$R_t := \nabla_{s,a}^2 r_t(\hat{s}_t, \hat{a}_t) \quad r_t := \nabla_{s,a} r(\hat{s}_t, \hat{a}_t) \tag{7.11}$$

Используя LQR с такой приближённой моделью динамики и награды, мы можем пересчитать оптимальную траекторию, проделав backward pass, а затем и forward pass (причём на шаге forward pass, естественно, пользуясь истинными точными функциями f, r , а не их разложениями в Тейлора). Технически, нужно только учесть, что в этой рассматриваемой задаче мы «переконстрировали» пространства состояний и действий: LQR тут на шаге t работает с «новым» пространством состояний $s_t - \hat{s}_t$ и «новым» пространством действий $a_t - \hat{a}_t$. Поэтому, чтобы окончательно получить оптимальную траекторию на этапе forward pass помимо использования истинной функции переходов нужно ещё учесть эти центрирования при вызове стратегии:

$$a_t = K_t(s_t - \hat{s}_t) + k_t + \hat{a}_t$$

Это соответствует простому учёту добавленных слагаемых в свободном векторе:

$$k_t \leftarrow k_t - K_t \hat{s}_t + \hat{a}_t \tag{7.12}$$

В полученной траектории снова раскладываем модели в Тейлора, и так далее до удовлетворения.

Алгоритм 31: Iterative LQR (iLQR)

Вход: f — функция динамики, r — функция награды.

Проинициализировать траекторию $\hat{s}_1, \hat{a}_1, \hat{s}_2 \dots \hat{s}_T, \hat{a}_T$ при помощи случайной стратегии.

На каждом шаге:

1. Получить $\mathbf{F}_t, \mathbf{f}_t, \mathbf{R}_t, \mathbf{r}_t$ по формулам (7.10) и (7.11)
2. Получить матрицы $\mathbf{K}_t, \mathbf{k}_t$ при помощи алгоритма LQR с матрицами динамики $\mathbf{F}_t, \mathbf{f}_t$ и награды $\mathbf{R}_t, \mathbf{r}_t$
3. Учесть коррекцию (7.12)
4. При помощи стратегии $\pi(s_t) = \mathbf{K}_t s_t + \mathbf{k}_t$ заспавнить траекторию $\hat{s}_1, \hat{a}_1, \hat{s}_2 \dots \hat{s}_T, \hat{a}_T$, используя честный прямой проход с использованием точных функций \mathbf{f}, \mathbf{r} .

Выход: $\hat{a}_1, \hat{a}_2, \dots, \hat{a}_T$ — план.

Мы построили планировщик для непрерывных пространств действий с произвольной (дифференцируемой) динамикой. Мы находимся в некотором состоянии, предполагаем свою будущую траекторию; рассматриваем некоторое приближение поведения среды, которое достаточно точно для предположенной траектории (это разложение в ряд Тейлора); находим оптимальную траекторию при помощи LQR; получаем новую траекторию; рассматриваем приближение поведения среды для новой траектории и так далее до сходимости.

В частности, если модель среды нам неизвестна, iLQR можно применять в качестве планировщика для обученных (например, нейросетевых) приближений динамики среды — см. общую схему model-based подхода 27.

ГЛАВА 8

Next Stage

Если уж мы рассматриваем задачу RL как попытку создания алгоритма «искусственного интеллекта», то мы должны дополнительно учесть следующие три факта:

- понятно, что в одной и той же среде агент может ставить себе совершенно разные задачи; интеллектуальное обучение должно позволять обобщать решения одних задач на другие, решать сложные задачи, состоящие из составных частей, и, наконец, уметь самостоятельно ставить самому себе «промежуточные» задачи.
- в общем случае, текущее наблюдение среды не описывает её состояние полностью, и агент, во-первых, должен обладать модулем памяти для запоминания предыдущих наблюдений, во-вторых, действовать в условиях неопределённости.
- наконец, в среде могут присутствовать другие агенты, которые могут иметь как схожие, так и противоположные цели, передавать вспомогательную информацию или, в частности, играть роль эксперта, демонстрирующих оптимальное (или полезное) поведение.

В этой главе мы обсудим ряд избранных идей в направлении этих соображений, которые не вписались в предыдущее повествование.

§8.1. Имитационное обучение

8.1.1. Клонирование поведения

Зачастую продемонстрировать то, как надо решать задачу, проще, чем описать её в терминах награды.

Пример 112: Чтобы обучить self-driving car, проще посадить реального водителя за руль и попросить собрать примеры траекторий, чем описать функцию награды, описывающую правила движения.

Пример 113: Чтобы обучить робота переливать воду из стакана в стакан, проще не придумать функцию награды, описывающую такую задачу, а взять руку робота и несколько раз, «держа его за ручку», перелить воду из стакана в стакан.

Допустим, некоторый эксперт π^{expert} повзаимодействовал со средой и собрал для нас набор траекторий (\mathcal{T}). Если мы уверены в крутизне эксперта (то есть готовы считать его стратегию оптимальной или в достаточной степени около-оптимальной), то задача обучения собственной стратегии π_θ на первый взгляд сводится к задаче обучения с учителем:

Определение 97: *Клонированием поведения* (behavioral cloning) называется обучение стратегии воспроизводить действия эксперта:

$$\sum_{\mathcal{T}} \sum_{s,a \in \mathcal{T}} \log \pi_\theta(a | s) \rightarrow \max_{\theta} \quad (8.1)$$

В случае успеха, мы можем надеяться на то, что наша стратегия π_θ будет вести себя не хуже эксперта. При достаточном количестве данных и достаточной крутизне эксперта такой вариант вполне может сработать. При этом мы можем даже не знать или не заниматься придумыванием функции награды: она никак не участвует в обучении.



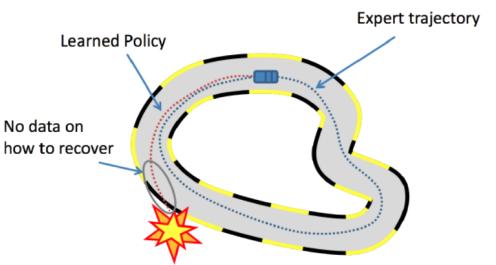
Кстати, учиться выбирать те действия из состояний, которые привели к наиболее удачным сессиям (эпизодам с наибольшей наградой) может быть полезно всегда (в частности, на этом основан кросс-энтропийный

метод 5). Например, в policy gradient алгоритмах можно пробовать запоминать эпизоды обучения, в которых агенту удалось набрать больше всего награды, и с некоторым весом использовать градиент (8.1), посчитанный по этим лучшим траекториям, для обучения актёра. Такой подход называется *самоимитацией* (self-imitation): просто учимся воспроизводить свои собственные же наилучшие попытки.

Однако, подобная «дистилляция знаний» одного актёра в другого не учитывает то, что мы работаем с последовательным принятием решений. Как только наша стратегия чуть-чуть отклонится от экспертного поведения (а это, так или иначе,inevitably) или просто получит необычный отклик от среды, она может оказаться в той области пространства состояний, где эксперта никогда не было, и примеров правильного поведения в обучающей выборке не встречалось. Тогда модель будет предсказывать действия для состояний, которых не было при обучении (этую проблему называют *covariate shift*), и в такой момент клонированное поведение может выдать сколько угодно безумные результаты.



Так или иначе, клонирование поведение может сыграть роль отличной инициализации, существенно ускоряющей процесс обучения. Если для off-policy обучения имеет смысл просто докинуть экспертные траектории в буфер, то для on-policy алгоритмов при наличии экспертных траекторий имеет смысл предобучить стратегию при помощи клонирования поведения.



Во многих задачах можно побороться с этой проблемой при помощи более «тищательного» или «хитрого» процесса сбора данных; например, специально закатываясь в те области MDP, куда может заехать «сломанная» стратегия.

Пример 114 — DAgger: Одна из универсальных идей звучит так: после клонирования поведения запустить полученную стратегию в среду, собрать набор тех состояний, которые она посетила, и попросить эксперта «разметить» их: выбрать оптимальные действия. Но такой алгоритм предполагает, что у нас есть подобное «средство разметки», за счёт которого задача сводится к обучению с учителем.

Пример 115 — Quadcopter Navigation in the Forest: Иногда возможно извернуться и придумать какое-нибудь ухищрение, как всё-таки получить в RL «правильные ответы». Например, в этом примере квадрокоптер учится лететь вдоль лесной тропинки, выбирая на каждом шаге из трёх действий: влево, вперёд или вправо. Чтобы собрать «обучающую выборку» для него, человек надел шлем с тремя камерами, смотрящими вправо, вперёд и влево, и пошёл по центру лесной тропинки. Собирается такой датасет: для камеры, смотрящей влево, правильный ответ — действие «вправо», для центральной — «вперёд», для правой — «влево». Всё свелось к обучению классификатора.

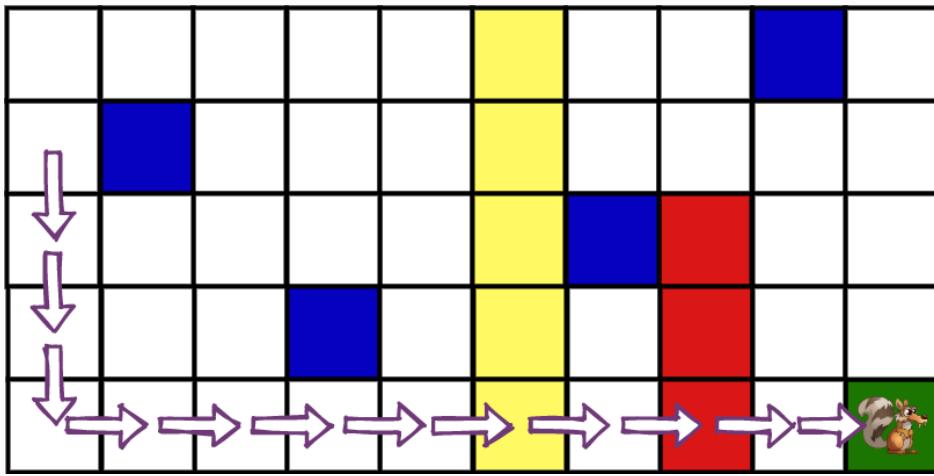
8.1.2. Обратное обучение с подкреплением (Inverse RL)

Почему клонирование поведения — неидеальное решение? На самом деле, в разных областях пространства состояний у наших решений различается «критичность»: если мы ошибёмся в одном месте, наше будущее поменяется не так сильно, и эксперт в таких областях сам может выбирать довольно разнообразные действия, а в других местах от нашего решения существенно зависит дальнейшая награда, и в них крайне важно безошибочно выбрать то же действие, что и эксперт. Это означает, что функция награды как обучающий сигнал более информативен. Но зачастую функция награды нам напрямую недоступна.

Определение 98: Задачей *обратного обучения с подкреплением* (inverse reinforcement learning, IRL) называется задача по набору траекторий (\mathcal{T}) оптимального агента восстановить функцию награды, которую он максимизирует.

За счёт обратного обучения с подкреплением можно попытаться получить функцию награды, которая позволит обучить при помощи всего арсенала RL алгоритмов куда более близкую к оптимальной стратегии, чем простое клонирование поведения. Однако понятно, что задача обратного обучения с подкреплением «некорректна» в том смысле, что допускает всякие дурацкие решения: например, $r(s, a) = 0$ всегда «подходит» в качестве ответа. Какой критерий качества в этой задаче, то есть как понять, насколько адекватна придуманная функция награды?

Пример 116: Рассмотрим клеточный мир, в котором агент может ходить вправо-влево-вниз-вверх. Для простоты также допустим, что функция награды — детерминированная, зависит только от состояний, и на клетках одного цвета её значения совпадают. Что мы тогда можем о ней сказать, имея на руках одну экспертную траекторию, порождённую оптимальной стратегией?



На самом деле, не так много. Агент, видимо, стремился в зелёную клетку; наверное, за неё полагается положительная награда. Через красную клетку агент пропшёл, хотя мог бы обойти за счёт более позднего попадания в зелёную клетку; видимо, это того не стоило, и за красную клетку награда, если и отрицательная, то совсем маленькая. Могла ли она быть положительной? Тогда бы эксперт, наверное, походил бы по красным клеткам; видимо, награда за зелёную сильно выгоднее. Синие клетки агент избегал; видимо, они или дают штраф, или ноль, поскольку если бы они давали бонус, то было бы выгодно добираться до зелёной клетки-цели через них. Наконец, про жёлтые клетки сказать почти ничего нельзя: агенту пришлось бы в любом случае пройти через них, чтобы добраться до зелёной клетки, и поэтому в них может быть как штраф (но не очень большой), так и бонус (но тоже не очень большой).

Ещё одна проблема задачи в том, что эксперт, обычно, не является абсолютно оптимальным. Люди не перемещаются по комнате до целей по строго прямой линии, а ходы шахматного эксперта не являются абсолютно истиной оптимальных действий.

Для упрощения формул будем везде далее полагать $\gamma = 1$. Введём следующее предположение: оптимальная стратегия π^* стохастична и генерирует такие траектории, что:

$$p(\mathcal{T} | \pi^*) \propto e^{R(\mathcal{T})} \prod_{t \geq 0} p(s_{t+1} | s_t, a_t) \quad (8.2)$$

Иначе говоря, вероятность оптимальной стратегии сгенерировать ту или иную траекторию пропорциональна суммарной кумулятивной награде. Мы считаем, что стратегия эксперта, сгенерировавшего для нас датасет, удовлетворяет этому предположению. Такое предположение позволяет нам записать правдоподобие траекторий: насколько вероятно увидеть траекторию \mathcal{T} , если она пришла из оптимального эксперта, максимизированного награду в терминах (8.2).

Откуда это предположение свалилось? На самом деле, это уже знакомый нам Maximum Entropy RL. Действительно: рассмотрим задачу поиска стратегии, которая порождает траектории из распределения (8.2):

$$\text{KL}(p(\mathcal{T} | \pi) \| p(\mathcal{T} | \pi^*)) \rightarrow \min_{\pi} \quad (8.3)$$

Теорема 86: Задача (8.3) эквивалентна задаче Maximum Entropy RL (6.7).

Доказательство. Распишем (8.3):

$$\text{KL}(p(\mathcal{T} | \pi) \| p(\mathcal{T} | \pi^*)) = \mathbb{E}_{\mathcal{T} \sim \pi} \underbrace{\sum_{t \geq 0} \log \pi(a_t | s_t) + \log p(s_{t+1} | s_t, a_t)}_{\log p(\mathcal{T} | \pi^*) \text{ из (8.2)}} - \quad (8.4)$$

$$- \mathbb{E}_{\mathcal{T} \sim \pi} \underbrace{\sum_{t \geq 0} \log p(s_{t+1} | s_t, a_t) - r_t - \text{const}(\pi)}_{\log p(\mathcal{T} | \pi^*) \text{ из (8.2)}}, \quad (8.5)$$

где $\text{const}(\pi)$ — нормировочная константа распределения (8.2). Убирая сокращающиеся логарифмы вероятностей переходов и домножая на минус единицу, получаем:

$$\mathbb{E}_{\mathcal{T} \sim \pi} \sum_{t \geq 0} [r_t - \log \pi(a_t | s_t)] \rightarrow \max_{\pi},$$

что есть в точности Maximum Entropy RL. ■

Оговоримся, что на самом деле у нас нет гарантий, что мы сможем проминимизировать KL-дивергенцию до точного нуля. Другими словами, нельзя сказать, что оптимальная стратегия в Max. Entropy RL обязательно удовлетворяет свойству (8.2).

Утверждение 79: Ноль в задаче (8.3) не обязательно достижим.

Контрпример. Рассмотрим MDP с единственным состоянием, в котором от действий ничего не зависит, но с вероятностью 0.5 агент получает $+\log 2$, а с вероятностью 0.5 агент получает $+0$, после чего игра заканчивается. Распределение (8.2) говорит, что оптимальная стратегия так выбирает действия, что траектория с наградой $+\log 2$ встречает с вероятностью $\frac{2}{3}$, а с наградой $+0$ — с вероятностью $\frac{1}{3}$, однако это невозможно: любая стратегия будет получать их с вероятностями 0.5. ■

Таким образом, теорема 86 говорит, что в Maximum Entropy RL можно считать, что оптимизация движет нашу стратегию в сторону (8.3), и поэтому это предположение для моделирования экспериментального поведения можно считать разумным, но в строгом смысле оно для оптимальных стратегий не выполняется.

8.1.3. Guided Cost Learning

Аппроксимируем функцию награды нейросетью $r_\theta(s, a)$. Тогда правдоподобие одной траектории в предложении (8.2) равно:

$$p_\theta(\mathcal{T} | \pi^*) = \frac{e^{R_\theta(\mathcal{T})} \prod_{t \geq 0} p(s_{t+1} | s_t, a_t)}{Z(\theta)}$$

где $R_\theta(\mathcal{T}) := \sum_{s, a \in \mathcal{T}} r_\theta(s, a)$ — текущая аппроксимация кумулятивной награды, а нормировочная константа, внимание, зависит от параметров нейросети θ :

$$Z(\theta) := \int_{\mathcal{T}} e^{R_\theta(\mathcal{T})} \prod_{t \geq 0} p(s_{t+1} | s_t, a_t) d\mathcal{T} \quad (8.6)$$

Рассмотрим логарифм правдоподобия:

$$\log p_\theta(\mathcal{T} | \pi^*) = R_\theta(\mathcal{T}) + \underbrace{\sum_{t \geq 0} \log p(s_{t+1} | s_t, a_t) - \log Z(\theta)}_{\text{const}(\theta)} \quad (8.7)$$

Максимизация логарифма правдоподобия имеет интересную интерпретацию: нам нужно выдавать большую награду на тех состояниях, которые эксперт посетил (первое слагаемое) и маленькую награду «всюду в остальных местах» (второе слагаемое). Тогда интеграл $\log Z(\theta)$ будет уменьшаться.

Как оптимизировать это правдоподобие? С первым слагаемым всё понятно: награда, моделируемая нейросеткой, дифференцируема по параметрам. Проблема заключается в нормировочном слагаемом.

Нам понадобится следующий фокус. Пусть у нас есть некоторая функция награды с параметрами θ . Пусть $\pi_{[\theta]}^*$ — стратегия, которая оптимально (в терминах Maximum Entropy фреймворка, в рамках предположения (8.2)) оптимизирует вот эту награду, которую мы предлагаем с текущими параметрами θ . Такая стратегия по определению будет в среде генерировать траектории из распределения

$$p(\mathcal{T} | \pi_{[\theta]}^*) := \frac{e^{R_\theta(\mathcal{T})} \prod_{t \geq 0} p(s_{t+1} | s_t, a_t)}{Z(\theta)} \quad (8.8)$$

и это в точности есть правдоподобие траекторий эксперта при текущих значениях параметров θ .

Теорема 87 — Guided Cost Learning: Градиент для оптимизации правдоподобия (8.7) траекторий эксперта по параметрам функции награды θ равен:

$$\mathbb{E}_{\mathcal{T} \sim \pi^*} \nabla_\theta R_\theta(\mathcal{T}) - \mathbb{E}_{\mathcal{T} \sim \pi_{[\theta]}^*} \nabla_\theta R_\theta(\mathcal{T}) \quad (8.9)$$

Доказательство. Рассмотрим градиент логарифма правдоподобия одной траектории:

$$\nabla \log p_\theta(\mathcal{T} | \pi^*) = \nabla R_\theta(\mathcal{T}) - \nabla \log Z(\theta)$$

Мы хотим оптимизировать правдоподобие в среднем по траекториям эксперта, однако нам нужен градиент нормировочной константы (общий для всех траекторий эксперта, поэтому из математического ожидания по ним эту константу можно вынести). Рассмотрим дифференцирование нормировочной константы отдельно:

$$\nabla \log Z(\theta) =$$

$$\begin{aligned}
&= \{\text{градиент логарифма}\} = \frac{1}{Z(\theta)} \nabla Z(\theta) = \\
&= \{\text{определение } Z(\theta) \text{ (8.6)}\} = \frac{1}{Z(\theta)} \int_{\mathcal{T}} \nabla_{\theta} e^{R_{\theta}(\mathcal{T})} \prod_{t \geq 0} p(s_{t+1} | s_t, a_t) d\mathcal{T} = \\
&= \{\text{дифференцируем экспоненту}\} = \frac{1}{Z(\theta)} \int_{\mathcal{T}} e^{R_{\theta}(\mathcal{T})} \prod_{t \geq 0} p(s_{t+1} | s_t, a_t) \nabla_{\theta} R_{\theta}(\mathcal{T}) d\mathcal{T} = \\
&= \{\text{определение } \pi_{[\theta]}^* \text{ (8.8)}\} = \mathbb{E}_{\mathcal{T} \sim p(\mathcal{T} | \pi_{[\theta]}^*)} \nabla R_{\theta}(\mathcal{T})
\end{aligned}$$

■

Отсюда напрашивается такая забавная идея. С первым мат.ожиданием в формуле градиента всё понятно: мы максимизируем награду в тех парах состояние-действие, которые встречались у эксперта. Затем мы берём и при помощи любого алгоритма Maximum Entropy RL оптимизируем нашу текущую награду $r_{\theta}(s, a)$, получая оптимальную относительно нашей текущей функции награды стратегию $\pi_{[\theta]}^*$ (точнее, её некоторое приближение). Отправляем её в реальную среду и собираем траектории $\mathcal{T} \sim \pi_{[\theta]}^*$. И чтобы получить второе слагаемое градиента, говорим, что в состояниях-действиях, которые встретились в траекториях этого «псевдо-эксперта», награду нужно минимизировать. Если наша награда стала настоящей, псевдо-эксперт сойдётся к эксперту, и градиент станет нулевым.

Конечно, мы не будем обучать RL-алгоритм для оптимизации стратегии $\pi_{[\theta]}^*$ до сходимости, и вместо этого будем чередовать шаг оптимизации по параметрам θ функции награды при помощи формулы (8.9) (где второе мат.ожидание оценено при помощи сэмплов из текущей $\pi_{[\theta]}^*$) и шаг оптимизации параметров самой $\pi_{[\theta]}^*$ при помощи RL-алгоритма. В пределе мы потенциально сойдёмся к той функции награды, которой пользовался истинный эксперт, и заодно к стратегии, её максимизирующей.

8.1.4. Generative Adversarial Imitation Learning (GAIL)

Guided Cost Learning со своим чередованием двух шагов оптимизации напоминает игру двух игроков. Оптимизация по параметрам награды говорит уменьшать слагаемое, соответствующее нормировочной константе, а оптимизация стратегии — увеличивать. У этого есть важная интерпретация, которая помогает понять, как на самом деле нужно обучаться с экспертных траекторий.

Сначала мы чуть-чуть перепишем нашу задачу оптимизации по π и по r .

Утверждение 80: Оптимизация функции награды по формуле (8.9) соответствует оптимизации следующего функционала:

$$\mathbb{E}_{\mathcal{T} \sim \pi^*} R(\mathcal{T}) - \max_{\pi} \mathbb{E}_{\mathcal{T} \sim \pi} \sum_{t \geq 0} (r(s_t, a_t) + \mathcal{H}(\pi(\cdot | s_t))) \rightarrow \max_r \quad (8.10)$$

Пояснение. Градиент максимума от функции есть градиент этой функции в точке максимума, а энтропия $\mathcal{H}(\pi(\cdot | s_t))$ не зависит от r , поэтому градиент второго слагаемого совпадает со вторым слагаемым (8.9). ■

Определение 99: *Occupancy measure* для стратегии π будем называть

$$\rho_{\pi}(s, a) := \pi(a | s) d_{\pi}(s), \quad (8.11)$$

где $d_{\pi}(s)$ — частоты посещения состояний (5.10).

По определению, мат.ожидания по траектории с таким обозначением можно записывать как

$$\mathbb{E}_{\mathcal{T} \sim \pi} \sum_{t \geq 0} f(s_t, a_t) = \mathbb{E}_{\rho_{\pi}} f(s, a)$$

Давайте воспользуемся этим обозначением в (8.10). Раскрывая определение мат.ожидания и переписывая оптимизацию по π как минимизацию, получаем такую «игру» (читать — поиск седловой точки):

$$\min_r \max_{\pi} \mathbb{E}_{\rho_{\pi}} r(s, a) - \mathbb{E}_{\rho_{\pi^*}} r(s, a) + \mathbb{E}_{\mathcal{T} \sim \pi} \sum_{t \geq 0} \mathcal{H}(\pi(\cdot | s_t)) \quad (8.12)$$

Сразу сделаем одно наблюдение: по определению occupancy measure однозначно задаёт стратегию π :

Утверждение 81:

$$\pi(a | s) = \frac{\rho_{\pi}(s, a)}{\int_{\mathcal{A}} \rho_{\pi}(s, a) da} \quad (8.13)$$

Значит, можно поиск стратегии интерпретировать как поиск осцилансу measure: действительно, в (8.12) последнее слагаемое — суммарный энтропийный бонус стратегии π — тоже можно переписать в терминах $\rho_\pi(s, a)$. Обозначим его как

$$\tilde{\mathcal{H}}(\rho_\pi) := \mathbb{E}_{\mathcal{T} \sim \pi} \sum_{t \geq 0} \mathcal{H}(\pi(\cdot | s_t)) = \mathbb{E}_{\rho_\pi} - \log \pi(a | s) = \{(8.13)\} = \mathbb{E}_{\rho_\pi} - \log \frac{\rho_\pi(s, a)}{\int_{\mathcal{A}} \rho_\pi(s, a) \, da}$$

Получаем такой взгляд на процесс обучения:

$$\min_r \max_{\rho_\pi} \mathbb{E}_{\rho_\pi} r(s, a) - \mathbb{E}_{\rho_{\pi^*}} r(s, a) + \tilde{\mathcal{H}}(\rho_\pi) \quad (8.14)$$

Глядя на (8.14), видно, что единственная возможная седловая точка — это случай $\rho_\pi \equiv \rho_{\pi^*}$: относительно $r(s, a)$ функционал внутри линеен¹, как только в какой-то паре s, a разница $\rho_\pi - \rho_{\pi^*} \neq 0$, функция награды может начать выдавать бесконечно большие значения соответствующего знака. На энтропийное слагаемое $\tilde{\mathcal{H}}(\rho_\pi)$ можно смотреть как на регуляризатор по частотам посещения пар состояния-действия. Минимизация по ρ_π с точностью до этого энтропийного регуляризатора — это поиск такого $\rho_\pi(s, a)$, который наиболее похож на $\rho_{\pi^*}(s, a)$.

Другими словами: задача обучения по экспертным траекториям заключается не в том, чтобы матчить нашу стратегию со стратегией эксперта, а в том, чтобы заматчить осцилансу measure — частоты посещений пар состояния-действие. Это сильно разные вещи: выбор одних действий в одной области пространства состояний влияет на осцилансу measure в других! Если мы матчим стратегии, то, вероятно, мы просто во всех состояниях пытаемся сделать нашу стратегию похожей на стратегию эксперта. А на самом деле мы хотим с одинаковой силой с экспертом частотой во все состояния попадать! И это значит, что имитационное обучение — это матчинг распределений $p(\mathcal{T} | \pi^*)$ и $p(\mathcal{T} | \pi)$!

Это рассуждение открывает следующую идею: давайте при обучении награды мы будем напрямую пытаться отличать распределения $\rho_{\pi^*}(s, a)$ и $\rho_\pi(s, a)$. Как различать распределения? Вспоминаем генеративно-состязательные сети (GAN). Там дискриминатор неявно учил различать, приходят ли сэмплы из одного класса или из другого, просто решая задачу бинарной классификации.

Но как GAN-ы связаны с обратным обучением с подкреплением, с обучением награды? Оказывается, это одно и то же. Чтобы увидеть это в чистом виде, сделаем следующий важный шаг: мы добавим регуляризатор и для оптимизации по r . Это можно мотивировать тем, что, как мы обсуждали ранее, задача обратного обучения с подкреплением «некорректна» и может иметь много разных решений. Итак, добавим некоторое слагаемое $\psi(r)$, которое будет смотреть на нашу выдаваемую функцию награды и некоторым образом её дополнительно штрафовать:

$$\min_r \max_{\rho_\pi} \psi(r) + \mathbb{E}_{\rho_\pi} r(s, a) - \mathbb{E}_{\rho_{\pi^*}} r(s, a) + \tilde{\mathcal{H}}(\rho_\pi) \quad (8.15)$$

Оказывается, ванильный GAN, различающий сэмплы пар s, a из распределений $p(\mathcal{T} | \pi^*)$ и $p(\mathcal{T} | \pi)$, соответствует просто определённому выбору регуляризатора!

Теорема 88: Выберем в (8.15) следующий регуляризатор:

$$\psi(r) := \mathbb{E}_{\rho_{\pi^*}} r(s, a) + \log(1 - e^{-r(s, a)}), \quad (8.16)$$

где штраф полагается бесконечно большим, если $r(s, a) \leq 0$. Тогда задача (8.15) примет вид:

$$\min_D \max_\pi -\mathbb{E}_{\rho_{\pi^*}} \log(1 - D(s, a)) - \mathbb{E}_{\rho_\pi} \log D(s, a) + \tilde{\mathcal{H}}(\rho_\pi), \quad (8.17)$$

где $D(s, a) \in (0, 1)$.

Доказательство. Подставим в (8.15) выбранный регуляризатор:

$$\begin{aligned} & \min_r \max_\pi \psi(r) + \mathbb{E}_{\rho_\pi} r(s, a) - \mathbb{E}_{\rho_{\pi^*}} r(s, a) + \tilde{\mathcal{H}}(\rho_\pi) = \\ &= \min_r \max_\pi \mathbb{E}_{\rho_{\pi^*}} \log(1 - e^{-r(s, a)}) + \mathbb{E}_{\rho_\pi} r(s, a) + \tilde{\mathcal{H}}(\rho_\pi) \end{aligned}$$

Сделаем замену переменных: вместо оптимизации по $r(s, a) > 0$ будем оптимизировать по $D(s, a)$, где $D(s, a) := e^{-r(s, a)}$ — произвольное число в диапазоне $(0, 1)$. Тогда $r(s, a) = -\log(D(s, a))$.

$$\min_D \max_\pi -\mathbb{E}_{\rho_{\pi^*}} \log(1 - D(s, a)) - \mathbb{E}_{\rho_\pi} \log D(s, a) + \tilde{\mathcal{H}}(\rho_\pi) \quad \blacksquare$$

¹поскольку можно переписать матожидания в следующем виде:

$$\mathbb{E}_{\rho_\pi} r(s, a) - \mathbb{E}_{\rho_{\pi^*}} r(s, a) = \int_S \int_{\mathcal{A}} (\rho_\pi(s, a) - \rho_{\pi^*}(s, a)) r(s, a) \, ds \, da$$

Видно, что это фактически линейный функционал $\langle \rho_\pi - \rho_{\pi^*}, r \rangle$ в пространстве функций $\mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$.

Итак, что мы получили в формуле (8.17): вместо награды будем обучать дискриминатор, решающий задачу бинарной классификации, где пары s, a из ρ_{π^*} образуют класс 1, а пары s, a из ρ_π — класс 0. Действительно, оптимизация (8.17) по D при фиксированной π выглядит так:

$$\mathbb{E}_{\rho_{\pi^*}} \log(1 - D(s, a)) + \mathbb{E}_{\rho_\pi} \log D(s, a) \rightarrow \max_D,$$

то есть мы просто учимся отличать пары s, a , порождённые (встреченные) экспертом от тех, что встречает наша текущая стратегия.

Оптимизация же по π (давайте вернёмся к оптимизации по стратегии) при фиксированном «дискриминаторе» D выглядит так:

$$\mathbb{E}_{\mathcal{T} \sim \pi} \sum_{t \geq 0} [-\log D(s_t, a_t) + \mathcal{H}(\pi(\cdot | s_t))] \rightarrow \max_\pi$$

То есть величина $-\log D(s, a)$ и будет являться наградой за шаг: именно такую кумулятивную дисконтируемую награду (плюс энтропийный бонус) оптимизирует стратегия π , она же «генератор» в этой схеме. Такой генератор учится порождать s, a , которые дискриминатор-награда не отличает от экспертных пар, только в отличие от обычного генератора здесь наши сгенерированные действия влияют на следующие состояния, и поэтому мы генерируем не «отдельные» сэмплы s, a , а целые цепочки траекторий $\mathcal{T} \sim \pi$.

8.1.5. Generative Adversarial Imitation from Observation (GAIfO)

Часто эксперт предоставляет нам траектории, в которых отсутствует информация о совершённых агентом действиях: есть лишь цепочки состояний s_0, s_1, s_2, \dots , которые посетил эксперт. Такая задача называется **имитационным обучением по наблюдениям** (imitation learning from observations).

Пример 117: Допустим, у вас есть покадровые анимации того, как персонаж делает сальто. Вы хотите научить робота с такими же конечностями делать тоже самое. В анимациях понятно, в каких координатах находились все конечности персонажа в каждый момент времени, и можно считать, что робот при выполнении задачи должен «посещать» те же цепочки состояний. Однако в анимации действий нету — вы не знаете, как нужно управлять роботом, чтобы получить ту же траекторию в реальной среде.

Оказывается, идея GAIL очень легко и просто расширяется на такую ситуацию. Мы просто хотим попадать в те же состояния, в которые попадал эксперт, поэтому дискриминатором $D(s) \in (0, 1)$ теперь будем пытаться различать состояния — порождены ли они экспертом $s \sim d_{\pi^*}(s)$ или же нашей текущей стратегией $s \sim d_\pi(s)$.

$$\min_D \max_\pi -\mathbb{E}_{d_{\pi^*}(s)} \log(1 - D(s)) - \mathbb{E}_{d_\pi(s)} \log D(s) + \mathbb{E}_{d_\pi(s)} \mathcal{H}(\pi(\cdot | s))$$

В методе Generative Adversarial Imitation from Observation (GAIfO) предлагается сделать хитрость, и различать не состояния, а пары «состояние-следующее состояние» s, s' . Другими словами, награда полагается зависящей от пары состояний $r(s, s')$, а дискриминатор $D(s, s')$ учится различать именно пары s, s' из экспертных траекторий и из порождаемых траекторий. Формально говоря, вводится альтернативное определение occupancy measure как вероятность встретить ту или иную пару s, s' в траекториях из стратегии π :

$$\nu_\pi(s, s') := d_\pi(s) \int_{\mathcal{A}} p(s' | s, a) \pi(a | s) da$$

Тогда минимаксная задача оптимизации принимает следующий вид:

$$\min_D \max_\pi -\mathbb{E}_{\nu_{\pi^*}} \log(1 - D(s, s')) - \mathbb{E}_{\nu_\pi} \log D(s, s') + \mathbb{E}_{d_\pi(s)} \mathcal{H}(\pi(\cdot | s))$$

§8.2. Внутренняя мотивация

8.2.1. Вспомогательные задачи

В обучении с подкреплением типична ситуация разреженной функции награды, когда агенту редко поступает сигнал от среды. Например, в худшем случае функция награды представляет собой константу, имеющую смысл «штрафа за потерю времени», а в конце эпизода нам приходит условно $+1$, если задача была успешно решена.

Определение 100: Задачей *поиска* будем называть задачу со следующей функцией награды:

$$r(s, a) = \begin{cases} +1 & s \in \mathcal{S}^+ \\ \text{const} & s \notin \mathcal{S}^+ \end{cases} \quad (8.18)$$

где \mathcal{S}^+ — множество терминальных состояний, $\text{const} \leq 0$ — штраф за потерю времени.

Задача поиска сложна тем, что сигнала от среды нет. Пока мы не решим задачу, оптимизируемый функционал представляет собой плато, а наши RL алгоритмы, как и любые методы оптимизации, работают за счёт разницы в сигнале.

Что в таких ситуациях делать? Первая возможность — учить модель динамики среды, если она неизвестна. Найти сигнал от среды это скорее всего не поможет, поскольку экспоненциальный перебор всевозможных будущих траекторий не сильно лучше случайного блуждания в среде.

Мы придумаем себе другую, вспомогательную задачу, которую мы сможем решать в *self-supervised* режиме — режиме, не требующем никакого сигнала от среды, никакой «разметки».

Определение 101: Для данной среды $(\mathcal{S}, \mathcal{A}, \mathcal{P})$ *вспомогательной задачей* называется задача обучения с подкреплением для MDP $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r^{\text{intr}})$, где r^{intr} — *внутренняя мотивация* (intrinsic motivation) или *внутренняя награда* (intrinsic reward function), определяемая самим агентом.

Слова «определенная самим агентом» означает, что эта функция нам не дана. Исходная задача, которую мы хотим решить, состоит в оптимизации в данной среде некоторой *внешней функцией награды* (extrinsic reward function) r^{extr} , также называемой *внешней мотивацией* (extrinsic motivation). Однако, если эта функция награды, например, всегда константна, как в задаче поиска (8.18), то нам необходимо откуда-то взять какой-то другой обучающий сигнал. Этот сигнал нам придётся придумать «самим себе», при помощи нового модуля в обучающейся системе.

Какой обучающий сигнал мы хотим получить? Во-первых, плотный, чтобы было, на чём обучаться базовому алгоритму. Во-вторых, осмысленный: награждающий за развитие каких-то «полезных» в среде навыков, которые могут пригодиться для взаимодействия, условно, вне зависимости от того, какой на самом деле окажется та внешне мотивированная задача, которую агент призван решать.

Пример 118: Идея, можно сказать, вдохновлена подобной «внутренней наградой» человека, поощряющей такое поведение, как игра, любопытство, стремление к познанию. В ходе игрового поведения, человек улучшает своё представление о том, как работает мир вокруг него, по каким законам он устроен и как он своими действиями может влиять на будущее состояние мира и вызывать те или иные явления. Такое более интеллектуальное «самообучение» не только ускоряет поиск сигнала от среды, но и позволяет агенту выработать широкий набор умений, которые скорее всего окажутся полезны для достижения заданной внешней наградой цели, какой бы она ни оказалась.

Придумать в общем случае такой сигнал непросто, и могут возникать ситуации, когда внутренняя мотивация «промотивирует» агента «залипнуть» в какой-то области среды.

Определение 102: Проблемой *прокрастинации* (procrastination) называется ситуация, когда в некоторой области в среде внутренняя мотивация выдаёт высокий (и не снижающийся с течением обучения) сигнал, перебивающий остальные мотивации агента.

8.2.2. Совмещение мотиваций

По умолчанию всегда считается, что внешняя и внутренняя мотивация складываются:

$$r(s, a) := r^{\text{extr}}(s, a) + \alpha r^{\text{intr}}(s, a), \quad (8.19)$$

где α — масштабирующий гиперпараметр. Для простоты далее будем считать $\alpha = 1$.

Заданная так награду можно оптимизировать любым «базовым» алгоритмом RL, ничего не подозревающим о разложении. Но понятно, что агенту доступно каждое слагаемое по отдельности (коли внешняя награда выдаётся средой, а внутренняя генерируется внутри самого алгоритма); как мы можем это использовать?

Пусть оценочные функции с индексом **extr** соответствуют оценочным функциям для внешней мотивации, с индексом **intr** — внутренней мотивации, без индекса — суммарной мотивации. Тогда:

Утверждение 82:

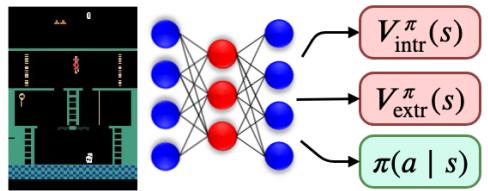
$$\begin{aligned} Q^\pi(s, a) &= Q_{\text{extr}}^\pi(s, a) + Q_{\text{intr}}^\pi(s, a) \\ V^\pi(s) &= V_{\text{extr}}^\pi(s) + V_{\text{intr}}^\pi(s) \end{aligned} \quad (8.20)$$

Доказательство. По определению. ■

Итак, если в алгоритме учится оценочная функция V^π или Q^π , то можно учить оценочные функции каждого слагаемого в награде по отдельности, например, в виде отдельной головы для каждой мотивации.



Этим фактом можно также пользоваться в ситуациях, когда награда представлена в виде суммы нескольких слагаемых (очень типичная ситуация), и агенту доступно полное разложение на эти слагаемые. Это позволит по ходу оптимизационного процесса «включать» и «отключать» мотивации, что бывает очень удобно.



Трюк можно применять даже в value-based методах, где мы учим Q^* , поскольку тот процесс можно интерпретировать как оценивание (обучение Q^π) текущей стратегии $\pi = \arg\max_a Q(s, a)$, хотя формально для истинной оптимальной оценочной функции Q^* разложение неверно:

Утверждение 83: В общем случае аналогичное разложение для Q^*, V^* неверно.

Доказательство. Максимум суммы может быть меньше суммы максимумов. Действительно, пусть первое действие даёт внешнюю +1, а второе действие даёт внутреннюю +1 (иначе по нулям); тогда оптимальные оценочные функции равны $V_{\text{intr}}^*(s) = V_{\text{extr}}^*(s) = +1$, но оптимальная V-функция для суммы наград равна не 2, а 1, поскольку выбрать одновременно два действия нельзя и между мотивациями придётся выбирать. ■

В Distributional-подходе при использовании этой идеи придётся предполагать независимость внутренней и внешней мотивации (что практически всегда неверно).

Утверждение 84: В общем случае аналогичное разложение для Z^π неверно.

Доказательство. Это связано с тем, что внутренняя и внешняя мотивация могут быть скоррелированы, и этой информации при доступе к отдельным $Z_{\text{extr}}^\pi(s, a)$ и $Z_{\text{intr}}^\pi(s, a)$ у нас нет. Действительно: пусть известно, что для некоторых s, a с вероятностью 0.5 $Z_{\text{extr}}^\pi(s, a)$ принимает значение +1, а иначе 0, и тоже самое для $Z_{\text{intr}}^\pi(s, a)$. Мы могли бы сказать, что распределение суммы $Z_{\text{extr}}^\pi(s, a) + Z_{\text{intr}}^\pi(s, a)$ имеет такой вид: с вероятностью 0.25 мы получим +2, с вероятностью 0.5 получим +1, и с вероятностью 0.25 не получим ничего. Однако, тогда мы предполагаем независимость этих случайных величин, а это может быть неправдой: например, вдруг мы получаем внешнюю +1 тогда и только тогда, когда получаем внутреннюю +1. Тогда $Z^\pi(s, a)$ суммы мотиваций есть на самом деле +2 с вероятностью 0.5 и +0 иначе. ■

Утверждение 85: В общем случае аналогичное разложение для Z^* неверно.

Разложение также позволяет использовать разные коэффициенты дисконтирования для разных мотиваций. Допустим, агент учит отдельно оценочную функцию будущей внутренней мотивации V^{intr} и будущей внешней мотивации V^{extr} . Тогда можно использовать разное дисконтирование γ^{intr} и γ^{extr} при их обучении. Для примера рассмотрим одношаговые таргеты:

$$y^{\text{extr}} := r + \gamma^{\text{extr}} V^{\text{extr}}(s')$$

$$y^{\text{intr}} := r + \gamma^{\text{intr}} V^{\text{intr}}(s')$$

Разложение также позволяет во внутренней мотивации игнорировать понятия эпизодов: это означает, что агент рассматривает весь процесс решения вспомогательной задачи как один большой никогда не заканчивающийся эпизод, не используя флаги `done` из среды при обучении V^{intr} . Такой агент учитывает, что если он окажется в терминальном состоянии, то произойдёт сброс среды, и он окажется в стартовом состоянии $s_0 \sim p(s_0)$, для которого значение $V^{\text{intr}}(s)$ может быть высоким. Это может мотивировать агента прерывать текущий «неудачный» эпизод ради того, чтобы начать новый, что иногда может быть полезно.

Пример 119 — Агент в яме: В среде существует некоторая труднодоступная область, которую агент внутренне мотивирован посетить. Агент предпринимает попытку добраться до области, но падает в яму. Оценочная функция для внешней мотивации знает, что из ямы уже не выбраться, и оценивает действие «закончить эпизод» как негативное («смерть»), остальные действия как безрезультатные (+0). Оценочной функции для внутренней мотивации, допустим, известно, что из ямы уже не выбраться, и, в случае, если она обучается эпизодично, оценивает любые действия как безрезультатные (+0). Если же оценочная функция игнорирует понятие эпизодов, агент знает, что он может произвести сброс среды и, в частности, попробовать ещё раз добраться до труднодоступной области. Это может промотивировать агента не сидеть в яме, оттягивая негативный, но неизбежный эффект смерти, а приступить к следующему эпизоду обучения.

8.2.3. Exploration Bonuses

Как строить r^{intr} ? Что должен делать агент, который попал в среду, и не получает никакого внешнего сигнала? На этот вопрос можно ответить по-разному.

Пример 120 — Минимизация хаоса (chaos minimization): Нужно искать наиболее «безопасные», стабильные области пространства состояний, где будущее наиболее предсказуемо, «избегать сюрпризов». Интуицией в такой вспомогательной задаче является идея о том, что многие изобретения человечества были созданы для защиты от сюрпризов, которые, зачастую, неприятны. Существуют среды, например, тетрис, где задача «минимизации хаоса» коррелирует с задачей самой игры; агент, решающий такую вспомогательную задачу без доступа к внешней функции награды, «неявно» решает исходную задачу.

Мы далее рассмотрим другой ответ — заниматься исследованием. Отчасти эта задача полностью противоположна минимизации хаоса: однако задачи не противоречат друг другу, ведь чтобы найти самую «спокойную» область среды и научиться до неё добираться, агенту потребуется заняться исследованием окружения и поиском этих самых стабильных областей. В этом смысле, задача исследования, вероятно, является наиболее универсальной вспомогательной задачей, которую агент может себе поставить в среде.

Мы постоянно встречались с дилеммой исследования-использования по ходу пьесы, однако теперь, когда оптимизировать внешний сигнал у нас не получается ввиду его отсутствия, «использовать» нам абсолютно нечего, и дилемма не стоит. Итак, считаем, что нам дана среда и есть задача «заисследовать её». Как формализовать такую задачу в терминах награды?

Понятно, что случайная стратегия, которой мы часто пользовались до этого для «исследований», является теоретическим решением (например, с точки зрения теоремы 28). Но интуитивно, куда более оптимальным поведением является некий интеллектуальный перебор состояний в среде.



Мы уже встречались с *исследовательскими бонусами* (exploration bonus) в контексте UCB-бандитов (раздел 7.1.4): там мы добавляли к нашей оценке Q-функции некоторое слагаемое, имевшее смысл «награды за то, что действие редко пробовалось в прошлом». Наша внутренняя мотивация тоже есть такая добавка, только теперь она должна оценивать новизну посещаемых областей в среде.

Попробуем исходить из схожих соображений: будем награждать агента за посещения тех состояний, в которых он был редко. Мы можем это сделать двумя способами.

Определение 103: Пусть $h(s) : \mathcal{S} \rightarrow \{0, 1 \dots N\}$ — некоторая хэш-функция состояний, называемая *оракулом* (oracle), и $n(i)$ — счётчик, сколько раз за время всего обучения нам встретились состояния с хэшем i . Тогда

$$r_{\text{intr}}(s, a) := \frac{1}{n(h(s))}$$

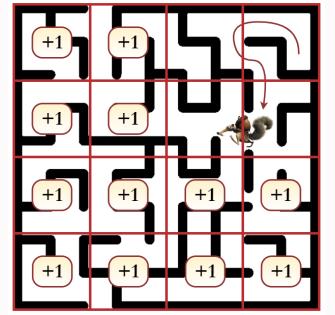
называется *нестационарным* исследовательским бонусом; награда

$$r_{\text{intr}}(s_t, a_t) := \mathbb{I}[\forall t' < t: s_t \neq s_{t'}],$$

то есть награждение $+1$, если мы попали в состояние, хэш для которого $h(s_t)$ не встречался до этого в течение данного эпизода, называется *эпизодичным* исследовательским бонусом.

Нестационарные исследовательские бонусы затухают с ходом обучения; в пределе мы, надеемся, посетим все состояния достаточное число раз, внутренняя мотивация затухнет и мы переключимся на внешнюю мотивацию. Плохо это тем, что такая мотивация нарушает стационарность формализма MDP, так как с ходом обучения меняются счётчики посещения. Это довольно типично, что внутренняя мотивация нестационарна: модуль внутренней мотивации принципиально есть часть обучающейся системы, и он тоже постепенно «обучается», следовательно, меняется. Для нас это значит, что нужно будет использовать on-policy алгоритмы для обучения на такой сигнал.

Эпизодические бонусы, конечно же, можно считать модификацией функции награды, и поэтому подобные оракулы можно считать «ручными эвристиками». Агент в том числе по итогам обучения научится в ходе одного эпизода «бегать по всему MDP». Это, однако, вполне может быть полезно в каких-нибудь лабиринтах или задачах, где агенту нужно что-то где-то найти в течение самой игры. Проблема эпизодических бонусов в том, что они формально нарушают предположение о полной наблюдаемости пространства состояний: функция награды зависит от всей прошлой истории посещения состояний в течение эпизода, и нам по-хорошему нужно переходить в формализм PoMDP.



Пример 121: В табличных MDP, где $|\mathcal{S}| < +\infty$, хэш-функция по сути не нужна: $h(s) = s$. В ряде сред подобный оракул можно придумать эвристически; например, если у агента есть понятие «координат», можно разделить пространство сеточкой, то есть поделив его на условные блоки, и награждать агента за «посещение большого числа блоков».

8.2.4. Дистилляция случайной сетки (RND)

Понятно, что в общем случае хэш-функцию придумать сложно, и нам нужен какой-то более универсальный способ оценки новизны. Мы воспользуемся трюком из машинного обучения: известно, что если некоторая модель обучалась решать задачу регрессии, то на входных данных, не похожих на примеры из обучающей выборки, ошибка её прогнозов будет больше. Отличие предсказания модели от истинного значения целевой переменной может быть использовано как приближение *оценки новизны* (novelty estimation) или *аномальности* данных.

Давайте возьмём какую-нибудь случайную задачу регрессии с состояниями в качестве входов. Нам важно лишь, что значение целевой переменной должно определяться по входному состоянию детерминировано и стационарно: ведь если таргет не детерминирован, то ошибка предсказывающей модели для состояния s будет (если, например, модель обучается на минимизацию MSE) в среднем равна дисперсии. В такой ситуации сигнал внутренней мотивации будет выше в тех областях пространства состояний, где дисперсия целевой переменной выше. Нестационарность, очевидно, нарушает идею подхода, поскольку ошибка модели будет связана с изменением целевой переменной, а не новизной входного состояния.

Пусть $\phi: \mathcal{S} \rightarrow \mathbb{R}^d$ — некоторая функция, строящая эмбеддинги для состояний; например, *случайная сеть* (random network) — нейросеть со случайно инициализированными весами, которые не обучаются и никак не изменяются. Пусть $f: \mathcal{S} \rightarrow \mathbb{R}^d$ учится предсказывать выход ϕ , используя в качестве обучающей выборки её значения на встречавшихся в ходе обучения состояниях.

Определение 104: Задача обучения одной нейросети f на входах-выходах другой (заданной, фиксированной) нейросети ϕ

$$\mathbb{E}_s \|f(s) - \phi(s)\|_2^2 \rightarrow \min_f,$$

где мат.ожидание \mathbb{E}_s берётся по произвольному буферу, называется *дистилляцией* (distillation).

Коли ошибка выше там, где новое состояние, мы можем использовать это значение в качестве обучающего сигнала:

$$r^{\text{intr}}(s) := \|f(s) - \phi(s)\|_2^2$$

Интуиция понятна: если мы «впервые» увидели состояние s , модель f ещё ни разу не видела, какой эмбеддинг выдаёт на нём наша случайная сеть ϕ , и поэтому скорее всего ошибётся. Такой сигнал будет мотивировать агента отправиться в ту область среды, где обучающаяся нейросеть плохо предсказывает выход случайно инициализированной нейросети.



Здесь и в аналогичных местах далее использование MSE не принципиально (можно использовать любую функцию потерь для регрессии). В частности, можно одну метрику использовать для функции потерь, и другую — для расчёта внутренней мотивации.



Авторы также предложили нормировать сигнал внутренней мотивации на его бегущее среднее отклонение, то есть «в среднем» выдавать некоторую константу. Это существенно упрощает масштабирование внутренней мотивации относительно внешней, но тогда «исследование» не будет естественно затухать с ходом обучения.

8.2.5. Любопытство

Определение 105: *Любопытством* (curiosity) называется ошибка модели мира агента.

Ошибка свидетельствует о том, что агент не всё знает о той области пространства состояний, в которой был произведён неверный прогноз. Использование любопытства как внутренней мотивации приводит к тому, что агент стремится в те области среды, которые ему «непонятны», в частности те, которые для агента новы, и «дообучить» своё представление о мире. Зачастую выбор действий, максимизирующих ошибку модели мира приводит к **возникновению игрового поведения** (emergence of playing behavior), когда агент «играет» с доступными для взаимодействия предметами.

Любопытство и построение внутренней мотивации на основе новизны состояний немного различаются. То, что состояние ново, не означает, что оно «незаисследовано», что мы ничего о нём не знаем; мы вполне можем обобщиться за счёт прошлого опыта и спокойно ориентироваться даже в том состоянии, которое технически увидели впервые.

Пример 122 — Бесконечные двери: В ряд стоит бесконечное количество одинаковых дверей, за каждой из которых ничего нет. Агенту доступен на вход, помимо прочего, номер двери. Сигнал, построенный на основе оценки новизны состояний, будет поощрять обнаружение новых дверей, поскольку их номер «нов» по сравнению со старыми. Любопытство же через некоторое время научится предсказывать, что при движении вдоль ряда агенту будут встречаться точно такие же двери, как и раньше, и что за дверьми ничего нет; сигнал затухнет. При этом, любопытство среагирует на любое изменение в описании двери (если все двери были, например, красными, а тут вдруг бац, и дверь синяя), или если за очередной дверью окажется что-то непредсказуемое.



На самом деле, разница достаточно условна: предполагается, что модель предсказания будущего способна «обобщаться» на новые состояния, что, в целом, может оказаться верным и для модели оценки новизны. Так, в приведённом примере модель оценки новизны также может перестать оценивать увеличение номера двери как «новые» состояния, и тогда поведение этих двух видов внутренней мотивации станет схожим.

Пусть внутри агента строится модель прямой динамики:

$$\mathbb{E}_{s,a,s'} \|f(s, a) - s'\|_2^2 \rightarrow \min_f$$

Тогда во время очередного обучающего эпизода ошибка модели f на каждом шаге может рассматриваться как любопытство и задавать внутреннюю мотивацию агента:

$$r^{\text{intr}}(s, a, s') := \|f(s, a) - s'\|_2^2 \quad (8.21)$$

Такой сигнал будет мотивировать агента искать не новые области, а те, в которых он не понимает, как работает среда. В некоторых средах такой сигнал, однако, может привести к прокрастинации.

Определение 106: *Шумным телевизором* (noisy TV) в среде называются принципиально непредсказуемые в силу стохастичности функции переходов явления.

Пример 123 — Шумный телевизор: В среде стоит сломавшийся телевизор, демонстрирующий гауссовский шум. На каждом шаге шум сэмплируется заново. Предсказать следующее значение шума по предыдущему в силу независимости сэмплов невозможно. В рамках концепции любопытства, агент мотивирован найти подобный «шумный телевизор» в среде (что может быть непросто) и получать наслаждение от непредсказуемости своих будущих наблюдений.



Шумные телевизоры по определению нерелевантны: они не имеют отношения к истинной задаче, и «отвлекают» агента, когда ошибка модели мира принципиально не снижаема. Агент, мотивированный любопытством находить в среде шумные телевизоры — это типичный пример прокрастинации.

Пример 124: «Шумные телевизоры» могут принимать самые разные формы: например, у агента в некоторой области пространства состояний сломались сенсоры, и зашумляются гауссовским шумом. Или, например, среда отправляет агента в одну комнату или в другую с вероятностью 0.5; агент не может предсказать, куда именно его перекинет. В видеоиграх могут встречаться всякие декоративные рандомизированные спецэффекты, не имеющие отношения к самой задаче.

С одной стороны, проблема проистекает из того, что мы приближаем стохастическую динамику среды детерминированной моделью. Если стохастика среды существенно влияет на будущее агента и его путь к целевым состояниям, знание и понимание вероятностей различных исходов и их состав, очевидно, является ценным знанием об окружающем мире. Но если задуматься, строить генеративную модель сложного мира — по сути, построить модель вселенной — очень сложная задача, и шумные телевизоры скорее всего задаются сложным распределением, которое в принципе будет плохо поддаваться изучению. В совокупности с нерелевантностью шумных телевизоров, смысла заниматься этим нет, и хочется как-то избежать попыток предсказывать будущие состояния шумных телевизоров вовсе².

Пример 125: Человек не пытается моделировать все без исключения окружающие сложные процессы, например, предсказывать поведение (траектории) всех наблюдаемых капель дождя или опадающих листьев. Вероятно, именно поэтому на них так легко «залипнуть».

8.2.6. Модель обратной динамики

Определение 107: Модель, аппроксимирующая $p(a | s, s')$, называется *моделью обратной динамики* (inverse dynamics model).

Такая модель тоже представляет собой пример модели мира. Агент, делая шаг в среде, может проверить, а правда ли он может по текущему состоянию s' и предыдущему состоянию s восстановить только что выбранное им действие. Отрицательный ответ может соответствовать ситуации, когда агент открыл новые явления в среде, попал в новую область пространства состояний. В таком случае, ошибка модели обратной динамики может быть использована в качестве любопытства.

Задача предсказывать действие по состоянию и следующему состоянию — самая обычная задача классификации для дискретных пространств действий и задача регрессии для непрерывных пространств. Преимуществом модели обратной динамики является то, что построение модели $\mathcal{S} \times \mathcal{S} \rightarrow \mathcal{A}$ сопоставимо по сложности с моделями, использующимися внутри основного RL-алгоритма: не требуется построение моделей, выдающих объекты из \mathcal{S} .

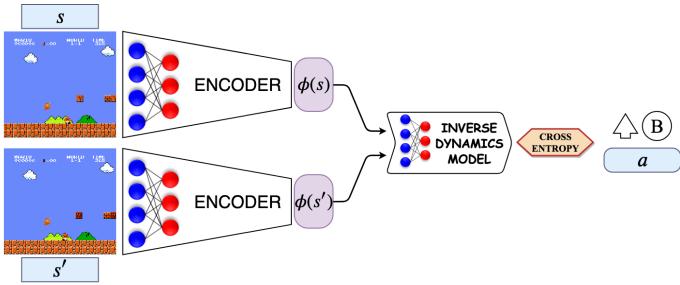
Для модели обратной динамики проблема «шумных телевизоров» в среде заменяется симметричной проблемой в пространстве действий \mathcal{A} . Из формализма MDP по формуле Байеса следует:

$$p(a | s, s') = \frac{p(s' | s, a)\pi(a | s)}{\int_{\mathcal{A}} p(s' | s, a)\pi(a | s) da} \quad (8.22)$$

Из формулы (8.22) понятно, что, во-первых, искомое распределение существенно зависит от используемой для порождения выборки стратегии π . Во-вторых, во многих средах пространство действий для некоторых областей \mathcal{S} может содержать принципиально неразличимые действия. Например, если пространство действий непрерывно, то мы скорее всего опять будем использовать детерминированную модель, а в дискретных пространствах действий, когда мы решаем задачу классификации, использование ошибки — $\log q(a | s, s')$, где q — наше приближение (8.22), вовсе не будет выдавать нулевую ошибку даже если мы выучили (8.22) идеально.

Пример 126: Например, если в данном состоянии s два действия в принципе эквивалентны, то есть $p(s' | s, a_1) \equiv p(s' | s, a_2)$, то агент будет мотивирован, находясь в s , совершать действия a_1, a_2 , поскольку его модель обратной динамики не сможет их различать, и классификатор будет размазывать вероятности между ними. Это типичная ситуация в видеоиграх, когда часто несколько комбинаций кнопок (считающиеся разными действиями) эквивалентны.

² поэтому считается, что сигналом любопытства должна быть не столько ошибка модели, сколько её изменение после дообучения. Иными словами, если модель продолжает ошибаться, «понимать явление» в среде не удаётся, и тогда необходимо почувствовать разочарование (убрать мотивационный сигнал) и бросить силы на исследование других областей среды. Тогда, если возле шумного телевизора ошибка модели мира не падает, агент перестанет на него отвлекаться. Однако построить масштабируемый алгоритм, эффективно замеряющий, как много информации о модели мира предоставил очередной переход (s, a, s') , чтобы превратить это значение во внутреннюю награду, довольно сложно, и далее мы разберём модуль любопытства, который основан на определении через абсолютное значение ошибки модели.



Поэтому модель обратной динамики для любопытства обычно не используют. У неё, однако, есть другое, куда более интересное и полезное применение. Рассмотрим модель обратной динамики с «самской» архитектурой; для непрерывных пространств действий задача выглядит так:

$$\mathbb{E}_{s,a,s'} \|g(\phi(s), \phi(s')) - a\|_2^2 \rightarrow \min_{g,\phi}, \quad (8.23)$$

а для классификации как

$$\mathbb{E}_{s,a,s'} \log g(a | \phi(s), \phi(s')) \rightarrow \max_{g,\phi}, \quad (8.24)$$

где $\phi(s) : \mathcal{S} \rightarrow \mathbb{R}^d$ строит некоторое латентное описание состояний, а затем функция g пытается восстановить действие a , случившиеся между двумя состояниями, по их латентным описаниям. Что можно сказать о том представлении состояний, которое выучит функция $\phi(s)$? Можно ожидать, что она будет оставлять от состояний только информацию, необходимую для предсказания промежуточных действий. Скорее всего, эта информация будет соответствовать описанию только тех объектов в среде, с которыми агент может непосредственно взаимодействовать, по изменению состояний которых можно судить о том, какое действие совершил агент. Таким образом, $\phi(s)$ будет выдавать описание состояний, очищенное от нерелевантных для агента явлений.

Определение 108: Латентные представления состояний, которые учат функция $\phi(s)$ из задачи (8.23) или (8.24), называются **контролируемым состоянием** (controllable state), а сама функция $\phi(s)$ — **фильтром** (filter).

Пример 127: Представьте, что в следующем кадре видеоигры с вероятностью 0.1 моргает декоративное солнышко. Если его моргание никак не связано с агентом и действиями, которые агент выбирает, то это типичный шумный телевизор. Модель прямой динамики пыталась бы безуспешно предсказывать его моргание, из-за чего в предсказаниях будущих состояний всегда была бы некоторая ошибка. Модели же обратной динамики не нужно ничего знать о солнышке, чтобы предсказывать промежуточные действия, поэтому модель быстро научится игнорировать солнышко во входных данных; в описаниях $\phi(s)$ информации о солнышке не будет. Таким образом, фильтр очистит описание состояний от этого лишнего элемента.

А что, если солнышко всё-таки как-то связано с агентом? Тогда в зависимости от состояния солнца — назовём его c , это часть информации внутри s — при каком-то действии a функция переходов отличается от другого действия \hat{a} : $p(s' | c, a) \neq p(s' | c, \hat{a})$. Тогда классификатору или регрессору в модели обратной динамики будет необходимо оставить информацию c внутри латентного представления $\phi(s)$, чтобы снизить ошибку при различении a и \hat{a} .

8.2.7. Внутренний модуль любопытства (ICM)

Свойство модели обратной динамики наводит на мысль, как можно построить защиту от шумных телевизоров. Для этого мы возьмём описание состояний и «почистим» их от шумных телевизоров при помощи фильтра $\phi(s)$, который переведёт описание состояния в некоторое латентное пространство, хранящее лишь частичную информацию о входе. А дальше модель прямой динамики построим в таком «отфильтрованном» латентном представлении:

$$\mathbb{E}_{s,a,s'} \|f(\phi(s), a) - \phi(s')\|_2^2 \rightarrow \min_f \quad (8.25)$$

Авторы алгоритма ICM предлагают обе модели обучать совместно, то есть в частности оптимизировать (8.25) по параметрам фильтра ϕ . Итого модель ICM выглядит следующим образом (рассмотрим для примера случай непрерывных действий):

$$\mathbb{E}_{s,a,s'} [\|g(\phi(s), \phi(s')) - a\|_2^2 + \alpha \|f(\phi(s), a) - \phi(s')\|_2^2] \rightarrow \min_{f,g,\phi}$$

где s, a, s' — произвольные тройки из любого буфера, α — масштабирующий гиперпараметр.

В этой задаче оптимизации градиенты нигде не останавливаются: это значит, что от фильтра ϕ требуется построение таких представлений, в рамках которых модели прямой динамики f «проще всего» предсказывать будущее. Понятно, что, если бы первого слагаемого (задачи обратной динамики) в таком функционале не было бы, оптимальным решением было бы $\phi(s) = \text{const}$. Можно считать, что модель прямой динамики выступает регуляризатором для модели обратной динамики: представления $\phi(s)$ одновременно должны содержать достаточно информации для предсказания выбранных действий и при этом быть максимально простыми.

В качестве внутренней мотивации используется только ошибка модели прямой динамики:

$$r^{\text{intr}}(s, a, s') := \|f(\phi(s), a) - \phi(s')\|_2^2$$

Решает ли ICM проблему шумных телевизоров? Теоретически, можно рассчитывать на то, что модель обратной динамики отфильтрует те шумные телевизоры, с которыми агент не может взаимодействовать. Но если у агента есть «пульт» от телевизора, если случайное непредсказуемое явление можно «затриггерить» определёнными действиями, то ICM всё равно приведёт к прокрастинации.

Пример 128 — Управляемый шумный телевизор: В среде присутствует телевизор, демонстрирующий изображение (возможно, осмысленное!) из разнообразного бесконечного набора. У агента есть пульт от телевизора, то есть специальное действие \hat{a} , при выборе которого изображение на телевизоре сменяется на случайное из набора. По факту смены изображения между s в s' модель обратной динамики может сделать однозначный вывод о том, что между состояниями было выбрано именно действие \hat{a} , следовательно в представлении $\phi(s)$ останется информация о содержимом телевизора. При этом, в силу случайности выбора изображения из набора, предсказать контент телевизора по предыдущему изображению и факту нажатия на пульт невозможно. Следовательно, модель прямой динамики в ICM будет ошибаться на тройках, содержащих \hat{a} , и агент будет мотивирован бесконечно выбирать действие \hat{a} .

Шумные телевизоры, с которыми агент может взаимодействовать, представляют собой любые стохастичные явления в среде, которые агент может «запускать» своими действиями. Это не столько проблема самого алгоритма ICM, сколько концептуальная проблема любопытства. Наличие у агента возможности взаимодействовать с шумным телевизором потенциально означает, что некоторая комбинация действий приводит к решению искомой задачи в среде (высокой внешней награде), а сам шумный телевизор — связан с задачей (в примере 128 с управляемым шумным телевизором задача гипотетически могла заключаться в поиске определённого изображения из набора, или совершении определённой комбинации действий при определённых условиях на текущее отображаемое изображение). При этом любопытство, как и любая внутренняя мотивация, вводится из соображений, что априорных знаний об истинной задаче агента не дано, и произвольные явления в среде должны быть исследованы.



В большинстве традиционных задач для тестирования алгоритмов RL такие управляемые телевизоры обычно отсутствуют; обычно, для того, чтобы «сломать» ICM, нужно строить специальную среду. Но понятно, что чем сложнее и реалистичнее рассматриваются задачи, тем больше вероятность натолкнуться на такое явление.

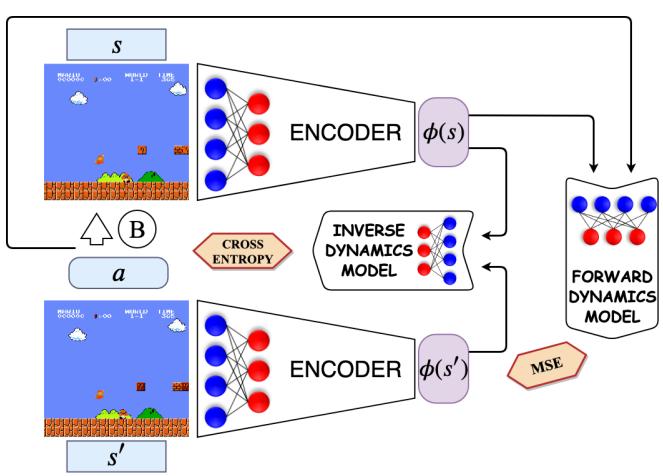
§8.3. Multi-task RL

8.3.1. Многозадачность

Понятно, что в одной и той же среде мы можем решать много разных задач. Для одной и той же среды каждая задача — в рамках нашего формализма MDP — может быть задана при помощи своей функции награды.

Определение 109: Для данной среды $(\mathcal{S}, \mathcal{A}, \mathcal{P})$ набором подзадач будем называть множество \mathcal{G} , элементы $g \in \mathcal{G}$ которого имеют уникальное признаковое описание, для которых задана функция награды $r^g(s, a)$ (и, возможно, свой набор терминальных состояний \mathcal{S}_g^+).

Пусть наша цель — научиться в среде решать не одну задачу, а много. Сформулируем задачу так. Нам дан набор подзадач и некоторое распределение над задачами $p(g)$. Будем считать, что в начале эпизода нам сэмплируется случайная задача, и дальше в этом эпизоде мы должны решать её. Оптимизировать будем среднюю



награду «по задачам»:

$$\mathbb{E}_{g \sim p(g)} \mathbb{E}_{\pi \sim \pi} \sum_{t \geq 0} \gamma^t r_t^g \rightarrow \max_{\pi} \quad (8.26)$$

Пример 129: Типичный пример \mathbf{g} — координаты целевой точки, до которой нужно добраться, или в которую нужно что-то переместить.

Пример 130: Например, если вы учите робота ходить, то вы можете в качестве \mathbf{g} брать направления движения. Тогда функция награды будет поощрять робота за, например, перемещение по вектору \mathbf{g} , а сам этот вектор будет генерироваться случайно в начале эпизода. Решая задачу (8.26), робот научится ходить во все стороны, и целевое направление можно будет подавать ему на вход!

Теорема 89: Задачу (8.26) можно свести к обычному MDP, добавив в состояния информацию о текущей решаемой задаче.

Доказательство. Действительно, пусть $\hat{\mathcal{S}} := \mathcal{S} \times \mathcal{G}$. Пусть начальное состояние будет определяться стохастично как (s_0, g) , где s_0 — начальное состояние нашей исходной среды (без ограничения общности считаем его фиксированным), $g \sim p(g)$. В функцию переходов $\hat{\mathcal{P}}$ просто добавим информацию о текущей решаемой задачи:

$$p(s' | s, g, a) := p(s' | s, a), \quad g' := g \quad (8.27)$$

Функцию награды определим как $\hat{r}(s, g, a) := r^g(s, a)$. Наконец, терминальными определим все пары $\{(s, g) \mid g \in \mathcal{S}_g^+\}$. Тогда оптимизируемый функционал для MDP $(\hat{\mathcal{S}}, \mathcal{A}, \hat{\mathcal{P}}, \hat{r})$ в точности совпадает с (8.26). ■

Итак, придуманная постановка задачи просто переводит нас в другое MDP: формально «ничего не изменилось». Но мы поняли важную вещь: решение многих задач в среде «параллельно» эквивалентно хранению в состояниях информации о текущей решаемой задачи. Коли так, и если у задач есть признаковое описание (это было важное предположение), то мы можем искать оценочные функции всего набора совместно: например, моделировать $Q^*(s, g, a)$, где описание \mathbf{g} решаемой задачи поступает модели на вход вместе с s . Это в точности эквивалентно тому, что \mathbf{g} хранилось бы как часть описание состояний; тогда оно тоже бы поступало на вход оценочным функциям.

Определение 110: Модель для аппроксимации оценочных функций сразу для набора подзадач называется *универсальной оценочной функцией* (universal value function). Аналогично можно рассматривать «*универсальную стратегию*» $\pi(a | s, g)$.

8.3.2. Мета-контроллеры

Обсудим, как можно формализм мультизадачности применить в обычной задаче RL. Часто в алгоритме у нас встречаются важные гиперпараметры, которые трудно заранее подобрать. Выделим два примера: коэффициент дисконтирования γ и коэффициент масштабирования внутренней мотивации α из уравнения (8.19). Вместо того, чтобы подбирать эти параметры «по сеточке», мы можем запустить multi-task RL, где \mathbf{g} — пара γ, α . Далее, вместо оценочных функций, например, $Q^*(s, a)$, будем учить универсальные оценочные функции $Q^*(s, a, \gamma, \alpha)$. Такая функция будет выдавать для данной пары s, a будущую награду с учётом поданного на вход дисконтирования γ и масштаба бонуса внутренней мотивации α . Такое обобщение хорошо и само по себе, поскольку снабдит модель оценочной функции вспомогательными градиентами.

Мета-контроллеры могут помочь автоматически определить, для каких именно значений гиперпараметров γ, α алгоритму легче всего максимизировать среднюю внешнюю награду за эпизод. Для этого перед каждым очередным запуском эпизода обучения многорукий бандит (раздел 7.1) выбирает γ, α , которые будут использоваться в текущем эпизоде обучения для сбора данных. После окончания каждого эпизода бандит считает наградой, полученной «из данного автомата» суммарную (не дисконтированную) внешнюю награду, то есть истинное значение счёта игры.

Этот трюк может успешно применяться для автоматического подбора гиперпараметров прямо по ходу самого обучения. Можно смотреть на это так: будто в формуле (8.26) мы начинаем учить «хорошее» $p(g)$.

8.3.3. Переразметка траекторий

Рассмотрим обучение универсальной оптимальной Q-функции $Q^*(s', g, a')$ для набора подзадач \mathcal{G} . Для данной тройки s, g, a при имеющемся сэмпле следующего состояния s' мы можем обучаться на стандартный одношаговый таргет:

$$y(s, g, a) := r^g(s, a) + \gamma \max_{a'} Q(s', g, a')$$

Сделаем важное наблюдение: допустим, нам известна функция награды r . Это довольно типичная ситуация в тех случаях, когда каким-то образом задан целый набор подзадач в среде, но при необходимости её также можно учить по собираемым сэмплам. Тогда заметим, что мы можем для имеющегося сэмпла $s' \sim p(s' | s, a)$ посчитать значение целевой переменной $y(s, \hat{g}, a)$ для любых $\hat{g} \in \mathcal{G}$. Действительно: в силу (8.27) вне зависимости от того, какую цель преследовал агент, собравший переход $(s, g, a, r, s', \text{done})$, информацию о следующем состоянии s' — сэмпл из функции переходов — можно использовать для обучения оценочной функции для любой задачи:

$$y(s, \hat{g}, a) := r^{\hat{g}}(s, a) + \gamma \max_{a'} Q(s', \hat{g}, a')$$

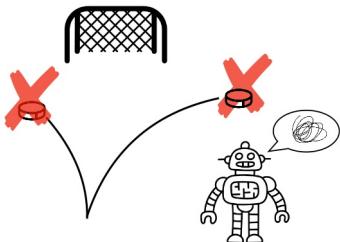
Пример 131: Допустим, вы стояли перед дверью (s), хотели пиццу (g), открыли дверь (a) и дверь открылась (s'). Тогда если бы вы, стоя перед дверью (s), хотели бы мороженое (\hat{g}) и открыли дверь (a), дверь всё равно бы открылась (s' не изменился).

Это открывает путь к **трансферу знаний** (transfer learning): опыт, собранный при решении одной задачи, можно использовать для обучения решению других задач. Понятно, что мы так можем «переразметить» не только переход, но целую траекторию.

Определение 111: Замена в собранной траектории цели g на другую цель \hat{g} и наград $r^g(s, a)$ на $r^{\hat{g}}(s, a)$ называется **переразметкой траектории** (trajectory relabeling).

В простейшем случае, если размерность пространства задач \mathcal{G} не очень большое, можно «сохранить» в буфере (или использовать для on-policy обучения в зависимости от использующегося алгоритма) переразмеченные траектории для всех \hat{g} . Однако, если \mathcal{G} континуально, или поднабор задач богатый, то необходимо выбирать, для каких \hat{g} проводить переразметку. Например, можно посэмплировать \hat{g} случайно; можно ли придумать что-то умнее?

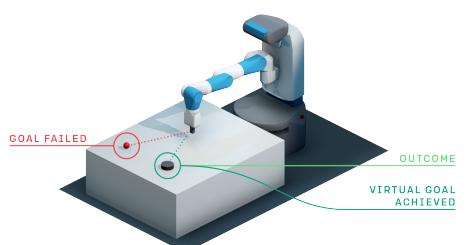
8.3.4. Hindsight Experience Replay (HER)



Идею hindsight-переразметки сначала обсудим на примере частного случая задачи RL, задачи поиска (8.18). Мы уже обсуждали как можно справиться с этой задачей при помощи внутренней мотивации (раздел 8.2); сейчас мы сможем при помощи формализма мультизадачности придумать ещё один интересный способ.

Пусть мы предприняли попытку достичь цели и, через некоторое число шагов, остановились в состоянии \bar{s} , так и не справившись с задачей. Очередной отрицательный пример с константной наградой не позволяет нам начать ничему обучаться... хотя постойте-ка. Мы же достигли состояния \bar{s} ! Давайте положим в него виртуальный тортик. За виртуальный тортик выдадим себе виртуальную $+1$. Теперь у нас есть положительный пример: если бы мы хотели достичь состояния \bar{s} , достичь виртуального тортика, то в ходе этой последней попытки мы всё делали правильно. Другими словами, мы говорим: я так и задумывал изначально, я с самого начала хотел добраться до тортика. В английском языке подобному «мышлению задним числом» соответствует выражение «*in hindsight*», и это слово часто используется для названия алгоритмов с этой идеей.

Hindsight Experience Replay может быть formalизован следующим образом. Формально мы зададимся набором подзадач, и все подзадачи будут являться задачи поиска (8.18). Тогда каждая функция награды r^g однозначно задаётся множеством терминальных состояний — подмножеством $\mathcal{S}_g^+ \subseteq \mathcal{S}$.



Определение 112: Задачей **навигации** (navigation) в среде с пространством состояний \mathcal{S} будем называть набор подзадач $\mathcal{G} \equiv \mathcal{S}$, в котором для решения задачи $g \in \mathcal{S}$ необходимо достичь состояния из \mathcal{S}_g^+ — близких по некоторой метрике состояний к g .

Пример 132: Самый простой и доступный всегда вариант — взять вырожденную метрику, и таким образом рассматривать $\mathcal{S}_g^+ := \{g\}$. В детерминированных средах такая задача даже осмысленна, но дальнейший алгоритм сработает с такой метрикой и для стохастичных сред (поскольку *in hindsight* мы всегда берём

реально достигнутые состояния). Но во многих задачах естественным образом возникают и другие метрики. Например, если вы перемещаете шайбу, и цель задаётся координатами, то можно в качестве метрики взять расстояние от шайбы до цели.

Навигация — это в некотором смысле «полный» набор подзадач: для любого состояния s найдётся задача, для которой оно является терминальным (победным): $\exists \mathcal{S}_g^+ \in \mathcal{G}: s \in \mathcal{S}_g^+$. Это важно для нас, чтобы мы для любой траектории могли найти задачу $g \in \mathcal{G}$, «решением» которой эта траектория является.

Допустим, обучается какой-либо off-policy алгоритм, работающий с реплей буфером. Все модели в алгоритме, принимающие на вход s , теперь также будут принимать описание задачи g . Будем сэмплировать $g \sim p(g)$, как и раньше, в начале эпизода; или же, если этот вариант нам недоступен, пытаться решить исходную задачу с «истинной» целью, которую обозначим за g^* .



Допустим, нам поставили весьма конкретную задачу научиться достигать $\mathcal{S}^+ \subseteq \mathcal{S}$. Возможно, мы не знаем даже описания целевых терминальных состояний («выйти из лабиринта», а где выход — непонятно); тогда положим описание этой «истинной» задачи каким-нибудь специальным вектором g^* .

Допустим, на очередном шаге мы не смогли решить задачу g^* , и очередной эпизод закончился в состоянии \bar{s} . Собранные переходы с $r = 0$ и $done = 0$ отправились в буфер. Но мы понимаем, что если бы мы хотели бы решить задачу g : $\bar{s} \in \mathcal{S}_g^+$, то данная траектория была победной. Тогда на последнем шаге, в конце эпизода, мы получили бы награду +1 и индикатор завершения эпизода $done = 1$. Итак, мы можем переразметить траекторию новой целью g , и добавить переразмеченнную траекторию в буфер.

В итоге, если раньше в буфер попадали только примеры с нулевым (константным) сигналом, и обучение было невозможно, теперь же у нас есть примеры с информативным сигналом.

Здесь важно озаботиться богатством реплей буфера для решения вспомогательных задач. Нужно иметь примеры не только правильных последних шагов, но и всех остальных; важно также не перебить вспомогательными задачами буфер и оставить примеры с целью g^* , чтобы алгоритм в конечном счёте научился решать и её.

Пример 133: Допустим, вы хотите научиться бить по шайбе так, чтобы она попадала в ворота. Вы бьёте по шайбе, но не попадаете; информативного сигнала нет. Тем не менее, люди каким-то образом способны учиться на подобных ошибках, не имея успешных примеров. Вероятно, мы в реальности мыслим в терминах «левее-правее», которых в формализме MDP нет. С точки зрения HER, мы просто учимся попадать в ту же точку, в которую действительно попали, поскольку у нас есть пример того, как это делается; «если бы ворота были в этой точке, я бы попал». И так, обучаясь попадать в различные точки, модель может обобщится на разные g , и однажды попадёт в том числе в реальные ворота g^* .

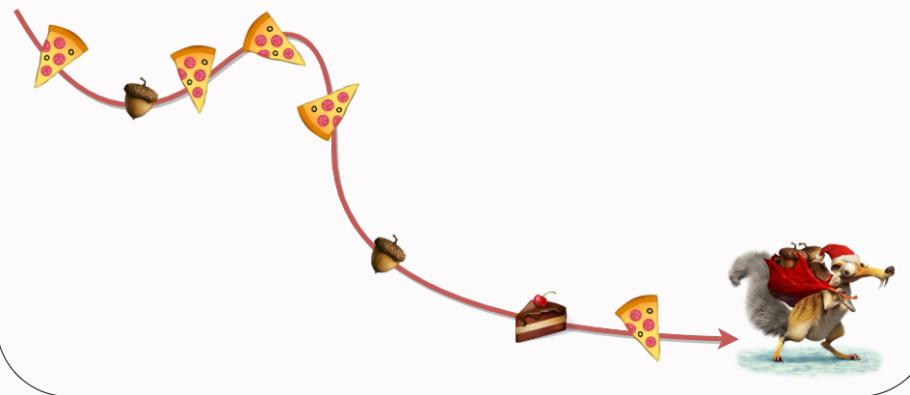
8.3.5. Hindsight Relabeling

Обобщим идею HER на произвольные multi-task задачи. Понятно, что примеры «хорошего» решения задачи намного ценнее примеров неудач: переразметкой мы боремся с тем, что в классическом машинном обучении называли бы «несбалансированностью выборки», увеличивая число примеров «более редкого класса» — положительного опыта, когда агент набирает больше награды, чем уже умеет набирать.

Итак, дан произвольный набор подзадач $r(s, a, g)$. Мы преследовали какую-то цель (неважно какую) и породили траекторию \mathcal{T} . Обозначим $R^g(\mathcal{T})$ суммарную награду за траекторию с точки зрения задачи g . Вопрос: для каких $g \in \mathcal{G}$ имеет смысл переразметить эту траекторию?

Пример 134: Допустим, мы собирали орешки и породили такую траекторию:

STATE SPACE



Для какой задачи данная траектория является «информативной», то есть примером хорошего решения? Очевидный ответ «для пиццы» внезапно неверен: да, мы собрали пиццу больше чем орешков, но вдруг собрать 5 пицц — это очень мало, и такая траектория наоборот является плохой с точки зрения задачи сбора пиццы?

А ещё данная траектория является успешным примером задачи «избежать львов», поскольку ни одного льва в траектории не было. То есть важно не сколько награды $R^g(\mathcal{T})$ мы собрали, а сравнение этого значения с тем, сколько может набрать хорошая стратегия решения задачи g .

Переформулируем вопрос: для какой задачи g из нашего параметрического семейства $r(s, a, g)$ наша траектория является экспертной? Итак, ключевое наблюдение: поиск хороших g для переразметки — это задача обратного обучения с подкреплением, которую мы обсуждали в разделе 8.1.2. В идеях Maximum Entropy Inverse RL лежит ответ на наш вопрос.

С точки зрения Maximum Entropy подхода (формулы (8.2)), траектория \mathcal{T} является экспертной для задачи g с вероятностью

$$p(\mathcal{T} | g) := \frac{e^{R^g(\mathcal{T})} \prod_{t \geq 0} p(s_{t+1} | s_t, a_t)}{Z(g)}$$

Заметим, что нормировочная константа для каждого g своя. Рассмотрим в качестве прайора $p(g)$, из которого нам, например, приходят задачи в начале эпизодов или возьмём какое-нибудь равномерное распределение. Тогда по формуле Байеса:

$$p(g | \mathcal{T}) \propto p(\mathcal{T} | g)p(g) = \frac{p(g)e^{R^g(\mathcal{T})}}{Z(g)} \quad (8.28)$$

Часть с вероятностями переходов сократились с нормировочной константой, поскольку они общие для всех g . Эта формула и говорит нам, что нужно использовать для поиска хорошей переразметки g не просто задачи, для которых мы собрали большую суммарную награду $R^g(\mathcal{T})$, а отнормированную на соответствующий интеграл $Z(g)$. Именно с вероятностями (8.28) цель g можно считать «экспертной».

Формула (8.28) даёт теоретический ответ на наш вопрос, но как использовать её на практике, то есть как сэмплировать из такого распределения? Здесь придётся ограничиться эвристиками. Если \mathcal{G} конечно и мало, то мы можем подсчитать для каждого g все суммарные награды $R^g(\mathcal{T})$, но с нормировочной константой дела обстоят плохо:

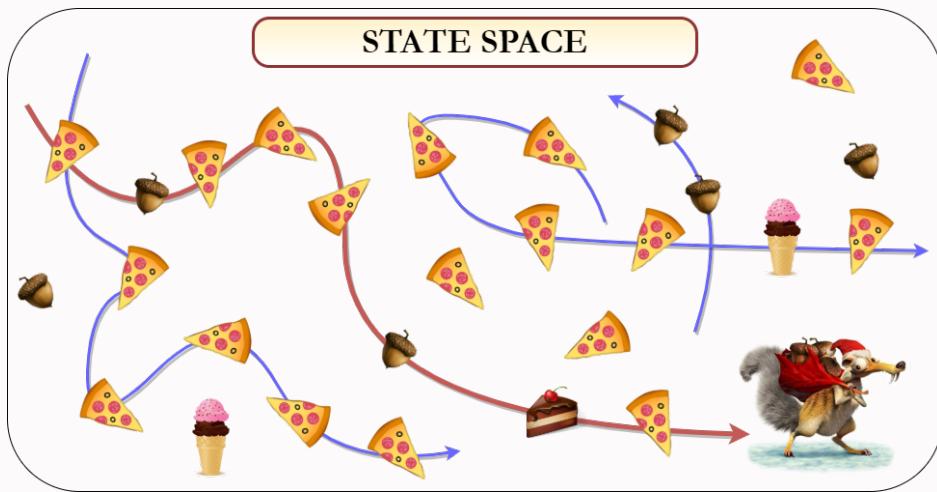
$$Z(g) = \int_{\mathcal{T}} e^{R^g(\mathcal{T})} \prod_{t \geq 0} p(s_{t+1} | s_t, a_t) d\mathcal{T},$$

где траектории, вообще говоря, должны быть порождены оптимальной стратегией, решающей задачу g ; мы можем попробовать считать, что текущая универсальная стратегия $\pi(a | s, g)$ и является экспертной для искомого g . Однако практические эвристики здесь сводятся к ещё большим упрощениям: например, к тому, чтобы взять какие-нибудь траектории из буфера $\mathcal{T}_1, \mathcal{T}_2 \dots \mathcal{T}_M$ и посчитать примерно среднюю награду, которую мы набираем как-нибудь так:

$$Z(g) \approx \frac{1}{M} \sum_{i=1}^M e^{R^g(\mathcal{T}_i)}$$

Если \mathcal{G} велико или, например, континуально, то просто для поиска хороших g сэмплируется случайно несколько целей-кандидатов, и поиск хороших целей для переразметки проводится среди них.

Пример 135: Продолжим пример 134 и посмотрим на несколько других встречавшихся траекторий из буфера.



Теперь мы видим, что собрать 5 пицц — не такое редкое явление; наша собственная стратегия когда-то уже получала даже больше. А вот до тортиков мы в засемплированных траекториях никогда не добирались, и поэтому нашу траекторию имеет смысл переразметить для задачи «добраться до тортика». Таким образом у нас в буфере появится пример сбора целого 1 тортика, что, согласно примерам других траекторий, «много» для этой задачи.

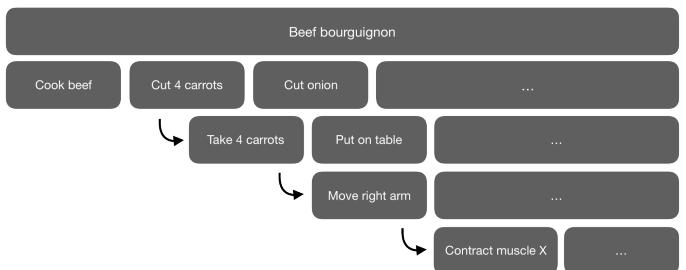
§8.4. Иерархическое обучение с подкреплением

8.4.1. Опции

Представьте, что ваша задача — приготовить суп. Пытаться решить задачу методом проб и ошибок вам не придётся: у вас есть рецепт. Однако, беда: в рецепте первым пунктом написано «возьмите чистую кастрюлю»... И учиться брать чистую кастрюлю, видимо, придётся методом проб и ошибок.

Большинство задач в сложных средах делятся на подзадачи. «Рецепта», позволяющего разложить сложную задачу на простые, однако, в общем случае нет, и агенту нужно не просто научиться решать набор подзадач, но и определять, какие именно подзадачи требуется выполнить для успеха.

Мы можем не вводить понятие подзадач или иерархичности и пытаться оптимизировать суммарную награду напрямую, как делали до этого. Но тогда наш рецепт для приготовления супа будет выглядеть примерно так: «сожмите вот этот перечень мышц, теперь вот этот, так-так-так, ваша рука начала подниматься...». Введение иерархий позволит агенту смотреть на задачу на более абстрактном уровне — на уровне выбора подзадач. Важно, что на этом уровне будет совершенно другой «масштаб времени»: число последовательных решений для решения всей задачи существенно сократится. Такое свойство «высокоуровневых стратегий» называется ***temporal abstraction***. Концептуально, именно на этом уровне агент должен заботиться о награде, описывающей основную задачу. Пока что, для начала, мы ограничимся чуть менее амбициозной задачей: давайте как-нибудь сделаем нашу стратегию «многоуровневой», например, следующим образом.



Определение 113: Для данного MDP **опцией** (*option*) g называется пара* (π_g, β_g) :

- π_g — стратегия для исходного MDP.
- $\beta_g: \mathcal{S} \rightarrow [0, 1]$ — **политика терминальности** (termination policy).

* в оригинале дополнительно рассматривалось множество $I_g \subseteq \mathcal{S}$ тех состояний, в которых опцию можно начать выполнять, однако в дальнейшем повествовании ситуация $I_g \neq \mathcal{S}$ нам не встретится.

Пусть у нас есть множество опций \mathcal{G} , то есть даны или несколько разных стратегий π_g , или просто универсальная стратегия $\pi(a | s, g)$. Эти стратегии, работающие «с исходным» MDP на уровне **примитивных действий** (primitive actions) (элементов \mathcal{A}), будем далее также называть **рабочими** (workers). Также мы заводим «высокоуровневую» стратегию, которую далее будем называть **менеджером** (manager) или **мастерстратегией** (master policy) $\hat{\pi}(g | s)$. Всё, что относится к менеджеру, будем помечать звёздочкой \star над буквами. Высокоуровневые действия также ещё называют **макро-действиями**, а примитивные действия — **микро-действиями**.

На очередном шаге менеджер, исходя из текущего состояния s_t , выбирает опцию, которая будет далее работать в среде: $g_t \sim \hat{\pi}(g_t | s_t)$. Выбранный рабочий генерирует примитивное действие: $a_t \sim \pi(a_t | s_t, g_t)$. Среда переходит в новое состояние $s_{t+1} \sim p(s_{t+1} | s_t, a_t)$. И в этот момент вызывается политика терминальности опции g : с вероятностью $\beta_{t+1} \sim \text{Bernoulli}(\beta_g(s_{t+1}))$ рабочий завершает свою работу и снова передаёт решение менеджеру (тот снова выбирает следующего рабочего, и так далее). Если же политика терминальности не срабатывает, менеджер не вызывается, и взаимодействовать со средой продолжает рабочий π_{g_t} .

Для удобства будем считать, что в рамках такого фреймворка в траекториях хранятся не только примитивные действия, но и решения менеджера вместе с решениями политики терминальности:

$$\mathcal{T} := (g_0, a_0, r_0, s_1, \text{done}_1, \beta_1, g_1, a_1, r_1, s_2, \text{done}_2, \beta_2 \dots)$$

причём $g' = g$ с вероятностью 1, если $\beta' = 1$, и $\beta_0 = 0$. Таким образом, вероятностная модель порождения траекторий задана так:

$$\begin{aligned} & \begin{cases} g_t \sim \hat{\pi}(g_t | s_t) & \text{если } \beta_t = 0 \\ g_t := g_{t-1} & \text{если } \beta_t = 1 \end{cases} \\ & a_t \sim \pi(a_t | s_t, g_t) \\ & s_{t+1} \sim p(s_{t+1} | s_t, a_t) \\ & \beta_t \sim \text{Bernoulli}(\beta_g(s_t)) \quad (t > 0) \end{aligned}$$

Мы поставим себе задачу обучать эту конструкцию end-to-end, то есть оптимизировать параметры стратегии менеджера, стратегий рабочих (опций) и политик терминальности опций напрямую с единственной целью максимизировать среднюю награду.

8.4.2. Semi-MDP

Как нам это всё чудо обучать? Начнём с менеджера. Для простоты будем считать, что набор опций нам дан, то есть даны распределения $\pi_g(a | s)$ и $\beta_g(s)$; хотим научиться обучать $\hat{\pi}(g | s)$. Попробуем понять, не живёт ли он в каком-то MDP. Поскольку мы решили, что на вход он получает текущее состояние \mathcal{S} , то его пространство состояний такое же, как и в исходном MDP. Его пространством действий являются макро-действия \mathcal{G} . На каждом шаге менеджер выбирает $g \in \mathcal{G}$, после чего среда для менеджера работает так: стратегия-рабочий для выбранного g бегает в настоящей среде и решает поставленную подзадачу, переводя среду в новое состояние \mathcal{S}' . То, в каком состоянии окажется среда по итогу, полностью задано вероятностной моделью — для менеджера это можно рассматривать как функцию переходов $p^*(\mathcal{S}' | \mathcal{S}, g)$. Наконец, награда для менеджера за этот шаг есть сумма собранной за время работы рабочего награды основной цели:

$$\hat{r}(s, a) := \sum_{t=0}^{\tau} \gamma^t r_t \tag{8.29}$$

где τ — число шагов, затраченных рабочим на решение задачи. И вот тут возникает нюанс.

Награду, который менеджер получит за второй шаг, необходимо дисконтировать на γ^τ , поскольку в настоящем MDP прошло τ шагов. Более того, это τ — случайная величина; рабочий мог потратить на решение как 10 шагов, так и 100. А значит, менеджер живёт не совсем в MDP; для него действия имеют различную продолжительность по времени. Если раньше в MDP все действия, можно считать, гарантированно «выполнялись» один шаг, то теперь, для менеджера, это не так.

Определение 114: MDP называется **полумарковским** (Semi-Markov decision process, sMDP), если его функция перехода $p(s', \tau | s, a)$ помимо следующего состояния возвращает время $\tau > 0$, «затраченное» на выполнение данного шага.

В общем случае в sMDP время шагов может быть и вещественным числом (в частности, так можно учитывать, что среда взаимодействует с агентом в режиме «реального времени»), но в нашем случае $\tau > 0$ — всегда натуральное число. Итак, при работе в sMDP в траекториях дополнительно необходимо хранить время каждого шага τ_t ; награда за t -ый шаг дисконтируется не на γ^t , а на $\gamma^{\sum_{i=0}^t \tau_i}$.

Ранее рассматриваемая теория достаточно естественно обобщается на данный кейс. Например, уравнения Беллмана должны учитывать время в дисконтировании; например, для Q-функции:

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

Адаптация Q-learning для sMDP выглядит соответствующе: для перехода $(s, a, r, \tau, s', \text{done})$ обновление выглядит так:

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha \left(r + \gamma^\tau \max_{a'} Q(s', a') \right)$$

То есть по прошедшему времени τ мы тоже стохастически аппроксимируем: используем сэмпл из буфера вместе с сэмплом s' .

8.4.3. Оценочная функция по прибытию (U-функция)

Итак, менеджер живёт в полумарковском процессе принятия решений sMDP $(\mathcal{S}, \mathcal{G}, \mathcal{P}^*, \mathbf{r}^*)$, где \mathbf{r}^* определено (8.29), а функция переходов³ \mathcal{P}^* задаётся процессом взаимодействия выбранного рабочего со средой с завершением по триггеру соответствующей политики терминальности. Можно обучать оценочные функции менеджера аналогом Q-learning для такого sMDP, но тогда для одного обновления понадобится получать один переход — ждать, пока сработает политика терминальности выбранного рабочего. Оказывается, это неизбежно.

Вместо этого удобнее работать с теорией на уровне одношаговых рекурсивных соотношений между величинами, где один шаг — это генерация одной случайной величины. Раньше в обычных MDP мы как делали: сгенерируем действие, и вся будущая награда — это Q-функция. Среда ответила нам сэмплом s' (ещё наградой за шаг и флагом `done`, но считаем, что это идёт «в одном комплекте») — и дальнейшая награда есть V-функция. Это было очень удобно, поскольку все эти оценочные функции легко выражались между собой. Нам надо поступить также для траектории, состоящей из случайных величин $\mathbf{g}, \mathbf{a}, s', \beta'$: выбор менеджера, выбор стратегии, отклик среды, выбор политики терминальности.

По определению, $\dot{Q}(s, g)$ обозначает следующее: менеджер сидел в некотором в состоянии s и выбрал опцию g , или, что не существенно, опция g уже была выбрана на предыдущем шаге, а политика терминальности не затриггерилась (в любом случае, в состоянии s активировалась опция g), и функция возвращает среднюю будущую награду. Мы опустим здесь верхний индекс π , подразумевая, что мы считаем оценочную функцию для всего комплекта подконтрольных нам распределений: стратегии менеджера, рабочего и политик терминальности. Попробуем получить одношаговое рекурсивное соотношение для $\dot{Q}(s, g)$, связав её с оценочной функцией для следующей случайной величины — выбором действия a рабочим. Каким рабочим? Ну, раз активирована опция g , то однозначно рабочим $\pi(a | s, g)$.

Мы уже поняли, что на стратегию рабочих можно смотреть как на универсальную стратегию в MDP с пространством состояний $\mathcal{S} \times \mathcal{G}$; мы можем определить $V(s, g)$ и $Q(s, g, a)$ — универсальные оценочные функции рабочих — как оценочные функции в таком MDP, как будущие награды после реализации поступающих на вход случайных величин. Тогда в силу структуры вероятностной модели:

Утверждение 86:

$$\dot{Q}(s, g) = V(s, g) = \mathbb{E}_{\pi_g(a|s)} Q(s, a, g) \quad (8.30)$$

Попробуем пойти дальше и построить одношаговое соотношение для $Q(s, a, g)$. Что случится после выбора действия a рабочим π_g ? Среда выдаст награду за шаг, перейдёт в следующее состояние s' , и вот тут дальше случится загвоздка: будет генерироваться бернулиевская β_g . Если политика терминальности затриггерилась, то первым следующем шагом генерации траектории будет сэмплирование действия менеджером, и мы по определению получим столько, сколько выдаёт V-функция менеджера $\dot{V}(s')$. Здесь важно, что $\dot{V}(s)$ есть хвост награды по траектории после попадания в s при условии (!) срабатывания триггера $\beta = 1$, она предполагает, что первым шагом менеджер сможет выбрать новую опцию. Но если $\beta = 0$, то действие выбирает текущий рабочий, и тогда будущая награда равна $V(s, g)$ (которое, как мы уже разобрались в (8.30), совпадает с $\dot{Q}(s, g)$). Работать с этим неудобно, поскольку мы не знаем, сработал ли триггер или нет, и поэтому для удобства вводится вспомогательная U-функция — это хвост награды по траектории после попадания в состояние s без каких-либо дополнительных ограничений.

Определение 115: Для sMDP, заданного MDP с набором опций $\{\pi_g, \beta_g\}$, U-функцией или **оценочной функцией по прибытию** (state-value function upon arrival) называется

$$U(s', g) := (1 - \beta_g(s')) \dot{Q}(s', g) + \beta_g(s') \dot{V}(s') \quad (8.31)$$

Что это за формула: на вход U-функция получает состояние, в которое «входит» агент, и текущую опцию. Дальше расписано матожидание по срабатыванию триггера. Если политика терминальности не срабатывает, следующей опцией снова будет g , поэтому дальнейшая награда эквивалентна $\dot{Q}(s', g)$. Иначе выбор переходит к менеджеру и мы получим $\dot{V}(s')$. Таким образом, по определению:

Утверждение 87:

$$Q(s, g, a) = r(s, a) + \gamma \mathbb{E}_{s'} U(s', g) \quad (8.32)$$

³мы, в принципе, можем полностью расписать это распределение $p(\dot{s}' | \tau | s, g)$, но это получится громоздко: нужно вычислить $a_0, s_1, a_1, \dots, s_{\tau-1}, a_{\tau-1}$, учесть вероятность попадания в $p(\dot{s}' | s_{\tau-1}, a_{\tau-1})$, срабатывание политики терминальности $\beta_g(\dot{s}')$ и несрабатывание политики терминальности для $s_1, s_2 \dots s_{\tau-1}$ внутри интеграла.

8.4.4. Intra-option обучение

Термин «*intra-option*» означает, что мы можем за счёт таких соотношений обучать функции менеджера, используя информацию после выполнения каждого микро-действия (после получения информации о s') вне зависимости от того, запускалась ли на данном шаге в принципе стратегия-менеджер или нет. Интуитивно основное соображение выглядит так: если стратегия-рабочий сделала один шаг в среде, а политика терминалности не сработала, то это всё равно что в этот момент менеджер снова выбрал того же самого рабочего.

Допустим, мы хотим обучить $\dot{Q}^*(s, g)$ — оптимальную Q-функцию менеджера — в предположении, что политика терминалности и стратегии рабочего зафиксированы и не меняются. Попробуем составить уравнение оптимальности для неё в аналогии с обычными уравнениями.

Определение 116: Для sMDP, заданного MDP с фиксированным набором опций $\{\pi_g, \beta_g\}$, *оптимальной U-функцией* называется

$$U^*(s', g) := (1 - \beta_g(s'))\dot{Q}^*(s', g) + \beta_g(s') \max_{g'} \dot{Q}^*(s', g') \quad (8.33)$$

В этом определении мы по сути просто взяли формулу обычной U-функции (8.31) и заменили в ней $\dot{V}(s')$ на $\max_{g'} \dot{Q}^*(s', g')$ в силу предположения оптимальности, что менеджер всегда выбирает наилучшую опцию g' .

Давайте теперь попробуем выразить $\dot{Q}^*(s, g)$ через неё же саму, используя $U^*(s', g)$.

Утверждение 88: Q-функция менеджера удовлетворяет следующему рекурсивному уравнению:

$$\dot{Q}^*(s, g) = \mathbb{E}_{\pi_g(a|s)} [r(s, a) + \gamma \mathbb{E}_{s'} U^*(s', g)] \quad (8.34)$$

Доказательство. Допустим, менеджер выбрал опцию g , и рабочий сделал один шаг в среде (генерировалось $a \sim \pi_g(a | s)$ и s' в исходном MDP). Дисконтирование случилось только на γ . После этого с вероятностью $\beta_g(s')$ менеджер сможет выбрать новую подзадачу (это бы соответствовало обычному уравнению оптимальности Беллмана, поскольку прошёл всего один шаг), а с вероятностью $1 - \beta_g(s)$ менеджеру не предоставляется выбора. В такой ситуации можно считать, что менеджер просто «обязан» снова выбрать g : вероятностная модель со следующего шага выглядит в точности также. ■

Итак, мы можем для обучения менеджера пользоваться Q-learning-ом для sMDP: просить рабочего g решить подзадачу, дождаться s', τ и делать один шаг обновления. Но «intra-option» рекурсивное уравнение (8.34) показывает интересную альтернативу: для обучения функции ценности менеджера для перехода $\mathbb{T} := (s, g, a, r, s', \text{done})$ можно рассчитать целевую переменную как

$$y(\mathbb{T}) := r(s, a) + \gamma(1 - \text{done})U^*(s', g), \quad (8.35)$$

где U-функция вычисляется полностью по формуле (8.33) (мат.ожидание по Бернулиевской β можем взять явно), и далее, как обычно, минимизировать MSE:

$$\mathbb{E}_{\mathbb{T}} \left(y(\mathbb{T}) - \dot{Q}^*(s, g) \right)^2 \rightarrow \min_{\dot{Q}^*}$$

Заметим, что переходы мы можем брать любые, в том смысле, что не обязательно, чтобы g было выбрано менеджером именно на данном шаге. Для корректного обучения мат.ожиданий нам требуется лишь, чтобы $a \sim \pi_g(a | s)$, $r = r(s, a)$ и $s' \sim p(s' | s, a)$. Обратим внимание на первое условие: сейчас мы считаем опции зафиксированным (частью стационарной среды). Если стратегии рабочих меняются (а они будут меняться, так как будут обучаться), условие стационарности нарушается, и off-policy режим обучения мы, естественно, теряем.

8.4.5. Обучение стратегий рабочих

Утверждение 89: В предположении оптимальности менеджера оптимальная Q-функция рабочего удовлетворяет следующему уравнению:

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

Доказательство. Доказательство в точности повторяет теорему 88 за тем исключением, что действие a уже подано оценочной функции на вход. ■

Мораль отсюда простая: чтобы учить оценочные функции рабочего, мы можем использовать те же таргеты $y(\mathbb{T})$ (8.35). Только теперь для рабочих оценочная функция дополнительно обусловлена на действие a .

$$\mathbb{E}_{\mathbb{T}} (y(\mathbb{T}) - Q^*(s, g, a))^2 \rightarrow \min_{Q^*}$$

Важный момент — здесь можно применять hindsight-приём, который мы встречали в разделе 8.3.3. Действительно, поскольку единственное, что теперь требуется от переходов — $s' \sim p(s' | s, a)$, мы можем проделать переразметку имеющегося в опыте перехода $\mathcal{T} := (s, g, a, r, s', \text{done})$ на $\hat{\mathcal{T}} := (s, \hat{g}, a, r, s', \text{done})$ для любого \hat{g} . То есть: что было бы, если бы в состоянии s мы пользовались бы опцией \hat{g} и выбрали бы действие a ? Тогда мы попали бы в состояние s' и получили бы награду r . Другой информации для получения прецедента для обучения $Q^*(s, \hat{g}, a)$ нам и не нужно. Это значит, что для рабочих мы можем обучать оценочные функции сразу для всех $g \in \mathcal{G}$.

Мы обсудили обучение Q-функций менеджера и рабочего с одношаговых таргетов; конечно же, имея на руках рекурсивные соотношения, мы можем построить полные аналоги всей стандартной теории. Например, для применения Policy Gradient подхода, нам нужны не сами оценочные функции, а их несмешённые оценки; в таких ситуациях достаточно иметь лишь достаточно обучать лишь Q-функцию менеджера. Действительно: если мы знаем $\dot{Q}(s, g)$, то тогда в силу (8.32):

$$Q(s, g, a) \approx r(s, a) + \gamma U(s', g), \quad s' \sim p(s' | s, a),$$

где $U(s', g)$ выражается через Q-функцию менеджера в силу формулы (8.31) — несмешённая оценка. В качестве бэйзлайна возможно использовать $V(s, g)$, которая совпадает с $\dot{Q}(s, g)$.

8.4.6. Обучение функций терминалности

Перейдём к оставшемуся открытому вопросу: как обучать политику терминалности? Рассмотрим такую ситуацию: мы сидим в состоянии s и выполняем опцию g . Нужно ли останавливать рабочего, полагая $\beta(s, g) = 1$, или можно продолжать действовать с его помощью? С точки зрения оптимальных оценочных функций, в первом случае мы получим $\dot{V}^*(s)$, а во втором $V^*(s, g)$. Но очевидно, что

$$\dot{V}^*(s) = \max_{\hat{g}} \dot{Q}^*(s, \hat{g}) \geq \dot{Q}^*(s, g) = V^*(s, g),$$

то есть оптимально прерывать рабочего на каждом шаге.

Это, конечно, затыка в нашей теории, поскольку если рабочий прерывается на каждом шаге, никакого temporal abstraction у нас не получится. Попробуем обратиться к Policy gradient подходу; мы сможем обучать функции терминалности, пользуясь формулой градиентов по их параметрам, когда сами политики терминалности будут стохастичны.

Пусть $\beta_\theta(s, g) \in [0, 1]$ выдаёт вероятность бернульлевской величины и дифференцируемо по параметрам θ . Пусть $\dot{A}(s, g) := \dot{Q}(s, g) - \dot{V}(s)$ — Advantage-функция менеджера. Функционал, который мы оптимизируем, для начального состояния s_0 , равен по определению $\dot{V}(s_0)$.

Теорема 90 — Termination Gradient Theorem:

$$\nabla_\theta \dot{V}(s) = -\mathbb{E}_{\mathcal{T}|s_0=s} \sum_{t \geq 0} \gamma^t \nabla_\theta \beta_\theta(s_t, g_t) \dot{A}(s_t, g_t) \quad (8.36)$$

Доказательство. Используя уравнения связи (8.30) и (8.32):

$$\nabla_\theta \dot{V}(s) = \nabla_\theta \mathbb{E}_g \mathbb{E}_a [r(s, a) + \gamma \mathbb{E}_{s'} U(s', g)]$$

Мат.ожидания здесь берутся по стратегиям менеджера и рабочего, не зависящих от θ , поэтому мы сразу проносим градиент вплоть до оценочной функции по прибытию, а дальше применяем стандартную для Policy Gradient технику.

$$\begin{aligned} \nabla_\theta U(s', g) &= \nabla_\theta [\beta_\theta(s', g) \dot{V}(s') + (1 - \beta_\theta(s', g)) \dot{Q}(s', g)] = \\ &= \nabla_\theta \beta_\theta(s', g) [\dot{V}(s') - \dot{Q}(s', g)] + \beta_\theta(s', g) \nabla_\theta \dot{V}(s') + (1 - \beta_\theta(s', g)) \nabla_\theta \dot{Q}(s', g) \end{aligned}$$

Здесь мы хотим вернуться к мат.ожиданиям по траекториям, и в том числе к мат.ожиданиям по β . Для этого достаточно заметить, что в полученном выражении ровно такое мат.ожидание и стоит. Действительно, убедимся, что

$$\beta_\theta(s', g) \nabla_\theta \dot{V}(s') + (1 - \beta_\theta(s', g)) \nabla_\theta \dot{Q}(s', g) = \mathbb{E}_{\beta_\theta} \mathbb{E}_{g'} \nabla_\theta \dot{Q}(s', g')$$

Если в выражении справа выпало $\beta_\theta = 0$, то дальше гарантировано $g' = g$, мат.ожидание по g' вырождается и мы получаем второе слагаемое из выражения слева. Если же в выражении справа выпало $\beta_\theta = 1$, то дальше $\mathbb{E}_{g'} \nabla_\theta \dot{Q}(s', g') = \nabla_\theta \dot{V}(s')$, и мы получаем первое слагаемое из выражения слева.

Собирая всё вместе, получаем рекурсивную формулу:

$$\nabla_\theta \dot{V}(s) = \mathbb{E}_g \mathbb{E}_a \gamma \mathbb{E}_{s'} [-\nabla_\theta \beta_\theta(s', g) \dot{A}(s', g) + \mathbb{E}_\beta \mathbb{E}_{g'} \nabla_\theta \dot{Q}(s', g')]$$

Собирая рекурсивную формулу в мат.ожидание по всей траектории, получаем доказываемое. ■

Крайне интуитивная формула (8.36) говорит ровно то, с чего мы начали обсуждение оптимального поведения политики терминалности: если $\dot{A}(s, g) > 0$, то текущая опция лучше той, которую в среднем выбрал бы текущий менеджер, и поэтому стоит продолжать её выполнять; $\beta(s, g)$ уменьшается. Но если $\dot{A}(s, g) < 0$, то менеджер сейчас может выбрать более хорошую опцию, более хорошего рабочего, и поэтому нужно передавать ему управление: $\beta(s, g)$ надо уменьшать.

Как мы знаем, для оптимальных стратегий $\max_g \dot{A}(s, g) = 0$, и поэтому такая градиентная оптимизация чревата вырождающими ситуациями. Типично, что менеджер начинает получать управление на каждом шаге. Бороться с этим приходится костылями: например, дополнительными регуляризациями на политику терминалности, чтобы она как можно реже передавала управление менеджеру, и менеджеры с рабочим учились в этих условиях с «неоптимальной» политикой терминалности. Популярное решение — добавить в формуле (8.36) к оценке Advantage небольшое положительное число.

8.4.7. Феодальный RL

В рамках подхода с опциями мы не факт, что выучим какие-то разумные подзадачи; скорее, мы надеемся, что, возможно, рабочие как-то «распределят» между собой области среды, за которые будут ответственны. Мы понимаем, что, деля сложную задачу на последовательность более мелких, мы можем переставать думать об исходной задаче на уровне выполнения подзадач.

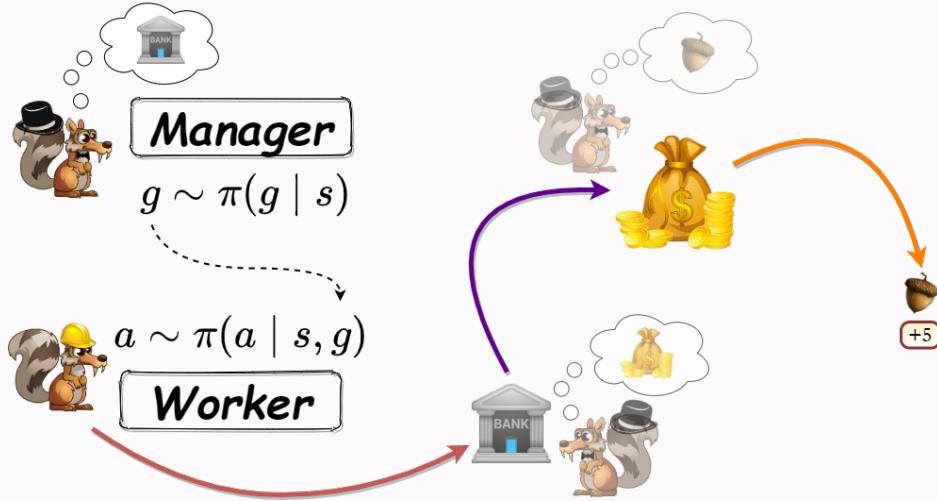
Пример 136: То есть: если было принято решение достать чистую кастрюлю, то истинную цель этой установки (награду за приготовление супа) можно не принимать в расчёты и оптимизировать только награду за доставание кастрюли. Такое мышление открывает возможности для переиспользования оптимальных стратегий доставания кастрюли для других задач (например, приготовления пельменей) — потенциальный путь к transfer learning.

Соображение лежит в основе принципа **феодализма** (feudalism), который можно сформулировать в виде следующих постулатов:

- стратегия, действующая на некотором уровне абстрактности, не должна задумываться о более высоких уровнях задачах («единственная задача вассала — выполнение задач, поставленных феодалом»).
- высокоуровневая стратегия не рассматривает примитивные действия \mathcal{A} («феодала не заботит, как вассал будет достигать поставленных ему целей»).
- при наличии нескольких уровней иерархии действует принцип «вассал моего вассала не мой вассал». В дальнейшем повествовании мы ограничимся двумя уровнями иерархии, но теорию легко можно обобщить на большее число уровней.

Итак, общая схема иерархического RL с двумя уровнями, к которой мы хотим прийти, выглядит так. Откуда берётся набор подзадач; необходимо придумать, откуда его брать. Заводится стратегия-менеджер и стратегии-рабочие для каждой подзадачи. Когда менеджер выбирает подзадачу g , запускается стратегия соответствующего рабочего π_g . Она делает несколько шагов в исходной среде, пока не решит задачу или не «сдастся» (кастрюли в шкафу не оказалось и нужно бежать в магазин за новой); критерий остановки процесса выполнения стратегии рабочего остаётся открытым. Далее менеджер снова принимает решение о следующей решаемой подзадаче, оптимизируя исходную награду; рабочие стремятся решить свои подзадачи, для чего им нужно как-то задать функцию награды.

Пример 137: В идеале, иерархический RL должен выглядеть как-то примерно так. Менеджер смотрит на текущее состояние и говорит: нужно добраться до банка! Откуда-то берётся функция награды, описывающая задачу g «добраться до банка», и соответствующая стратегия рабочего $\pi(a | s, g)$ в течение многих шагов эту задачу решает. В банке происходит определение того, что задача решена, и управление снова передаётся менеджеру. Тот видит, что агент находится в банке, и выбирает новую задачу g , например, «найти кучу денег». Вызывается новая стратегия рабочего, он снова решает задачу, менеджер выбирает следующую подзадачу, и так далее.



Иными словами, задачи и подзадачи в концепции RL всегда задаются MDP, и для подобных иерархических RL-алгоритмов нам нужно определиться, что есть MDP для той стратегии, которая принимает решение о подзадачах, и что есть MDP подзадач.

Строить такую идеальную схему, чтобы алгоритм сам смог выучить и набор подзадач, и механизмы определения моментов передачи управления менеджеру, мы на текущий момент не умеем. В теории опционов мы абстрагировались от идей подзадач и оптимизировали всю нашу иерархическую конструкцию на максимизацию исходной функции награды, а основная сложность заключалась в обучении политики терминалности, и с ней же были связаны основные причины нестабильности процесса обучения. В рамках двух следующих алгоритмов мы будем следовать заветам феодального RL и выберем какой-нибудь набор подзадач, но упростим схему, убрав политику терминалности и заменив её на эвристики.

8.4.8. Feudal Networks (FuN)

Напрашивается в качестве набора подзадач брать задачу достижения некоторого состояния $\mathbf{g} \in \mathcal{S}$. Основная идея *феодальных сетей* (feudal networks) в том, чтобы задавать рабочим не целевое состояние, а направление, в котором нам хотелось бы сдвинуться, в пространстве латентных описаний состояний. Для этого менеджер переводит текущее состояние s_t в латентное пространство $\mathbf{z}_t := f_\theta(s_t) \in \mathbb{R}^d$, где θ — параметры, после чего генерирует вектор $\mathbf{g}_t \in \mathbb{R}^d$. Выданный менеджером вектор нормируется, так что $\|\mathbf{g}_t\|_2 = 1$: эта хитрость нужна, чтобы запрашиваемое направление не близко к тривиальному нулевому смещению и не достаточно «большое».

Вместо политик терминалности вводится следующее упрощение: менеджер всё-таки будет вызываться на каждом шаге, и для каждого s_t будет генерироваться целевое направление \mathbf{g}_t ; но обучаться он будет, исходя из предположения, что рабочий действительно выполняет поставленную задачу, и через K шагов, где K — гиперпараметр, агент успешно сдвигается в указанном менеджером направлении. Иначе говоря, для функции переходов менеджера (которая составлена композицией настоящей функции перехода и стратегии рабочего) вводится следующее предположение:

$$\mathbf{P}(z_{t+K} = z_t + \mathbf{g}_t) = 1$$

Здесь $z_{t+K} := f_\theta(s_{t+K})$ — латентное описание состояния через K шагов. В жёсткой форме с подобной функцией переходов работать всё равно не столь удобно, поэтому вместо этого предположим более мягкую форму:

$$p(z_{t+K} | z_t, \mathbf{g}_t) \propto \exp(d(z_{t+K} - z_t, \mathbf{g}_t)), \quad (8.37)$$

где d — некоторое расстояние между векторами, например, косинусное (с учётом того, что $\|\mathbf{g}_t\|_2 = 1$):

$$d(z_{t+K} - z_t, \mathbf{g}_t) := \frac{(z_{t+K} - z_t)^T \mathbf{g}_t}{\|z_{t+K} - z_t\|_2}$$

В таких предположениях нас волнует лишь направление $z_{t+K} - z_t$, в котором сдвинулось состояние; можно считать, что его мы тоже отнормировали, и тогда предположение (8.37) просто задаёт распределение на единичной сфере с центром в z_t , у которого мода находится в \mathbf{g} , и чем менее скоррелировано выбранное направление и «цель» \mathbf{g} , тем меньше вероятность.

Определение 117: Распределение

$$p(d | \mathbf{g}) \propto \exp(d^T \mathbf{g}), \quad (8.38)$$

где $\|\mathbf{g}\| = 1$ и $\|d\| = 1$, называется *распределением Мизес-Фишера* (von Mises-Fisher distribution).

Утверждение 90: Нормировочная константа распределения Мизес-Фишера (8.38) не зависит от \mathbf{g}

Доказательство. Рассмотрим нормировочную константу для \mathbf{g} :

$$\int_{\|\mathbf{d}\|=1} \exp(\mathbf{d}^T \mathbf{g}) d\mathbf{d}$$

Возьмём какое-нибудь другое единичное направление $\hat{\mathbf{g}}$. Сделаем замену переменных: пусть \mathbf{M} — матрица поворота, переводящая $\hat{\mathbf{g}}$ в \mathbf{g} : $\mathbf{g} = \mathbf{M}\hat{\mathbf{g}}$. Тогда сделаем замену переменных: $\hat{\mathbf{d}} := \mathbf{M}^T \mathbf{d}$. Если \mathbf{d} пробегает единичную сферу, то так как \mathbf{M} — матрица поворота, $\hat{\mathbf{d}}$ тоже пробегает единичную сферу; определитель матрицы поворота \mathbf{M} также единичен, поэтому при замене переменной новых множителей не возникает. Получаем

$$\int_{\|\mathbf{d}\|=1} \exp(\mathbf{d}^T \mathbf{M}\hat{\mathbf{g}}) d\mathbf{d} = \int_{\|\hat{\mathbf{d}}\|=1} \exp(\hat{\mathbf{d}}^T \hat{\mathbf{g}}) d\hat{\mathbf{d}},$$

что есть нормировочная константа для $\hat{\mathbf{g}}$. ■

Менеджер может считать, что, выбирая действие \mathbf{g}_t , он выбирает состояние, в котором окажется через K шагов, причём предлагается считать, что это распределение задано (8.37), то есть:

$$\log \dot{\pi}(\mathbf{g}_t | s_t) := \log p(z_{t+c} | z_t, \mathbf{g}_t) = d(z_{t+c} - z_t, \mathbf{g}_t) + \text{const}(\theta)$$

Подставляя это в формулу градиента для актёра, получается (для одного перехода) следующая формула:

$$\nabla_{\theta}^{\text{manager}} = \nabla_{\theta} d(z_{t+K} - z_t, \mathbf{g}_t(\theta)) \dot{\mathbf{A}}(s_t, \mathbf{g}_t)$$

Здесь $\dot{\mathbf{A}}$ — Advantage-оценка критика менеджера (оцениваемая стандартным для Policy Gradient методом через обучение единственной функции \dot{V}), а зависимость $z_t, z_{t+K}, \dot{\mathbf{A}}$ от параметров θ игнорируется.

Награда для рабочего в задаче следования тем направлениям, которые указывает менеджер, может быть сформулирована так:

$$r_t^g := \frac{1}{K} \sum_{i=0}^K d(z_t - z_{t-i}, g_{t-i}),$$

то есть для каждого из последних K шагов проверяется, насколько точно рабочий следует указаниям менеджера, смещается ли в латентном пространстве менеджера описание состояния в указанном им направлении. Эта награда смешивается с наградой из исходного MDP; можно считать, что r_t^g для рабочего — внутренняя мотивация, а истинная награда из среды — внешняя мотивация.

8.4.9. HIRO

Алгоритм HIRO в целом похож на FuN, хотя общая схема иерархической стратегии выглядит чуть-чуть по-другому. В этом алгоритме авторы рассматривали задачи непрерывного управления и могли для простоты положить $\mathcal{G} \equiv \mathcal{S}$, не переводя состояния в латентное пространства. Менеджер выбирает цель $\mathbf{g}_t \in \mathcal{S}$ раз в K шагов, где K — гиперпараметр. В последующие $K - 1$ моментов времени цели определяются по предыдущей по следующему «векторному» правилу:

$$g_{t+1} - s_{t+1} = g_t - s_t \quad \Rightarrow \quad g_{t+1} = g_t + s_{t+1} - s_t$$

Другими словами, если на шаге t менеджер решил, что состояние s_t должно меняться «в направлении» \mathbf{g}_t , то дальние следующие K шагов мы считаем, что хотим этому направлению следовать. Вызов раз в K шагов кажется костылём, но позволяет избежать альтернативы с эвристикой FuN, в котором мы при обучении менеджера предполагали, что рабочий успешно выполняет возложенную на него задачу; тут же менеджера можно обучать любовым подходом: он живёт даже не в sMDP, а обычном MDP, поскольку всегда гарантируется, что выбранное макро-действие будет выполняться в точности K шагов. Однако, поскольку рабочие будут постепенно обучаться, это MDP нестационарное.

Внутренней мотивацией рабочего, выполняющего цель \mathbf{g} , полагается расстояние от состояния, в которое попал агент, до целевого состояния $s_t + \mathbf{g}_t$:

$$r^{\text{intr}}(s_t, \mathbf{g}_t, s_{t+1}) = -\|s_{t+1} - (s_t + \mathbf{g}_t)\|_2$$

Как и в FuN, такая внутренняя мотивация смешивается со внешней мотивацией — наградой из основного MDP.

Заметим, что рабочего можно спокойно обучать в off-policy режиме (он живёт в самом обычном MDP). Хотелось бы научиться обучать в off-policy режиме и менеджера, но понятно, что пространство действий \mathcal{G} для менеджера с ходом обучения меняет своё семантическое значение. Предлагается лайфхак, очень похожий на

идею переразметки траекторий из multi-task RL (см. раздел 8.3.3). Сохраним для менеджера в реплей буфера для s_t, g_t всю последующую траекторию $a_t, s_{t+1} \dots a_{t+K-1}, s_{t+K}$. Допустим, мы засэмплировали этот переход из буфера и хотим использовать для обучения. Проблема в том, что π^g для g из перехода уже может быть совершенно другим, и вероятность, что он сгенерировал бы такую траекторию, неприемлемо низкая. Идея: попробуем найти другое $\hat{g} \in \mathcal{G}$, для которого эта вероятность довольно высокая. То есть, мы приблизительно попытаемся решить следующую задачу:

$$\log p(a_t, s_{t+1} \dots a_{t+K-1}, s_{t+K} | s_t, \hat{g}) \rightarrow \max_{\hat{g}}$$

Рассмотрим, из чего состоит правдоподобие. Мы предполагаем, что менеджер делает выбор в момент времени t , то есть менеджер больше на это правдоподобие не влияет. Есть логарифмы переходов $p(s' | s, a)$, которые не зависят от \hat{g} , поэтому их можно опустить. Остаются только правдоподобия выборов действий:

$$\sum_{t=t}^{t+K-1} \log \pi_{\hat{g}}(a_t | s_t) \rightarrow \max_{\hat{g}}$$

Почти всегда мы можем для данного \hat{g} посчитать значение выражения. Предлагается взять несколько (штук восемь–девять) g , посчитать значение выражения и выбрать наилучшее.

Какую хорошую стратегию перебора для сэмплирования кандидатов g выбрать? Поскольку агент в итоге сдвинулся на вектор $s_{t+K} - s_t$, то можно использовать этот вектор в качестве центра гауссианы, из которой будет проводиться сэмплирование. Также предлагается попробовать взять g из буфера и сам центр $s_{t+K} - s_t$.

§8.5. Частично наблюдаемые среды

8.5.1. Частично наблюдаемые MDP

Рассматривавшиеся MDP до этого являлись **полностью наблюдаемыми** (fully observable): агенту в качестве наблюдения было доступно всё состояние целиком. Конечно, это довольно-таки существенное упрощение.

Определение 118: MDP называется **частично наблюдаемым** (partially observable, принятое сокращение — PoMDP), если дополнительно задано множество \mathcal{O} , называемое **пространством наблюдений** (observation space), и распределение $p(o | s)$, определяющая вероятность получить то или иное наблюдение агента $o \in \mathcal{O}$ в момент времени, когда мир находится в состоянии $s \in \mathcal{S}$.

Если MDP — «управляемая» марковская цепь, то PoMDP можно рассматривать как «управляемую» **скрытую марковскую цепь**: действия влияют на то, как будут порождаться следующие состояния, но для наблюдения доступны не сами состояния, а только какие-то другие случайные величины, про которые, однако, известно, что они зависят только от текущего состояния. При этом состояния всё также удовлетворяют свойству марковости, а функция переходов и процесс генерации наблюдений по состоянию — свойству стационарности.

В PoMDP стратегия должна уметь принимать решение на основании не только текущего наблюдения, но всей истории цепочки наблюдений, так как в них может содержаться информация о текущем состоянии среды. Естественно, эта задача более приближена к реальности, но сильно сложнее. Агенту теперь нужна модель **памяти** (memory), способ хранения и учёта всей истории в течение эпизода.

Заметим, что награда по своему определению наблюдаема. Если раньше можно было считать, что награда — часть состояния, то теперь награда «по логике» модели есть часть наблюдений, а значит, формально может рассматриваться как функция от наблюдений. Далее будем работать в соглашении, что награда — детерминированная функция от наблюдений.

8.5.2. Belief MDP

Допустим, задано PoMDP, и нам доступны все распределения (вероятности переходов и вероятности наблюдений). Пусть мы знаем вероятность $p(s_0)$, какое состояние генерируется изначально (будем считать, стохастично). Допустим, начался новый эпизод, и мы получили наблюдение $o_0 \in \mathcal{O}$. Что мы можем сказать о том, в каком состоянии s_0 мы на самом деле оказались? Ответ на этот вопрос формально даёт формула Байеса:

$$p(s_0 | o_0) = \frac{p(o_0 | s_0)p(s_0)}{\int_{\mathcal{O}} p(o_0 | s_0)p(s_0) d\omega} \quad (8.39)$$

Здесь в числителе стоит вероятность первого наблюдения и априорное распределение на состояниях, а в знаменателе стоит нормировочная константа. Пока будем считать, что мы умеем брать любые интегралы (например, если пространства состояний и наблюдений конечны и малы).

Допустим, мы выбрали действие a_0 , попали в состояние s_1 (которое мы не видим) и получили наблюдение o_1 . Что мы можем сказать о том, в каком состоянии s_1 мы находимся? Опять же, пользуясь формулой Байеса и используя вероятностную модель PoMDP, мы можем получить точную формулу наших представлений о том,

в каком состоянии на текущий момент пребывает агент. Чтобы проделать вывод, обозначим за $\mathcal{T}_{:t}$ наблюдаемую (действия, награды, наблюдения, но не состояния) траекторию до момента времени t , включая последнее наблюдение:

$$\mathcal{T}_{:t} := (o_0, a_0, r_0, o_1, a_1, r_1 \dots o_{t-1}, a_{t-1}, r_{t-1}, o_t)$$

Определение 119: Вероятность пребывания в том или ином состоянии при условии наблюдаемой истории $p(s_t | \mathcal{T}_{:t})$ называется *belief state* и сокращённо обозначается b_t .

По определению $b_0(s_0) = p(s_0 | o_0)$, которую мы получили в по формуле Байеса в (8.39). Как обновить belief state после совершения одного шага в среде (получения одного наблюдения и выбора одного действия)?

Теорема 91: С точностью до нормировочной константы:

$$b_t(s_t) \propto p(o_t | s_t) \int_{\mathcal{S}} p(s_t | s_{t-1}, a_{t-1}) b(s_{t-1}) ds_{t-1} \quad (8.40)$$

Доказательство. Нашей целью является посчитать $b_t(s_t) := p(s_t | \mathcal{T}_{:t}) = p(s_t | o_t, a_{t-1}, \mathcal{T}_{:t-1})$. Применим формулу Байеса:

$$b_t(s_t) \propto p(o_t | s_t, a_{t-1}, \mathcal{T}_{:t-1}) p(s_t | a_{t-1}, \mathcal{T}_{:t-1}) = (*)$$

Первый множитель здесь по определению вероятностной модели есть $p(o_t | s_t)$, а во втором множителе, чтобы воспользоваться функцией переходов, нужно проинтегрировать по неизвестному нам предыдущему состоянию s_{t-1} :

$$\begin{aligned} (*) &= p(o_t | s_t) \int_{\mathcal{S}} p(s_t, s_{t-1} | a_{t-1}, \mathcal{T}_{:t-1}) ds_{t-1} = \\ &= p(o_t | s_t) \int_{\mathcal{S}} p(s_t | s_{t-1}, a_{t-1}) p(s_{t-1} | \mathcal{T}_{:t-1}) ds_{t-1} \end{aligned}$$

Осталось заметить, что по определению $p(s_{t-1} | \mathcal{T}_{:t-1}) = b_{t-1}(s_{t-1})$ — информация о текущем состоянии с прошлого шага. ■

Пример 138: Допустим, в PoMDP с 10 состояниями и действиями вправо-влево получаем детерминировано в качестве наблюдения бинарный флаг: есть ли пальма или нет. Начальное состояние определяется случайно равномерно. Посмотрим, как меняется belief state, если агент заспавнился в самом левом состоянии и дальше выбирает действия «вправо».

Belief state — «достаточная статистика» для описания всей предыдущей истории. Посчитав его, мы собрали абсолютно всю имеющуюся у нас информацию. Это замечание позволяет формально свести задачу в PoMDP к обычному MDP, но требует знания всех распределения и возможности пересчитывать belief state:

Определение 120: Для данного PoMDP *Belief MDP* называется следующее MDP:

- Пространством состояний является пространство $\mathbf{P}(\mathcal{S})$ распределений в \mathcal{S} ;
- Пространством действий является \mathcal{A} ;
- Функция переходов $p(b' | b, a)$, где $b, b' \in \mathbf{P}(\mathcal{S})$ устроена так: сэмплируются $s \sim b(s)$, $s' \sim p(s' | s, a)$, $b' \sim p(o' | s')$, после чего новый belief state рассчитывается по формуле:

$$b'(s') \propto p(o' | s') \int_{\mathcal{S}} p(s' | s, a) b(s) ds$$

- Функция награды для текущего состояния b и действия a устроена так: для того же сэмпла s , использованного в переходе, выдаётся награда $r(s, a)$.

Теория Belief MDP позволяет предложить алгоритмы динамического программирования наподобие Value Iteration и Policy Iteration для решения задач в PoMDP. Однако, в реальности работать с Belief MDP в средах со сложным пространством состояний довольно неудобно: если все распределения неизвестны, интегралы в формуле обновления (8.40) не берутся, и даже просто хранить «распределение над состояниями» невозможно, то не совсем понятно, что можно извлечь от этой теории в сложных средах. Понятно, что при взаимодействии со средой агент должен помимо прочего стремиться выполнять те действия, которые дают наибольшую информацию о belief state, и позволяют «локализоваться» в пространстве состояний.

8.5.3. Рекуррентные сети в Policy Gradient

Видимо, поэтому пока самое распространённое решение для работы в PoMDP — добавление в архитектуры моделей рекуррентных слоёв (GRU или LSTM). При использовании рекуррентных стратегий скрытое состояние инициализируется нулевым вектором в начале каждого эпизода и «хранит всю необходимую информацию» о предыдущих наблюдениях, а все модели на вход получают лишь то, что поступает с сенсоров. Де-факто все наши модели оценочных функций и стратегии становятся функциями от всей истории $o_0, o_1, o_2 \dots o_t$. Обсудим, как это меняет ранее встречавшиеся алгоритмы.

Чистые on-policy алгоритмы, такие как REINFORCE, A2C и мета-эвристики, практически не меняются. В A2C лишь стоит иметь в виду, что градиенты текут только по тому роллауту, который собран на текущей итерации; таким образом, агенту будет тяжело научиться запоминать информацию дольше, чем на N шагов, где N — длина роллаутов.

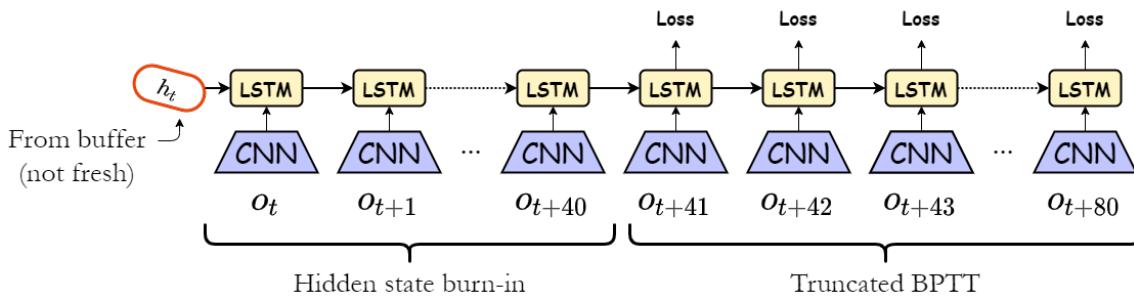
В PPO скрытое состояние модели сохраняется для всего собранного датасета. Далее из датасета сэмплируются не мини-батчи, а мини-батчи роллаутов (некоторой длины N). Затем скрытое состояние инициализируется сохранённым значением при сборе датасета (поскольку стратегия «не меняется сильно», считается, что это сохранённое состояние более-менее не устарело), и модель прогоняется на засэмплированном роллауте для подсчёта всех градиентов.

8.5.4. R2D2

В off-policy алгоритмах сохранять значение скрытого состояния в реплей буфере на первый взгляд бессмысленно; модель может изменится сколько угодно сильно, и сохранённые значения памяти будут неактуальны. С течением обучения скрытое состояние рекуррентных сетей просто меняет своё семантическое значение: модель со свежими весами просто по-другому будет интерпретировать латентное представление.

Рассмотрим две эвристики для преодоления этой проблемы из алгоритма R2D2. Из реплей буфера генерируются роллауты некоторой длины N . Скрытое состояние инициализируется сохранённым в буфере значением (которое потенциально «протухло»). Модель прогоняется заново на всём роллауте, но функция потерь и backpropagation считается лишь на последних $M < N$ шагов. Таким образом обучение проводится стандартным для рекуррентных сетей подходом backpropagation through time (BPTT).

Первые же $N - M$ шагов нужны исключительно для «**прогрева**» (burn-in) скрытого состояния, чтобы его семантическое значение начало более-менее соответствовать свежим весам модели. Это не так долго вычислительно, как заново прогонять модель по всему эпизоду; так можно было бы получить «честные» актуальные значения памяти, но параллелизовать такие вычисления по батчу, в котором встречаются роллауты из разных моментов эпизодов, неудобно и достаточно дорого. Естественно, аналогичный прогон проводится для таргет-сети.



Вторая эвристика заключается в том, чтобы «разогретое» состояние после $N - M$ шагов сохранить в буфере для того роллаута, который начинается с соответствующего перехода, и таким образом сделать сохранённое значение чуть более актуальным.

Как обычно, длина M той части траектории, вдоль которой пропускаются градиенты, и ограничивает то время, «на которое» агент может теоретически научиться что-то запоминать. Однако, увеличивать его может быть чревато не только тем, что алгоритм станет вычислительно сложным, но и тем, что в батче оказываются

M очень похожих друг на друга переходов, и перебивать эту коррелированность лоссов придётся размером мини-батча.



Может быть удобно для ускорения сэмплирования хранить в реплей буфере эпизоды, уже разбитые на фрагменты удобного размера. Например, авторы R2D2 выбирают $N = 80$, $M = 40$, и при сборе данных собирают 40 последовательных переходов, которые упаковываются в буфер единственным комплектом; из-за такой оптимизации промежуточные переходы в этом роллауте никогда не будут засэмплированы для обучения как, например, стартовые (с которых начинается роллаут из N шагов). Похоже, что это не так критично, зато сильно упрощает код и сэмплирование мини-батчей.

8.5.5. Neural Episodic Control (NEC)

Эпизодичная память означает, что вместо (или дополнительно к) рекуррентных связей в качестве памяти используется набор ранее встретившихся состояний или их латентных описаний. Механизм работы с такой памятью очень схож с механизмом внимания. Рассмотрим алгоритм Neural Episodic Control в качестве примера применения такой идеи.

Для каждого действия a будем хранить **словарь** (dictionary), то есть некоторый набор ключ-значение $M_a := (k, v)$. **Ключом** (key) $k \in \mathbb{R}^d$ является вектор-эмбеддинг, описывающий некоторое состояние s из опыта агента. **Значением** (value) v является скаляр, приближённо оценивающий $Q^*(s, a)$.

При взаимодействии со средой агент действует следующим образом. Текущее состояние s подаётся на вход сети с параметрами θ , которая выдаёт его описание $q_\theta(s) \in \mathbb{R}^d$. Выходом $Q_\theta(s, a)$ модели является результат применения **механизма внимания** (attention) к словарю M_a с **запросом** (query) $q_\theta(s)$: для каждого ключа k в словаре определяется похожесть запроса на ключ $\rho(q_\theta(s), k)$ при помощи некоторой функции близости ρ , например:

$$\rho(q, k) := \frac{1}{\|q - k\|_2^2 + \delta},$$

где δ — небольшая константа для защиты от деления на ноль. Затем для каждой пары ключ-значение определяется его вес:

$$w(k) \propto \rho(q, k),$$

где нормировочная константа вычисляется из соображения $\sum_{k, v \in M_a} w(k) = 1$. Наконец, выход дифференцируемого чтения из словаря определяется как сумма значений с вычисленными весами:

$$Q_\theta(s, a) := \sum_{k, v \in M_a} w(k)v$$

Поскольку размер словаря может быть достаточно большой, а процедуру нужно проводить для каждого $a \in \mathcal{A}$, предлагается использовать в формуле две аппроксимации. Во-первых, учитываются только топ-50 самых значимых (имеющих наибольший вес $w(k)$) пар ключей-значений. Иными словами, по расстоянию ρ находятся 50 ближайших к $q_\theta(s)$ ключей в словаре, и веса с учётом нормировочной константы вычисляются только для них; только они используются для вычисления финальной суммы. Во-вторых, поиск топ-50 ближайших соседей проводится приближённо, для ускорения вычислений.

Видно, что при фиксированном словаре M_a функция $Q_\theta(s, a)$, вычисленная таким образом, дифференцируема по параметрам.

При помощи такой модели Q-функции агент выбирает действие ϵ -жадно и получает для выбранного действия a следующее состояние s' и награду за шаг r . Теперь он может посчитать для примера s, a целевую переменную (авторы используют N -шаговую оценку, для чего нужно провести N шагов взаимодействия со средой; для простоты положим $N = 1$):

$$\hat{Q}(s, a) = r + \gamma \max_{a'} Q_\theta(s', a')$$

где значения Q_θ также получены проведением всей вышеописанной процедуры. Полученная пара $(q_\theta(s), \hat{Q}(s, a))$ добавляется в словарь M_a и используется при дальнейших проходах через сеть. У словарей, естественно, есть некоторое ограничение по объёму, и при добавлении новой пары в заполненный словарь предлагается выкидывать самую редко используемую пару (k, v) : ту, для которой число попаданий в топ-50 ближайших соседей наименьшее (для этого достаточно дополнительно хранить счётчики использований).

Тройка $(s, a, \hat{Q}(s, a))$ добавляется в реплей буфер. На этапе обучения, который проводится после каждого шага взаимодействия со средой, сэмплируется мини-батч троек $(s, a, \hat{Q}(s, a))$ из буфера, и модель с текущими словарями M_a прогоняется на s, a для получения оценки $Q_\theta(s, a)$:

$$(Q_\theta(s, a) - \hat{Q}(s, a))^2 \rightarrow \min_{\theta, k, v}$$

Запись, что минимизация ведётся как по θ , так и по k, v , означает, что минимизация ведётся не только по параметрам сети, но и по ключам и значениям, хранящимся на текущий момент в словаре M_a . Таким образом, эмбеддинги и значения обновляются в том числе для хранящихся в «эпизодичной памяти» состояний.

Заметим, что $\hat{Q}(s, a)$ здесь — целевая переменная, посчитанная в момент сбора перехода, и она не пересчитывается; иначе говоря, используется по сути on-policy режим обучения. Интерпретация такого хода в том, что отказ пересчитывать значение целевой переменной $\hat{Q}(s, a)$ по сути эквивалентен использованию замороженной таргет-сети; но, чтобы эти значения не устаревали сильно, размер реплей буфера придётся существенно сократить (авторы используют реплей буфер размера 10^5 , что на порядок меньше типично используемого буфера при off-policy обучении).

§8.6. Мульти-агентное обучение с подкреплением

8.6.1. Связь с теорией игр

Попробуем представить ситуацию, когда в среде действует два агента. Функция переходов теперь зависит от действий каждого из агентов $p(s' | s, a^1, a^2)$, а каждый агент оптимизирует свою функцию награды $r^i(s, a^1, a^2)$, где i — номер агента.

Рассмотрим сеттинг, аналогичный бандитам: игра с одним состоянием, заканчивающаяся после первого выбора. Допустим также детерминированность функций наград. Мы получим в чистом виде задачу, исследующуюся в *теории игр*: заданы две функции $r^1(a^1, a^2), r^2(a^1, a^2)$, зависящие от действий обоих игроков; игроки оптимизируют каждый свою функцию, при этом производя выбор действий одновременно. Отсюда видно, что любой мульти-агентный RL тесно связан с теорией игр: в частности, при обучении агенты могут застрять не только в локальных оптимумах, но и в **равновесиях Нэша** (Nash equilibrium), когда ни один другой агент не может поменять свою стратегию в предположении неизменности стратегий других агентов так, чтобы увеличить свою награду.

В общем случае в среде действует N агентов. Будем обозначать как a^i действие i -го агента; \vec{a} — вектор из всех действий всех агентов; a^{-i} — действия всех агентов, кроме i -го. Тогда функция переходов теперь выглядит как $p(s' | s, \vec{a})$, а i -ый агент оптимизирует свою функцию награды $r^i(s, \vec{a})$. Природа этих функций наград может быть самой разной.

Определение 121: Если функции наград всех агентов совпадают, то задача называется **кооперативной**.

Определение 122: Задача, в которой сумма наград всех агентов за шаг нулевая $\sum_i r^i(s, \vec{a}) = 0$, называется **антагонистической** или **игрой с нулевой суммой**.

Это предельные случаи частных ситуаций: если r^1 в целом велико там, где велико r^2 , то эти два агента, видимо, должны научиться вместе выполнять какую-то задачу. Если же большое r^1 влечёт малое r^2 , то агенты противостоят друг другу. В общем случае у каждого агента может быть какая-то своя задача; этот общий случай иногда называют **смешанной** (mixed) задачей мульти-агентного RL. Зачастую агенты действуют в условиях частичной наблюдаемости, и тогда у каждого агента могут быть свои наблюдения; это означает, что у i -го агента своя функция $p^i(o | s)$, определяющая наблюдение по текущему состоянию мира. Для простоты далее частичная наблюдаемость опущена.

8.6.2. Централизация обучения

Децентрализованный сеттинг означает, что у агентов нет какой-либо общего «сервера» с общими данными, в том числе во время обучения. Простейший подход мульти-агентного RL в рамках такого сеттинга — завести для каждого агента свою стратегию $\pi^i(a | s)$ и оценочную функцию $Q^i(s, a^i)$, которая будет зависеть только от действия i -го агента, и оптимизировать их с опыта данного агента обычными RL алгоритмами. Тот факт, что в среде действуют другие агенты, игнорируется. Важный момент: если другие агенты не обучаются (не изменяются с течением времени), то такой подход обучения i -го агента корректен.

Утверждение 91: Если все агенты, кроме i -го, стационарны, то i -ый агент живёт в MDP с функцией переходов

$$p(s' | s, a^i) := \int p(s' | s, a^i, a^{-i}) \pi^{-i}(a^{-i} | s) da^{-i},$$

где $\pi^{-i}(a^{-i} | s)$ — стратегия остальных агентов, в предположении их независимого выбора своих действий равная

$$\pi^{-i}(a^{-i} | s) := \prod_{j \neq i} \pi^j(a^j | s)$$

Это означает, что если в среде действует другой агент, использующий стационарную стратегию, среда остаётся стационарной и вся теория остаётся справедлива. Однако, если в среде действует другой агент, который обучивается (и, значит, меняет свою стратегию с течением времени), то для прочих агентов это означает, что постоянно изменяются «законы физики» окружающей среды, и мы под изученную теорию подпадать перестаём.

В нестационарных средах естественно использовать on-policy обучение. Запуск какого-нибудь on-policy алгоритма с игнорированием факта присутствия других обучающихся агентов обычно оказывается серьёзным бэйзлайном.

Полностью централизованный сеттинг возможен для агентов, например, оптимизирующих одну и ту же функцию награды, находящихся в кооперации. Полная централизация означает, что для построения системы из нескольких кооперирующихся агентов все они рассматриваются как единый агент. Наблюдение есть совокупность наблюдений всех агентов, действия — набор действий для каждого из агентов, и тогда по сути задача сведена к «одноагентной» ситуации. В большинстве случаев такой вариант не годится по самой постановке задачи: наличие единого «центрального сервера», который бы принимал решения за всех агентов в среде, может быть просто-напросто очень дорогим. Хотя бы потому что агентов может быть много, и пространство состояний и, главное, действий, взорвётся. В большинстве случаев хочется, чтобы агенты принимали решения на основе только имеющейся у них информации.

Зачастую доступен интересный промежуточный вариант (**частичная централизация**), когда у каждого из кооперирующихся агентов хранится и обучается персональная стратегия, но, например, оценочная функция $Q(s, a^1, a^2 \dots)$ хранится и обучается на общем для всех агентов «сервере». Важно, что эта оценочная функция нужна в on-policy алгоритмах только для обучения, и при использовании полученных стратегий «сервер» не потребуется. Это означает, что по окончании обучения в таком сеттинге для использования стратегии никакого центрального сервера не понадобится. Такой подход называется **centralised training with decentralised execution** (CTDE), и многие алгоритмы для мульти-агентного RL строятся именно в рамках этой парадигмы.

В рамках парадигмы CTDE, помимо прочего, возможен **parameter sharing**: если задача симметрична для каких-то агентов, то веса их стратегий считаются общими. Градиенты для их обучения тогда вычисляются на общем сервере; дальше во время использования агенты используют одну и ту же модель для принятия решений. Другими словами, в рамках CTDE подхода при желании обучить колонию из 100 одинаковых муравьёв, вам понадобится лишь одна модель; без частичной централизации же, в децентрализованном сеттинге, подразумевается, что каждый из муравьёв должен как-то сам обучиться, и тогда у каждого из сотни получится в итоге своя модель.

8.6.3. Self-play в антагонистических играх

В симметричных антагонистических играх обучение на опыте игр с самим собой оказалось мощным инструментом. Этот подход называют **self-play**, и заключается он в том, что за игроков играет одна и та же стратегия (такие игры обычно симметричны, и поэтому можно считать, что применяется parameter sharing). Собранный опыт со стороны каждого игрока можно использовать для обучения, и он обладает одним очень ценным свойством: сбалансированностью. На какой бы стадии обучения ни был агент, играет ли он на уровне гранд-мастера или бесконечно сильно тупит, при игре с самим собой в половине случаев он «выигрывает», в половине — «проигрывает». Разница в сигнале всегда даёт возможность дальнейшего улучшения.

В режиме self-play можно запустить любые алгоритмы обучения: как model-free, так и model-based. В частности, в model-based алгоритмах вроде AlphaGo и MuZero в ходе планирования можно при построении дерева ходы за оппонента проводить при помощи тех же моделей, но не максимизировать, а минимизировать награду. Понятно, что идея применима как в ситуации, когда игроки ходят одновременно, так и когда игроки ходят по очереди (что более типично для игр вроде шахмат и го).

Конечно, self-play не гарантирует, что по итогам обучения стратегия будет способна адаптироваться к произвольному оппоненту. Можно показать, что к обученным self-play методам стратегии можно применить **adversarial-атаку**: зафиксировать её и обучать стратегию оппонента побеждать. Такие стратегии, которые «взламывают» стратегию оппонента, зачастую ведут себя очень странно, и могут победить, ничего не делая.

Пример 139 — Adversarial Policy: По [данной ссылке](#) в примерах слева показаны игры стратегии, обученной в режиме self-play. На вид кажется, что полученная стратегия невероятно крута. Далее эта стратегия фиксировалась, и обучалась с нуля новая стратегия оппонента «побеждать» фиксированную. Примеры этих игр показаны справа: новая стратегия оппонента просто падает, сбивает с толку «крутую» стратегию, и та отправляется в какой-то нокаут.

8.6.4. QMix в кооперативных играх

Рассмотрим кооперативный сеттинг, где все агенты максимизируют одну и ту же функцию награды. Будущую кумулятивную награду после выполнения набора действий \vec{a} в состоянии s для набора политик $\pi^i(a^i | s)$ обозначим как $Q^{\text{tot}}(s, \vec{a})$, скаляр. Попробуем построить алгоритм в рамках парадигмы CTDE, то есть обучить «на центральном сервере» стратегии $\pi^i(a^i | s)$, которые смогут дальше действовать в среде без использования подобного сервера.

Допустим, мы знаем $Q^{\text{tot}}(s, \vec{a})$. Тогда оптимально выбрать действия

$$\underset{\vec{a}}{\operatorname{argmax}} Q^{\text{tot}}(s, \vec{a}),$$

однако есть две проблемы. Во-первых, если агентов много, поиск такого аргмаксимума может стать слишком сложной задачей, ровно как и моделирование такой оценочной функции. Во-вторых, в рамках CTDE в частично наблюдаемом сеттинге каждый агент на вход вместо s на самом деле получает своё собственное наблюдение a^i и не знает наблюдения других агентов, когда Q^{tot} принимает на вход наблюдения всех агентов. С обоими проблемами можно побороться следующим образом.

Определение 123: *Смешивающей сетью* (mixing network) назовём моделирование $Q^{\text{tot}}(s, \vec{a})$ в следующем виде:

$$Q^{\text{tot}}(s, \vec{a}) \approx Q_{\phi}^{\text{tot}}(Q^1(s, a^1), Q^2(s, a^2), \dots, Q^N(s, a^N), s), \quad (8.41)$$

где ϕ — параметры, $Q^i(s, a^i)$ — скаляры, зависящие только от той информации, которая доступна i -му агенту.

В частности, в частично наблюдаемом сеттинге $Q^i(s, a^i)$ вместо s принимает на вход наблюдение i -го агента a^i и моделируется рекуррентной сетью. Сразу оговоримся, что здесь $Q^i(s, a^i)$, несмотря на принятое обозначение, не является оценочной функцией и не имеет смысла будущей кумулятивной награды. Это лишь некоторая промежуточная вспомогательная величина, через которую по некоторым правилам выражается $Q^{\text{tot}}(s, \vec{a})$, награда «всей команды».

Пример 140 — Value decomposition network: Рассмотрим самый простой пример смешивающей сети, не использующей параметров ϕ вовсе:

$$Q_{\phi}^{\text{tot}}(s, \vec{a}) := \sum_i Q^i(s, a^i)$$

У этой декомпозиции есть интересное свойство:

$$\underset{\vec{a}}{\operatorname{argmax}} Q^{\text{tot}}(s, \vec{a}) = \begin{bmatrix} \underset{a^1}{\operatorname{argmax}} Q^1(s, a^1) \\ \underset{a^2}{\operatorname{argmax}} Q^2(s, a^2) \\ \vdots \\ \underset{a^N}{\operatorname{argmax}} Q^N(s, a^N) \end{bmatrix} \quad (8.42)$$

Другими словами, будущая награда команды моделируется через какие-то псевдооценочные функции каждого агента команды в отдельности. Нам крайне интересно свойство (8.42).

Определение 124: Скажем, что смешивающая сеть **консистентна** (consistent), если для всех состояний выполняется (8.42).

Консистентность означает, что каждому агенту вовсе не нужно знать наблюдения других агентов или обращаться к центральному серверу для выбора оптимального действия: он берёт лишь значение своей псевдооценочной функции $Q^i(s, a^i)$ и считает аргмакс по нему. Свойство (8.42) гарантирует, что это даст аргмаксимум по всему набору действий \vec{a} всей команды. Поскольку выбор декомпозиции — наш произвол, нам хочется выбрать такое параметрическое семейство смешивающих сетей, которое гарантирует это свойство. В целом, это не так сложно.

Теорема 92: Смешивающая сеть (8.41) консистентна, если

$$\frac{dQ_{\phi}^{\text{tot}}}{dQ^i} \geq 0, \quad (8.43)$$

то есть для всех состояний s и при любых значениях параметров ϕ награда команды монотонно зависит от значений псевдооценочных функций агентов $Q^i(s, a^i)$.

Доказательство. В силу монотонности, для любых действий \vec{a} :

$$Q_{\phi}^{\text{tot}}(\max_{a^1} Q^1(s, a^1), \dots, \max_{a^N} Q^N(s, a^N), s) \geq Q_{\phi}^{\text{tot}}(Q^1(s, a^1), \dots, Q^N(s, a^N), s),$$

следовательно максимизация всех аргументов монотонной функции максимизирует саму функцию. ■

Можно ли придумать какое-нибудь богатое семейство смешивающих сетей с параметрами, для которых было бы выполнено (8.43)? Давайте возьмём нейросеть с параметрами ϕ , которая принимает на вход скаляры $Q^1(s, a^1), \dots, Q^N(s, a^N)$ и выдаёт скаляр Q_{ϕ}^{tot} . Тогда:

Утверждение 92: Пусть нейросеть полностью связная, состоит из чередования линейных слоёв и монотонных функций активаций, и все параметры нейросети ϕ_i (кроме, возможно, смещений в линейных слоях) неотрицательны. Тогда выполнено (8.43).

Доказательство. Композиция монотонных преобразований монотонно; линейная комбинация с положительными весами (и произвольными смещениями) также монотонно не убывает как функция от входов. ■

Итак, любая нейросеть с положительными весами подходит нам в качестве смешивающей. Сделаем ещё одно наблюдение: эта нейросеть очень маленькая, ведь на вход ей подаётся вектор размерности N , где N — число агентов, а на выходе скаляр. Заметим также, что веса могут зависеть от состояния: мы можем использовать разные ϕ для разных состояний s , и отсюда возникает идея, как можно повысить гибкость нашей value decomposition network.

Определение 125: Гиперсетью (hypernetwork) называется нейросеть, выдающая веса для другой нейросети.

Мы заведём гиперсеть $\phi_\theta(s)$, которая для данного состояния s будет с параметрами θ выдавать на центральном сервере веса для смешивающей сети. В частности, $\phi_\theta(s)$ уже не обязано быть монотонным и может моделироваться произвольной обычной нейросетью. Мы получим, что в каждом состоянии у нас есть какое-то своё монотонное преобразование псевдооценочных функций каждого агента в оценочную функцию команды, и это очень удобно.

Итого в алгоритме QMIX оценочная функция моделируется в следующем виде:

$$Q_{\phi_\theta(s)}^{\text{tot}}(s, \vec{a}, \psi) := Q_{\phi_\theta(s)}^{\text{tot}}(Q^1(s, a^1, \psi^1), \dots, Q^N(s, a^N, \psi^N)),$$

где ψ^i — параметры псевдооценочной функции i -го агента (возможен parameter sharing, если задача симметрична для некоторых агентов), θ — параметры гиперсети, $\phi_\theta(s)$ — получающиеся параметры смешивающей нейросети.

Процесс обучения стандартный и похож на обычный DQN. При взаимодействии со средой i -ый агент использует свою псевдооценочную функцию $Q^i(s, a^i)$ и ведёт себя ε -жадно. Собранные переходы (s, \vec{a}, r, s') собираются на центральном сервере. Для перехода $\mathbb{T} := (s, \vec{a}, r, s')$ целевая переменная строится как

$$y(\mathbb{T}) := r + \gamma \max_{\vec{a}'} Q_{\phi_{\theta^-}(s)}^{\text{tot}}(s', \vec{a}', \psi^-),$$

где θ^-, ψ^- — замороженные параметры таргет-сети. Далее с таким таргетом минимизируется MSE по всем параметрам:

$$\mathbb{E}_{\mathbb{T}}(y(\mathbb{T}) - Q_{\phi_\theta(s)}^{\text{tot}}(s, \vec{a}, \psi))^2 \rightarrow \min_{\theta, \psi}$$

После окончания обучения центральный сервер, на котором хранится гиперсеть и смешивающая сеть, не нужны, и агенты используют лишь локальные псевдооценочные функции $Q^i(s, a^i)$.

8.6.5. Multi-Agent DDPG (MADDPG) в смешанных играх

Рассмотрим смешанную игру, где функции наград агентов произвольны. На «центральном сервере» будем хранить общую оценочную функцию, которая теперь должна выдавать будущие награды сразу для всех агентов.

Определение 126: При данных политиках $\pi^j(a_j | s)$ централизованной оценочной функцией (centralized state-action value function) обозначим $\vec{Q}(s, \vec{a})$, возвращающую вектор, i -ая компонента которого $Q^i(s, \vec{a})$ равна будущей кумулятивной награде, которую получит i -ый агент после выполнения набора действий \vec{a} в состоянии s .

На центральном сервере понятно, как учить такую Q-функцию, причём это можно делать в off-policy режиме. Для произвольного перехода из буфера $\mathbb{T} := (s, \vec{a}, \vec{r}, s')$ строим таргет

$$y(\mathbb{T}) := \vec{r} + \gamma \vec{Q}(s', \vec{a}'),$$

где \vec{a}' порождены оцениваемыми $\pi^j(a'_j | s')$ (для стабилизации процесса — таргет-сетями для них), и минимизируем стандартное MSE:

$$\mathbb{E}_{\mathbb{T}}(\vec{Q}(s, \vec{a}) - y(\mathbb{T}))^2 \rightarrow \min_{\vec{Q}}$$

Процесс необходимо проводить на центральном сервере, поскольку для вычисления таргета необходимы текущие стратегии всех агентов. Вообще, от этого ограничения можно избавиться, если агентам в их наблюдениях доступны действия всех остальных агентов. Тогда мы сможем перейти к децентрализованному обучению следующим образом: i -ый агент может локально хранить и обучать лишь интересующую его компоненту $Q^i(s, \vec{a})$. При построении таргета нужны стратегии оппонентов; их i -ый агент будет **моделировать**. Для моделирования собираются наблюдения (s, a^{-i}) , то есть какие действия предприняли остальные агенты в состоянии s , и дальше агент учится предсказывать действия других игроков по этой обучающей выборке в supervised-режиме:

$$\mathbb{E}_{s, a^{-i}} \log \mu(a^{-i} | s) \rightarrow \max_{\mu}$$

обычно с добавлением энтропийного регуляризатора. Имея такое приближение стратегий оппонентов на руках, таргет для обучения критика рассчитывается по формуле

$$y^i(\mathbb{T}) := r^i + Q^i(s', a'^i, a'^{-i}),$$

где $a'^i \sim \pi^i(a'^i | s')$, а $a'^{-i} \sim \mu(a'^{-i} | s')$.

Как использовать централизованную оценочную функцию для обучения актёров? Для обучения i -го актёра с параметрами θ^i можно применить формулу policy gradient:

$$\nabla_{\theta^i} = \mathbb{E}_s \mathbb{E}_{a^i \sim \pi^i(a^i | s)} \nabla_{\theta^i} \log \pi^i(a^i | s) Q^i(s, a^i, a^{-i}) \quad (8.44)$$

Действительно, для формулы градиентов по θ^i остальные стратегии можно считать фиксированными; поэтому стандартная формула применима. В ней состояния s приходят из опыта взаимодействия агентов со средой, а действия других агентов a^{-i} должны быть порождены этими самыми стратегиями прочих агентов.

Авторы MADDPG предлагают перейти здесь от policy gradient-схемы к использованию формулы аля DDPG, и забить на частоты посещения состояний. Тогда в формуле (8.44) можно брать s , a^{-i} — из приближения стратегий других агентов. По аналогии с DDPG также можно обучать детерминированную стратегию $\pi^i(s)$ с параметрами θ^i , оптимизируя

$$\mathbb{E}_s \mathbb{E}_{a^i} Q^i(s, \pi^i(s), a^{-i}) \rightarrow \max_{\theta^i},$$

где s берутся из реплей буфера, a^{-i} моделируются или, в случае CTDE-обучения, берутся из честных стратегий $\pi^j(s)$, $j \neq i$; в последнем случае матожидание по a^{-i} вырождается.

8.6.6. Системы коммуникации (DIAL)

Пожалуй, главной особенностью взаимодействия агентов вроде людей в природе является такое явление, как язык. Давайте попробуем в парадигме RL промоделировать, как агенты могут передавать друг другу сообщения.

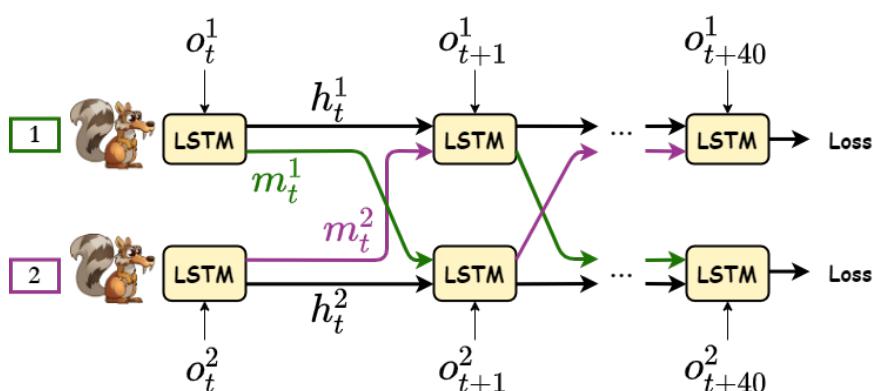
Определение 127: Скажем, что у агента i есть **канал связи** (communication channel), если на каждом шаге агент помимо действия a^i выбирает некоторое **сообщение** (message) m^i , которое не влияет на функцию переходов и функции награды, и которое все агенты получают вместе со своими наблюдениями на следующем шаге.

Конечно, можно обобщить это понятие и сказать, что канал связи есть между конкретными агентами i, j , и этот канал может быть односторонним, двухсторонним, и так далее, но нам это сейчас не принципиально. Мы можем использовать эту идею в произвольных мульти-агентных средах, в том числе обучая агентов децентрализованно, рассматривая m^i как часть пространства действий агентов.



В таком подходе удобно выбирать дискретное пространство сообщений, а при моделировании Q-функции вместо того, чтобы выдавать оценку для каждой пары действий (a^i, m^i) отдельно выдаются оценки для a^i и отдельно для m^i ; дальше агент ϵ -жадно выбирает действие для среды и отдельно ϵ -жадно выбирает сообщение для отправки. Это небольшая декомпозиция упрощает пространство действий.

Рассмотрим чуть более хитрое применение канала связи под названием Differentiable Inter-Agent Learning (DIAL), применимое для кооперативных задач (с единой функцией награды) в парадигме CTDE. Здесь естественно выбрать непрерывное пространство сообщений. На центральном сервере во время обучения просто заметим, что любые модели агента j , на шаге t минимизирующие свои стандартные функции потерь, дифференцируемы по входным наблюдениям и в том числе по полученным от других агентов сообщениям μ_{t-1}^i . Этую информацию о градиенте можно использовать для обучения i -го агента выдавать хорошие сообщения! Концептуально, схема для двух агентов в условиях частичной наблюдаемости выглядит примерно так:



На рисунке \mathbf{o}_t^i — наблюдение, поступающее на вход i -му агенту на шаге t . Оно обрабатывается рекуррентной сетью, которое выдаёт, помимо оценочных функций и вероятностей действий, скрытое состояние \mathbf{h}_t^i для передачи самому себе на следующий шаг и сообщение \mathbf{m}_t^i для передачи другим агентам на следующий шаг (как видно на схеме, между этими двумя сущностями практически нет никакой разницы). Далее это сообщение поступает на вход моделям всех других агентов на шаге $t + 1$. В конце вычислений вычисляется некоторая функция потерь в зависимости от используемого алгоритма (например, MSE для оценочных функций или суррогатная функция потерь для обучения стратегии), и этот градиент проходит не только назад по времени, но и по каналам связи в модели других агентов: как видно, вся эта схема end-to-end дифференцируема.

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

$$f(y) \rightarrow \min_y,$$

оказалась 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 (2.5) 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:

$$\begin{cases} f(\phi) \approx f(\phi_0) + \langle \nabla_{\phi} f(\phi)|_{\phi=\phi_0}, \phi - \phi_0 \rangle \rightarrow \min_{\phi} \\ \|\phi - \phi_0\|_2^2 \leq \alpha \end{cases}$$

где α — learning rate, а условие $\|\phi - \phi_0\|_2^2 \leq \alpha$ 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 141: $\mathcal{N}(0, 100)$ 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 учтёт структуру нашего функционала:

$$\begin{cases} f(\phi) \approx f(\phi_0) + \langle \nabla_\phi f(\phi)|_{\phi=\phi_0}, \phi - \phi_0 \rangle \rightarrow \min_\phi \\ \text{KL}(q(x | \phi_0) \| q(x | \phi)) \leq \alpha \end{cases}$$

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 93:

$$\nabla_\phi \text{KL}(q(x | \phi_0) \| q(x | \phi))|_{\phi=\phi_0} = 0$$

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 128: 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 93: 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.

$$\nabla_\phi^2 \text{KL}(q(x | \phi_0) \| q(x | \phi))|_{\phi=\phi_0} = \nabla_\phi^2 [\text{const}(\phi) - \mathbb{E}_{q(x|\phi_0)} \log q(x | \phi)]|_{\phi=\phi_0} = F_q(\phi_0) \quad ■$$

Прежде чем двинуться дальше, остановимся на паре важных для нас свойств матрицы Фишера.

Теорема 94 — 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):

$$\begin{aligned} (*) &= \mathbb{E}_{q(x|\phi)} \frac{\nabla_\phi q(x | \phi)}{q(x | \phi)} \left(\frac{\nabla_\phi q(x | \phi)}{q(x | \phi)} \right)^T = \\ &= \{\text{градиент логарифма}\} = \mathbb{E}_{q(x|\phi)} \nabla_\phi \log q(x | \phi) (\nabla_\phi \log q(x | \phi))^T \end{aligned} \quad ■$$

Утверждение 94: Любая матрица Фишера симметрична и положительно определена.

Теорема 95 — 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 якобианом J . Тогда:

$$F_q(\phi) = J F_q(\nu) J^T \quad (\text{A.3})$$

Доказательство.

$$\begin{aligned} F_q(\phi) &= \mathbb{E}_{q_\phi(x|\phi)}(\nabla_\phi \log q(x|\phi))(\nabla_\phi \log q(x|\phi))^T = \\ &= \{q_\phi(x|\phi) \equiv q_\nu(x|\nu)\} = \mathbb{E}_{q_\nu(x|\nu)}(\nabla_\phi \log q(x|\nu))(\nabla_\phi \log q(x|\nu))^T = \\ &= \{\text{chain rule}\} = \mathbb{E}_{q_\phi(x|\phi)} J \nabla_\nu \log q(x|\nu) (\nabla_\nu \log q(x|\nu))^T J^T = \\ &= J F_q(\nu) J^T \end{aligned} \quad \blacksquare$$

* будем 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:

$$\begin{cases} f(\phi) \approx f(\phi_0) + \langle \nabla_\phi f(\phi)|_{\phi=\phi_0}, \phi - \phi_0 \rangle \rightarrow \min_\phi \\ (\phi - \phi_0)^T F_q(\phi_0)(\phi - \phi_0) \leq \alpha \end{cases}$$

По сути, матрица Фишера задаёт нам приближённо метрику¹, 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 129: *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 96 — 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.

$$\begin{aligned} J^T \tilde{\nabla}_\phi f(\phi) &= J^T F_q(\phi)^{-1} \nabla_\phi f(\phi) = \\ &= \{\text{репараметризация матрицы Фишера (A.3)}\} = J^T (J F_q(\nu) J^T)^{-1} \nabla_\phi f(\phi) \\ &= \{\text{свойства обратной матрицы}\} = J^T J^{-T} F_q(\nu)^{-1} J^{-1} \nabla_\phi f(\phi) \\ &= \{\text{chain rule}\} = F_q(\nu)^{-1} J^{-1} J \nabla_\nu f(\nu) = \\ &= F_q(\nu)^{-1} \nabla_\nu f(\nu) = \\ &= \tilde{\nabla}_\nu f(\nu) \end{aligned} \quad \blacksquare$$

¹ А 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 СМА-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 95:

$$\nabla_{\mu} \log q(\theta | \mu, \Sigma) = \Sigma^{-1} (\theta - \mu)$$

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 взвешивания [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 96:

$$\nabla_{\Sigma} \log q(\theta | \mu, \Sigma) = -\frac{1}{2} \Sigma^{-1} + \frac{1}{2} \Sigma^{-1} (\theta - \mu) (\theta - \mu)^T \Sigma^{-1}$$

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 **векторизации матрицы vec**, вытягивающей все элементы матрицы в вектор.

Утверждение 97: 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 130: 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 142:

$$\begin{aligned} \begin{pmatrix} 3 & 0 \\ 1 & -2 \end{pmatrix} \otimes \begin{pmatrix} 4 & 0 & -1 \\ -4 & 2 & 5 \end{pmatrix} &= \begin{pmatrix} 3 \begin{pmatrix} 4 & 0 & -1 \\ -4 & 2 & 5 \end{pmatrix} & 0 \begin{pmatrix} 4 & 0 & -1 \\ -4 & 2 & 5 \end{pmatrix} \\ 1 \begin{pmatrix} 4 & 0 & -1 \\ -4 & 2 & 5 \end{pmatrix} & -2 \begin{pmatrix} 4 & 0 & -1 \\ -4 & 2 & 5 \end{pmatrix} \end{pmatrix} = \\ &= \begin{pmatrix} 12 & 0 & -3 & 0 & 0 & 0 \\ -12 & 6 & 15 & 0 & 0 & 0 \\ 4 & 0 & -1 & -8 & 0 & 2 \\ -4 & 2 & 5 & 8 & -4 & -10 \end{pmatrix} \end{aligned}$$

Теорема 97: 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 98: 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 98: 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 99: Матрица Фишера для нормального распределения имеет вид:

$$F_q(\lambda) = \begin{pmatrix} \Sigma^{-1} & \mathbf{0} \\ \mathbf{0} & \frac{1}{2}(\Sigma^{-1} \otimes \Sigma^{-1}) \end{pmatrix}$$

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 матожидание 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 матожидание 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 $F_{\text{vec } \Sigma \text{ vec } \Sigma}$.

$$\begin{aligned} & -\mathbb{E}_\theta D_\Sigma \nabla_{\text{vec } \Sigma} \log q(\theta | \mu, \Sigma)[H] = \\ &= -\mathbb{E}_\theta D_\Sigma \text{vec}(\nabla_\Sigma \log q(\theta | \mu, \Sigma)) [H] = \\ &= \{\text{подставляем (A.14)}\} = \\ &= -\mathbb{E}_\theta D_\Sigma \text{vec}\left(-\frac{1}{2} \Sigma^{-1} + \frac{1}{2} \Sigma^{-1} S \Sigma^{-1}\right) [H] = \\ &= \{\text{вносим минус, } D_\Sigma \text{ и } \mathbb{E}_\theta\} = \\ &= \text{vec}\left(\frac{1}{2} D_\Sigma \Sigma^{-1}[H] - \frac{1}{2} \mathbb{E}_\theta D_\Sigma (\Sigma^{-1} S \Sigma^{-1}) [H]\right) = \\ &= \{\text{дифф. билинейной функции}\} = \\ &= \text{vec}\left(\frac{1}{2} D_\Sigma \Sigma^{-1}[H] - \frac{1}{2} D_\Sigma \Sigma^{-1}[H] \mathbb{E}_\theta S \Sigma^{-1} - \frac{1}{2} \Sigma^{-1} \mathbb{E}_\theta S D_\Sigma \Sigma^{-1}[H]\right) = \\ &= \{\text{дифф. обратной матрицы (A.7)}\} = \\ &= \text{vec}\left(-\frac{1}{2} \Sigma^{-1} H \Sigma^{-1} + \frac{1}{2} \Sigma^{-1} H \Sigma^{-1} \mathbb{E}_\theta S \Sigma^{-1} + \frac{1}{2} \Sigma^{-1} \mathbb{E}_\theta S \Sigma^{-1} H \Sigma^{-1}\right) = \\ &= \{\mathbb{E}_\theta S = \Sigma\} = \text{vec}\left(-\frac{1}{2} \Sigma^{-1} H \Sigma^{-1} + \frac{1}{2} \Sigma^{-1} H \Sigma^{-1} + \frac{1}{2} \Sigma^{-1} H \Sigma^{-1}\right) = \\ &= \frac{1}{2} \text{vec}(\Sigma^{-1} H \Sigma^{-1}) = \\ &= \{\text{свойство (A.12)}\} = \frac{1}{2} (\Sigma^{-1} \otimes \Sigma^{-1}) \text{vec}(H) \end{aligned}$$

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 99: 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 $\lambda = (\mu, \Sigma)$.

Теорема 100: 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 2.8 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 101: 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 28. 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:

$$\sum_{k \geq 0} \alpha_k(s, a) = +\infty \quad \sum_{k \geq 0} \alpha_k(s, a)^2 < +\infty \quad (\text{A.18})$$

Докажем, что $Q_n(s, a) \xrightarrow{n \rightarrow +\infty} Q^*(s, a)$ с вероятностью 1.

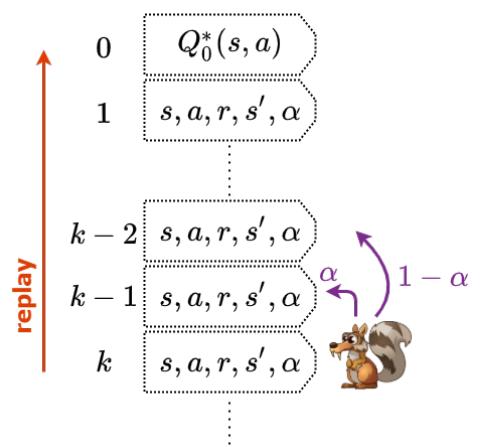
A.3.1. Action Replay Process

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 131: 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 результаты работы Q-learning-a. Попробуем построить MDP, которое было бы в некотором смысле «похожее» 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 такий фокус:

Теорема 102: 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 идейно 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 103:

$$p_{\text{ARP}}(s' | s, n, a) \xrightarrow{n \rightarrow +\infty} p(s' | s, a) \quad (\text{A.20})$$

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 (3.27) для величин $\mathbb{I}[s'_n(s, a) = s']$. При этом алфы удовлетворяют условиям сходимости (3.26)! Значит, по теореме о сходимости экспоненциального сглаживания 27 эти величины в пределе сходятся к мат.ожиданию случайной величины $\mathbb{E}[s'_n(s, a) = s']$. Поскольку $s'_n(s, a)$ для любого n генерировался из $p(s' | s, a)$, то

$$\mathbb{E}[s'_n(s, a) = s'] = p(s' | s, a),$$

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 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 104: 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, начинаящийся 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 доминировать на γ^k , следовательно, можно 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 105: 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.

$$\nu(n) = \max_{s, a} |Q_{\text{ARP}}^*(s, n, a) - Q^*(s, a)| =$$

$$\begin{aligned}
&= \{\text{уравнение оптимальности Беллмана (3.17); 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,n,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 (3.24) } \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 106:

$$\nu(n) \xrightarrow{n \rightarrow +\infty} 0$$

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'

$$p_{\text{ARP}}(n' < m_1 | s, m_2, a, s') < \frac{\epsilon}{k}$$

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;

Цикл докладов Advanced 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](#) Сэттона-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 Monroe, 1951], была обнаружена после этого в [Tsitsiklis, 1994], что позволило доказать более сильные утверждения вроде сходимости $\text{TD}(\lambda)$. Наиболее общую форму алгоритмов off-policy оценивания стратегии и оценку Retrace представили в [Munos et al., 2016] уже в 2016-ом году.

[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. Twin («Clipped Double») 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]. Distributional подход и алгоритм c51 был инициирован в [Bellemare et al., 2017]; идея перехода к квантильной аппроксимации и алгоритм QR-DQN был описан в [Dabney et al., 2018b]; алгоритм IQN предложен в [Dabney et al., 2018a]. Эквивалентность distributional-алгоритмов с обычным подходом в табличном сеттинге была показана в [Lyle et al., 2019].

В [главе 5](#) метод пробрасывания градиентов через стохастические узлы вычислительного графа REINFORCE и его применение к задаче RL были придуманы в [Williams, 1992]. Actor-Critic методы, в которых учится как стратегия, так и оценочная функция, позволяющая обучаться с неполных эпизодов, предложены в [Sutton et al., 2000]. Применение Policy Gradient подхода с нейросетевой аппроксимацией и алгоритм A2C предложен в [Mnih et al., 2016]. Список разных вариаций Policy Gradient алгоритмов можно найти в [блоге Lil'log](#) [Weng, 2018]. TRPO был предложен в [Schulman et al., 2015a]; обучение с длинных роллаутов привело к использованию GAE оценок, что было предложено в [Schulman et al., 2015b]. Алгоритм PPO описан в [Schulman et al., 2017], однако стандартные реализации, добившиеся высоких результатов на бенчмарках и способствовавшие распространению алгоритма, использовали дополнительные инженерные эвристики; на их существенное влияние на результаты обращено внимание в [Engstrom et al., 2019]. Позже в [Achiam et al., 2017] была представлена более точная нижняя оценка на погрешность суррогатной функции, давшая теоретическое обоснование использованию усреднённой по состояниям KL-дивергенции, которая и была представлена в тексте.

Применение идей policy gradient и value-based подхода для обучения нейросетей в задаче непрерывного управления, описанное в [главе 6](#), предложено в [Lillicrap et al., 2015] в алгоритме DDPG. Более стабильная версия этого алгоритма TD3 описана в [Fujimoto et al., 2018]. Алгоритм Soft Q-learning, использующий теорию Maximum Entropy RL для обучения нейросетей, описан в [Haarnoja et al., 2017]; алгоритм Soft Actor-Critic, ставший практическим алгоритмом для работы с непрерывными действиями в рамках этого сеттинга, предложен в [Haarnoja et al., 2018].

За полным погружением в математику, стоящую за многорукими бандитами ([глава 7.1](#)), можно обратиться к книге [Lattimore and Szepesvári, 2020]. Нижняя оценка регрета (теорема Лай-Роббинса) получена в [Lai and Robbins, 1985]. Асимптотическая оптимальность алгоритма UCB показана в [Auer et al., 2002]. Сама задача многоруких бандитов была впервые рассмотрена Томпсоном в [Thompson, 1933]; он же эвристически предложил сэмплирование Томпсона, которое позже тоже оказалось асимптотически оптимальным [Kaufmann et al., 2012]. Пример 104 с нахождением кратчайшего маршрута в графе взят из туториала по сэмплированию Томпсона [Russo et al., 2017]. Алгоритм, обучающий байесовские модели динамики и награды для табличных MDP и использующих сэмплирование Томпсона для разрешения дилеммы exploration-exploitation в MDP взят из [Osband et al., 2013].

Несмотря на то, что описанный в [главе 7](#) model-based подход исследовался в RL всегда, применение моделей мира и концепция сновидений популяризовалась благодаря статье [Ha and Schmidhuber, 2018]. Победа алгоритма AlphaGo в го на основе совмещения MCTS и нейросетей в итоге была обобщена сначала в алгоритм AlphaZero [Silver et al., 2018], а затем и на случай неизвестной динамики в алгоритм μ -Zero [Schrittwieser et al., 2019]. Теория линейно-квадратичных регуляторов восходит ещё к оптимальному управлению; по LQR обычно ссылаются на [Bemporad et al., 2002], а по расширению iLQR — на [Li and Todorov, 2004].

Фреймворк Maximum Entropy Inverse RL, рассмотренный в [главе 8.1](#), был предложен в [Ziebart et al., 2008]. Обобщение алгоритма из этой статьи с табличного случая на произвольный в виде процедуры Guided Cost Learning описана в [Finn et al., 2016]. Связь задачи с минимизацией расстояния между осциллярующим measure была исследована в [Ho and Ermon, 2016], где был предложен алгоритм GAIL. Расширение GAIL на случай, когда в записях эксперта доступны только наблюдения (алгоритм GAIfO) представлен в [Torabi et al., 2018]. Пример 115 со сведением к классификации задачи обучения квадрокоптера полётом по лесу описан в статье [Giusti et al., 2015].

Моделирование любопытства и скуки, описанное в [главе 8.2](#) у агентов было описано ещё Шмидхубером в далёком 1991 году [Schmidhuber, 1991] вместе с проблемой шумных телевизоров. Эвристика RND придумана в [Burda et al., 2018]; фильтрующие свойства модели обратной динамики, понятие контролируемого состояния и алгоритм ICM описаны в [Pathak et al., 2017]. Пример минимизации хаоса 120 основан на [Berseth et al., 2019].

В [главе 8.3](#) переразметка траекторий произвольными целями, также называемая алгоритмом Intentional-Unintentional, описана в [Cabi et al., 2017]. Идея HER придумана в [Andrychowicz et al., 2017]. Обобщение идеи для переразметки произвольных траекторий и связь этой задачи с обратным обучением с подкреплением одновременно замечена в [Eysenbach et al., 2020] и [Li et al., 2020]. Мета-контроллеры для автоматического подбора гиперпараметров использовались в алгоритме Agent57, обошедшем человека сразу во всех 57 играх Atari [Badia et al., 2020].

В иерархическом RL, описанном в [главе 8.4](#), алгоритм Option-Critic и формулы градиентов для обучения политики терминальности представлены в [Bacon et al., 2017]. Концепция феодализма и феодальные сети FuN предложены в [Vezhnevets et al., 2017]. Обучение похожей иерархической схемы в off-policy при помощи переразметки в виде алгоритма HIRO описано в [Nachum et al., 2018].

Задача обучения в условиях частичной наблюдаемости [главы 8.5](#) поставлена в [Smallwood and Sondik, 1973]. Обучение рекуррентных сетей в RL в рамках алгоритма DRQN описано в [Hausknecht and Stone, 2015]; эвристики разогрева и хранения скрытого состояния предложены в алгоритме R2D2 [Horgan et al., 2018]. Эпизодичная память NEC предложена в [Pritzel et al., 2017].

В [главе 8.6](#) Adversarial-атаки на алгоритмы, обученные в режиме self-play, продемонстрированы в [Gleave et al., 2019]. Алгоритм QMix для кооперативных игр и идея моделировать смешивающую сеть в классе монотонных функций предложена в [Rashid et al., 2018]. Общий алгоритм MADDPG и идея моделирования других агентов предложены в [Lowe et al., 2017]. Идея моделировать и учить протоколы коммуникации между агентами описаны в [Foerster et al., 2016].

Изображения взяты из рассмотренных статей, книги [Sutton and Barto, 2018] 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 [вот этого твита](#); судя по всему, это незаознанный концепт-арт для некой игры «Трагедия белок»... а вот, пригодился!



Литература

- [Achiam et al., 2017] Achiam, J., Held, D., Tamar, A., and Abbeel, P. (2017). Constrained policy optimization. In *International Conference on Machine Learning*, pages 22–31. PMLR.
- [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.
- [Andrychowicz et al., 2017] Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, O. P., and Zaremba, W. (2017). Hindsight experience replay. In *Advances in neural information processing systems*, pages 5048–5058.
- [Auer et al., 2002] Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256.
- [Bacon et al., 2017] Bacon, P.-L., Harb, J., and Precup, D. (2017). The option-critic architecture. In *Thirty-First AAAI Conference on Artificial Intelligence*.
- [Badia et al., 2020] Badia, A. P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, D., and Blundell, C. (2020). Agent57: Outperforming the atari human benchmark. *arXiv preprint arXiv:2003.13350*.
- [Bellemare et al., 2017] Bellemare, M. G., Dabney, W., and Munos, R. (2017). A distributional perspective on reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning- Volume 70*, pages 449–458. JMLR. org.
- [Bemporad et al., 2002] Bemporad, A., Morari, M., Dua, V., and Pistikopoulos, E. N. (2002). The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1):3–20.
- [Berseth et al., 2019] Berseth, G., Geng, D., Devin, C., Finn, C., Jayaraman, D., and Levine, S. (2019). Smirl: Surprise minimizing rl in dynamic environments. *arXiv preprint arXiv:1912.05510*.
- [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*.
- [Burda et al., 2018] Burda, Y., Edwards, H., Storkey, A., and Klimov, O. (2018). Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*.
- [Cabi et al., 2017] Cabi, S., Colmenarejo, S. G., Hoffman, M. W., Denil, M., Wang, Z., and De Freitas, N. (2017). The intentional unintentional agent: Learning to solve many continuous control tasks simultaneously. *arXiv preprint arXiv:1707.03300*.
- [Dabney et al., 2018a] Dabney, W., Ostrovski, G., Silver, D., and Munos, R. (2018a). Implicit quantile networks for distributional reinforcement learning. *arXiv preprint arXiv:1806.06923*.
- [Dabney et al., 2018b] Dabney, W., Rowland, M., Bellemare, M. G., and Munos, R. (2018b). Distributional reinforcement learning with quantile regression. In *Thirty-Second AAAI Conference on Artificial Intelligence*.
- [Engstrom et al., 2019] Engstrom, L., Ilyas, A., Santurkar, S., Tsipras, D., Janoos, F., Rudolph, L., and Madry, A. (2019). Implementation matters in deep rl: A case study on ppo and trpo. In *International Conference on Learning Representations*.
- [Eysenbach et al., 2020] Eysenbach, B., Geng, X., Levine, S., and Salakhutdinov, R. (2020). Rewriting history with inverse rl: Hindsight inference for policy improvement. *arXiv preprint arXiv:2002.11089*.
- [Finn et al., 2016] Finn, C., Levine, S., and Abbeel, P. (2016). Guided cost learning: Deep inverse optimal control via policy optimization. In *International conference on machine learning*, pages 49–58.
- [Foerster et al., 2016] Foerster, J., Assael, I. A., De Freitas, N., and Whiteson, S. (2016). Learning to communicate with deep multi-agent reinforcement learning. In *Advances in neural information processing systems*, pages 2137–2145.
- [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.
- [Giusti et al., 2015] Giusti, A., Guzzi, J., Cireşan, D. C., He, F.-L., Rodríguez, J. P., Fontana, F., Faessler, M., Forster, C., Schmidhuber, J., Di Caro, G., et al. (2015). A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robotics and Automation Letters*, 1(2):661–667.
- [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.
- [Gleave et al., 2019] Gleave, A., Dennis, M., Wild, C., Kant, N., Levine, S., and Russell, S. (2019). Adversarial policies: Attacking deep reinforcement learning. *arXiv preprint arXiv:1905.10615*.
- [Ha and Schmidhuber, 2018] Ha, D. and Schmidhuber, J. (2018). World models. *arXiv preprint arXiv:1803.10122*.
- [Haarnoja et al., 2017] Haarnoja, T., Tang, H., Abbeel, P., and Levine, S. (2017). Reinforcement learning with deep energy-based policies. *arXiv preprint arXiv:1702.08165*.
- [Haarnoja et al., 2018] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*.
- [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.
- [Hausknecht and Stone, 2015] Hausknecht, M. and Stone, P. (2015). Deep recurrent q-learning for partially observable mdps. *arXiv preprint arXiv:1507.06527*.
- [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*.
- [Ho and Ermon, 2016] Ho, J. and Ermon, S. (2016). Generative adversarial imitation learning. In *Advances in neural information processing systems*, pages 4565–4573.
- [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.
- [Kaufmann et al., 2012] Kaufmann, E., Korda, N., and Munos, R. (2012). Thompson sampling: An asymptotically optimal finite-time analysis. In *International conference on algorithmic learning theory*, pages 199–213. Springer.
- [Kauten, 2018] Kauten, C. (2018). Super Mario Bros for OpenAI Gym. GitHub.
- [Lai and Robbins, 1985] Lai, T. L. and Robbins, H. (1985). Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4–22.
- [Lattimore and Szepesvári, 2020] Lattimore, T. and Szepesvári, C. (2020). *Bandit algorithms*. Cambridge University Press.
- [Li et al., 2020] Li, A. C., Pinto, L., and Abbeel, P. (2020). Generalized hindsight for reinforcement learning. *arXiv preprint arXiv:2002.11708*.
- [Li and Todorov, 2004] Li, W. and Todorov, E. (2004). Iterative linear quadratic regulator design for nonlinear biological movement systems. In *ICINCO (1)*, pages 222–229.

- [Lillicrap et al., 2015] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.
- [Lowe et al., 2017] Lowe, R., Wu, Y. I., Tamar, A., Harb, J., Abbeel, O. P., and Mordatch, I. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in neural information processing systems*, pages 6379–6390.
- [Luke, 2013] Luke, S. (2013). *Essentials of metaheuristics*, volume 2. Lulu Raleigh.
- [Lyle et al., 2019] Lyle, C., Bellemare, M. G., and Castro, P. S. (2019). A comparative analysis of expected and distributional reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4504–4511.
- [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., 2016] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937.
- [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*.
- [Munos et al., 2016] Munos, R., Stepleton, T., Harutyunyan, A., and Bellemare, M. G. (2016). Safe and efficient off-policy reinforcement learning. *arXiv preprint arXiv:1606.02647*.
- [Nachum et al., 2018] Nachum, O., Gu, S. S., Lee, H., and Levine, S. (2018). Data-efficient hierarchical reinforcement learning. In *Advances in neural information processing systems*, pages 3303–3313.
- [Osband et al., 2013] Osband, I., Russo, D., and Van Roy, B. (2013). (more) efficient reinforcement learning via posterior sampling. *arXiv preprint arXiv:1306.0940*.
- [Pathak et al., 2017] Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 16–17.
- [Pritzel et al., 2017] Pritzel, A., Uria, B., Srinivasan, S., Puigdomenech, A., Vinyals, O., Hassabis, D., Wierstra, D., and Blundell, C. (2017). Neural episodic control. *arXiv preprint arXiv:1703.01988*.
- [Rashid et al., 2018] Rashid, T., Samvelyan, M., De Witt, C. S., Farquhar, G., Foerster, J., and Whiteson, S. (2018). Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. *arXiv preprint arXiv:1803.11485*.
- [Robbins and Monro, 1951] Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407.
- [Russo et al., 2017] Russo, D., Van Roy, B., Kazerouni, A., Osband, I., and Wen, Z. (2017). A tutorial on thompson sampling. *arXiv preprint arXiv:1707.02038*.
- [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*.
- [Schmidhuber, 1991] Schmidhuber, J. (1991). A possibility for implementing curiosity and boredom in model-building neural controllers. In *Proc. of the international conference on simulation of adaptive behavior: From animals to animats*, pages 222–227.
- [Schrittwieser et al., 2019] Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. (2019). Mastering atari, go, chess and shogi by planning with a learned model. *arXiv preprint arXiv:1911.08265*.
- [Schulman et al., 2015a] Schulman, J., Levine, S., Abbeel, P., Jordan, M. I., and Moritz, P. (2015a). Trust region policy optimization. In *Icml*, volume 37, pages 1889–1897.
- [Schulman et al., 2015b] Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2015b). High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.
- [Schulman et al., 2017] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

- [Silver et al., 2018] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144.
- [Smallwood and Sondik, 1973] Smallwood, R. D. and Sondik, E. J. (1973). The optimal control of partially observable markov processes over a finite horizon. *Operations research*, 21(5):1071–1088.
- [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.
- [Sutton et al., 2000] Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063.
- [Szita and Lörincz, 2006] Szita, I. and Lörincz, A. (2006). Learning tetris using the noisy cross-entropy method. *Neural computation*, 18(12):2936–2941.
- [Thompson, 1933] Thompson, W. R. (1933). On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294.
- [Torabi et al., 2018] Torabi, F., Warnell, G., and Stone, P. (2018). Generative adversarial imitation from observation. *arXiv preprint arXiv:1807.06158*.
- [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*.
- [Vezhnevets et al., 2017] Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., and Kavukcuoglu, K. (2017). Feudal networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1703.01161*.
- [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, 2018] Weng, L. (2018). Policy gradient algorithms. *lilianweng.github.io/lil-log*.
- [Weng, 2019] Weng, L. (2019). Evolution strategies. *lilianweng.github.io/lil-log*.
- [Williams, 1992] Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.
- [Ziebart et al., 2008] Ziebart, B. D., Maas, A. L., Bagnell, J. A., and Dey, A. K. (2008). Maximum entropy inverse reinforcement learning. In *Aaaai*, volume 8, pages 1433–1438. Chicago, IL, USA.

