

# Методы машинного обучения

ИУ-5, магистратура, 2 семестр,  
весна 2023 года



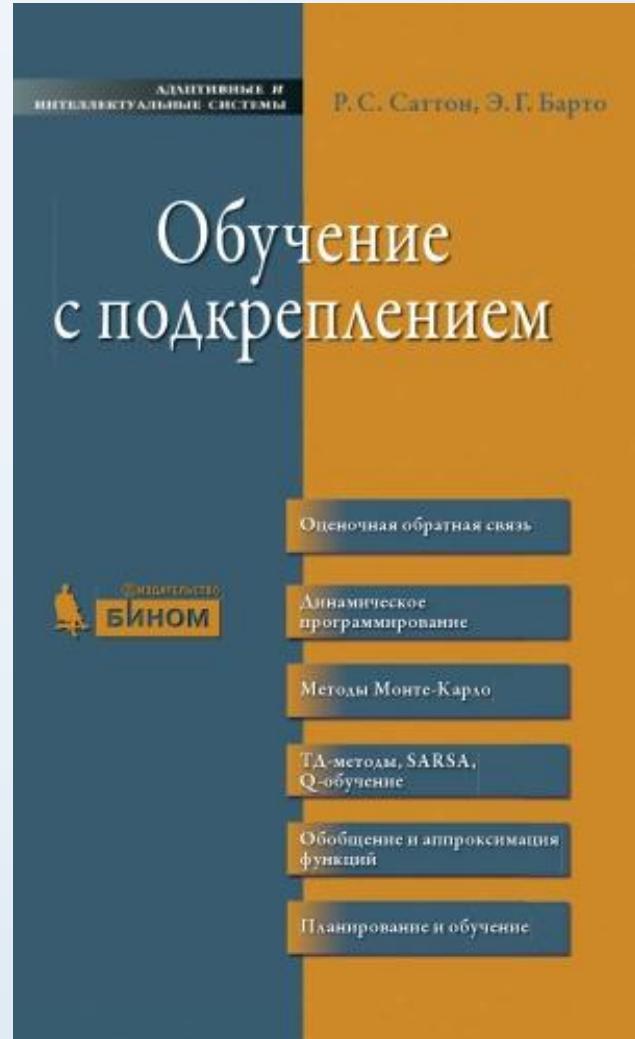


# Введение в обучение с подкреплением (reinforcement learning – RL)



# КНИГИ

- [1] Саттон Р. С., Барто Э. Дж. Обучение с подкреплением: Введение. 2-е изд. / пер. с англ. А. А. Слинкина. – М.: ДМК Пресс, 2020. – 552 с.
- Основополагающий учебник по обучению с подкреплением. Рассмотрена история развития обучения с подкреплением и основные методы.



Издание 2014 года



Издание 2020 года

# КНИГИ

- [2] Лапань Максим. Глубокое обучение с подкреплением. AlphaGo и другие технологии. — СПб.: Питер, 2020. — 496 с.
- Подробно рассмотрены теоретические основы, большое количество примеров.



# КНИГИ

- [3] Равичандиран Судхарсан. Глубокое обучение с подкреплением на Python. OpenAI Gym и TensorFlow для профи. — СПб.: Питер, 2020. — 320 с.

## Глубокое обучение с подкреплением на Python

OPENAI GYM И TENSORFLOW ДЛЯ ПРОФИ

Судхарсан Равичандиран



# КНИГИ

- [4] Грессер Лаура, Кенг Ван Лун.  
Глубокое обучение с подкреплением:  
теория и практика на языке Python. —  
СПб.:Питер, 2022. — 416 с.

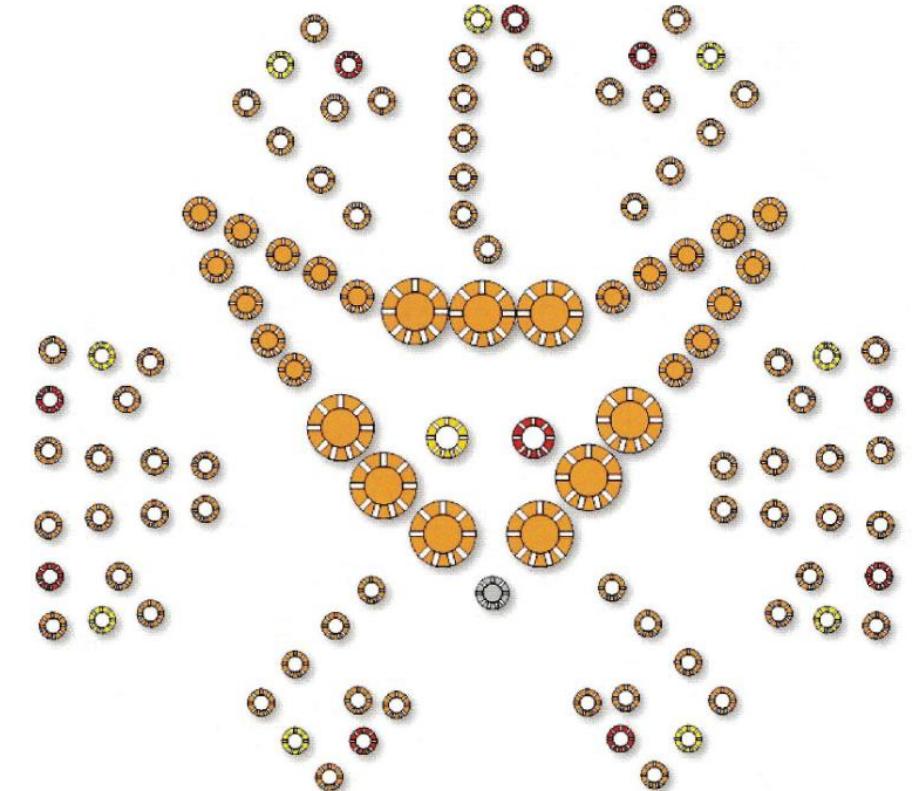


# КНИГИ

- [5] Алфимцев А.Н. Мультиагентное обучение с подкреплением: учебное пособие - Москва: Изд-во МГТУ им. Н.Э.Баумана, 2021 – 222 с.

А.Н. Алфимцев

## МУЛЬТИАГЕНТНОЕ ОБУЧЕНИЕ С ПОДКРЕПЛЕНИЕМ



# КНИГИ

- [6] Лонца А. Алгоритмы обучения с подкреплением на Python / пер. с англ. А. А. Слинкина. – М.: ДМК Пресс, 2020. – 286 с.

Алгоритмы обучения с подкреплением на Python



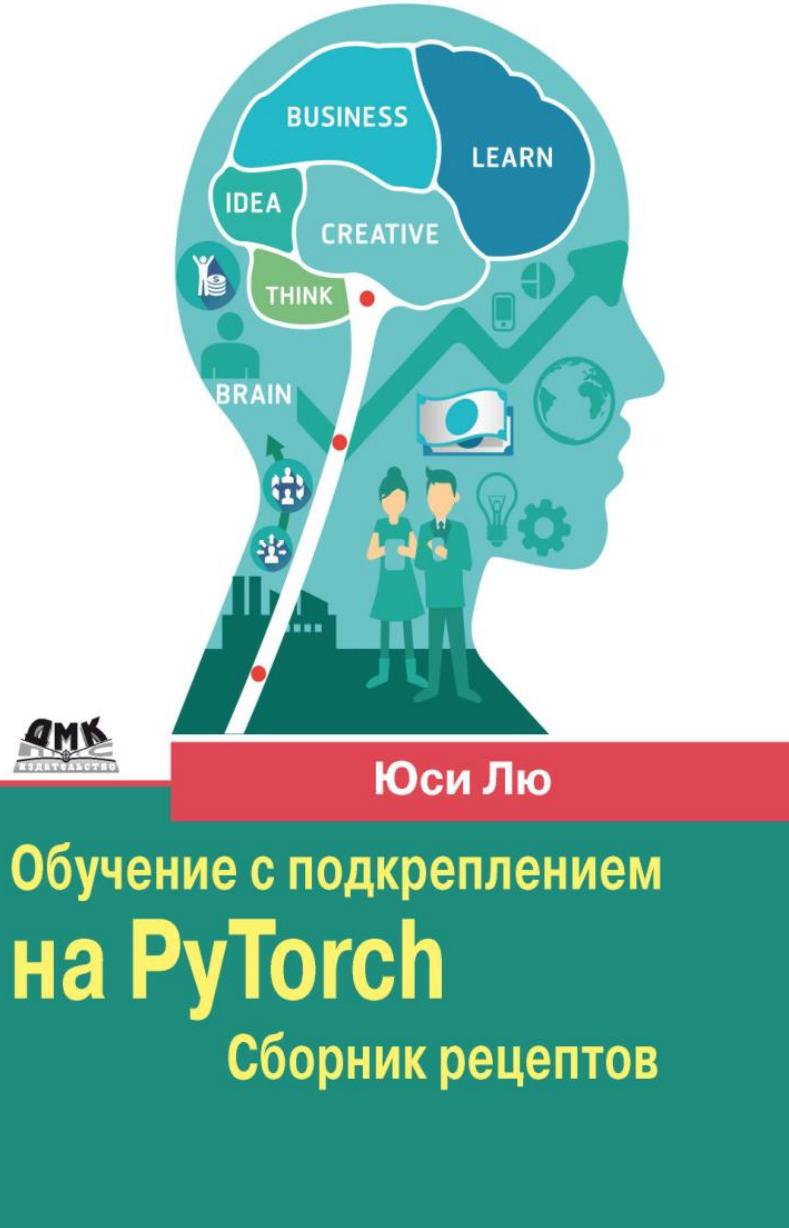
Андреа Лонца

Алгоритмы обучения  
с подкреплением  
на Python

# КНИГИ

- [7] Лю Ю. Обучение с подкреплением на PyTorch: сборник рецептов / пер. с англ. А. А. Слинкина. – М.: ДМК Пресс, 2020. – 282 с.

Обучение с подкреплением на PyTorch Сборник рецептов



# КНИГИ

- [8] Конспект по обучению с подкреплением –  
<https://arxiv.org/abs/2201.09746>

---



---

Конспект по обучению с подкреплением

---

qbrick@mail.ru

25 января 2022 г.



arXiv:2201.09746v1 [cs.LG] 19 Jan 2022

[Аннотация](#)

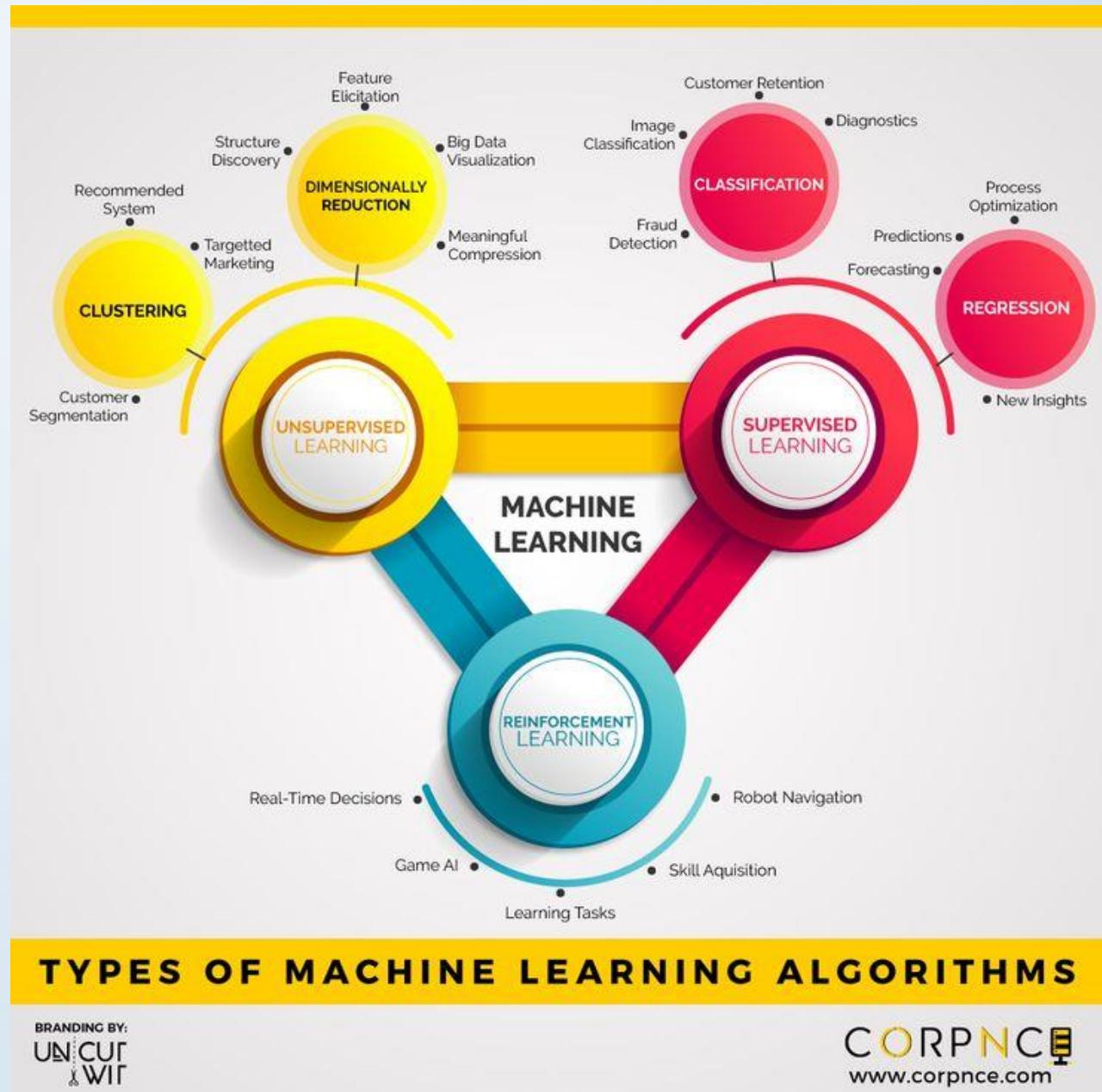
Современные алгоритмы глубокого обучения с подкреплением способны решать задачи искусственного интеллекта методом проб и ошибок без использования каких-либо априорных знаний о решаемой задаче. В этом конспекте собраны принципы работы основных алгоритмов, достигших прорывных результатов во многих задачах 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.

# Основные концепции RL

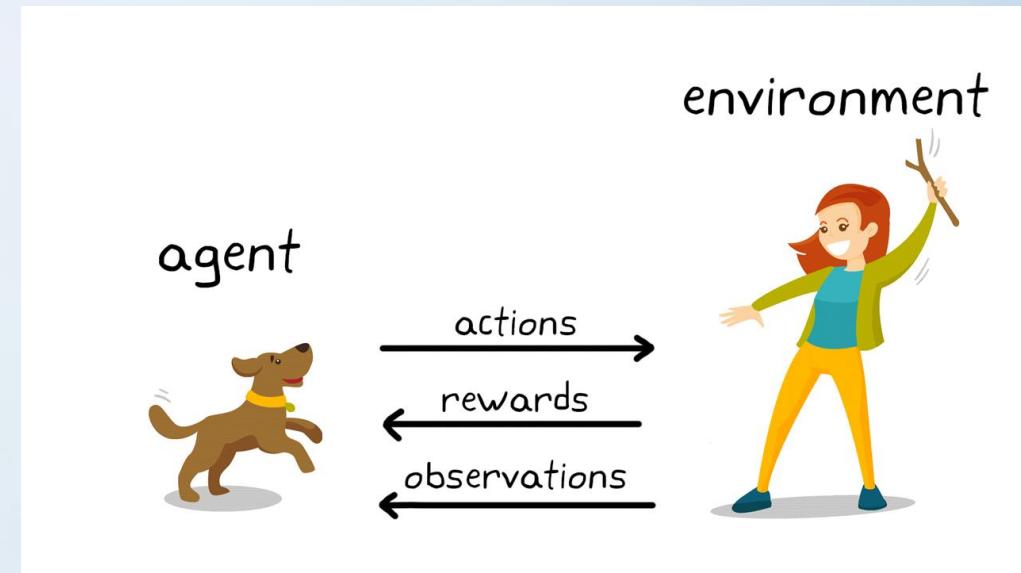
# Типы («Классификация») задач ML

- Обучение с учителем (supervised learning)
  - Классификация
  - Регрессия
  - Прогнозирование временных рядов
- Обучение без учителя (unsupervised learning)
  - Кластеризация
  - Методы понижения размерности
- Обучение с подкреплением (reinforcement learning)

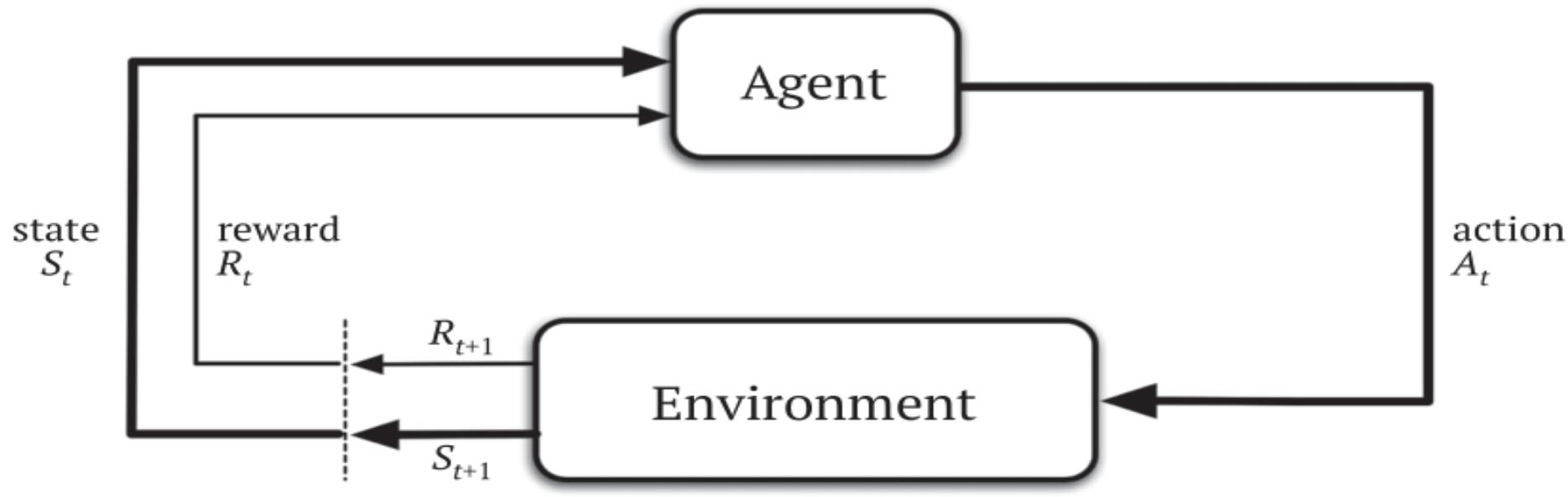


# Типичный алгоритм RL неформально [3]

1. Агент взаимодействует со средой, выполняя действие.
2. Агент выполняет действие и переходит из одного состояния в другое.
3. Агент получает награду на основании выполненного действия.
4. В зависимости от награды агент понимает, было действие хорошим или плохим.
5. Если действие было хорошим, то есть если агент получил положительную награду, то агент предпочитает выполнить это действие еще раз.
6. В противном случае агент пытается выполнить другое действие, приводящее к положительной награде. Таким образом, по сути, происходит процесс обучения методом проб и ошибок.



# Типичный алгоритм RL формально [1]



Взаимодействие между агентом и окружающей средой в марковском процессе принятия решений.

- Агент и среда взаимодействуют на каждом шаге дискретной последовательности временных шагов,  $t = 0, 1, 2, 3, \dots$
- На каждом временном шаге  $t$  агент получает некоторое представление состояния окружающей среды  $S_t$  и, исходя из него, выбирает действие  $A_t$ .
- На следующем шаге агент в качестве последствия своего действия получает числовое вознаграждение  $R_{t+1}$  и оказывается в новом состоянии  $S_{t+1}$ .
- В результате порождается траектория  $S_0 \rightarrow A_0 \mid R_1, S_1 \rightarrow A_1 \mid R_2, S_2 \rightarrow A_2 \dots$

# Основные понятия RL - 1 [3]

- **Агент (agent)** — программный модуль, способный принимать осмысленные решения, играет роль обучаемого в RL. Агенты выполняют действия, контактируя со средой, и получают награды в зависимости от своих действий. Сумму наград, полученных агентом от среды, принято называть **«возвратом»**.
- **Политика (policy)** (называемая также **«стратегией»**) определяет поведение агента в среде, направленное на достижение какой-либо цели. Способ выбора агентом действия, которое он будет выполнять, зависит от политики.
  - Допустим, агенту необходимо добраться из пункта А в пункт Б. Существует множество возможных вариантов маршрута; одни пути короткие, другие длинные. Эти пути называются **«политиками»**, потому что они представляют собой способ выполнения действия для достижения цели. Политика часто обозначается символом  $\pi$  и может быть реализована таблицей соответствия или сложным процессом поиска.

# Основные понятия RL - 2 [3]

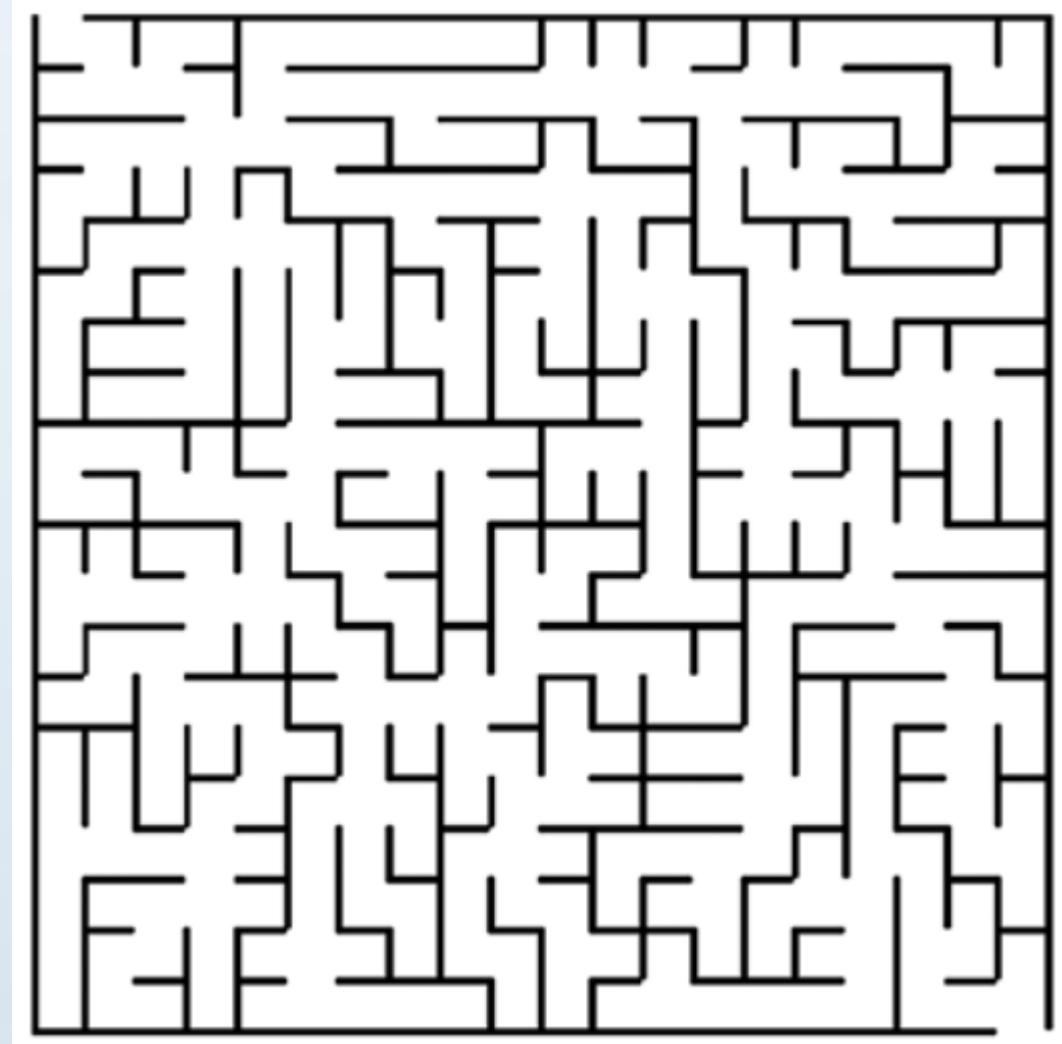
- **Функция ценности (value function)** определяет, насколько «хорошо» для агента A пребывание в конкретном состоянии. Она зависит от политики и часто обозначается  $v(s)$ . Ее значение равно ожидаемому суммарному результату, получаемому агентом, начиная с исходного состояния.
  - Функций ценности может быть несколько, «оптимальной функцией ценности»  $v_*(s)$  называется та, которая обеспечивает наибольшую ценность по всем состояниям по сравнению с другими функциями ценности. Аналогичным образом «оптимальной политикой» называется политика, имеющая оптимальную функцию ценности.
- **Модель (model)** является представлением среды с точки зрения агента. Обучение делится на два типа: с моделью и без модели. В обучении с моделью агент эксплуатирует ранее усвоенную информацию для выполнения задачи, а при обучении без модели агент просто полагается на метод проб и ошибок для выполнения правильного действия.
  - Допустим, агенту необходимо добраться из пункта А в пункт Б. В обучении с моделью агент использует имеющуюся у него карту, а в обучении без модели агент методом проб и ошибок пробует разные пути и выбирает самый быстрый.
  - Если в качестве модели используется глубокая нейронная сеть, то говорят о **глубоком обучении с подкреплением**.

# Основные понятия RL - 3 [1]

- **Компромисс между исследованием и использованием. Дилемма исследование-использование.** *Exploration-Exploitation dilemma*.
- Чтобы получить большое вознаграждение, обучающийся с подкреплением агент должен предпочитать действия, которые были испробованы в прошлом и принесли вознаграждение. Агента, который всегда выбирает наиболее выгодный на текущий момент действия принято называть «жадным».
- Но чтобы найти такие действия, он должен пробовать действия, которые раньше не выбирал. Агент должен использовать уже приобретенный опыт, чтобы получить вознаграждение, но должен продолжать исследования, чтобы выбирать более эффективные действия в будущем.
- Дилемма состоит в том, что одного лишь исследования или использования недостаточно для успешного решения задачи. Агент должен пробовать разные действия и неуклонно отдавать предпочтение тем, которые кажутся наилучшими.
- Не существует однозначно эффективного баланса между исследованием и использованием. Все зависит от длительности сессии, особенностей среды и других факторов.
  1. В RL агент должен действовать, невзирая на значительную неопределенность окружающей среды. Как правило, агент начинает с «исследовательских» ходов, чтобы получить первичную информацию о среде.
  2. В случае короткой сессии более выигрышными являются «жадные» стратегии.
  3. В случае длинной сессии более выигрышными являются стратегии со значительной «исследовательской» составляющей.

# Пример задачи RL - лабиринт [3]

- Цель игры — добраться до выхода и не заблудиться в лабиринте.
- Агент — тот, кто перемещается по лабиринту.
- Среда — лабиринт.
- Состояние — позиция в лабиринте, на которой в данный момент находится агент.
- Агент выполняет действие, перемещаясь из одного состояния в другое.
- Агент получает положительный результат, если при выполнении действия он не наталкивается на препятствие, и отрицательный результат, если при своем действии он наталкивается на препятствие и не может добраться до конечной точки.



# Типы сред в RL - 1 [3]

- **Детерминированные и стохастические среды.**
- Среда называется **детерминированной**, если последствия действий полностью известны по текущему состоянию. Например, в шахматной партии известен точный результат хода каждой фигурой.
- Среда называется **стохастической**, если результат не может быть определен по текущему состоянию из-за повышенной неопределенности. Пример подбрасывание кубика 1-6.

# Типы сред в RL - 2 [3]

- **Среды с полной и неполной информацией.**
- В среде **с полной информацией** агент может определить состояние системы в любой момент времени. Например, в шахматной партии состояние системы (то есть положение всех фигур на доске) известно в любой момент времени, так что игрок может принять оптимальное решение.
- В среде **с неполной информацией** агент не может определить состояние системы в любой момент времени. (Типичный пример – игра в карты, когда карты противника неизвестны)
  - Среды с неполной информацией в настоящий момент особенно активно изучаются в RL. Большинство практических задач связано именно с такими средами.
  - Пример задачи – подбор параметра станка по частично известным характеристикам брака детали. Фактически, в этом случае мы используем обучение на основе датасета (как в обучении с учителем).

# Типы сред в RL - 3 [3]

- **Дискретные и непрерывные среды.**
- Если существует конечный набор доступных действий для перехода из одного состояния в другое, среда называется **дискретной**. Например, в шахматной партии набор ходов конечен.
- Если существует бесконечный набор доступных действий для перехода из одного состояния в другое, среда называется **непрерывной**. Например мы подаем силу или скорость в качестве параметра физической модели.
- А если определяем угловой курс корабля?

# Типы сред в RL - 4 [3]

# Типы сред в RL - 5 [3]

- **Одноагентные и многоагентные среды.**
- В **одноагентной** среде используется только один агент, а в **многоагентной** среде агентов несколько.
- Многоагентные среды в основном являются стохастическими, так как они обладают повышенным уровнем неопределенности.

# Марковские процессы принятия решений (МПР)

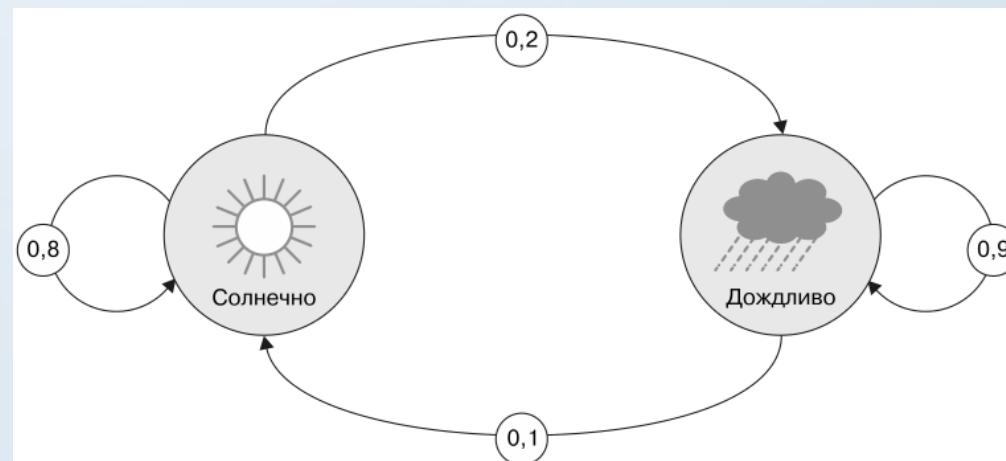
# МППР – 1 [3]

- Марковское свойство гласит, что «будущее зависит только от настоящего, но не от прошлого» («будущее связано с прошлым через настоящее»).
- Марковская цепь представляет собой вероятностную модель, которая для прогнозирования следующего состояния зависит только от текущего состояния, но не от предыдущих состояний, будущее условно независимо от прошлого. Марковская цепь соответствует марковскому свойству.
- Пример марковского процесса «пасмурно» → «дождь» (состояния до «пасмурно» не учитываются).
- Пример немарковского процесса – бросок кубика (не зависит от предыдущего состояния).

## МППР – 2 [2,3]

- Переход из одного состояния марковской цепи в другое называется «переходом», а его вероятность называется «вероятностью перехода».
- Вероятности перехода можно свести в таблицу, которую называют «марковской таблицей», а также представить в виде графа.

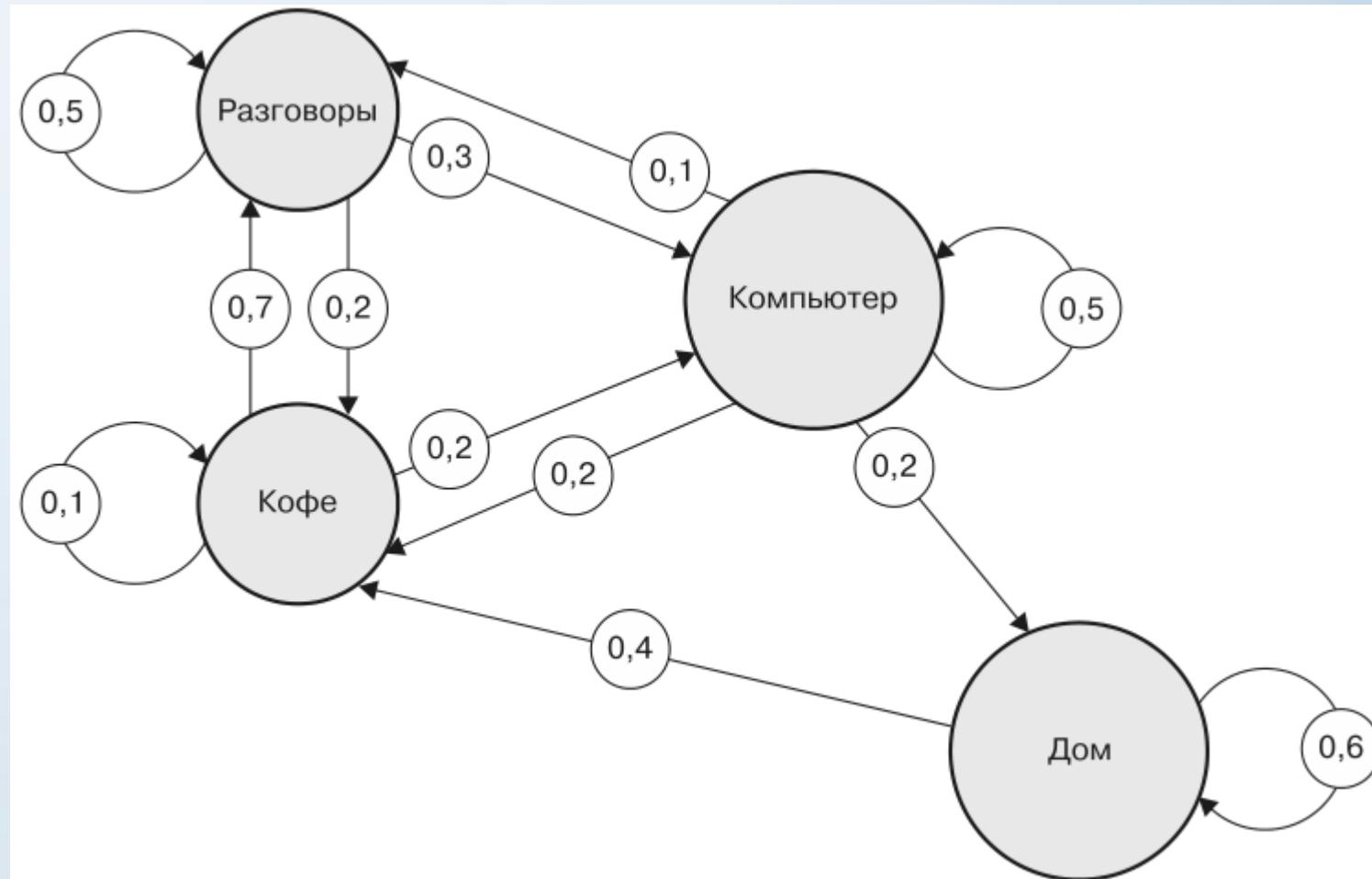
	Солнечно	Дождливо
Солнечно	0,8	0,2
Дождливо	0,1	0,9



# МППР – 3 [2]

- **Пример работы офисного сотрудника.**
- Считается, что его рабочий день обычно начинается из состояния «дом» и всегда с «кофе», без исключений (нет переходов «дом → компьютер» и «дом → разговоры»).
- Рабочие дни всегда заканчиваются (то есть происходит переход в состояние «дом» из состояния «компьютер»).

	Дом	Кофе	Разговоры	Компьютер
Дом	60 %	40 %	0 %	0 %
Кофе	0 %	10 %	70 %	20 %
Разговоры	0 %	20 %	50 %	30 %
Компьютер	20 %	20 %	10 %	50 %



## МППР – 4 [3]

Марковский процесс принятия решений (Markov Decision Process – MDP) (в некоторых источниках также называемый «марковский процесс с вознаграждением») является расширением марковских цепей. Включает следующие параметры:

1. Набор состояний ( $S$ ), в которых может находиться агент.
2. Набор действий ( $A$ ), которые могут выполняться агентом для перехода из одного состояния в другое.
3. Вероятность перехода  $P_{ss'}^a$ , из состояния  $s$  в состояние  $s'$  посредством выполнения действия  $a$ .
4. Вероятность награды  $R_{ss'}^a$ , получаемой агентом при переходе из состояния  $s$  в состояние  $s'$  посредством выполнения действия  $a$ .
5. Поправочный коэффициент  $\gamma$  (гамма), управляющий соотношением важности немедленных и будущих наград.

# МППР – 5 [2,3]

- Агент пытается максимизировать сумму наград (накопленную награду), полученную от среды, а не каждую немедленную награду. Сумма наград, полученных агентом от среды, называется **«возвратом»** или **«доходом»** ( $G$  – gain):

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

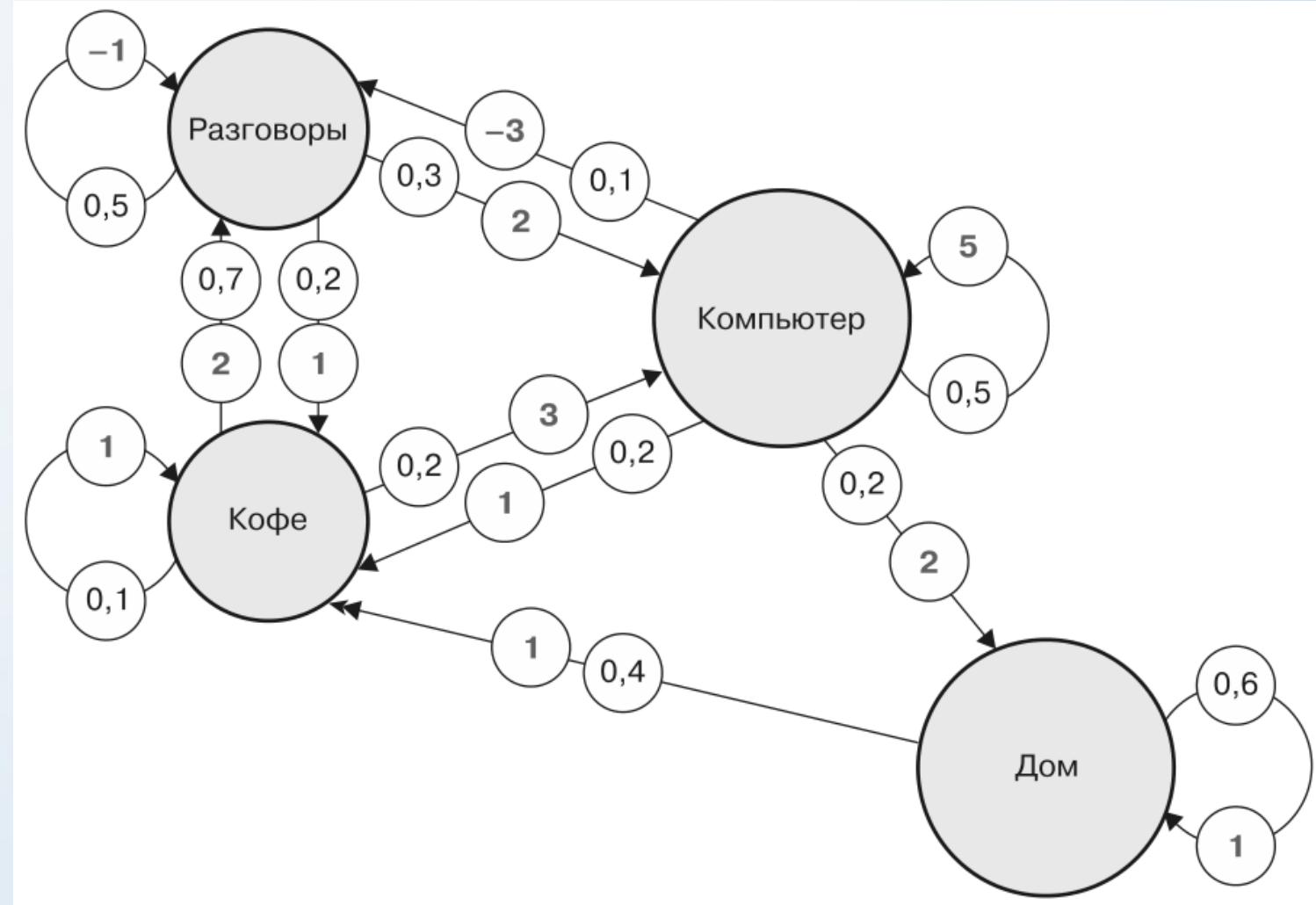
- Для каждого момента времени  $t$  вычисляется доход как сумма последующих вознаграждений, но более отдаленные из них умножаются на поправочный коэффициент  $\gamma$  (коэффициент дисконтирования), возвещенный в степень, равную числу шагов, на которое мы отстоим от начальной точки в момент времени  $t$ .
- Поправочный коэффициент определяет относительную важность будущих и немедленных наград. Его значение лежит в диапазоне от 0 до 1. Поправочный коэффициент 0 означает, что немедленные награды более важны, а поправочный коэффициент 1 означает, что будущие награды важнее немедленных.
- На основе дохода можно определить ценность состояния:

$$V(s) = \mathbb{E}[G_t | S_t = s]$$

- Ценность состояния определяется как средняя (или ожидаемая) выгода, которую агент получает находясь в состоянии  $s$ .
- В случае  $\gamma = 1$  для многих графов  $V(s) \rightarrow \infty$ . Этот бесконечный результат является одной из причин введения поправочного коэффициента  $\gamma$  в марковский процесс с вознаграждением вместо простого суммирования всех будущих вознаграждений. В данном случае мы хотим ограничить горизонт, до которого будем проводить вычисления. Такое ограничение дает коэффициент  $\gamma$  со значением меньше 1.

# МППР – 6 [2]

- Граф переходов с вероятностями перехода и вознаграждениями (целые числа).
- Чтобы вычислить ценность состояния  $V(s)$  для случая  $\gamma = 0$ , нужно суммировать ценности всех переходов, умножив их на вероятности.
- В случае  $\gamma = 1$  для данного графа  $V(s) \rightarrow \infty$  для любого состояния.



$$V(\text{Разговоры}) = -1 \cdot 0,5 + 2 \cdot 0,3 + 1 \cdot 0,2 = 0,3;$$

$$V(\text{Кофе}) = 2 \cdot 0,7 + 1 \cdot 0,1 + 3 \cdot 0,2 = 2,1;$$

$$V(\text{Дом}) = 1 \cdot 0,6 + 1 \cdot 0,4 = 1,0;$$

$$V(\text{Компьютер}) = 5 \cdot 0,5 + (-3) \cdot 0,1 + 1 \cdot 0,2 + 2 \cdot 0,2 = 2,8.$$

# МППР – 7 [2,3]

- Формально политикой (стратегией)  $\pi$  называется отображение состояний на вероятности выбора каждого возможного действия.
- Если агент следует стратегии  $\pi$  в момент  $t$ , то  $\pi(a|s) = P[A_t = a|S_t = s]$  – вероятность того, что  $A_t = a$  при условии  $S_t = s$  (вероятность выбора действия  $a$  и дальнейшего перехода в состояние  $s'$  при условии нахождения в состоянии  $s$ ).
- Рассмотрение выше функции ценности  $V(s)$  также предполагало нахождение в рамках определенной стратегии  $V(s) \equiv V_\pi(s)$ .
- Функция ценности действия (функция ценности состояния-действия, q-функция):

$$\begin{aligned}q_\pi(s, a) &= \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \\&= \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a]\end{aligned}$$

- Таким образом, q-функция задает ожидаемый доход, начиная с состояния  $s$  с действием  $a$  в соответствии с политикой  $\pi$ .

# Уравнения Беллмана [1,2,3]

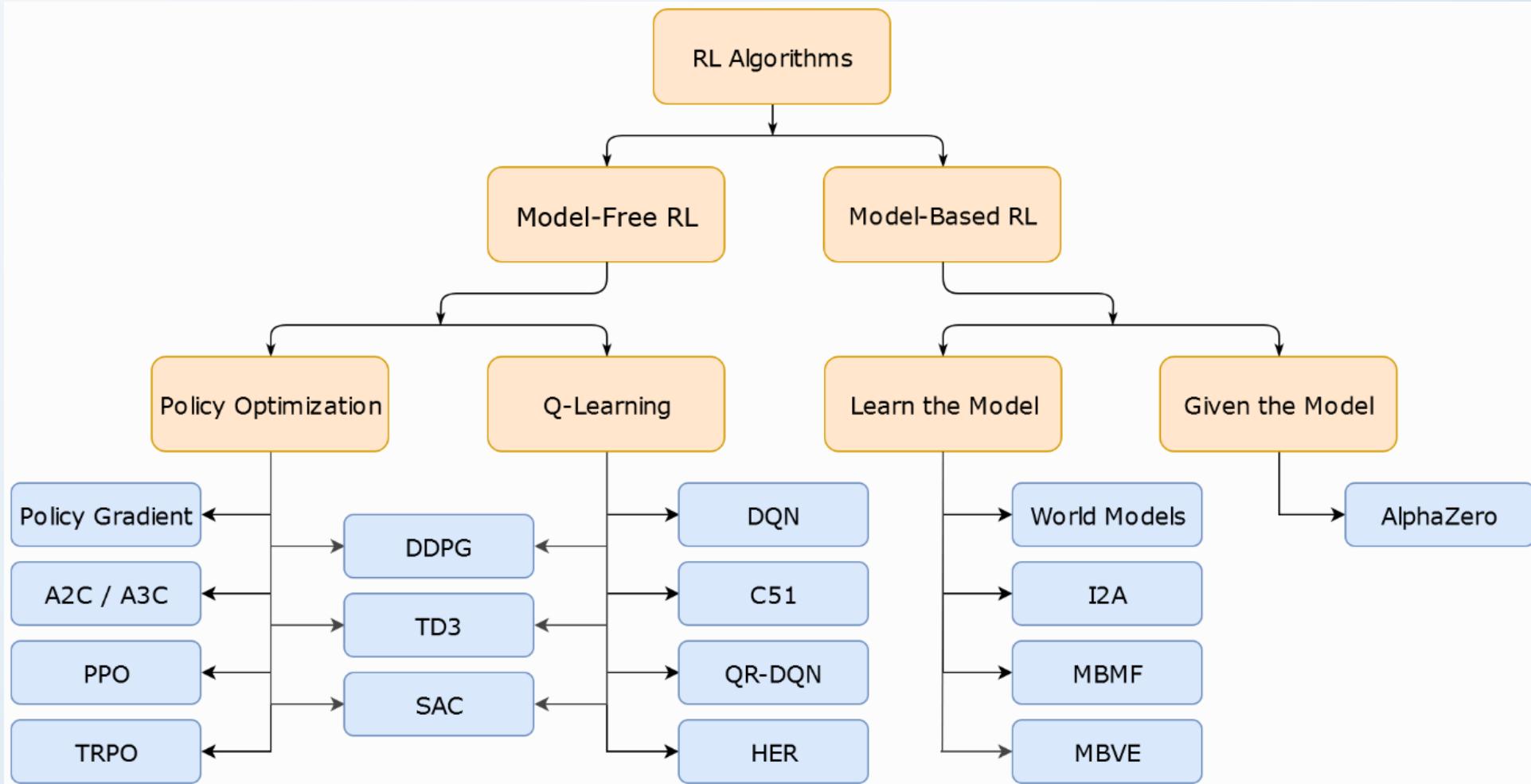
- Вернемся к определению дохода:  $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$
- Для момента времени  $t+1$ :  $G_{t+1} = R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \dots$
- Тогда  $G_t = R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \dots) = R_{t+1} + \gamma G_{t+1}$
- Уравнение Беллмана для функции ценности действия  $V_{\pi}(s) = \mathbb{E}_{\pi}[G_t | S_t = s] = \mathbb{E}_{\pi}[R_{t+1} + \gamma G_{t+1} | S_t = s]$
- Уравнение оптимальности Беллмана (где \* означает наилучшее значение или наилучшую стратегию):

$$V_{\pi}(s) = \max_a (q_{\pi^*}(s, a)) = \max_a (\mathbb{E}_{\pi^*}[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a])$$

- Физический смысл: оптимальная ценность состояния соответствует действию, которое дает максимально возможное ожидаемое немедленное вознаграждение плюс дисконтированное отдаленное вознаграждение (с помощью коэффициента  $\gamma$ ) для следующего состояния.
- Другая формулировка принципа Беллмана: на каждом шаге следует выбирать оптимальное управление в предположении об оптимальности управлений для всех последующих шагов.
- Таким образом, значения ценности дают нам не только наилучшее из возможных вознаграждений, но и в основном оптимальную стратегию для получения этой награды: если агенту известны ценности для каждого состояния, то он автоматически знает, как собрать все эти вознаграждения. Благодаря принципу оптимальности Беллмана агенту в любом его состоянии достаточно выбрать действие с максимальным ожидаемым вознаграждением, которое представляет собой сумму немедленного вознаграждения и отдаленного вознаграждения, дисконтированного на один шаг.
- Принцип оптимальности Беллмана является важной теоретической концепцией, но решение уравнений Беллмана напрямую практически не используется, так как связано с большими вычислительными затратами.

# Таксономия алгоритмов обучения с подкреплением

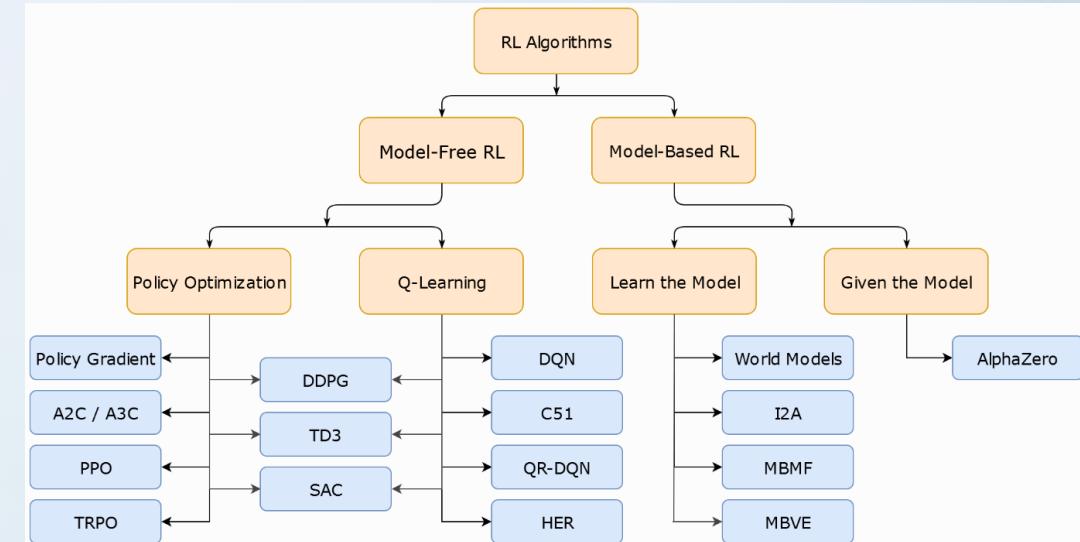
# Таксономия RL-алгоритмов - 1



1. Краткая таксономия
2. Более детальная таксономия

# Таксономия RL-алгоритмов - 2

- **Model-Free RL** – у агента нет модели среды.
- **Model-Based RL** – у агента есть модель среды (но создание исчерпывающей модели, как правило, невозможно).
- Model-Based RL предполагает использование техник автоматизированного планирования.
- **Given the Model** означает, что модель полностью задана для агента.
- **Learn the Model** означает, что агент изучает среду в процессе решения задачи.
- Методы **Policy Optimization** направлены на решение задачи оптимизации применительно к политике  $\pi(a|s)$ . Оптимизация политики является основной задачей обучения, поэтому данные методы рассматриваются как стабильные и надежные.
- Методы **Q-обучения** построены на аппроксимации q-функции  $q_\pi(s, a)$ . Обычно используется целевая функция, основанную на уравнениях Беллмана. Как правило, в этом подходе не используется информация о политике. Поэтому можно использовать данные, собранные в любой момент обучения, независимо от того, как агент изучал среду, на каком шаге были получены данные.
- Также методы **Policy Optimization** и **Q-обучения** не являются взаимоисключающими, поэтому существуют алгоритмы, комбинирующие оба подхода.



# Среды для разработки алгоритмов обучения с подкреплением

Список сред - <https://github.com/clvrai/awesome-rl-envs>

conda env create -f environment.yml

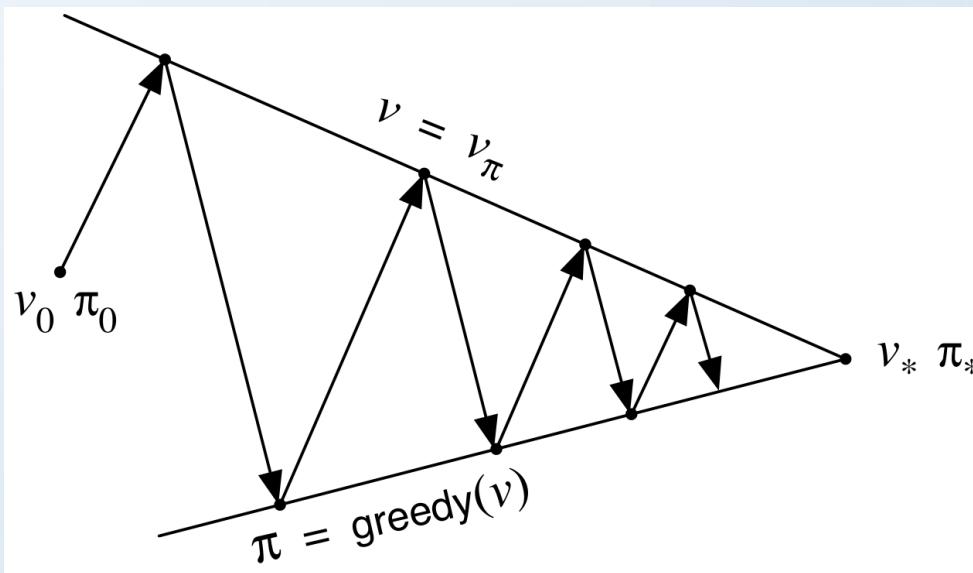
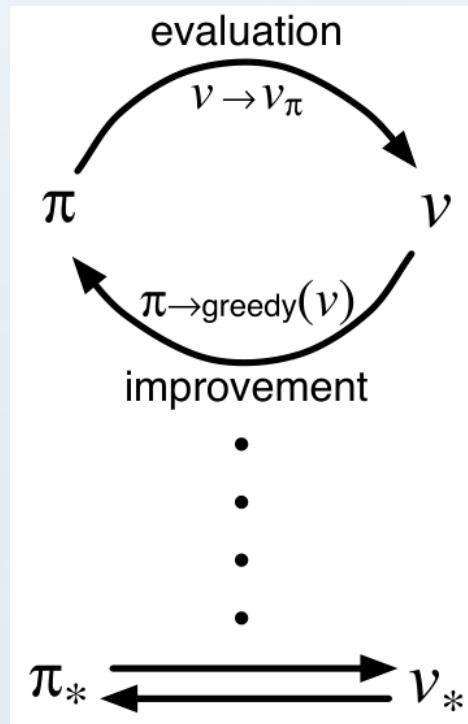
# OpenAI Gym (Gymnasium)

- Gym – старая версия - <https://github.com/openai/gym>
  - Документация - <https://www.gymlibrary.dev/>
- Gymnasium – новая версия - <https://github.com/Farama-Foundation/Gymnasium>
  - Документация - <https://gymnasium.farama.org/>
  - Примеры среды - [https://gymnasium.farama.org/environments/toy\\_text/](https://gymnasium.farama.org/environments/toy_text/)
  - Базовый пример - [https://gymnasium.farama.org/content/basic\\_usage/](https://gymnasium.farama.org/content/basic_usage/)

# Алгоритмы с использованием динамического программирования

# Policy Iteration -1 [1]

- Итерация по стратегиям (политикам). Метод предполагает итеративное выполнение двух шагов:
  - Оценивание стратегии (Policy Evaluation)
  - Улучшение стратегии (Policy Improvement)
- Таким образом получается последовательность монотонно улучшающихся стратегий и функций ценности:



$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \cdots \xrightarrow{I} \pi_* \xrightarrow{E} v_*,$$

где  $\xrightarrow{E}$  обозначает *оценивание стратегии*, а  $\xrightarrow{I}$  – *улучшение стратегии*

# Policy Iteration -2 [1]

- **Оценивание стратегии (Policy Evaluation).**
- Вероятность перехода из состояния  $s$  в состояние  $s'$  с действием  $a$  и наградой  $r$ :

$$p(s', r | s, a) \doteq \Pr\{S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a\}$$

- Для состояния  $s$  и действия  $a$  функция ценности состояния при стратегии  $\pi$ :

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t | S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] \\ &= \sum_a \pi(a | s) \sum_{s', r} p(s', r | s, a) [r + \gamma v_\pi(s')], \end{aligned}$$

где  $\pi(a | s)$  – вероятность предпринять действие  $a$  в состоянии  $s$  при стратегии  $\pi$

- Начальное приближение  $v_0$  выбирается произвольно.
- Заключительному состоянию, если оно существует, должна быть сопоставлена ценность 0.
- Каждое следующее приближение получается применением уравнения Беллмана для  $v$  в качестве правила обновления:

$$\begin{aligned} v_{k+1}(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s] \\ &= \sum_a \pi(a | s) \sum_{s', r} p(s', r | s, a) [r + \gamma v_k(s')] \end{aligned}$$

# Policy Iteration -3 [1]

- **Оценивание стратегии (Policy Evaluation).**
- Для вычисления следующего приближения  $v_{k+1}$  по  $v_k$  алгоритм итеративного оценивания стратегии применяет одну и ту же операцию к каждому состоянию  $s$ : заменяет старую ценность  $s$  новой ценностью, вычисленной по старым ценностям следующих за  $s$  состояний и ожидаемым немедленным вознаграждениям на всех одношаговых переходах при следовании оцениваемой стратегии.
- На каждой итерации итеративного оценивания стратегии ценность каждого состояния обновляется один раз с целью получить новую приближенную функцию ценности  $v_{k+1}$ .
- Рассмотрим алгоритм итеративного оценивания стратегий. Алгоритм проверяет величину  $\Delta = \max|v_{k+1}(s) - v_k(s)|$  после каждого прохода и останавливается, если она достаточно мала.

## Алгоритм итеративного оценивания стратегии для оценивания $V \approx v_\pi$

Вход:  $\pi$ , стратегия, подлежащая оцениванию

Параметр алгоритма: небольшая пороговая величина  $\theta > 0$ , определяющая точность оценки

Инициализировать  $V(s)$  для всех  $s \in \mathcal{S}^+$  произвольным образом с той оговоркой, что  $V(\text{terminal}) = 0$

Повторять:

$$\Delta \leftarrow 0$$

Повторять для каждого  $s \in \mathcal{S}$ :

$$v \leftarrow V(s)$$

$$V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r | s, a) [r + \gamma V(s')]$$

$$\Delta \leftarrow \max(\Delta, |v - V(s)|)$$

пока не окажется  $\Delta < \theta$

# Policy Iteration -4 [1]

- **Улучшение стратегии (Policy Improvement).**
- Предположим, что мы определили функцию ценности  $v_\pi$  для произвольной детерминированной стратегии  $\pi$ . Для некоторого состояния  $s$  мы хотели бы знать, стоит или не стоит изменять стратегию, так чтобы она детерминировано выбирала действие  $a$ , не относящееся к текущей стратегии. Один из способов ответить на этот вопрос – рассмотреть, что будет, если выбрать  $a$  в состоянии  $s$ , а затем следовать существующей стратегии  $\pi$ . Ценность при таком поведении равна:

$$\begin{aligned} q_\pi(s, a) &\doteq \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a] \\ &= \sum_{s', r} p(s', r | s, a)[r + \gamma v_\pi(s')]. \end{aligned}$$

- В этом помогает **теорема об улучшении стратегии**. Пусть  $\pi$  и  $\pi'$  любая пара детерминированных стратегий, такая, что для всех состояний  $s$ :

$$q_\pi(s, \pi'(s)) \geq v_\pi(s)$$

- Тогда стратегия  $\pi'$  должна быть не хуже, чем  $\pi$ . Иначе говоря, она должна приносить не меньший ожидаемый доход для всех состояний  $s$ :

$$v_{\pi'}(s) \geq v_\pi(s)$$

# Policy Iteration -5 [1]

- **Улучшение стратегии (Policy Improvement).**
- Зная стратегию и ее функцию ценности, мы можем оценить изменение стратегии, состоящее в замене одного действия в одном состоянии. Обобщение – рассмотреть изменение всех возможных действий во всех состояниях, выбирая в каждом состоянии то действие, которое кажется наилучшим согласно функции  $q_{\pi}(s, a)$ . То есть рассмотреть новую жадную стратегию  $\pi'$ , определенную следующим образом:

$$\begin{aligned}\pi'(s) &\doteq \operatorname{argmax}_a q_{\pi}(s, a) \\ &= \operatorname{argmax}_a \mathbb{E}[R_{t+1} + \gamma v_{\pi}(S_{t+1}) | S_t = s, A_t = a] \\ &= \operatorname{argmax}_a \sum_{s', r} p(s', r | s, a) [r + \gamma v_{\pi}(s')],\end{aligned}$$

- где  $\operatorname{argmax}(a)$  обозначает значение  $a$ , при котором следующее далее выражение достигает максимума (возможные неоднозначности разрешаются произвольным образом). Жадная стратегия выбирает действие, которое кажется наилучшим в краткосрочной перспективе – после заглядывания вперед на один шаг – согласно функции  $v_{\pi}$ . По построению, жадная стратегия удовлетворяет условиям теоремы об улучшении стратегии, поэтому она заведомо не хуже исходной стратегии.

# Policy Iteration -6 [1]

- **Улучшение стратегии (Policy Improvement).**
- Процесс конструирования новой стратегии, улучшающей исходную путем жадного выбора относительно функции ценности исходной стратегии, называется улучшением стратегии.
- Предположим, что новая жадная стратегия  $\pi'$  столь же хороша, но не лучше старой стратегии  $\pi$ . Тогда  $v_\pi = v_{\pi'}$ , и для всех состояний  $s$ :

$$\begin{aligned}v_{\pi'}(s) &\doteq \max_a \mathbb{E}[R_{t+1} + \gamma v_{\pi'}(S_{t+1}) | S_t = s, A_t = a] \\&= \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma v_{\pi'}(s')].\end{aligned}$$

- Но это то же самое, что уравнение оптимальности Беллмана, и потому  $v_\pi$  должна совпадать с  $v_*$ , а обе стратегии  $\pi$  и  $\pi'$  должны быть оптимальными. Таким образом, улучшение стратегии обязано давать строго лучшую стратегию всегда, кроме случая, когда исходная стратегия уже оптимальна.

# Policy Iteration -7 [1]

Алгоритм итерации по стратегиям (с использованием итеративного оценивания стратегии) для оценивания  $\pi \approx \pi_*$

- Инициализация  
 $V(s) \in \mathbb{R}$  и  $\pi(s) \in \mathcal{A}(s)$  выбираются произвольно для всех  $s \in \mathcal{S}$

- Оценивание стратегии

Повторять:

$$\Delta \leftarrow 0$$

Повторять для каждого  $s \in \mathcal{S}$ :

$$v \leftarrow V(s)$$

$$V(s) \leftarrow \sum_{s',r} p(s', r | s, \pi(s)) [r + \gamma V(s')]$$

$$\Delta \leftarrow \max(\Delta, |v - V(s)|)$$

пока не окажется  $\Delta < \theta$  (небольшое положительное число, определяющее точность оценки)

- Улучшение стратегии

$policy-stable \leftarrow true$

Для каждого  $s \in \mathcal{S}$ :

$$old-action \leftarrow \pi(s)$$

$$\pi(s) \leftarrow \arg \max_a \sum_{s',r} p(s', r | s, a) [r + \gamma V(s')]$$

Если  $old-action \neq \pi(s)$ , то  $policy-stable \leftarrow false$

Если  $policy-stable$ , то остановиться и вернуть  $V \approx v_*$  и  $\pi \approx \pi'$ ; иначе перейти к 2

# Policy Iteration (реализация)-1

- Реализация использует фрагменты кода
  - [https://github.com/escape-velocity-labs/beginner\\_rl](https://github.com/escape-velocity-labs/beginner_rl)
  - <https://aleksandarhaber.com/policy-iteration-algorithm-in-python-and-tests-with-frozen-lake-openai-gym-environment-reinforcement-learning-tutorial/>
- Файл «flake.py» - информация о среде:

Пространство состояний:  
Discrete(16)

Пространство действий:  
Discrete(4)

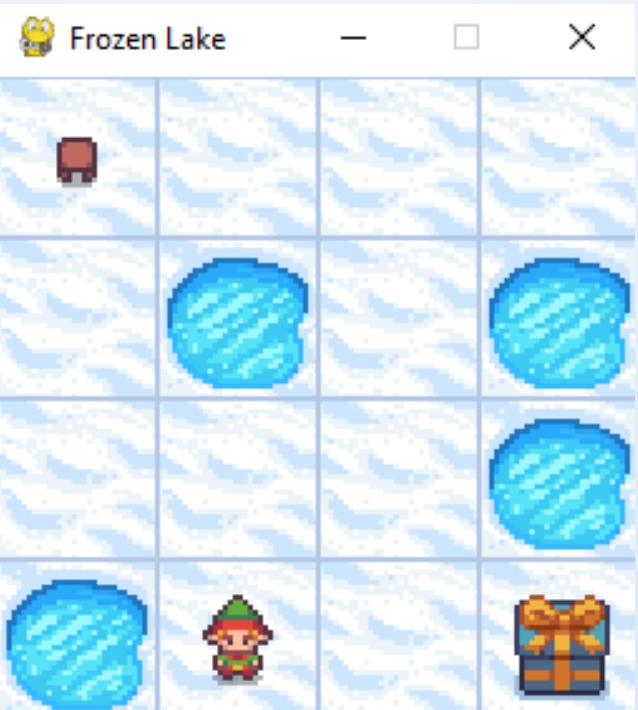
Диапазон наград:  
(0, 1)

Вероятности для 0 состояния и 0 действия:  
[(0.3333333333333333, 0, 0.0, False),  
(0.3333333333333333, 0, 0.0, False),  
(0.3333333333333333, 4, 0.0, False)]

Вероятности для 0 состояния:  
{0: [(0.3333333333333333, 0, 0.0, False),  
(0.3333333333333333, 0, 0.0, False),  
(0.3333333333333333, 4, 0.0, False)],  
1: [(0.3333333333333333, 0, 0.0, False),  
(0.3333333333333333, 4, 0.0, False),  
(0.3333333333333333, 1, 0.0, False)],  
2: [(0.3333333333333333, 4, 0.0, False),  
(0.3333333333333333, 1, 0.0, False),  
(0.3333333333333333, 0, 0.0, False)],  
3: [(0.3333333333333333, 1, 0.0, False),  
(0.3333333333333333, 0, 0.0, False),  
(0.3333333333333333, 0, 0.0, False)]}

# Policy Iteration (реализация)-2

- Файл «policy\_iteration.py» - реализация алгоритма



Алгоритм выполнился за 1000 шагов.

## Стратегия:

```
array([[1. , 0. , 0. , 0. ],
       [0. , 0. , 0. , 1. ],
       [0. , 0. , 0. , 1. ],
       [0. , 0. , 0. , 1. ],
       [1. , 0. , 0. , 0. ],
       [0.25, 0.25, 0.25, 0.25],
       [0.5 , 0. , 0.5 , 0. ],
       [0.25, 0.25, 0.25, 0.25],
       [0. , 0. , 0. , 1. ],
       [0. , 1. , 0. , 0. ],
       [1. , 0. , 0. , 0. ],
       [0.25, 0.25, 0.25, 0.25],
       [0.25, 0.25, 0.25, 0.25],
       [0. , 0. , 1. , 0. ],
       [0. , 1. , 0. , 0. ],
       [0.25, 0.25, 0.25, 0.25]])
```

# Value Iteration - 1 [1]

- Value Iteration – итерация по ценности.
- Недостаток итерации по стратегиям заключается в том, что на каждой итерации нужно оценивать стратегию, что является длительным итеративным вычислением, требующим нескольких проходов по множеству состояний.
- Шаг оценивания стратегии в алгоритме итерации по стратегиям можно усечь несколькими способами, не жертвуя гарантиями сходимости. Важный частный случай – остановка оценивания после всего одного прохода (одного обновления каждого состояния). Этот алгоритм и называется **итерацией по ценности**.
- Его можно записать в виде простой операции обновления, которая объединяет шаги улучшения стратегии и усеченного оценивания стратегии:

$$\begin{aligned}v_{k+1}(s) &\doteq \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s, A_t = a] \\&= \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma v_k(s')]\end{aligned}$$

- Обновление в алгоритме итерации по ценности идентично обновлению в алгоритме оценивания стратегии с тем отличием, что нужно взять максимум по всем действиям.
- Алгоритм итерации по ценности на каждом проходе объединяет один проход алгоритма оценивания стратегии с одним проходом алгоритма улучшения стратегии.

# Value Iteration - 2 [1]

## Алгоритм итерации по ценности для оценивания $\pi \approx \pi_*$

Параметр алгоритма: небольшая пороговая величина  $\theta > 0$ , определяющая точность оценки

Инициализировать  $V(s)$  для всех  $s \in \mathcal{S}^+$  произвольным образом с той оговоркой, что  $V(\text{terminal}) = 0$

Повторять:

$$\Delta \leftarrow 0$$

Повторять для каждого  $s \in \mathcal{S}$ :

$$v \leftarrow V(s)$$

$$V(s) \leftarrow \max_a \sum_{s',r} p(s', r | s, a) [r + \gamma V(s')]$$

$$\Delta \leftarrow \max(\Delta, |v - V(s)|)$$

пока не окажется  $\Delta < \theta$

Вывести детерминированную стратегию  $\pi \approx \pi_*$  такую, что

$$\pi(s) \leftarrow \arg \max_a \sum_{s',r} p(s', r | s, a) [r + \gamma V(s')]$$

# Value Iteration - 3 [1]

Алгоритм итерации по стратегиям (с использованием итеративного оценивания стратегии) для оценивания  $\pi \approx \pi_*$

1. Инициализация  
 $V(s) \in \mathbb{R}$  и  $\pi(s) \in \mathcal{A}(s)$  выбираются произвольно для всех  $s \in \mathcal{S}$

2. Оценивание стратегии

Повторять:

$$\Delta \leftarrow 0$$

Повторять для каждого  $s \in \mathcal{S}$ :

$$v \leftarrow V(s)$$

$$V(s) \leftarrow \sum_{s',r} p(s', r | s, \pi(s)) [r + \gamma V(s')]$$

$$\Delta \leftarrow \max(\Delta, |v - V(s)|)$$

пока не окажется  $\Delta < \theta$  (небольшое положительное число, определяющее точность оценки)

3. Улучшение стратегии

$policy-stable \leftarrow true$

Для каждого  $s \in \mathcal{S}$ :

$$old-action \leftarrow \pi(s)$$

$$\pi(s) \leftarrow \arg \max_a \sum_{s',r} p(s', r | s, a) [r + \gamma V(s')]$$

Если  $old-action \neq \pi(s)$ , то  $policy-stable \leftarrow false$

Если  $policy-stable$ , то остановиться и вернуть  $V \approx v_*$  и  $\pi \approx \pi'$ ; иначе перейти к 2

Алгоритм итерации по ценности для оценивания  $\pi \approx \pi_*$

Параметр алгоритма: небольшая пороговая величина  $\theta > 0$ , определяющая точность оценки

Инициализировать  $V(s)$  для всех  $s \in \mathcal{S}^+$  произвольным образом с той оговоркой, что  $V(terminal) = 0$

Повторять:

$$\Delta \leftarrow 0$$

Повторять для каждого  $s \in \mathcal{S}$ :

$$v \leftarrow V(s)$$

$$V(s) \leftarrow \max_a \sum_{s',r} p(s', r | s, a) [r + \gamma V(s')]$$

$$\Delta \leftarrow \max(\Delta, |v - V(s)|)$$

пока не окажется  $\Delta < \theta$

Вывести детерминированную стратегию  $\pi \approx \pi_*$  такую, что

$$\pi(s) \leftarrow \arg \max_a \sum_{s',r} p(s', r | s, a) [r + \gamma V(s')]$$

- На практике отмечается, что алгоритм итерации по стратегиям является более надежным и сходится быстрее, чем алгоритм итерации по ценности.

## Обучение на основе временных различий (temporal-difference – TD)

В данном разделе основным источником является [3].

## TD – 1 [1, 3]

- Алгоритмы итерации по стратегиям и итерации по ценности предполагают, что известен граф переходов с вероятностями перехода и вознаграждениями.
- А если граф и вероятности неизвестны? Можно учиться только на основе проб и ошибок, наблюдая среду, выполняя действия и получая вознаграждения.
- При этом предполагается, что агент может многократно «тренироваться» выполняя действия в среде в течение нескольких эпизодов, и отдельные эпизоды могут завершаться неудачно.
- Основой TD-методов является Q-матрица(количество состояний x количество действий). Q-матрица выполняет роль аналогичную стратегии (политике).

## TD – 2 [3]

- Алгоритм SARSA (State-Action-Reward-State-Action)

1. Инициализация  $Q$ -функции произвольными значениями.
2. Выбор действия из состояния с использованием эпсилон-жадной стратегии ( $\varepsilon > 0$ ) и переход в новое состояние.
3. Обновление  $Q$  предыдущего состояния по следующему правилу:

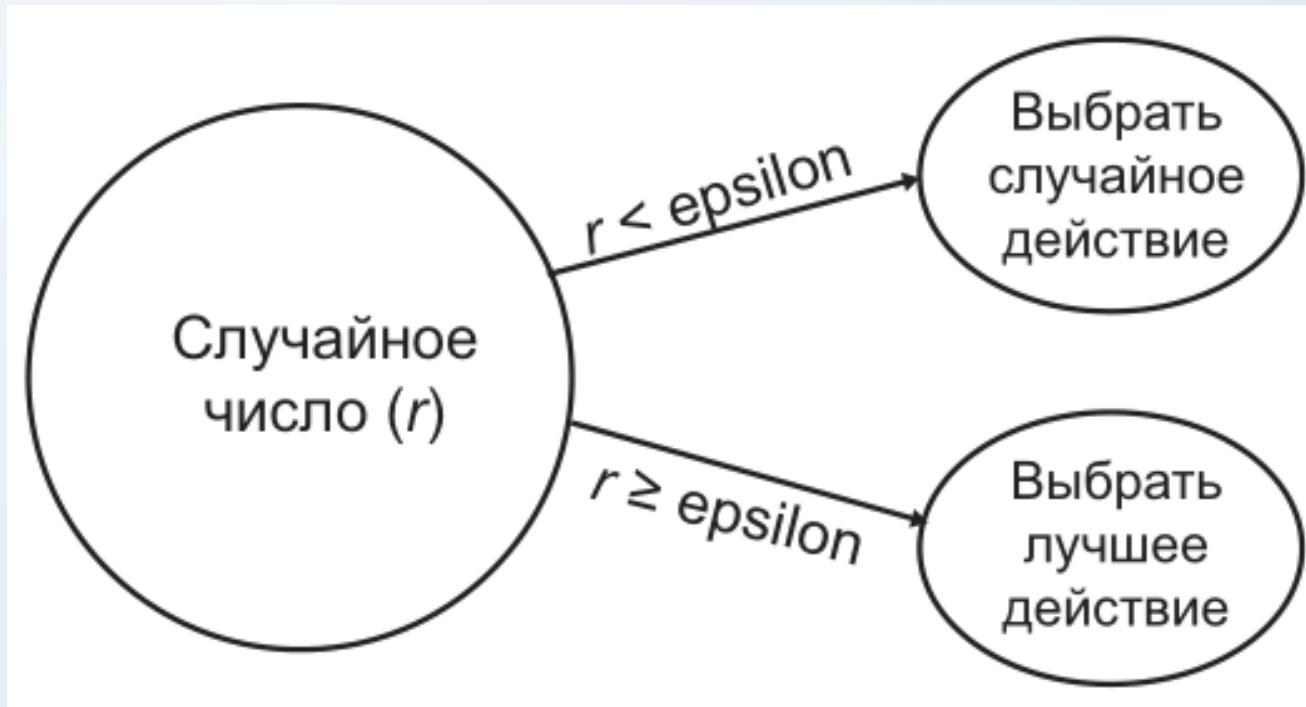
$$Q(s, a) = Q(s, a) + \alpha(r + \gamma Q(s', a') - Q(s, a)),$$

где  $a'$  – действие, выбранное по эпсилон-жадной стратегии ( $\varepsilon > 0$ ).

Коэффициент альфа часто также обозначают как learning rate.

# TD – 3 [3]

- Эпсилон-жадная стратегия



# TD – 4 [3]

- Алгоритм SARSA – пример с замерзшим озером.
- Допустим, мы находимся в состоянии (4,2) и выбираем действие на основании эпсилон-жадной стратегии. С вероятностью 1 – epsilon находится лучшее действие — «направо»:

	1	2	3	4
1	S	F	F	F
2	F	H	F	H
3	F	F	F	H
4	H	(F)	F	G

Направо

Сост.	Действ.	Цен.
(4,2)	Вверх	0,3
(4,2)	Вниз	0,5
(4,2)	Направо	0,8

## TD – 5 [3]

- Алгоритм SARSA – пример с замерзшим озером.
- Мы оказываемся в состоянии (4,3) после выполнения действия «направо» в состоянии (4,2). Как обновить ценность предыдущего состояния (4,2)? Будем считать, что значение  $a$  равно 0,1, награда — 0,3, а коэффициент дисконтирования — 1:

$$Q(s, a) = Q(s, a) + \alpha(r + \gamma Q(s', a') - Q(s, a));$$

$$Q((4,2), \text{направо}) = Q((4,2), \text{направо}) + 0,1 (0,3 + 1Q((4,3), \text{действие})) - \\ - Q((4,2), \text{направо}).$$

- Для определения значения  $Q((4,3), \text{действие})$  в SARSA используется эпсилон-жадная стратегия.

# TD – 6 [3]

- Алгоритм SARSA – пример с замерзшим озером.
- В состоянии (4,3) исследованы два действия и используется эпсилон-жадная стратегия: либо исследование с вероятностью epsilon, либо выбор с вероятностью 1 – epsilon. Допустим с вероятностью epsilon мы исследуем новое действие «направо», тогда:

	1	2	3	4			
1	S	F	F	F	Сост.	Действ.	Цен.
2	F	H	F	H	(4,2)	Вверх	0,3
3	F	F	F	H	(4,2)	Вниз	0,5
4	H	(F)	F	G	(4,2)	Направо	0,8
Направо				(4,3)	Вверх	0,1	
Направо				(4,3)	Вниз	0,3	
Направо				(4,3)	Направо	0,9	

$$Q((4,2), \text{направо}) = Q((4,2), \text{направо}) + 0,1(0,3 + 1(Q(4,3), \text{направо}) - Q((4,2), \text{направо}));$$

$$\begin{aligned}Q((4,2), \text{направо}) &= 0,8 + 0,1 (0,3 + 1(0,9) - 0,8) = \\&= 0,8 + 0,1 (0,3 + 1(0,9) - 0,8) = \\&= 0,84.\end{aligned}$$

# TD – 7 [3]

- **Алгоритм Q-обучения** один из наиболее используемых алгоритмов обучения с подкреплением:

1. Инициализация  $Q$ -функции произвольными значениями.
2. Выбор действия из состояния с использованием эпсилон-жадной стратегии ( $\varepsilon > 0$ ) и переход в новое состояние.
3. Обновление  $Q$  предыдущего состояния по следующему правилу:

$$Q(s, a) = Q(s, a) + \alpha(r + \gamma \max Q(s', a') - Q(s, a)).$$

4. Повторение шагов 3 и 4 до достижения завершающего состояния.

# TD – 8 [3]

- Алгоритм Q-обучения – пример с замерзшим озером.
- Допустим, мы находимся в состоянии (3,2) с двумя возможными действиями («налево» и «направо»).
- В Q-обучении действие выбирается с использованием эпсилон-жадной стратегии: либо с вероятностью epsilon исследуется новое действие, либо с вероятностью 1 – epsilon выбирается лучшее из известных действий. Допустим, с вероятностью epsilon было исследовано новое действие «вниз»:

	1	2	3	4
1	S	F	F	F
2	F	H	F	H
3	F	(F)	F	H
4	H	F	F	G

Вниз

Сост.	Действ.	Цен.
(3,2)	Направо	0,1
(3,2)	Налево	0,5
(3,2)	Вниз	0,8

## TD – 9 [3]

- Алгоритм Q-обучения – пример с замерзшим озером.
- Мы выполнили действие «вниз» в состоянии (3,2) и достигли нового состояния (4,2) с использованием эпсилон-жадной стратегии. Как обновить ценность предыдущего состояния (3,2) с использованием правила обновления?

Допустим, значение  $\alpha$  равно 0,1, а поправочный коэффициент – 1:

$$Q(s, a) = Q(s, a) + \alpha(r + \gamma \max Q(s', a) - Q(s, a));$$

$$Q((3,2), \text{вниз}) = Q((3,2), \text{вниз}) + 0,1 (0,3 + 1 \max [Q((4,2), \text{действие})] - Q((3,2), \text{вниз})).$$

# TD – 10 [3]

- Алгоритм Q-обучения – пример с замерзшим озером.
- Что такое  $\max [Q((4,2), \text{ действие})]$  для состояния  $(4,2)$ ? Исследованы только три действия («вверх», «вниз» и «направо»), поэтому максимум определяется только для этих действий. (Здесь эпсилон-жадная стратегия не применяется, а просто выбирается действие с максимальной ценностью.) Теперь можно подставить значения по приведенной Q-таблице:

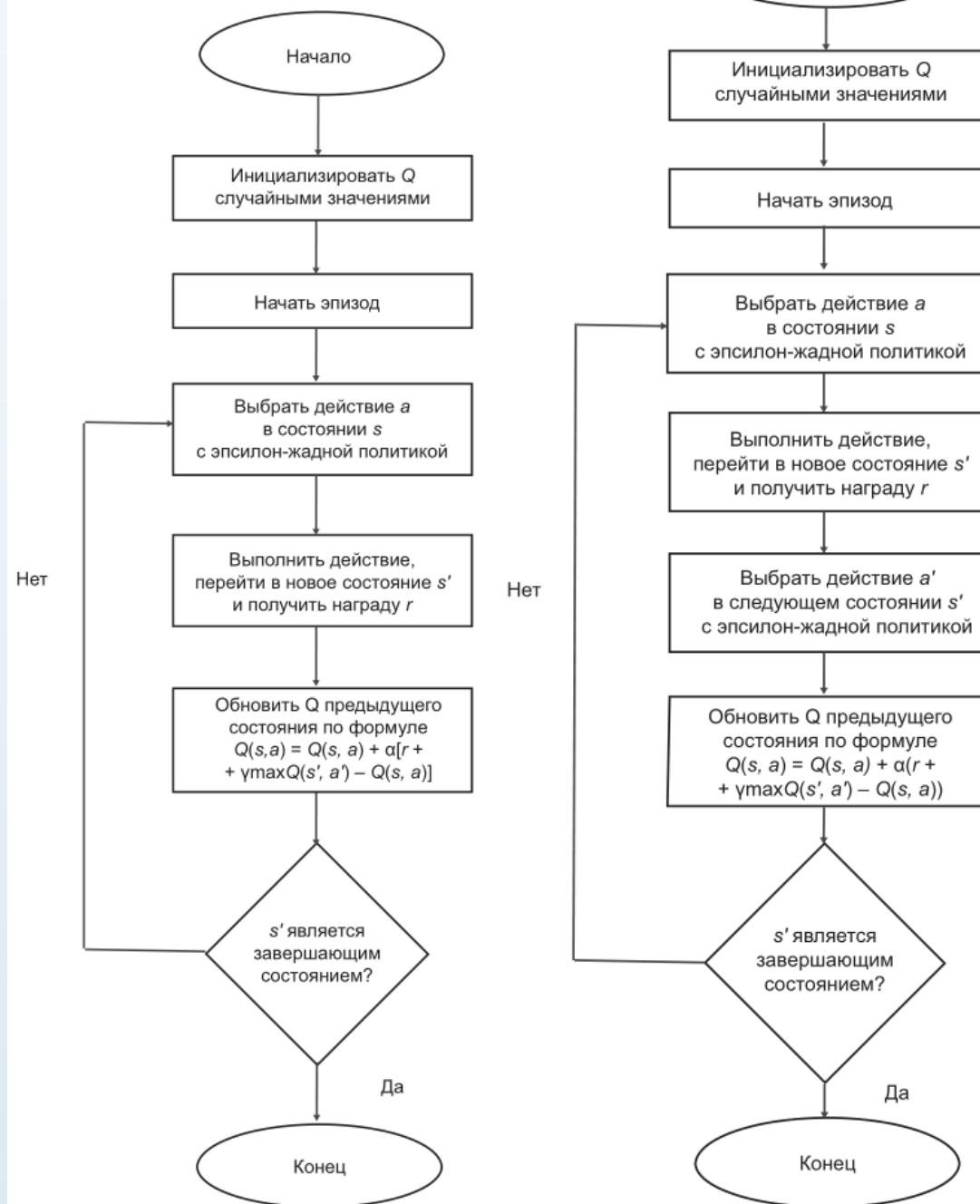
	1	2	3	4
1	S	F	F	F
2	F	H	F	H
3	F	(F)	F	H
4	H	F	F	G

Сост.	Действ.	Цен.
(3,2)	Направо	0,1
(3,2)	Налево	0,5
(3,2)	Вниз	0,8
(4,2)	Вверх	0,3
(4,2)	Вниз	0,5
(4,2)	Направо	0,8

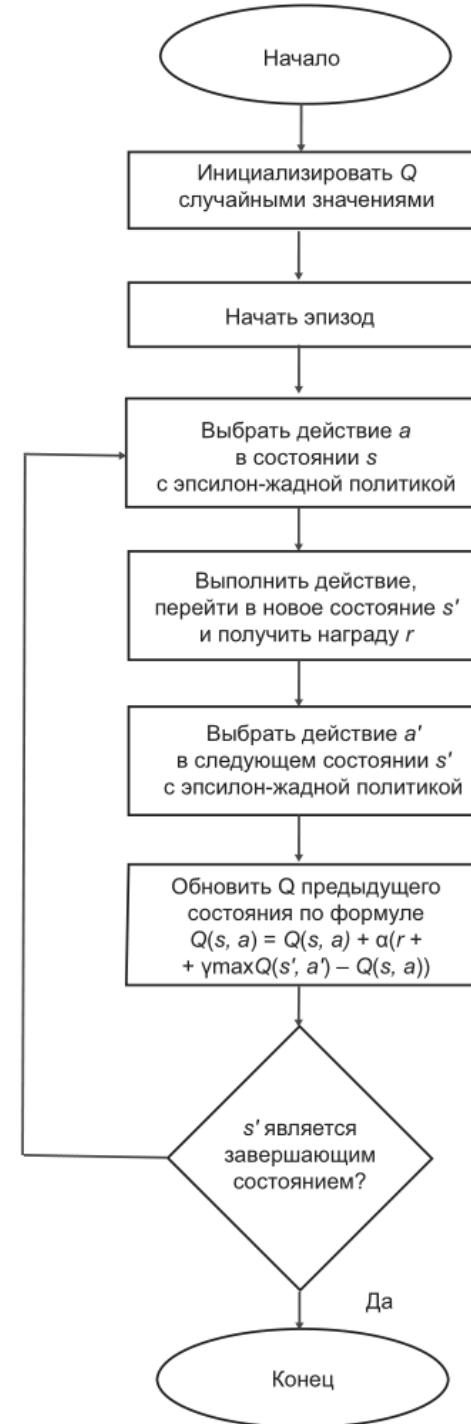
$$\begin{aligned}Q((3,2), \text{ вниз}) &= 0,8 + 0,1 (0,3 + 1\max [0,3, 0,5, 0,8] - 0,8) = \\&= 0,8 + 0,1(0,3 + 1(0,8) - 0,8) = \\&= 0,83.\end{aligned}$$

# TD – 11 [3]

## Q-обучение



## SARSA

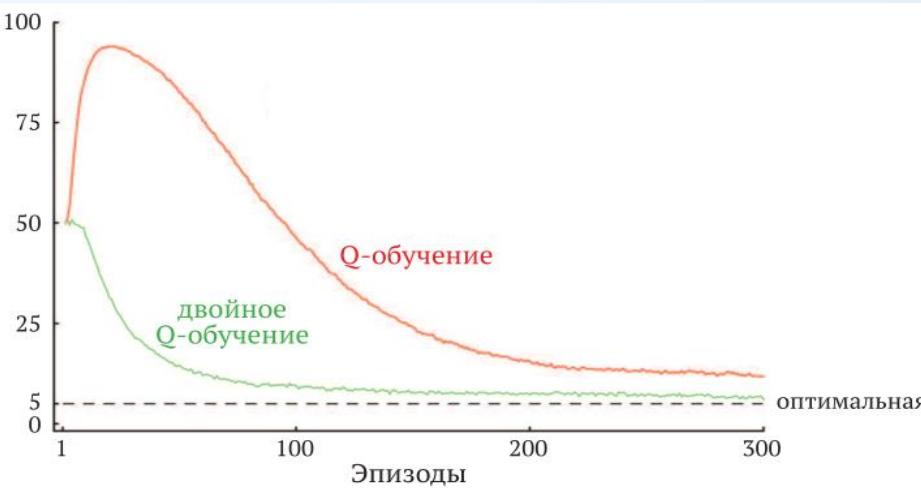


# TD – 12 [1]

- **Двойное Q-обучение.**
- В Q—обучении используется одна и та же выборка как для определения действия, доставляющего максимум, так и для оценки его ценности.
- Разобьем все множество игр на два подмножества и будем использовать их для обучения двух независимых оценок,  $Q_1(a)$  и  $Q_2(a)$  истинной ценности  $q(a)$  для всех действий  $a$ . Тогда можно было бы взять одну оценку  $Q_1$  для определения доставляющего максимум действия  $A^* = \text{argmax}_a Q_1(a)$ , а другую,  $Q_2$  для оценки ценности этого действия,  $Q_2(A^*) = Q_2(\text{argmax}_a Q_1(a))$ . Тогда эта оценка будет несмещенной в том смысле, что  $\mathbb{E}[Q_2(A^*)] = q(A^*)$ . Этот процесс можно повторить, поменяв обе оценки ролями, и получить тем самым вторую несмещенную оценку,  $Q_1$ .
- Хотя мы обучаем две оценки, при каждой игре обновляется только одна. Двойное обучение требует двойного объема памяти, но не увеличивает объем вычислений на каждом шаге.
- Обновление производится по правилу:

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha [R_{t+1} + \gamma \underset{a}{\text{argmax}} Q_1(S_{t+1}, a) - Q_1(S_t, A_t)]$$

# TD – 13 [1]



## Двойное Q-обучение для оценивания $Q_1 \approx Q_2 \approx q_*$

Параметры алгоритма: размер шага  $\alpha \in (0, 1]$ , небольшое  $\varepsilon > 0$

Инициализировать  $Q_1(s, a)$  и  $Q_2(s, a)$  для всех  $s \in \mathcal{S}^+$ ,  $a \in \mathcal{A}(s)$  произвольным образом с ограничением  $Q(\text{terminal}, \cdot) = 0$

Повторять для каждого эпизода:

    Инициализировать  $S$

    Повторять для каждого шага эпизода:

        Выбрать  $A$  в состоянии  $S$ , следуя  $\varepsilon$ -жадной стратегии относительно  $Q_1 + Q_2$

        Предпринять действие  $A$ , наблюдать  $R, S'$

        С вероятностью 0.5

$$Q_1(S, A) \leftarrow Q_1(S, A) + \alpha(R + \gamma \max_a Q_1(S', a) - Q_1(S, A))$$

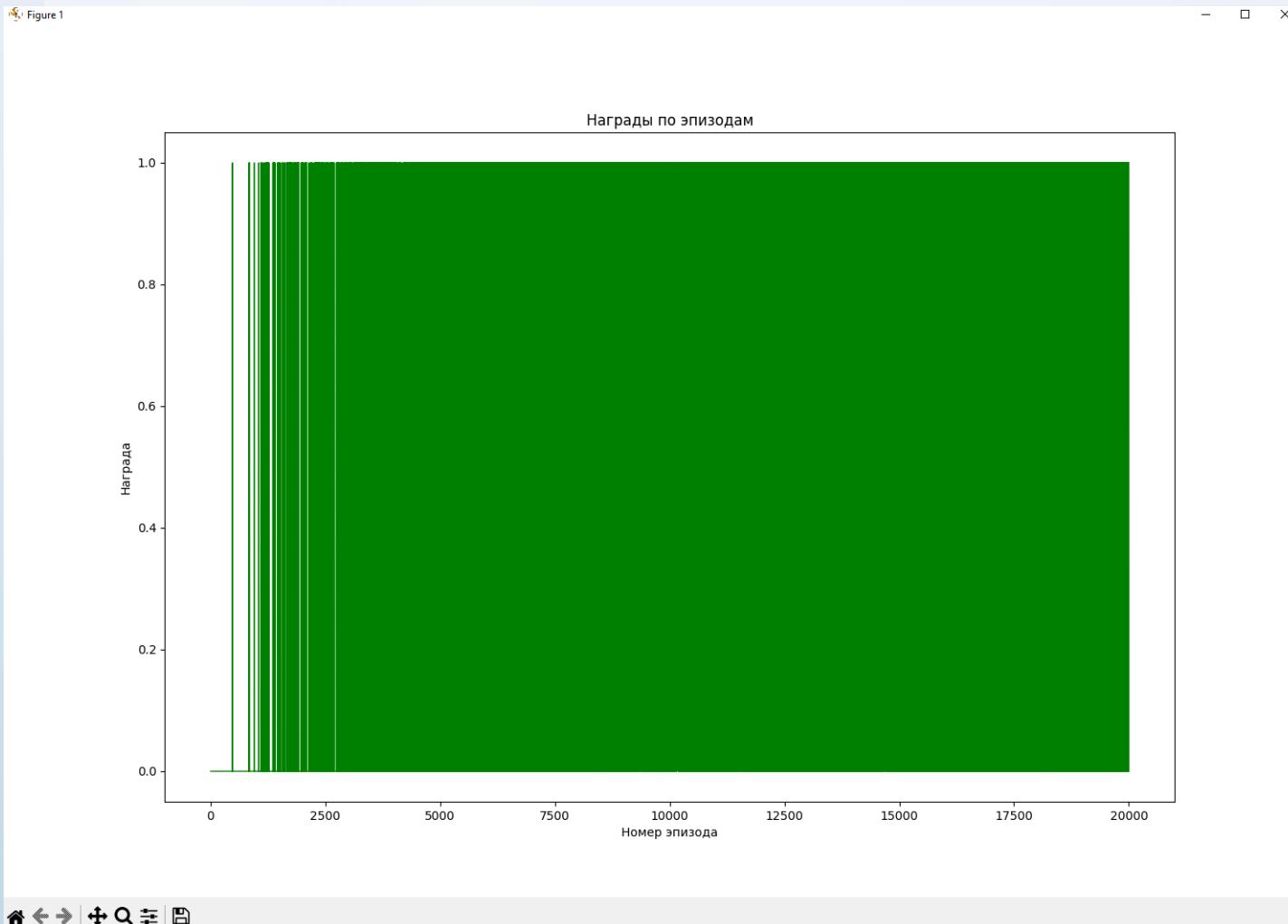
иначе

$$Q_2(S, A) \leftarrow Q_2(S, A) + \alpha(R + \gamma \max_a Q_2(S', a) - Q_2(S, A))$$

$$S \leftarrow S'$$

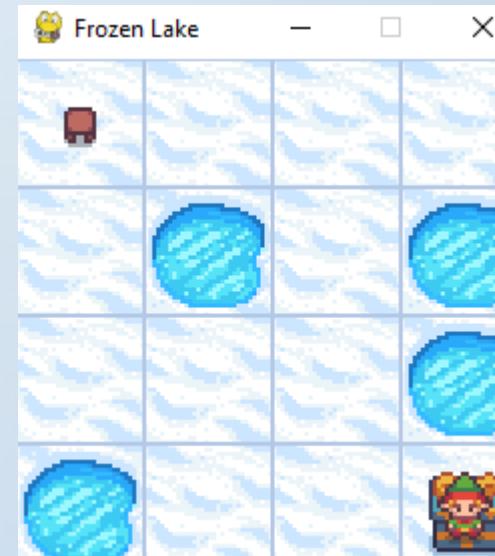
пока состояние  $S$  не является заключительным

# SARSA (реализация)

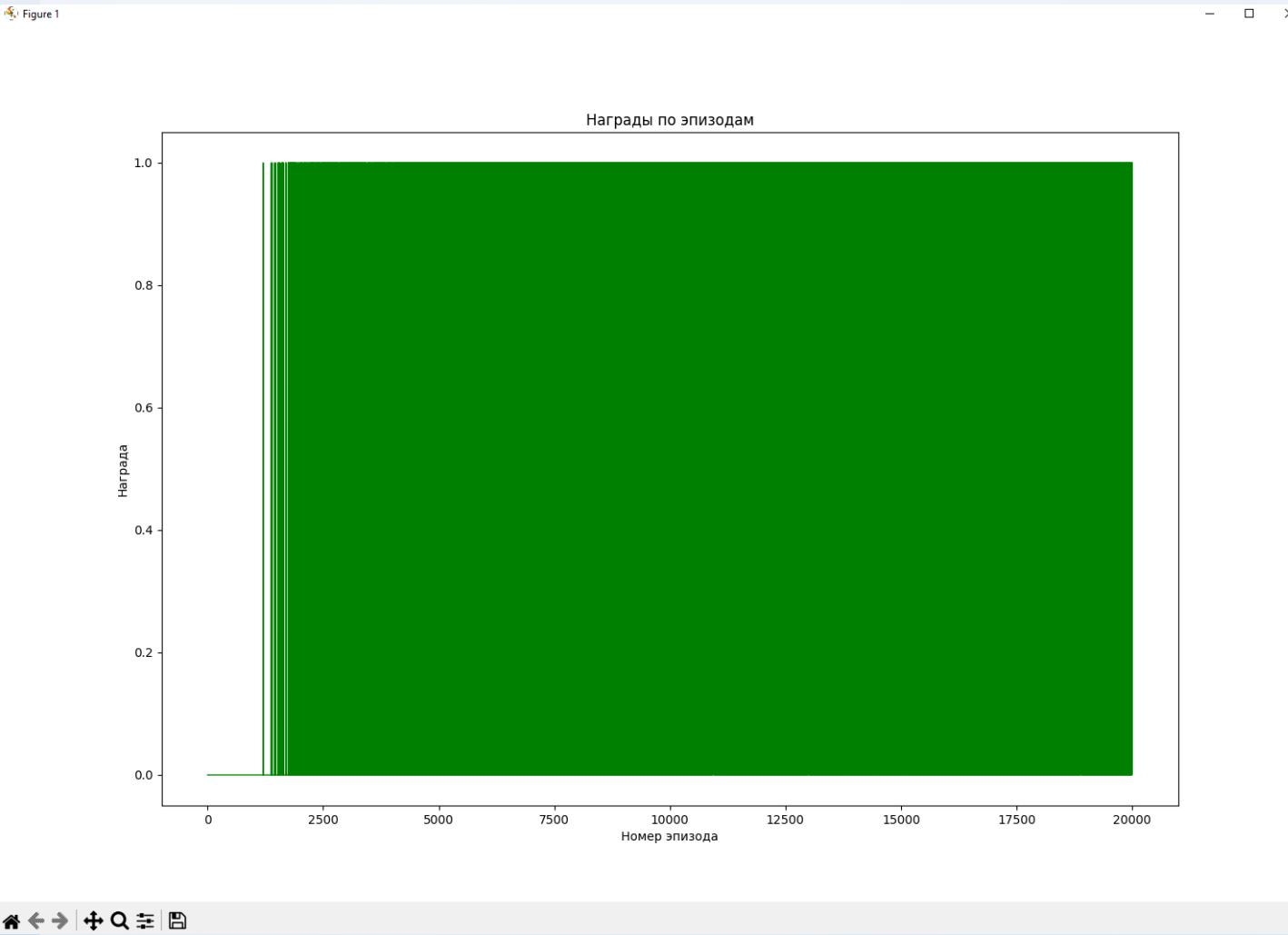


Вывод Q-матрицы для алгоритма SARSA

```
[[0.38216648 0.30723478 0.33378078 0.30713628]
 [0.23242619 0.23367427 0.13772201 0.31712797]
 [0.21795482 0.19988926 0.22011904 0.28291834]
 [0.11986107 0.10170468 0.08569299 0.22044406]
 [0.43065207 0.21561427 0.29686316 0.2646549 ]
 [0.          0.          0.          0.          ]
 [0.22519623 0.08317576 0.13788537 0.05957966]
 [0.          0.          0.          0.          ]
 [0.23069854 0.35512696 0.28555748 0.49891438]
 [0.38440052 0.57863489 0.42491115 0.37962661]
 [0.51987306 0.26088264 0.31532494 0.23344073]
 [0.          0.          0.          0.          ]
 [0.          0.          0.          0.          ]
 [0.23325215 0.3167114 0.68102546 0.38474884]
 [0.61575314 0.83543403 0.68455773 0.68125371]
 [0.          0.          0.          0.          ]]
```

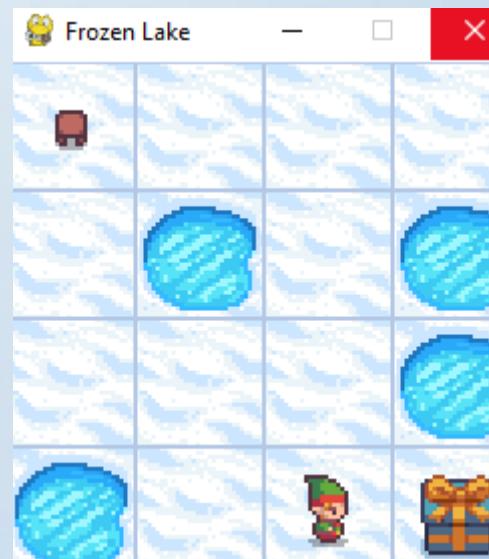


# Q-обучение (реализация)

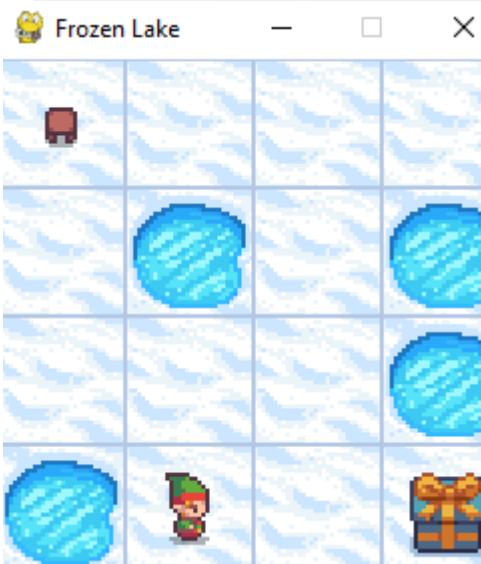
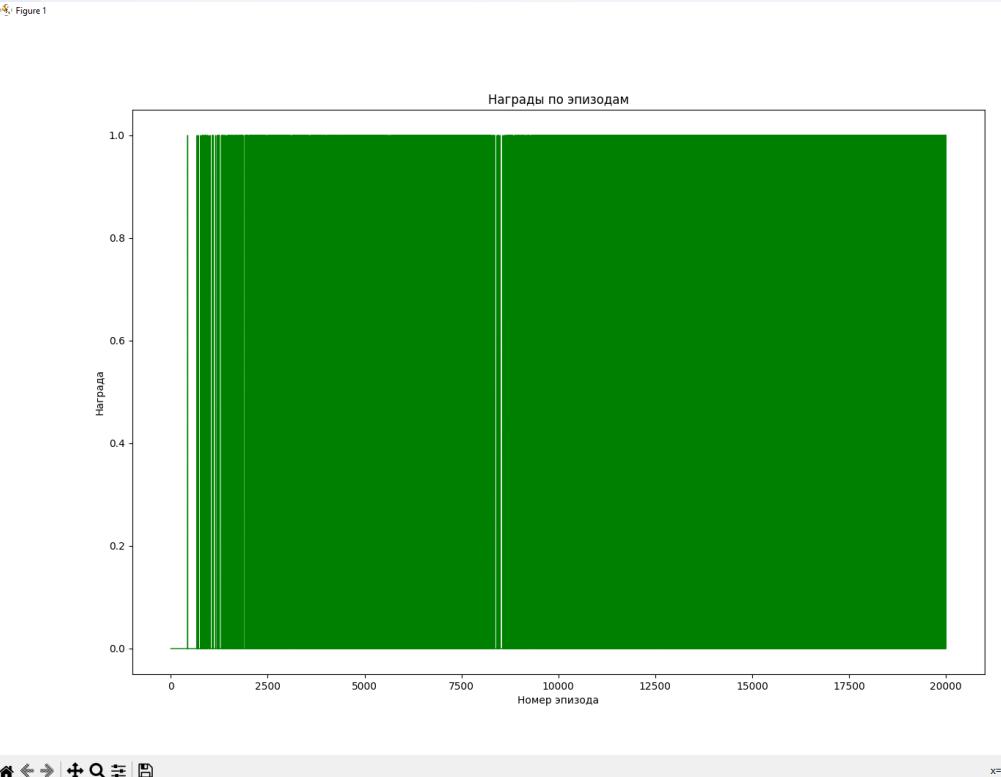


Вывод Q-матрицы для алгоритма Q-обучение

```
[[0.41895898 0.3636856 0.3706785 0.33596404]
 [0.25609021 0.24808708 0.2440506 0.37277415]
 [0.26989511 0.25609009 0.25695483 0.30877636]
 [0.18449975 0.21471353 0.14607019 0.28758917]
 [0.44963199 0.31589262 0.36953857 0.29142537]
 [0. 0. 0. 0. ]
 [0.12853205 0.11866465 0.35542996 0.11285844]
 [0. 0. 0. 0. ]
 [0.37486634 0.22520384 0.33614331 0.51469077]
 [0.43933738 0.56717126 0.38593626 0.3630791 ]
 [0.62260627 0.29403698 0.20464556 0.29897471]
 [0. 0. 0. 0. ]
 [0. 0. 0. 0. ]
 [0.35935024 0.45094139 0.67824738 0.44187873]
 [0.70859751 0.79012698 0.70832067 0.70295904]
 [0. 0. 0. 0. ]]
```



# Двойное Q-обучение (реализация)



Вывод Q-матриц для алгоритма Двойное Q-обучение

Q1

```
[[0.35314661 0.27427651 0.29243106 0.28460246]
 [0.20330257 0.13535115 0.07462703 0.34163522]
 [0.27956628 0.1978547 0.15509153 0.19280975]
 [0.01708955 0.0884367 0.02456676 0.03207056]
 [0.37639782 0.2102929 0.26645907 0.22386635]
 [0. 0. 0. 0.]
 [0.29971258 0.12240043 0.0728916 0.04271613]
 [0. 0. 0. 0.]
 [0.22845635 0.37392978 0.27926878 0.41614133]
 [0.34017609 0.46242552 0.33929408 0.38965475]
 [0.48357077 0.26921605 0.38567278 0.27074839]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0.46239467 0.52052533 0.54354514 0.48178331]
 [0.63411614 0.80283983 0.6327113 0.66497548]
 [0. 0. 0. 0.]]
```

Q2

```
[[0.35486092 0.2880091 0.28713354 0.29223306]
 [0.14407971 0.12711564 0.05758227 0.28373972]
 [0.26096882 0.14278114 0.10589478 0.14024671]
 [0.03742391 0.08722028 0.02882476 0.02464166]
 [0.37393245 0.20633217 0.25208296 0.27951412]
 [0. 0. 0. 0.]
 [0.32476767 0.09626511 0.11957979 0.02437922]
 [0. 0. 0. 0.]
 [0.19638027 0.33329458 0.30472086 0.42798371]
 [0.35909404 0.45999003 0.35690147 0.3284546]
 [0.5203945 0.40932188 0.34965732 0.18659782]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0.41351555 0.39446717 0.57842584 0.44249399]
 [0.59201276 0.75411454 0.63746887 0.6991378]
 [0. 0. 0. 0.]]
```

# Глубокое обучение с подкреплением

# Глубокие методы [2]

- Для чего используются глубокие методы в обучении с подкреплением?
- 1) Представим, что у нас есть среда FrozenLake размером  $31\ 600 \times 31\ 600$ , что примерно соответствует миллиарду состояний. Чтобы рассчитать даже грубую аппроксимацию для каждого состояния этой среды, понадобятся сотни миллиардов переходов, равномерно распределенных по состояниям, что почти невозможно с вычислительной точки зрения.
- 2) Среды Atari - <https://www.gymlibrary.dev/environments/atari/>
- Разрешение экрана составляет  $210 \times 160$  пикселов, а каждый пиксель окрашен в один из 128 цветов. Каждый кадр экрана содержит  $210 \times 160 = 33\ 600$  пикселов, а общее количество экранов, отличающихся хотя бы одним пикселом, —  $128^{33\ 600}$ , что немного больше, чем  $10^{70\ 802}$ .
- Если просто перечислить все возможные состояния Atari за один раз, то даже у суперкомпьютера на это уйдут миллиарды лет. Кроме того, большая часть этой работы окажется пустой тратой времени, так как большинство комбинаций никогда не будет показано в течение даже длительного игрового процесса.
- **Проблема:** рассмотренные ранее методы обучения с подкреплением хорошо работают только для ограниченных сред.
- **Решение:** использование методов глубокого обучения в качестве модели агента. Нейронная сеть является универсальным аппроксиматором и позволяет находить приближенные значения.

# Глубокий Value-based подход **Алгоритм Deep Q Network (DQN) и его модификации**

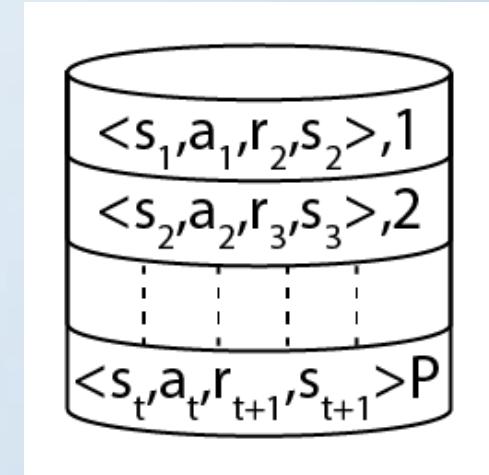
Value-based подход подразумевает, что алгоритм ищет не саму стратегию, а оптимальную Q-функцию.

# DQN – 1 [3]

- Deep Q Network (DQN) — один из наиболее известных алгоритмов глубокого обучения с подкреплением (Deep RL).
- Является разновидностью Q-обучения.
- Основная идея состоит в том, чтобы построить приближение Q-функции с некоторым параметром  $\theta$  в форме  $Q(s, a; \theta) \approx Q^*(s, a)$ .
- Используется нейронная сеть, чтобы аппроксимировать значение Q для всех возможных действий в каждом состоянии, ее принято называть Q-сетью.
- В данном разделе будут использованы примеры:
  - [https://pytorch.org/tutorials/intermediate/reinforcement\\_q\\_learning.html](https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html)
  - [https://pytorch.org/tutorials/intermediate/mario\\_rl\\_tutorial.html](https://pytorch.org/tutorials/intermediate/mario_rl_tutorial.html)

# DQN – 2 [3, 8, <https://paperswithcode.com/method/experience-replay> ]

- **Experience Replay (Replay Memory, Replay Buffer)**
- Воспроизведение опыта (буфер память воспроизведения) - это метод запоминания повторений, используемый в обучении с подкреплением, когда сохраняется опыт агента на каждом шаге в наборе данных, **объединенных по многим эпизодам** в память воспроизведения.
- Данные из памяти воспроизведения выбираются случайно и используются для обучения в методах на основе value-based подхода.
- Такой подход решает проблему автокорреляции, приводящую к нестабильному обучению, делая проблему более похожей на обучение с учителем.



# DQN – 3 [3, 8]

- **Описание алгоритма – 1**
- Необходимо найти политику, которая оптимизирует награду:

$$\pi^*(s) = \operatorname{argmax}_a Q^*(s, a)$$

- Поскольку  $Q^*$  неизвестна, то будем аппроксимировать ее нейронной сетью. Q-функция задается уравнением Беллмана:

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

- Необходимо минимизировать ошибку временных различий:

$$\delta = Q(s, a) - (r + \gamma \max_a Q(s', a))$$

# DQN – 4 [3, 8]

- **Описание алгоритма – 2**
- В некоторых модификациях алгоритма в качестве функции ошибки используется квадратичная функция потерь  $\delta^2$ , в рассматриваемом примере будем использовать функцию потерь Хьюбера:

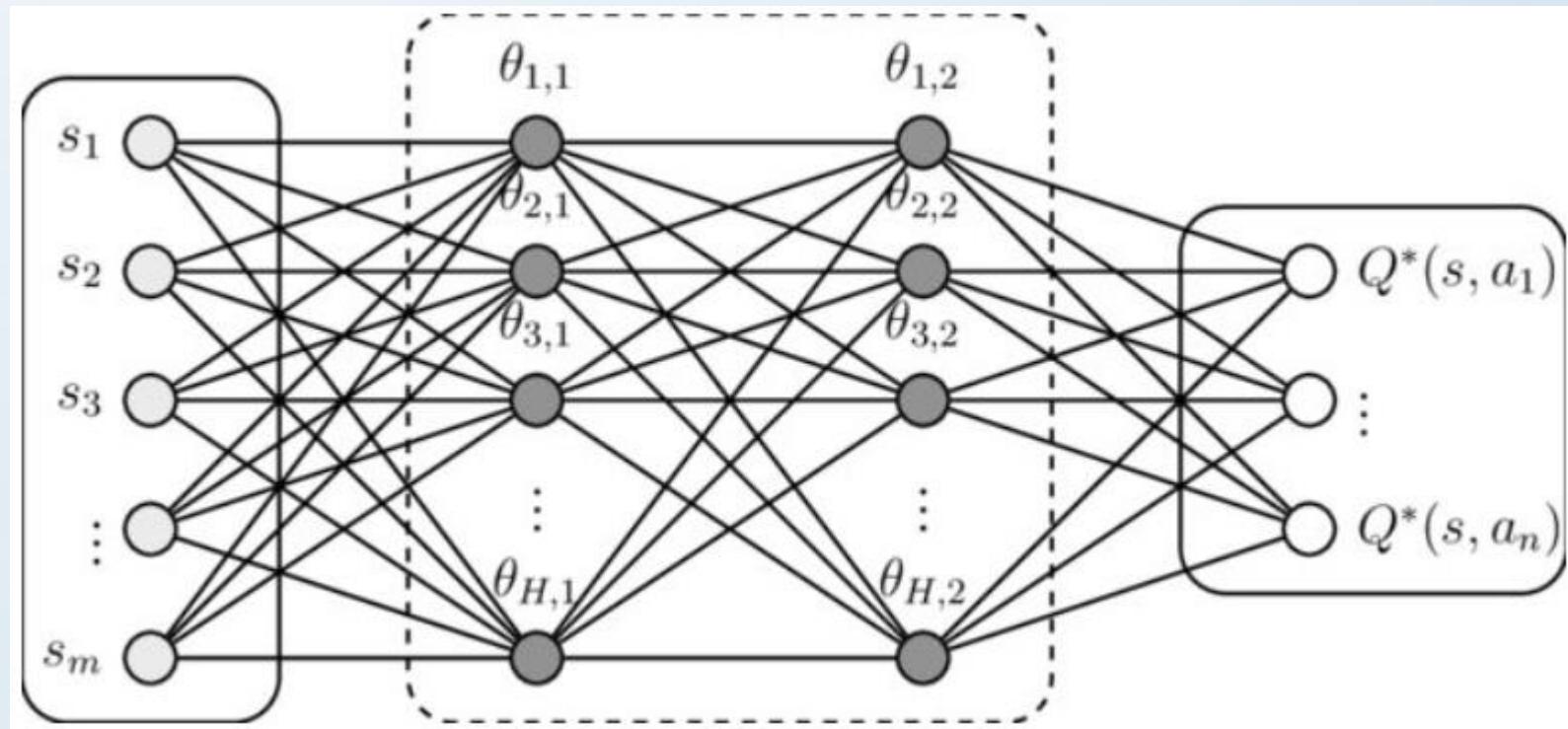
$$\mathcal{L} = \frac{1}{|B|} \sum_{(s,a,s',r) \in B} \mathcal{L}(\delta)$$

where  $\mathcal{L}(\delta) = \begin{cases} \frac{1}{2}\delta^2 & \text{for } |\delta| \leq 1, \\ |\delta| - \frac{1}{2} & \text{otherwise.} \end{cases}$

- где  $B$  – batch, сэмплируемый из Replay Memory.

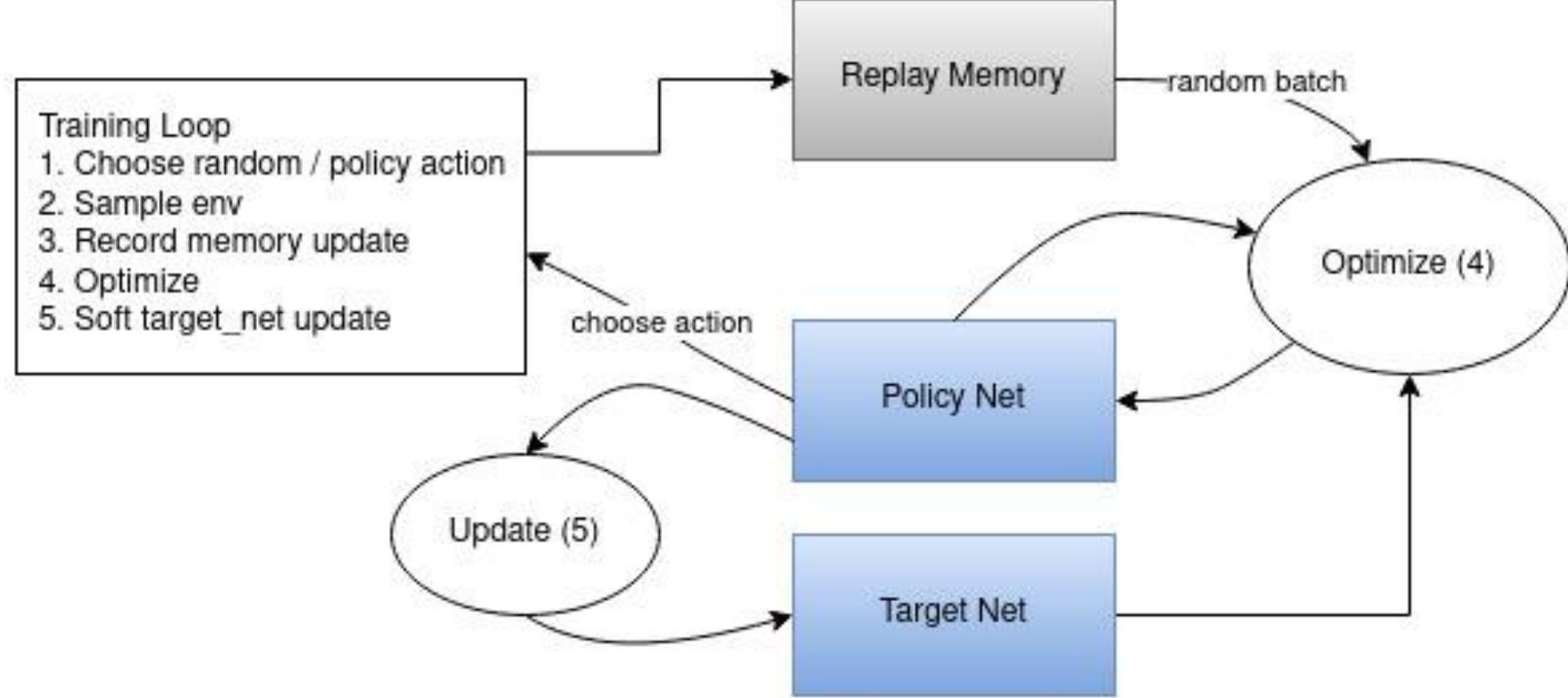
# DQN – 5 [3, 8]

- В примере используется среда Cart Pole  
[https://www.gymlibrary.dev/environments/classic\\_control/cart\\_pole/](https://www.gymlibrary.dev/environments/classic_control/cart_pole/)
- где B – batch, сэмплируемый из Replay Memory.
- Примерная структура нейронной сети:



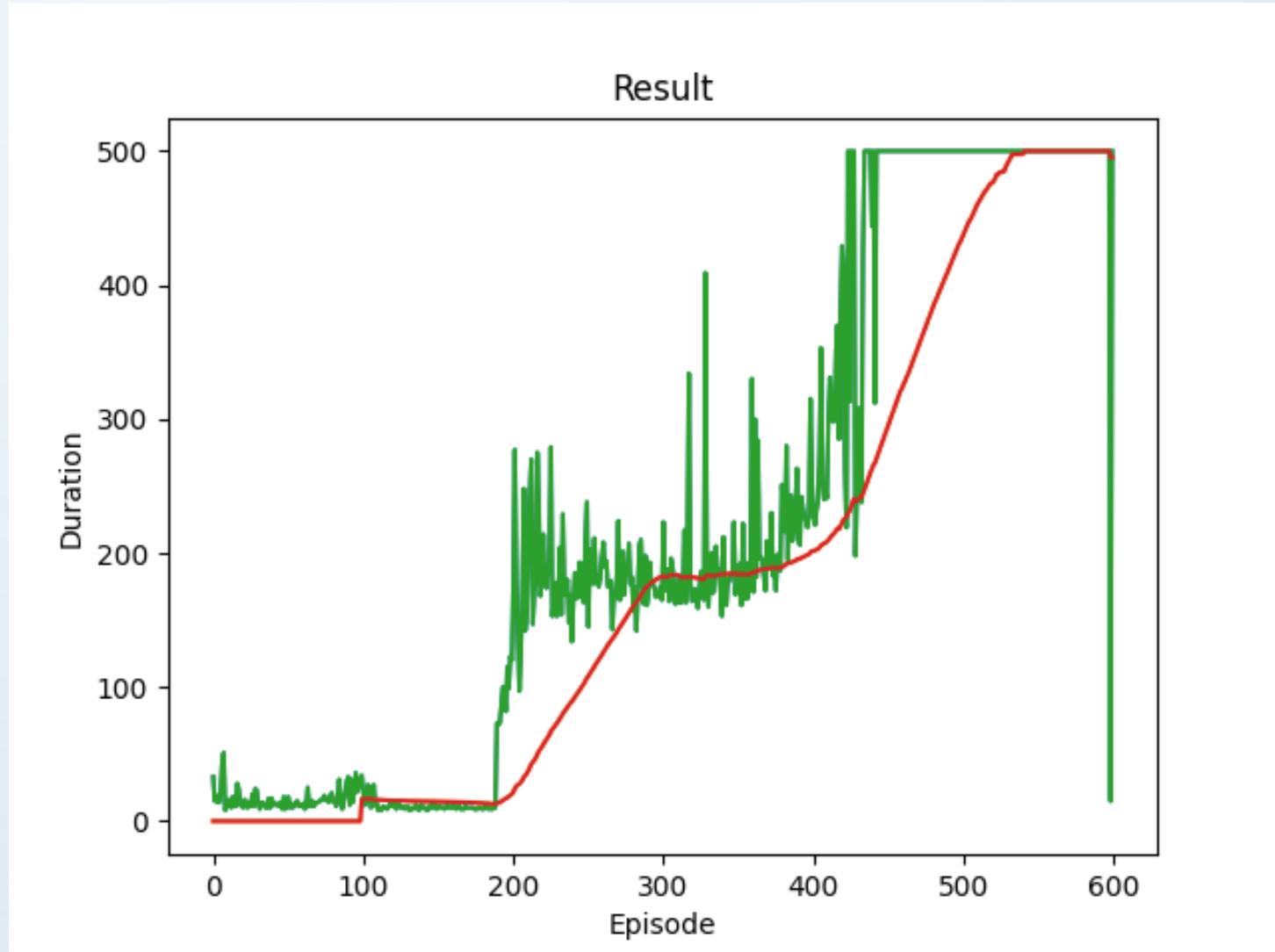
# DQN – 6 [3, 8]

- Схема работы:



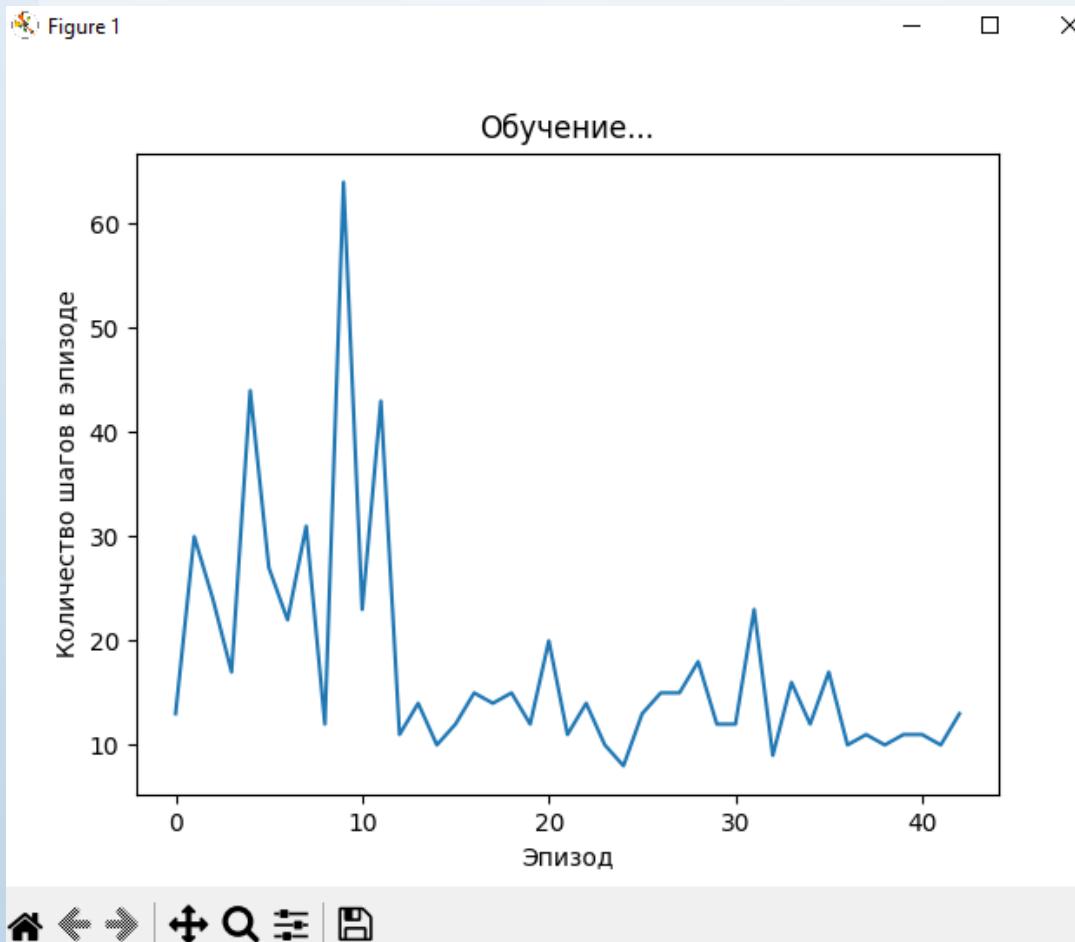
# DQN – 7 [3, 8]

- Обучение на GPU:



# DQN – 8 [3, 8]

- Эксперименты с примером «dqn\_1.py»:
- Данные об эпизоде:  $[(1, 1.0), (0, 1.0), (1, 1.0), (1, 1.0), (1, 1.0), (1, 1.0), (1, 1.0), (0, 1.0), (0, 1.0), (0, 1.0), (1, 1.0), (1, 1.0), (0, 1.0), (1, 1.0), (1, 1.0)]$  – в формате (действие, награда)

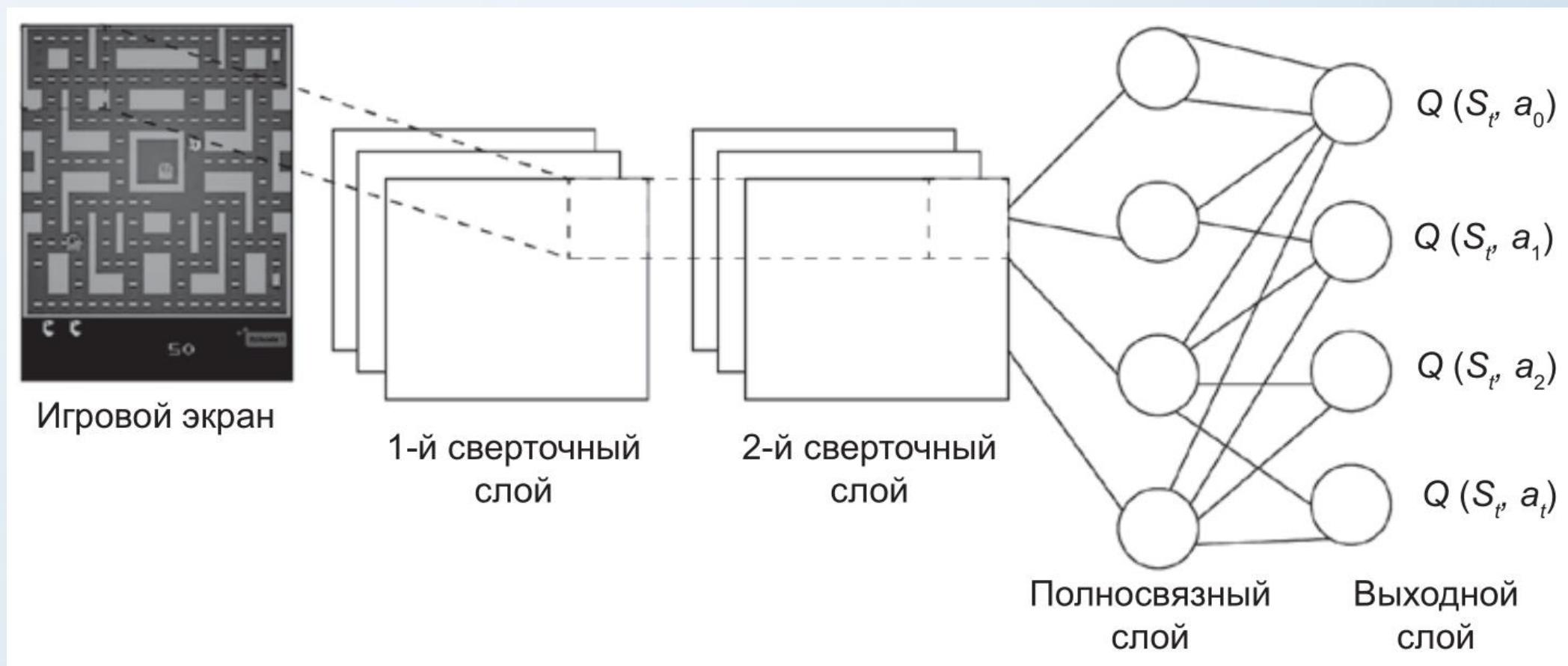


pygame window

```
ALGO_NAME = ----
def __init__(self, env, eps=0.1):
    # Среда
    self.env = env
    # Размерности Q-матрицы
    self.nA = env.action_space.n
    self.nS = env.observation_space.n
    # и сама матрица
    self.Q = np.zeros((self.nS, self.nA))
    # Значения коэффициентов
    self.c = np.zeros(self.nA)
    # Порог выбора случайной
    self.thresh = 0.05
```

# DQN – 9 [3, 8]

- Пример реализации игр Atari
- [https://pytorch.org/tutorials/intermediate/mario\\_rl\\_tutorial.html](https://pytorch.org/tutorials/intermediate/mario_rl_tutorial.html)
- Структура нейронной сети:



# DQN – 10 [3, 8]

- **Двойная сеть DQN.**
- Одна из проблем с сетями DQN заключается в том, что они склонны переоценивать значения Q. Это происходит из-за оператора `max` в уравнении Q-обучения. Оператор `max` использует одно значение для выбора и оценки действия.
- Для решения этой задачи можно определить две разные независимо обучаемые Q-функции. Одна будет использоваться для выбора действия, а другая — для оценки:

$$y_i^{\text{DQN}} = r + \gamma \max_{a'} Q(s', a'; \theta')$$

$$y_i^{\text{DoubleDQN}} = r + \gamma Q(s, \operatorname{argmax} Q(s, a; \theta^-); \theta')$$

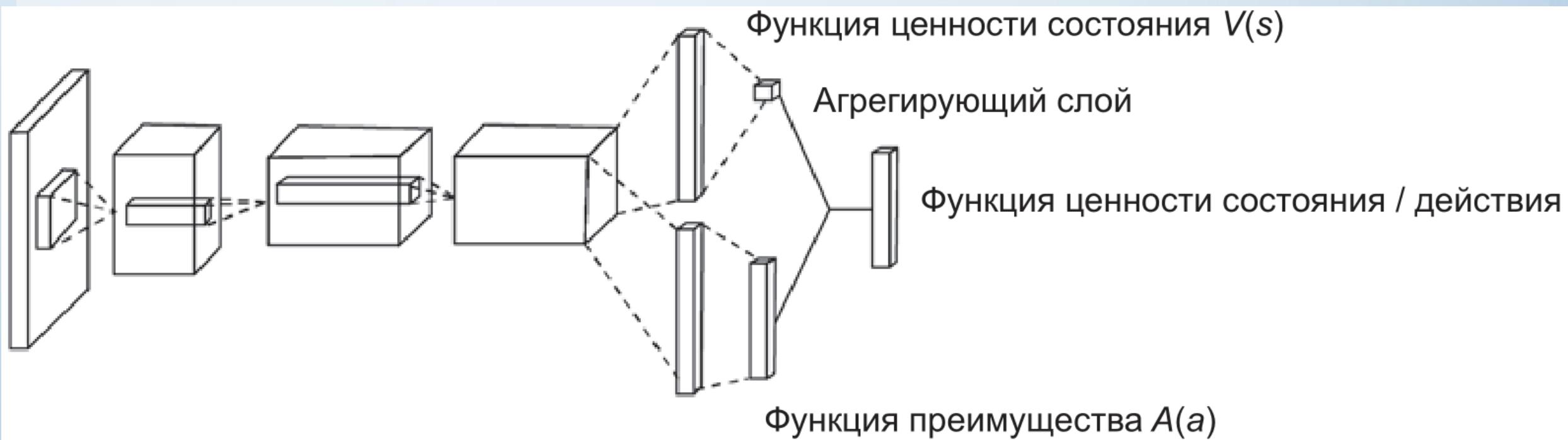
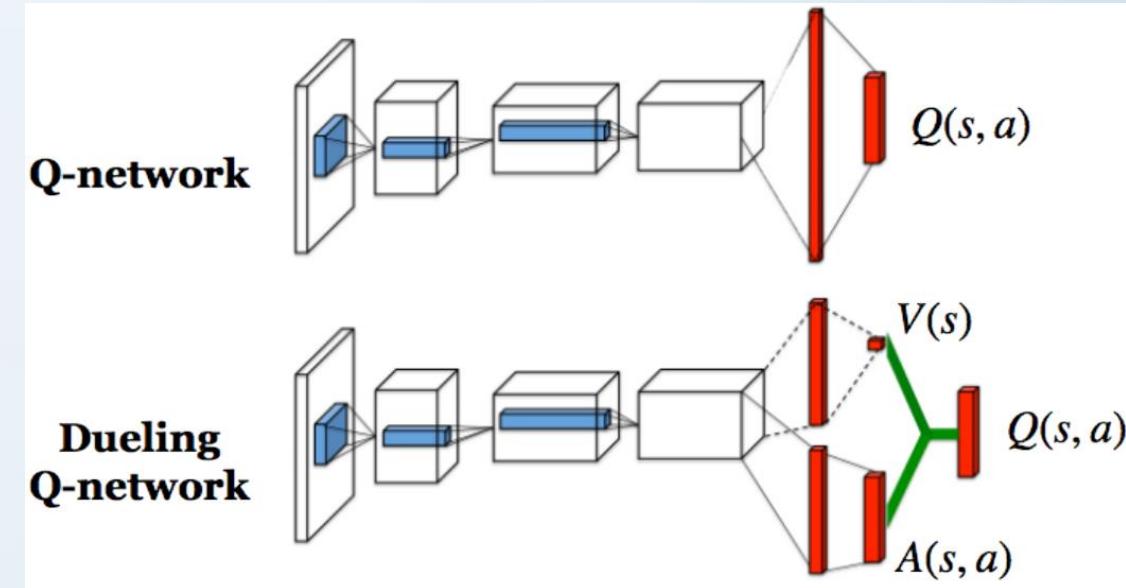
- Таким образом, используются две Q-функции с разными весами. Q-функция с весами  $\theta'$  используется для выбора действия, а другая Q-функция с весами  $\theta^-$  — для оценки действия. Также можно поменять местами роли этих двух Q-функций.

# DQN – 11 [3, 8]

- **Дуэльная сеть DQN – 1.**
- Q-функция определяет, насколько хорошо для агента выполнение действия  $a$  в состоянии  $s$ , а функция ценности определяет, насколько хорошо для агента пребывание в состоянии  $s$ .
- Введем новую функцию, называемую функцией преимущества; она может быть определена как разность между функцией ценности и средней ценности этого состояния.
- Функция преимущества указывает, насколько хорошо для агента выполнение данного действия по сравнению с другими действиями.
- Таким образом, функция ценности задает желательность состояния, а функция преимущества задает желательность действия.
- Что произойдет, если объединить эти функции? Результат сообщит, насколько желательно для агента выполнение действия  $a$  в состоянии  $s$ , то есть фактически мы получим Q-функцию.
- Таким образом, Q-функцию можно определить как сумму функции ценности и функции преимущества:  $Q(s, a) = V(s) + A(a)$ .

# DQN – 12 [3, 8]

- Дуэльная сеть DQN – 2.
- Архитектура дуэльной сети:



# DQN – 13 [3, 8]

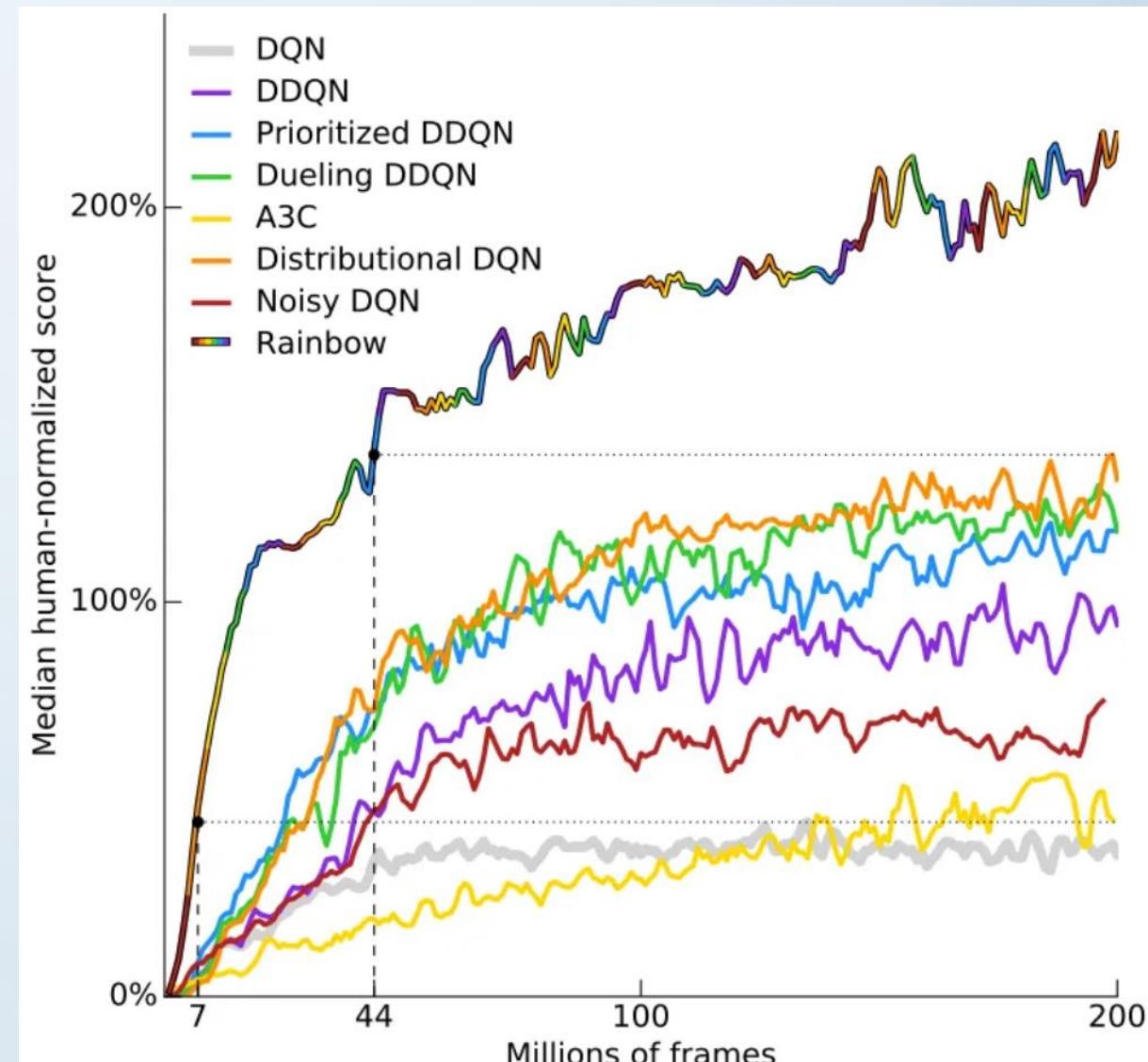
- **Дуэльная сеть DQN – 3.**
- Архитектура дуэльной DQN практически не отличается от архитектуры DQN, не считая того, что полносвязный уровень в конце делится на два потока. Один поток вычисляет функцию ценности, а другой — функцию преимущества. В конце эти два потока объединяются агрегирующим слоем, и получается Q-функция.
- Зачем разбивать вычисление Q-функции на два потока? Во многих состояниях вычисление оценок ценности всех действий не так уж важно, особенно если состояние имеет большое пространство действий; большинство действий не будет влиять на состояние. В таких случаях дуэльная сеть DQN оценивает Q точнее обычной архитектуры DQN:
  - Первый поток (поток функции ценности) полезен при большом количестве действий в состоянии, а также в ситуациях, в которых не обязательно оценивать ценность каждого действия.
  - Второй поток (поток функции преимущества) полезен в тех случаях, когда сети приходится решать, какое действие является предпочтительным по сравнению с другими.
- Агрегирующий слой объединяет значения этих двух потоков и формирует Q-функцию. Таким образом, дуэльная сеть превосходит стандартную архитектуру DQN по эффективности и надежности.

# DQN – 14 [3, 8]

- Noisy DQN
  - Реализуют добавление шума с обучающей и, главное, зависимой от состояния (входа в модель) дисперсией. Являются более надежными по сравнению с эпсилон-жадным подходом.
- Prioritized Experience Replay
  - Чтобы улучшить обучение, можно выбирать кортежи, которые в прошлом приводили к наиболее значимым ошибкам с идеей, что мы хотим снова оценить этот опыт и уменьшить ошибку.
- Multi-step DQN
  - Учитывается награда не за один шаг, а за несколько шагов, что делает алгоритм более надежным.
- Distributional RL
  - Вместо значений наград рассматриваются их распределения, что делает алгоритм более устойчивым.

# DQN – 15 [3, 8]

- **Rainbow DQN.**
- Считается одним из лучших алгоритмов глубокого обучения. Совмещает 6 модификаций алгоритма DQN:
  - Double DQN
  - Dueling DQN
  - Noisy DQN
  - Prioritized Experience Replay
  - Multi-step DQN
  - Distributional RL
- Алгоритм Rainbow, сочетающий в себе шесть методов, был протестирован по сравнению с алгоритмами DQN, использующими по одному методу.
- На приведенном рисунке представлена средняя производительность по 57 играм Atari, где 100% соответствует производительности на уровне человека.
- <https://paperswithcode.com/method/rainbow-dqn>
- <https://towardsdatascience.com/rainbow-dqn-the-best-reinforcement-learning-has-to-offer-166cb8ed2f86>



Empirical performance, averaged over 57 Atari games. Rainbow DQN strongly outperforms individual DQN techniques, both in terms of rate of learning and final performance [source: DeepMind paper]

# Policy Gradient подход Алгоритм REINFORCE и семейство Actor-Critic

# Policy Gradient – 1 [2, 8]

- **Policy Gradient – градиенты по стратегиям.** Почему интересен анализ стратегий?
  1. Стратегия — это то, что мы ищется при решении задачи RL. Когда агент получает наблюдение и должен принять решение о том, что делать дальше, нужна прежде всего стратегия, а не ценность состояния или отдельно взятое действие.
  2. Другая причина, по которой стратегии могут быть более привлекательными, чем ценности, связана со средами с большим количеством действий или в предельном случае с пространством непрерывных действий. Если действие не является небольшим дискретным набором, но имеет скалярное значение, привязанное к нему, например угол поворота руля или скорость, то задача оптимизации усложняется, так как Q обычно представляется сильно нелинейной нейронной сетью, поэтому поиск аргумента, который максимизирует функцию значения ценности, может быть трудным. В таких случаях гораздо целесообразнее избегать ценностей и работать со стратегией напрямую.
  3. Стратегия естественным образом представляется как вероятность действий, поэтому данный подход особенно интересен в случае стохастических сред.

# Policy Gradient – 2 [2, 8]

- В методах Policy Gradient не используется подход, основанный на вычислении Q. Используется подход аналогичный матрице стратегии – возвращается вероятностное распределение действий, которое задается выходами нейронной сети.
- Для N взаимоисключающих действий возвращаются N чисел, представляющих вероятность выполнения каждого действия в данном состоянии, которое передается в качестве входа нейронной сети.
- Дополнительным преимуществом такого представления действий, как вероятность, является его гладкость: если немного изменить веса сети, выход сети также изменится. При этом даже небольшое изменение весов может привести к переходу к другому действию.
  - Если вывод — это распределение вероятностей, то небольшое изменение весов обычно дает незначительное изменение распределения выходных значений, например увеличение вероятности одного действия по сравнению с другими.
  - Это очень важно, так как методы градиентной оптимизации прежде всего выполняют небольшое изменение параметров модели для улучшения результатов.

# Policy Gradient – 3 [2, 8]

- **Отличия Policy Gradient от DQN.**
- Методы, основанные на стратегии, напрямую оптимизируют то, что нам нужно: поведение агента. Методы, основанные на ценностях, такие как DQN, делают то же самое косвенно, сначала выучивая значение ценности, а затем вырабатывая основанную на нем стратегию.
- Методы, основанные на стратегии, — это методы онлайн-обучения, они запрашивают свежие примеры из среды. Методы, основанные на ценностях, могут использовать старые данные, основанные на старой стратегии, продемонстрированные человеком или полученные из других источников.
- Методы, основанные на стратегии, обычно менее эффективно используют данные, это значит, что им требуется больше взаимодействовать со средой. Методы, основанные на ценностях, могут работать с большими буферами примеров. Тем не менее эффективное использование данных среды не предполагает того, что основанные на ценностях методы эффективны в вычислительном отношении и зачастую верно обратное. В приведенном ранее примере во время обучения нам нужно обратиться к НС только один раз, чтобы получить вероятности действий. В DQN нам требуется обработать обучающий набор два раза: один — для текущего состояния, другой — для следующего состояния в обновлении функции Беллмана.

# Policy Gradient – 4 [1, 2, 8]

- **Теорема о градиенте стратегии.**
- Дает аналитическое выражение градиента меры качества по параметру стратегии (это именно то, что необходимо аппроксимировать для применения градиентного спуска), не содержащее производной распределения состояний. Теорема о градиенте стратегии для эпизодического случая утверждает, что:

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a q_\pi(s, a) \nabla \pi(a | s, \boldsymbol{\theta})$$

- где градиенты представлены векторами-столбцами, состоящими из частных производных по элементам  $\theta$ , а  $\pi$  обозначает стратегию, соответствующую вектору
- параметров  $\theta$ . Символ  $\propto$  означает «пропорционально».

# Policy Gradient – 5 [1, 2, 8]

- Алгоритм REINFORCE.

1. Инициализировать сеть случайными весами.
2. Проиграть  $N$  полных эпизодов, сохраняя их переходы  $(s, a, r, s')$ .
3. Для каждого шага  $t$  каждого эпизода  $k$  рассчитать дисконтированное значение общего вознаграждения для следующих друг за другом шагов  $Q_{k,t} = \sum_{i=0}^t \gamma^i r_i$ .
4. Рассчитать функцию потерь для всех переходов  $L = -\sum_{k,t} Q_{k,t} \log(\pi(s_{k,t}, a_{k,t}))$ .
5. Обновить веса сети, используя SGD и минимизируя потери.
6. Повторять, начиная с шага 2, до достижения сходимости.

# Policy Gradient – 6 [1, 2, 8]

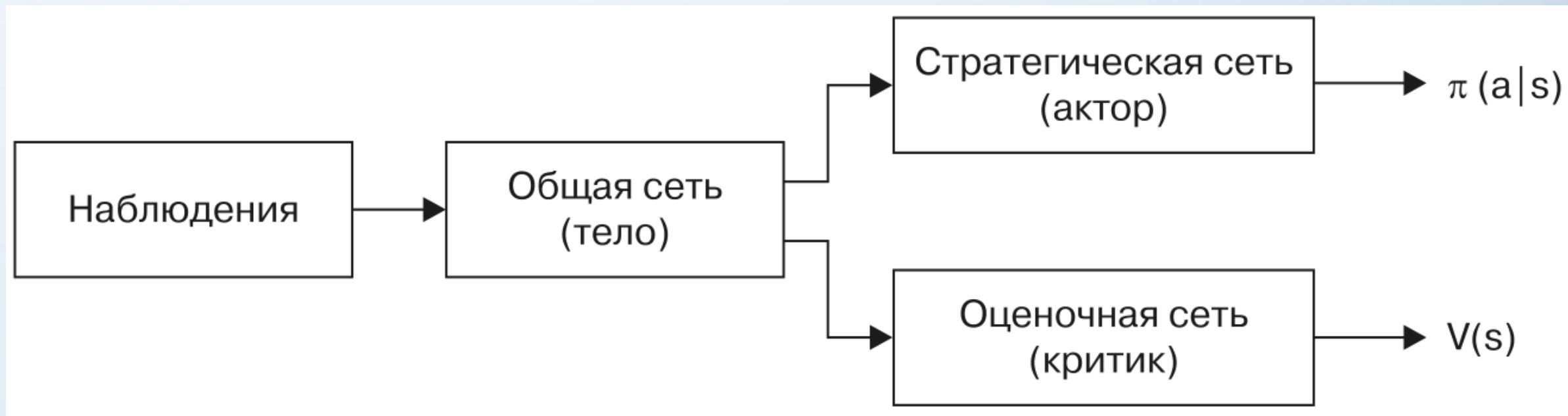
- **Алгоритм REINFORCE имеет несколько важных отличий от Q-обучения.**
- Нет необходимости в явном исследовании среды. В Q-обучении использовалась эпсилон-жадная стратегия, чтобы исследовать среду и не дать агенту действовать в соответствии с неоптимальной стратегией. Теперь с вероятностями, возвращаемыми сетью, исследование выполняется автоматически. В начале сеть инициализируется случайными весами и возвращает равномерное распределение вероятностей. Это распределение соответствует случайному поведению агента.
- Не используется Experience Replay (воспроизведение опыта), так как нельзя обучаться на данных, полученных из старой стратегии. Это и хорошо и плохо. Плюс в том, что такие методы обычно сходятся быстрее. Минус в том, что они обычно требуют гораздо большего взаимодействия со средой, чем методы онлайн-обучения, такие как DQN.
- Не нужна целевая сеть. В DQN использовалась целевая сеть, чтобы нарушить корреляцию в аппроксимации Q-значений, но Q-значения больше не аппроксимируются.

# Policy Gradient – 7 [1, 2, 8]

- **Алгоритмы семейства Actor-Critic (Исполнитель-Критик)**
- Характерной особенностью всех алгоритмов семейства является использование двух нейронных сетей:
  1. Стратегическая сеть, которая возвращает распределение вероятностей действий, называется актором, поскольку говорит, что делать.
  2. Другая сеть называется критиком, так как она позволяет понять, насколько хороши были действия агента.
- На практике стратегическая и оценочная сети частично перекрываются, главным образом из-за соображений эффективности и сходимости. В этом случае стратегия и ценность реализуются как разные головы сети, получая выходные данные из общего тела и преобразуя их в распределение вероятностей и одно число, представляющее ценность состояния. Это помогает обеим сетям совместно задействовать функции низкого уровня, но комбинировать их по-разному.

# Policy Gradient – 8 [1, 2, 8]

- Алгоритм A2C (Advantage Actor-Critic)
- Архитектура сети:



# Policy Gradient – 9 [1, 2, 8]

- **Алгоритм A2C (Advantage Actor-Critic)**

1. Инициализируем параметры сети  $\theta$  случайными значениями.
2. Проигрываем  $N$  шагов в среде, применяя текущую стратегию  $\pi_\theta$ , сохраненные состояние  $s_t$ , действие  $a_t$ , вознаграждение  $r_t$ .
3.  $R = 0$ , если достигнут конец эпизода, или  $V_\theta(s_t)$ .
4. Для  $i = t - 1 \dots t_{\text{start}}$  (обратите внимание на то, что шаги обрабатываются в обратном порядке):
  - $R \leftarrow r_i + \gamma R$ ;
  - накапливаем градиенты по стратегиям  $\partial\theta_\pi \leftarrow \partial\theta_\pi + \nabla_\theta \log \pi_\theta(a_i | s_i) (R - V_\theta(s_i))$ ;
  - накапливаем градиенты значений  $\partial\theta_v \leftarrow \partial\theta_v + \frac{\partial (R - V_\theta(s_i))^2}{\partial\theta_v}$ .
5. Обновляем параметры сети, используя накопленные градиенты, перемещаемся в направлении градиентов по стратегиям  $\partial\theta_\pi$  и в противоположном направлении от градиентов значений  $\partial\theta_v$ .
6. Повторяем, начиная с шага 2, до достижения сходимости.

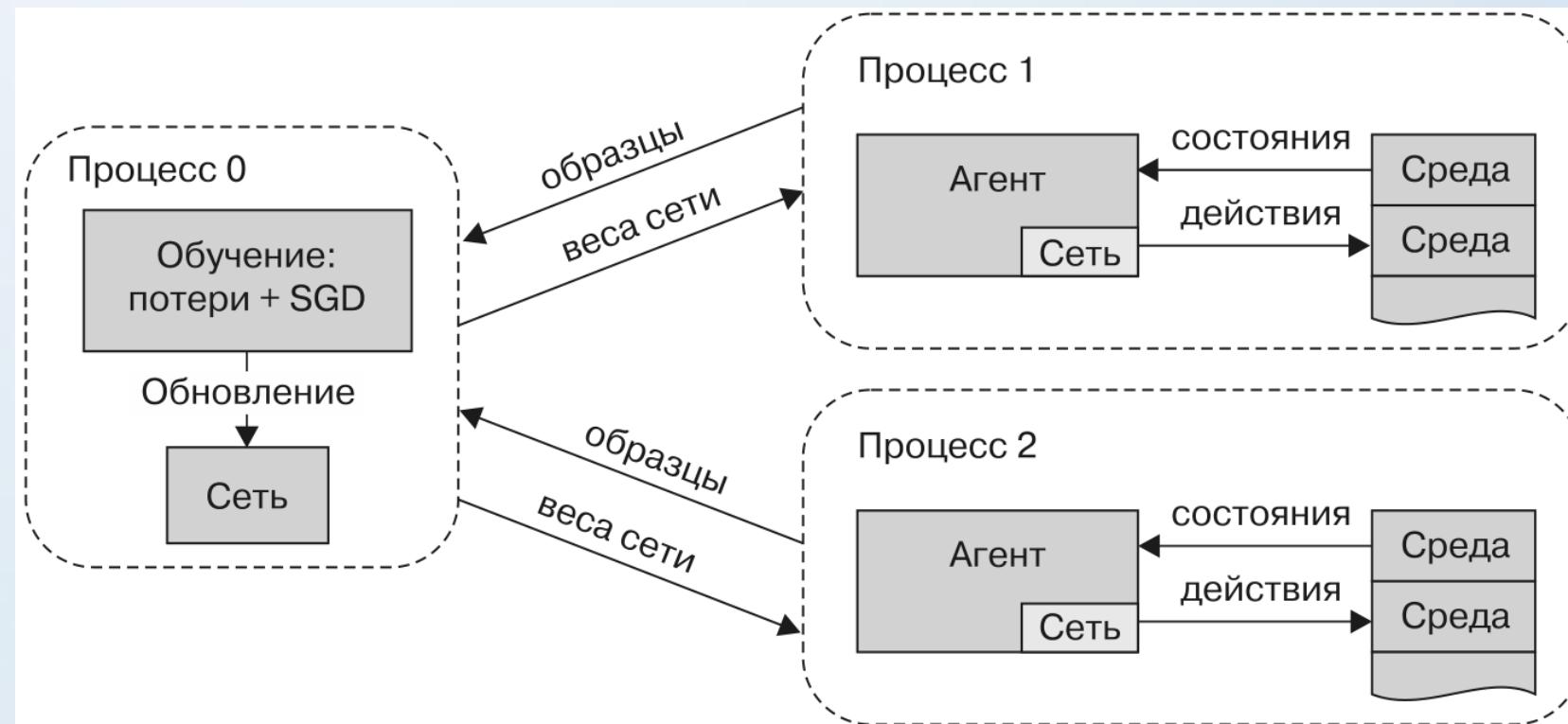
# Policy Gradient – 10 [1, 2, 8]

- Асинхронный Алгоритм Actor-Critic – A3C (**Asynchronous Advantage Actor-Critic**).
- Один из подходов к повышению стабильности семейства методов градиентов по стратегиям — параллельное использование нескольких сред. Причина этого в корреляции между примерами в обучающем наборе, нарушающей предположение о независимости и одинаковом распределении, что является критическим для оптимизации стохастического градиентного спуска (SGD).
- Негативным последствием такой корреляции становится очень высокая дисперсия градиентов, говорящая о том, что обучающий набор данных содержит очень похожие примеры и все они толкают сеть в одном направлении. Ввиду коррелированности примеров шаг градиентного спуска может быть совершенно неверным в глобальном смысле, поскольку все эти примеры могут быть взяты из одного удачного или неудачного эпизода.
- Для устранения корреляции используют распараллеливание с применением нескольких сред. Применительно к распараллеливанию для актора-критика существует два подхода.

# Policy Gradient – 11 [1, 2, 8]

- Асинхронный Алгоритм Actor-Critic – A3C.

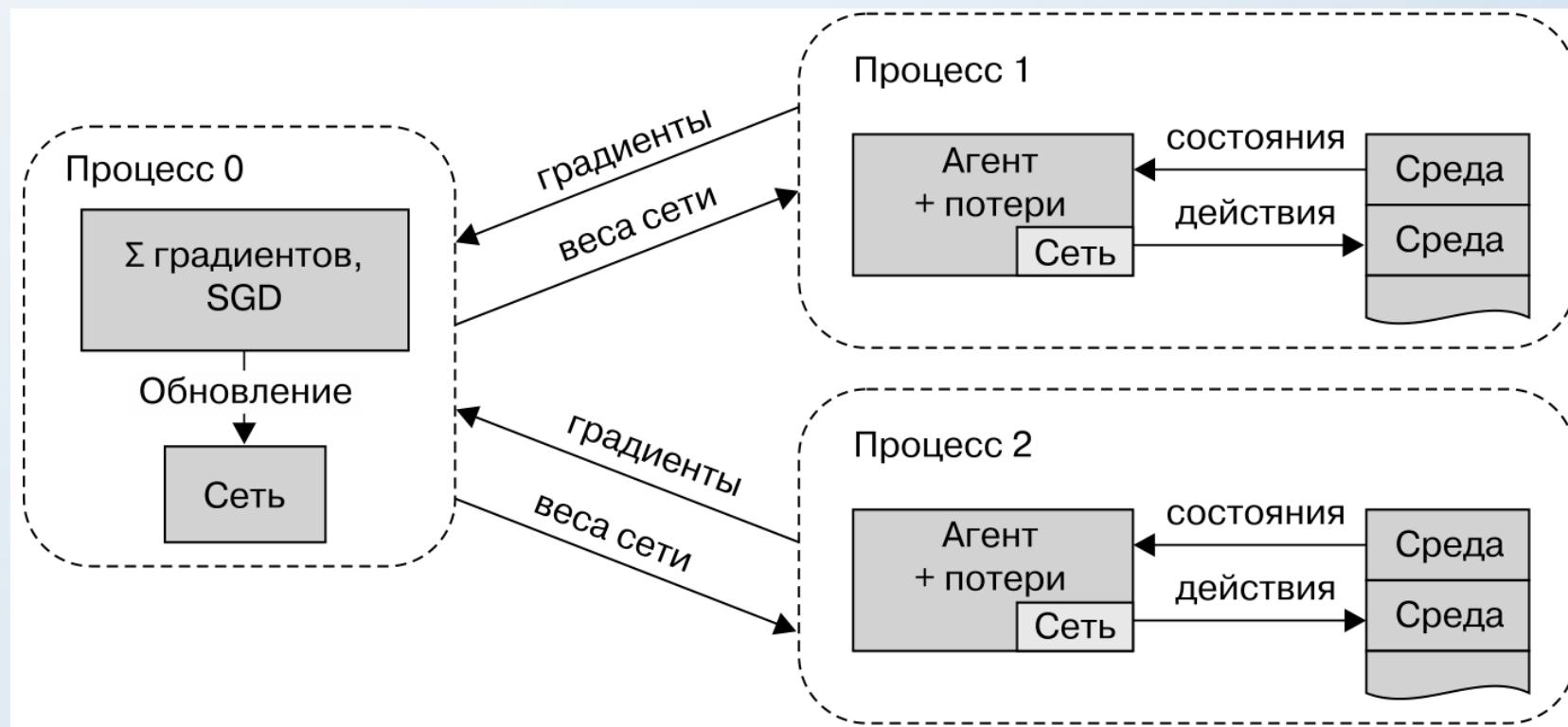
1. Параллелизм на уровне данных. У нас может быть несколько процессов, каждый из которых взаимодействует с одной или несколькими средами и предоставляет переходы ( $s, r, a, s'$ ). Все эти примеры собираются вместе в одном процессе обучения, который рассчитывает потери и выполняет обновление SGD. Затем обновленные параметры нейронной сети необходимо передать всем другим процессам для использования в дальнейших взаимодействиях.



# Policy Gradient – 12 [1, 2, 8]

- Асинхронный Алгоритм Actor-Critic – A3C.

2. Параллелизм на уровне градиентов. Поскольку цель обучения — вычисление градиентов для обновления сети, у нас может быть несколько процессов, вычисляющих градиенты на их собственных обучающих примерах. Затем эти градиенты могут суммироваться для обновления SGD в одном процессе. Конечно, обновленные веса сети также необходимо распространять среди всех рабочих процессов, чтобы данные собирались с помощью текущей стратегии.



# Policy Gradient – 13 [1, 2, 8]

- **Асинхронный Алгоритм Actor-Critic – A3C.**
- Самая тяжелая операция при оптимизации A3C — это процесс обучения, который состоит из расчета потерь по примерам данных (прямой проход) и расчета градиентов относительно этих потерь.
- Этап оптимизации SGD довольно легковесный — в основном это просто добавление масштабированных градиентов к весам сети. Перемещая вычисление потерь и градиентов из центрального процессса во втором подходе, можно устранить основное потенциальное узкое место и сделать процесс значительно более масштабируемым.
- На практике выбор метода зависит от ресурсов и целей. Если стоит одна задача оптимизации и имеется большое количество распределенных вычислительных ресурсов, например несколько десятков графических процессоров, расположенных на нескольких машинах в сетях, то параллелизм на уровне градиентов — лучший подход для ускорения обучения.
- Тем не менее на одном отдельно взятом графическом процессоре оба метода покажут одинаковую производительность, и первый подход, как правило, реализовать проще, так как не нужно работать с низкоуровневыми значениями градиентов.

# Policy Gradient – 14

Примеры реализации алгоритмов Actor-Critic:

- <https://mengxinji.github.io/Blog/2019-04-08/Actor-Critic/>
- <https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f>
- <https://www.kaggle.com/code/maulberto3/cartpole-v1-pytorch-rl-actor-critic>