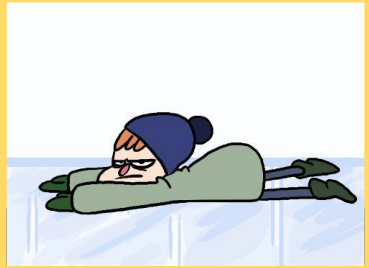
RL 2-02

DQN

Начнем в 20:01

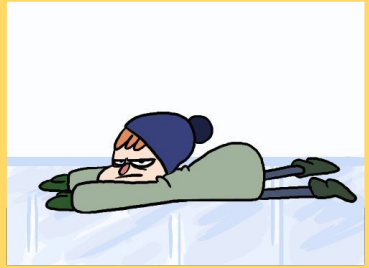
otus.ru

Как объяснить инопланетянину разницу между глаголами "ползти" и "скользить"?



Reward hacking

Как объяснить инопланетянину разницу между глаголами "ползти" и "скользить"?



Зачем?

The Surprising Creativity of Digital Evolution: A Collection of Anecdotes from the Evolutionary Computation and Artificial Life Research Communities

Joel Lehman[1†], Jeff Clune[1, 2†], Dusan Misevic[3†], Christoph Adami[4], Lee Altenberg[5], Julie Beaulieu[6], Peter J Bentley[7], Samuel Bernard[8], Guillaume Beslon[9], David M Bryson[4], Patryk Chrabaszcz[11], Nick Cheney[2], Antoine Cully[12], Stephane Doncieux[13], Fred C Dyer[4], Kai Olav Ellefsen[14], Robert Feldt[15], Stephan Fischer[16], Stephanie Forrest[17], Antoine Frénoy[18], Christian Gagné[6] Leni Le Goff[13], Laura M Grabowski[19], Babak Hodjat[20], Frank Hutter[11], Laurent Keller[21], Carole Knibbe[9], Peter Krcah[22], Richard E Lenski[4], Hod Lipson[23], Robert MacCurdy[24], Carlos Maestre[13], Risto Miikkulainen[26], Sara Mitri[21], David E Moriarty[27], Jean-Baptiste Mouret[28], Anh Nguyen[2], Charles Ofria[4], Marc Parizeau [6], David Parsons[9], Robert T Pennock[4], William F Punch[4], Thomas S Ray[29], Marc Schoenauer[30], Eric Schulte[17], Karl Sims, Kenneth O Stanley[1,31], François Taddei[3], Danesh Tarapore[32], Simon Thibault[6], Westley Weimer[33], Richard Watson[34], Jason Yosinski[1]

†Organizing lead authors

1 Uber AI Labs, San Francisco, CA, USA
2 University of Wyoming, Laramie, WY, USA
3 Center for Research and Interdisciplinarity, Paris, France
4 Michigan State University, East Lansing, MI, USA
5 Univeristy of Hawai'i at Manoa, HI, USA
6 Université Laval, Quebec City, Quebec, Canada
7 University College London, London, UK
8 INRIA, Institut Camille Jordan, CNRS, UMR5208, 69622 Villeurbanne, France
9 Université de Lyon, INRIA, CNRS, LIRIS UMR5205, INSA, UCBL, Lyon, France
10 French Institute of Petroleum, Rueil-Malmaison, France
11 University of Freiburg, Freiburg, Germany
12 Imperial College London, London, UK
13 Sorbonne Universités, UPMC Univ Paris 06, CNRS, Institute of Intelligent Systems and Robotics (ISIR), Paris, France
14 University of Oslo, Oslo, Norway
15 Chalmers University of Technology, Gothenburg, Sweden
16 INRA, Jouy-en-Josas, France
17 University of New Mexico Albuquerque, NM, USA
18 Institute of Integrative Biology, ETH Zürich, Switzerland
19 State University of New York at Potsdam, NY, USA
20 Sentient Technologies, San Francisco, CA, USA
21 Department of Fundamental Microbiology, University of Lausanne, 1015, Lausanne, Switzerland
22 Charles University Prague, Prague, Czech Republic
23 Columbia University, New York, NY, USA
24 University of Colorado, Boulder, CO, USA
25 Université de Pau, Pau, France
26 University of Texas at Austin, Austin, USA
27 Apple Inc.
28 Inria Nancy Grand - Est, Villers-lès-Nancy, France
29 University of Oklahoma, Norman, Oklahoma
30 Inria, Université Paris-Saclay, France

https://arxiv.org/abs/1803.03453

Evolving Virtual Creatures

Karl Sims

Thinking Machines Corporation
245 First Street, Cambridge, MA 02142

Abstract

This paper describes a novel system for creating virtual creatures that move and behave in simulated three-dimensional physical worlds. The morphologies of creatures and the neural systems for controlling their muscle forces are both generated automatically using genetic algorithms. Different fitness evaluation functions are used to direct simulated evolutions towards specific behaviors such as swimming, walking, jumping, and following.

A genetic language is presented that uses nodes and connections as its primitive elements to represent directed graphs, which are used to describe both the morphology and the neural circuitry of these creatures. This genetic language defines a hyperspace containing an indefinite number of possible creatures with behaviors, and when it is searched using optimization techniques, a variety of successful and interesting locomotion strategies emerge, some of which would be difficult to invent or build by design.

1 Introduction

A classic trade-off in the field of computer graphics and animation is that of complexity vs. control. It is often difficult to build interesting or realistic virtual entities and still maintain control over them. Sometimes it is difficult to build a complex virtual world at all, if it is necessary to conceive, design, and assemble each component. An example of this trade-off is that of kinematic control vs. dynamic simulation. If we directly provide the positions and angles for moving objects, we can control each detail of their behavior, but it might be difficult to achieve physically plausible motions. If we instead provide forces and torques and simulate the resulting dynamics, the result will probably look correct, but then it can be very difficult to achieve the desired behavior, especially as the objects we want to control become more complex. Methods have been developed for dynamically controlling specific objects to successfully crawl, walk, or even run [11,12,16], but a new control algorithm must be carefully designed each time a new behavior or morphology is desired.

Optimization techniques offer possibilities for the automatic generation of complexity. The genetic algorithm is a form of artificial evolution, and is a commonly used method for optimization. A Darwinian "survival of the fittest" approach is employed to search for optima in large multidimensional spaces [5,7]. Genetic algorithms permit virtual entities to be created without requiring an understanding of the procedures or parameters used to generate them. The measure of success, or *fitness*, of each individual can be calculated automatically, or it can instead be provided interactively by a user. Interactive evolution allows procedurally generated results to be explored by simply choosing those that are the most aesthetically desirable for each generation [2,18,19,21].

The user sacrifices some control when using these methods, especially when the fitness is procedurally defined. However, the potential gain in automating the creation of complexity can often compensate for this loss of control, and a higher level of user influence is still maintained by the fitness criteria specification.

In several cases, optimization has been used to automatically generate dynamic control systems for given articulated structures: de Garis has evolved weight values for neural networks [4], Ngo and Marks have performed genetic algorithms on stimulus-response pairs [14], and van de Panne and Fiume have optimized sensor-actuator networks [15]. Each of these methods has resulted in successful locomotion of two-dimensional stick figures.

The work presented here is related to these projects, but differs in several respects. In previous work, control systems were generated for fixed structures that were user-designed, but here entire creatures are evolved: the optimization determines the creature morphologies as well as their control systems. Also, here the creatures' bodies are three-dimensional and fully physically based. The three-dimensional physical structure of a creature can adapt to its control system, and vice versa, as they evolve together. The "nervous systems" of creatures are also completely determined by the optimization: the number of internal nodes, the connectivity, and the type of function each neural node performs are included in the genetic description of each creature, and can grow in complexity as an evolution proceeds. Together, these remove the necessity for a user to provide any specific creature information such as shape, size, joint constraints, sensors, actuators, or internal neural parameters. Finally, here a developmental process is used to generate the creatures and their control systems, and allows similar components including their local neural circuitry to be defined once and then replicated, instead of requiring each to be separately specified. This approach is related to L-systems, graftal grammars, and object instancing techniques [6,8,10,13,20].

It is convenient to use the biological terms *genotype* and *phenotype* when discussing artificial evolution. A *genotype* is a coded representation of a possible individual or problem solution. In biological systems, a genotype is usually composed of DNA and contains the instructions for the development of an organism. Genetic algorithms typically use populations of genotypes consisting of strings of binary digits or parameters. These are read to produce *phenotypes* which are then evaluated according to some fitness criteria and selectively reproduced. New genotypes are generated by copying, mutating, and/or combining the genotypes of the most fit individuals, and as the cycle repeats the population should ascend to higher and higher levels of fitness.

Variable length genotypes such as hierarchical Lisp expressions

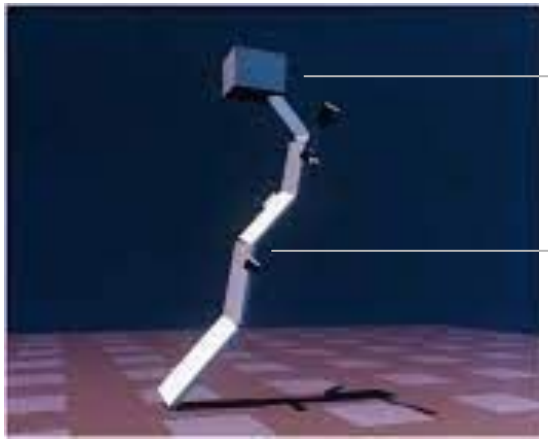
SIGGRAPH '94, July 24-29, Orlando, Florida

https://www.karlsims.com/papers/siggraph94.pdf

Экзотический пример №1

Как?

- Создали виртуальный мир с гравитацией и трением
- Расположили в нем виртуальное существо:



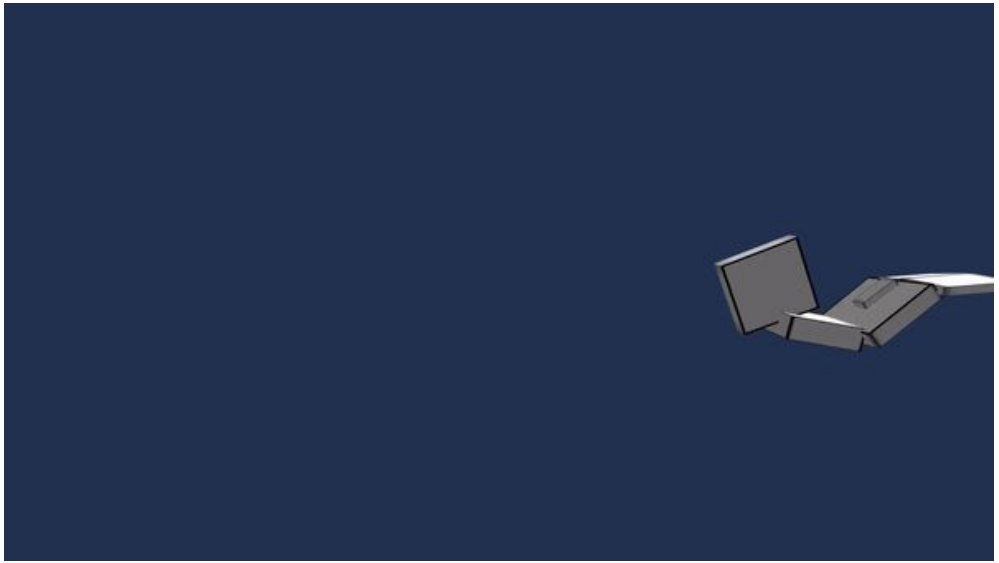
Кубическая голова с мотором и запасом энергии

Шарниры, соединяющие блоки, которые могут блокироваться, а могут оставаться подвижными

Экзотический пример №1

Хотели, чтобы виртуальное существо само изобрело ползание.

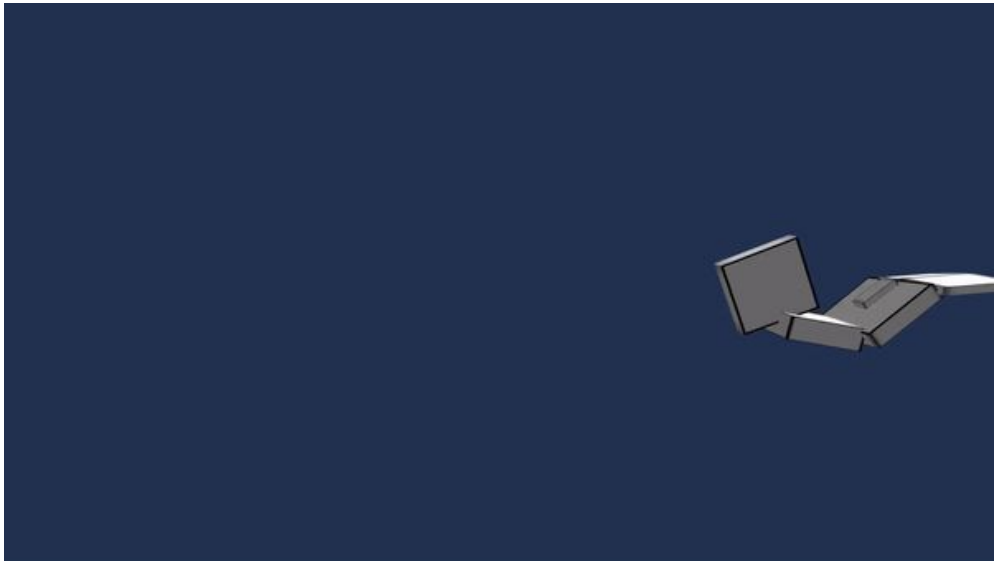
- Поэтому каждый раз ставили его на старт.
- Запускали таймер.
- Чем дальше за 5 секунд хвост оказывался от старта, тем большее “поощрение” получало существо.



Экзотический пример №1

Однако, первые попытки провалились. Потому что существо нашло способ выполнять задачу гораздо проще

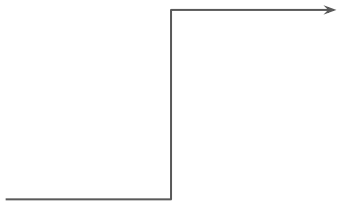
- Поэтому каждый раз ставили его на старт.
- Запускали таймер.
- Чем дальше за 5 секунд хвост оказывался от старта, тем большее “поощрение” получало существо.



Экзотический пример №1

Однако, первые попытки провалились. Потому что существо нашло способ выполнять задачу гораздо проще

- Поэтому каждый раз ставили его на старт.
- Запускали таймер.
- Чем дальше за 5 секунд хвост оказывался от старта, тем большее “поощрение” получало существо.



Виной тому как раз нечеткая постановка задачи, определение.

Попытайтесь догадаться, как существо оказывалось максимально далеко, не ползая?

Экзотический пример №1

Оно поднималось вверх,
отталкиваясь от пола и
падало, совершая
кувырок.

Тем самым хвост
оказывался очень
далеко.

Экзотический пример №1

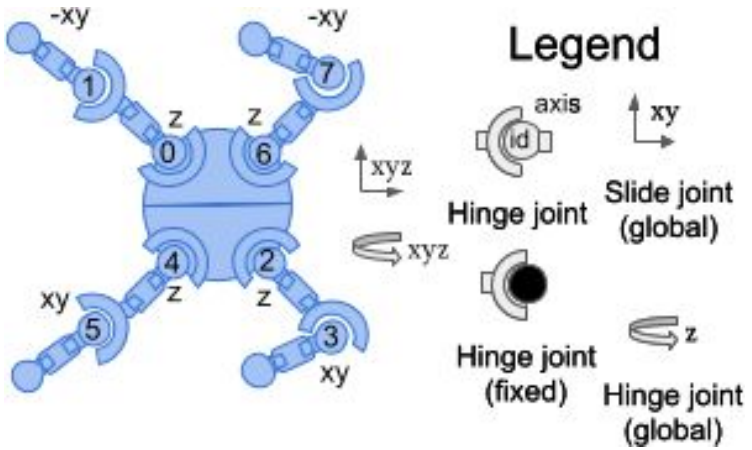
Оно поднималось вверх, отталкиваясь от пола и падало, совершая кувырок.

Тем самым хвост оказывался очень далеко.

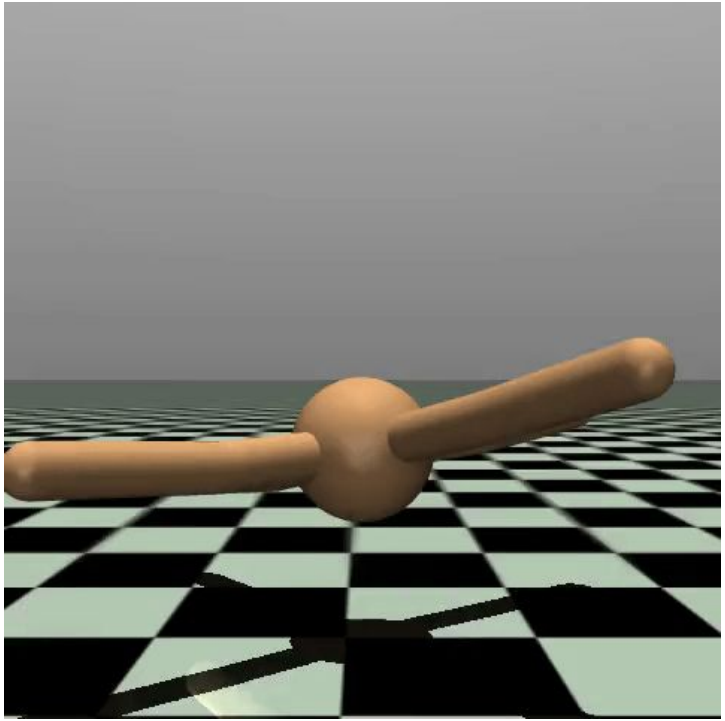
Экзотический пример №1

Вся проблема в том, что исследователи не поставили целью экономить энергию и снизить травмоопасность движения

Научить муравья ползать



Кто из них скользит?




сидит стоит лежит

Q-learning

Проблемы

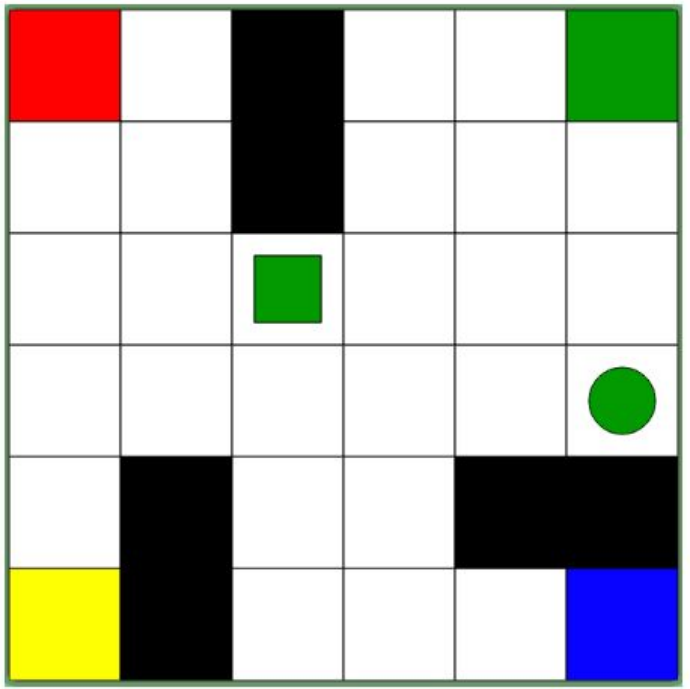
1. Не знаем переходов
2. Много состояний

```
delta = 0.0
new_values = defaultdict(lambda: 0.0)
for state in self.mdp.get_states():
    for action in self.mdp.get_actions(state):
        new_value = 0.0
        for (new_state, probability) in self.mdp.get_transitions(state, action):
            reward = self.mdp.get_reward(state, action, new_state)
            new_value += probability * (reward + (self.mdp.get_discount_factor() * self.V[new_state]))
        self.Q[(state, action)] = new_value

    # V(s) = max_a{Q(s, a)}
    max_q = float('-inf')
    arg_max_q = None
    for action in self.mdp.get_actions(state):
        if max_q < self.Q[(state, action)]:
            max_q = self.Q[(state, action)]
            arg_max_q = action
```

Проблемы

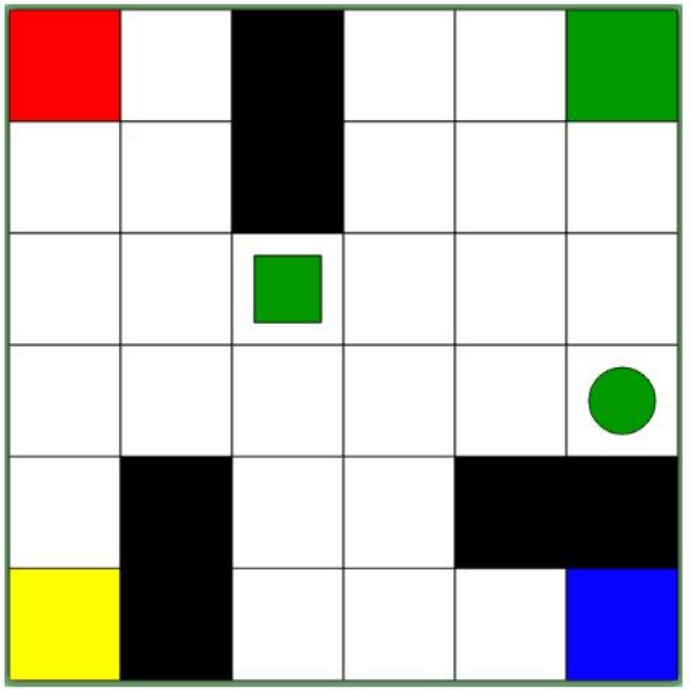
1. Не знаем переходов
2. Много состояний



Model-free

1. **Не знаем переходов**
2. Много состояний

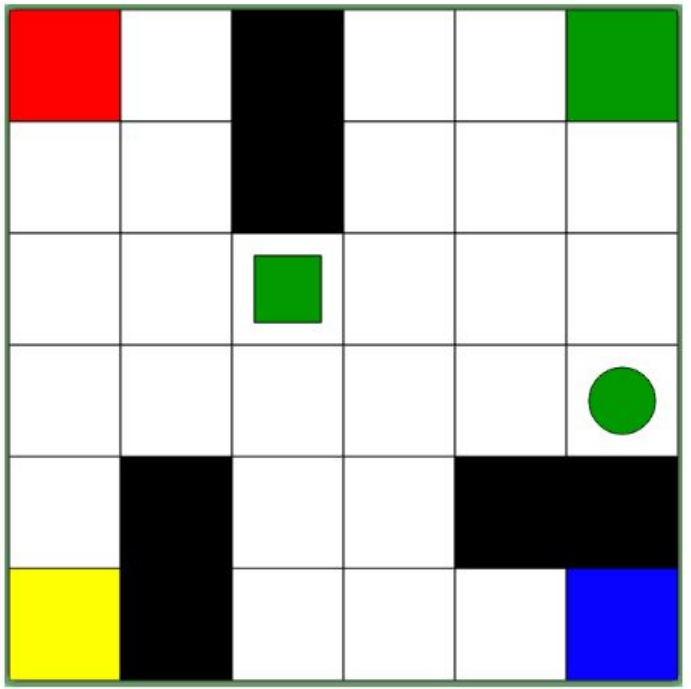
https://github.com/guiferviz/rl_udacity/tree/master/mystery_game



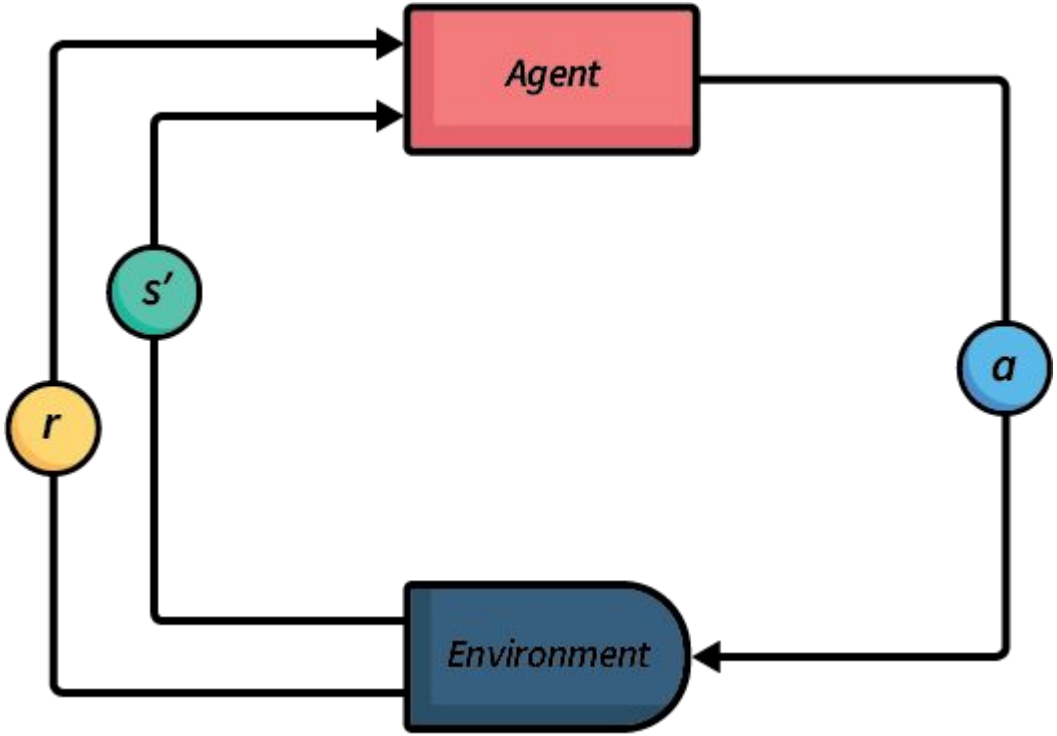
Q-learning

1. **Не знаем переходов**
2. Много состояний

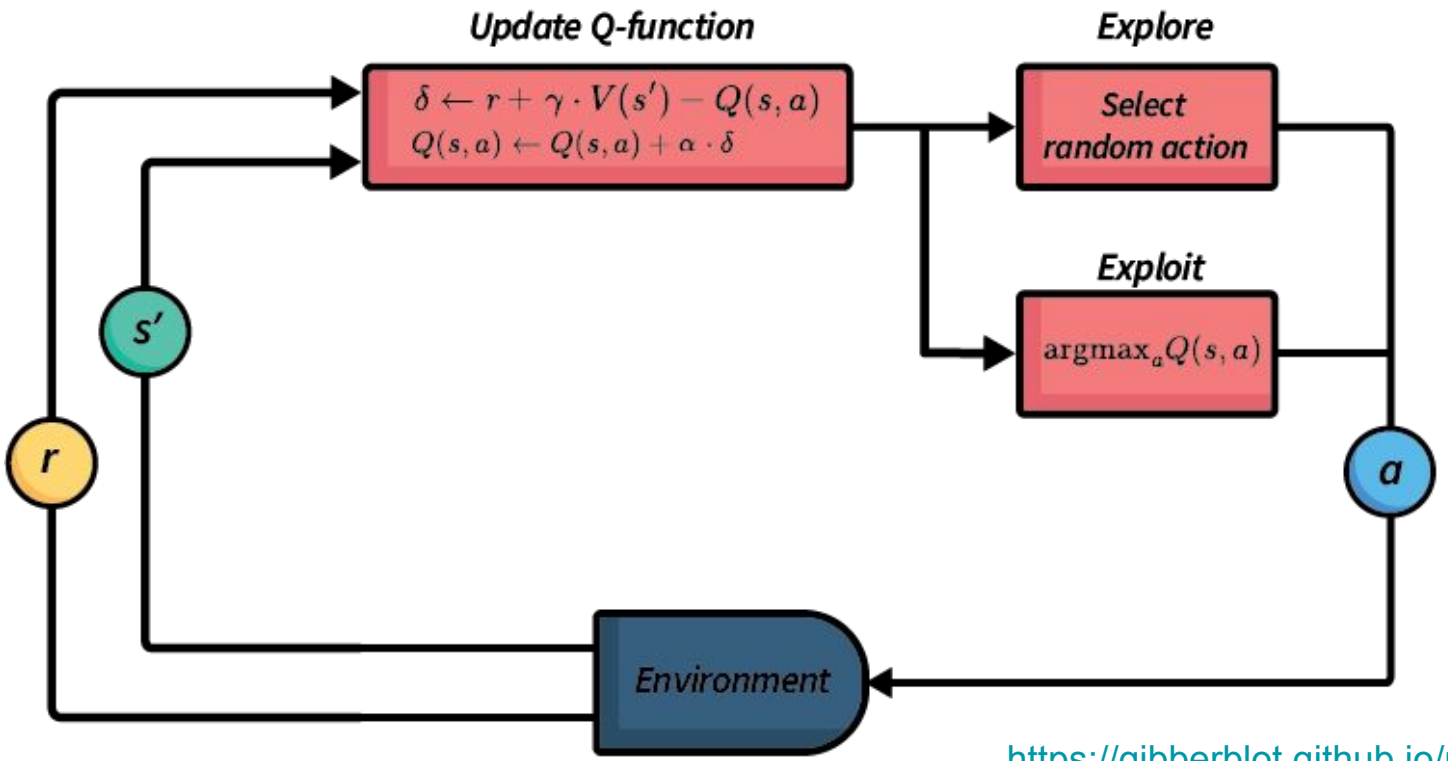
https://github.com/guiferviz/rl_udacity/tree/master/mystery_game



Q-learning

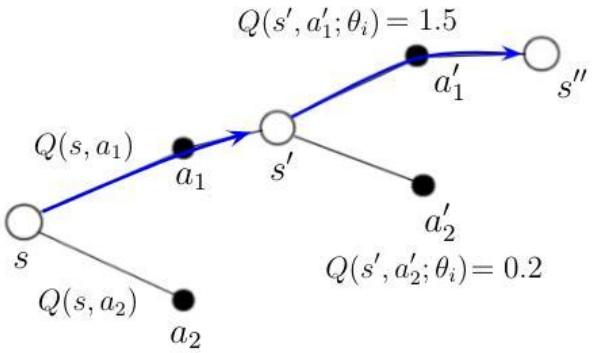


Q-learning



https://gibberblot.github.io/rl-notes/single-agent/temporal-difference-learning.html

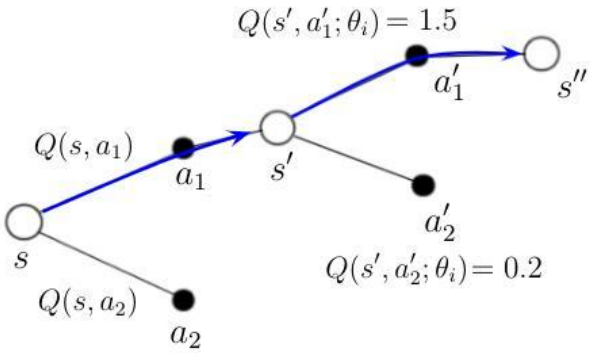
Q-learning



- Ищем не саму политику, а Q-функцию.
- Выручает, когда не знаем вероятности переходов

$$Q(s,a) \leftarrow r + \gamma \max_{a'} Q(s',a')$$

Q-learning

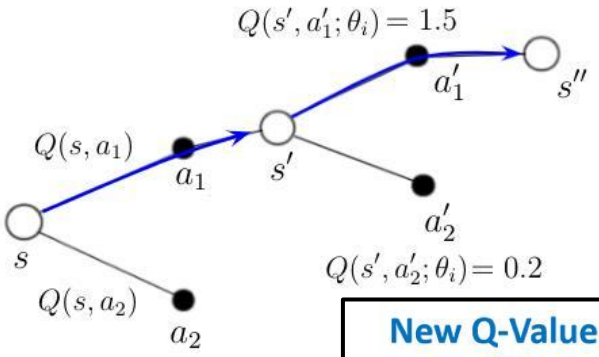


- Ищем не саму политику, а Q-функцию.
- Выручает, когда не знаем вероятности переходов

$$Q(s,a) \leftarrow r + \gamma \max_{a'} Q(s',a')$$

$V(s')$

DQN



- **Идея:** Будем более стабильны и сойдемся быстрее

New Q-Value Current Q-Value TD Error

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

Learning Rate
(0, 1]

Discount Factor [0,1]
γ = 0: Only Immediate Reward
γ = 1: Only Future Reward

The maximum estimated future reward for the next state s' over all possible actions a'.

Экспоненциальное сглаживание

Экспоненциальным сглаживанием (exponential smoothing) для последовательности $x_1, x_2, x_3 \dots$ будем называть следующую оценку:

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

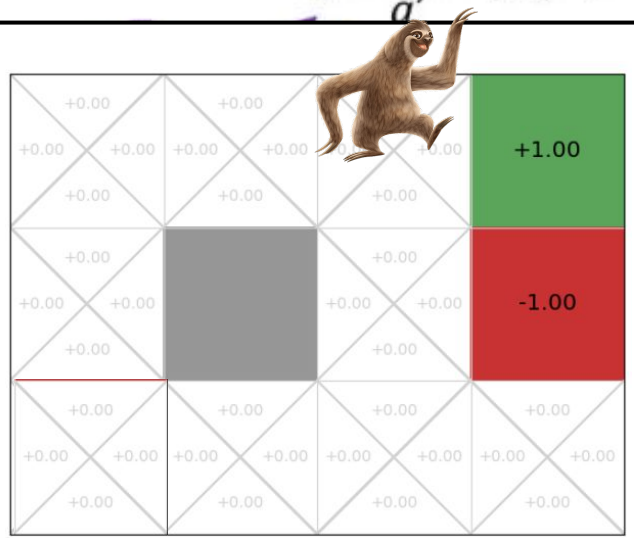
где m_0 — некоторое начальное приближение, последовательность $\alpha_k \in [0, 1]$ — гиперпараметр, называемый ***learning rate***.

https://www.geogebra.org/m/RgYaYxgQ

Пример

$$Q(s,a) \leftarrow r + \gamma \max_{a'} Q(s',a')$$

State	Up	Down	Right	Left
(0, 0)	0.50	0.42	0.39	0.42
(0, 1)	0.56	0.44	0.51	0.51
(0, 2)	0.58	0.51	0.63	0.57
(1, 0)	0.09	0.18	0.06	0.43
(1, 1)	0.00	0.00	0.00	0.00
(1, 2)	0.64	0.65	0.74	0.59
(2, 0)	0.41	0.00	0.00	0.00
(2, 1)	0.69	0.09	-0.24	0.24
(2, 2)	0.79	0.61	0.90	0.65
(3, 0)	-0.02	0.00	0.00	0.00
(3, 1)	0.00	0.00	0.00	0.00
(3, 2)	0.00	0.00	0.00	0.00

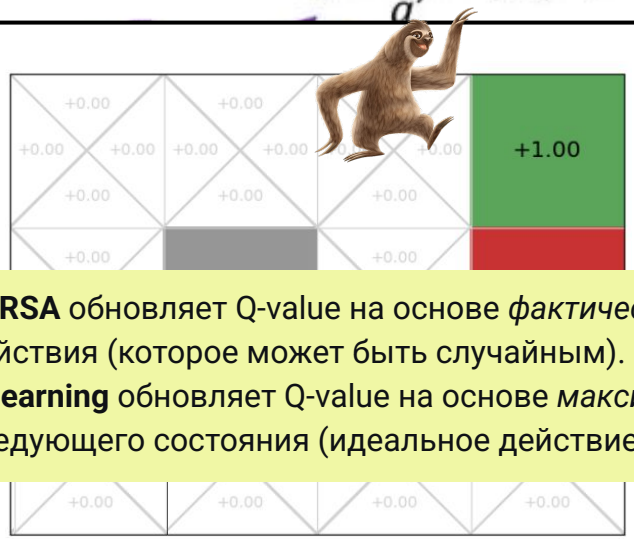


$$\begin{aligned}
Q((2,2),Up) &\leftarrow Q((2,2),Up) + \alpha[r + \gamma \max_{a'} Q((2,2),a') - Q((2,2),Up)] \\
&\leftarrow 0.79 + 0.1[0 + 0.9 \cdot Q((2,2),Right) - Q((2,2),Up)] \\
&\leftarrow 0.79 + 0.1[0 + 0.9 \cdot 0.90 - 0.79] \\
&\leftarrow 0.792
\end{aligned}$$

SARSA

$$Q(s,a) \leftarrow r + \gamma \max_{a'} Q(s',a')$$

State	Up	Down	Right	Left
(0, 0)	0.50	0.42	0.39	0.42
(0, 1)	0.56	0.44	0.51	0.51
(0, 2)	0.58	0.51	0.63	0.57
(1, 0)	0.09	0.18	0.06	0.43
(1, 1)	0.00	0.00	0.00	0.00
(1, 2)	0.64	0.65	0.74	0.59
(2, 0)	0.41	0.00	0.00	0.00
(2, 1)	0.69	0.09	-0.24	0.24
(2, 2)	0.79	0.61	0.90	0.65
(3, 0)	-0.02	0.00	0.00	0.00
(3, 1)	0.00	0.00	0.00	0.00
(3, 2)	0.00	0.00	0.00	0.00



- **SARSA** обновляет Q-value на основе *фактического* следующего действия (которое может быть случайным).
- **Q-learning** обновляет Q-value на основе *максимального* Q-value следующего состояния (идеальное действие).

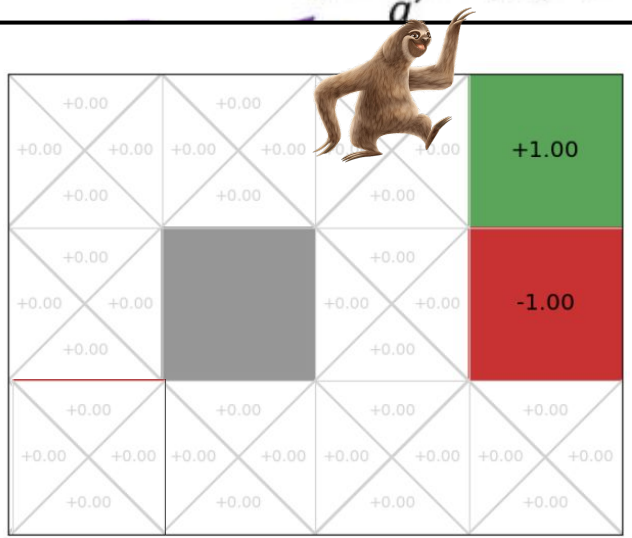
$$\begin{aligned} Q((2,2),Up) &\leftarrow Q((2,2),Up) + \alpha[r + \gamma \max_{a'} Q((2,2),a') - Q((2,2),Up)] \\ &\leftarrow 0.79 + 0.1[0 + 0.9 \cdot Q((2,2),Right) - Q((2,2),Up)] \\ &\leftarrow 0.79 + 0.1[0 + 0.9 \cdot 0.90 - 0.79] \\ &\leftarrow 0.792 \end{aligned}$$

0.72

SARSA

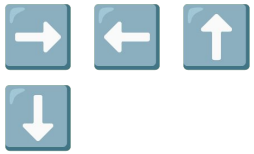
$$Q(s,a) \leftarrow r + \gamma \max_{a'} Q(s',a')$$

State	Up	Down	Right	Left
(0, 0)	0.50	0.42	0.39	0.42
(0, 1)	0.56	0.44	0.51	0.51
(0, 2)	0.58	0.51	0.63	0.57
(1, 0)	0.09	0.18	0.06	0.43
(1, 1)	0.00	0.00	0.00	0.00
(1, 2)	0.64	0.65	0.74	0.59
(2, 0)	0.41	0.00	0.00	0.00
(2, 1)	0.69	0.09	-0.24	0.24
(2, 2)	0.79	0.61	0.90	0.65
(3, 0)	-0.02	0.00	0.00	0.00
(3, 1)	0.00	0.00	0.00	0.00
(3, 2)	0.00	0.00	0.00	0.00

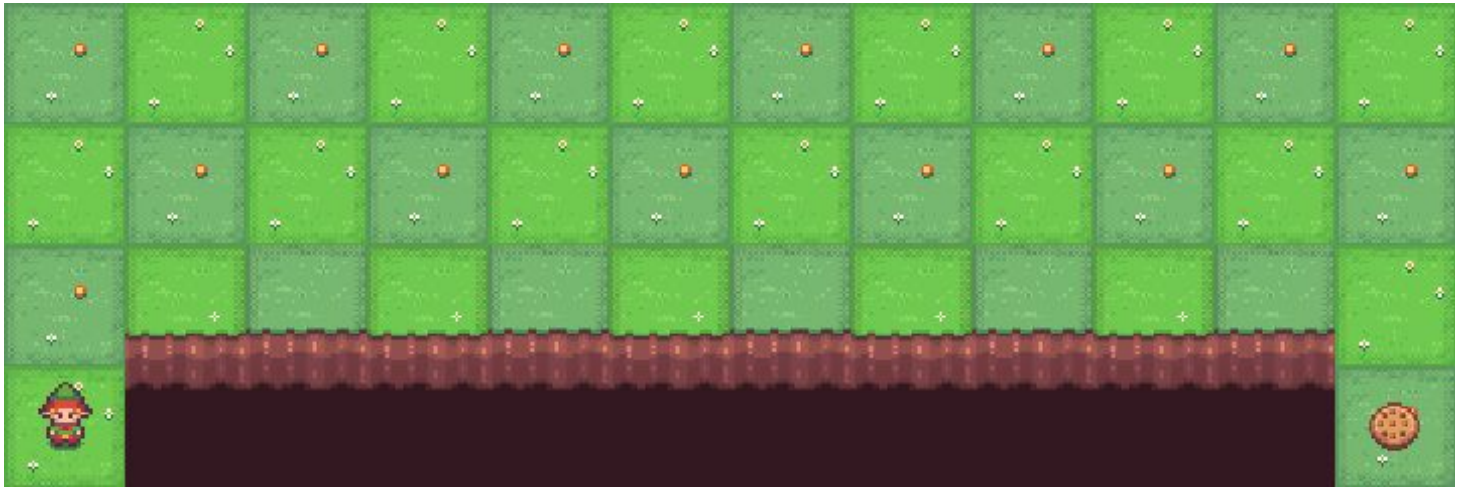


$$\begin{aligned} Q((2,2),Up) &\leftarrow Q((2,2),Up) + \alpha[r + \gamma \max_{a'} Q((2,2),a') - Q((2,2),Up)] \\ &\leftarrow 0.79 + 0.1[0 + 0.9 \cdot Q((2,2),Right) - Q((2,2),Up)] \\ &\leftarrow 0.79 + 0.1[0 + 0.9 \cdot 0.90 - 0.79] \\ &\leftarrow 0.792 \quad 0.7758 \quad 0.72 \end{aligned}$$

Cliff World

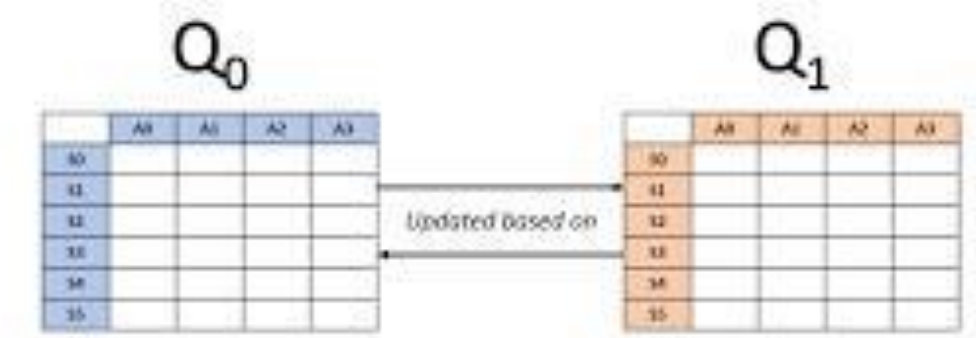


- **Агент SARSA** (on-policy), который иногда ошибается (exploration), "поймёт", что идти рядом с обрывом опасно, потому что в одном из экспериментов он мог бы с него упасть. Он выучит более безопасный, хоть и чуть более длинный, путь.
- **Агент Q-learning** (off-policy), который всегда смотрит на "идеальное" продолжение, выучит, что ближайший путь — лучший, потому что в его уравнениях он никогда не падает с обрыва (`max` всегда выбирает действие, которое уводит от края).



Проблемы

- Если состояний ооочень много, то будет тяжело



$$max_a = argmax(Q_1(s'))$$

$$Q_1(s,a) = (1-\alpha)Q_1(s,a) + \alpha[r + \gamma Q_0(s', max_a)]$$

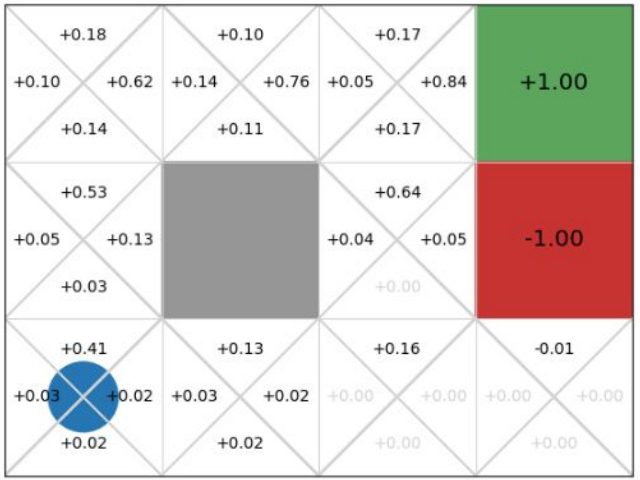
New Q value

Old Q value

Maximum Q value given the next state

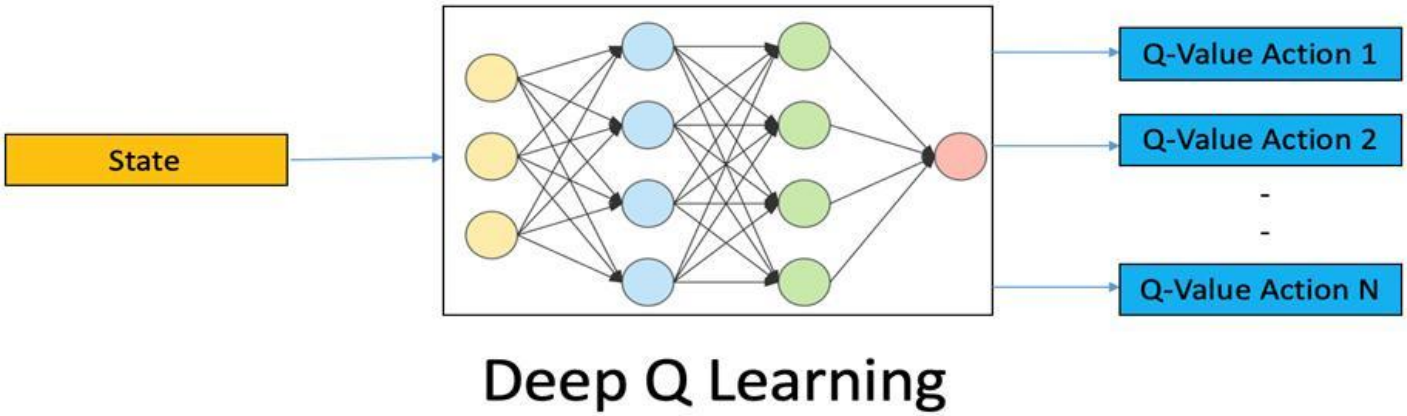

DQN

$$Q(s,a) \leftarrow \underbrace{Q(s,a)}_{\text{old value}} + \overbrace{\alpha}^{\text{learning rate}} \cdot [\underbrace{\overbrace{r}^{\text{reward}} + \overbrace{\gamma}^{\text{discount factor}} \cdot V(s')}_{\text{TD target}} - \overbrace{Q(s,a)}^{\text{do not count extra } Q(s,a)}]$$



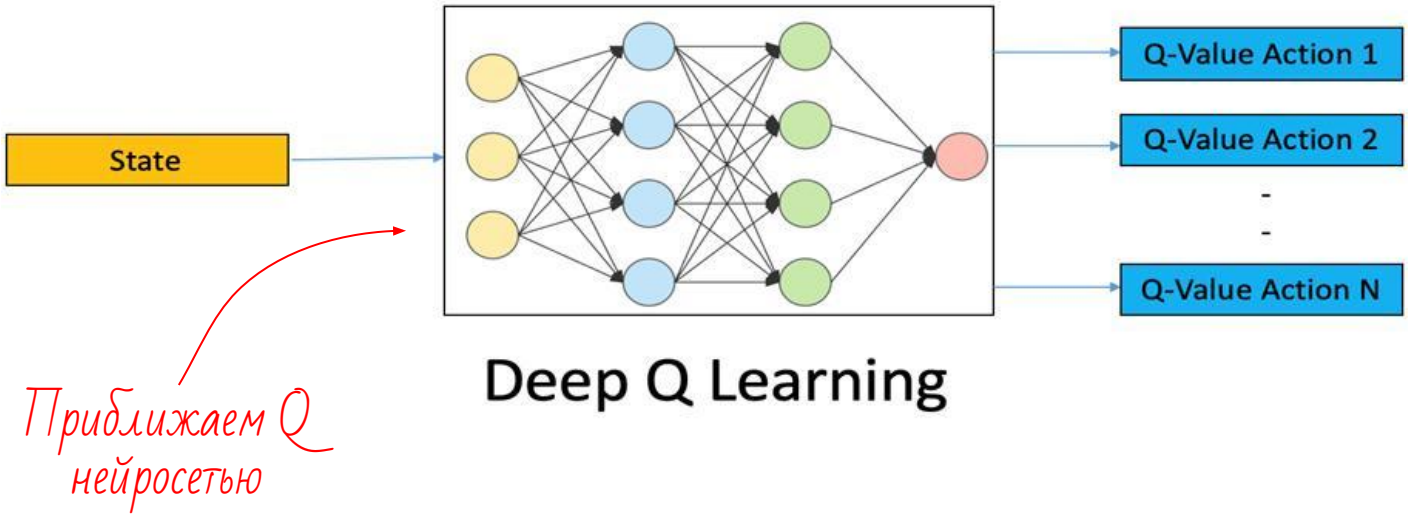
DQN

- **Сразу:** Количество состояний должно быть все равно конечно
- Архитектура:



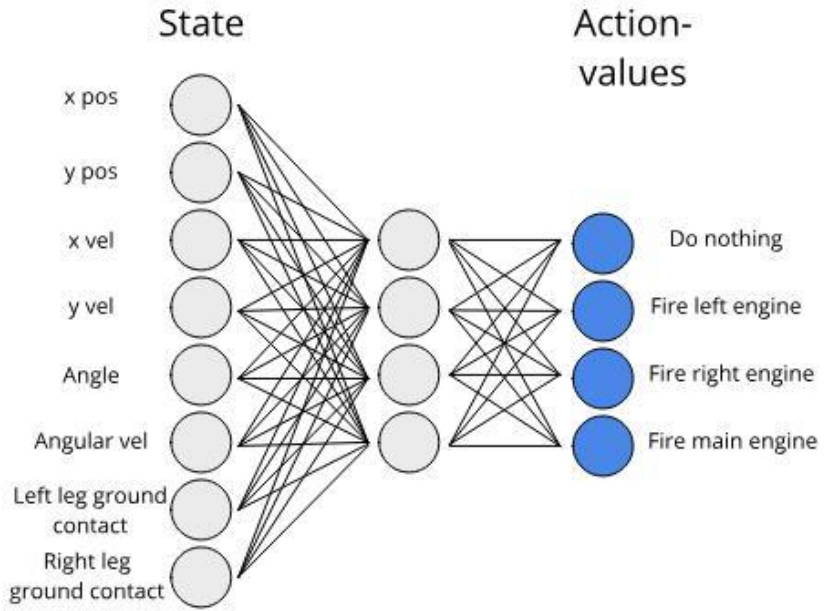
DQN

- **Сразу:** Количество состояний должно быть все равно конечно
- Архитектура:

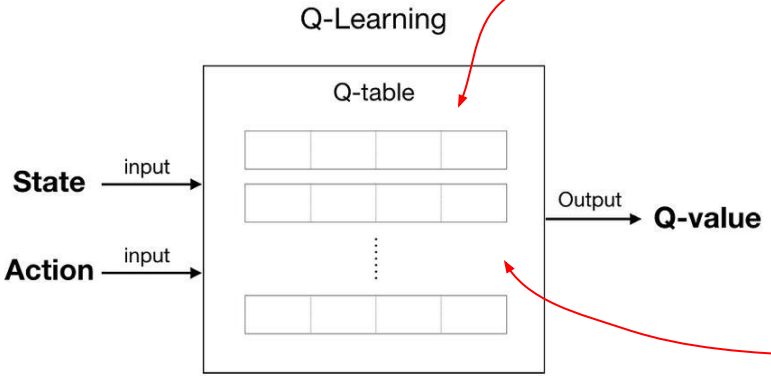


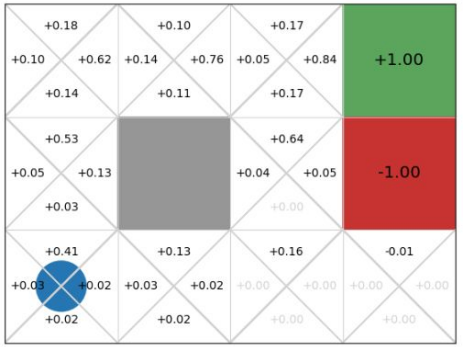
DQN

- **Сразу:** Количество состояний должно быть все равно конечно
- Архитектура:



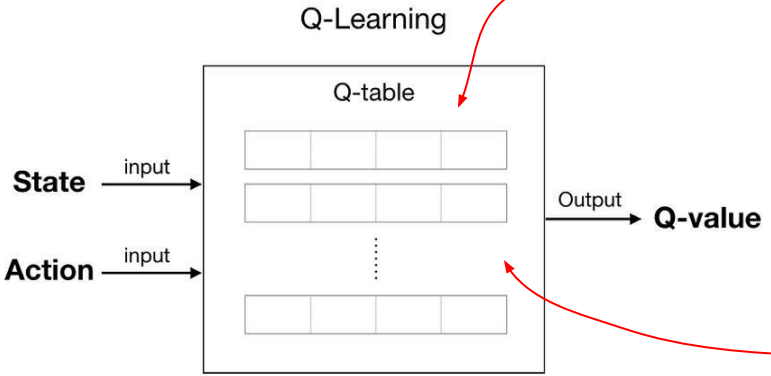
Отличия от Q-learning

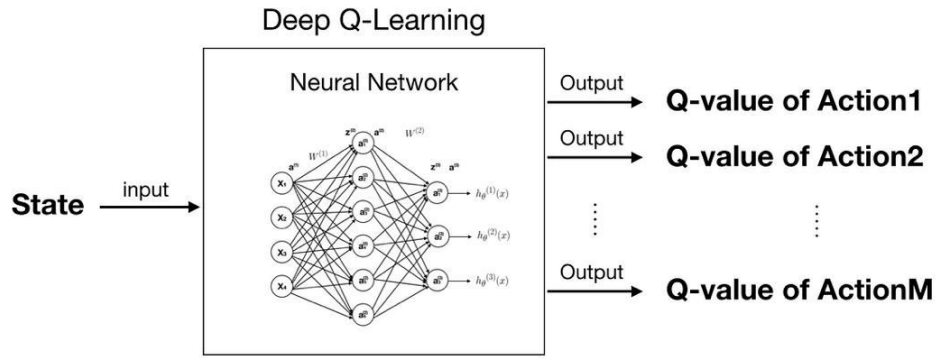


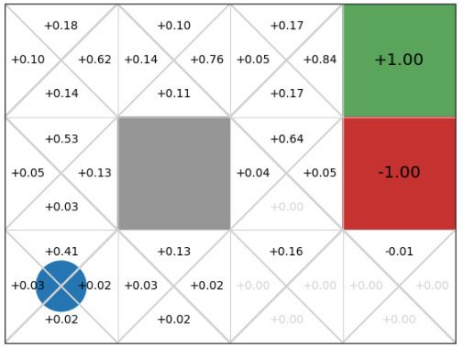


State	Up	Down	Right	Left
(0, 0)	0.50	0.42	0.39	0.42
(0, 1)	0.56	0.44	0.51	0.51
(0, 2)	0.58	0.51	0.63	0.57
(1, 0)	0.09	0.18	0.06	0.43
(1, 1)	0.00	0.00	0.00	0.00
(1, 2)	0.64	0.65	0.74	0.59
(2, 0)	0.41	0.00	0.00	0.00
(2, 1)	0.69	0.09	-0.24	0.24
(2, 2)	0.79	0.61	0.90	0.65
(3, 0)	-0.02	0.00	0.00	0.00
(3, 1)	0.00	0.00	0.00	0.00
(3, 2)	0.00	0.00	0.00	0.00

Отличия от Q-learning

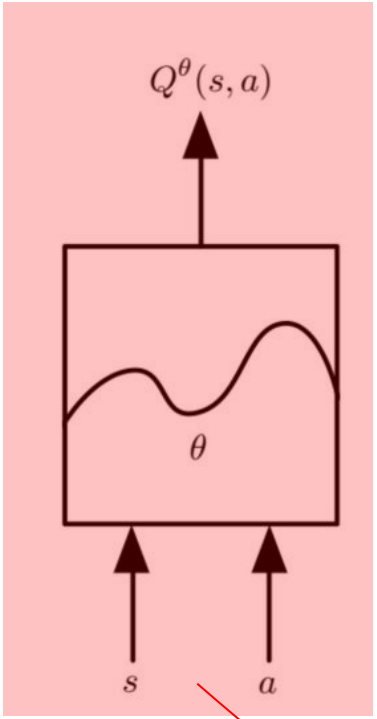


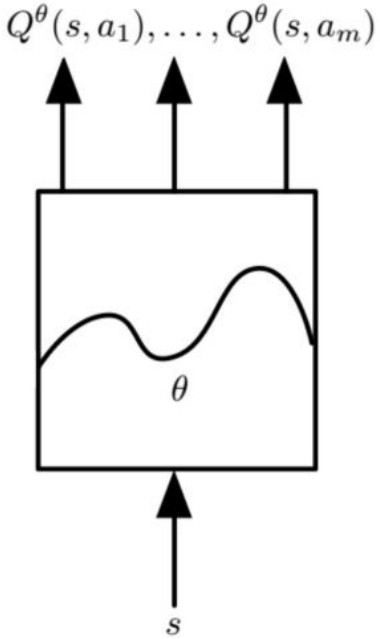




State	Up	Down	Right	Left
(0, 0)	0.50	0.42	0.39	0.42
(0, 1)	0.56	0.44	0.51	0.51
(0, 2)	0.58	0.51	0.63	0.57
(1, 0)	0.09	0.18	0.06	0.43
(1, 1)	0.00	0.00	0.00	0.00
(1, 2)	0.64	0.65	0.74	0.59
(2, 0)	0.41	0.00	0.00	0.00
(2, 1)	0.69	0.09	-0.24	0.24
(2, 2)	0.79	0.61	0.90	0.65
(3, 0)	-0.02	0.00	0.00	0.00
(3, 1)	0.00	0.00	0.00	0.00
(3, 2)	0.00	0.00	0.00	0.00

Можно и так:



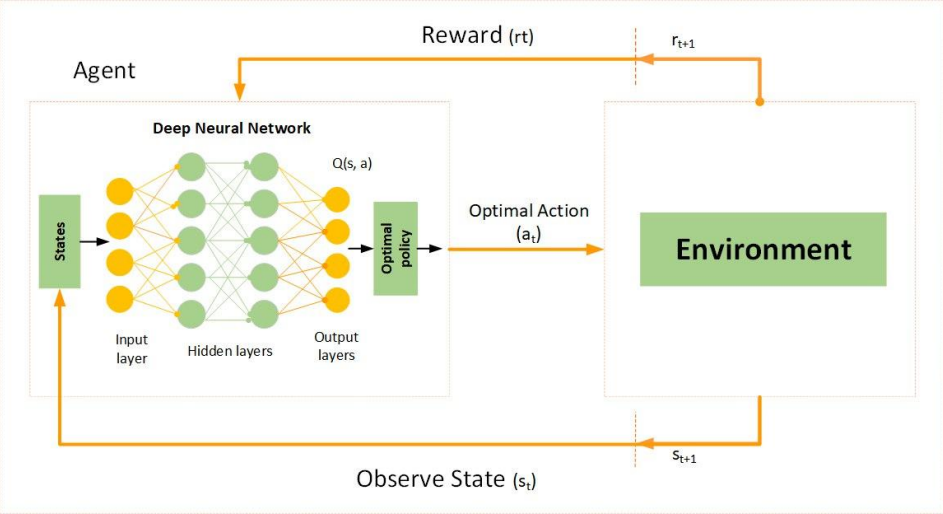


Почему так обычно не делают?

Как обучаем?

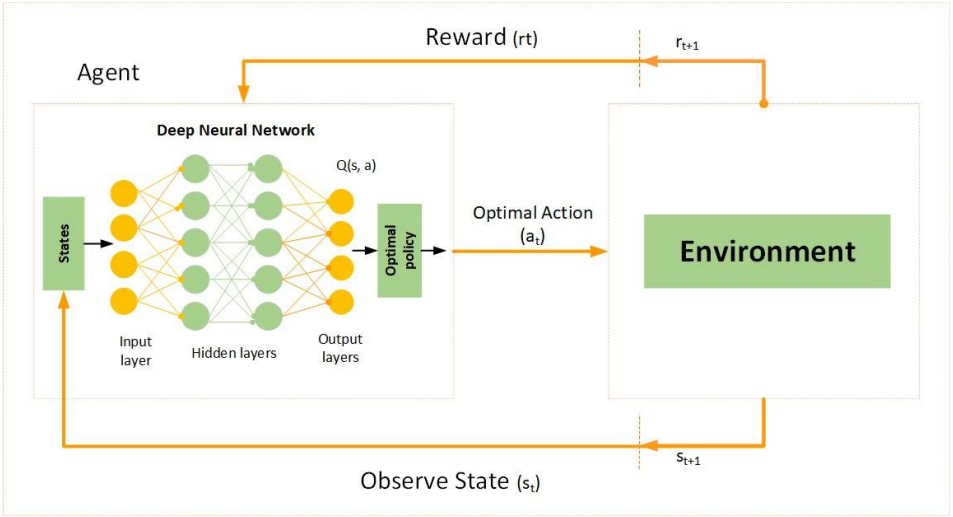
- **Loss?**

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



Как обучаем?

- **Loss?**



$$Q_{best}(s_t, a_t) = R_{t+1} + \gamma \max_a Q(s_{t+1}, a)$$

Estimated TD-target

Discount factor

Reward

Maximum next-state Q-value

Как обучаем?

- **Loss?**

TD-target is UNKNOWN!

Current Q-value

$$\text{Loss} = Q_{best}(s_t, a_t) - Q(s_t, a_t)$$

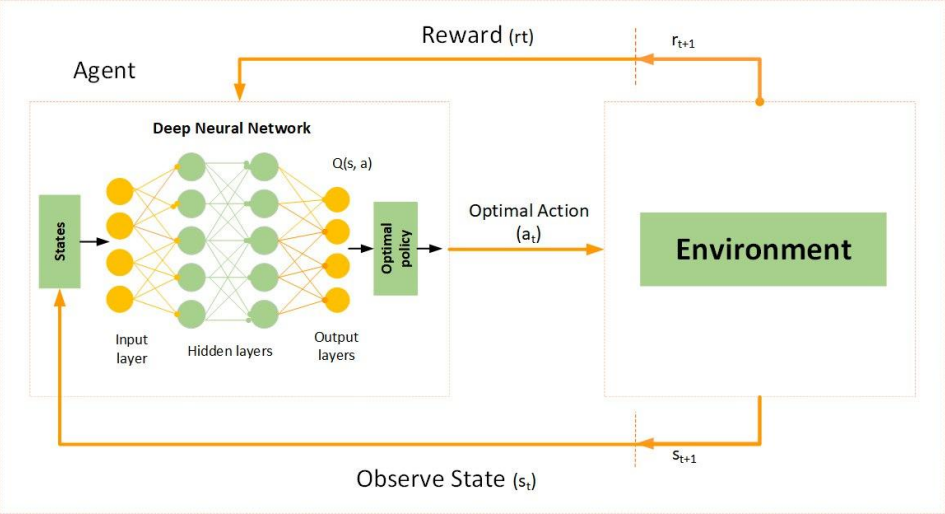
Estimated TD-target

Discount factor

$$Q_{best}(s_t, a_t) = R_{t+1} + \gamma \max_a Q(s_{t+1}, a)$$

Reward

Maximum next-state Q-value



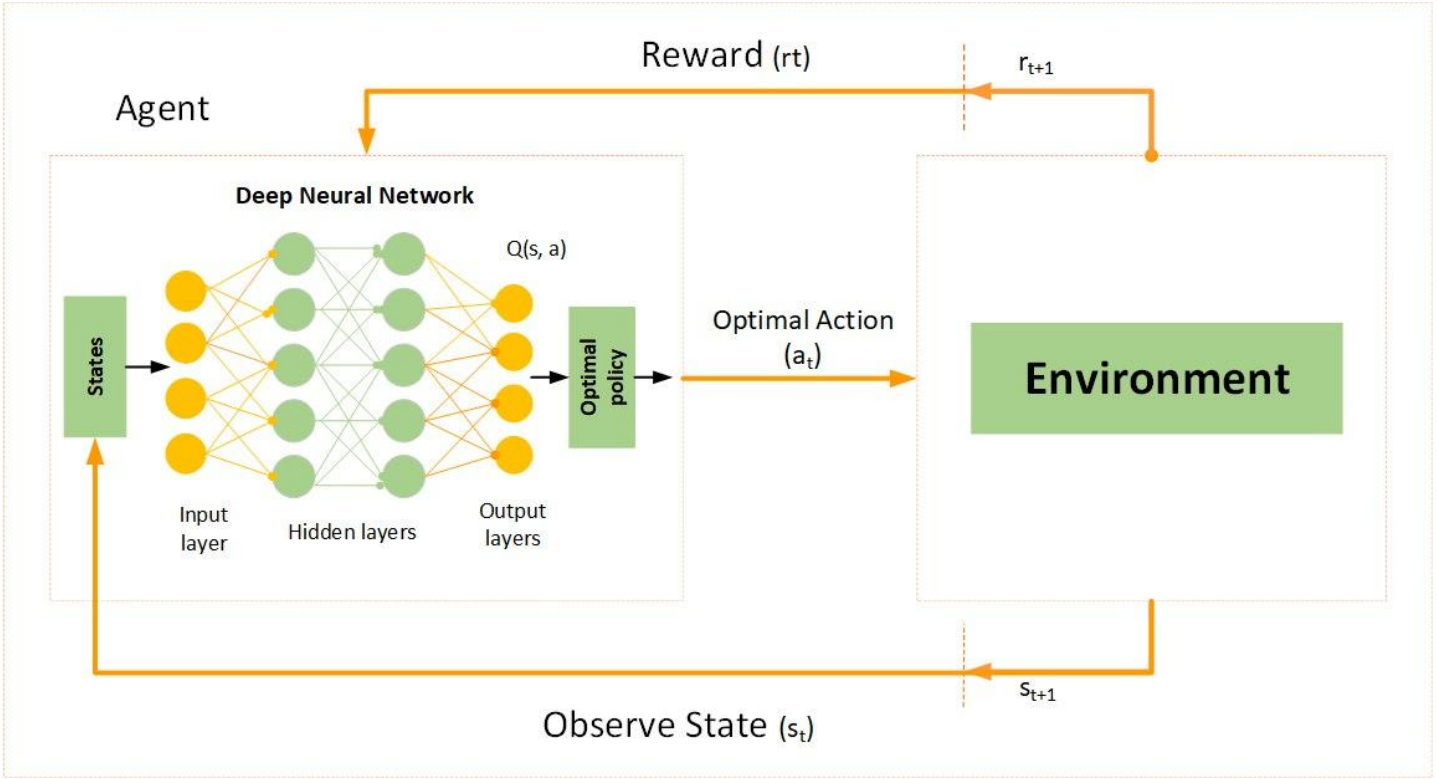
Как обучаем?

- **Loss?**

- Инициализируем нейросеть Q^{θ}, ε=1.
- Получаем данные.

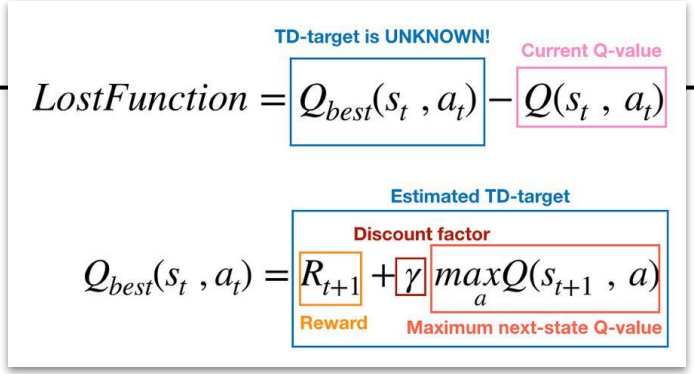
  Из состояния S_t действуем $A_t \sim \pi(\cdot|S_t)$, где π - ε-жадная политика, получаем награду R_t и переходим в состояние S_{t+1}. Сохраняем набор в память: (S_t, A_t, R_t, S_{t+1}).
- Обучаем сеть:
  - Получаем из памяти выборку(s_j, a_j, r_j, s'_j)
  - Обновляем таргеты: $y_j = \begin{cases} r_j & if\ s'_j\ final \\ r_j + \gamma \max\limits_{a'} Q^{\theta}(s'_j, a') & otherwise \end{cases}$
  - $Loss(\theta) = \left(y_i - Q^{\theta}(s_i, a_i)\right)^2, \qquad \theta \leftarrow \theta - \alpha \nabla_{\theta} Loss(\theta)$

Как обучаем?


Reward (rt)
r_{t+1}
Agent
Deep Neural Network
Q(s, a)
States
Optimal policy
Optimal Action (a_t)
Environment
Input layer
Hidden layers
Output layers
s_{t+1}
Observe State (s_t)

Проблемы

- Работает плохо 😂



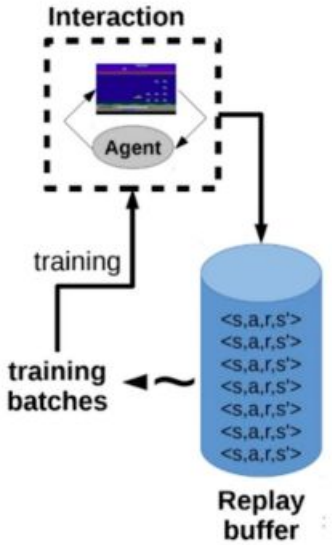
Причины:

1. Соседние состояния отличаются незначительно – сильно коррелированные данные для обучения.
2. Значение таргета и значение Q-функции в лоссе предсказывает одна и та же сеть. В ней стоит max – предсказания слишком оптимистичны.

Улучшение 1

Replay buffer

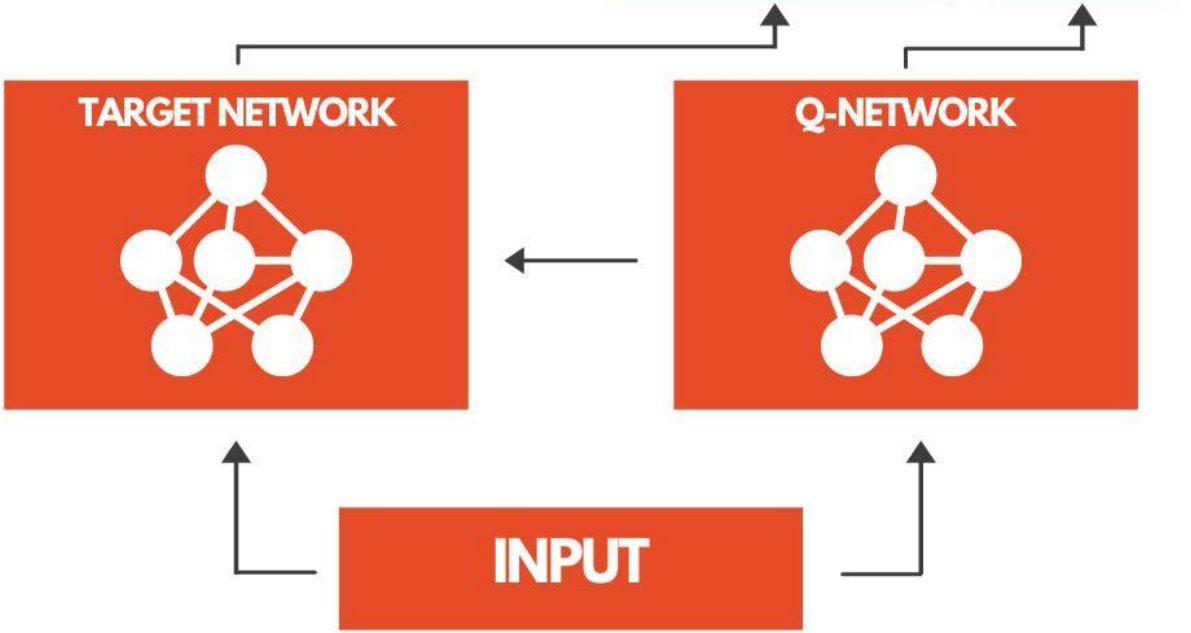
- Наполняем Buffer:
  - Набираем данные и складываем их в Buffer (s_j, a_j, r_j, s'_j)
- Обучаем сеть:
  - Получаем из памяти выборку (минибатч) $\{(s_j, a_j, r_j, s'_j)\}_{j=1}^{n}$ не последовательных, а именно случайно взятых эпизодов и учим на них.
  - Обновляем таргеты: $y_j = \begin{cases} r_j & if\ s'_j\ final \\ r_j + \gamma \max\limits_{a'} Q^{\theta}(s'_j, a') & otherwise \end{cases}$
  - $Loss(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - Q^{\theta}(s_i, a_i)\right)^2, \qquad \theta \leftarrow \theta - \alpha\nabla_{\theta}Loss(\theta)$



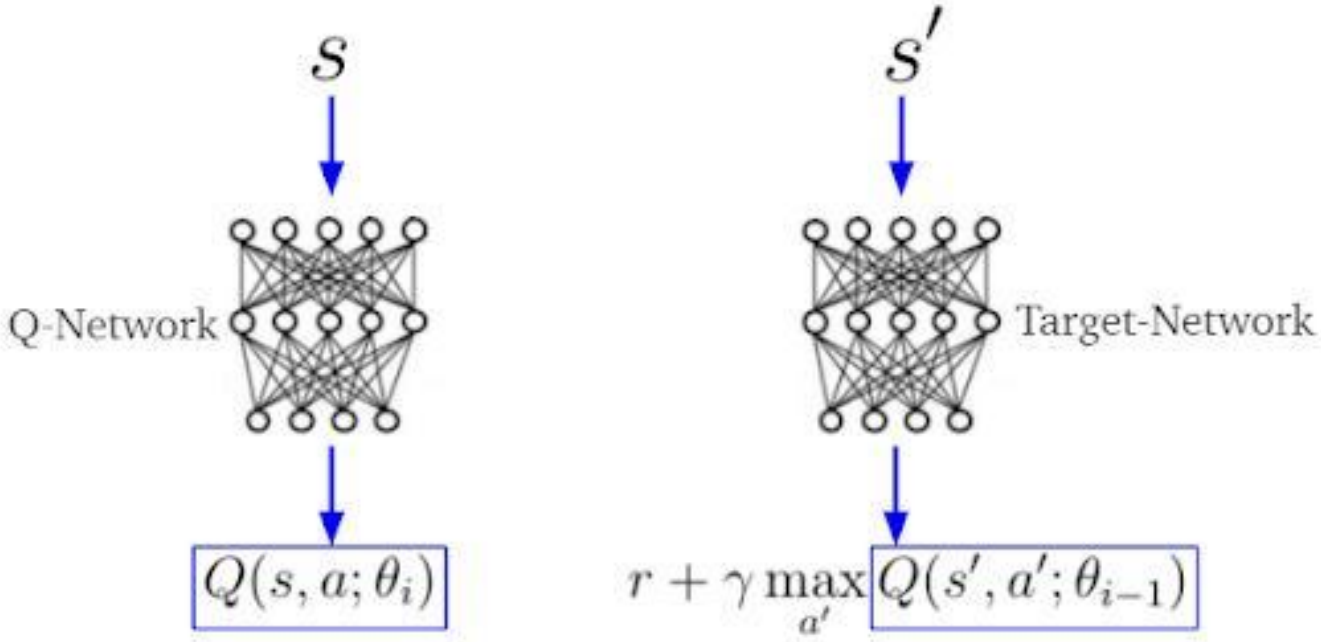
Улучшение 2

Fixed Target Network

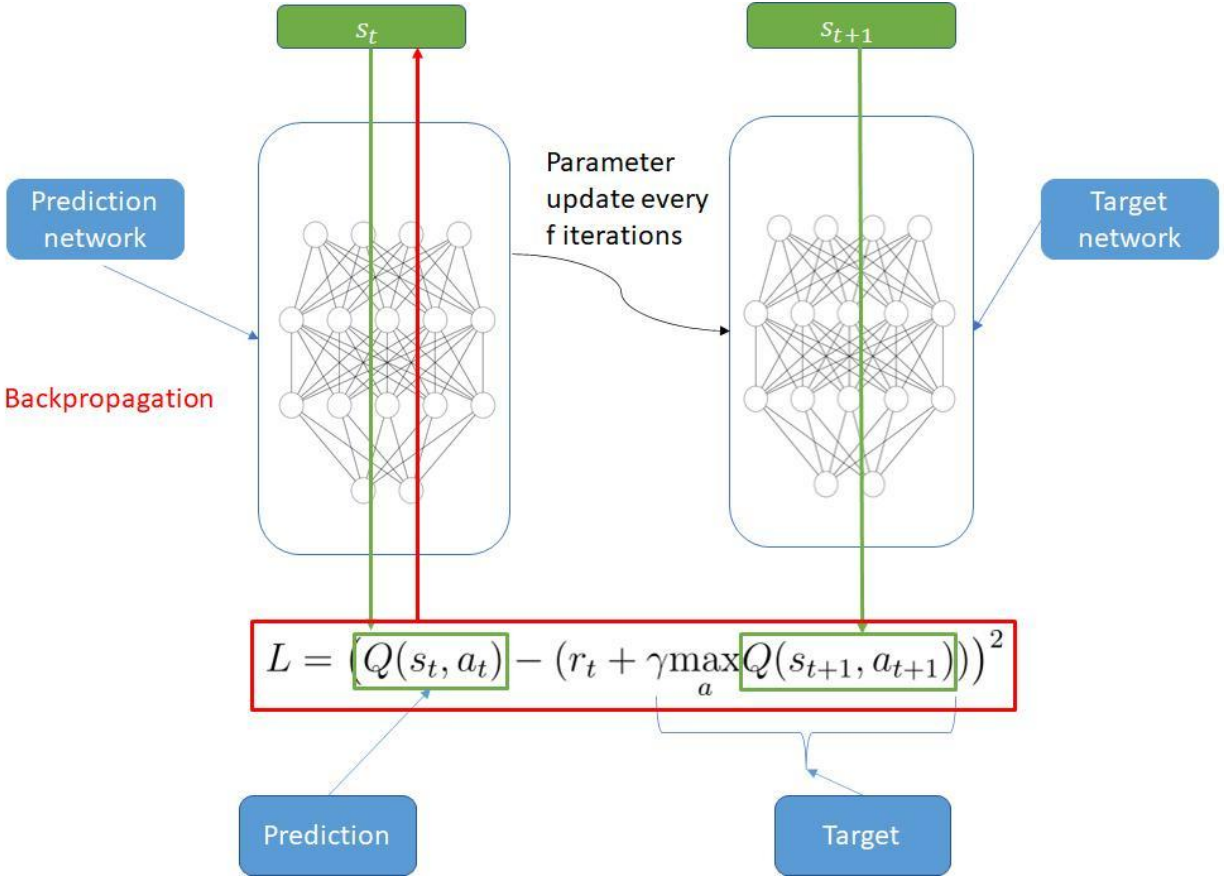
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big]$$



Fixed Target Network



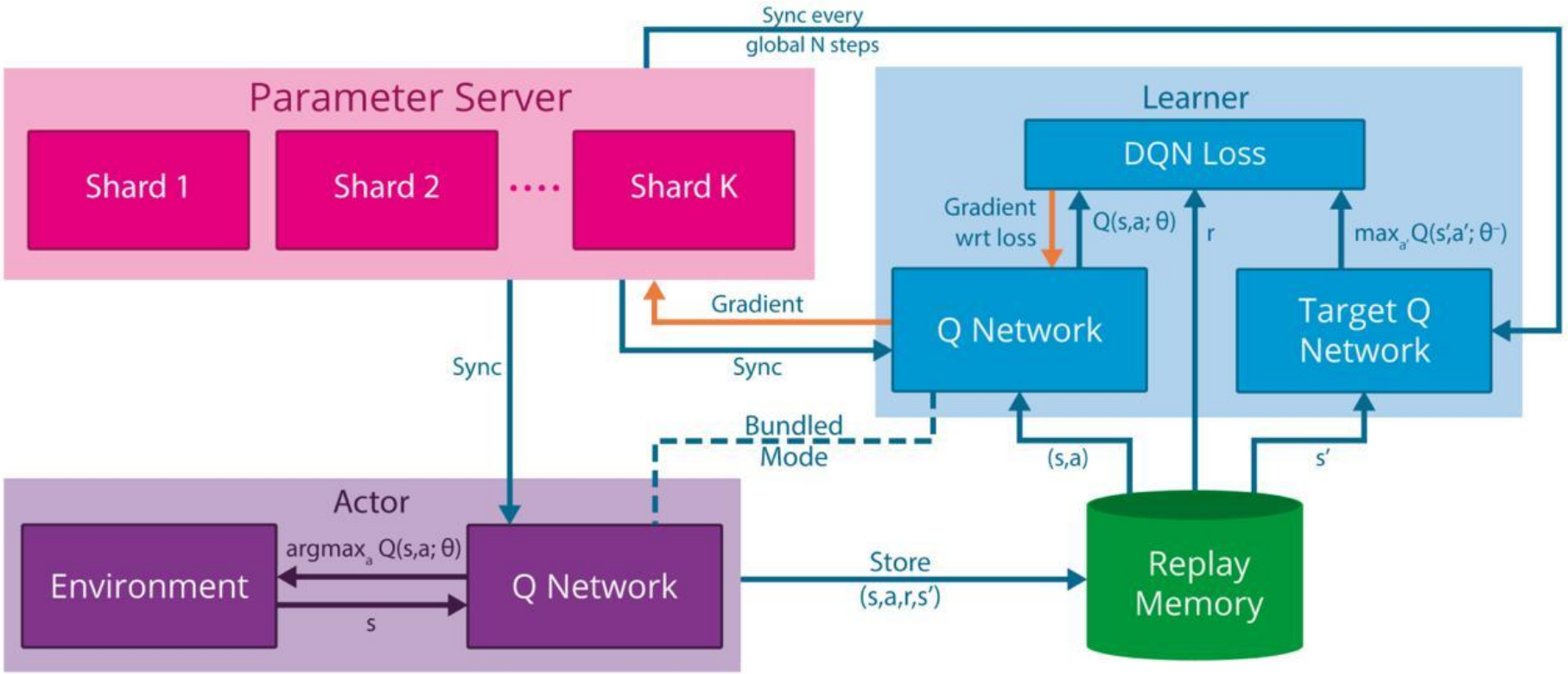
Fixed Target Network



Fixed Target Network

- Вводим две сети.
  - Одна используется для вычисления таргетов.
  - Вторая контролирует величину изменений.
- Обучаем сеть:
  - Фиксируем сеть ($\theta = \theta'$) и вычисляем по ней таргеты.
  - Обновляем таргеты на другой сети: $y_j = \begin{cases} r_j & if\ s'_j\ final \\ r_j + \gamma \max\limits_{a'} Q^{\theta'}(s'_j, a') & otherwise \end{cases}$
  - $Loss(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - Q^{\theta}(s_i, a_i)\right)^2$, $\theta \leftarrow \theta - \alpha\nabla_{\theta}Loss(\theta)$
  - Обновляем параметры фиксированной сети по обучаемой сети:
    - С периодичностью в несколько десятков эпизодов (*Hard target network*)
    - Берем взвешенное обновление $\theta \leftarrow \tau\theta + (1-\tau)\theta'$ (*Soft target network*)

Все вместе


Sync every
global N steps
Parameter Server
Shard 1
Shard 2
Shard K
Learner
DQN Loss
Gradient
wrt loss
Q(s,a; θ)
r
max_a' Q(s',a'; θ⁻)
Gradient
Sync
Q Network
Target Q
Network
Sync
Bundled
Mode
(s,a)
s'
Actor
Environment
argmax_a Q(s,a; θ)
s
Q Network
Store
(s,a,r,s')
Replay
Memory

Резюме