

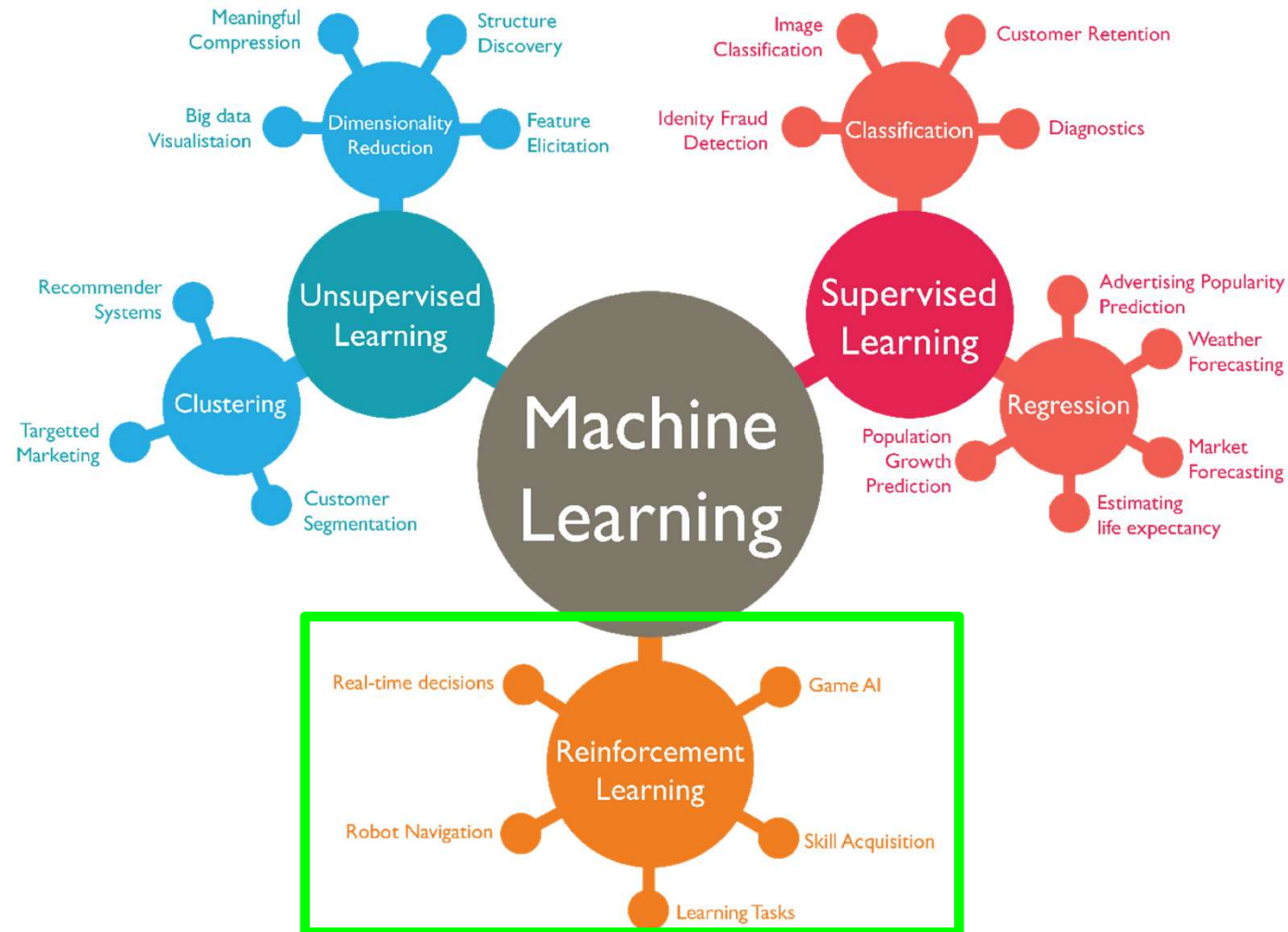
# Actor-Critic method

- Управление устройством
- Аркада
- Торговый бот

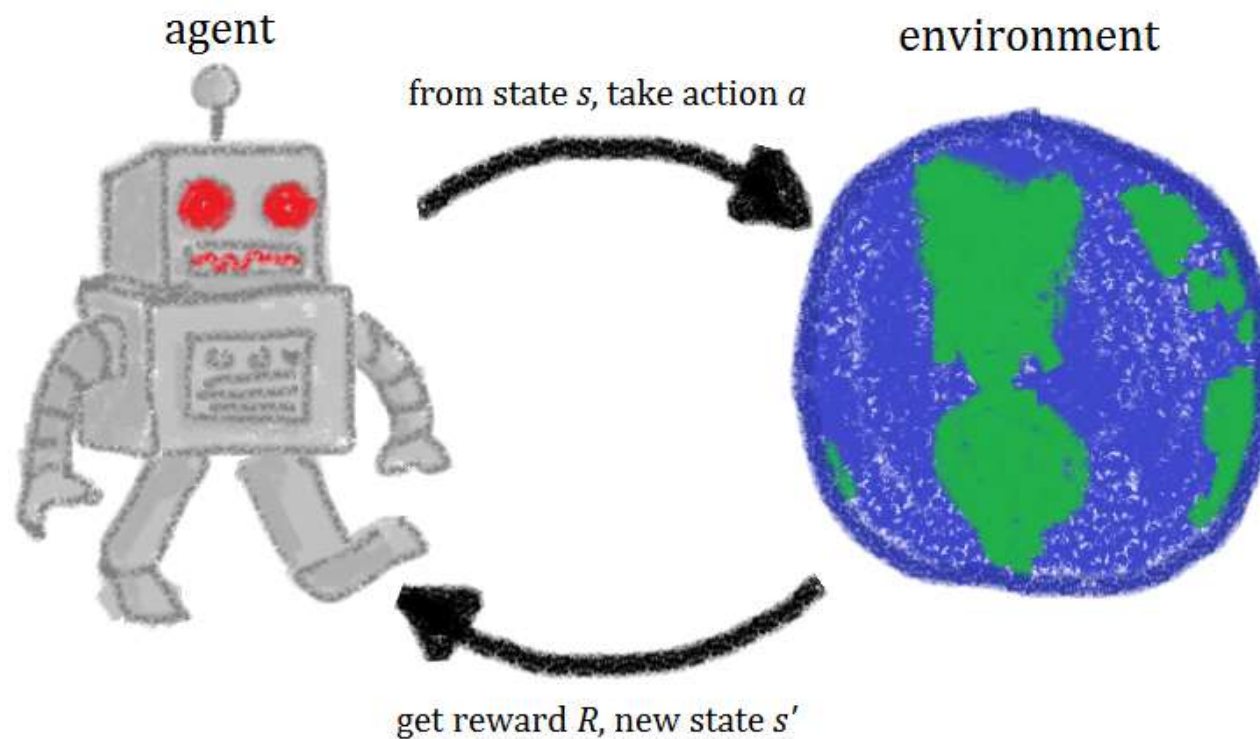
[https://t.me/devdvAI,](https://t.me/devdvAI)

<https://github.com/akumidv/startup-khv-ai-study>

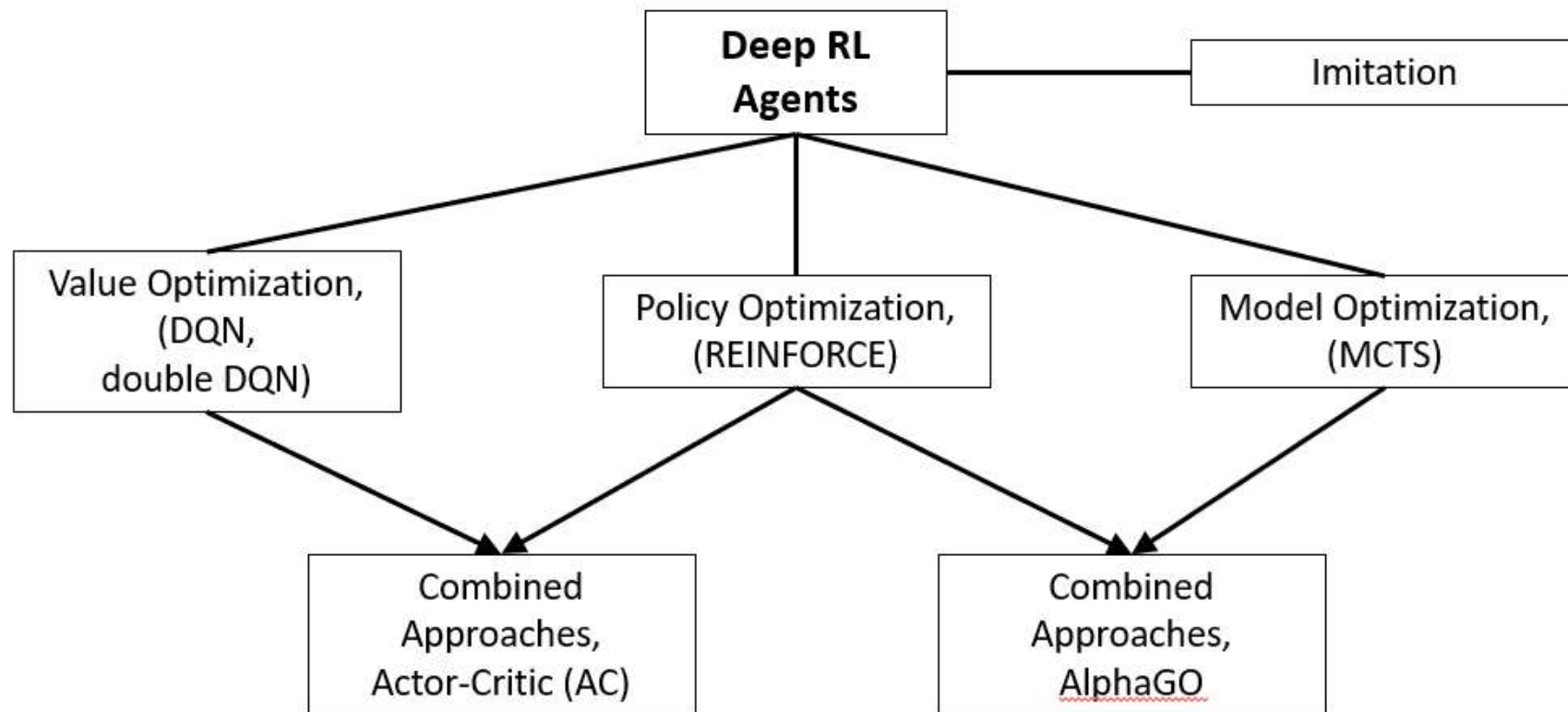
# Машинное обучение



# Понятие проблемы обучения с подкреплением



# Виды обучения агентов



# Задачи агента при обучении с подкреплением

- Автопилоты:
  - Оптимизация траекторий и динамическое планирование
  - Планирования движений
  - Оптимизация управления (колеса, скорость, тормоз)
- Бизнес
  - Оптимизация маршрутов бизнес-процессов
  - Минимизация энергопотребления
  - Максимизация прибыли от акций/криптовалют
- Прочие
  - Управление климатом в датацентрах
  - Персонализированные рекомендации
  - ИИ боты для персонажей компьютерных игр
  - Сопровождение терапии на основе диагноза, лекарств и анализов

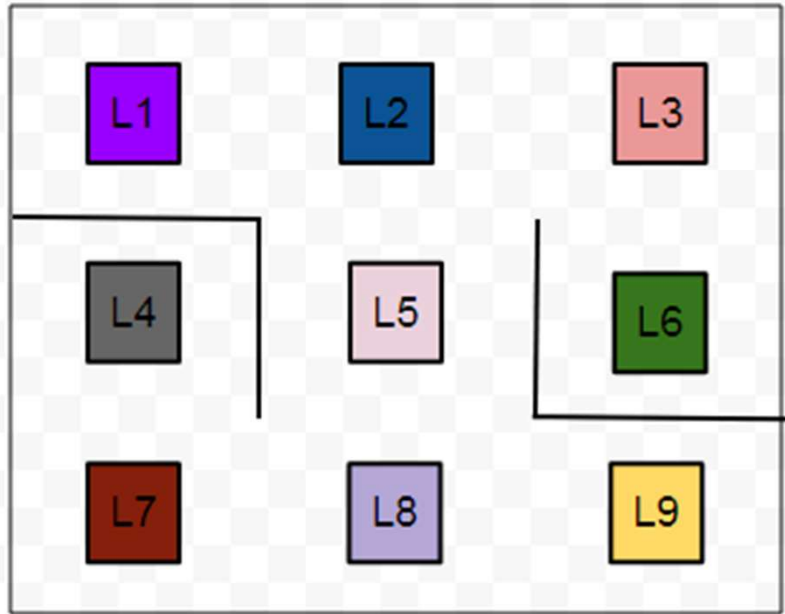


# Q and Value based reinforcement learning:

## Q-learning

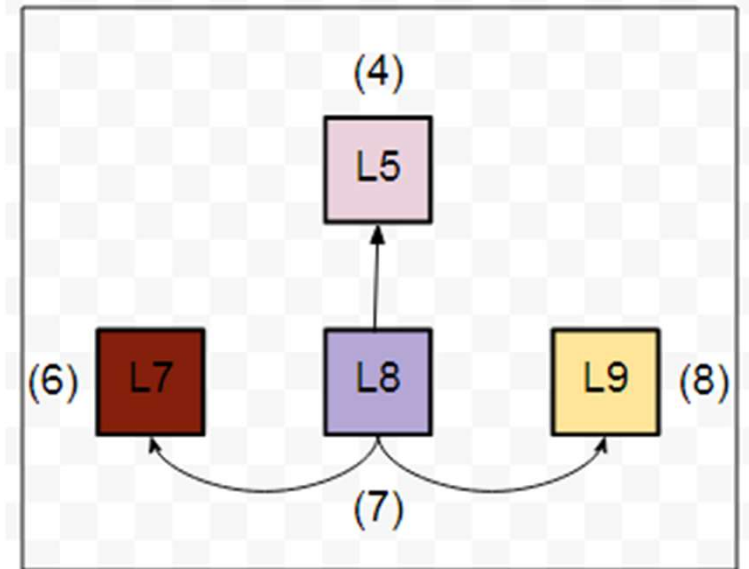
Источник: <https://blog.floydhub.com/an-introduction-to-q-learning-reinforcement-learning/>

# Пример: как роботу пройти из L9 в L1



L1-L9 состояния (s)

Переходы (a)



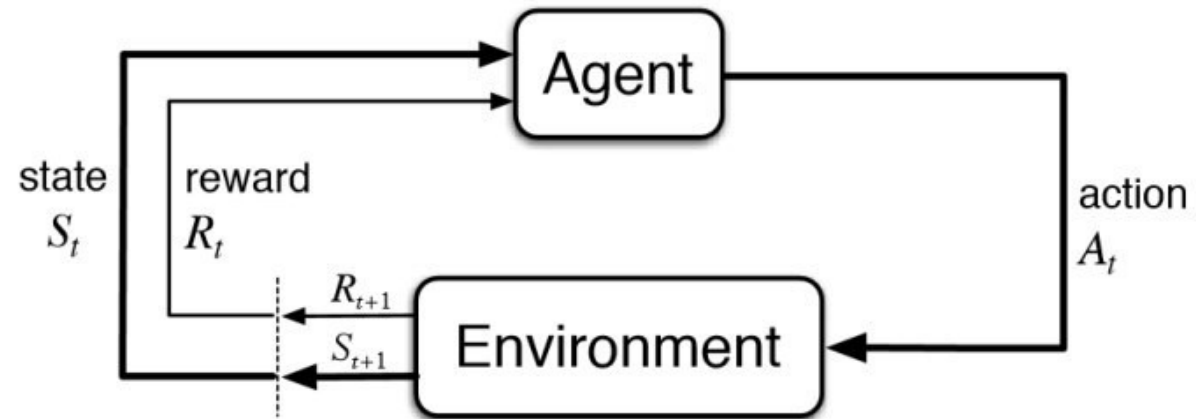
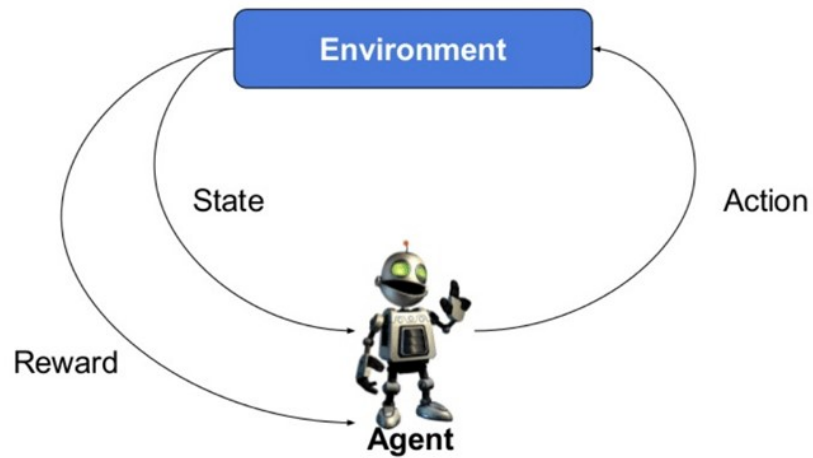




# SARSA алгоритм

State – Action – Reward – State - Action

Typical RL scenario



# Состояние и память для последовательного принятия решения

↑		
↑		
↑	←	A



1		
1		
1	1	1

1		
1	?	
A	1	



1		
0.9		
0.81	0.729	<u>Starting point</u>

Память  $V(s) = \max a(0 + 0.9 * 1) = 0.9$

# Уравнение Беллмана – последовательное принятие решения

$$V(x_0) = \max_{\{a_t\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \beta^t F(x_t, a_t),$$

$$V(s) = \max_a (R(s, a) + \gamma V(s'))$$

- $s$  = состояние (комната лабиринта)
- $a$  = действие action
- $s'$  = предыдущее состояние
- $\gamma$  = дисконтирование (уменьшение ценности)
- $R(s, a)$  = вознаграждение при действии  $a$  в состоянии  $s$
- $V(s)$  = ценность состояния

$$V(s) = \max_a (0 + 0.9 \cdot 1) = 0.9$$

	1	0.9
1	0.9	0.81
0.9	0.81	0.729
0.81	0.729	0.66

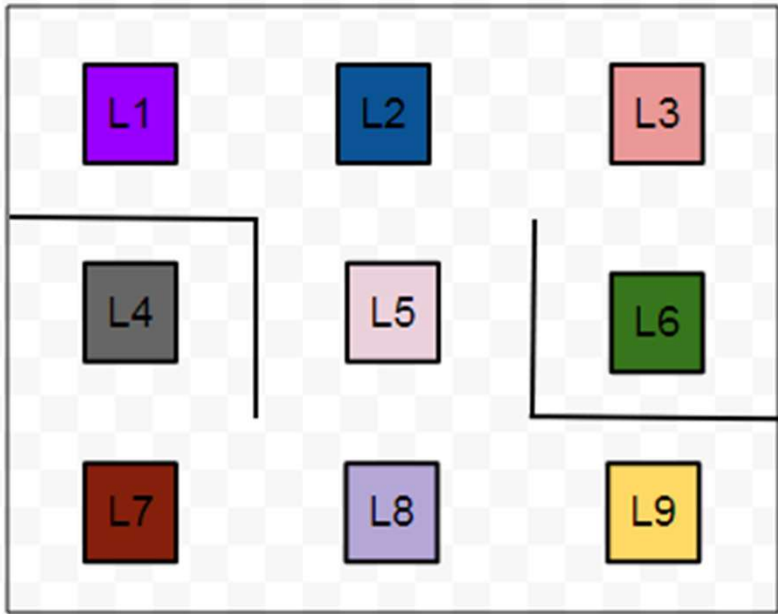
## Если окружение меняется?

В одном и том же месте, но состояниями  
меняющимися со временем и награда будет  
разной

Временная разница – Temporal difference

$$TD(a,s) = Q(s,a) - Q_{t-1}(s,a)$$

# PAC4ET 01\_Q\_Learning.ipynb



```
[62] route = get_optimal_route('L9', 'L1', Q)
```

L1	[3996.	2249.	0.	0.	0.	0.	0.	0.	0.]
L2	[2998.	0.	1688.	0.	0.	0.	0.	0.	0.]
L3	[ 0.	2249.	0.	0.	0.	1267.	0.	0.	0.]
L4	[ 0.	0.	0.	0.	0.	0.	951.	0.	0.]
L5	[ 0.	2249.	0.	0.	0.	0.	0.	1267.	0.]
L6	[ 0.	0.	1688.	0.	0.	0.	0.	0.	0.]
L7	[ 0.	0.	0.	714.	0.	0.	0.	1267.	0.]
L8	[ 0.	0.	0.	0.	1688.	0.	951.	0.	951.]
L9	[ 0.	0.	0.	0.	0.	0.	0.	1267.	0.]

L1 L2 L3 L4 L5 L6 L7 L8 L9



route

```
['L9', 'L8', 'L5', 'L2', 'L1']
```

# Ограничение моделей с памятью

---

- Нужно рассчитать и хранить все состояния для всех действий
- Агент (робот) выбирает из заданных извне состояний, а не оценивает среду
- Не предусмотрена возможность(вероятность) сбоя
- Нет динамического формирования начальных условий. Если есть – многомерное пространство (параметрическое)



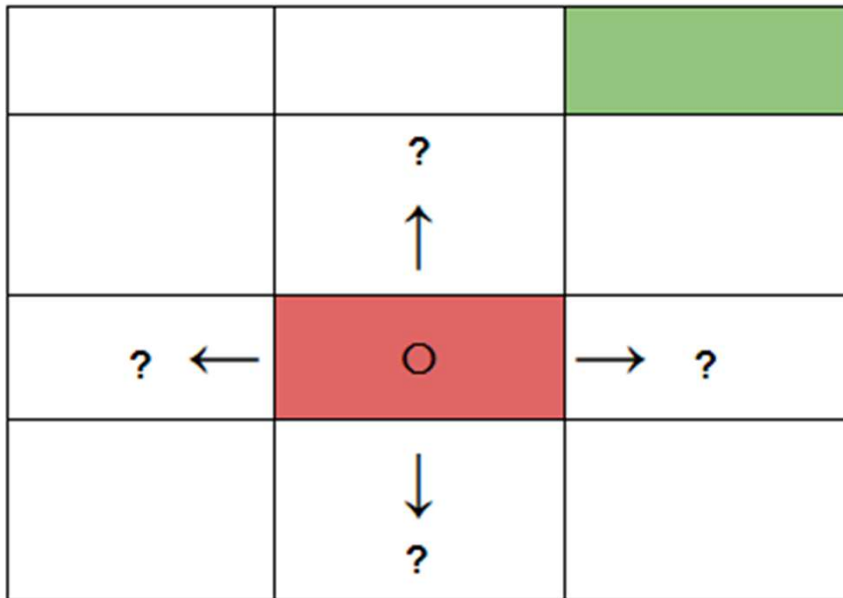
- Решение – цепи Маркова – принятие решение в ситуациях частично случайного окружения и частично под контролем актора

# Цепи Маркова

**Беллман**  $V(s) = \max_a (R(s,a) + \gamma V(s'))$

**Марков**  $V(s) = \max_a (R(s,a) + \gamma \sum_{s'} P(s,a,s') V(s'))$

$P(s, a, s')$  – вероятность  
перемещения в  $s$  из  $s'$   
 $\gamma \sum_{s'} P(s,a,s') V(s')$  - ожидание случайности  
выбора



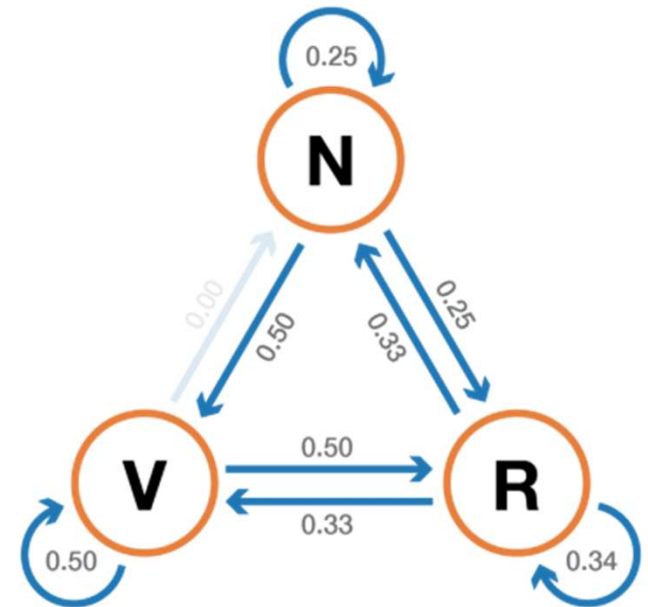
Цепи Маркова – stochastic policy based RL

# Цепи Маркова – в двух словах

$$P(\text{future} \mid \text{present, past}) = P(\text{future} \mid \text{present, ~~past~~})$$

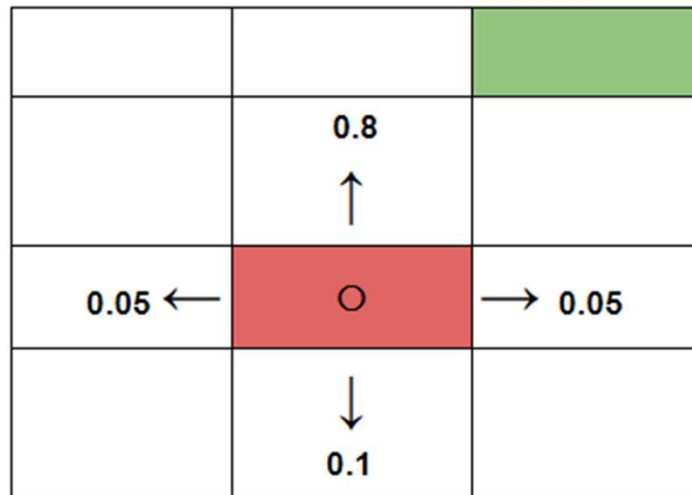
Markov property 

Следующее состояние  
зависит только от  
настоящего состояния и  
вероятности.





# Цепи Маркова: формула вознаграждения



$$V(s) = \max_a ( R(s,a) + \gamma (0.8V(\text{up}) + 0.1V(\text{down}) + \dots) )$$

Расчет вознаграждения после достижения цели.

# Q-learning от качества решения (Quality)

	$0.8 (V(s1))$ ↑	
$0.05 \leftarrow$ $(V(s4))$	○	$\rightarrow 0.05 (V(s2))$
	↓ $0.1 (V(s3))$	



С ценностью

	$Q(s1, a1)$ ↑	
$Q(s4, a4) \leftarrow$	○	$\rightarrow Q(s2, a2)$
	↓ $Q(s3, a3)$	


С качеством решения

Что такое качество решения?

## Качество решения = Q-values

$$V(s) = \max_a \left( R(s, a) + \gamma \sum_{s'} P(s, a, s') V(s') \right)$$


$$Q(s, a) = R(s, a) + \gamma \sum_{s'} (P(s, a, s') V(s'))$$


$$Q(s, a) = R(s, a) + \gamma \sum_{s'} \left( P(s, a, s') \max_{a'} Q(s', a') \right)$$

# Policy-based reinforcement learning

Источник: <https://medium.com/intro-to-artificial-intelligence/reinforce-a-policy-gradient-based-reinforcement-learning-algorithm-84bde440c816>

# Маршрут

Награда за следующий шаг

$$r_t = R(s_t, a_t, s_{t+1})$$

Маршрут с видимым горизонтом

$$R(\tau) = \sum_{t=0}^T r_t.$$

Маршрут с бесконечным горизонтом и дисконтированием

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t.$$

# $\gamma$ - Fog of war?

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t.$$



## Сумма вознаграждений – определяет маршрут

С шага 0 до шага  $T-1$  перед финальным состоянием

$$R(\tau) = \sum_{t=0}^{T-1} R(s_t, a_t)$$

$$\tau = (s_0, a_0, \dots, s_{T-1}, a_{T-1})$$

состояния и переходы по маршруту

# Различие ценности и политики

---

**ЧЕМ ОТЛИЧАЕТСЯ ПОЛИТИК ОТ ШАХМАТИСТА?**



## Модель ценности и политики в формуле

$$V^*(s) = \max_a Q^*(s, a) \quad \forall s \in \mathbb{S}$$

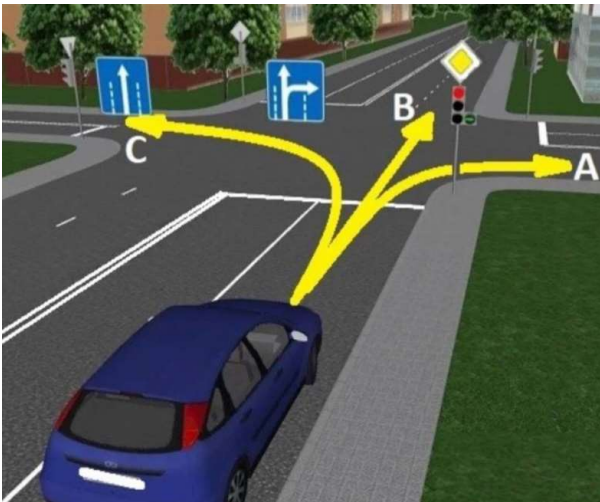
$$\pi^*(s) = \boxed{\arg} \max_a Q^*(s, a) \quad \forall s \in \mathbb{S}$$

# Политика (policy)

Вероятная для  
следующего состояния  
от текущего и действия

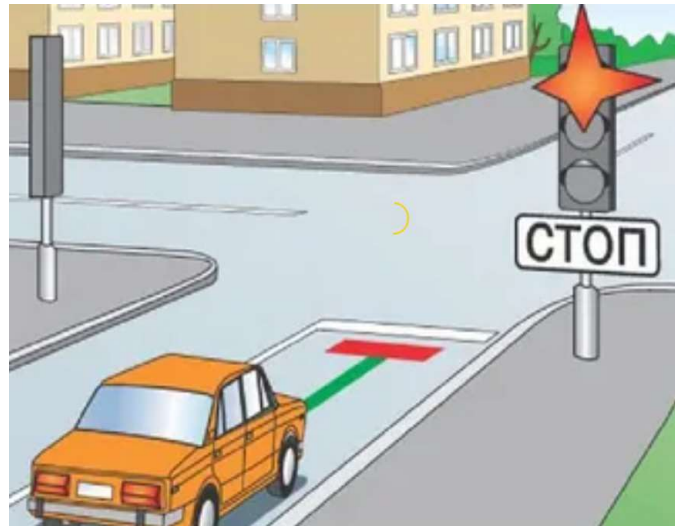
$$p(s_{t+1}|s, a)$$

$$\pi(a, s) = Pr(a|s)$$



Детерминированная,  
если вероятность 1

$$\mu(s) = a$$



Заданная  
модель



# Вероятность траектории

$$P(\tau|\theta) = p(s_0) \prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t) \pi_{\theta}(a_t, s_t)$$

↑  
Вероятность  
начального  
состояния

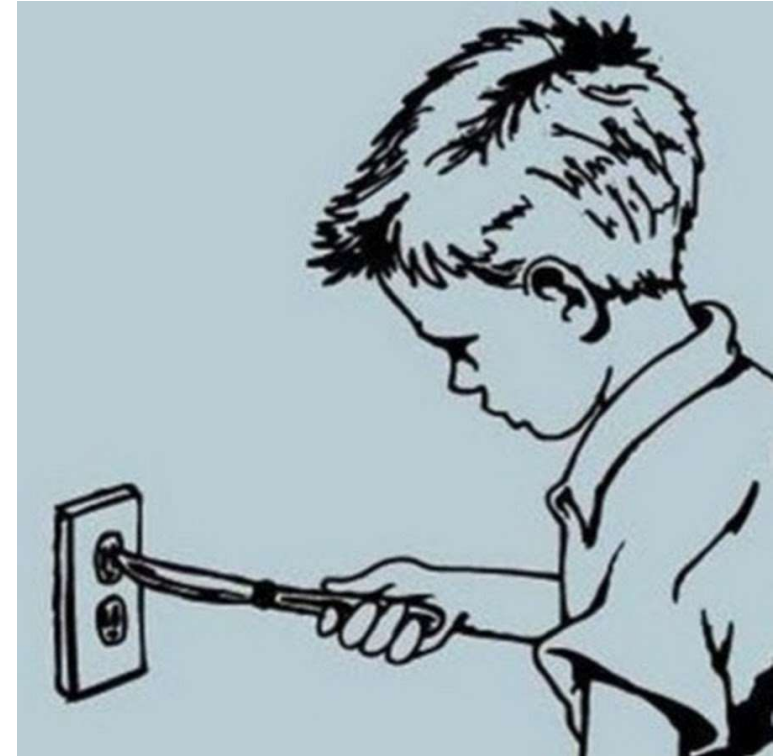
↑  
Вероятность  
перехода в  
новое  
состояние из  
текущего с  
действием

↑  
Параметричес  
кая функция  
политики  
перехода в  
следующее  
состояние

# Как найти политику

---

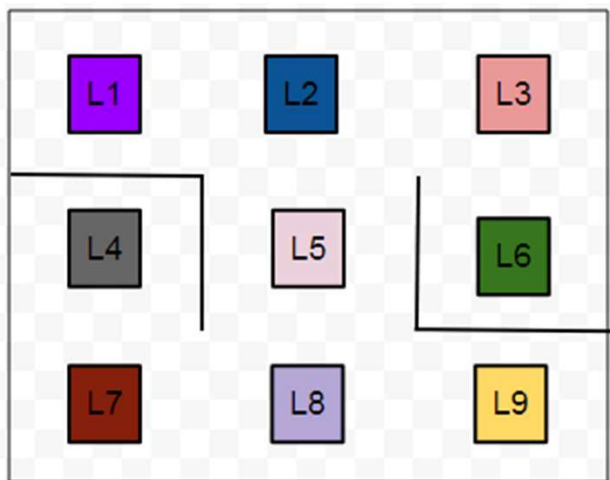
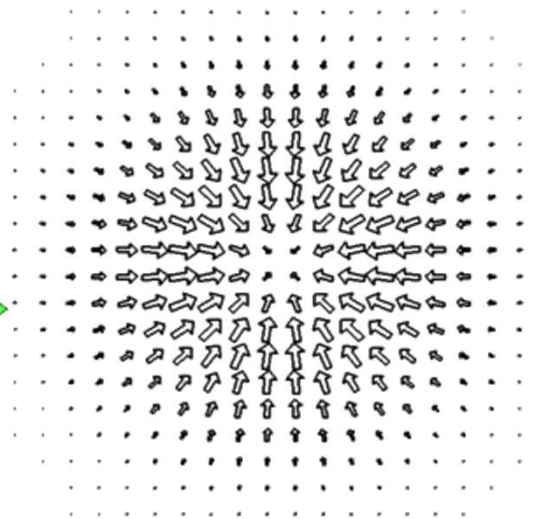
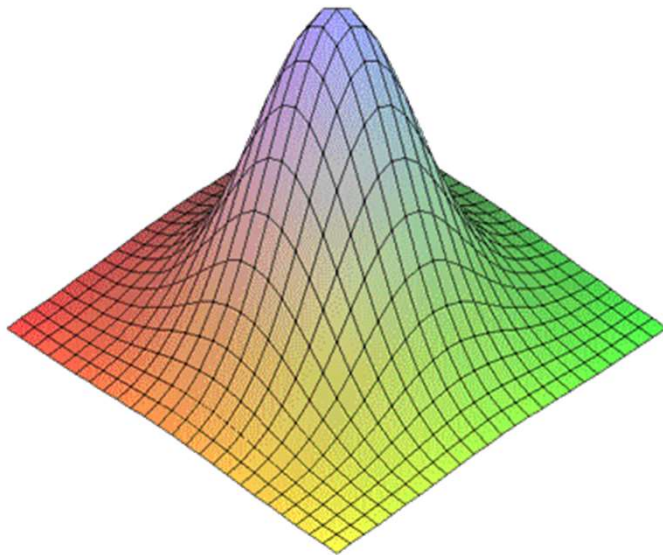
- Мы можем получать состояния
- У нас ограниченный набор действий
- Как найти  $\pi$  содержащую параметры  $\theta$ ?



# Policy gradients: $\pi_{\theta}(a|s)$

Источник:

# Что такое градиент



```
[62] route = get_optimal_route('L9', 'L1', Q)
```

ИНТУИТИВНО

L1	[[3996.	2249.	0.	0.	0.	0.	0.	0.	0.]
L2	[2996.	0.	1688.	0.	0.	0.	0.	0.	0.]
L3	[0.	2249.	0.	0.	0.	1267.	0.	0.	0.]
L4	[0.	0.	0.	0.	0.	0.	951.	0.	0.]
L5	[0.	2249.	0.	0.	0.	0.	0.	1267.	0.]
L6	[0.	0.	1688.	0.	0.	0.	0.	0.	0.]
L7	[0.	0.	0.	714.	0.	0.	0.	1267.	0.]
L8	[0.	0.	0.	0.	1688.	0.	951.	0.	951.]
L9	[0.	0.	0.	0.	0.	0.	0.	1267.	0.]



route

```
['L9', 'L8', 'L5', 'L2', 'L1']
```

# Как посчитать градиент?

Градиент ( $\nabla$ , набла) - вектор показывающий направление наибольшего изменения функции. Сумма частных производных.

$$\varphi = f(x, y, z)$$

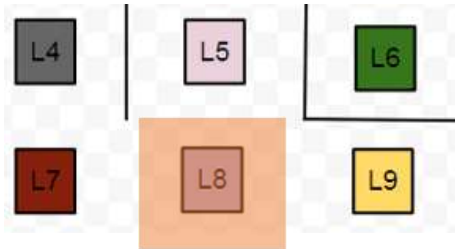
$$\text{grad}(\varphi) = \nabla \varphi = \frac{\partial \varphi}{\partial x} i + \frac{\partial \varphi}{\partial y} j + \frac{\partial \varphi}{\partial z} k$$

Производная - показывает скорость изменения функции в заданной точке или **тангенс угла** наклона касательной к графику функции.

## ЕСЛИ НЕТ ФУНКЦИИ, НО ЕСТЬ ЗНАЧЕНИЯ ФУНКЦИИ?

# Тангенс

Угловым коэффициентом или тангенсом угла наклона — отношение изменения  $y$  к  $x$



	L1	L2	L3	L4	L5	L6	L7	L8	L9	
L7	[	0.	0.	0.	714.	0.	0.	1267.	0.]	
L8	[	0.	0.	0.	0.	1688.	0.	951.	0.	951.]
L9	[	0.	0.	0.	0.	0.	0.	1267.	0.]	]

$Y8\_9 = 951$

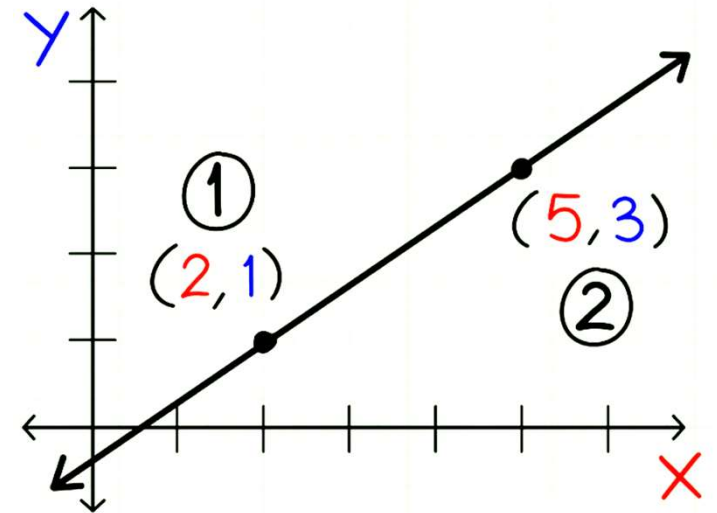
$Y8\_7 = 951$

$Y8\_5 = 1688$

$tg\_a\_9 =$

$tg\_a\_7 =$

$th\_a\_5 =$



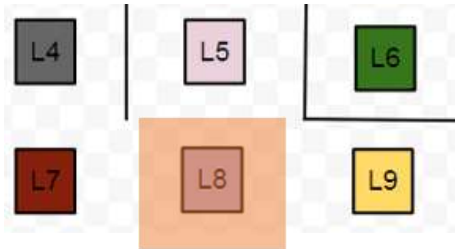
$$m = \frac{Y_2 - Y_1}{X_2 - X_1}$$

$$m = \frac{3 - 1}{5 - 2} = \frac{2}{3} = 0.67$$



# Тангенс

Угловым коэффициентом или тангенсом угла наклона — отношение изменения y к x



	L1	L2	L3	L4	L5	L6	L7	L8	L9
L7	0.	0.	0.	714.	0.	0.	0.	1267.	0.
L8	0.	0.	0.	0.	1688.	0.	951.	0.	951.
L9	0.	0.	0.	0.	0.	0.	0.	1267.	0.

$$Y8\_9 = 951$$

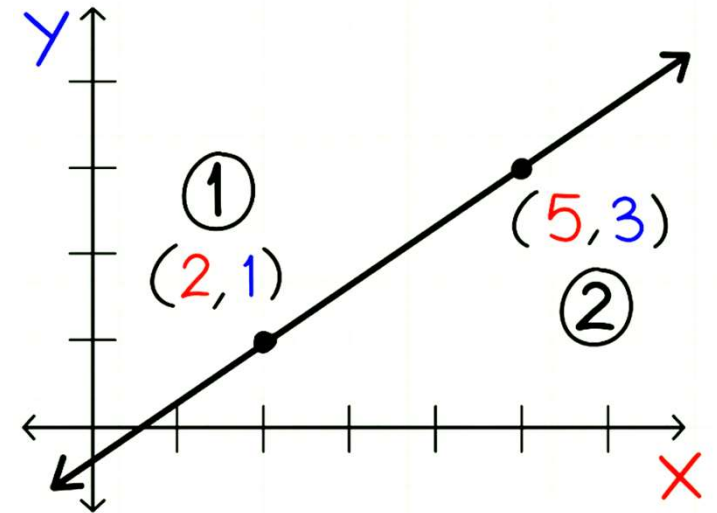
$$Y8\_7 = 951$$

$$Y8\_5 = 1688$$

$$tg\_a\_9 = (951 - 0)/-1$$

$$tg\_a\_7 = (1267 - 0)/1$$

$$th\_a\_5 = (1688 - 0)/1$$



$$m = \frac{Y_2 - Y_1}{X_2 - X_1}$$

$$m = \frac{3 - 1}{5 - 2} = \frac{2}{3} = 0.67$$

# $\pi_\theta(a|s)$ – параметризованная политика

- Задача её максимизации - поиск максимума функции  $J(\pi_\theta)$ , где  $E_{\pi_\theta}$  ожидание траектории для политики с параметрами.

$$J(\pi_\theta) = E_{\pi_\theta} \left[ \sum_{t=0}^{T-1} R(s_t, a_t) \right] = E_{\pi_\theta} [R(\tau)] = \sum_{\tau} P(\tau | \pi_\theta) R(\tau)$$

Вероятная  $\pi_\theta(a|s)$

Методом градиента  $\nabla$  - направления для возрастания(убывания) значения

$$\nabla J(\pi_\theta) = \nabla E_{\pi_\theta} [R(\tau)]$$

Градиент дает направление наибольшего(наименьшего) изменения.  
Значение указывает на максимальную степень изменения.

## Обновление параметров политики

$$\theta = \theta + \alpha \nabla E_{\pi_{\theta}}[R(\tau)]$$



$$\theta = \theta + \alpha \nabla J(\theta)$$

$\alpha$  - Learning rate

## $\nabla J(\pi_\theta)$ - пропустим?

$$\begin{aligned}\nabla E_{\pi_\theta}[R(\tau)] &= \nabla_\theta \sum_\tau P(\tau|\theta) R(\tau) \\&= \sum_\tau \nabla_\theta P(\tau|\theta) R(\tau) , \text{ bring gradient under summation} \\&= \sum_\tau P(\tau|\theta) \frac{\nabla_\theta P(\tau|\theta)}{P(\tau|\theta)} R(\tau) , \text{ both multiply and divide by } P(\tau|\theta) \\&= \sum_\tau P(\tau|\theta) \nabla_\theta \log P(\tau|\theta) R(\tau) , \text{ log-derivative trick based on } \nabla_\theta \log(z) = \frac{\nabla_\theta z}{z} \\&= E_{\pi_\theta}(\nabla_\theta \log P(\tau|\theta) R(\tau)) , \text{ return to expectation form}\end{aligned}$$

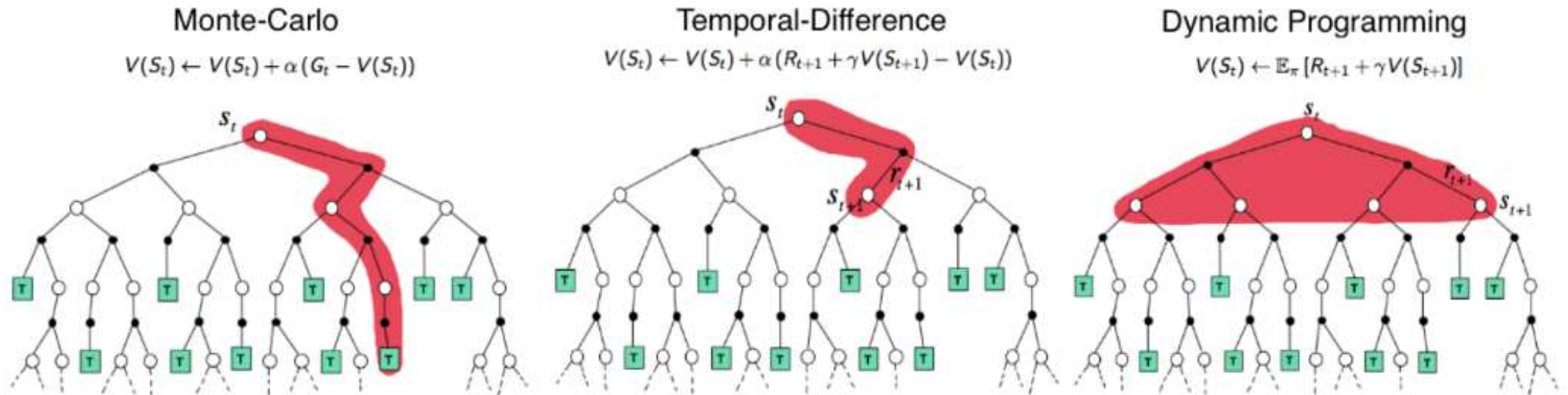
Вероятность, что траектория приведет  
к конечному состоянию

$$P(\tau|\theta) = p(s_0) \prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t) \pi_\theta(a_t, s_t)$$

$$\nabla J(\theta) = E_{\pi_\theta} \left( \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t, s_t) R(\tau) \right)$$

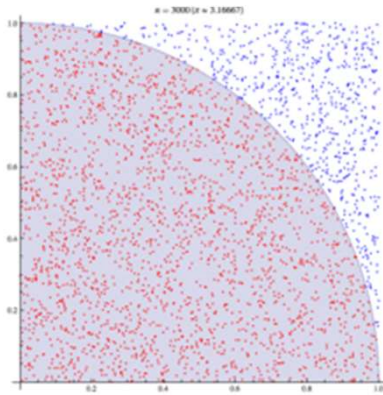
# REINFORCE – метод усиления

## ВЫБОР ГРАДИЕНТА ПОЛИТИКИ МЕТОДОМ МОНТЕ-КАРЛО



Поиск параметров для определения маршрута из среды

# МЕТОД МОНТЕ-КАРЛО



$$P_1 = S_{\text{круг}} / S_{\text{квадрата}} = \pi R^2 / a^2 = \pi R^2 / (2 R)^2 = \pi R^2 / (2 R)^2 = \pi / 4$$

$$P_2 = N_{\text{попавших в круг}} / N_{\text{точек}}$$

# РАСЧЕТ МАТ.ОЖИДАНИЯ МОНТЕ-КАРЛО

$$\int_a^b f(x) dx$$

$$\mathbb{E}f(u) = \int_a^b f(x)\varphi(x) dx,$$

$$\mathbb{E}f(u)$$

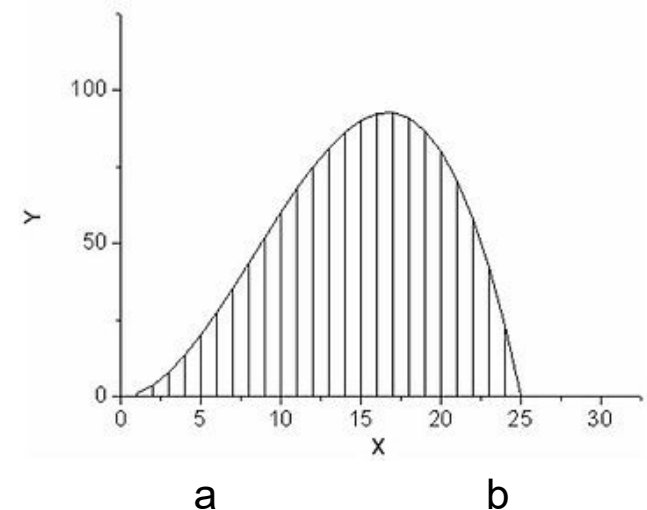
математическое ожидание случайной величины

$$\varphi(x) = \frac{1}{b-a}$$

плотность распределения случайной величины


$$\int_a^b f(x) dx = (b-a)\mathbb{E}f(u),$$

$$\mathbb{E}f(u) = \frac{1}{N} \sum_{i=1}^N f(u_i)$$



# Максимизация функции градиентом выборкой по Монте-карло

$$\nabla J(\theta) = E_{\pi_{\theta}} \left( \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t, s_t) R(\tau) \right)$$


$$\nabla J(\theta) \approx \frac{1}{N} \sum_{\tau \in N} \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t, s_t) R(\tau)$$

$N$  – количество траекторий для одного обновления градиента



# Алгоритм

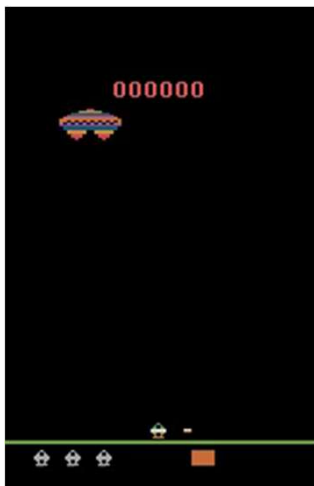
1. Инициализируем параметры  $\theta$  политики случайно
2. Рассчитываем для  $N$  траекторий в соответствии с параметрами политики  $\pi_\theta(a|s)$
3. Рассчитываем награду для траекторий  $R(\tau)$
4. Рассчитываем градиент
$$\nabla J(\theta) \approx \frac{1}{N} \sum_{\tau \in N} \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t, s_t) R(\tau)$$
5. Обновляем параметры политики
$$\theta = \theta + \alpha \nabla J(\theta)$$
6. Повторяем шаги 2-5 до  $N$

Источник <https://towardsdatascience.com/policy-gradient-methods-104c783251e0>

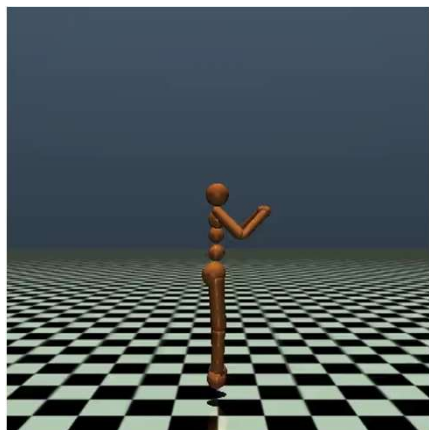
# Примеры

# GYM

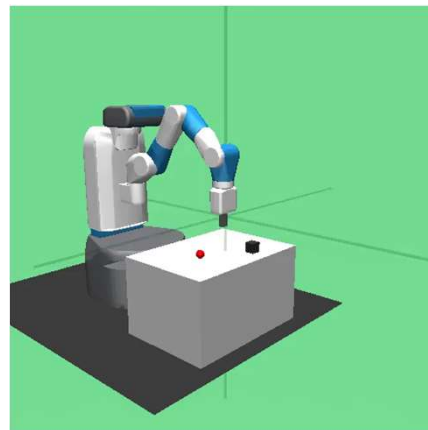
Разработка и сравнение обучающих алгоритмов с подкреплением



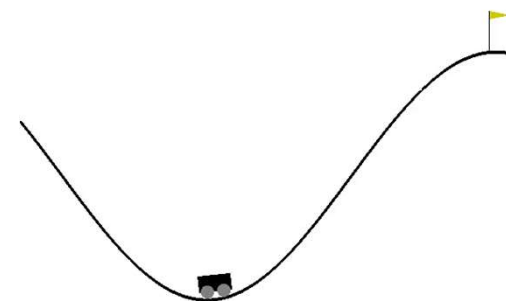
Аркада



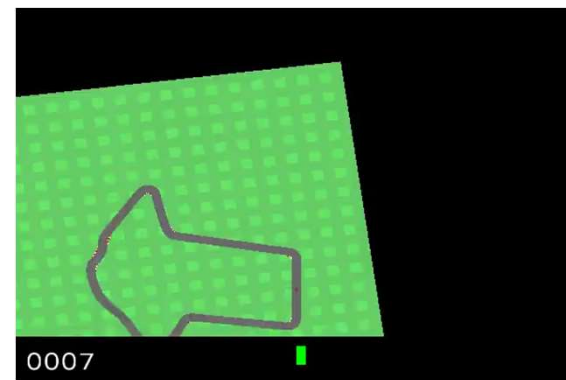
Управление движением



Симуляция задач основанных на цели



Последовательное управление



Управление

# Установка GYM под Windows

- <https://towardsdatascience.com/how-to-install-openai-gym-in-a-windows-environment-338969e24d30>
- **Build Tools for Visual Studio, Windows SDK, MSVC C++ build tools, C++ CMake** <https://visualstudio.microsoft.com/downloads/>
- **Swig** по ссылке <http://www.swig.org/download.html>
- `pip install gym`
- `pip install git+https://github.com/Kojoley/atari-py.git # ToyText`
- `pip install Box2D # + swing для Atari environments`
- `pip install box2d box2d-kengz`
- `pip install gym[atari] # Atari`
- `pip install gym[accept-rom-license]`
- `pip install gym[box2d]`
- **Xming** <https://sourceforge.net/projects/xming/>
- Перед запуском IDE: `set DISPLAY=:0` (можно в окружении)

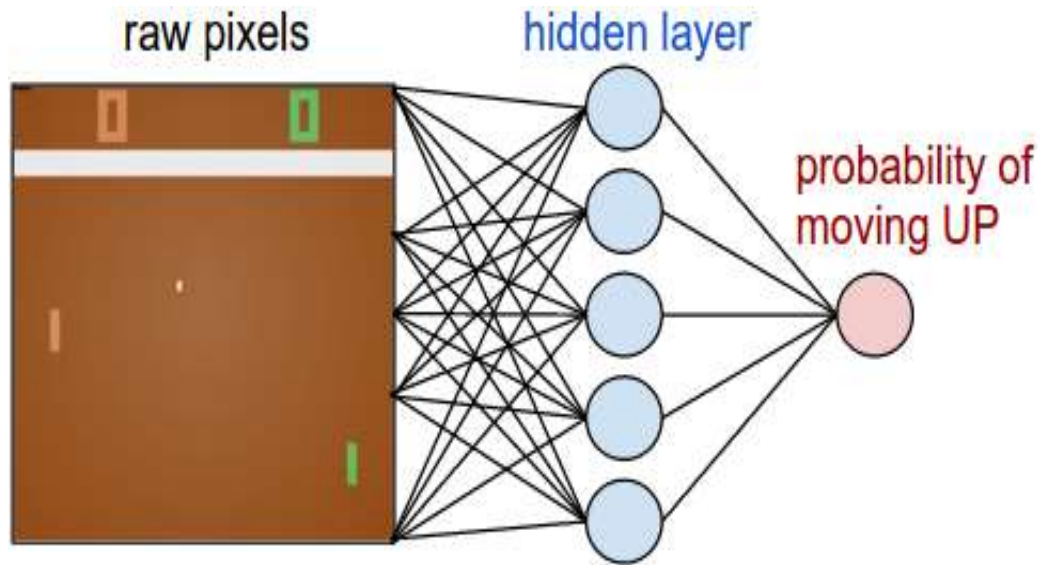
```
# Проверка
import gym
```

```
env = gym.make('CartPole-v0')
env.reset()
```

```
for _ in range(1000):
    env.render()
    env.step(env.action_space.sample())
```

```
env.close()
```

# Игра в пинпонг. Простая НС



- Policy predicts probability of moving UP
- W1 – 200 нейронов с 80x80 весами
- W2 – 1 нейрон с 80x80 весами
- 2000 эпизодов.  
Winrate 18/3 (average -18.0)

02\_pong\_gradient.py

<http://karpathy.github.io/2016/05/31/rl/> - простая реализация градиентного спуска

## 02. АЛГОРИТМ ДЛЯ КАЖДОГО ЭПИЗОДА ИГРЫ

---

1. Готовим изображение – конвертируем в вектор размером 80x80
2. Определяем разницу с предыдущим вектором
3. Определяем вероятное действие на основе политики
4. Определяем случайное число и если оно меньше вероятности, действие ВВЕРХ, иначе вниз.
5. Сохраняем разницу текущего вектора и предыдущего вектора картинки ( $x_s$ )
6. Сохраняем состояние слоя нейрона ( $h_s$ )
7. Сохраняем коэффициент поощрения выбранного действия ( $dlogps$ )
8. Проводим наблюдение для выбранного действия
9. Сохраняем награду ( $drs$ )

## 02. ПОСЛЕ ОКОНЧАНИЯ ЭПИЗОДА

1. Преобразуем массив состояний по кол-ву действий в эпизоде в вектор (erx)
2. Преобразуем массив по ходам выходов нейронов в эпизоде в вектор (eph)
3. Преобразуем массив по ходам вероятностей действий в вектор erpdlogp
4. Преобразуем массив наград в вектор
5. Рассчитываем награду в зависимости от количества шагов за которые  $R(\tau)$  она получена (отрицательная проигрыш, положительная выигрыш партии) (discount\_rewards) и нормализуем значения  $\log \pi_{\theta}(a_t, s_t) R(\tau)$
6. Модифицируем значение вероятности шага на основе награды erpdlogp
7. Рассчитываем градиент и формируем изменение весов модели
8. Для каждого батча меняем веса модели на основе градиента и коэффициента обучения

# Игра в пинпонг. Тензорфлоу

- Та же самая логика, но нейронная сеть на основе TensorFlow
  - Policy предсказывает вероятность вверх
  - Входной слой 80x80
  - Слой 1– 512 полносвязанных нейронов
  - Выходной слой – 1 нейрон

03\_Policy\_Gradient\_for\_Pong.py

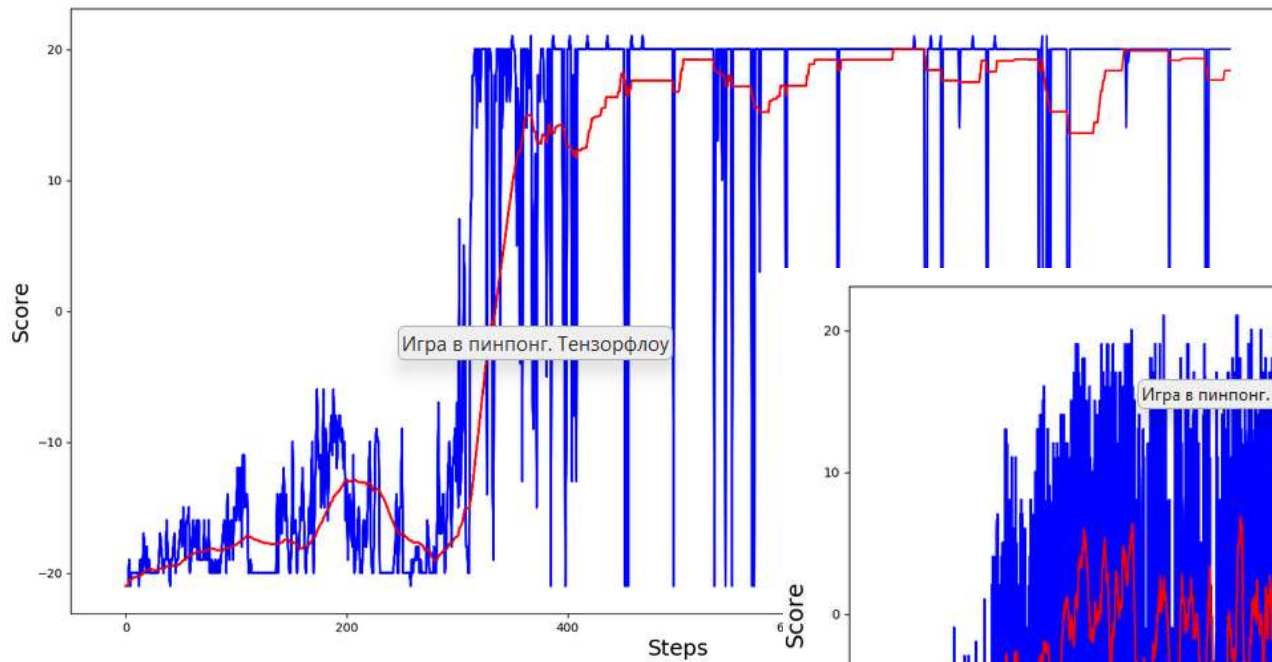
- 85/10000, average: -18.08
- 134/10000, average: -16.00

<https://blog.floydhub.com/spinning-up-with-deep-reinforcement-learning/> - код для тензорфлоу с ноутбуком

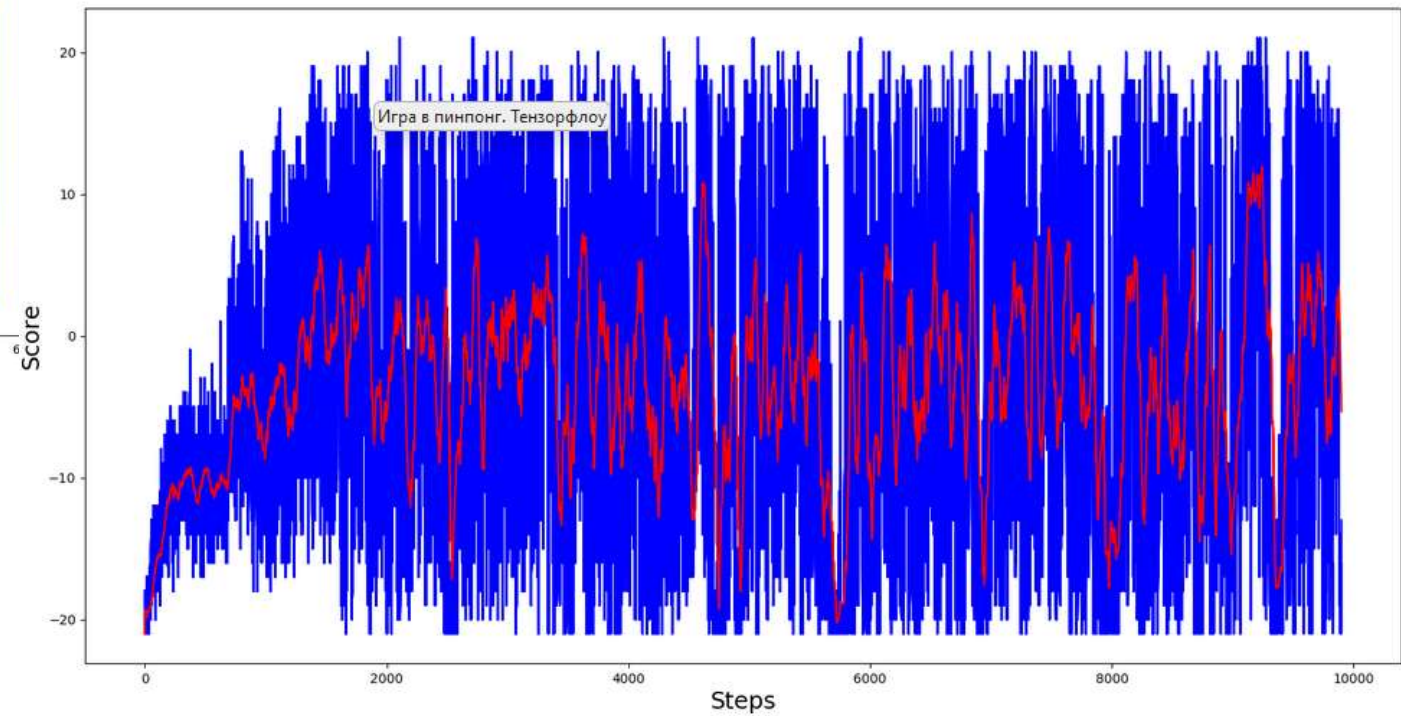


# Кривая обучения

PongDeterministic-v4

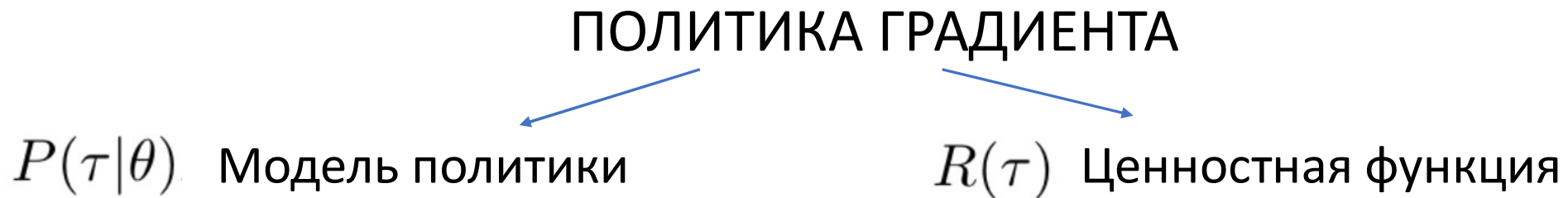


Pong-v0



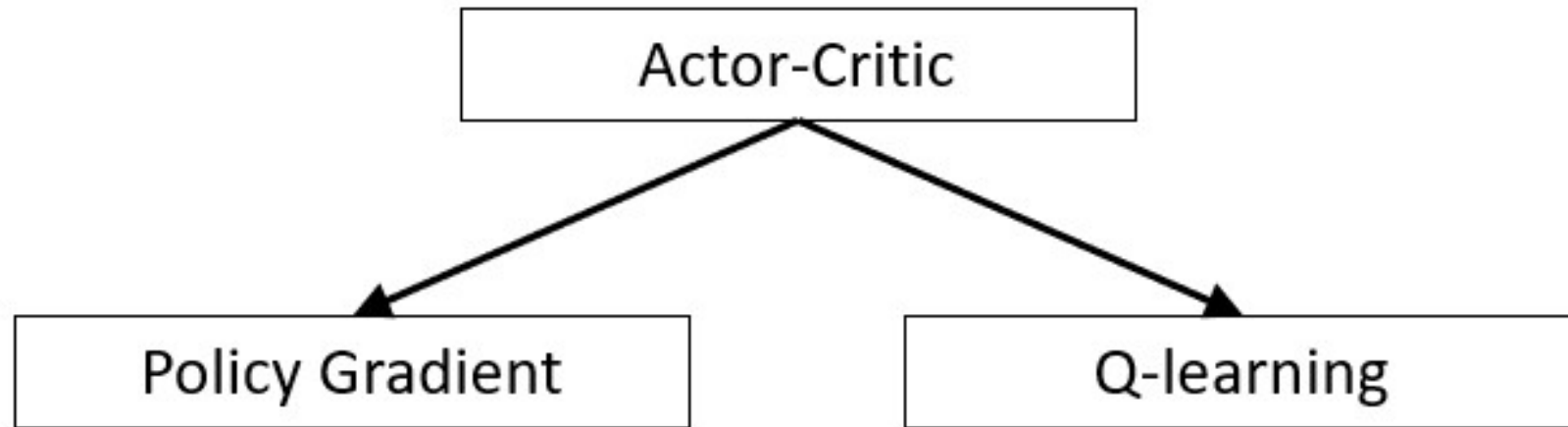
# Actor-Critic

# Policy Gradient и Actor-Critic



Для определения направления обновления модели политики  
можно отдельно рассчитывать ценностную функцию

# Policy Gradient и Actor-Critic



Актор

Обновляем параметры  $\theta$  для  $\pi_{\theta}(a|s)$  на основе критики

Критик

Ценность на основе действия  $Q_w(a|s)$  или на основе состояния  $V_w(s)$ , где  $w$  параметры функции

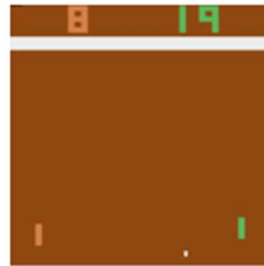
Источник: <https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html>  
<https://pylessons.com/A2C-reinforcement-learning/>

# Модель работы

Move  
down



Actor



Bad  
action



Critic

# Алгоритм

- Инициализируем  $s$  состоянием, параметры политики  $\theta$  и параметры функции ценности  $w$ , получаем действие  $a \sim \pi_\theta(a | s)$

- Цикл для  $t = 1 \dots T$

- Определяем награду  $r_t \sim R(s, a)$  и следующее состояние  $s' \sim P(s' | s, a)$
- Получаем следующее действие  $a' \sim \pi_\theta(a' | s')$
- Обновляем параметры политики  $\theta \leftarrow \theta + \alpha_\theta Q_{w(s,a)} \nabla_\theta \ln \pi_\theta(a | s)$
- Вычисляем коррекцию временной разницы (TD error) для ценности

$\delta t = r_t + \gamma Q_w(s', a') - Q_w(s, a)$  и обновляем параметры функции ценности

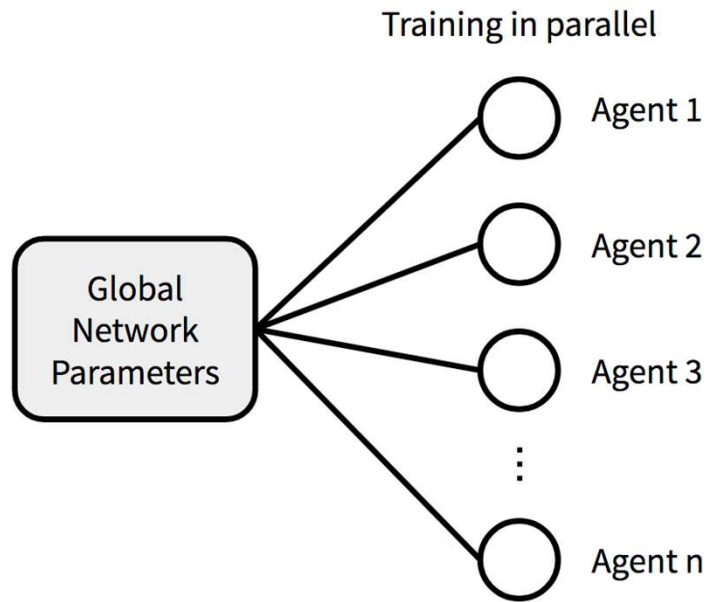
$$w \leftarrow w + \alpha_w \delta t \nabla_w Q_w(s, a)$$

- Обновляем  $a \leftarrow a'$  и состояние  $s \leftarrow s'$

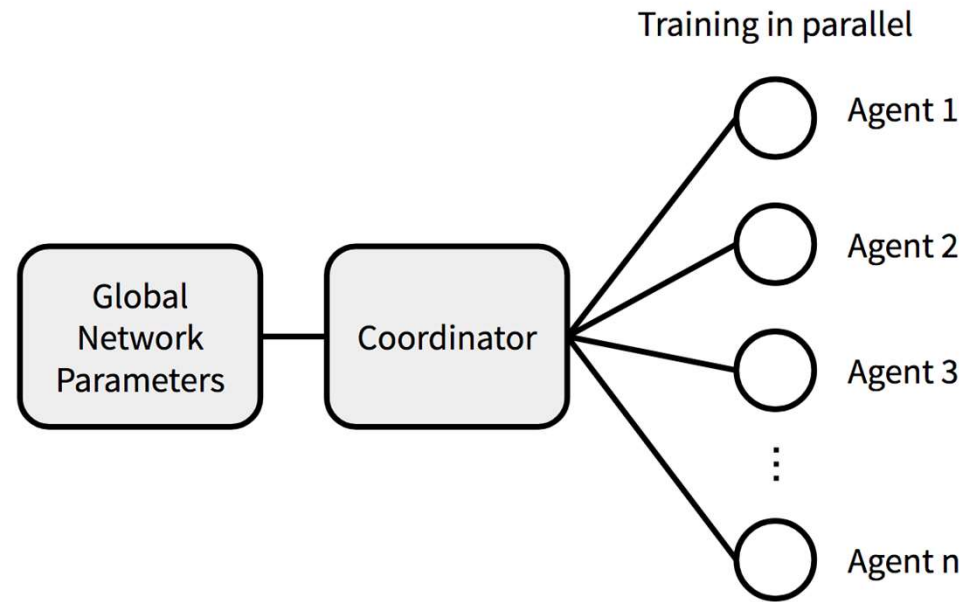
Коэффициенты обучения (learning rate)  $\alpha_\theta$  и  $\alpha_w$  определяем ретроспективно

# Actor-Critic

- A3C Asynchronous Advantage Actor-Critic
- A2C Asynchronous Actor-Critic



**A3C (Async)**



**A2C (Sync)**

# Отличия кода 04\_Actor-Critic.ipynb

```
def OurModel(input_shape, action_space, lr):
    X_input = Input(input_shape)

    X = Flatten(input_shape=input_shape)(X_input)

    X = Dense(512, activation="elu", kernel_initializer='he_uniform')(X)

    action = Dense(action_space, activation="softmax", kernel_initializer='he_uniform')(X)
    value = Dense(1, kernel_initializer='he_uniform')(X)

    Actor = Model(inputs = X_input, outputs = action)
    Actor.compile(loss='categorical_crossentropy', optimizer=RMSprop(lr=lr))

    Critic = Model(inputs = X_input, outputs = value)
    Critic.compile(loss='mse', optimizer=RMSprop(lr=lr))

    return Actor, Critic
```

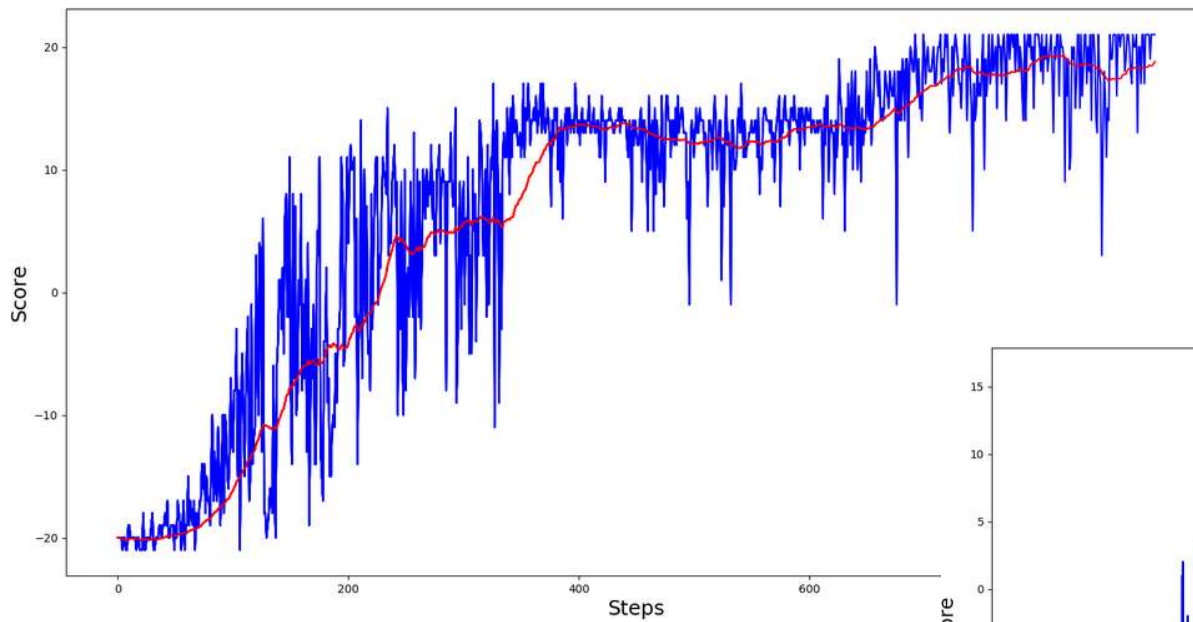


# Отличия кода 04\_Actor-Critic.ipynb

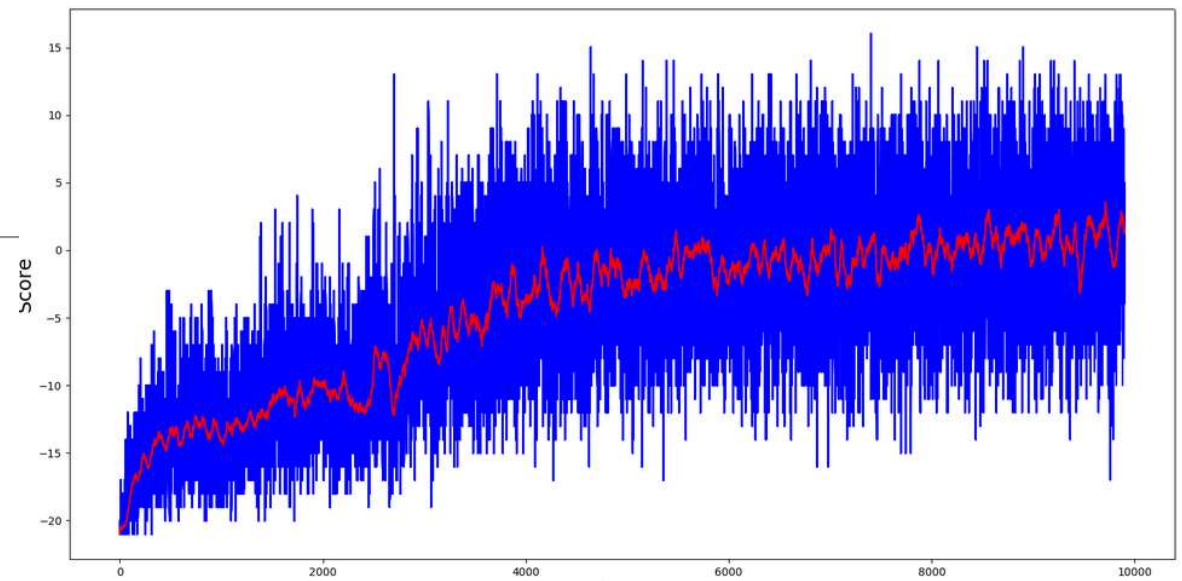
```
def replay(self):  
    # reshape memory to appropriate shape for training  
    states = np.vstack(self.states)  
    actions = np.vstack(self.actions)  
  
    # Compute discounted rewards  
    discounted_r = self.discount_rewards(self.rewards)  
  
    # Get Critic network predictions  
    values = self.Critic.predict(states)[: , 0]  
    # Compute advantages  
    advantages = discounted_r - values  
    # training Actor and Critic networks  
    self.Actor.fit(states, actions, sample_weight=advantages, epochs=1, verbose=0)  
  
    self.Critic.fit(states, discounted_r, epochs=1, verbose=0)  
    # reset training memory  
    self.states, self.actions, self.rewards = [], [], []
```

# Кривая обучения

PongDeterministic-v4

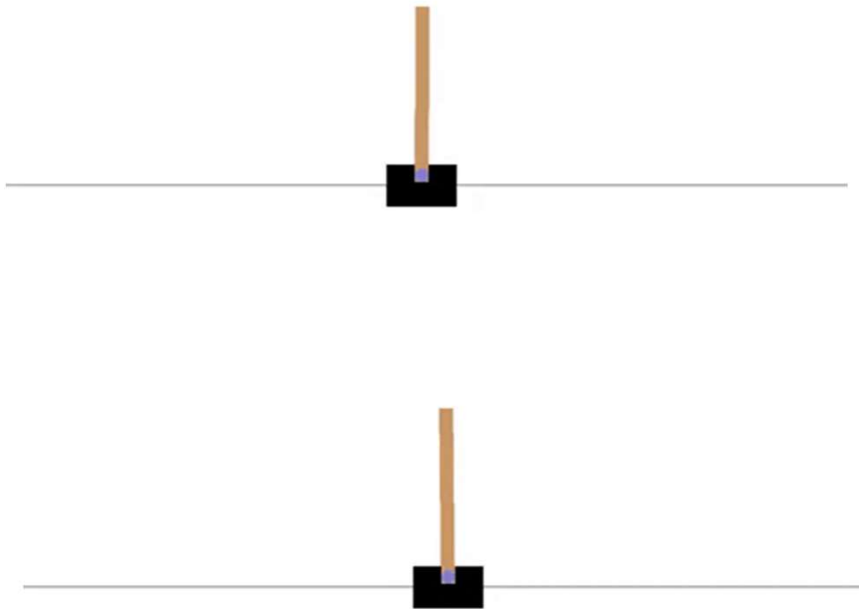


Pong-v0



# Управление тележкой

## 05\_actor\_critic\_cartpole.ipynb



```
running reward: 110.67 at episode 290  
running reward: 101.67 at episode 300  
running reward: 134.66 at episode 310  
running reward: 152.66 at episode 320  
running reward: 165.53 at episode 330  
running reward: 172.56 at episode 340  
running reward: 183.57 at episode 350  
running reward: 190.16 at episode 360  
running reward: 189.86 at episode 370  
running reward: 189.27 at episode 380  
running reward: 193.57 at episode 390  
Solved at episode 395!
```

Источник [https://keras.io/examples/rl/actor\\_critic\\_cartpole/](https://keras.io/examples/rl/actor_critic_cartpole/)

# Торговый бот

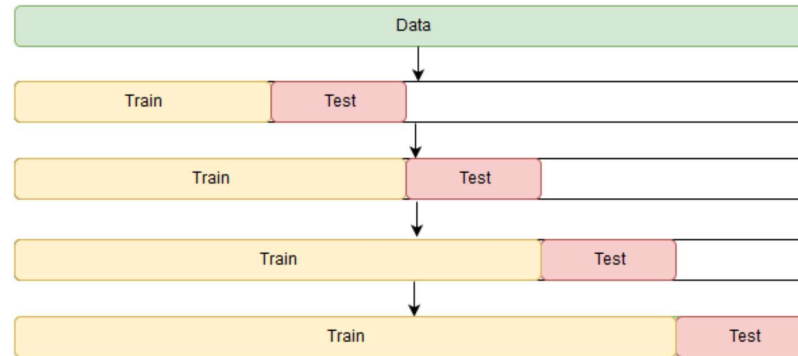
A2C

# ГРАФИК КАКОГО ТИКЕРА?

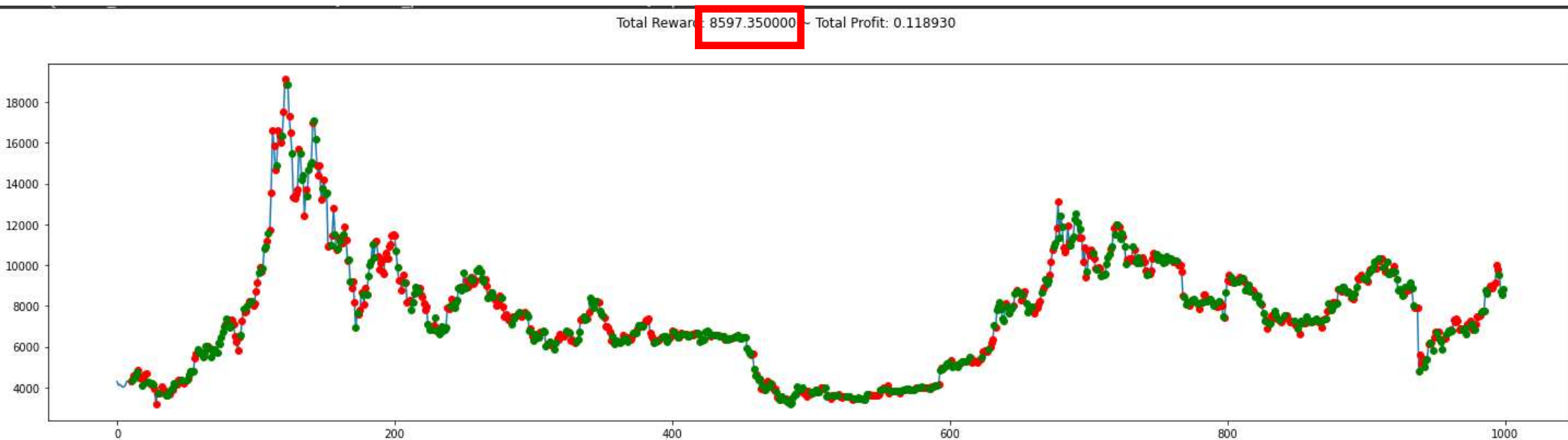
---



# Понятие окна для временного ряда



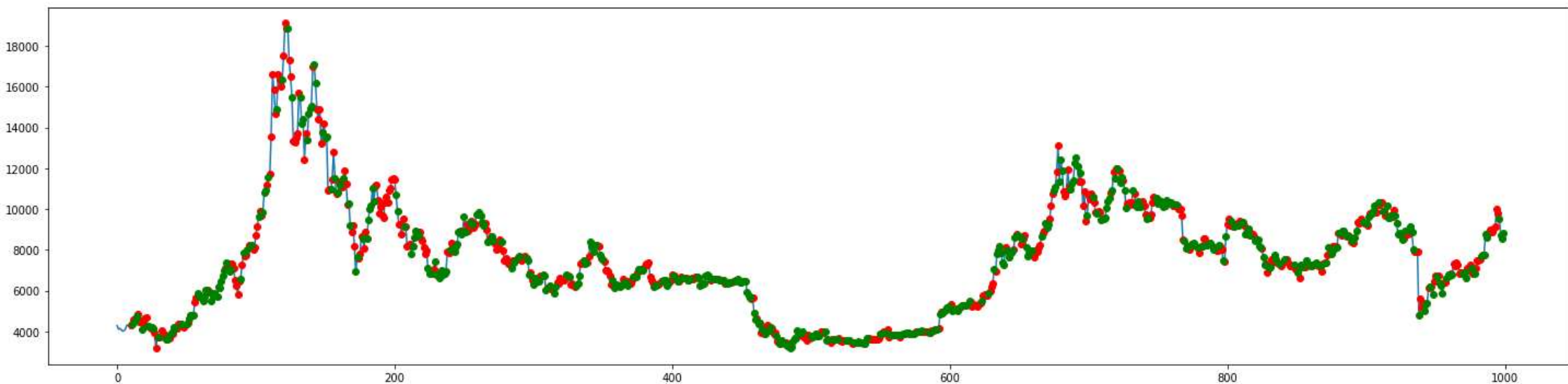
# Торговый бот: 1000 данных, окно 10



Источник <https://analyticsindiamag.com/creating-a-market-trading-bot-using-open-ai-gym-anytrading/>

# Торговый бот

Total Reward: 8597.350000 - Total Profit: 0.118930



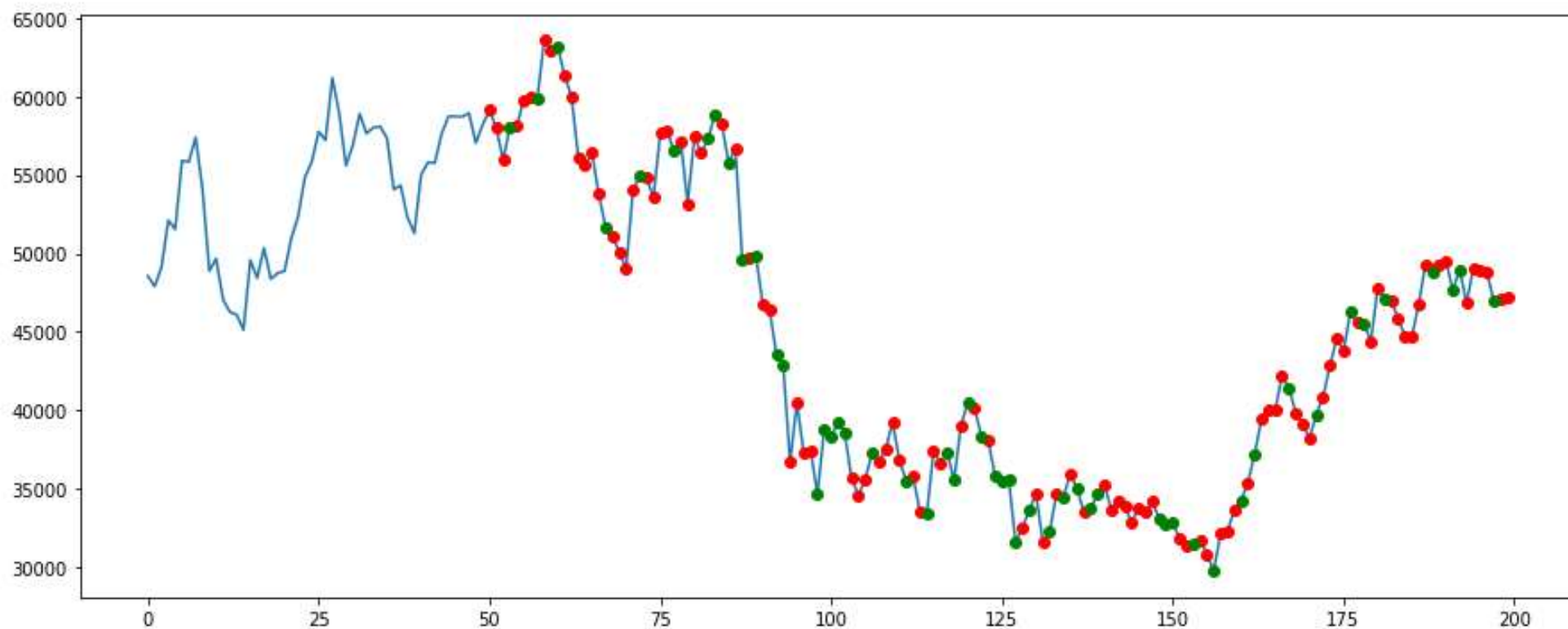
**ЭТО БЫЛИ СЛУЧАЙНЫЕ ДЕЙСТВИЯ(!)**

Источник <https://analyticsindiamag.com/creating-a-market-trading-bot-using-open-ai-gym-anytrading/>



# Результат обучения: 150 бар, окно 50

Total Reward 3075.900000 ~ Total Profit: 0.638905



# Та же модель, те же данные: 150 бар, окно 50

Total Reward: -8252.510000 ~ Total Profit: 0.481476



# Пример АЗС из научной статьи

Deep Robust Reinforcement Learning for Practical Algorithmic Trading YANG LI, WANSHAN ZHENG, ZIBIN ZHENG

## ТЕСТОВЫЙ ДАТАСЕТ Jan-2008 to Jan-2018

	APPL		IBM	
	AR	SR	AR	SR
Buy and Hold	33.48%	-	-2.75%	-
Basic DQN	41.13%	2.1	5.63%	1.3
SDAEs-LSTM DQN (ours)	66.69%	3.8	12.31%	2.1
Basic A3C	63.50%	3.6	10.02%	1.8
SDAEs-LSTM A3C (ours)	85.33%	4.3	18.93%	2.5

## ПРОВЕРОЧНЫЙ ДАТАСЕТ Jan-2008 to Jan-2018

**TABLE 3.** The performance of ma

	APPL		IBM	
	ACC	AR(3)	ACC	AR(3)
SVM	0.52	44.5%	0.50	-1.2%
DNN	0.48	41.2%	0.52	1.4%
CNN	0.52	45.2%	0.54	3.4%
LSTM	0.57	50.5%	0.60	6.7%

Что не так?

# APLE: 911% 2008-2017, 60% 2017



# КОНТАКТЫ

---

Обсуждаем

<https://t.me/devdvAI>



Репозиторий

<https://github.com/akumidv/startup-khv-ai-study>

Андрей Куминов

+7 914 770 5846

<https://facebook.com/akuminov>

<https://vk.com/akumidv>