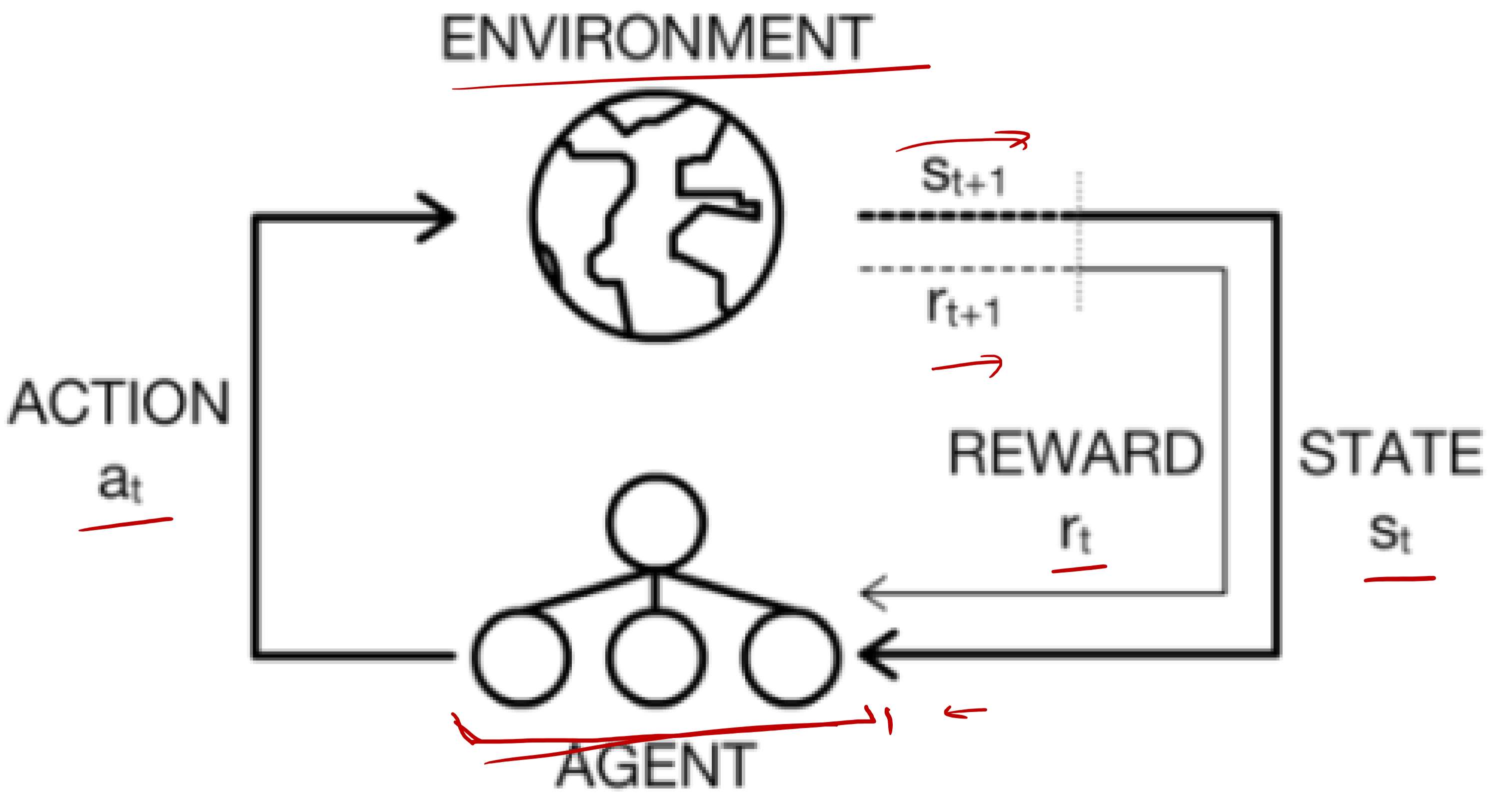
13

Обучение с подкреплением (RL)

Обучение с подкреплением Reinforcement Learning



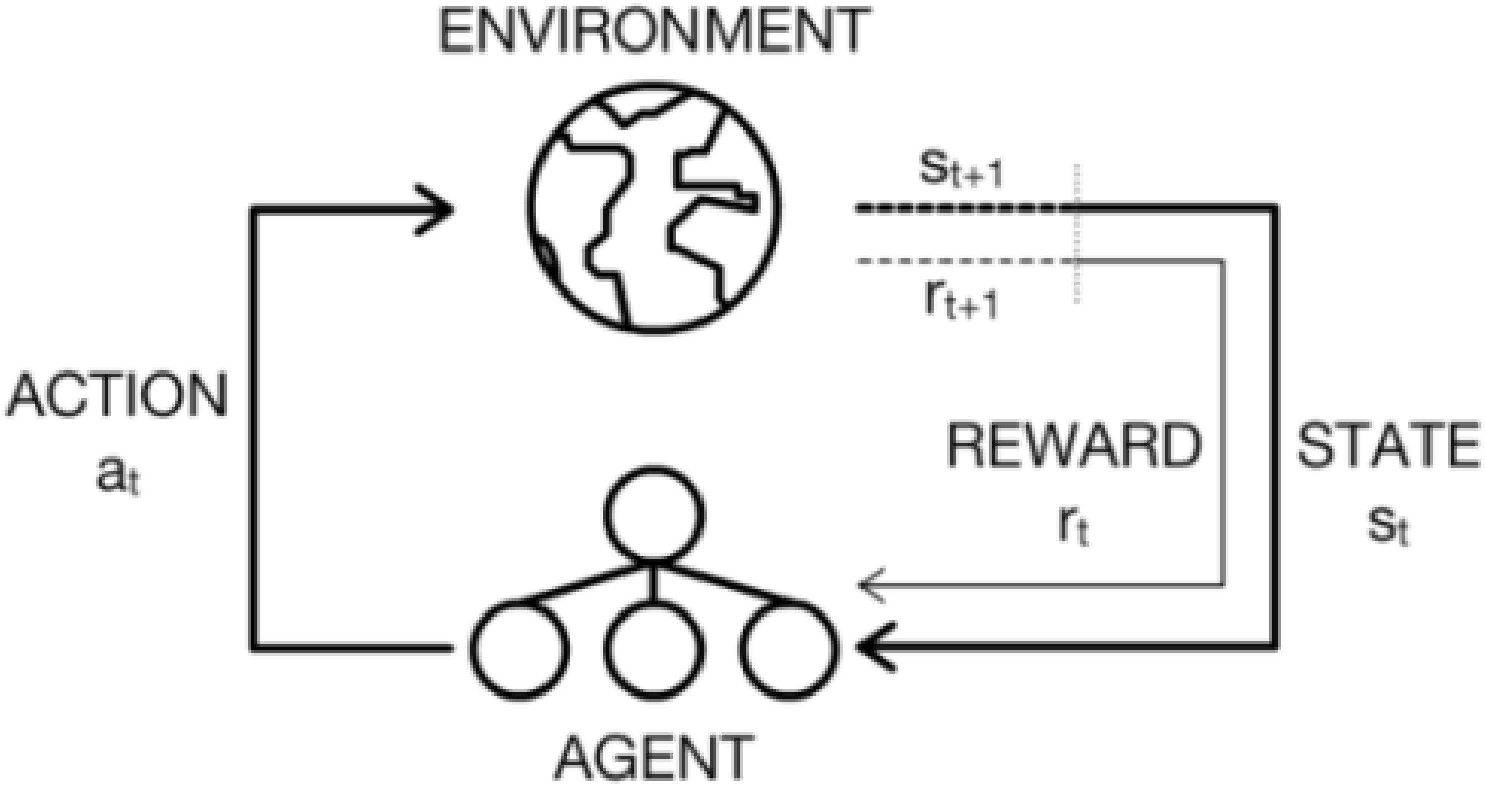
s_t – состояние среды (state) на шаге t
a_t– действие агента (action) на шаге t
r_t - награда (reward) на шаге t

Среда может:

- Быть недетерминированной
- Иметь внутреннее состояние

[Source](#)

Основные определения



s_t – состояние среды (state) на шаге t
a_t– действие агента (action) на шаге t
r_t - награда (reward) на шаге t

$\boldsymbol{p(a|s)}$ **- policy function**

$\boldsymbol{v(s)}$ **- value function**

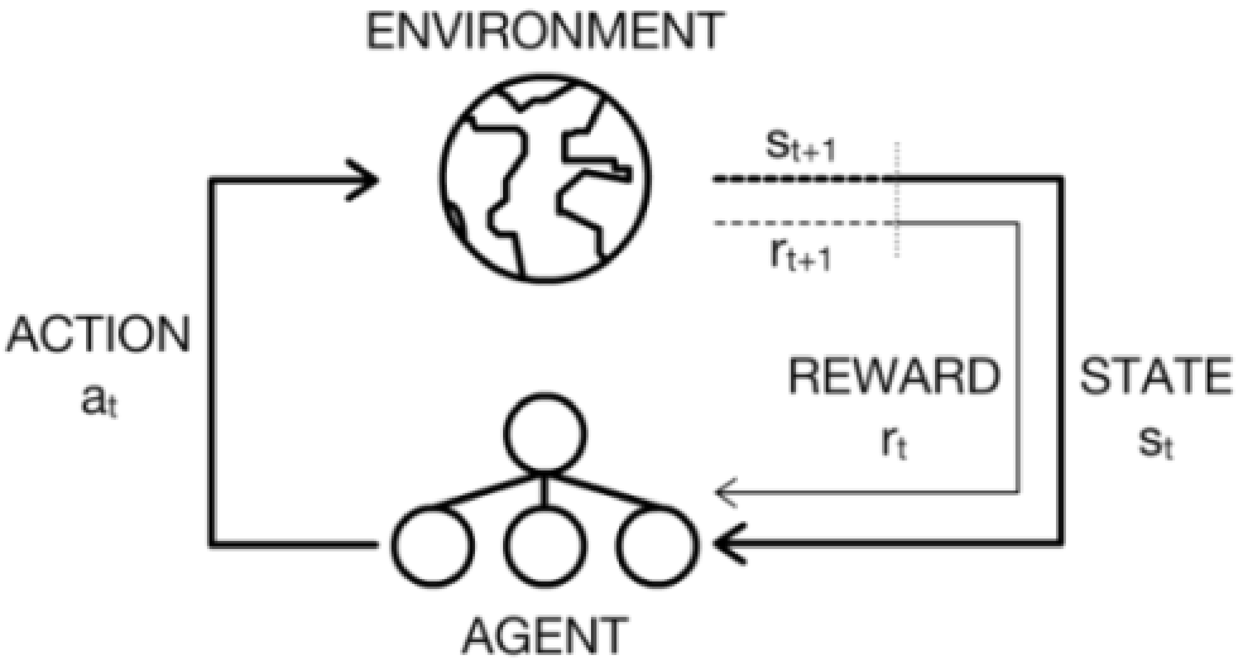
$Q(s,a)$ - Q-function

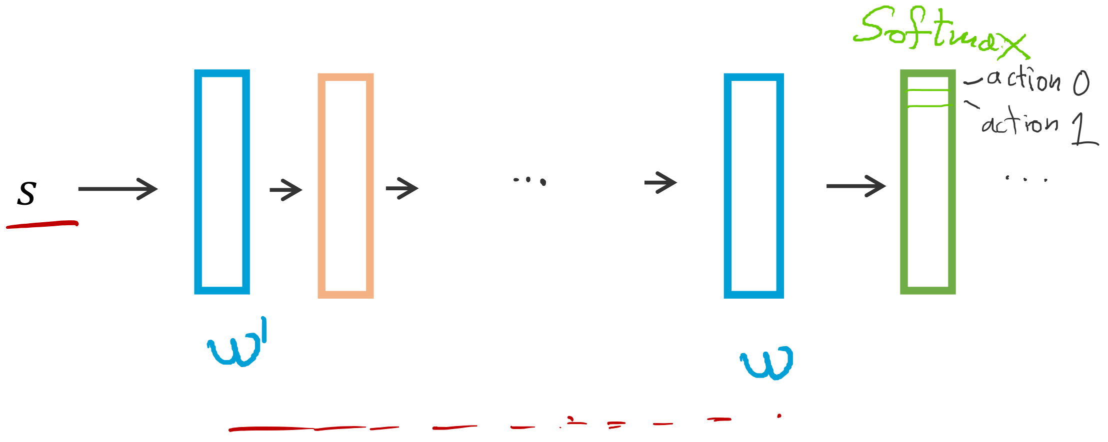
Deep Reinforcement Learning

$p(a|s)$ - policy function

$v(s)$ - value function

$Q(s,a)$ - Q-function



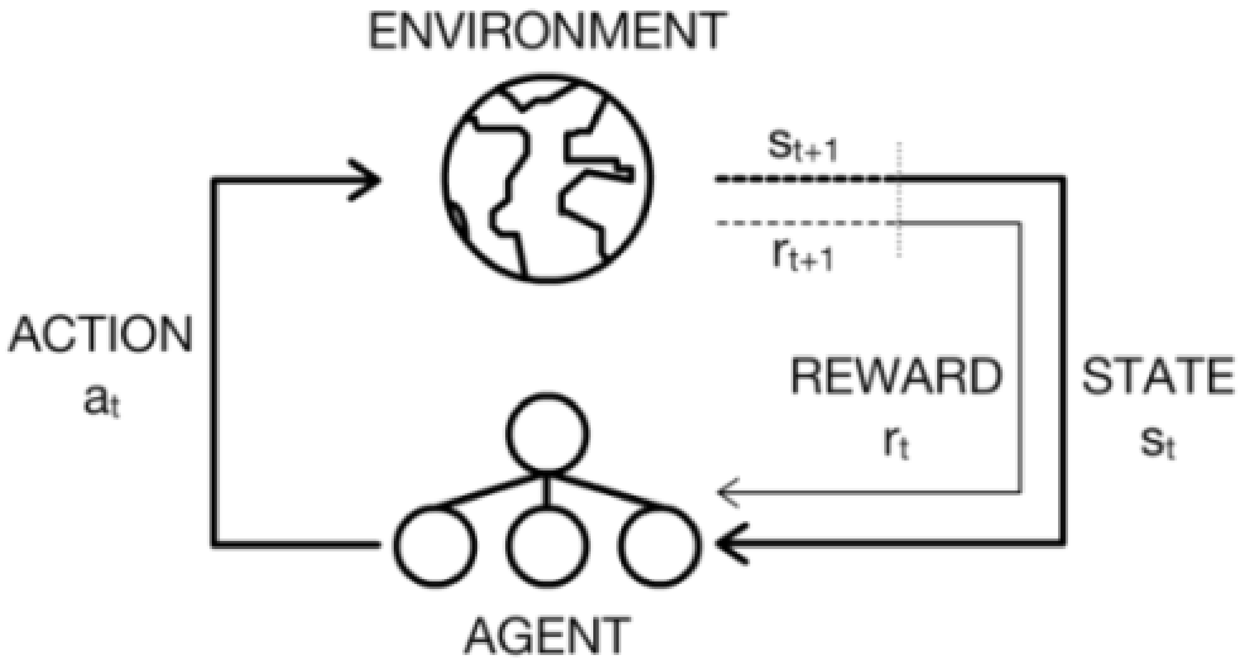


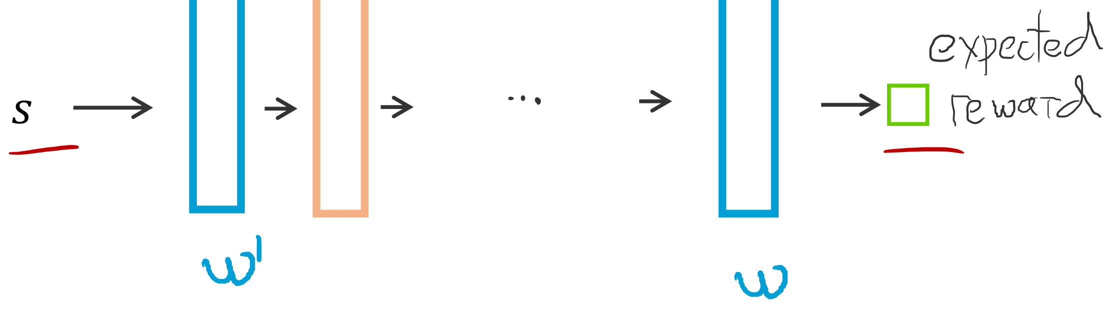
Deep Reinforcement Learning

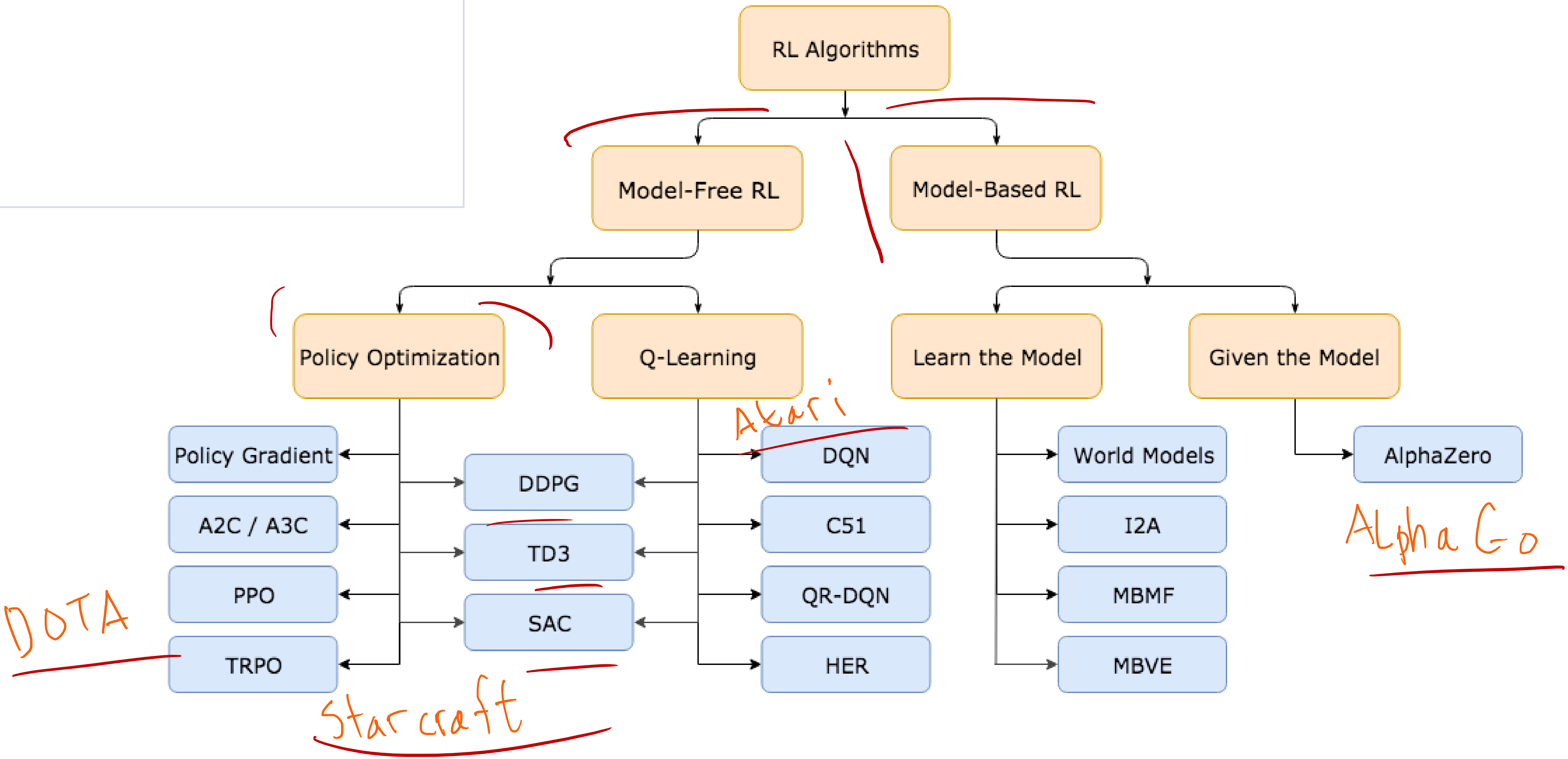
$p(a|s)$ - policy function

$v(s)$ - value function

$Q(s, a)$ - Q-function

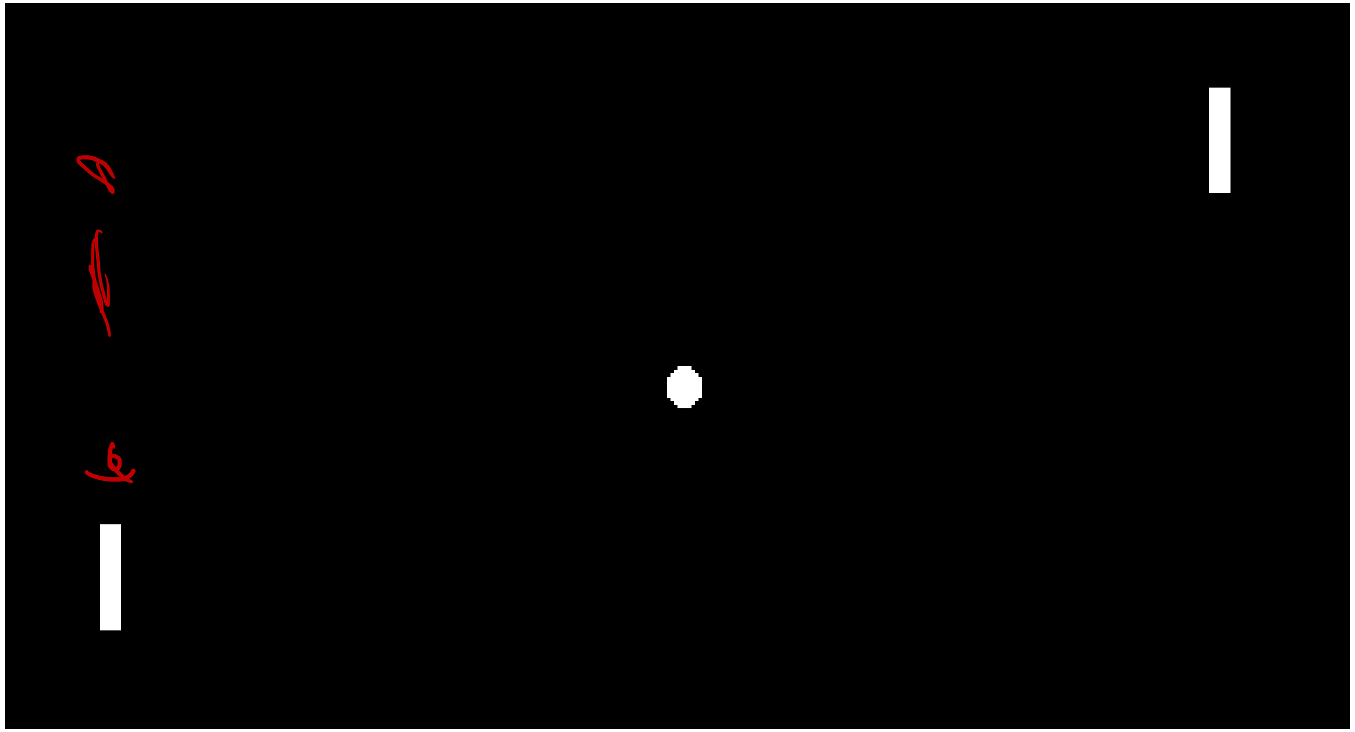






OpenAI Spinning Up in RL

Policy Gradients на примере

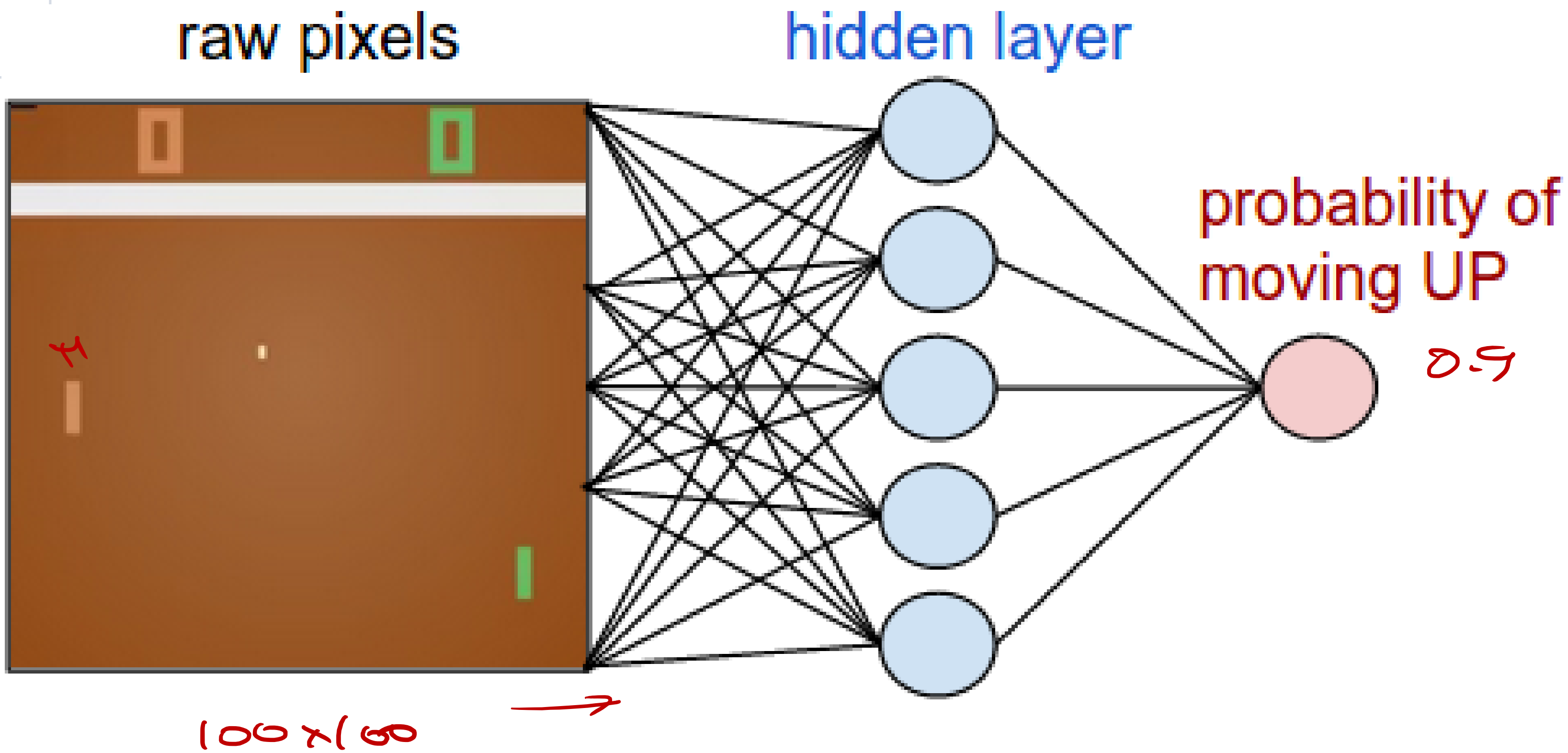


http://karpathy.github.io/2016/05/31/rl/

Например!

100

+1
-1

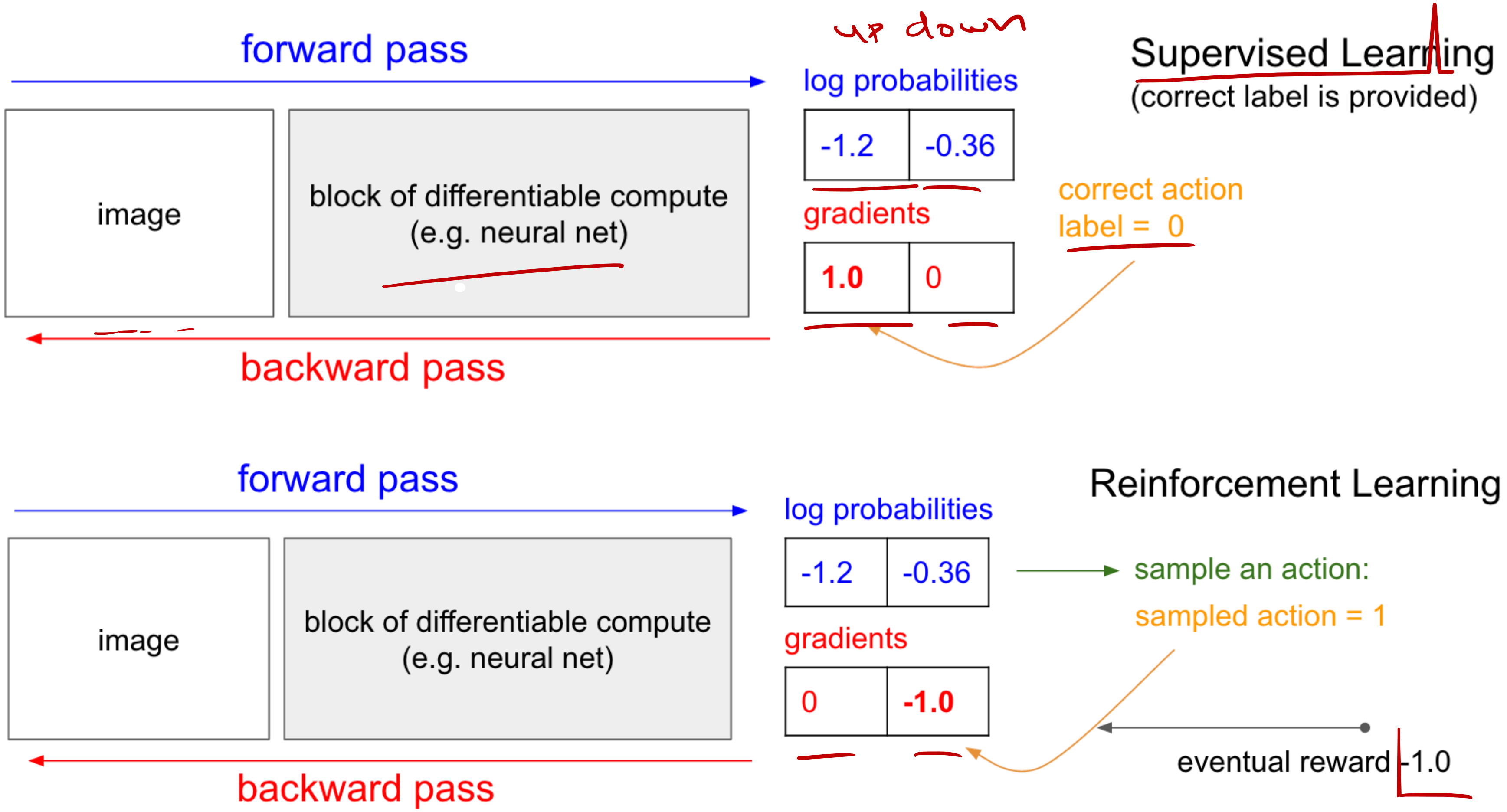


Необходимый трюк: передавать нейросети разницу между текущим и прошлым кадром

http://karpathy.github.io/2016/05/31/rl/

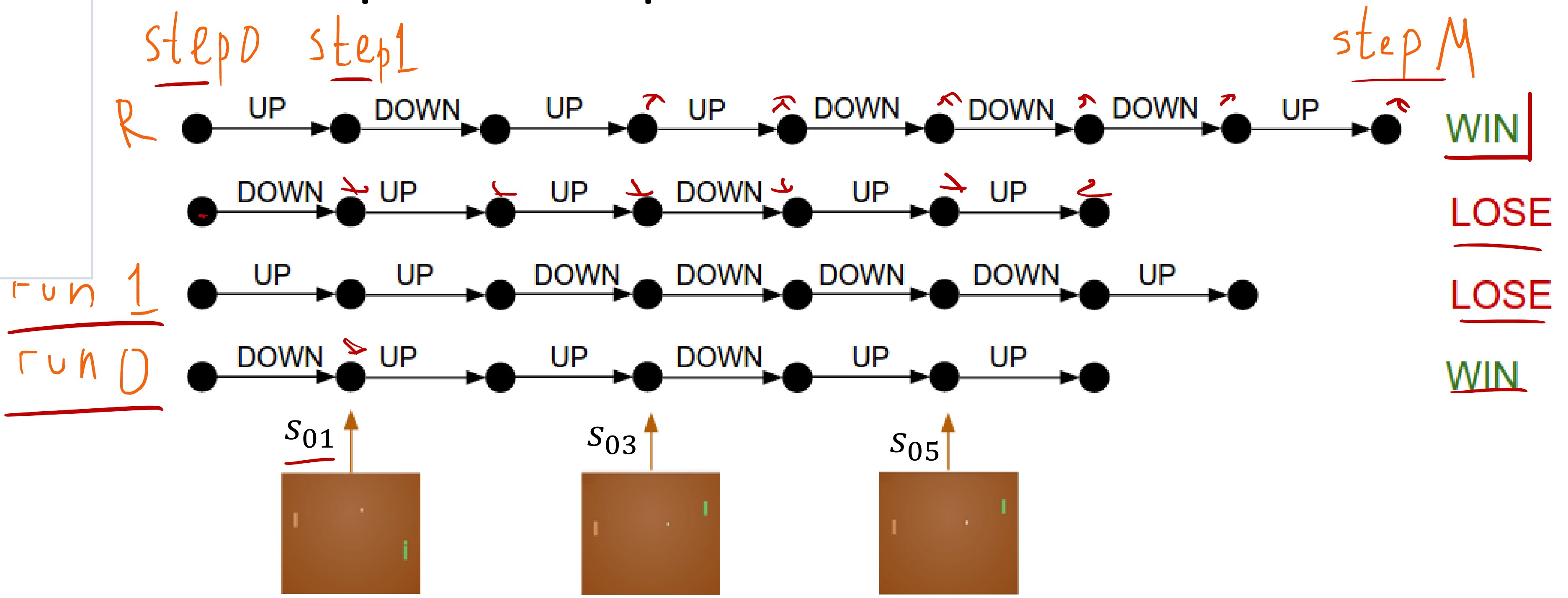
Policy Gradients

Σ ln p xi



http://karpathy.github.io/2016/05/31/rl/

Набираем прогоны rollout



X Y

s_{00}	a_{00}	r_{00}
s_{01}	a_{01}	r_{01}

…

s_{RM}	a_{RM}	r_{RM}

s_{ij} - состояние на шаге j прогона i

a_{ij} - действие на шаге j прогона i

r_{ij} - награда после шага j прогона i

В нашем (простом) случае награды на всех шагах прогона одинаковые $r_{ij} = r_i$

http://karpathy.github.io/2016/05/31/rl/

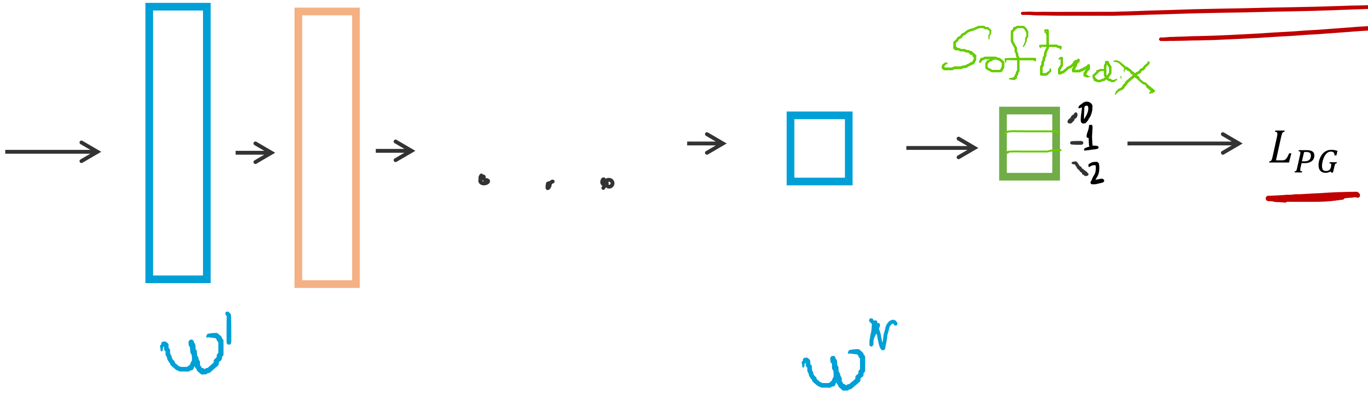
Тренируем модель

$y = 1$

$$L_{CE} = -\sum_i \ln p(c = gt_i|x_i)$$

$$L_{PG} = -\sum_i r_i \ln p(c = a_i|x_i)$$

X	Y	
s_0	a_0	r_0
s_1	a_1	r_1
…		
s_N	a_N	r_N



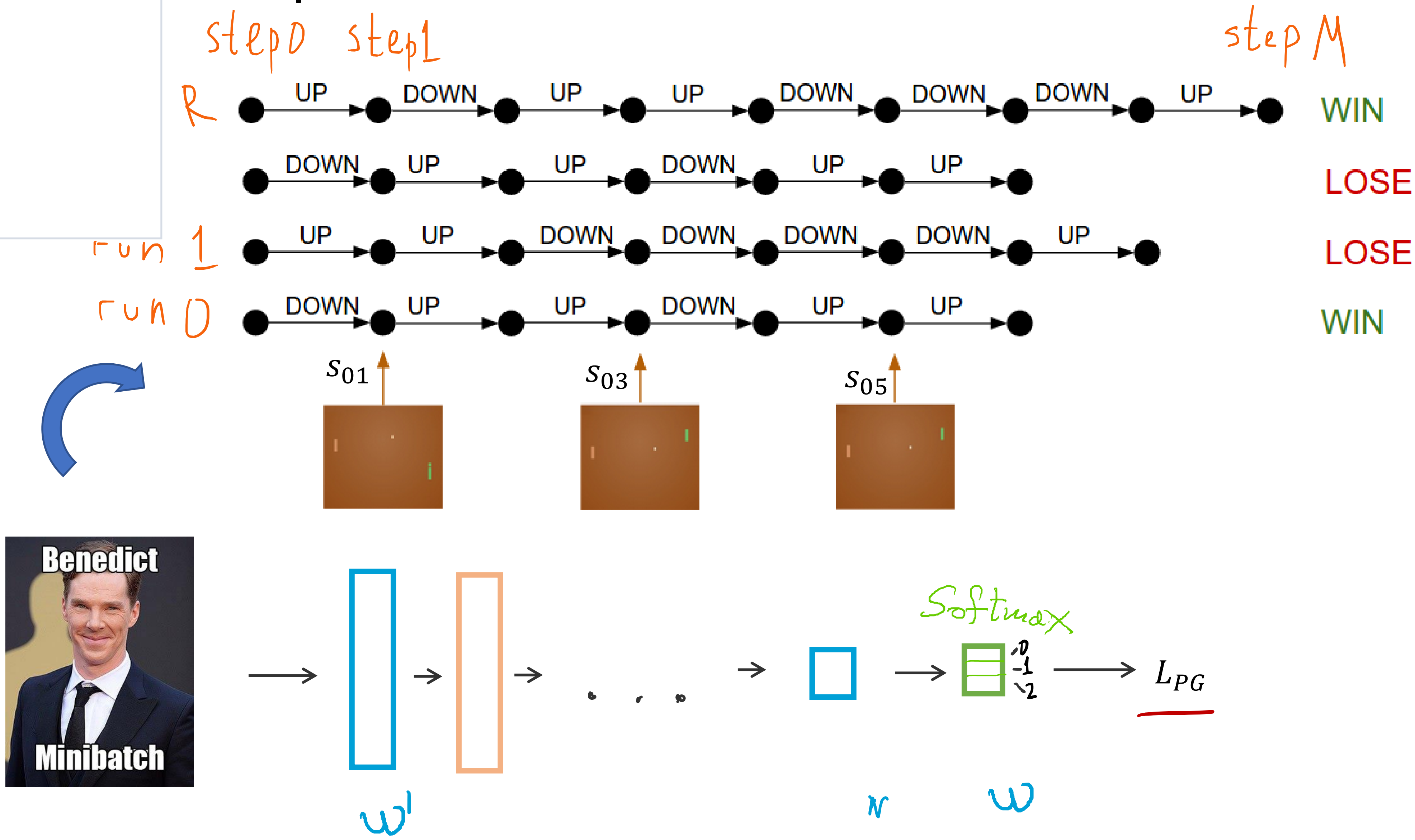
Проходим по всем данным один раз и…

Repeat

X	Y	
s_{00}	a_{00}	r_{00}
s_{01}	a_{01}	r_{01}

...

s_{RM}	a_{RM}	r_{RM}




I KNOW
REINFORCEMENT LEARNING
memegenerator.net

Вывод на пальцах

веса сети

$p(x|s,\theta)$ - policy function

$f(x)$ - reward from environment

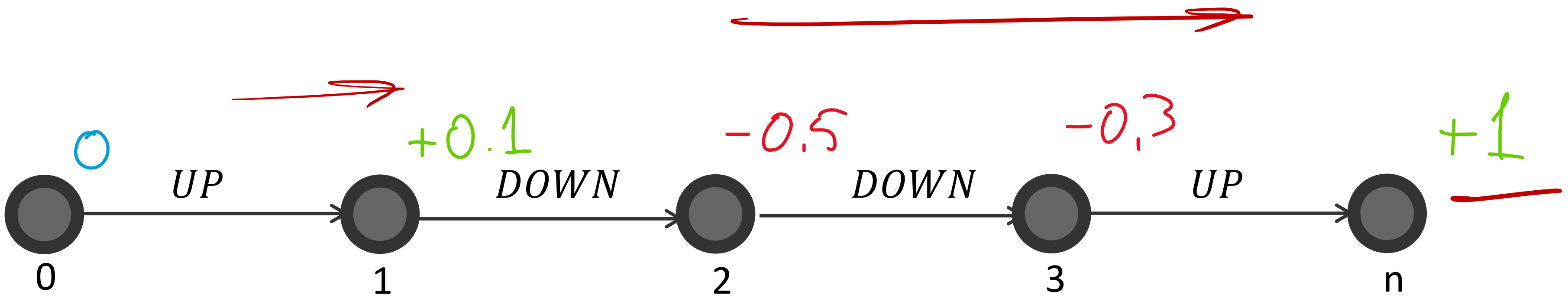
Хотим двигать θ так, чтобы увеличивать $E[f(x)]$

$$
\begin{aligned}
\nabla_\theta E_x[f(x)] &= \nabla_\theta \sum_x p(x) f(x) && \text{definition of expectation} \\
&= \sum_x \nabla_\theta p(x) f(x) && \text{swap sum and gradient} \\
&= \sum_x p(x) \boxed{\frac{\nabla_\theta p(x)}{p(x)}} f(x) && \text{both multiply and divide by } p(x) \\
&= \sum_x p(x) \nabla_\theta \log p(x) f(x) && \text{use the fact that } \nabla_\theta \log(z) = \frac{1}{z} \nabla_\theta z \\
&= E_x[f(x) \nabla_\theta \log p(x)] && \text{definition of expectation}
\end{aligned}
$$

$\sum_i r_i \log p(s_i)$

http://karpathy.github.io/2016/05/31/rl/

Discounted rewards



$R_0 = r_0 + r_1 + \cdots + r_n$

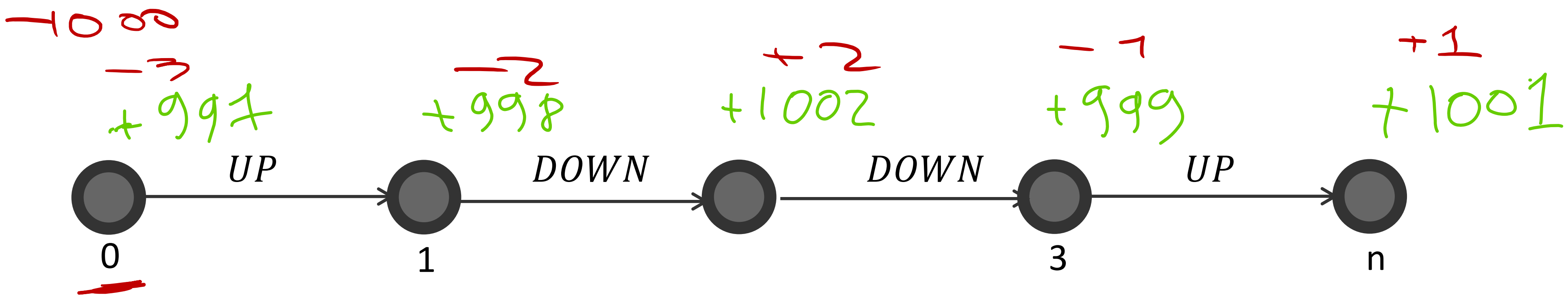
$R_2 = r_2 + r_3 + r_4 + \cdots + r_n$

$R_0 = r_0 + \gamma\, r_1 + \gamma^2\, r_2 + \cdots + \gamma^n r_n$

$R_2 = r_2 + \gamma\, r_3 + \gamma^2\, r_3 + \cdots + \gamma^{n-2} r_n$

$\gamma < 1$

Baseline



$$R_s = \sum_{i=0 \ldots n-s} \gamma^i r_{s+i}$$

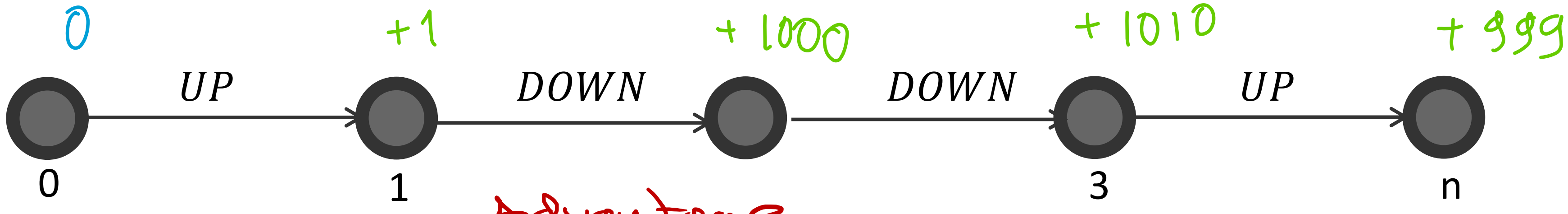
$\gamma < 1$

$$R_s = \sum_{i=0 \ldots n-s} \gamma^i r_{s+i} - b$$

$$b = E_{rollouts}\left[\sum_i \gamma^i r_{s+i}\right]$$

REINFORCE '92

Actor-Critic

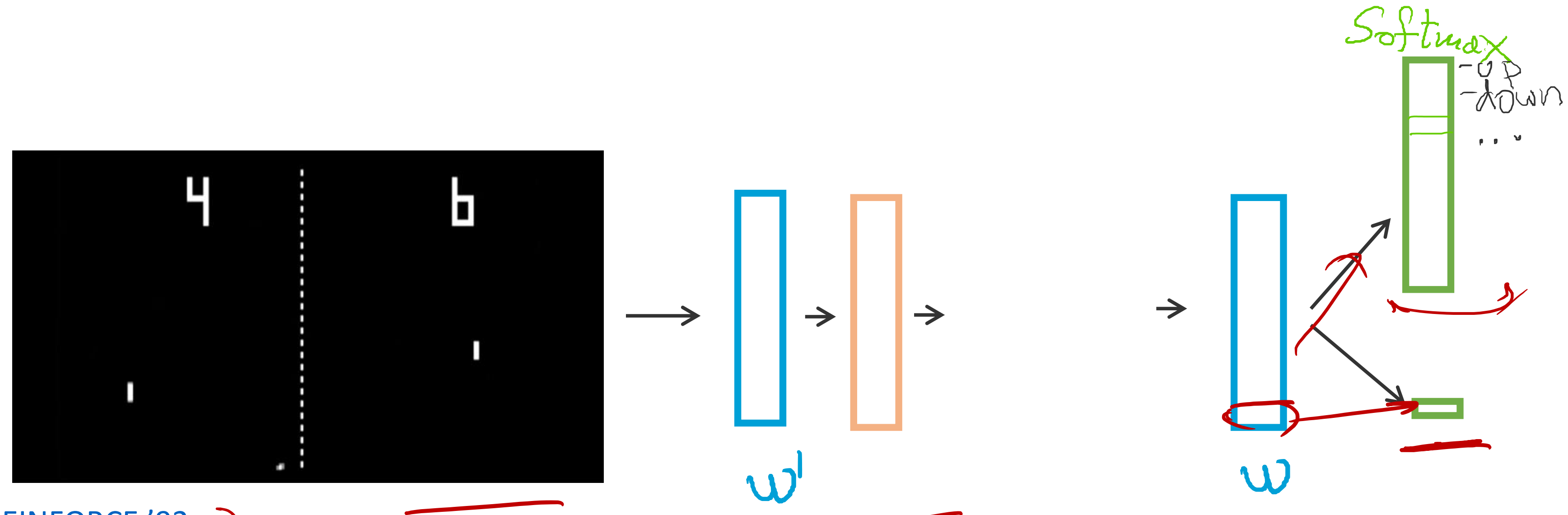


Advantage

$$R_s = \sum_{i=0 \ldots n-s} \gamma^i r_{s+i} - b$$

$$b = \mathbb{E}_{rollouts}\left[\sum_i \gamma^i r_{s+i}\right]$$

$V(state)$

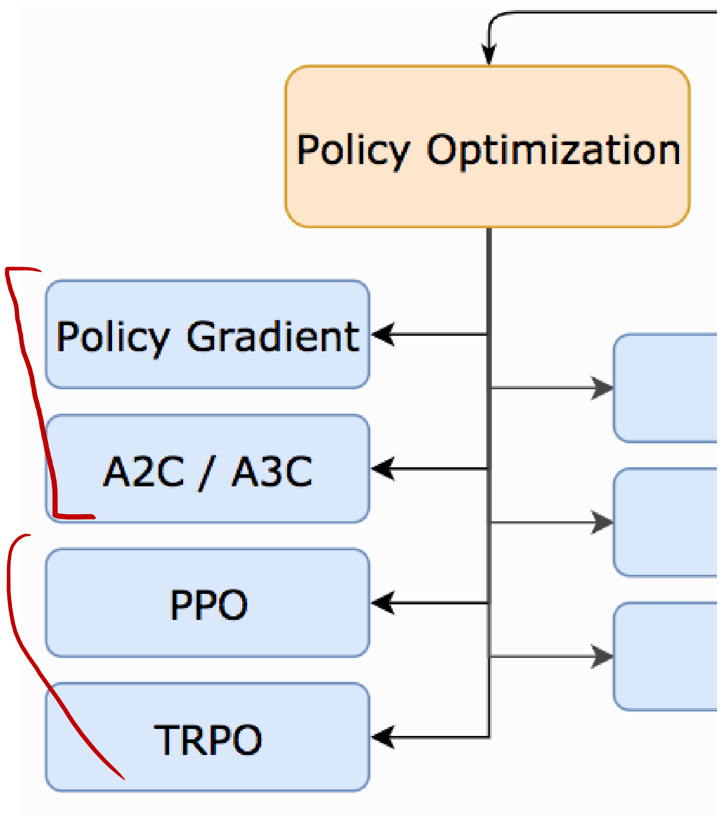


$$\log p(a) \sum_i \left(\gamma^i r_{s+i} - V\right)$$

$$L_2\left(V - \sum_i \gamma^i r_{s+i}\right)$$

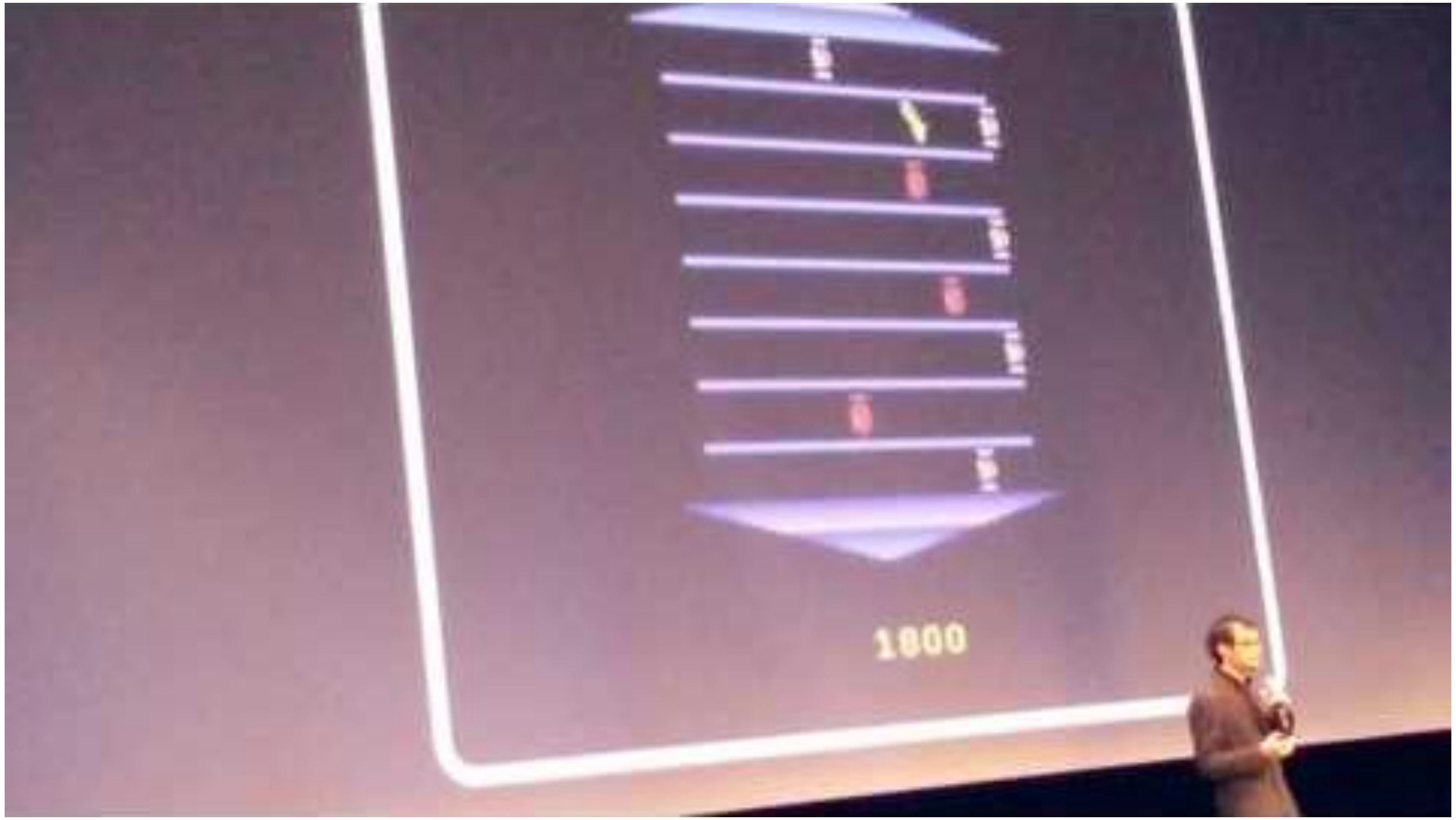
REINFORCE '92

Прогресс продолжается



[OpenAI Spinning Up in RL]()

Deep Q Learning (DQN)

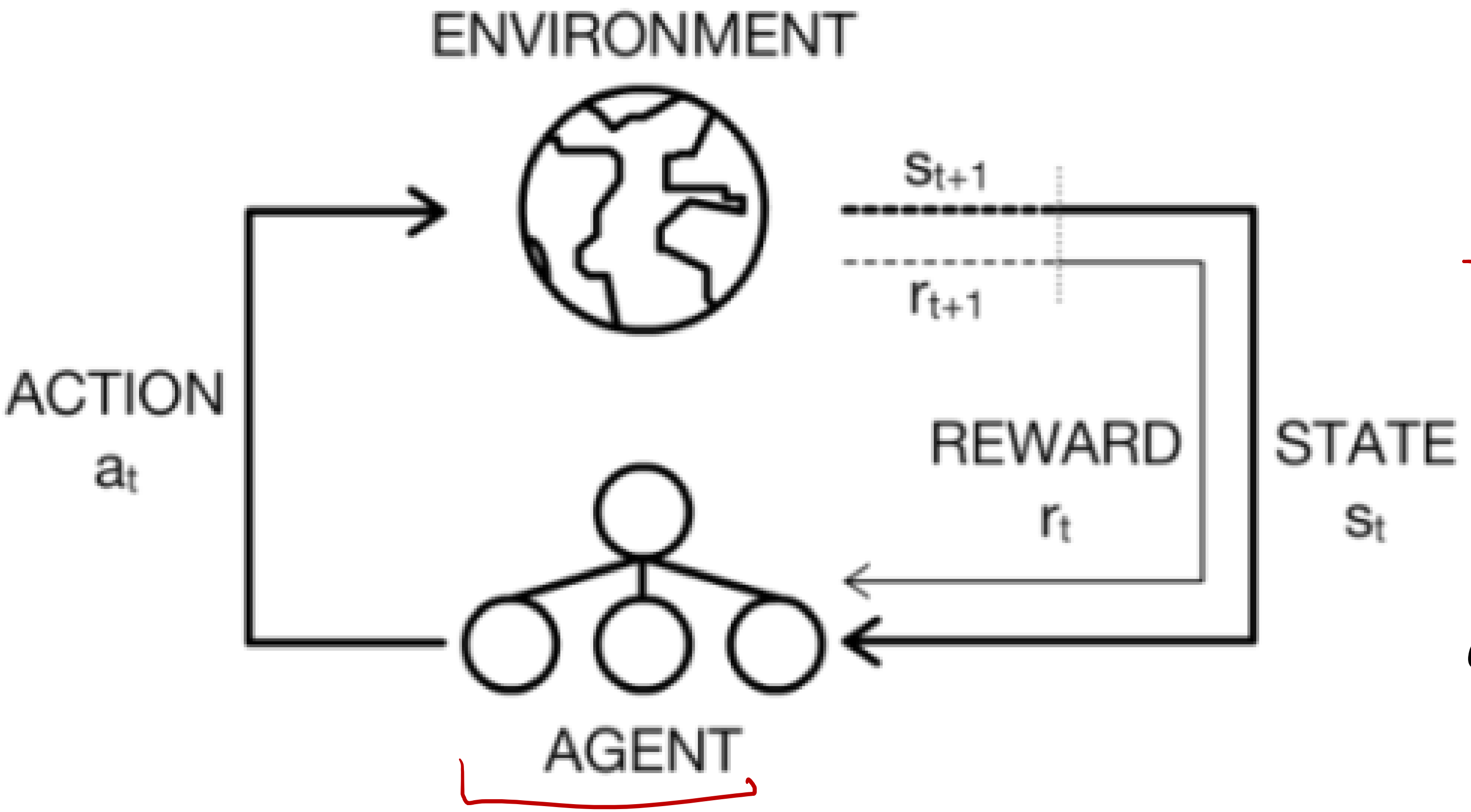


[Google Acquires Artificial Intelligence Startup DeepMind For More Than $500M]


Так. Приготовьтесь!
Сейчас будет сложно.
memesmix.net

Q-function

$Q(s,a) \to r$



$p(a|s)$ - policy function

$v(s)$ - value function

$\boldsymbol{Q(s,a)}$ **- Q-function**

$Q^*(s,a)$ – Q-функция оптимальной стратегии

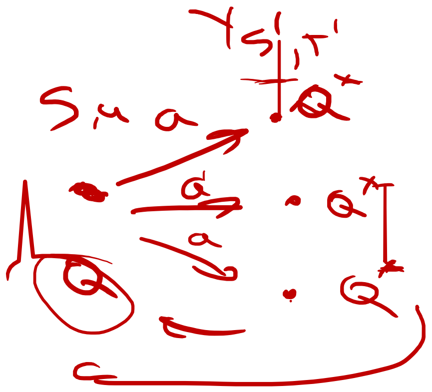
Bellman equation

$\gamma < 1$

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

$Q^*(s,a)$ – Q-функция оптимальной стратегии

$\mathcal{E}$ – пространство возможных следующих состояний



Итеративное приближение:

$$Q_{i+1}(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q_i(s',a') | s,a\right]$$

Приближаем Q^* нейросетью с параметрами θ:

$$Q(s,a;\theta) \approx Q^*(s,a)$$

Формулируем функцию ошибки как приближние Q^* :

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)}\left[(y_i - Q(s,a;\theta_i))^2\right]$$

$$y_i = \mathbb{E}_{s' \sim \mathcal{E}}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) | s,a\right]$$

[Playing Atari with Deep Reinforcement Learning’13]

Алгоритм DQN

Algorithm 1 Deep Q-learning with Experience Replay

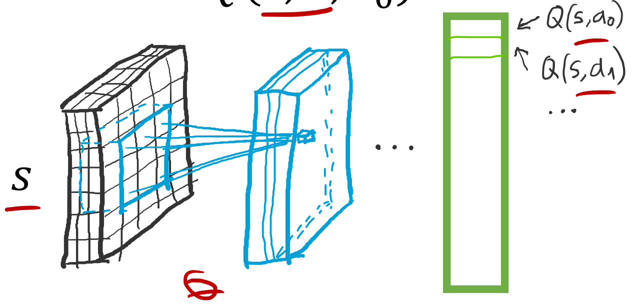
Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
    Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
    for $t = 1, T$ **do**
        With probability ϵ select a random action a_t
        otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
        Execute action a_t in emulator and observe reward r_t and image x_{t+1}
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
        Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
    end for
end for

Playing Atari with Deep Reinforcement Learning’13

Алгоритм DQN

D

$Q(s, a, \theta_0)$



[Playing Atari with Deep Reinforcement Learning'13]

Algorithm 1 Deep Q-learning with Experience Replay

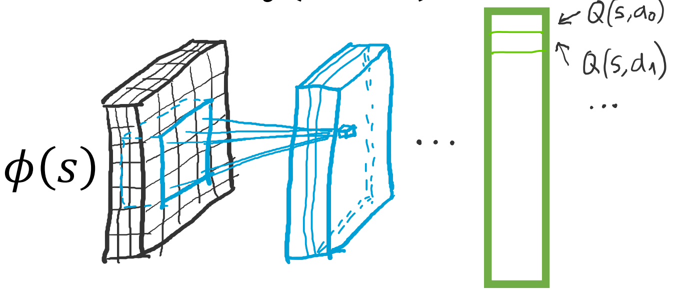
Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
    Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
    for $t = 1, T$ **do**
        With probability ϵ select a random action a_t
        otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
        Execute action a_t in emulator and observe reward r_t and image x_{t+1}
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
        Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
    end for
end for

Алгоритм DQN

D

ϕ_1

$Q(s, a, \theta)$



Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\underline{\phi_1 = \phi(s_1)}$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

[Playing Atari with Deep Reinforcement Learning’13]

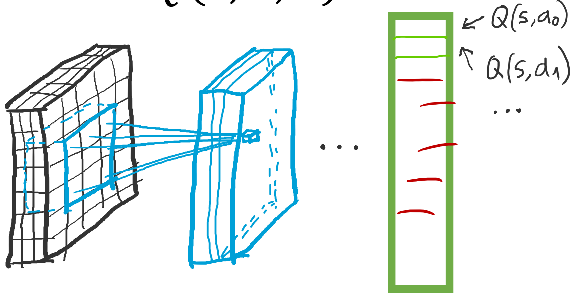
Алгоритм DQN

D

$\phi_1 \; a_1$

$Q(s, a, \theta)$

$\phi(s)$



Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Playing Atari with Deep Reinforcement Learning'13

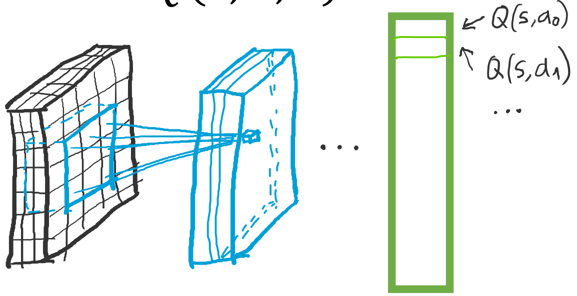
Алгоритм DQN

D

$\phi_1 \; a_1 \; r_1 \; \phi_2$

$Q(s, a, \theta)$

$\phi(s)$



Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

[Playing Atari with Deep Reinforcement Learning'13]

Алгоритм DQN

D

$\phi_1\ a_1\ r_1\ \phi_2$

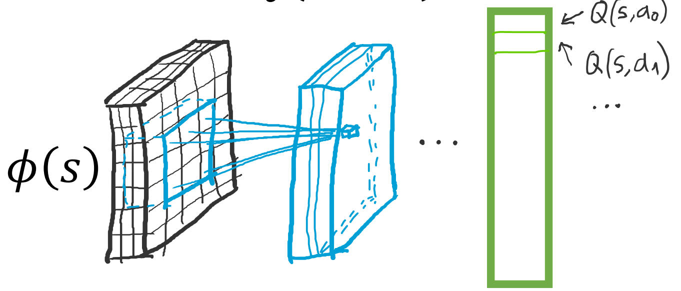
$\phi_2\ a_2\ r_2\ \phi_3$

$\dots$

$\phi_n\ a_n\ r_n\ \phi_{n+1}$

$\phi_1\ a_1\ r_1\ \phi_2$

$Q(s,a,\theta)$

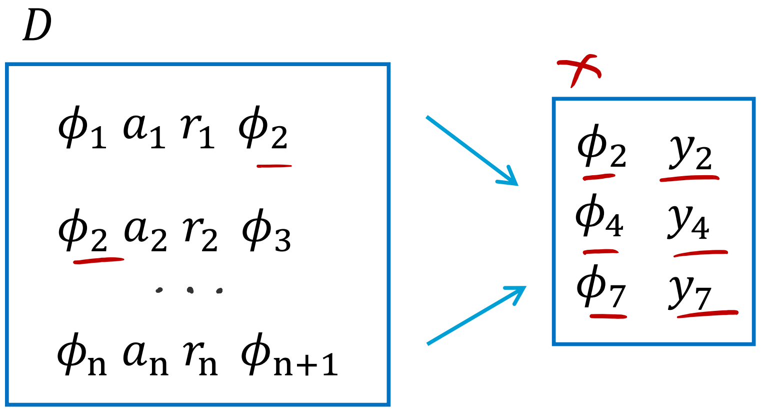


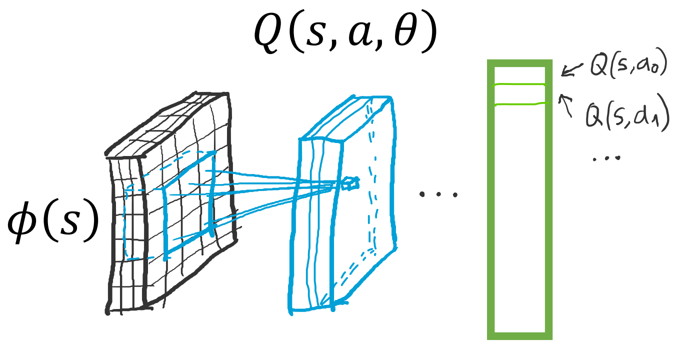
Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

[Playing Atari with Deep Reinforcement Learning'13]

Алгоритм DQN



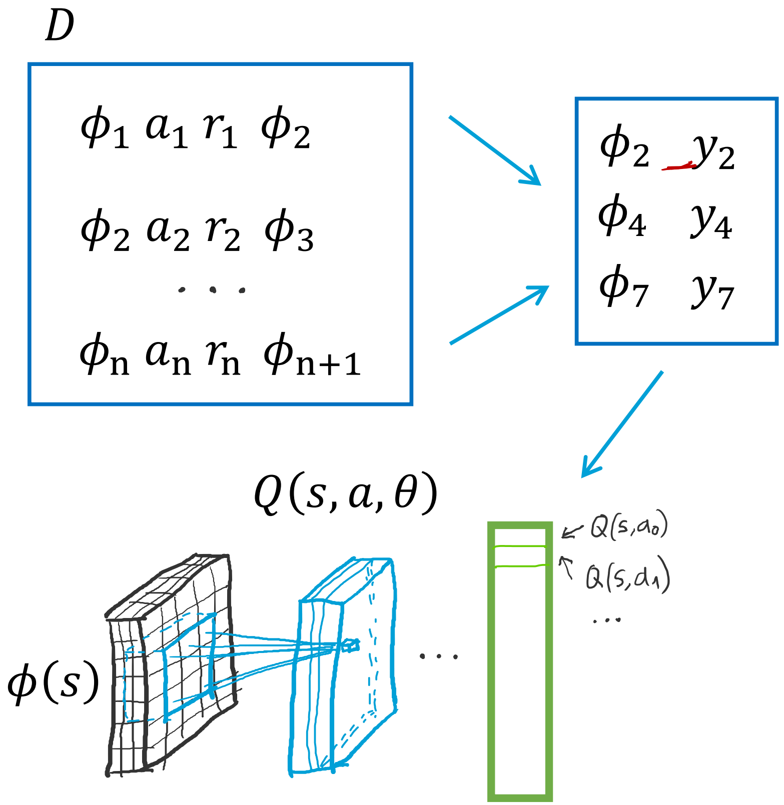


Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Playing Atari with Deep Reinforcement Learning'13

Алгоритм DQN



Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $\underline{(y_j - Q(\phi_j, a_j; \theta))^2}$ according to equation 3
  end for
end for

[Playing Atari with Deep Reinforcement Learning'13]

Алгоритм DQN

D

$\phi_1\ a_1\ r_1\ \phi_2$

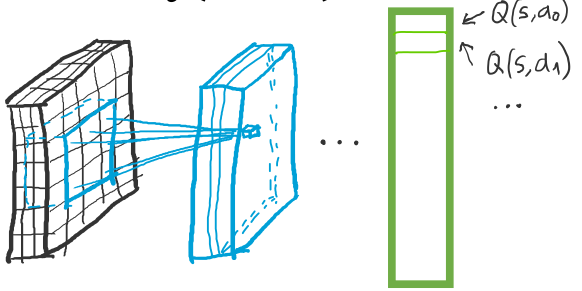
$\phi_2\ a_2\ r_2\ \phi_3$

$\dots$

$\phi_n\ a_n\ r_n\ \phi_{n+1}$

$Q(s, a, \theta)$

$\phi(s)$

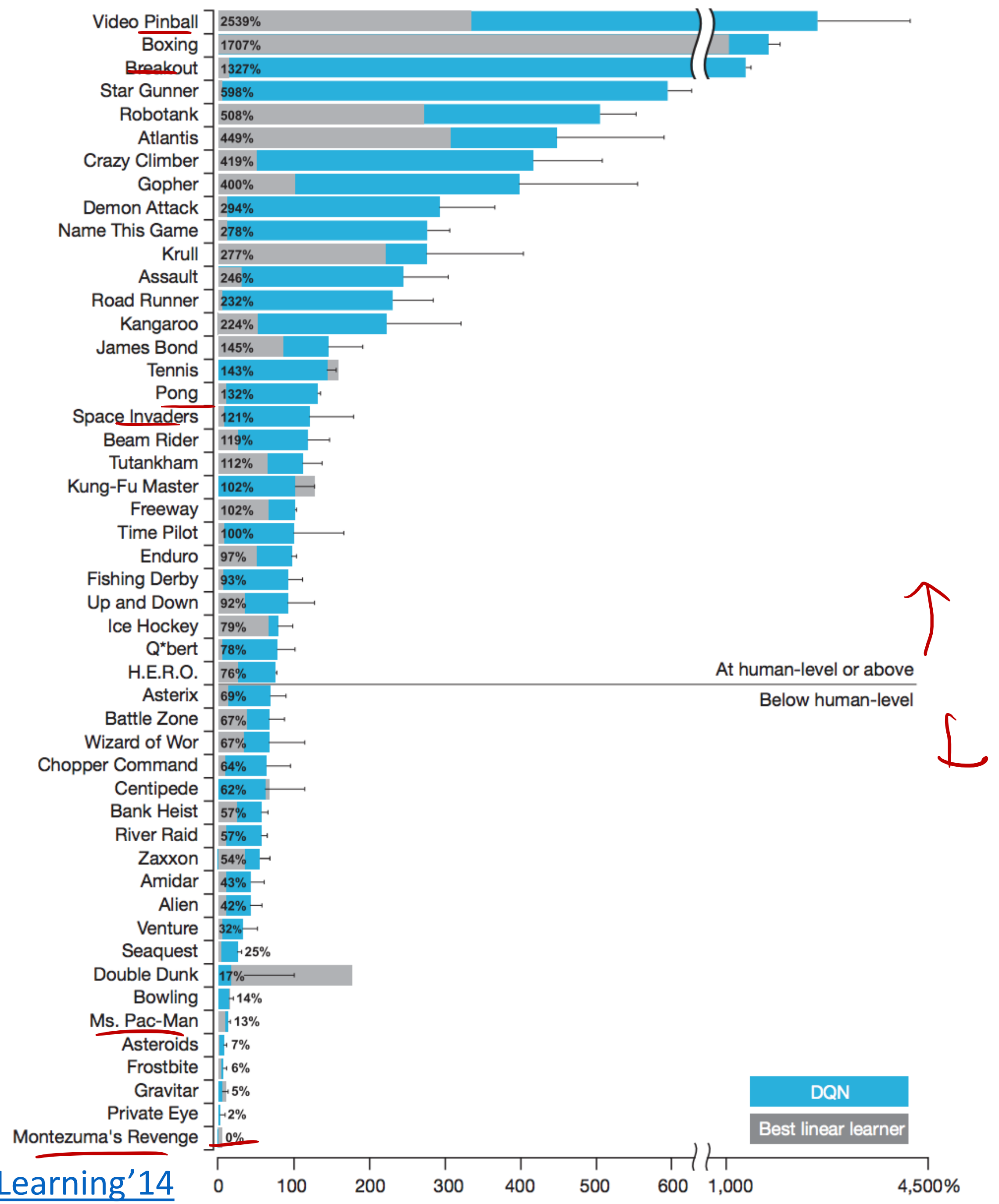


Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

[Playing Atari with Deep Reinforcement Learning'13]

Also, Habr

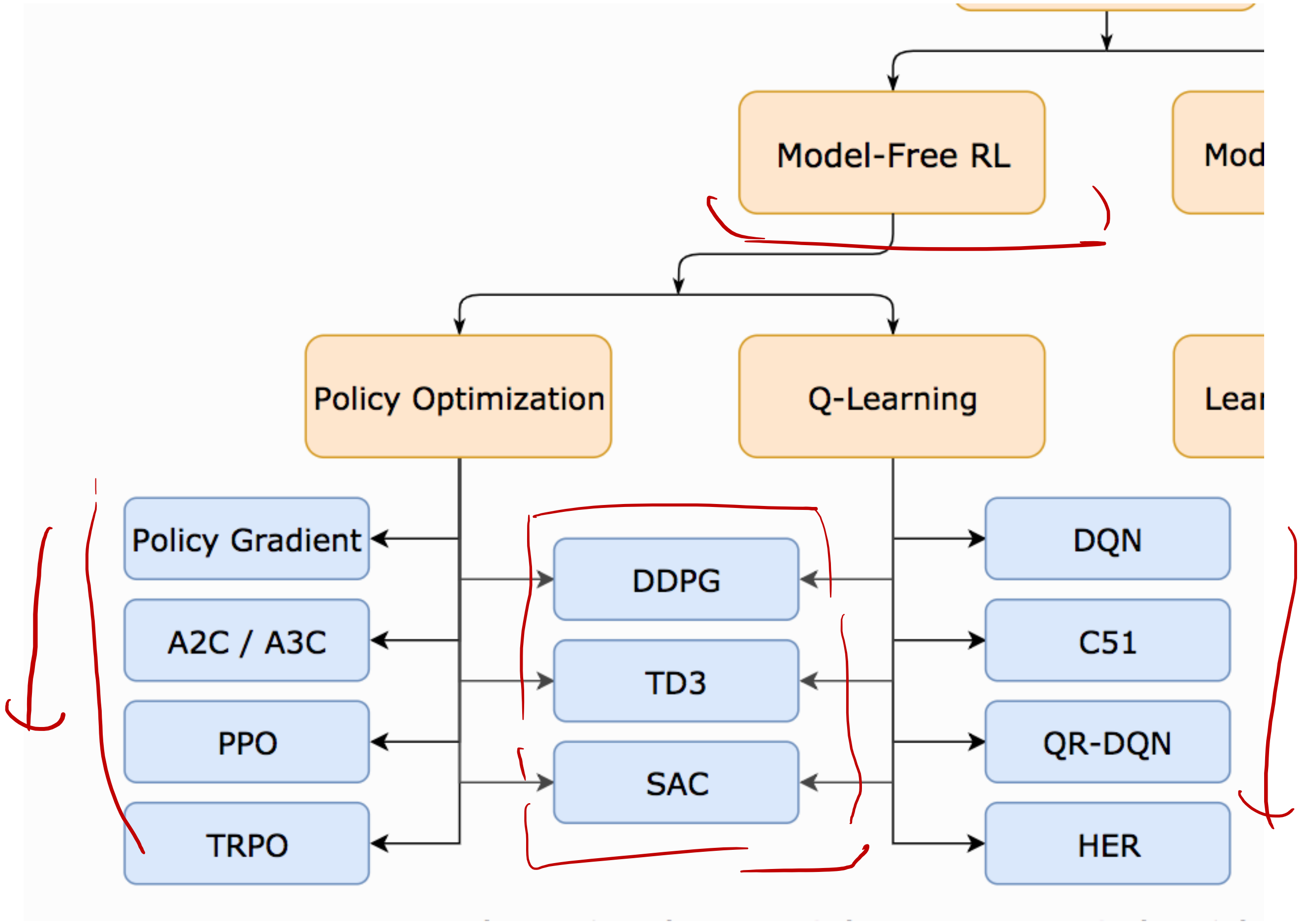


[Human-level control through Deep Reinforcement Learning'14]

Q-learning vs policy gradients

Policy Gradients	**Q-learning**
On-policy	Off-policy
Эффективнее исследует	Эффективнее сэмплирует
Проще	Сложнее
Чаще работает	Когда работает, сходится стабильнее

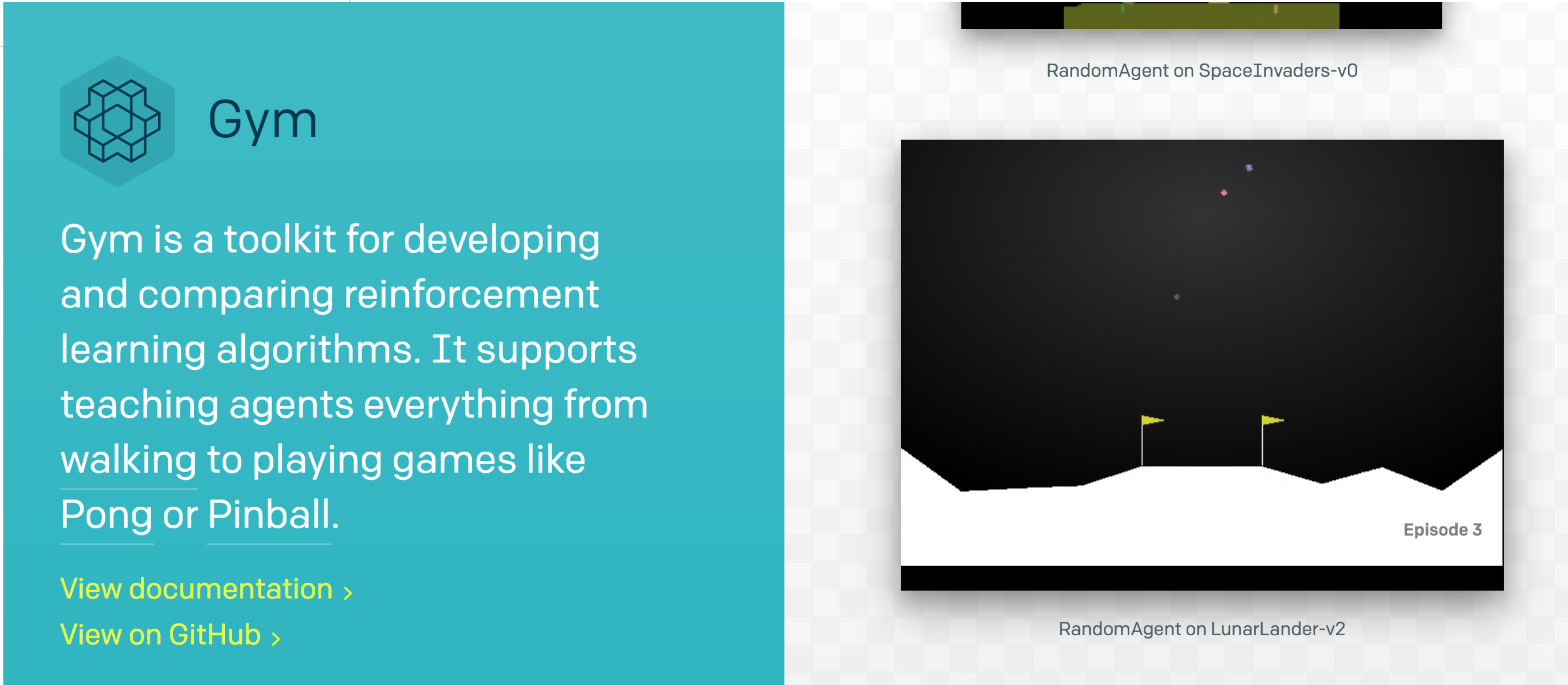
Дальнейшая работа



OpenAI Spinning Up in RL


WHEN YOU WAKE UP
EXPECTATION
REALITY

Как с этим поиграться?



[OpenAI Gym]

Создаем environment

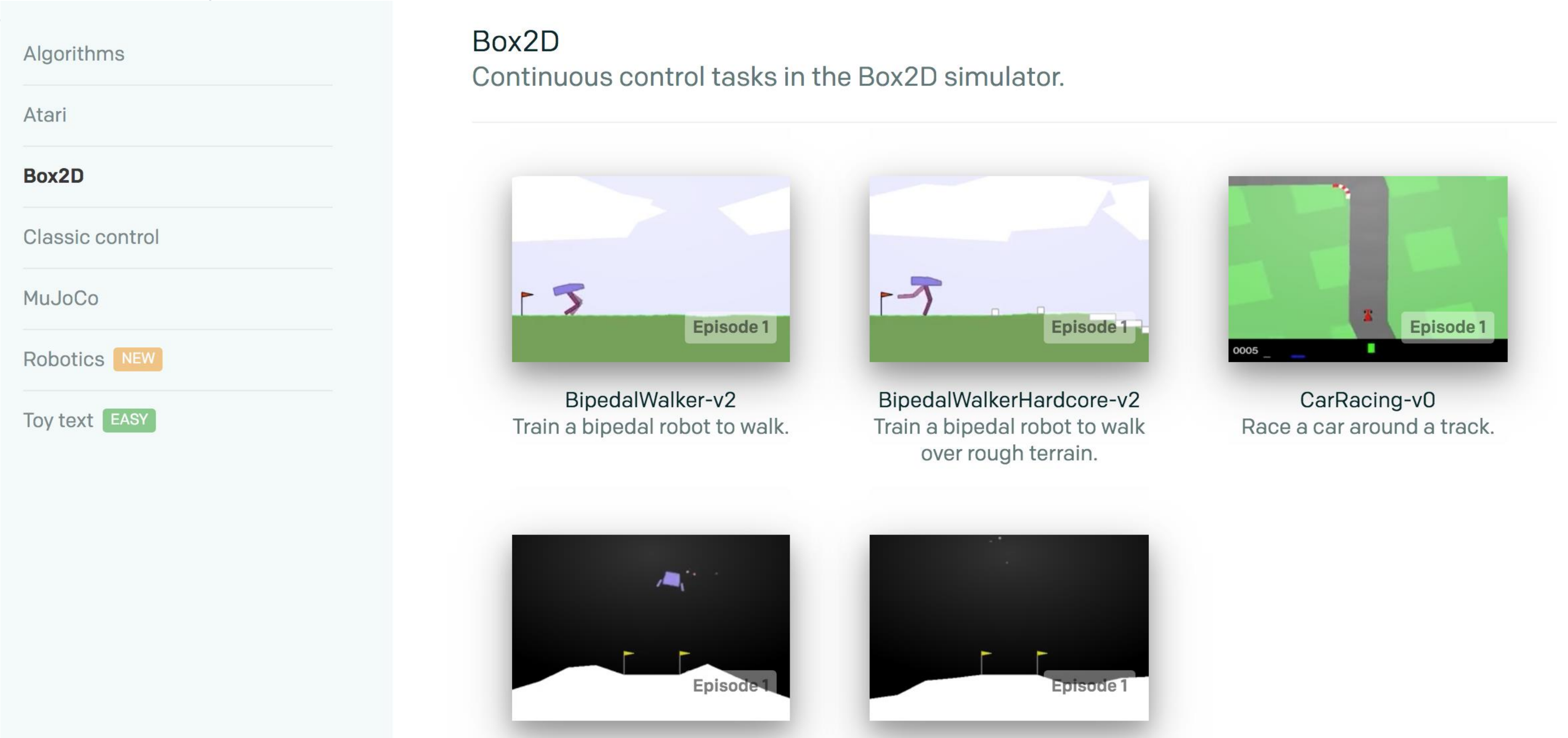
```
pip install gym
pip install atari-py
```

```
import gym
env = gym.make('MsPacman-v0')
env.reset()
print("Action space:",env.action_space)
print("Action meanings:", env.unwrapped.get_action_meanings())
print("Observation space:", env.observation_space)
```

```
Action space: Discrete(9)
Action meanings: ['NOOP', 'UP', 'RIGHT', 'LEFT', 'DOWN', 'UPRIGHT',
Observation space: Box(210, 160, 3)
```

Environments

Внезапный бонус для тех кто сделал задания 5 и 6!



OpenAI Gym Environments

ICLR

ICLR