Lecture 12. Intro to Reinforcement Learning

Alex Avdyushenko

Kazakh-British Technical University

November 26, 2022

Five-minutes block

Five-minutes block

- Describe (or draw) the general architecture of the autoencoder
- What is the main advantage of the self-supervised learning method?
- Please, write down GAN loss functions

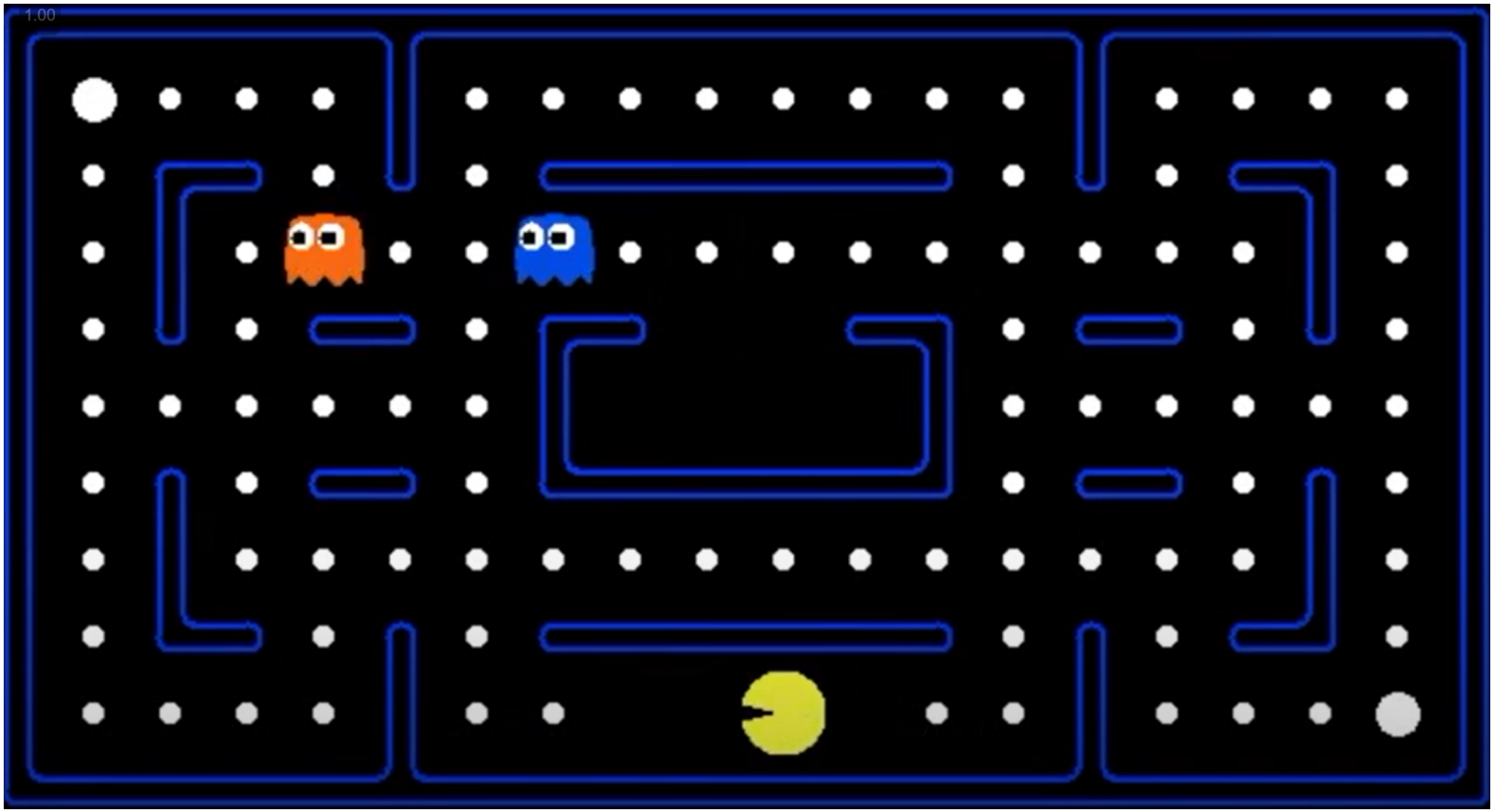
Problem Statement

- Up to this point, we either restored the function from the training set (X, Y) (supervised learning) or searched for the structure in the set of objects X (unsupervised learning)

Problem Statement

- Up to this point, we either restored the function from the training set (X, Y) (supervised learning) or searched for the structure in the set of objects X (unsupervised learning)
- But how does learning work in real world? Usually we do some action and get the result, gradually learning

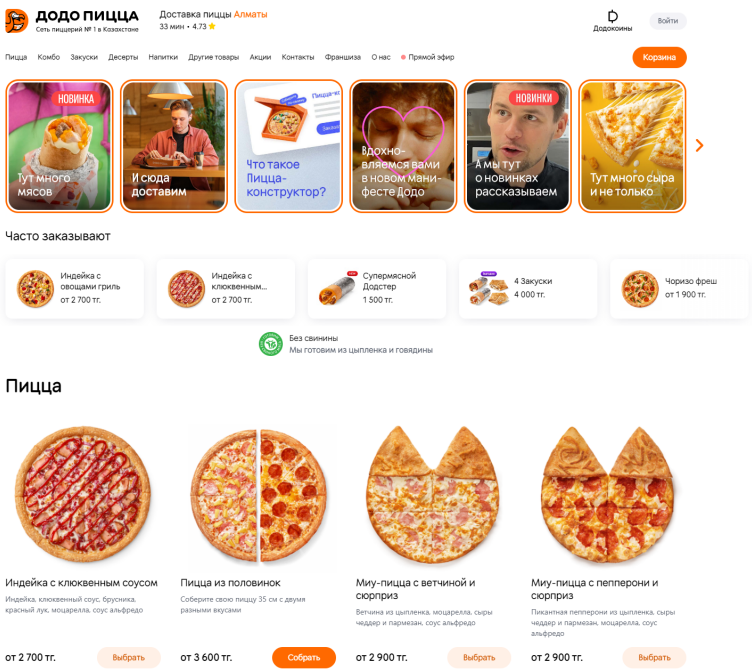
Real world

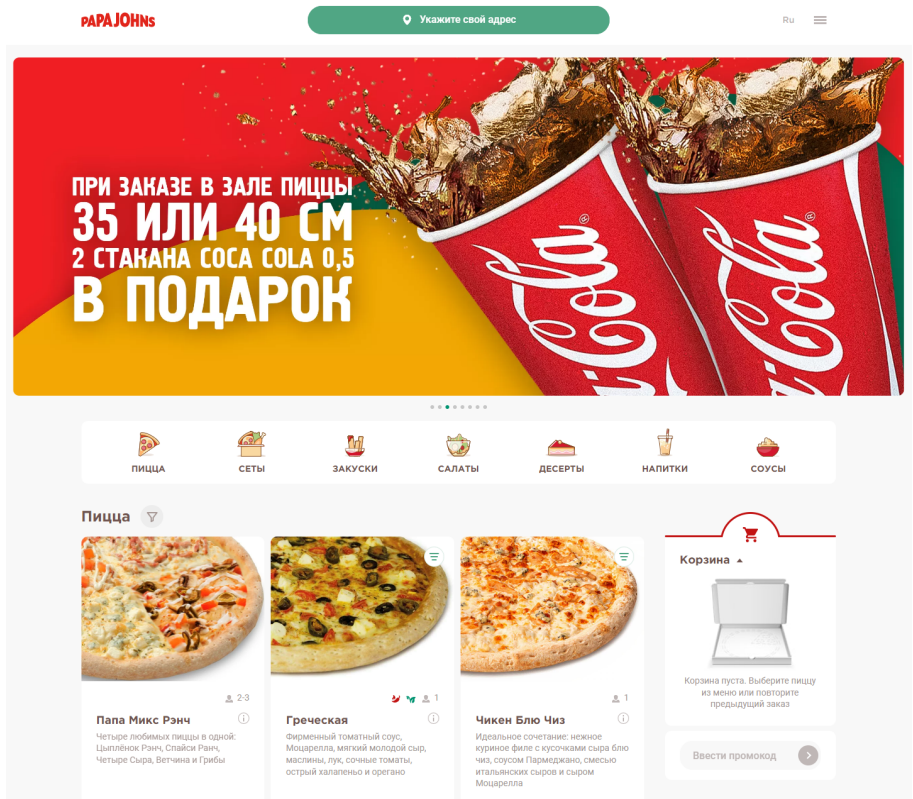


Deep Reinforcement Learning in Pac-man

More motivation

For example, we need to select the main page of a pizzeria website to attract the clients





What approaches are there?

- A/B testing — many users see potentially bad options
- multi-armed bandits — a special case of reinforcement learning

Bernoulli's one-armed bandit

Probability of winning $\theta = 0.05$

Multi-Armed Bandits

$\theta = 0.02$

$\theta = 0.01(\text{min})$

$\theta = 0.05$

$\theta = 0.1(\text{max})$

We do not know the true probabilities, but we want to come up with a strategy that maximizes the payoff (reward)

Mathematical statement of the problem

Given possible actions

$$x_1, \ldots, x_n$$

Mathematical statement of the problem

Given possible actions

$$x_1, \ldots, x_n$$

At the next iteration t, for each action x_i^t performed, we get the answer

$$y_i^t \sim q(y|x_i^t, \Theta),$$

Mathematical statement of the problem

Given possible actions

$$x_1, \ldots, x_n$$

At the next iteration t, for each action x_i^t performed, we get the answer

$$y_i^t \sim q(y|x_i^t, \Theta),$$

which brings us a **reward**

$$r_i^t = r(y_i^t)$$

Mathematical statement of the problem

Given possible actions

$$x_1, \ldots, x_n$$

At the next iteration t, for each action x_i^t performed, we get the answer

$$y_i^t \sim q(y|x_i^t, \Theta),$$

which brings us a **reward**

$$r_i^t = r(y_i^t)$$

There is an optimal action x_{i^*} (sometimes $x_{i_t^*}$)

$$\forall i: \; E(r_{i_t^*}^t) \geq E(r_i^t)$$

Mathematical statement of the problem

Given possible actions

$$x_1, \ldots, x_n$$

At the next iteration t, for each action x_i^t performed, we get the answer

$$y_i^t \sim q(y|x_i^t, \Theta),$$

which brings us a **reward**

$$r_i^t = r(y_i^t)$$

There is an optimal action x_{i^*} (sometimes $x_{i_t^*}$)

$$\forall i: \; E(r_{i_t^*}^t) \geq E(r_i^t)$$

Question

How to evaluate different strategies?

Measure of quality

Measure of quality

The quality measure of the multi-armed bandit algorithm a is usually **regret**

$$R(a) = \sum_{t=1}^{T} \left(E(r_{i_t^*}^t) - E(r_{i_t^a}^t) \right)$$

Under synthetic conditions (when we know the probabilities), we can consider

$$E\left(R\right) = \int_{\Theta} R(a) d\Theta$$

Multi-Armed Bandits

Possible applications:

- advertising banners
- recommendations (goods, music, movies etc.)
- slot machines

Approaches:

- Thompson sampling
- Upper Confidence Bound (UCB)
- ε-greedy strategy

Note

The main difference from more general reinforcement learning approach is that there is no environment state in multi-armed bandits

Multi-Armed Bandits

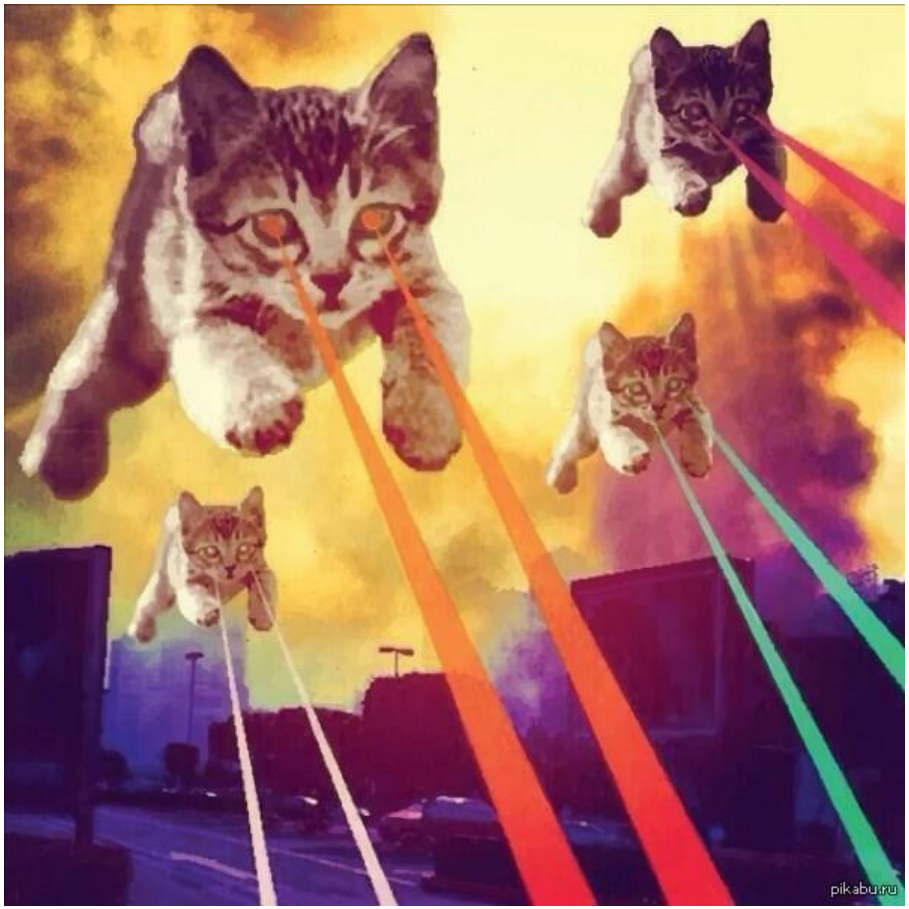
Disadvantage

Do not take into account delayed effects
For example, the effect of clickbait in advertising

Multi-Armed Bandits

Disadvantage

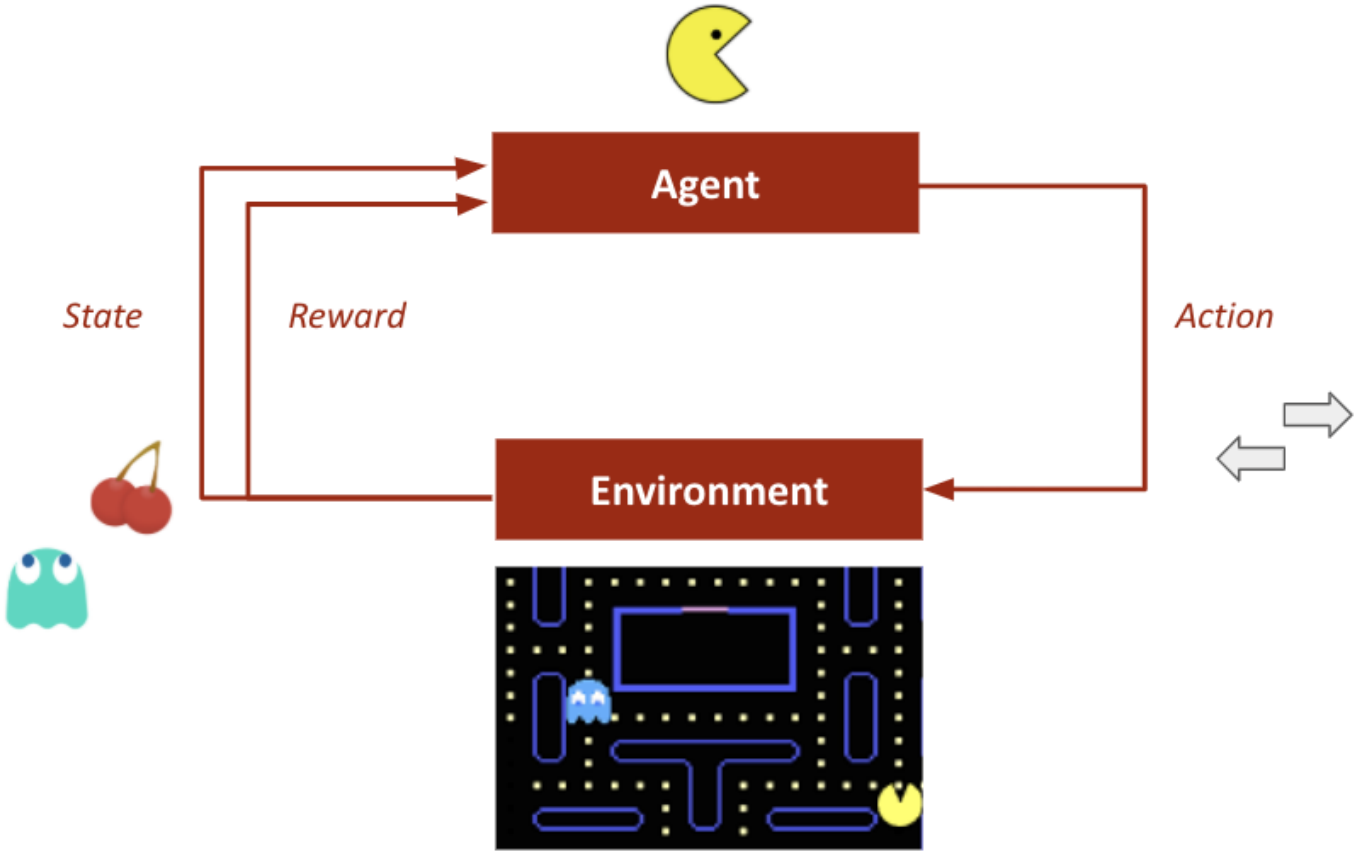
Do not take into account delayed effects
For example, the effect of clickbait in advertising



SHOCK! Cats want to enslave us

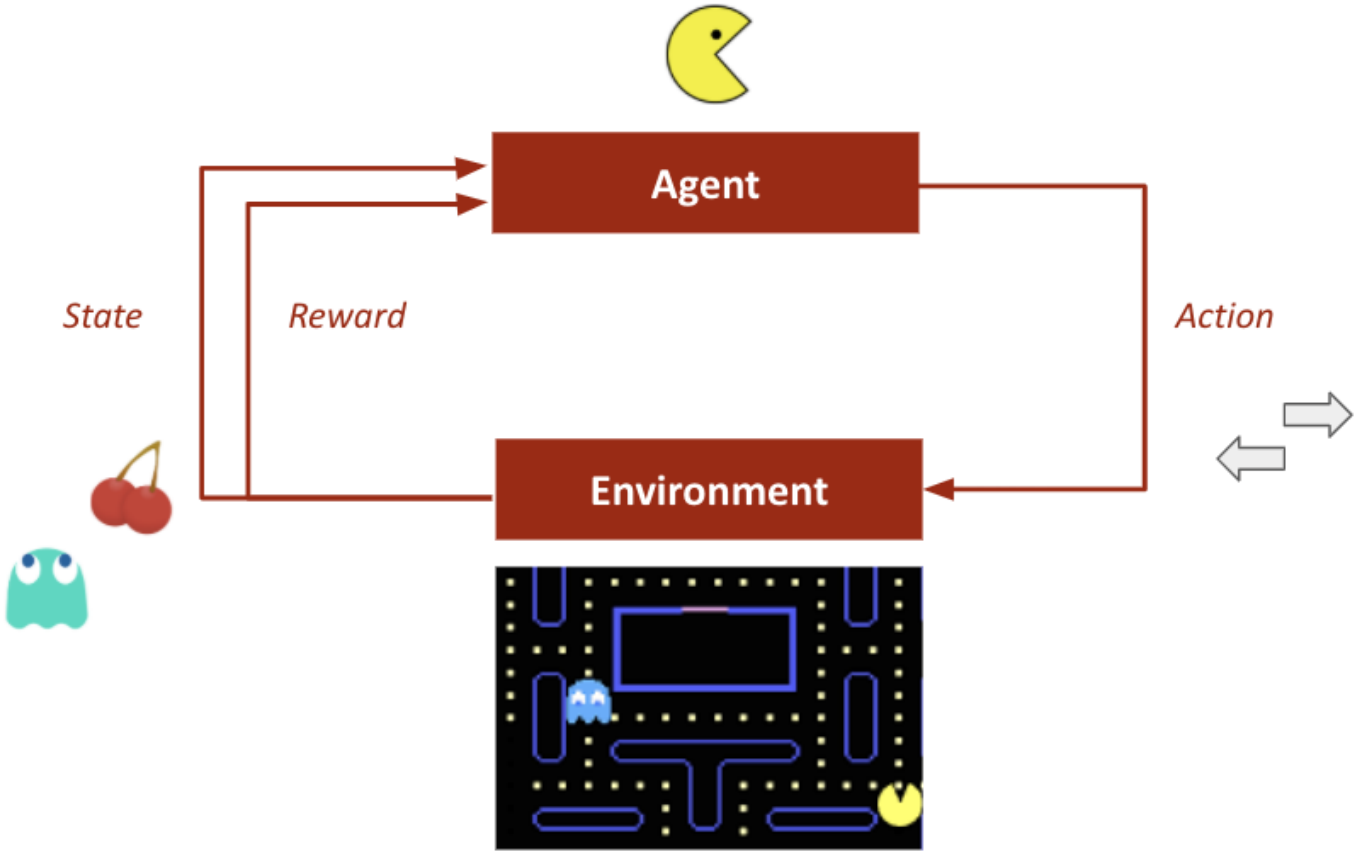
Reinforcement Learning

Examples



Reinforcement Learning

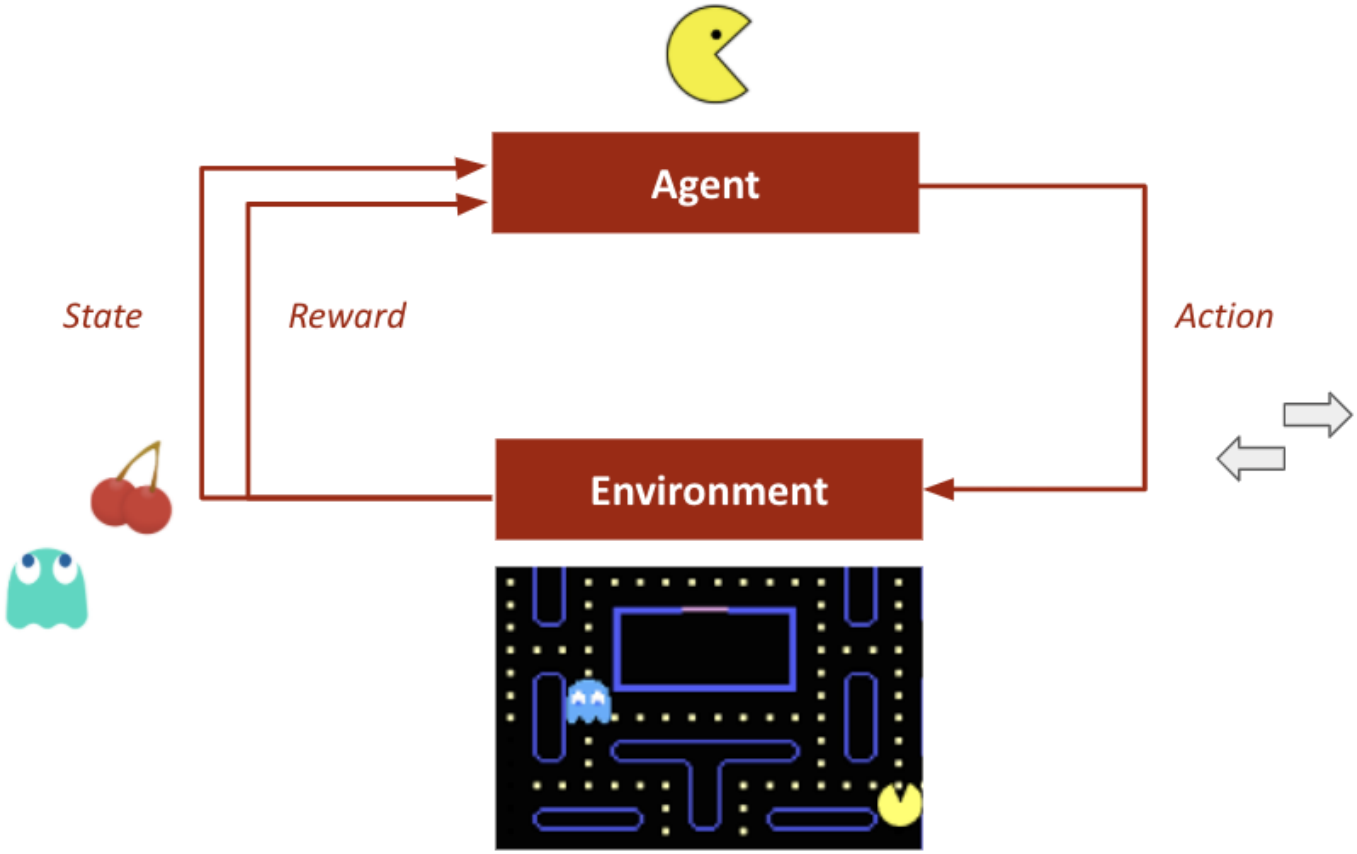
Examples



- robot control

Reinforcement Learning

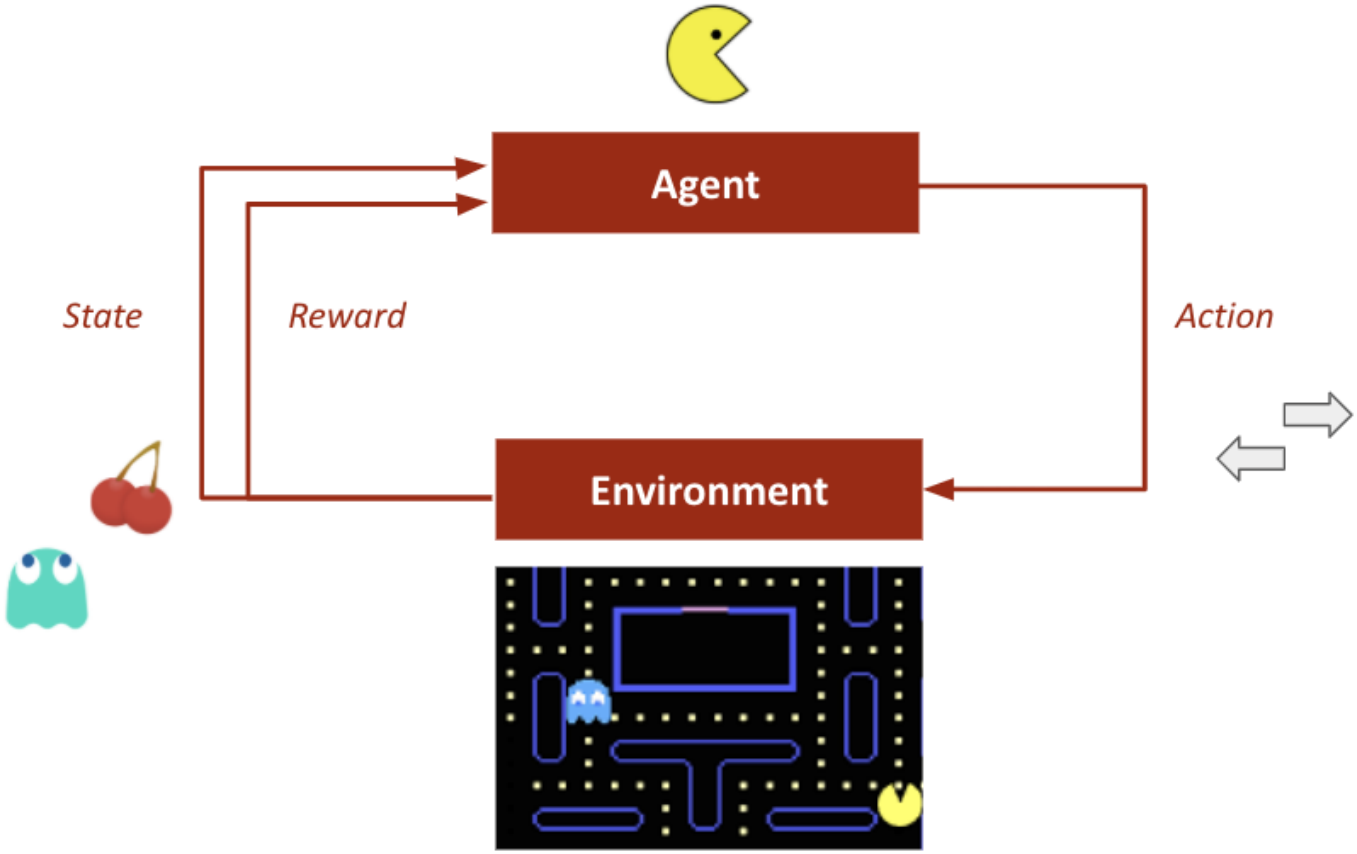
Examples



- robot control
- (video) games

Reinforcement Learning

Examples



- robot control
- (video) games
- security management

Agent Model in a Changing Environment

Definitions:

- state $s \in S$
- agent action $a \in A$
- reward $r \in R$
- state transition dynamics $P(s_{t+1}|s_t, a_t, \ldots, s_{t-i}, a_{t-i}, \ldots, s_0, a_0)$
- win function

$$r_t = r(s_t, a_t, \ldots, s_0, a_0)$$

- total reward $R = \sum_t r_t$
- agent policy $\pi(a|s)$

Task:

$$\pi(a|s) : \mathbb{E}_\pi[R] \to \max$$

Cross-entropy method. Algorithm

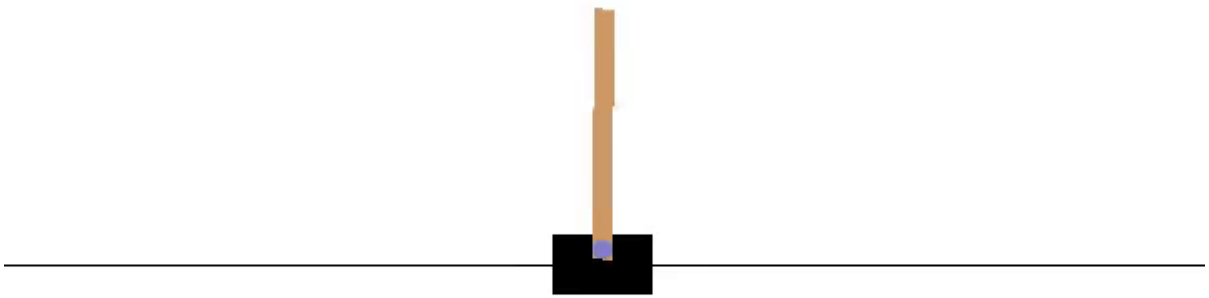
Trajectory — $[s_0, a_0, s_1, a_1, s_2, \ldots, a_{T-1}, s_T]$

Initialize strategy model $\pi(a|s)$

Repeat:

- play N sessions
- choose the best K of them and take their trajectories
- adjust $\pi(a|s)$ so that s is able to maximize the probabilities of actions from the best trajectories

Training Example



Cross-entropy method. Implementation with a table

As a strategy model, we simply take a matrix π of dimension $|S| \times |A|$
$\pi(a|s) = \pi_{s,a}$
after selecting the best trajectories, we obtain a set of pairs

$$\text{Elite} = [(s_0, a_0), (s_1, a_1), (s_2, a_2), \ldots, (s_H, a_H)]$$

and maximize the likelihood

$$\pi_{s,a} = \frac{\sum\limits_{s_t, a_t \in \text{Elite}} [s_t = s][a_t = a]}{\sum\limits_{s_t, a_t \ in \text{Elite}} [s_t = s]}$$

There is a problem..

There is a problem..

that there are a lot of states:

Approximate cross-entropy method

Possible solutions:

- split the state space into sections and treat them as states

Approximate cross-entropy method

Possible solutions:

- split the state space into sections and treat them as states
- get probabilities from $\pi_\theta(a|s)$ machine learning model: linear model, neural network, random forest

Approximate cross-entropy method

Possible solutions:

- split the state space into sections and treat them as states
- get probabilities from $\pi_\theta(a|s)$ machine learning model: linear model, neural network, random forest
- often these probabilities need to be further developed later

Approximate cross-entropy method

Example

Approximate cross-entropy method

Example

- As a strategy model, we take just a neural network π_θ

Approximate cross-entropy method

Example

- As a strategy model, we take just a neural network π_θ
- Initialize with random weights

Approximate cross-entropy method

Example

- As a strategy model, we take just a neural network π_θ
- Initialize with random weights
- At each iteration, after selecting the best trajectories, we obtain a set of pairs

$$\text{Elite} = [(s_0, a_0), (s_1, a_1), (s_2, a_2), \ldots, (s_H, a_H)]$$

Approximate cross-entropy method

Example

- As a strategy model, we take just a neural network π_θ
- Initialize with random weights
- At each iteration, after selecting the best trajectories, we obtain a set of pairs

$$\text{Elite} = [(s_0, a_0), (s_1, a_1), (s_2, a_2), \ldots, (s_H, a_H)]$$

- and perform optimization

$$\pi = \arg\max_\theta \sum_{s_i, a_i \in \text{Elite}} \log \pi(a_i|s_i) = \arg\max_\theta \mathscr{L}(\theta)$$

Approximate cross-entropy method

Example

- As a strategy model, we take just a neural network π_θ
- Initialize with random weights
- At each iteration, after selecting the best trajectories, we obtain a set of pairs

$$\text{Elite} = [(s_0, a_0), (s_1, a_1), (s_2, a_2), \ldots, (s_H, a_H)]$$

- and perform optimization

$$\pi = \arg\max_\theta \sum_{s_i, a_i \in \text{Elite}} \log \pi(a_i|s_i) = \arg\max_\theta \mathscr{L}(\theta)$$

- that is

$$\theta_{t+1} = \theta_t + \alpha \nabla \mathscr{L}(\theta)$$

Why is it called cross-entropy method?

$$KL(p_1(x)\|p_2(x)) = E_{x\sim p_1(x)} \log \frac{p_1(x)}{p_2(x)} =$$

$$= \underbrace{E_{x\sim p_1(x)} \log p_1(x)}_{\text{entropy}} - \underbrace{E_{x\sim p_1(x)} \log p_2(x)}_{\text{cross entropy}}$$

Disadvantages of the cross-entropy method

Disadvantages of the cross-entropy method

- it is unstable for small samples

Disadvantages of the cross-entropy method

- it is unstable for small samples
- in the case of a non-deterministic environment, it chooses lucky cases (random played in favor of the agent)

Disadvantages of the cross-entropy method

- it is unstable for small samples
- in the case of a non-deterministic environment, it chooses lucky cases (random played in favor of the agent)
- it is focusing on behavior in simple states

Disadvantages of the cross-entropy method

- it is unstable for small samples
- in the case of a non-deterministic environment, it chooses lucky cases (random played in favor of the agent)
- it is focusing on behavior in simple states
- it ignores a lot of information

Disadvantages of the cross-entropy method

- it is unstable for small samples
- in the case of a non-deterministic environment, it chooses lucky cases (random played in favor of the agent)
- it is focusing on behavior in simple states
- it ignores a lot of information
- there are tasks in which the end never comes (stock market game)

Q-Learning

- sometimes it is not necessary to finish the game to evaluate the effectiveness of the strategy
- the effect of the action may appear later

Markov Decision Process

Definitions:

- state $s \in S$
- agent action $a \in A$
- reward $r \in R$
- transition dynamics
  $P(s_{t+1}|s_t, a_t, \ldots, s_{t-i}, a_{t-i}, \ldots, s_0, a_0) =$
  $P(s_{t+1}|s_t, a_t)$ **(Markov's assumption)**
- win function

$$r_t = r(s_t, a_t)$$

(Markov's assumption)

- total reward $R = \sum_t r_t$
- agent policy

$$\pi(a|s)$$

Task:

$$\pi(a|s) : \mathbb{E}_\pi[R] \to \max$$

Important Features

Average absolute win:

$$\mathbb{E}_{s_0 \sim p(s_0)}, \mathbb{E}_{a_0 \sim \pi(a|s_0)}, \mathbb{E}_{s_1, r_0 \sim P(s', r|s, a)} \cdots [r_0 + r_1 + \cdots + r_T]$$

State value function:

$$V^{\pi}(s) = \mathbb{E}_{\pi}[R_t | s_t = s] = \mathbb{E}_{\pi} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \right],$$

Action value function in state:

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}[R_t | s_t = s, a_t = a] = \mathbb{E}_{\pi} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a \right],$$

where π is the strategy followed by the agent, $\gamma \in [0, 1]$

TD-training

temporal difference

We arbitrarily initialize the function $V(s)$ and the strategy π
Then we repeat

- initialize s
- for each agent step
  - select a by strategy π
  - do action a, get result r and next state s'
  - update function $V(s)$ by formula

$$V(s) = V(s) + \alpha\left(r + \gamma V(s') - V(s)\right)$$

  - go to the next step by assigning $s := s'$

Bellman Equation

for the optimal value function Q^*

$$Q^*(s,a) = \mathbb{E}_\pi \left[r_t + \gamma \max_{a'} Q^*(s_{t+1}, a') \mid s_t = s, a_t = a \right]$$

Note

The greedy strategy π with respect to $Q^*(s,a)$ "choose the action that maximizes the Bellman equations" is optimal.

State utility recalculation

$$V(s) = \max_a [r(s,a) + \gamma V(s'(s,a))]$$

that is, with probabilistic transitions

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

Iterative State Utility Recalculation Formula

$$\forall s \; V_0(s) = 0$$
$$V_{i+1}(s) = \max_a [r(s,a) + \gamma \mathbb{E}_{s' \sim P(s'|s,a)} V_i(s')]$$

State utility recalculation

$$V(s) = \max_a [r(s,a) + \gamma V(s'(s,a))]$$

that is, with probabilistic transitions

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

Iterative State Utility Recalculation Formula

$$\forall s \; V_0(s) = 0$$
$$V_{i+1}(s) = \max_a [r(s,a) + \gamma \mathbb{E}_{s' \sim P(s'|s,a)} V_i(s')]$$

Question

Note To use this formula in practice, we need to know the transition probabilities $P(s'|s,a)$

Utility of an action

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

The strategy of the game is defined as follows

$$\pi(s) : \arg\max_a Q(s, a)$$

Utility of an action

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

The strategy of the game is defined as follows

$$\pi(s) : \arg\max_a Q(s, a)$$

Again due to stochasticity

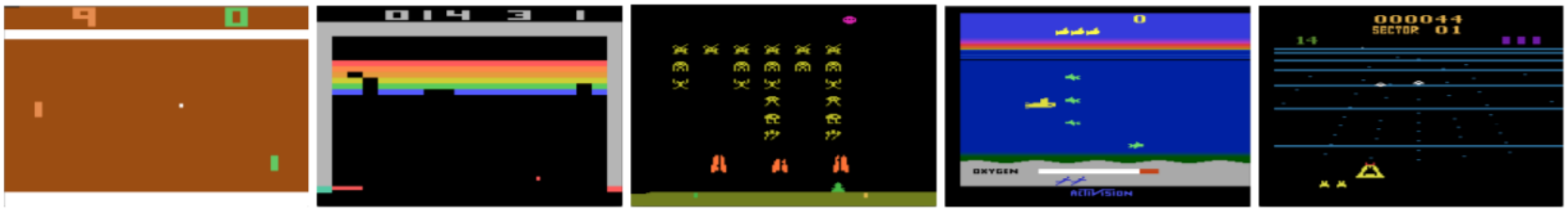
$$Q(s, a) = \mathbb{E}_{s'}[r(s, a) + \gamma V(s')]$$

It is possible to estimate the expected value without an explicit distribution using the Monte Carlo method and averaging over the outcomes:

$$Q(s_t, a_t) \leftarrow \alpha \left(r_t + \gamma \max_a Q(s_{t+1}, a) \right) + (1 - \alpha) Q(s_t, a_t)$$

DQN (Deep Q-Learning Network)

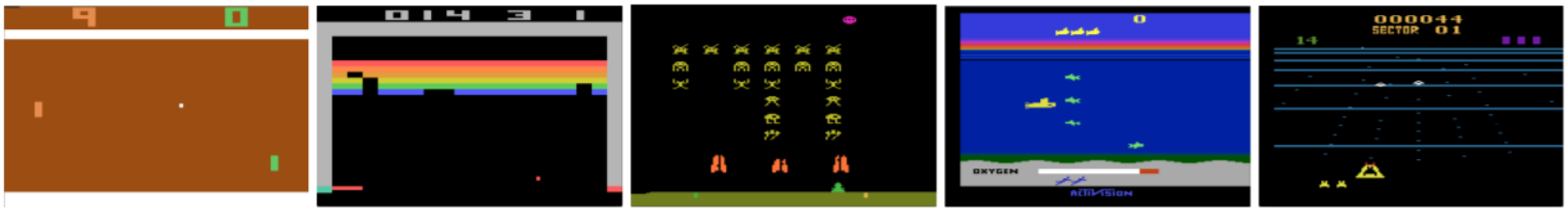
Environment is Atari game emulator, each frame $210 \times 160 pix,\ 128 col$



Pong | Breakout | Space Invaders | Seaquest | Beam Rider

DQN (Deep Q-Learning Network)

Environment is Atari game emulator, each frame $210 \times 160 pix$, $128 col$



Pong | Breakout | Space Invaders | Seaquest | Beam Rider

s states: 4 consecutive frames compressed to 84×84
Actions a: 4 to 18, depending on the game
r Rewards: current in-game SCORE
Value function $Q(s, a; w)$: ConvNN with input s and $|A|$ outputs

---

V.Mnih et al. (DeepMind). Playing Atari with deep reinforcement learning. 2013

DQN Method

Saving trajectories $(s_t, a_t, r_t)_{t=1}^{T}$ in reply memory for repeated experience replay

Approximation of the optimal value function $Q(s_t, a_t)$ for fixed current network parameters w_t

$$y_t = \begin{cases} r_t, \text{ if state } s_{t+1} \text{ terminal} \\ r_t + \gamma \max\limits_{a} Q(s_{t+1}, a; w_t), \text{ otherwise} \end{cases}$$

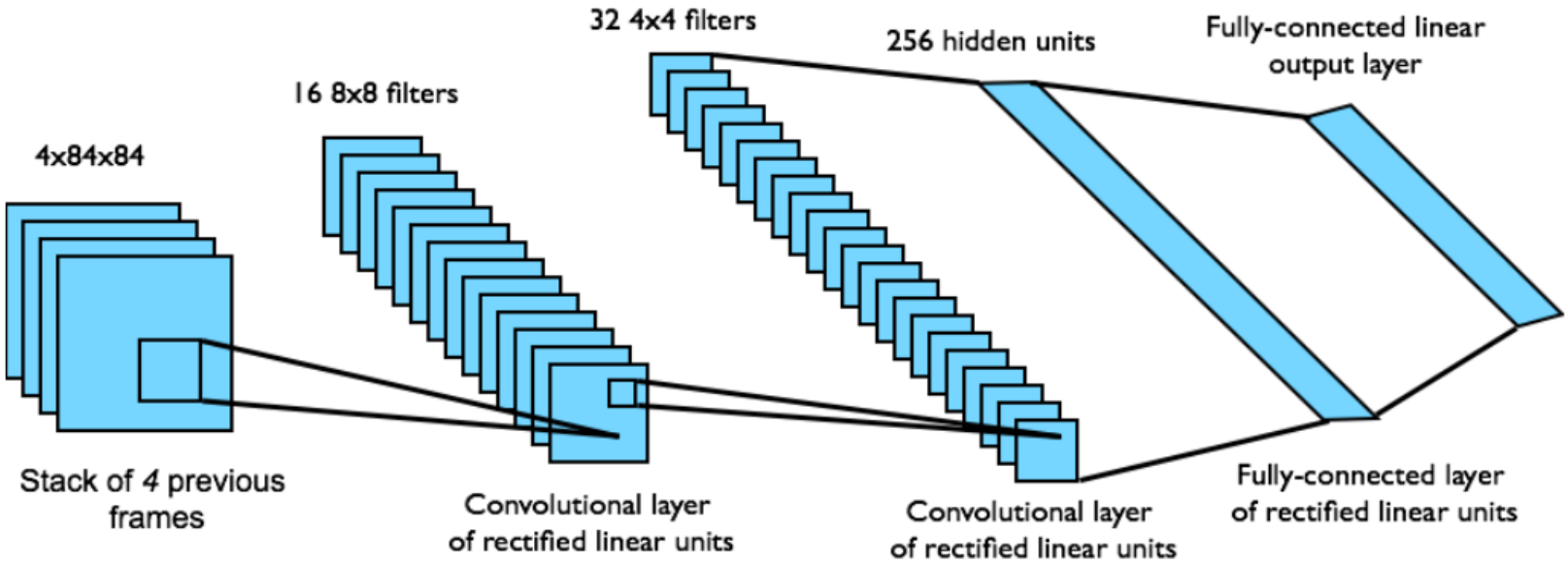
Loss function for training the neural network model $Q(s, a; w)$:

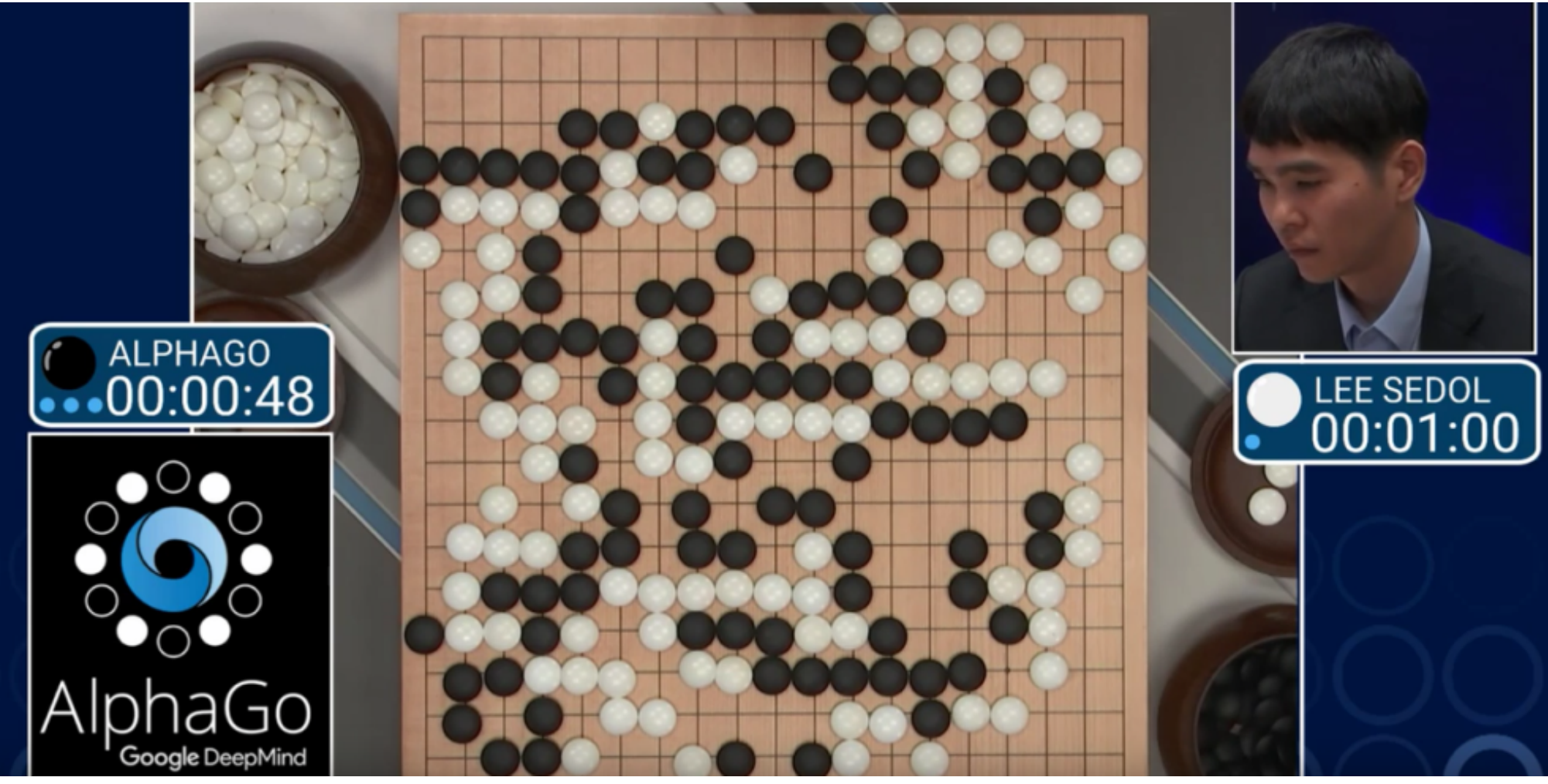
$$\mathscr{L}(w) = (Q(s_t, a_t; w) - y_t)^2$$

Stochastic gradient SGD (by mini-batches of length 32):

$$w_{t+1} = w_t - \eta \left(Q(s_t, a_t; w_t) - y_t\right) \nabla_w Q(s_t, a_t; w_t)$$

Network architecture for the value function




ALPHAGO
00:00:48
AlphaGo
Google DeepMind
LEE SEDOL
00:01:00


PRODUCTION
DeepMind
StarCraft II
DEMONSTRATION
AlphaStar
7:01
REPLAY
Catalyst LE
BLIZZARD ENTERTAINMENT
2 AlphaStar
85 /102
275 +1287
130 +627
39
46
SUPPLY
MINERALS
GAS
WORKERS
ARMY
WEAPONS & ARMOR
0 LiquidMaNa
76 /94
390 +1679
124 +313
38
38

Summary

- we got acquainted with the reinforcement learning problem statement
- remembered multi-armed bandits
- got acquainted with the methods of cross-entropy, TD-learning and Q-learning
- discussed Deep Q-learning Network
- did NOT discuss the central method in robotics — policy gradient descent

Summary

- we got acquainted with the reinforcement learning problem statement
- remembered multi-armed bandits
- got acquainted with the methods of cross-entropy, TD-learning and Q-learning
- discussed Deep Q-learning Network
- did NOT discuss the central method in robotics — policy gradient descent

What else can you see?

- main book on the topic: Reinforcement Learning: An Introduction, Richard S. Sutton and Andrew G. Barto
- Educational resource on RL produced by OpenAI
- Yuxi Li. Resources for Deep Reinforcement Learning. 2018
- Intro to RL with David Silver