# An introduction to RL and deep RL (2)

BABAK BADNAVA
MULTI-AGENT SYSTEMS LAB, IUST

This slide is mainly taken from silver's slides

مرکز تحقیقات هوش پارت

# Deep neural networks for supervised tasks

- Data: sample, label
- Using deep neural networks: $x \xrightarrow[w_1]{} h_1 \xrightarrow[w_2]{} \dots \xrightarrow[w_n]{} h_n \xrightarrow[w_{n+1}]{} y$
- $h_i$ could be any kind of function
- We have:
  - A sample ($x$)
  - Its label ($y^*$)
  - Network's prediction ($y$)

# Deep neural networks for supervised tasks (2)

- Define a loss function: $x \xrightarrow[w_1]{} h_1 \xrightarrow[w_2]{} \dots \xrightarrow[w_n]{} h_n \xrightarrow[w_{n+1}]{} y \longrightarrow l(y)$
- Loss function could be:
  - Mean-squared error: $l(y) = ||y^* - y||^2$
  - Log likelihood: $l(y) = \log P[y^*|x]$
  - ….
- Minimize loss
- Using different optimization algorithms

# Overview of approximation methods

- Value estimation
  - Use a neural network to approximate value function
  - Define a policy function on top of it
- Policy gradient
  - Use a neural network to approximate policy function directly

# Let’s apply it to SARSA

- Action value function:
  - $Q_\pi(s,a) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \ \dots | S_t = s, A_t = a]$
- Represent it by Q-network with weights w
  - $Q_\pi(s,a) \approx Q(s,a,w)$
- Define loss function:

$$\mathcal{L}(w) = \mathbb{E}\left[\left(\underbrace{r + \gamma Q(s', a', w)}_{\text{target}} - Q(s, a, w)\right)^2\right]$$

# How about Q-learning?

- Loss function:

$$\mathcal{L}(w) = \mathbb{E}\left[\left(\underbrace{r + \gamma \max_{a'} Q(s', a', w)}_{\text{target}} - Q(s, a, w)\right)^2\right]$$

- Optimize this using SGD, using $\frac{\partial L(w)}{\partial w}$

$$\frac{\partial \mathcal{L}(w)}{\partial w} = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w)\right) \frac{\partial Q(s, a, w)}{\partial w}\right]$$

# Summary

- Use this network to estimate value function instead of a Q-table

*State, action* → *Neural network* → *Q(s, a, w)*

- Nothing else have been changed
- Does it work well?

# Compare

**Sarsa (on-policy TD control) for estimating $Q \approx q_*$**

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize $S$
  Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
  Loop for each step of episode:
    Take action $A$, observe $R$, $S'$
    Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until $S$ is terminal

Algorithm from Sutton 2018

# Compare (2)

- Initialize weights
- Loop for each episode:
  - Initialize s
  - Choose a from s using policy derived from Q(s, a, w)
  - Loop for each step of episode:
    - Take action a, observe r, s’
    - Choose a’ from s’ using policy derived from Q(s’, a’, w)
    - Compute gradient of loss and update the network
    - s <- s’; a <- a’
  - Until s is terminal

# Does it work well?

- Oscillates or diverges
- Why?
  - Data is sequential
    - Successive samples are correlated
  - Policy changes rapidly with slight change to Q-values
    - Distribution of data will change
  - Scale of rewards and Q-values is unknown

# What to do?

- Use experience reply
  - Break correlation between data
- Freeze target Q-network
  - Avoid oscillations
- Clip rewards or normalize network adaptively

# Stable Deep RL: experience replay

- To remove correlations, build data-set from agent's own experience
- Take action $a_t$ according to epsilon-greedy policy
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$in replay memory
- Sample random mini-batch of transitions $(s, a, r, s')$ from repay memory
- Optimise MSE

$$\mathcal{L}(w) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}}\left[\left(r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w)\right)^2\right]$$

مرکز تحقیقات هوش پارت

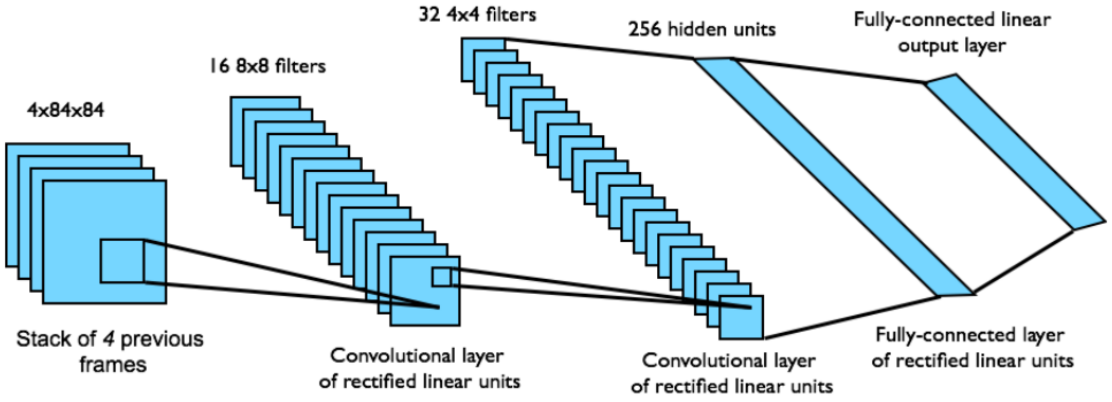
# Stable Deep RL: fixed target Q-network

- Use target Q-network with fixed parameter
- Choose action based on the target Q-network
- Compute Q-learning target w.r.t old, fixed parameter
- Optimize MSE
- Periodically update fixed parameters $w^- \leftarrow w$

$$\mathcal{L}(w) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[ \left( r + \gamma \max_{a'} Q(s', a', w^-) - Q(s, a, w) \right)^2 \right]$$

# DQN architecture in Atari

- End-to-end learning of values **Q(s, a)** from pixels **s**
- Input state **s** is stack of raw pixels from last 4 frames
- Output is Q(s, a) for 18 joystick/button positions
- Reward is change in score for that step



# Policy gradient

- Represent policy by deep network $a = \pi(s, u)$ with weights $u$
- Define objective function as total discounted reward

$$J(u) = \mathbb{E}\left[r_1 + \gamma r_2 + \gamma^2 r_3 + \ldots\right]$$

- Optimise objective end-to-end by SGD
  - Adjust policy parameters $u$ to achieve more reward

$$\frac{\partial J(u)}{\partial u} = \mathbb{E}_s\left[\frac{\partial Q^\pi(s,a)}{\partial u}\right]$$
$$= \mathbb{E}_s\left[\frac{\partial Q^\pi(s,a)}{\partial a}\frac{\partial \pi(s,u)}{\partial u}\right]$$

# Actor-Critic method

- Actor is a policy $\pi(s, u)$ $\quad s \xrightarrow[u_1]{} \ldots \xrightarrow[u_n]{} a$
- Critic is value function $Q(s, a, w)$ with parameter $w$

$$s, a \xrightarrow[w_1]{} \ldots \xrightarrow[w_n]{} Q$$

- Critics provide loss function for actor

$$s \xrightarrow[u_1]{} \ldots \xrightarrow[u_n]{} a \xrightarrow[w_1]{} \ldots \xrightarrow[w_n]{} Q$$

- Gradient back propagates from critic into actor

$$\frac{\partial a}{\partial u} \longleftarrow \ldots \longleftarrow \frac{\partial Q}{\partial a} \longleftarrow \ldots \longleftarrow$$

مرکز تحقیقات هوش پارت

# Actor-Critic: Learning rules

- Critic estimates value of current policy by Q-learning

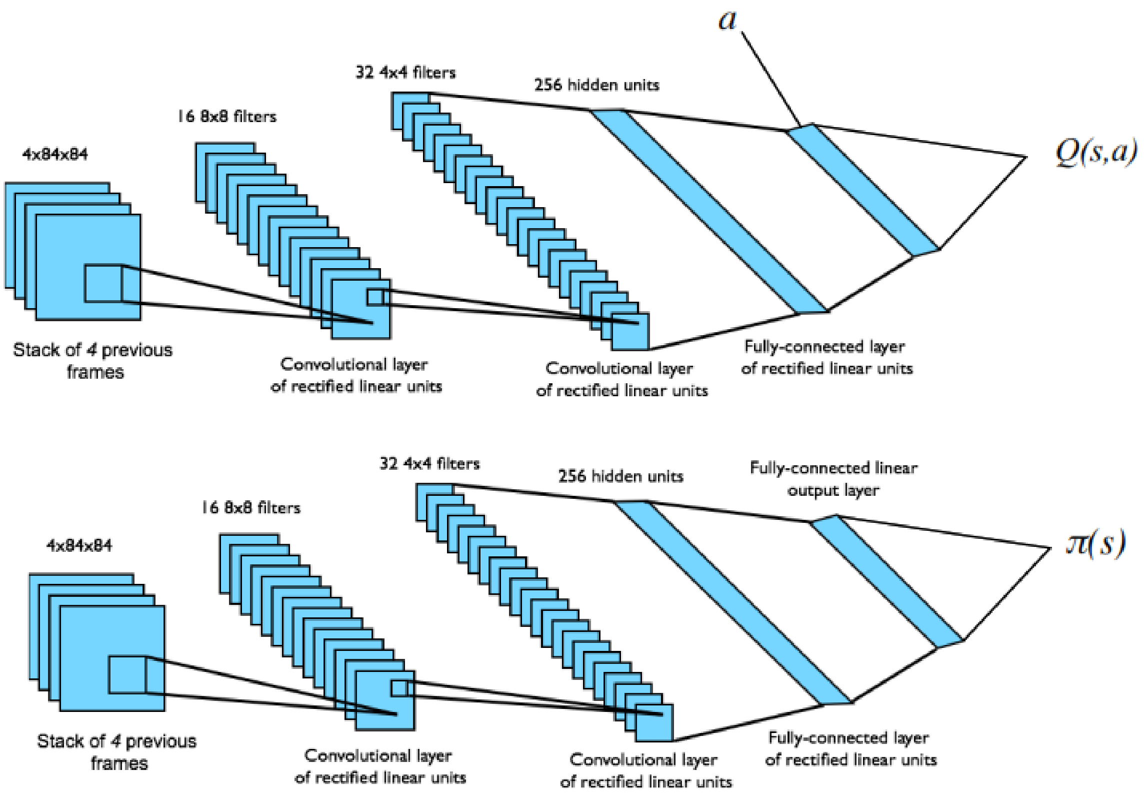
$$\frac{\partial \mathcal{L}(w)}{\partial w} = \mathbb{E}\left[\left(r + \gamma Q(s', \pi(s'), w) - Q(s, a, w)\right)\frac{\partial Q(s, a, w)}{\partial w}\right]$$

- Actor updates policy in direction that improves Q

$$\frac{\partial J(u)}{\partial u} = \mathbb{E}_s\left[\frac{\partial Q(s, a, w)}{\partial a}\frac{\partial \pi(s, u)}{\partial u}\right]$$

# Architecture and tips

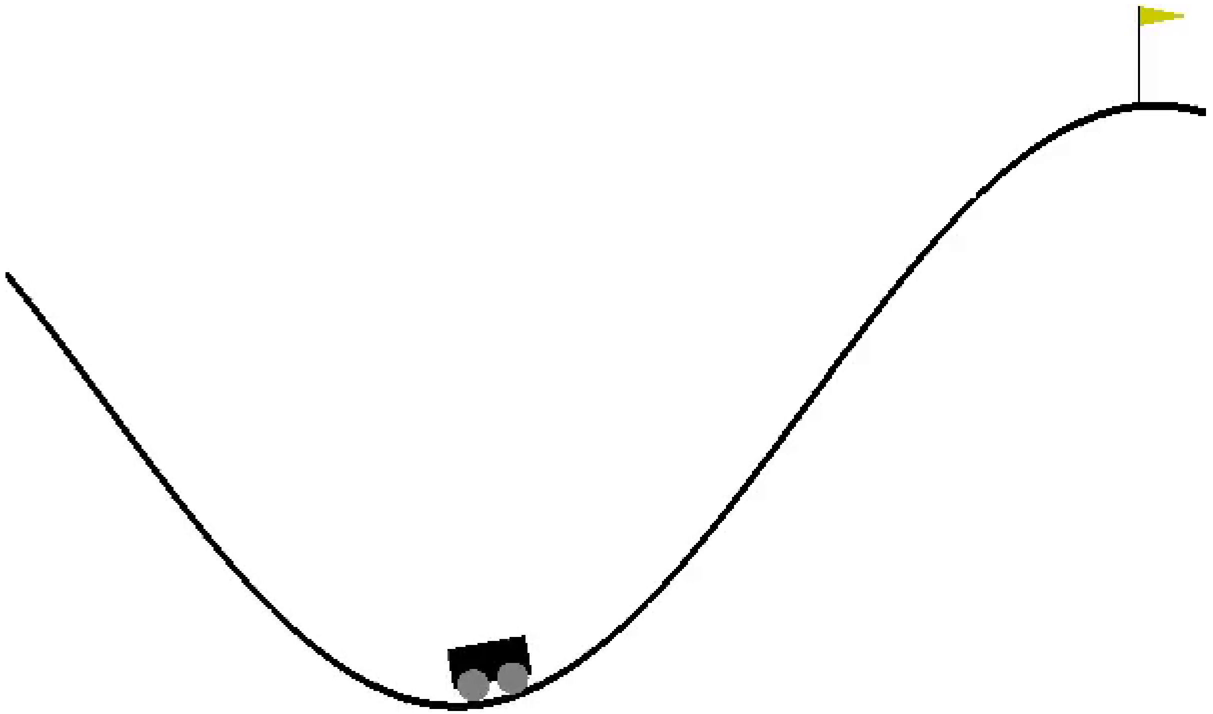
- Use experience reply
- Freeze target network



# How to improve?

- Run multiple simulation simultaneously instead of reply memory
  - Try this one as a practice
  - And think about its advantageous
- Use Transfer Learning
- …

# A simple python code



# References

- David Silver’s slide
- An introduction to Reinforcement Learning by Sutton 2nd edition
- Berkeley’s deep RL boot camp materials available at here

مرکز تحقیقات هوش پارت

Questions?