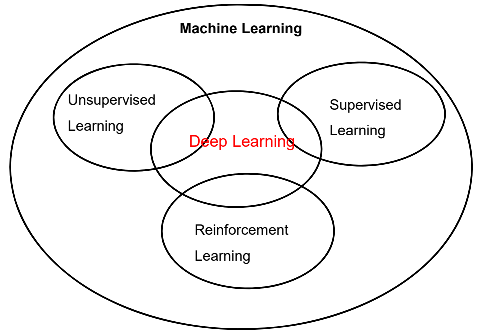
Deep reinforcement learning

Qiang Ji

Introduction



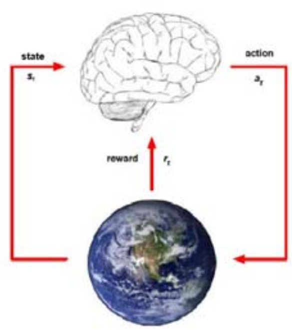
So far, we have shown deep learning application to: supervised learning (CNN and LSTM) and unsupervised learning (GANs)

We now discuss deep learning application to reinforcement learning.

Outlines

- Introduction
- Markov Decision Process (MDP)
- Partial Observed Markov Decision Process (POMDP)
- Deep Reinforcement Learning as MDP
  - Value-Based Deep RL: Q-function method
  - Policy-Based Deep RL :Policy gradient method
- Application of deep reinforcement learning

Reinforcement Learning



Learning to control a system so as to maximize some numerical value which represents a long-term objective.

- There is no supervisor, only a reward signal
- Directly interacts with the world
- Agent's actions affect the state of the world
- Feedback is delayed, not instantaneous
- Time really matters -sequential, non i.i.d data
- Learning via implicit or weak supervision. Most human learning is weak and implicit
- Autonomous learning or lifelong learning

=> MDP framework

Slides from David Silver, Google DeepMind

Markov Decision Process (MDP)

MDP** consists of a five tuple process* ***(S, A, P, R, γ)

***S** - set of states of the world (observed).*

***A** - set of actions.*

***P** - state transition probability matrix, which specifies the dynamics.*

In particular, $\mathbf{P}^{a}_{ss'}$*=p(s'|s, a) specifies the probability of transitioning to next state s' given current state s and current action a.*

***R** - reward, where* $r(s,a)$ *gives the immediate reward from doing action a at state s.*

***γ** – discount factor (<1).*

Markov property: $P[s_{t+1}|s_t] = P[s_{t+1}|s_1, \ldots, s_t]$

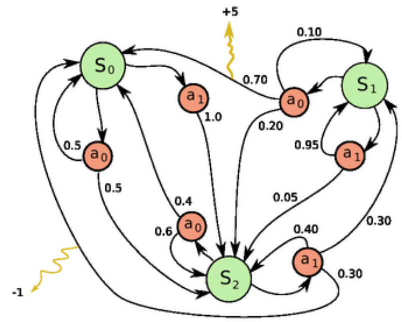
- next state only depends on current state

Policy $\pi: s \rightarrow a$ that maps state s to action a

- deterministic a=$\pi(s)$ or stochastic policy $\pi(s,a)$=$P[a|s]$.

***MDP inference**: identify the optimal policy that maximizes the expected current and future rewards*

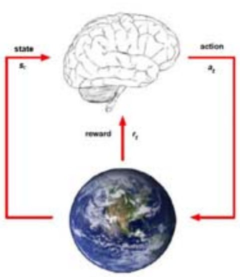
Markov Decision Process (MDP)



Example of a simple MDP with three states (green circles) and two actions (orange circles), with two rewards (yellow arrows).

Picture from wikipedia

Markov Decision Process (MDP)



- At each step *t* the agent:
  - Receives state s_t
  - Executes action a_t based on π
  - Receives scalar reward r_t
- The environment:
  - Receives action a_t
  - Emits state s_t based on $p^{a}_{ss'}$
  - Emits scalar reward r_t
- Repeats until the goal is achieved

Slides from David Silver, Google DeepMind

Video demo

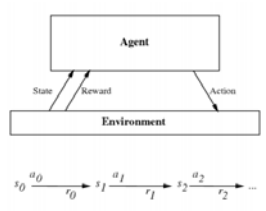
- This demo shows that the robot, though impressive in keeping its balance and in performing its task, cannot understand the human's implicit feedback and keep repeating the same tasks.

https://www.youtube.com/watch?v=zkv-_LqTeQA

https://www.youtube.com/watch?v=aFuA50H9uek

State-Action Value Function

$$Q^{\pi}(s_t = s, at = a) = E\left[\sum_{k=0}^{T} \gamma^k r_{t+k} \,|\pi\right]$$



State-action value function is the expected total reward starting from current state s, taking action a, and then following policy π. γ (<1)is a discount factor . T is the end of an episode (reaching the goal). The expectation is taken over policy π and transition probabiltiy P.

Optimal state-action value function

$Q^*(s, \mathrm{a}) = \max_{\pi} Q^{\pi}(s, a)$

Optimal policy

$\pi^*(s)$ =argmax$_a$ $Q^*(s, \mathrm{a})$

MDP – Solving

Solving MP involves identifying the optimal policy π. It has two approaches

1. Value Iteration

1) Start from an initial Q*, iteratively update Q* using Bellman equation until convergence. It can be proven that the final Q* is optimal.
2) Obtain the optimal policy π from the final Q*

Depending on if the transition probability $p(s_{t=1}|s_t,a_t)$ is known or not, it can be performed either through model-based or model-free approach

2. **Policy Iteration**

1) Start from an initial π
2) Compute Q^{π} using Bellman equation
3) Update π using updated Q^{π}
4) Repeat step 2-3 until convergence

Bellman Equation

State-action value function can be computed recursively:

$$Q^{\pi}(s_t, a_t) = E\left[\sum_{k=0}^{T} \gamma^k r_{t+k} \,|\pi\right]$$
$$= r(s_t, a_t) + \gamma E[Q^{\pi}(s_{t+1}, a_{t+1})]$$

The optimal state-action value function can also be computed recursively

$$\mathrm{Q}^*(s_t, a_t) = r(s_t, a_t) + \gamma E\left[\max_{a_{t+1} \in A} Q^*(s_{t+1}, a_{t+1})\right]$$

In fact, it can be proved that the recursive updating can be applied to any initial Q^0, with sufficient iterations $Q^0 \to Q^1 \to Q^2 \to Q^3 \ldots \to Q^*$

Value Iteration-Model based approach

Assume $p(s_{t+1}|s_t, a_t)$ known

$$\mathrm{Q}^*(s_t, a_t) = r(s_t, a_t) + \gamma E\left[\max_{a_{t+1} \in A} Q^*(s_{t+1}, a_{t+1})\right]$$
$$= r(s_t, a_t) + \gamma \sum_{s_{t+1}} p(s_{t+1}|s_t, a_t) \max_{a_{t+1} \in A} \{\mathrm{Q}^*(s_{t+1}, a_{t+1})\}$$

Value Iteration-Model-free Approach

We don't know state transition probability $p(s_{t+1} | s_t, a_t)$.
We can use samples obtained from the agent's interaction with the environment

1. If we assume the state transition is deterministic, i.e. s_{t+1} is given
$$Q^*(s_t, a_t) \leftarrow r(s_t, a_t) + \gamma \max_{a_{t+1} \in A} \{Q^*(s_{t+1}, a_{t+1})\}$$

2. If we assume the state transition is probabilistic:
$$\begin{aligned} Q^*(s_t, a_t) &\leftarrow Q^*(s_t, a_t) + \alpha \left(r(s_t, a_t) + \gamma \max_{a_{t+1} \in A} \{Q^*(s_{t+1}, a_{t+1})\} - Q^*(s_t, a_t) \right) \\ &= Q^*(s_t, a_t)[1 - \alpha] + \alpha \left(r(s_t, a_t) + \gamma \max_{a_{t+1} \in A} \{Q^*(s_{t+1}, a_{t+1})\} \right) \end{aligned}$$

- **Model-free with deterministic transition**

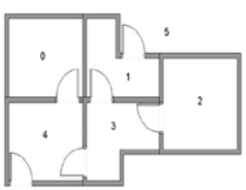
At state (room) s_t, take action a to go to s_{t+1} room and MUST arrive at room s_{t+1}, *i.e.*, $p(s_{t+1} | s_t, at) = 1$

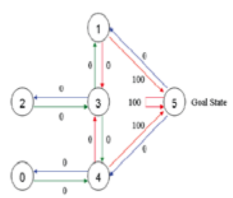
- **Model-based**

At state (room) s_t, take action a to go to s_{t+1} room, will arrive at room s_{t+1} with transition probability $p(s_{t+1} | s_t, a_t) \neq 1$

Example of Value iteration (Q-learning)

- A house has 5 rooms, the goal is to go out the house.
- If there is a door between two rooms, there exists an edge.
- The reward which directly points to outside is set to 100, others are set to 0.





Model-free approach

- Reward matrix and initial Q matrix

6 States: 0, 1, 2, 3, 4, 5
- 0-room 0, 1-room 1, .., 5-outside

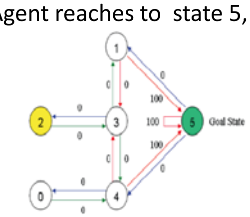
6 Actions: 0, 1, 2, 3,4, 5
- 0-go to room 0, 1-go to room 1, ..., 5-go outside

$$\begin{array}{c} \text{Action} \\ \begin{array}{cc} \text{State} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \end{array} \end{array}$$
$$R = \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix}$$

action, state:
$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

- Episode 1: Start from room 1, go to room 5.

Agent reaches to state 5, this episode is over.



action, state:
$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

$$\begin{aligned} Q(1,5) &= R(1,5) + 0.8 * \max\{Q(5,1), Q(5,4), Q(5,5)\} \\ &= 100 + 0.8 * \max\{0,0,0\} \\ &= 100. \end{aligned}$$

- Reward matrix and initial Q matrix

$$R=\begin{array}{c} \text{State} \\ 0\\1\\2\\3\\4\\5 \end{array}\overset{\text{Action}}{\begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}} \qquad Q=\begin{array}{c} \\ 0\\1\\2\\3\\4\\5 \end{array}\begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

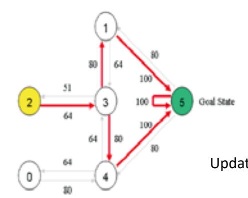
- Episode 2: Start from room 3, go to room 1

$$\begin{aligned} Q(3,1) &= R(3,1) + 0.8 * \max\{Q(1,3), Q(1,5)\} \\ &= 0 + 0.8 * \max\{0, 100\} \\ &= 80. \end{aligned} \qquad Q=\begin{array}{c} \\ 0\\1\\2\\3\\4\\5 \end{array}\begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

- After several episodes, Q will converge to

Normalize: /500

$$Q=\begin{array}{c} \\ 0\\1\\2\\3\\4\\5 \end{array}\begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 400 & 0 \\ 0 & 0 & 0 & 320 & 0 & 500 \\ 0 & 0 & 0 & 320 & 0 & 0 \\ 0 & 400 & 256 & 0 & 400 & 0 \\ 320 & 0 & 0 & 320 & 0 & 500 \\ 0 & 400 & 0 & 0 & 400 & 500 \end{bmatrix} \end{array} \Rightarrow Q=\begin{array}{c} \\ 0\\1\\2\\3\\4\\5 \end{array}\begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 80 & 0 \\ 0 & 0 & 0 & 64 & 0 & 100 \\ 0 & 0 & 0 & 64 & 0 & 0 \\ 0 & 80 & 51 & 0 & 80 & 0 \\ 64 & 0 & 0 & 64 & 0 & 100 \\ 0 & 80 & 0 & 0 & 80 & 100 \end{bmatrix} \end{array}$$



Updated reward functions from Q

- Reward matrix and initial Q matrix

$$R=\begin{array}{c} \text{State} \\ 0\\1\\2\\3\\4\\5 \end{array}\overset{\text{Action}}{\begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}} \qquad Q=\begin{array}{c} \\ 0\\1\\2\\3\\4\\5 \end{array}\begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

- Episode 3: at room 1, go to room 5

Agent reaches to room 5, this episode is over.

$$\begin{aligned} Q(1,5) &= R(1,5) + 0.8 * \max\{Q(5,1), Q(5,4), Q(5,5)\} \\ &= 100 + 0.8 * \max\{0, 0, 0\} \\ &= 100. \end{aligned} \qquad Q=\begin{array}{c} \\ 0\\1\\2\\3\\4\\5 \end{array}\begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Model-based Method

- Reward matrix and initial Q matrix

$$R=\begin{array}{c} \text{State} \\ 0\\1\\2\\3\\4\\5 \end{array}\overset{\text{Action}}{\begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}} \qquad Q=\begin{array}{c} \\ 0\\1\\2\\3\\4\\5 \end{array}\begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

$P(s_{t+1}=5 \mid s_t=1, a_t=5)=0.8$

$P(s_{t+1}=3 \mid s_t=1, a_t=5)=0.2$

- Episode 1: Start from room 1, go to room 5

Agent reaches to room 5, this episode is over.

$$\begin{aligned} Q(1,5) = R(1,5) &+ 0.8 * 0.8 * max\{Q(5,1), Q(5,4), Q(5,5)\} \\ &+ 0.2 * 0.8 * max\{Q(3,1), Q(3,2), Q(3,5)\} \\ = 100 &+ 0.8 * 0.8 * \max\{0,0,0\} + 0.2 * 0.8 * \max\{0,0,0\} \\ &= 100 \end{aligned} \qquad Q=\begin{array}{c} \\ 0\\1\\2\\3\\4\\5 \end{array}\begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Deep Reinforcement Learning

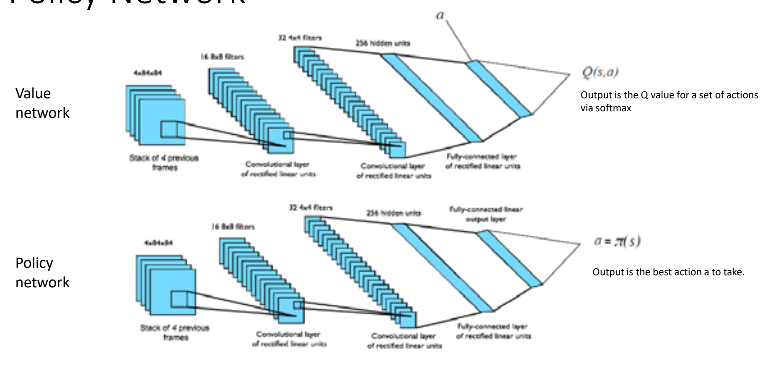
Use deep neural networks to represent:

1) Q-function--- Value-Based Deep RL, i.e. deep Q-network
2) Policy--- Policy-Based Deep RL ,i.e., deep policy network

Advantage:

1) Solve large problems, *eg.* Go game
2) States or actions are in large space or even continues
3) Nonlinear function approximator

Framework of Deep value network and Deep Policy Network



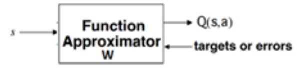
Slides from David Silver, Google DeepMind

Value-Based *vs* Policy-Based

- Value-Based
  1) Learn value function (Q function)
  2) Implicit policy-derive best policy π from Q
- Policy-Based
  1) No value function Q
  2) Learn best policy π directly

Value-Based Deep RL

- Represent Q function by a deep Q-network with weights ***w****, input s (images), and output* $Q(s, a)$



- Define an objective function by mean-squared error in Q-value between two consecutive times with Bellman equation

$$L(\boldsymbol{w}) = \sum_t E\left[\left(r(s_t, a_t) + \gamma \max_{a_{t+1} \in A}\left\{Q\left(s_{t+1}\ \ , a_{t+1}\ \ , \boldsymbol{w}\right)\right\} - Q(s_t, a_t, \boldsymbol{w})\right)^2\right]$$
$$w *= \arg min_w L(w)$$

Essentially, the network learns Bellman recursive updating equation

Policy-Based Deep RL :Policy gradient method

- Represent policy by a deep neural network

  Approximate policy $a = \pi(s, \boldsymbol{w})$ input: state s, output action a, **w** NN parameters

- Define objective function as total expected reward (i.e., Q value) and learn the policy π that maximizes the expected reward

$$Q = E[r_1 + \gamma^1 r_2 + \gamma^2 r_3 + \cdots]$$

- Intuitions: collect a bunch of trajectories (sequences of actions), and find the π that makes the good trajectories more probable

Advantages and disadvantages of Policy-based RL

Advantages

1) Better convergence properties
2) Effective in high-dimensional or continuous action spaces
3) Can learn stochastic policies

Disadvantages

1) Typically converge to a local optimum
2) Inefficient

Slides from David Silver, Google DeepMind

Policy Gradient Method

- Let τ represent a whole trajectory:

$$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_{T-1}, a_{T-1}, r_{T-1})$$

- Let expected R(τ) represent the reward for τ. Given N trajectories, $\tau_1, \tau_2, .., \tau_N$, the total expected rewards for N trajectories are

$$R(\mathbf{w}) = \sum_{i=1}^{N} p(\tau_i \mid \mathbf{w}) R(\tau_i)$$

where p(τ|w) is the probability for trajectory τ , which can be written as a function of the parameters of the policy function π

- **w** can be derived from $R(\mathbf{w}) = \sum_{i=1}^{N} \underbrace{\sum_{t_i=1}^{T_i} \gamma^{t_i} r_{t_i}}_{R(\tau_i)} \underbrace{\sum_{t_i=1}^{T_i} \log\pi(a_{t_i} \mid s_{t_i}, \mathbf{w})}_{p(\tau_i \mid \mathbf{w})}$

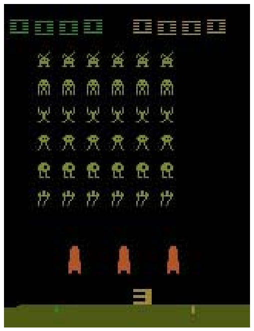
$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmax}} R(\mathbf{w})$$

- Interpretation: using good trajectories (high R) as supervised examples in classification / regression

Slides from John Schulman, OpenAI

Applications

Atari Game (from pixels to actions)



- State is few consecutive images
- Q-learning is used for finding optimal policy
- Deep neural network approximates action-value function => need to train this network

Input = 4 last frames

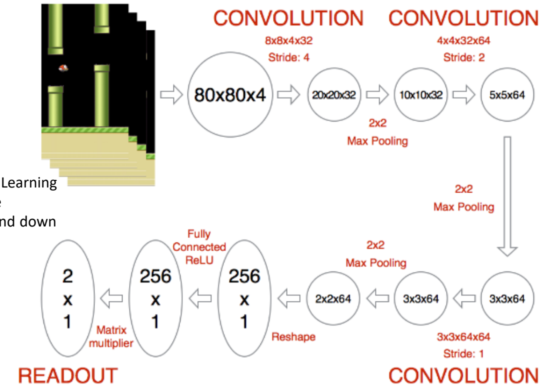
Output = Q values for all possible actions

Reward=the score

Goal=learn the strategy to maximize the display scores

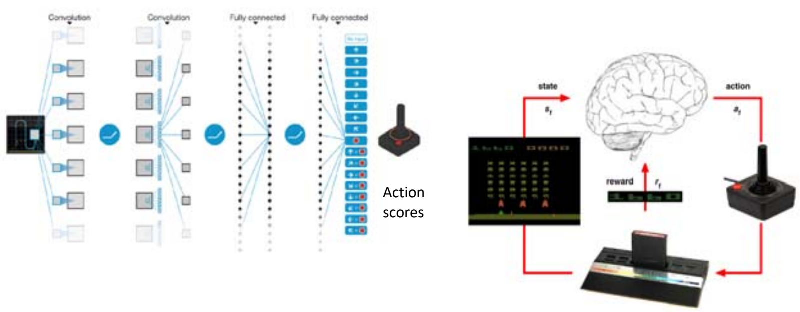
Deep Reinforcement Learning for Flappy Bird

- Policy-based Learning
- Input: image
- Output: up and down action



Youtube demo: https://www.youtube.com/watch?v=THhUXIhjkCM

Atari Game Framework



Youtube demo: the video shows the agent learn a surprising strategy to breakout in the end
https://www.youtube.com/watch?v=V1eYniJ0Rnk

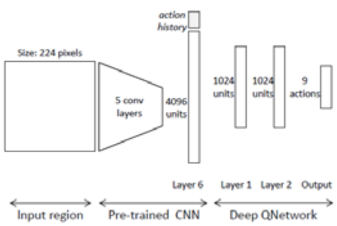
Robotics and control

State: sensor reading
Action: torque at joints
Rewards: navigate to target location

Youtube demo:
https://www.youtube.com/watch?v=SHLuf2ZBQSw&feature=youtu.be

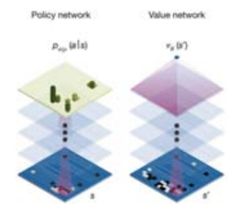
Slides from John Schulman

Object Localization



- Decides how to transform the bounding box to progressively localize the object.
- State: feature vector of the observed region and the history of taken actions
- Rewards: IOU between detection and ground truth
- Action:

AlphaGo



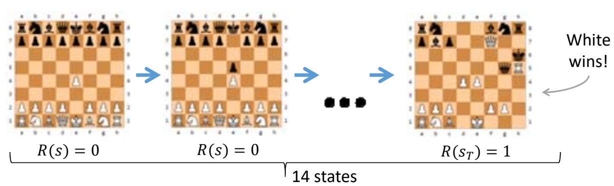
- The reward is to win the game.
- MCMC sampling is used to decide which state to go next.
- Simulation with self-play allows identifying the best unexplored path given current state. It allows to learn on demand and on the fly.

Silver D, Huang A, Maddison C J, et al. Mastering the game of Go with deep neural networks and tree search[J]. Nature, 2016, 529(7587): 484-489.

https://www.tastehit.com/blog/google-deepmind-alphago-how-it-works/

Chess

- States: board
- Actions: legal moves
- Reward:
  - $\forall s \neq s_T: R(s) = 0$
  - $R(s_T) = \pm 1$ (win/loss)



AlphaGo Zero

- Google DeepMind introduced AlphaGo Zero in 2017.
- AlphaGo Zero training is solely based on game rules through self-play, without using any prior human plays
- It achieves super-human performance by beating AlphaGo Lee by 100:0 after 3 days training.

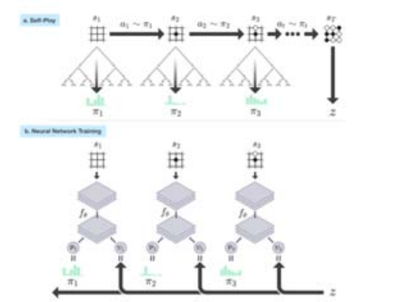


Figure 1: **Self-play reinforcement learning in *AlphaGo Zero*. a** The program plays a game $s_1, \ldots, s_T$ against itself.