

DEEP LEARNING

Lecture 10: Deep Reinforcement Learning

Dr. Yang Lu

Department of Computer Science and Technology

luyang@xmu.edu.cn



Supervised Learning

- Given data: (x, y) , x is data, y is label.
- Goal: Learn a function to map $x \rightarrow y$, namely posterior probability

$$P(Y|X = x)$$

- Examples: Classification, regression, object detection, face recognition, sentiment classification, etc.



→ cat

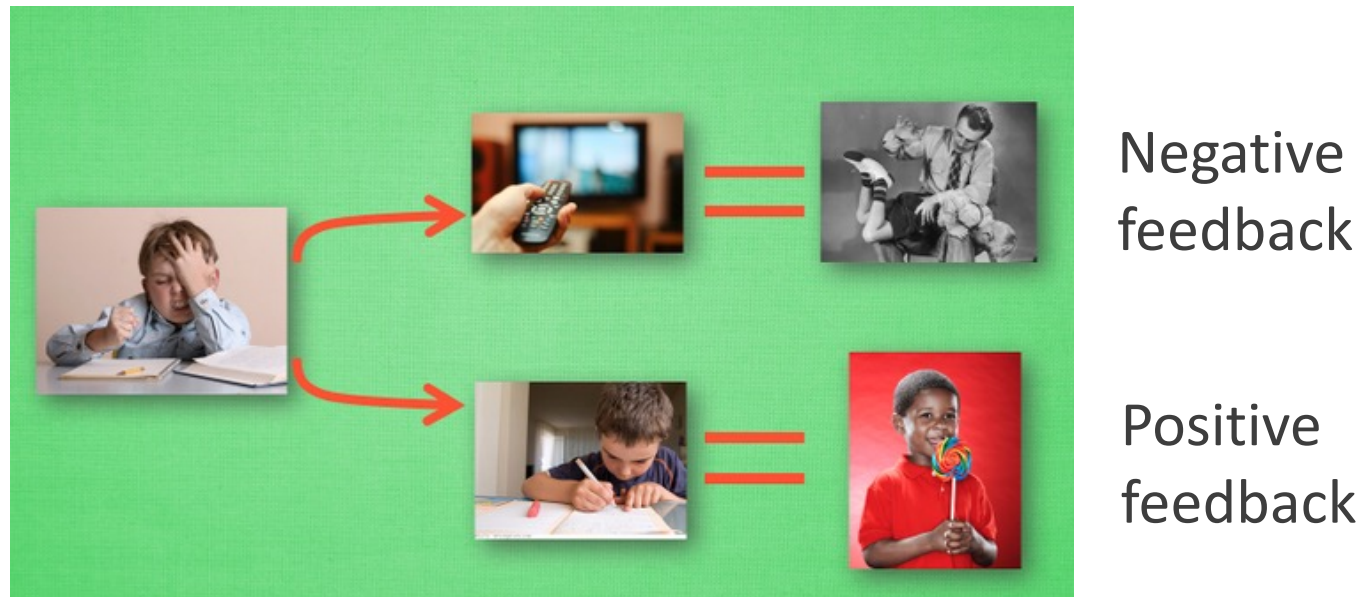


→ car



Reinforcement Learning

- When we are kids, nobody gives us training data for good behaviors and bad behaviors.
- We learn by **trial and error** with feedbacks from parents and teachers.

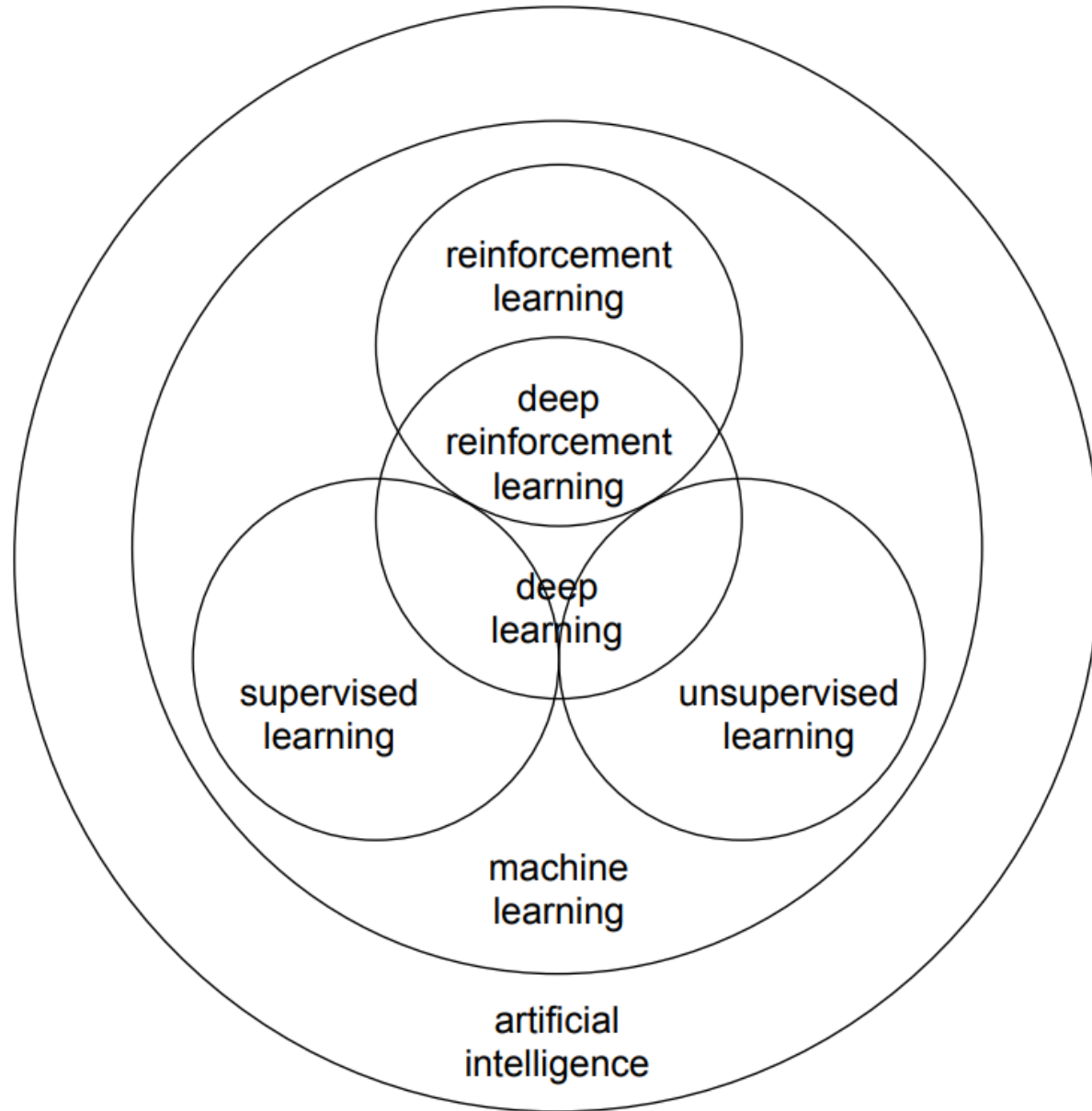


Reinforcement Learning

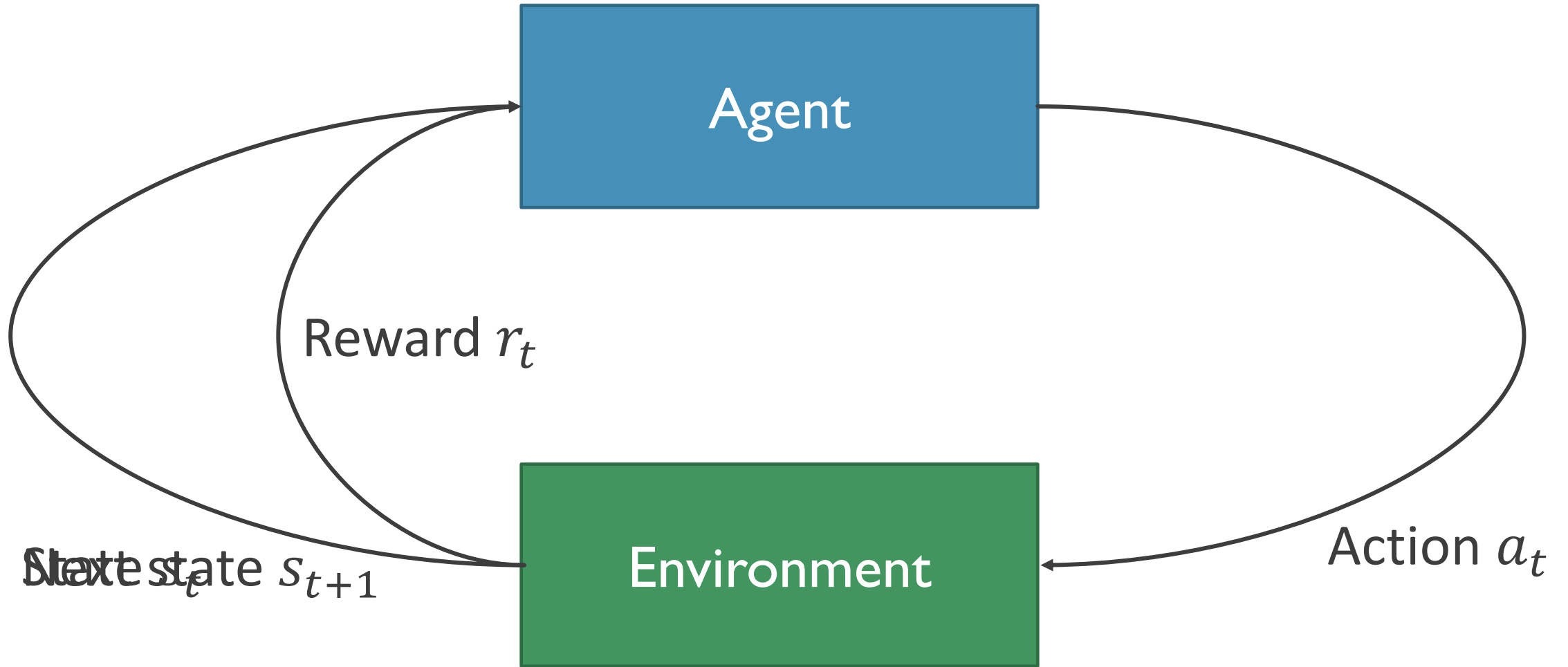
What makes reinforcement learning different from other machine learning paradigms?

- There is **no supervisor**, only a **reward** signal.
- Feedback is **delayed**, not instantaneous.
- **Time** really matters (sequential, non i.i.d data).
- Agent's actions affect the subsequent data it receives.





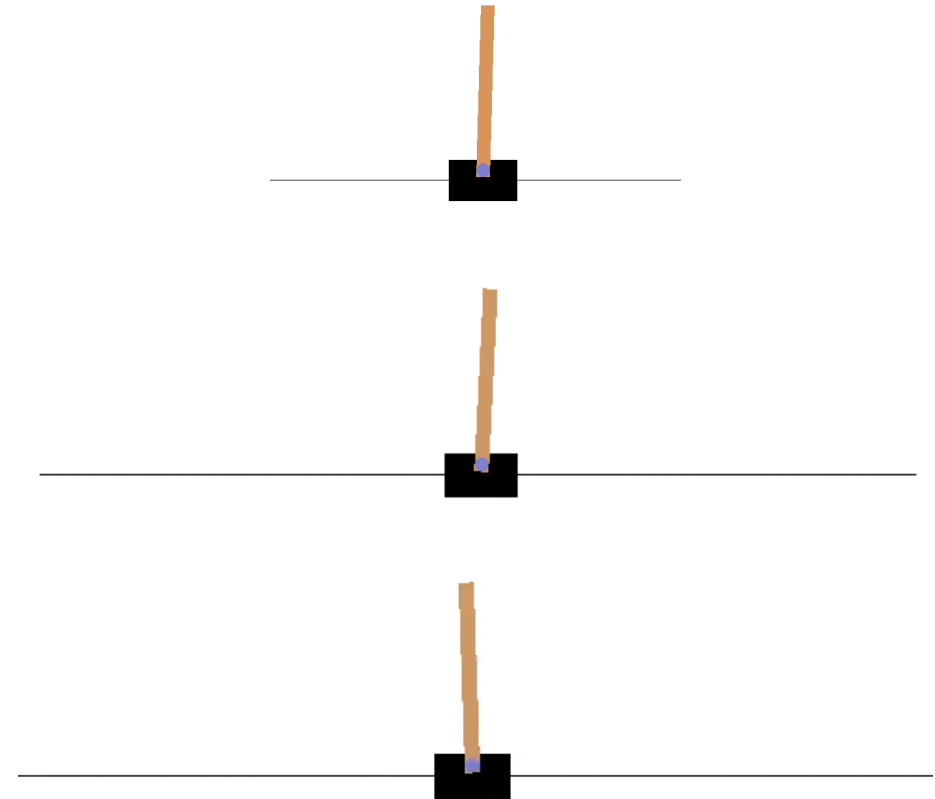
Reinforcement Learning



Reinforcement Learning

Cart-Pole Problem

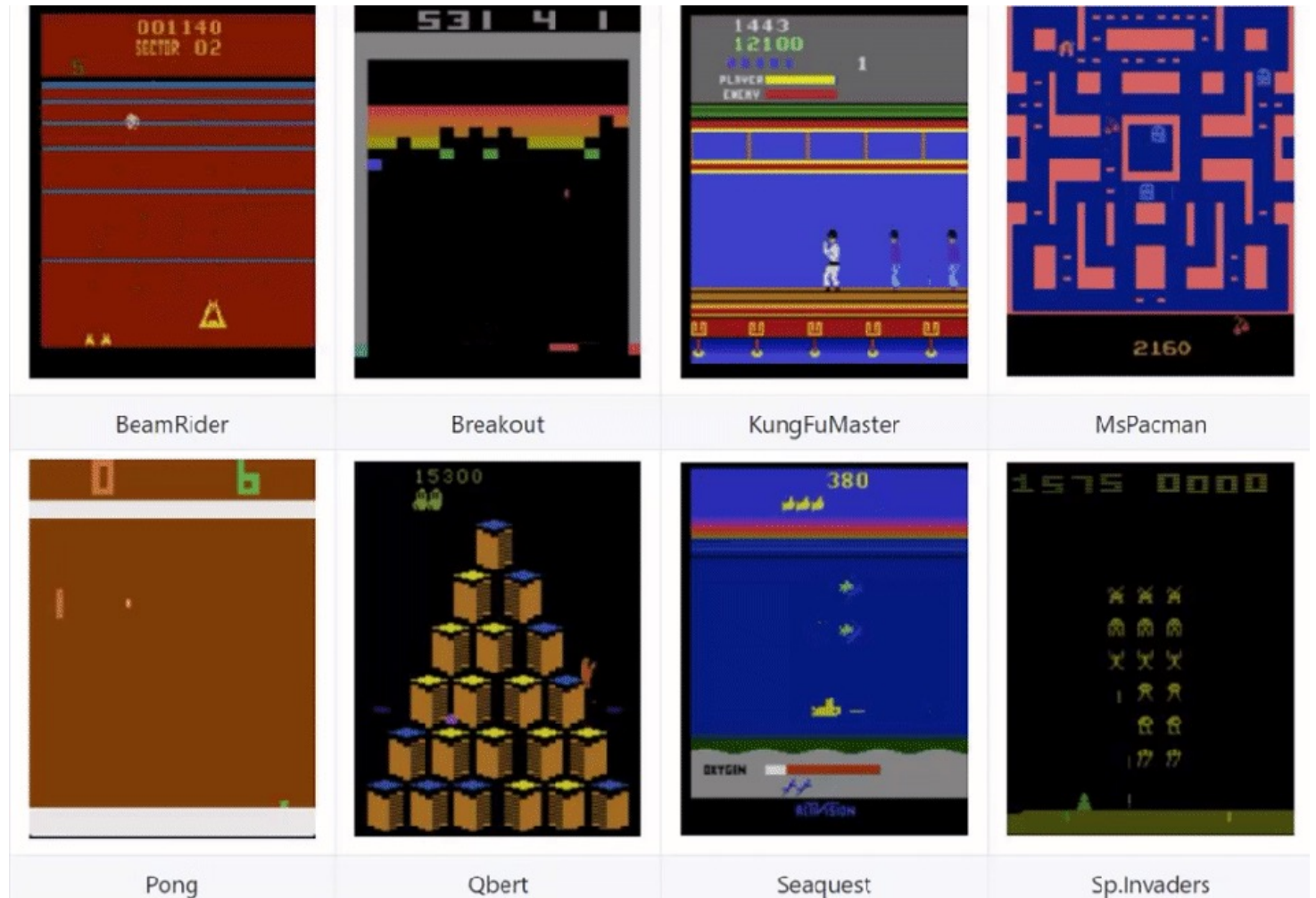
- **Objective:** Balance a pole on top of a movable cart.
- **State:** angle, angular speed, position, horizontal velocity
- **Action:** horizontal force applied on the cart.
- **Reward:** 1 at each time step if the pole is upright.



Reinforcement Learning

Atari Games

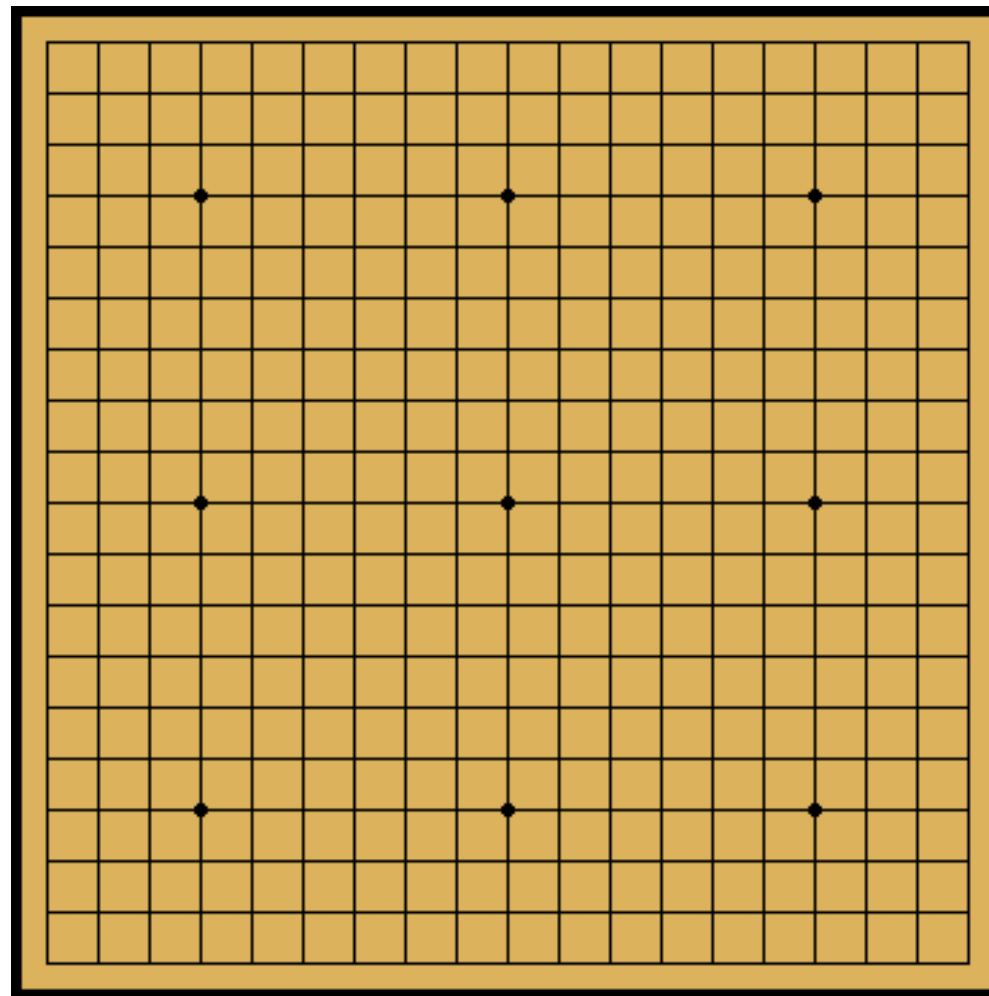
- **Objective:** Complete the game with the highest score.
- **State:** Raw pixel inputs of the game state.
- **Action:** Game controls e.g. Left, Right, Up, Down.
- **Reward:** Score increase / decrease at each time step.



Reinforcement Learning

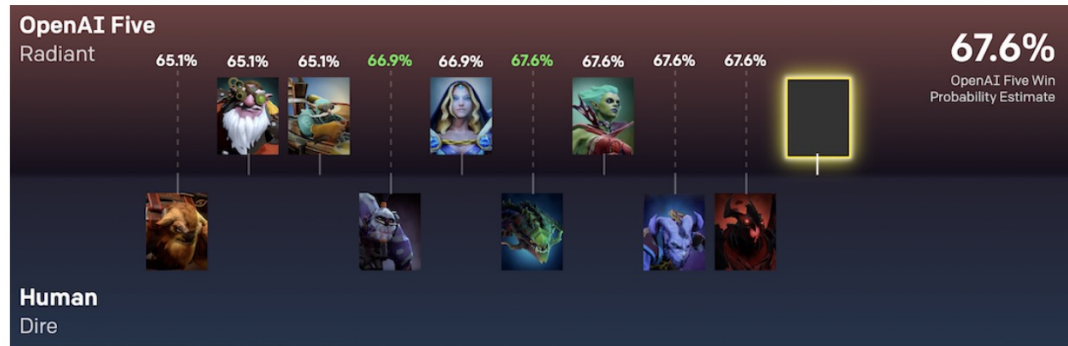
Go

- **Objective:** Win the game.
- **State:** Position of all pieces.
- **Action:** Where to put the next piece down.
- **Reward:** 1 if win at the end of the game, 0 otherwise.



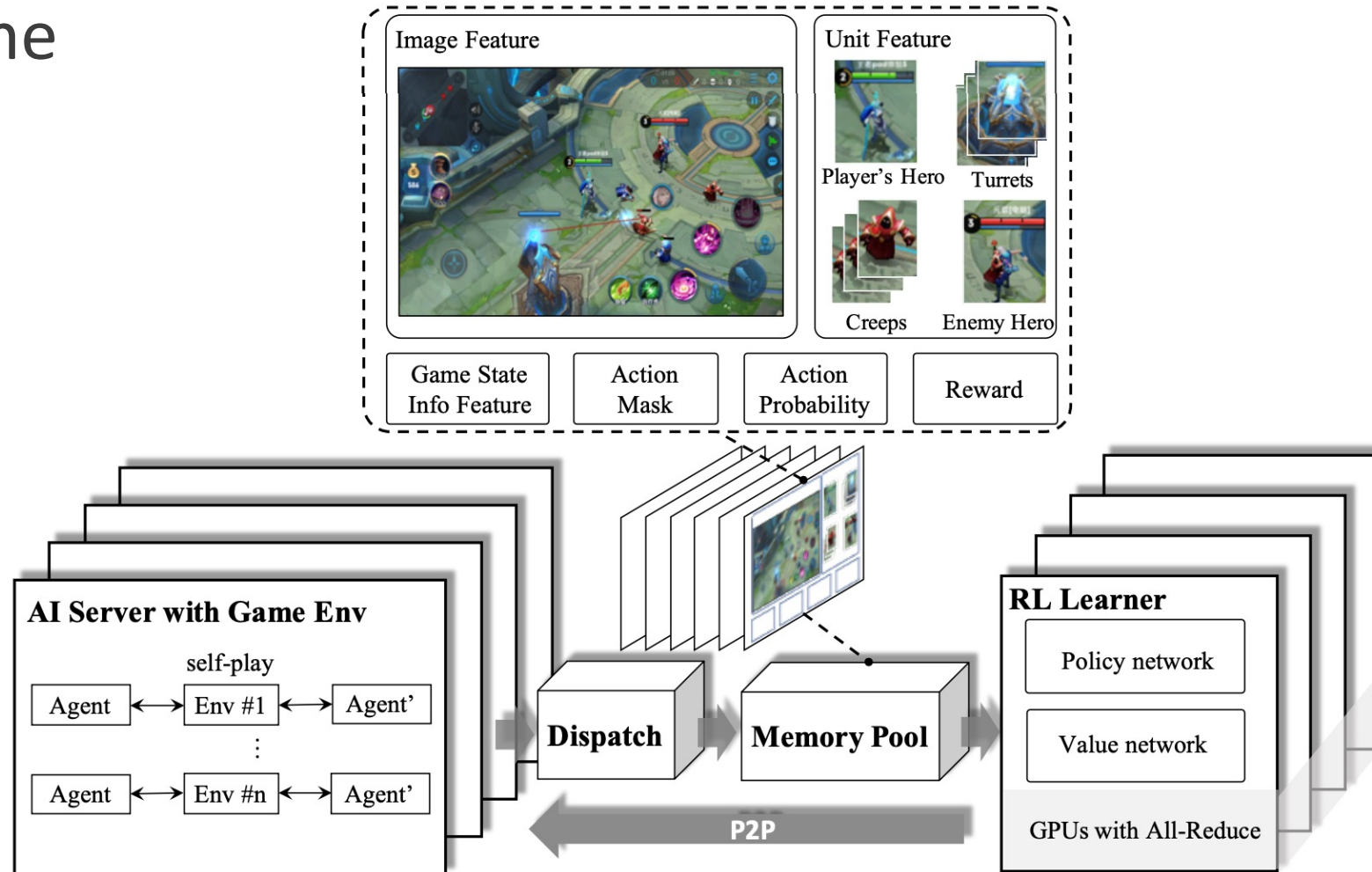
Applications of Reinforcement Learning

■ Play game



Applications of Reinforcement Learning

■ Play game



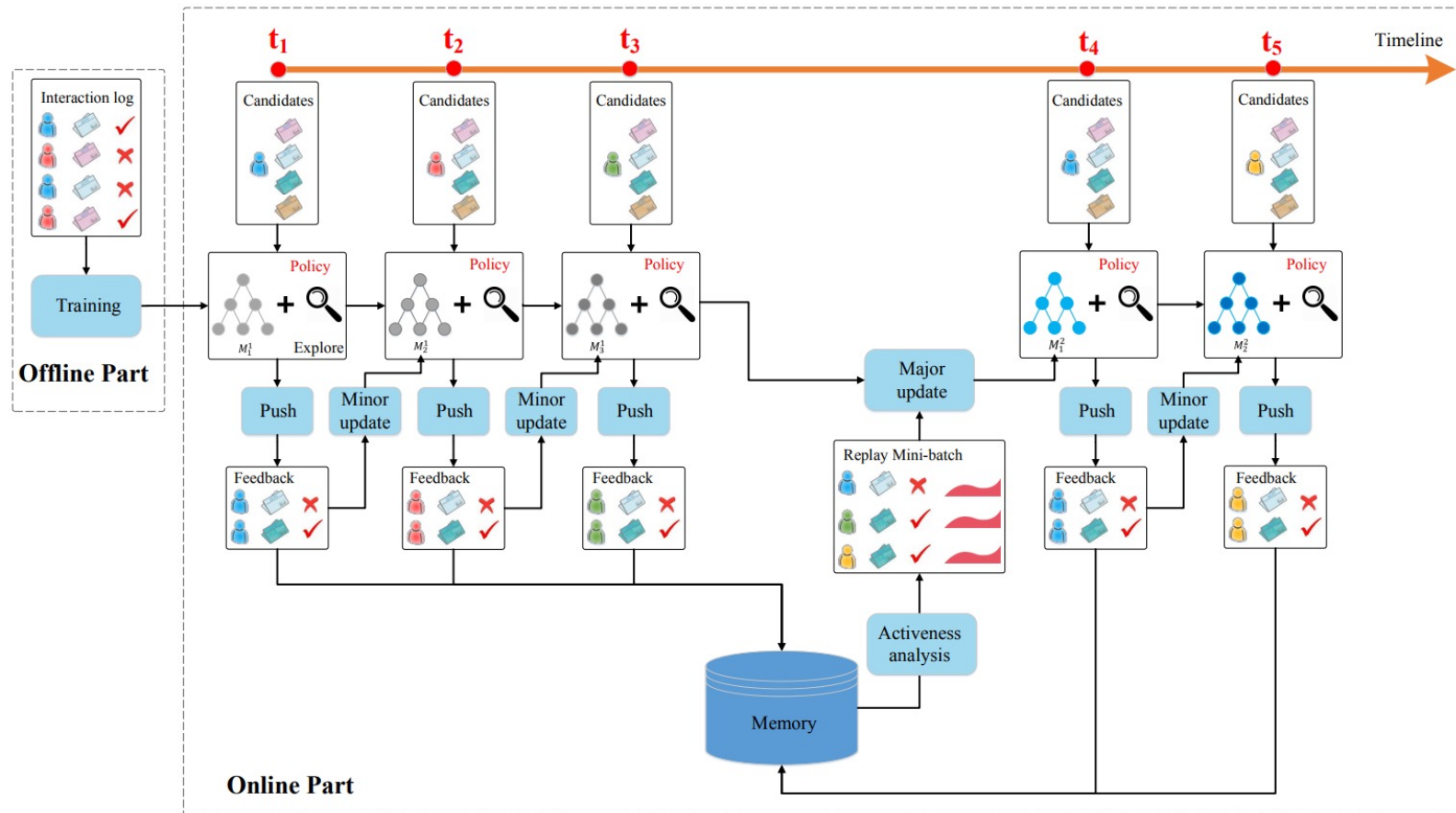
Applications of Reinforcement Learning

- AlphaGo [DeepMind, Nature 2016]:
 - Required many engineering tricks
 - Bootstrapped from human play
 - Beat 18-time world champion Lee Sedol
- AlphaGo Zero [Nature 2017]:
 - Simplified and elegant version of AlphaGo
 - No longer bootstrapped from human play
 - Beat (at the time) #1 world ranked Ke Jie
- Alpha Zero (Dec. 2017)
 - Generalized to beat world champion programs on chess and shogi as well
- MuZero (Nov. 2019)
 - Plans through a learned model of the game



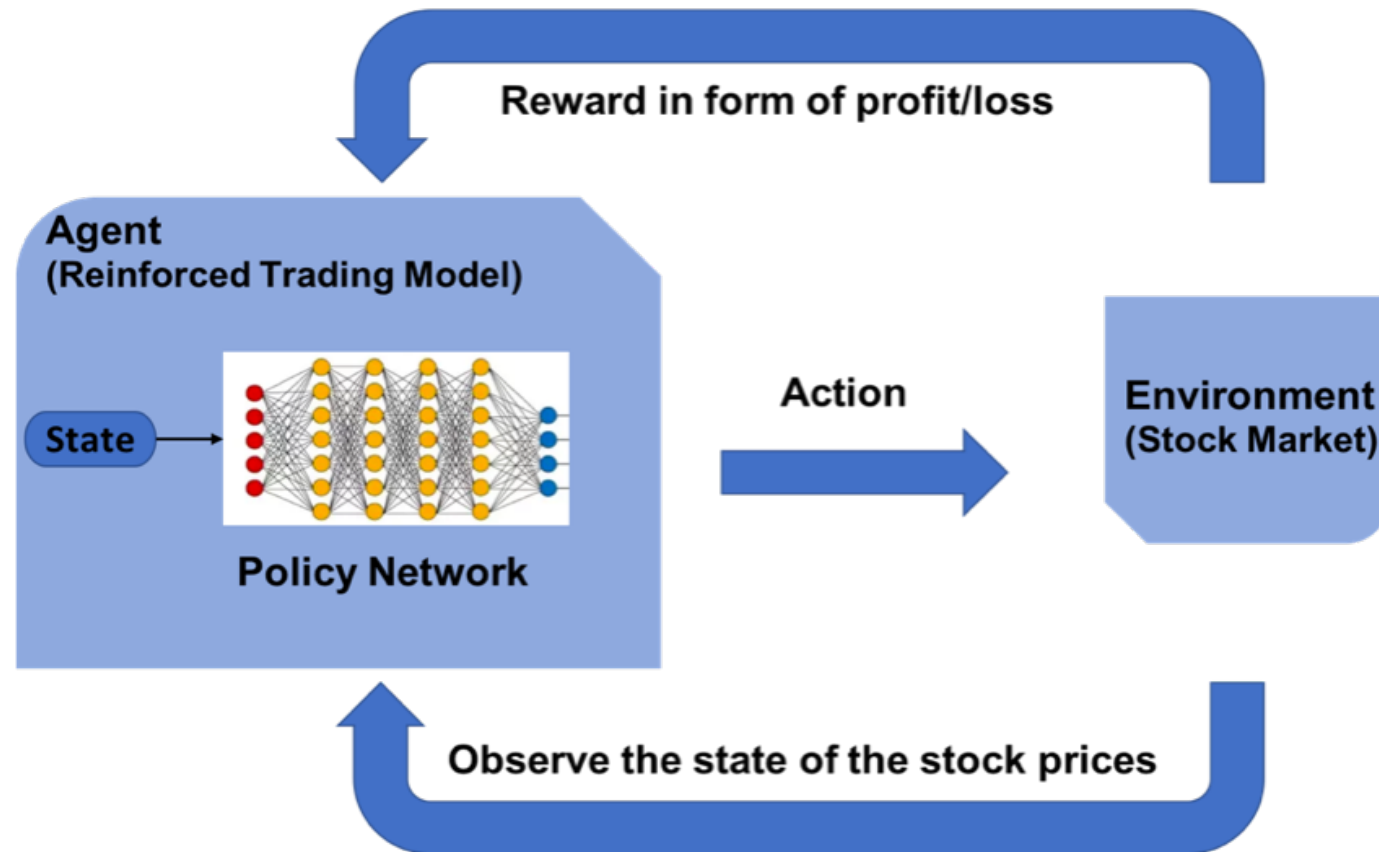
Applications of Reinforcement Learning

Recommender system



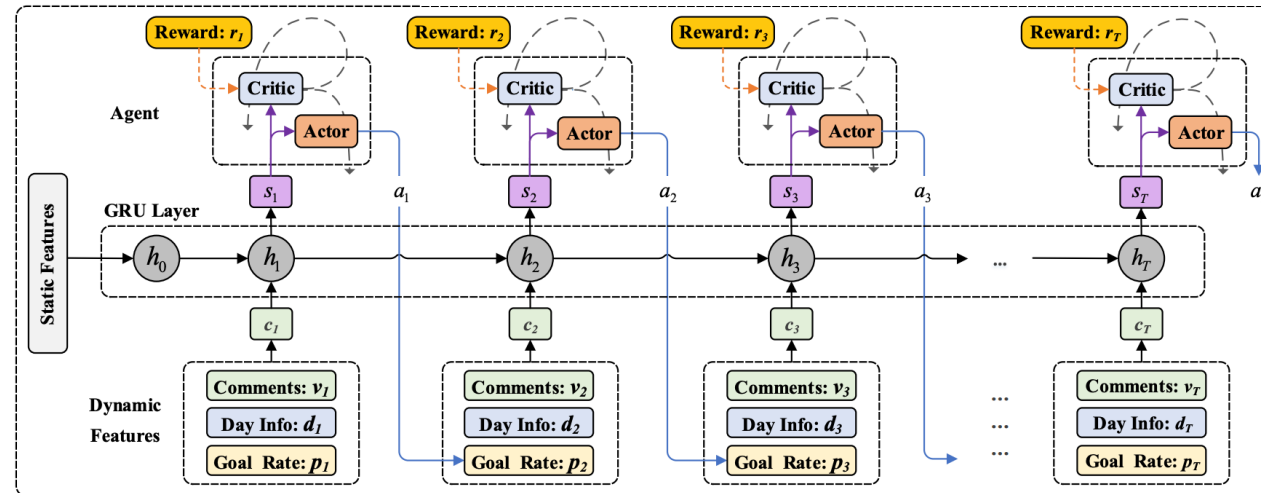
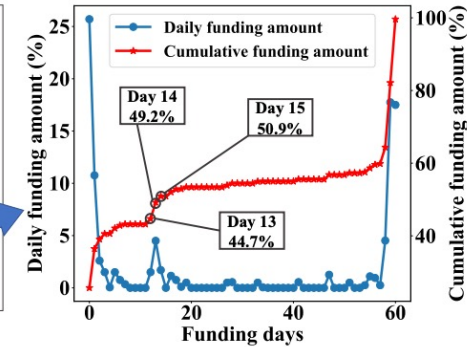
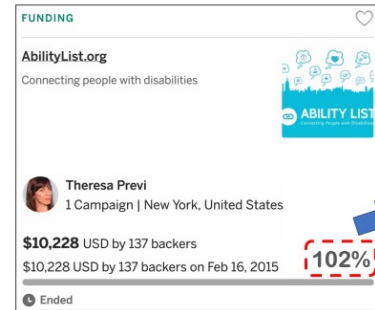
Applications of Reinforcement Learning

■ Financial trading



Applications of Reinforcement Learning

Business analysis

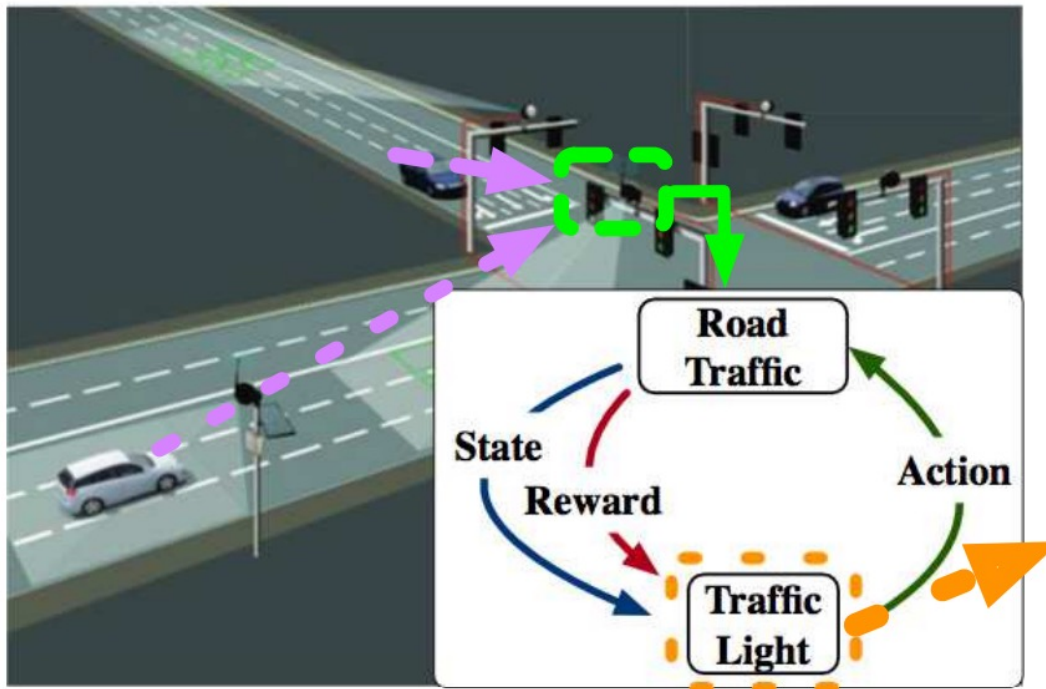


Crowdfunding dynamics tracking

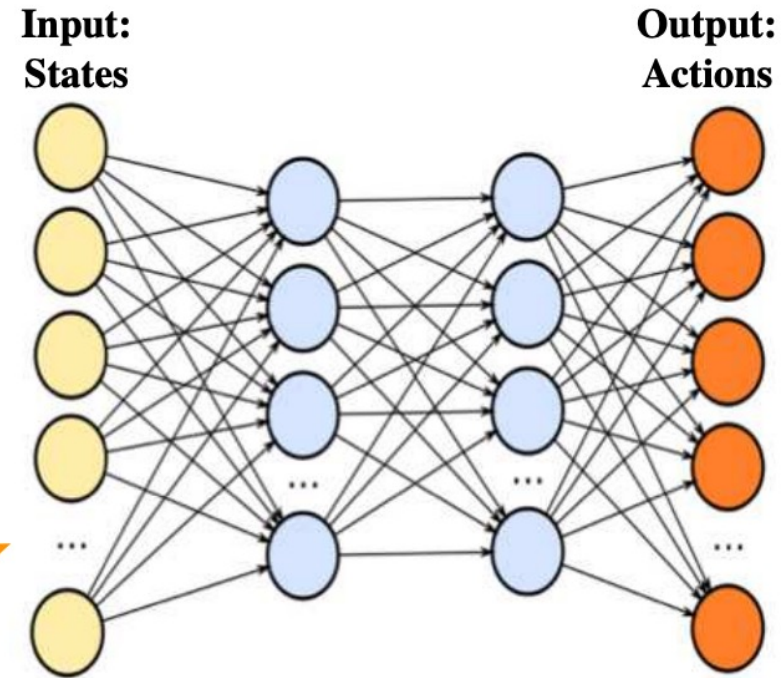


Applications of Reinforcement Learning

Smart transportation



Reinforcement Learning



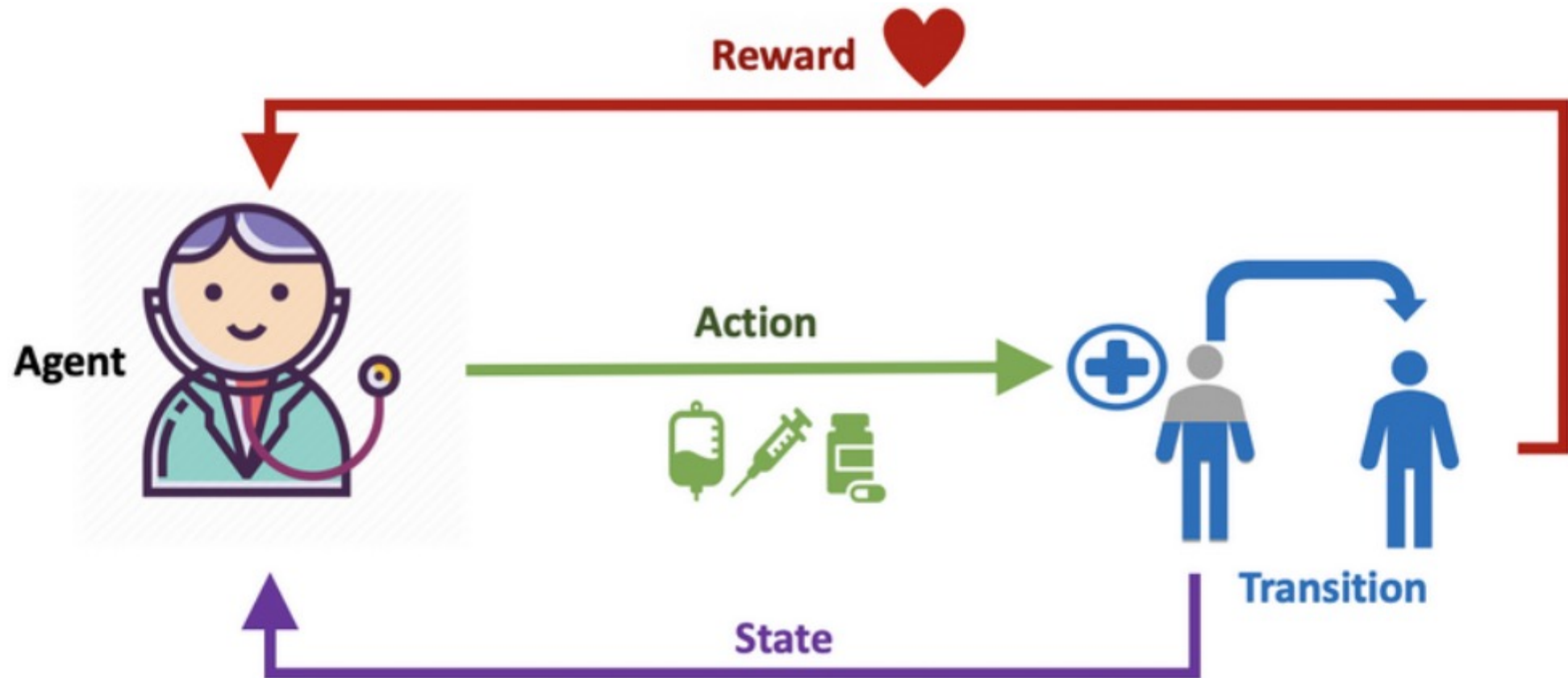
Deep Learning

Traffic light control model



Applications of Reinforcement Learning

Healthcare



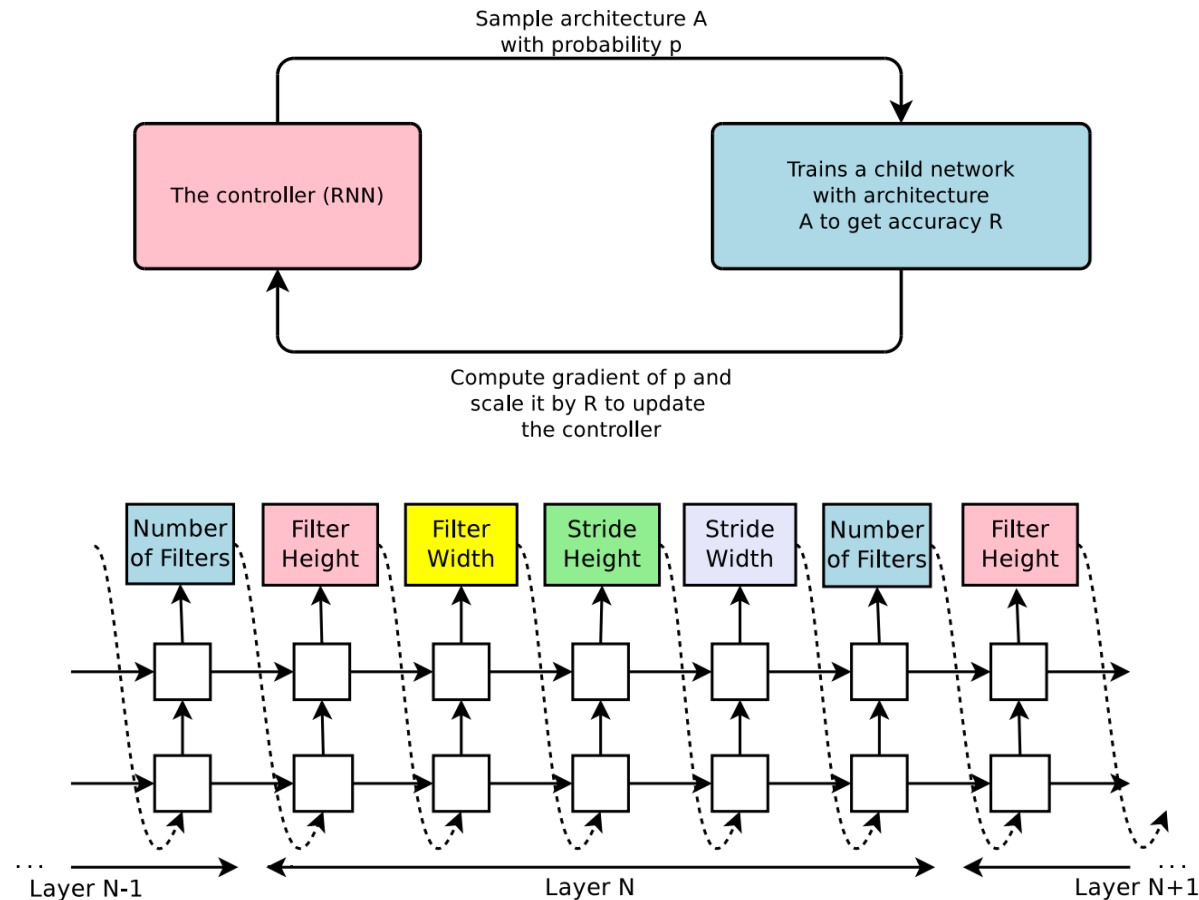
Applications of Reinforcement Learning

■ Robotics



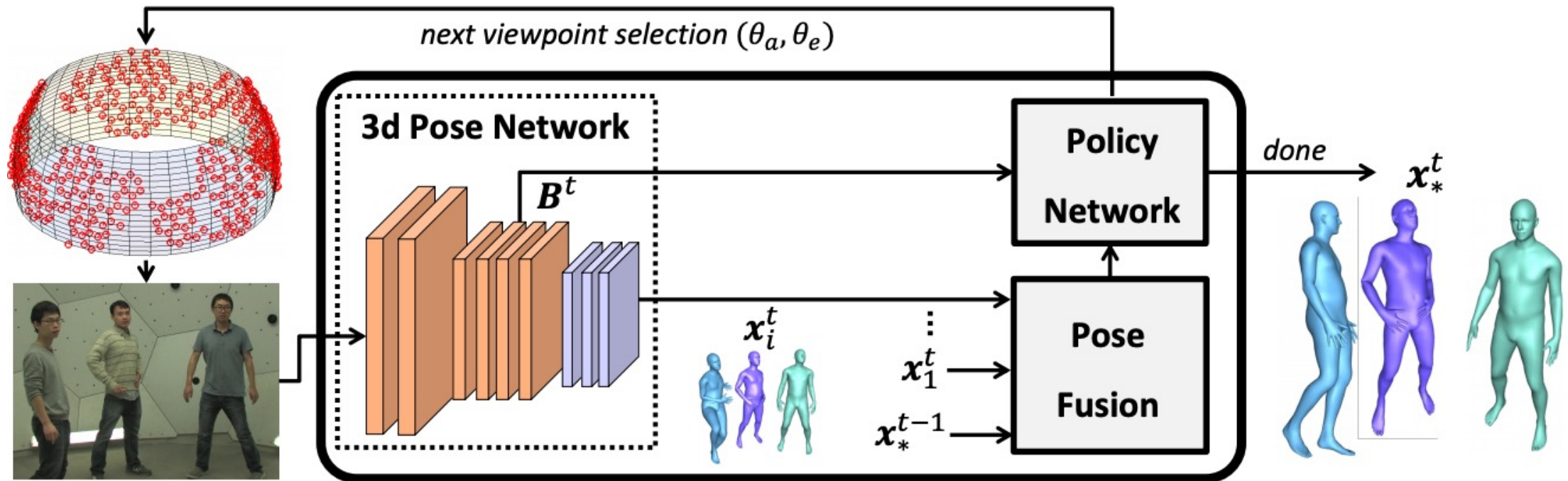
Applications of Reinforcement Learning

■ Network architecture search



Applications of Reinforcement Learning

■ CV

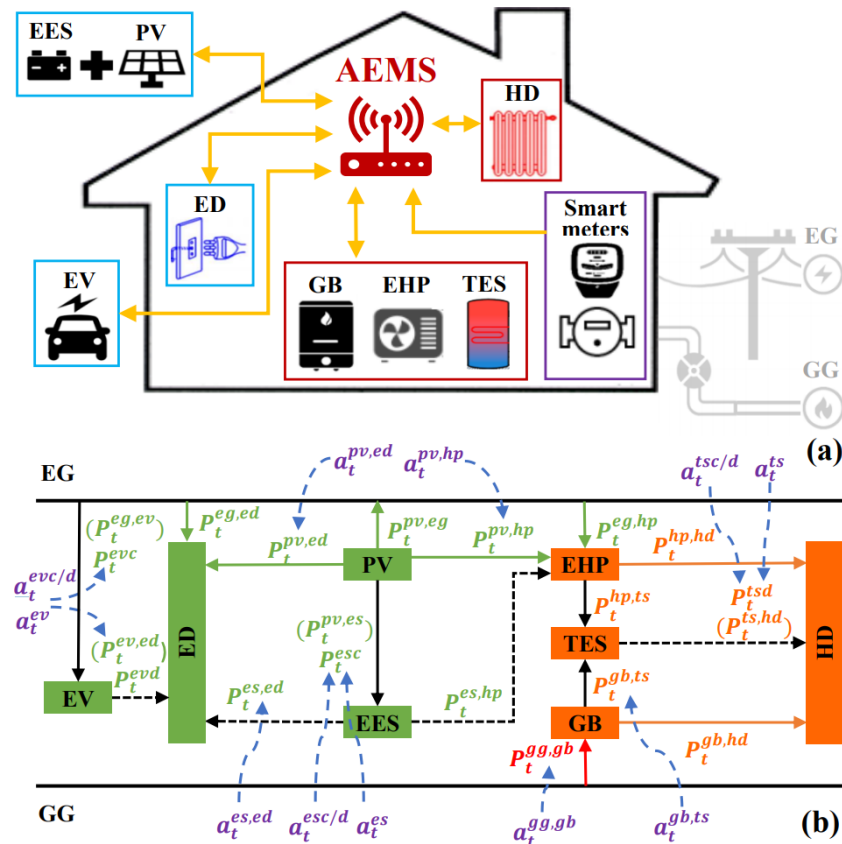


Active human pose estimation



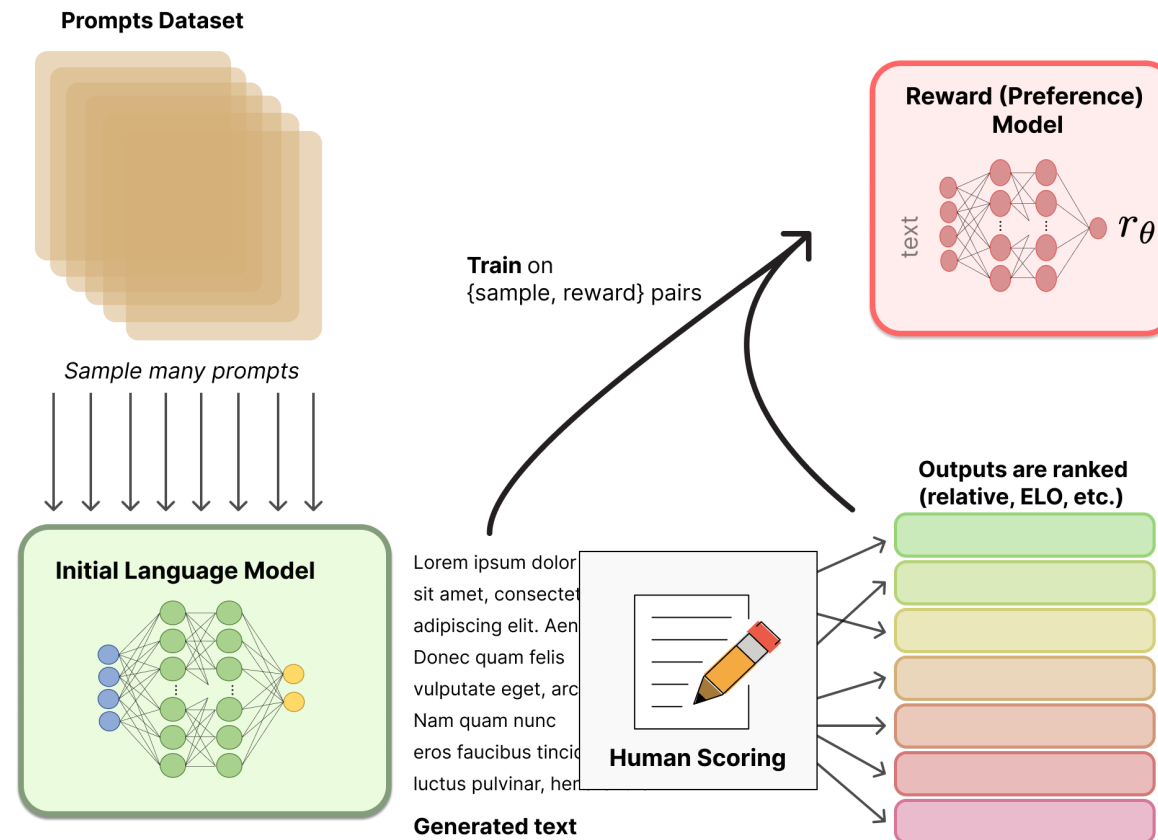
Applications of Reinforcement Learning

■ Energy management



Applications of Reinforcement Learning

- ChatGPT (by reinforcement learning from human feedback)



Outlines

- Markov Decision Process
- Q-Learning
- Deep Q Network
- Policy Gradient
- Actor-Critic



MARKOV DECISION PROCESS

Markov Decision Process

- Mathematical formulation of the RL problem.
- A **Markov decision process (MDP)** is a Markov reward process with decisions. It is an environment in which all states are Markov.

Definition

A **Markov decision process** is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- \mathcal{S} is a finite set of states.
- \mathcal{A} is a finite set of actions.
- \mathcal{P} is a state transition probability matrix,
$$\mathcal{P}_{s \rightarrow s'}^a = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$$
- \mathcal{R} is a reward function, $\mathcal{R}_s^a = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$.
- γ is a discount factor $\gamma \in [0, 1]$.



Markov Decision Process

Definition

A **policy** π is a distribution over actions given states,

$$\pi(a|s) = P[A_t = a | S_t = s]$$

- A policy fully defines the behavior of an agent.
- MDP policies depend on the current state (not the history).

Policy is the thing that we should learn in RL. In short, what should I do on the current state.

Return

Definition

The **return** G_t is the total discounted reward from time-step t .

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}.$$

- The discount $\gamma \in [0,1]$ is the present value of future rewards.
- This values immediate reward above delayed reward.
 - γ close to 0 leads to “myopic” evaluation.
 - γ close to 1 leads to “far-sighted” evaluation.
- Objective: find policy π^* that **maximizes the return**.



Return

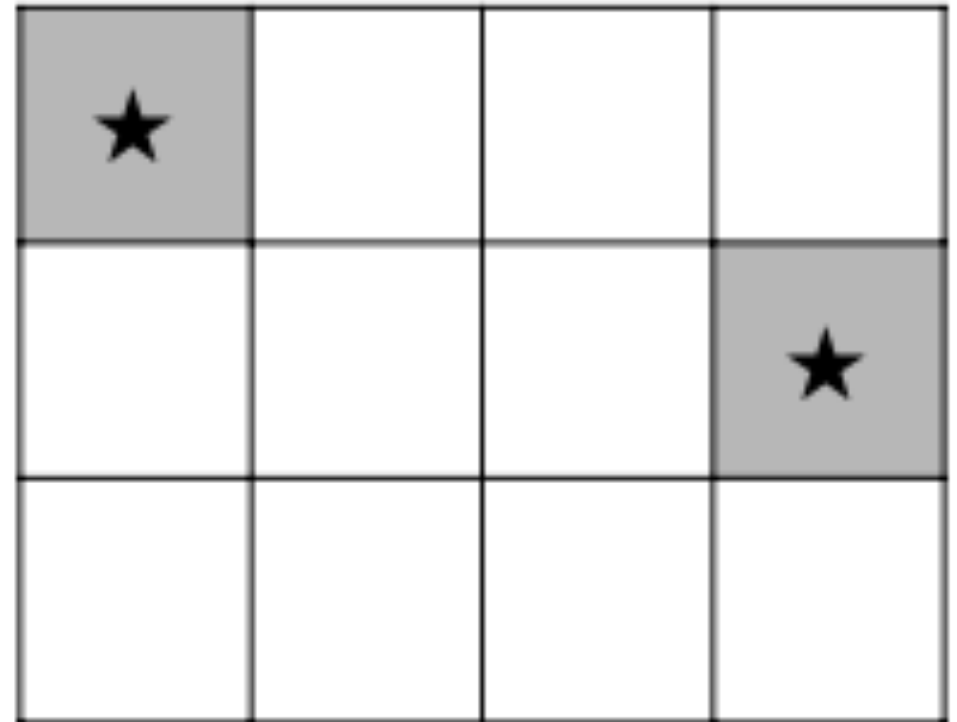
- Most Markov reward and decision processes are discounted. Why?
- Mathematically convenient to discount rewards.
 - Avoids infinite returns in cyclic Markov processes.
 - Uncertainty about the future may not be fully represented.
 - If the reward is financial, immediate rewards may earn more interest than delayed rewards.
 - Animal/human behavior shows preference for immediate reward.



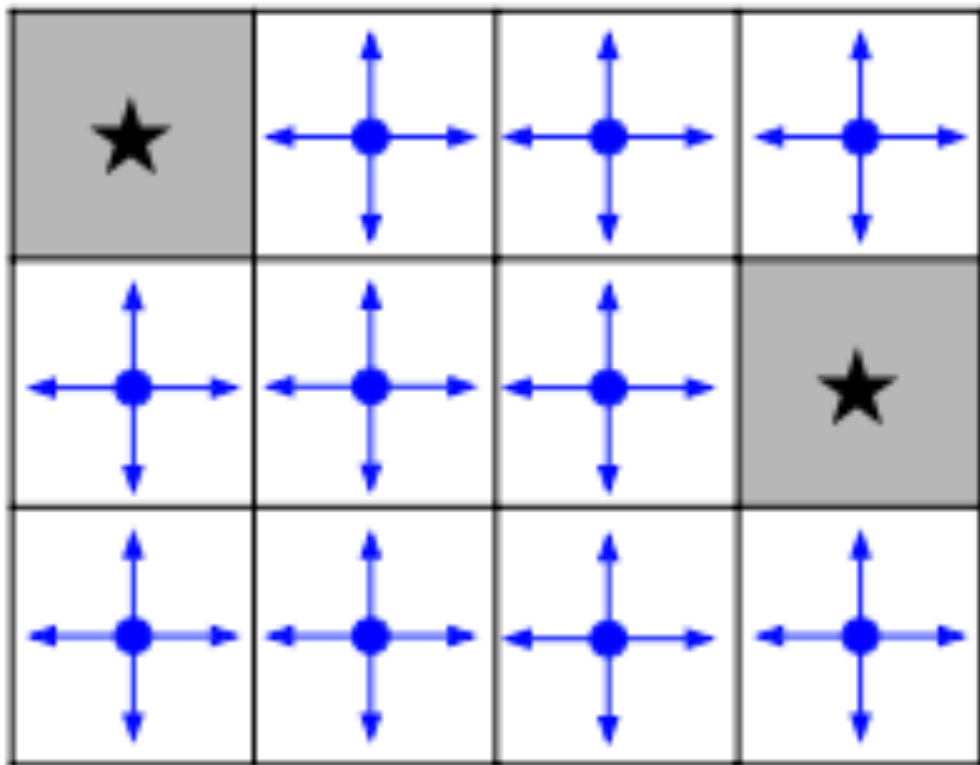
A Simple MDP Example

- Actions = {right, left, up, down}.
- Reward is set at -1 for each transition.
- Objective: reach one of terminal states (greyed out) in least number of actions.

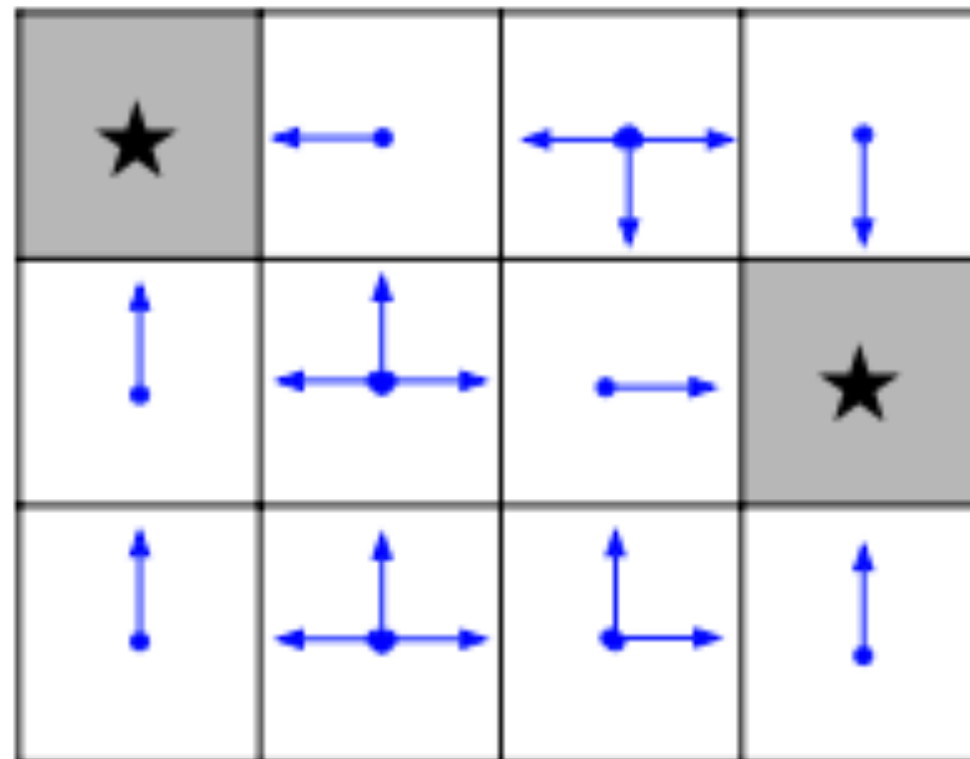
states



A Simple MDP Example



Random policy



Optimal policy



Q-LEARNING

Value Function and Q-Value Function

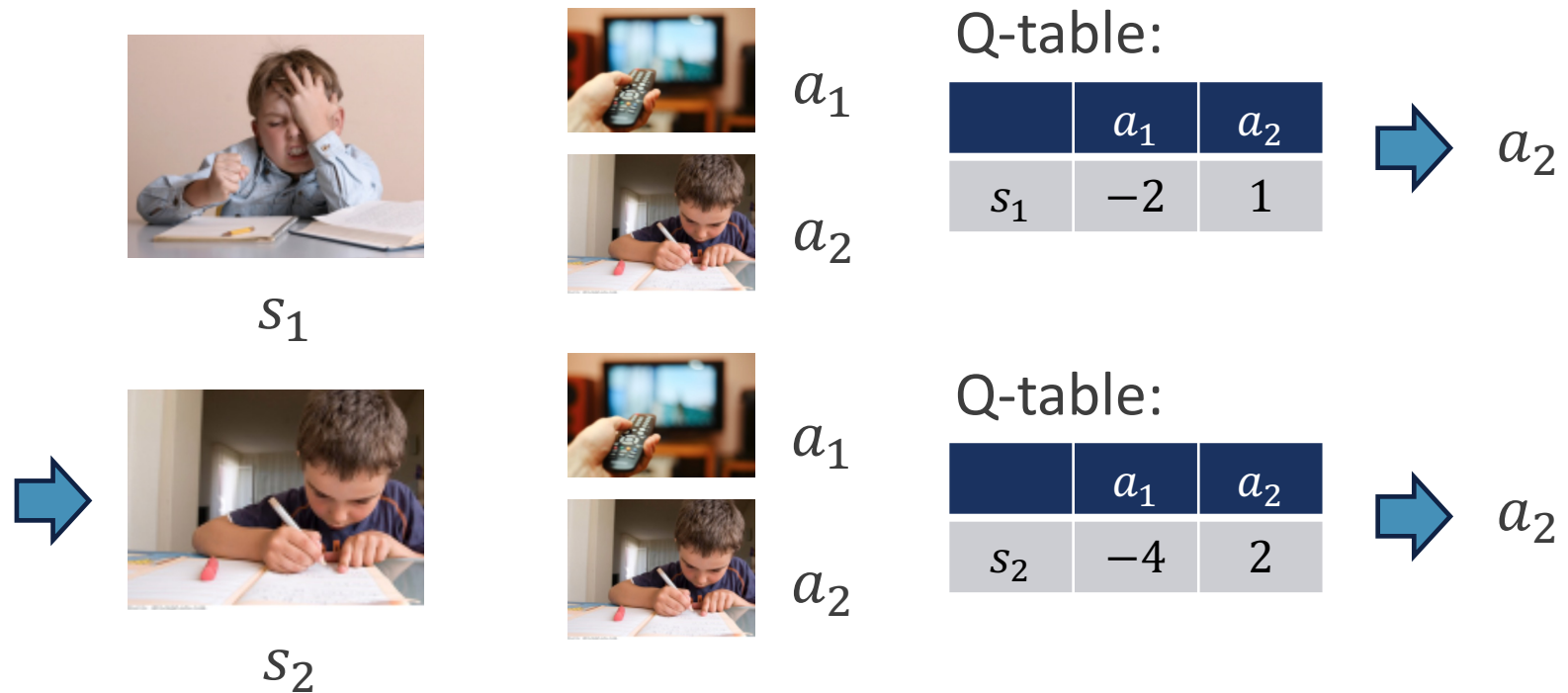
- Assume we know the return G_t for every state s with every action a . How should we do?
 - On state s , try every action a and choose the one with max return.
- How to **estimate the return** given a state-action pair?
- The **Q-value function** at state s and action a , is the expected cumulative reward from taking action a in state s and then following the policy:

$$Q_{\pi}(s, a) = \mathbb{E}[G_t | S_t = s, A_t = a]$$



Q-Learning

- The Q-values $Q_{\pi}(s, a)$ can be represented by a table called **Q-table**.
- It stores the **estimated returns (not reward!)** obtained by action a under state s .



Q-Learning

- **Store the expected return** for every state and every action.
- When we are in a state, just select the action with maximum Q-value.
- Now, when the Q-table is not accurate (e.g. random initialized), how to update based on a series of actions?

Q-table:

	a_1	a_2
s_1	-2	1
s_2	-4	2
...

Q-Learning

The Q-value function can be decomposed into two parts:

- Immediate reward R_{t+1} .
- Discounted value of successor state $\gamma \max_a Q_\pi(S_{t+1}, a)$.

$$\begin{aligned} Q_\pi(s, a) &= \mathbb{E}[G_t | S_t = s, A_t = a] \\ &= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+2} + \dots | S_t = s, A_t = a] \\ &= \mathbb{E}[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+2} + \dots) | S_t = s, A_t = a] \\ &= \mathbb{E}[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \\ &\leq \mathbb{E}\left[R_{t+1} + \gamma \max_a Q_\pi(S_{t+1}, a) | S_t = s, A_t = a\right] \end{aligned}$$

- The optimal Q-value function $Q_\pi^*(s, a)$ takes the equality:

$$Q_\pi^*(s, a) = \mathbb{E}\left[R_{t+1} + \gamma \max_a Q_\pi(s_{t+1}, a) | S_t = s, A_t = a\right].$$



Q-Learning

- We have two observations before and after we take action a_t .
 - **Before**: we only know the estimated return from Q-table.
 - **After**: we receive real reward.
- Therefore, we have an **estimated value** and a **real value** before and after we take action a_t .
 - **Estimated value**: $Q_{\pi}(s_t, a_t)$.
 - **Real value**: $R_{t+1} + \gamma \max_a Q(s_{t+1}, a)$.



Q-Learning

- Just like supervised learning, we correct our model by the difference between prediction and ground truth:

$$Q_{\pi}(s_t, a_t) \leftarrow \underbrace{Q_{\pi}(s_t, a_t)}_{\text{estimation}} + \underbrace{\alpha}_{\text{lr}} \cdot \underbrace{\left(\underbrace{R_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount}} \cdot \underbrace{\max_a Q_{\pi}(s_{t+1}, a)}_{\text{optimal future}} \right)}_{\text{real value}} - \underbrace{Q_{\pi}(s_t, a_t)}_{\text{estimation}}$$

difference

ϵ -Greedy

- Now, the only problem left is how to take action from Q-table.
 - Always select the maximum value?
- ϵ -greedy is a strategy to deal with exploration-exploitation tradeoff.
 - ϵ ranges $[0,1]$, indicating the probability to use the local optimal solution.
- For example, when $\epsilon = 0.9$.
 - In 90% situations, we take the action with maximum Q-value.
 - In 10% situations, we take random actions.

Q-Learning

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

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 $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

Initialize S

Loop for each step of episode:

Choose A from S using policy derived from Q (e.g., ε -greedy)

Take action A , observe R, S'

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$$

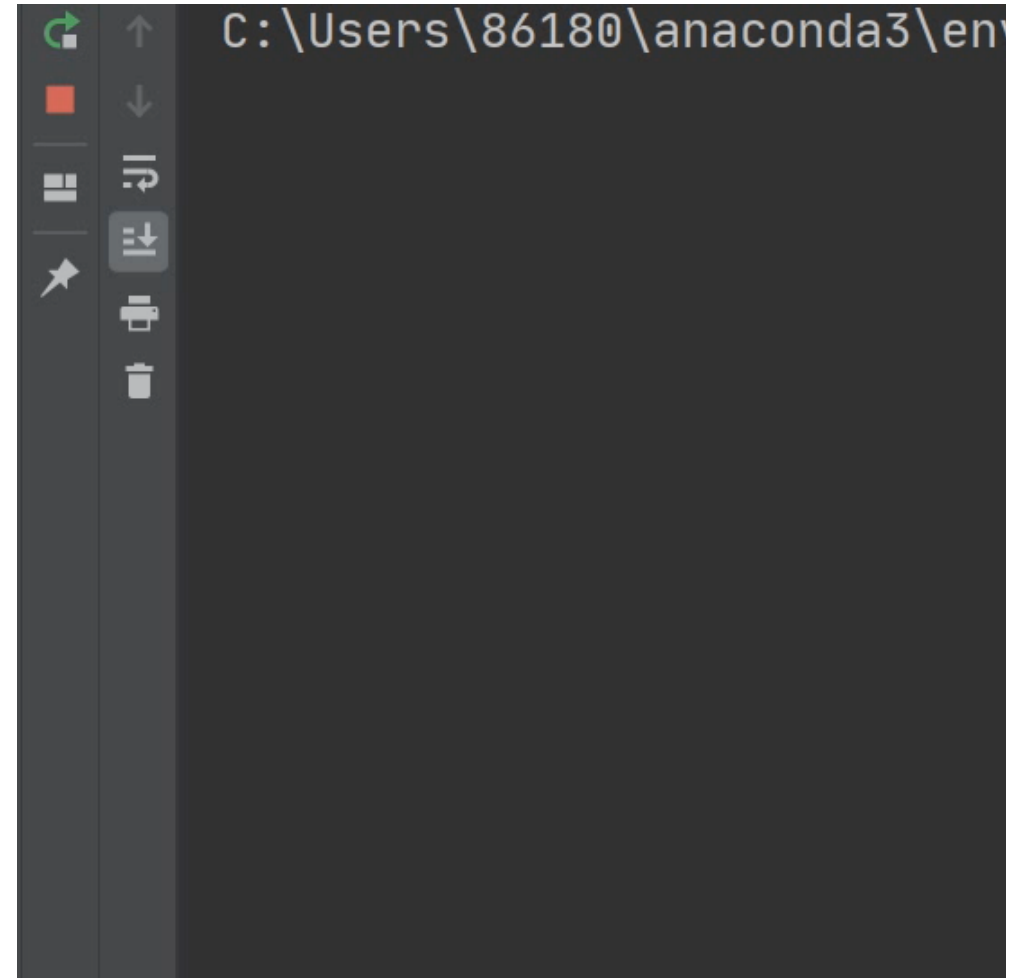
$S \leftarrow S'$

until S is terminal



Q-Learning Demo

- A simple example for Reinforcement Learning using table lookup Q-learning method.
- An agent "o" is on the left of a 1 dimensional world, the treasure is on the rightmost location.
- Run this program and to see how the agent will improve its strategy of finding the treasure.
- Run this demo by yourself in Lecture 10.ipynb



Problem of Q-Learning

- Not scalable.
 - Must compute $Q_{\pi}(s, a)$ for every state-action pair.
- If the number of states is extremely huge, it is computationally infeasible to compute for entire state space!
 - E.g. Go board has a 19×19 grid of lines. How many states?
- What we want is not we store the Q-table, we **aim to estimate the return from current state s with action a** .
 - Given s and a as input, estimate return as output, how to do?



DEEP Q NETWORK

Deep Q Network

Playing atari with **deep** reinforcement learning

[V Mnih](#), [K Kavukcuoglu](#), [D Silver](#), [A Graves](#)... - arXiv preprint arXiv ..., 2013 - arxiv.org

We present the first **deep** learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning. The model is a convolutional ...

☆ Save 剪 Cite Cited by 13348 Related articles All 42 versions 》

- Deep Q Network (DQN) uses deep neural network to estimate the action-value function.

$$Q_{\pi}(s, a; \theta) \approx Q_{\pi}^*(s, a)$$

- θ is network parameters.
- Now, the estimated and real Q-value becomes:
 - Estimated value: $Q_{\pi}(s_t, a_t; \theta)$.
 - Real value: $R_{t+1} + \gamma \max_a Q(s_{t+1}, a; \theta)$.
- How to update? Gradient descent:

$$\nabla_{\theta} L_t = \mathbb{E}[R_{t+1} + \gamma \max_a Q(s_{t+1}, a; \theta) - Q_{\pi}(s_t, a_t; \theta)]$$



Deep Q Network

- Now new problems appear.
- Training samples in a time interval are **highly correlated**.
 - They are not i.i.d!
 - Inefficient learning.
- The current parameters determine the next data sample that the parameters are trained on.
 - Get stuck in a poor local minimum, or diverge catastrophically.

Experience Replay

- Idea: store transitions in memory and random sample some for training at each step.

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**

 Initialize 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}

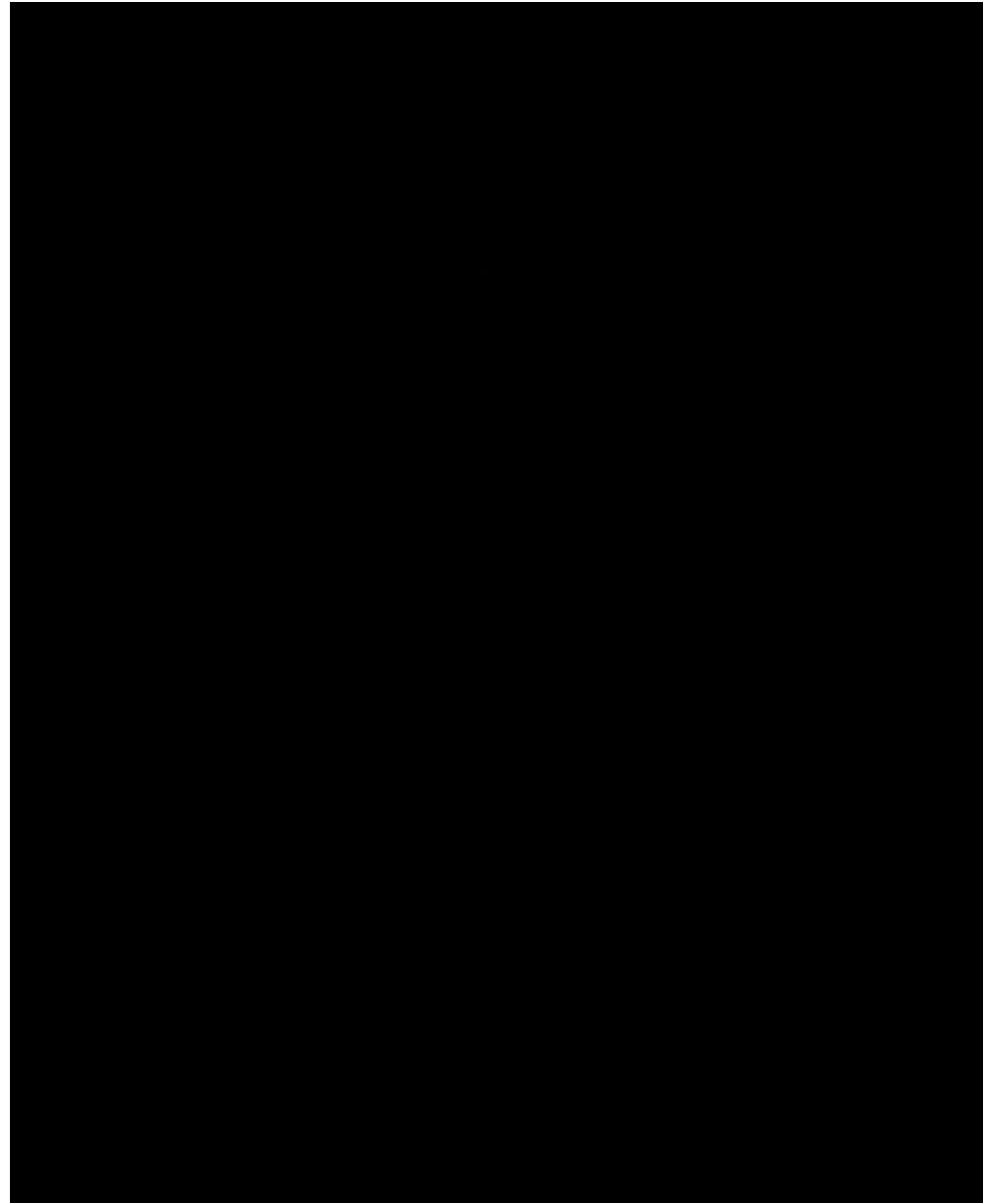
 Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$

 Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3

end for

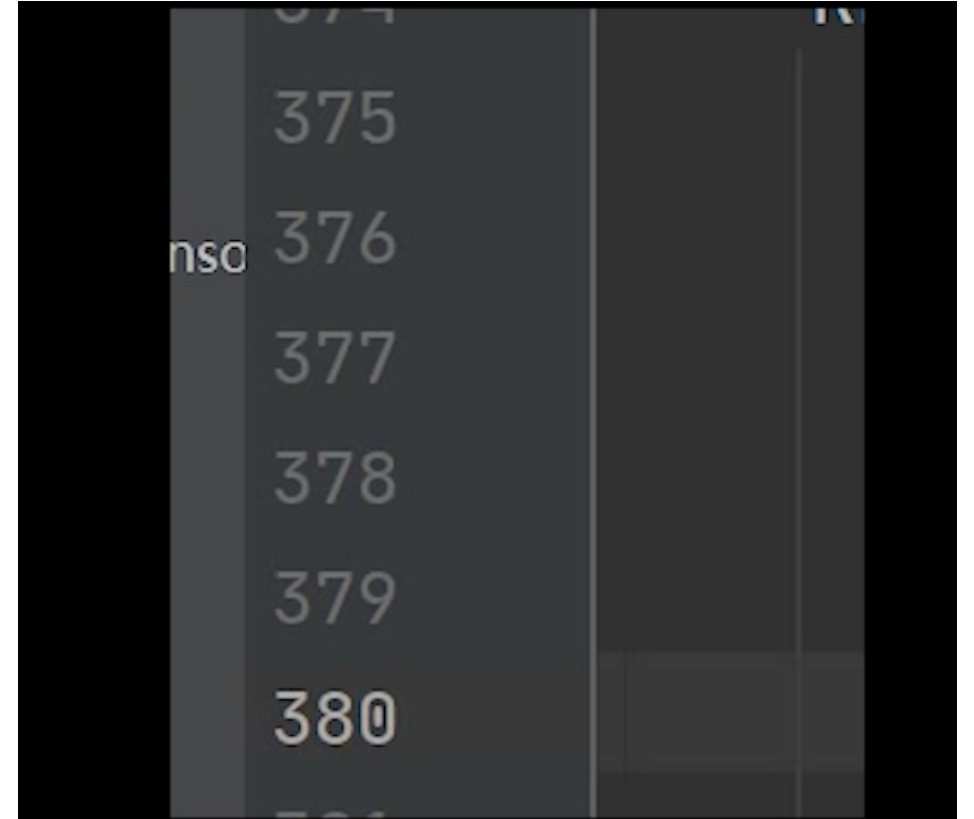
end for





DQN Demo

- Reinforcement learning maze example.
 - Red rectangle: explorer.
 - Black rectangles: hells [reward = -1].
 - Yellow bin circle: paradise [reward = +1].
 - All other states: ground [reward = 0].
- Run this demo by yourself in Lecture 10.ipynb



Problems with Q-Learning

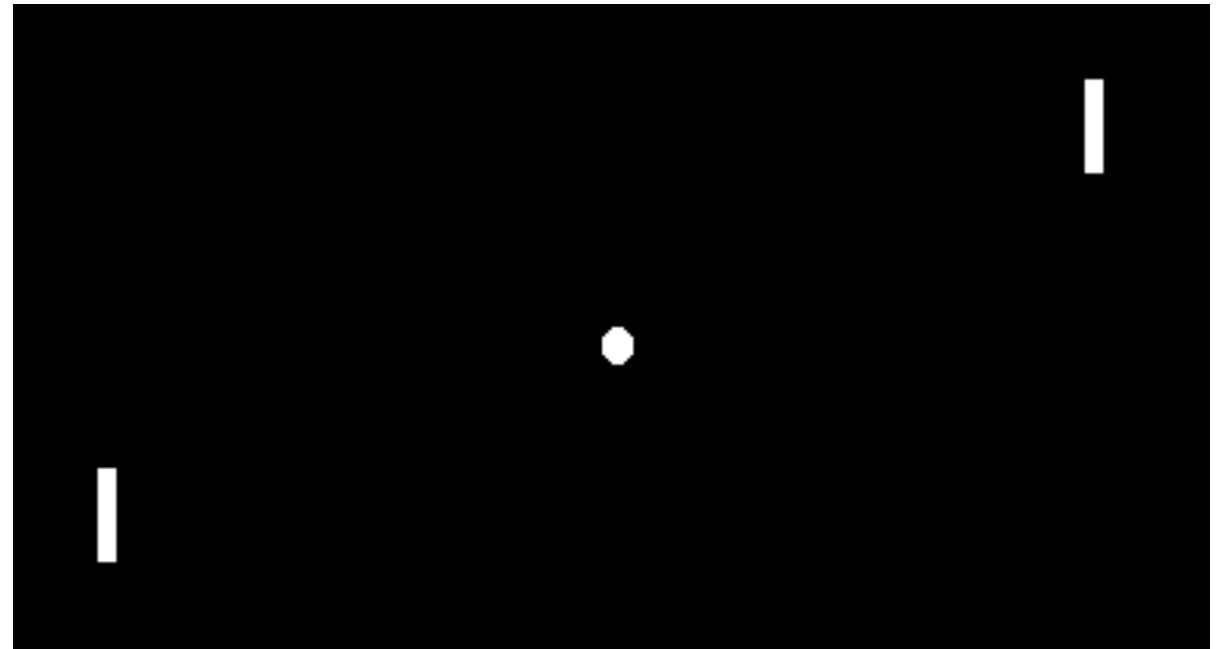
- Q-learning steps:
 - Estimate return, given state and action, by Q-function.
 - Select the action with maximum return by ϵ -greedy.
- Problems:
 - The Q-function can be very complicated, e.g. in Dota 2, the screenshot is the current state. (how many states there?)
 - Can't only deal with continuous actions, e.g. cast hero's spell at some position, or hold a key for some time.
- Instead, can we learn a policy directly, e.g. finding the best policy from a collection of policies?



POLICY GRADIENT

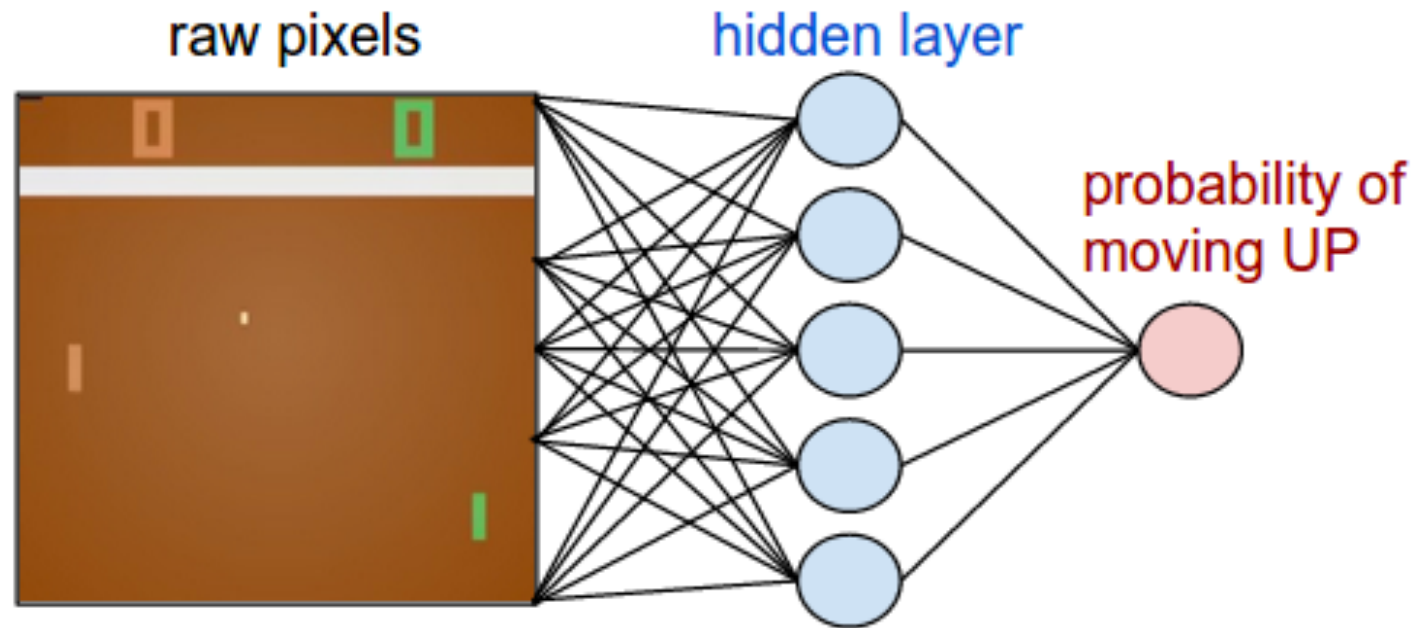
The Game of Pong

- The game of Pong is an excellent example of a simple RL task using policy gradient.
- Input: a 210x160 image.
 - How many states?
- Output: UP or DOWN.
- Reward:
 - +1 if the ball went past the opponent,
 - -1 reward if we missed the ball,
 - 0 otherwise.
- Our goal is to move the paddle so that we get lots of reward.



Policy Network

- Define a **policy network** that implements our player (or “agent”).
- This network will take the state of the game and decide what we should do (move UP or DOWN).

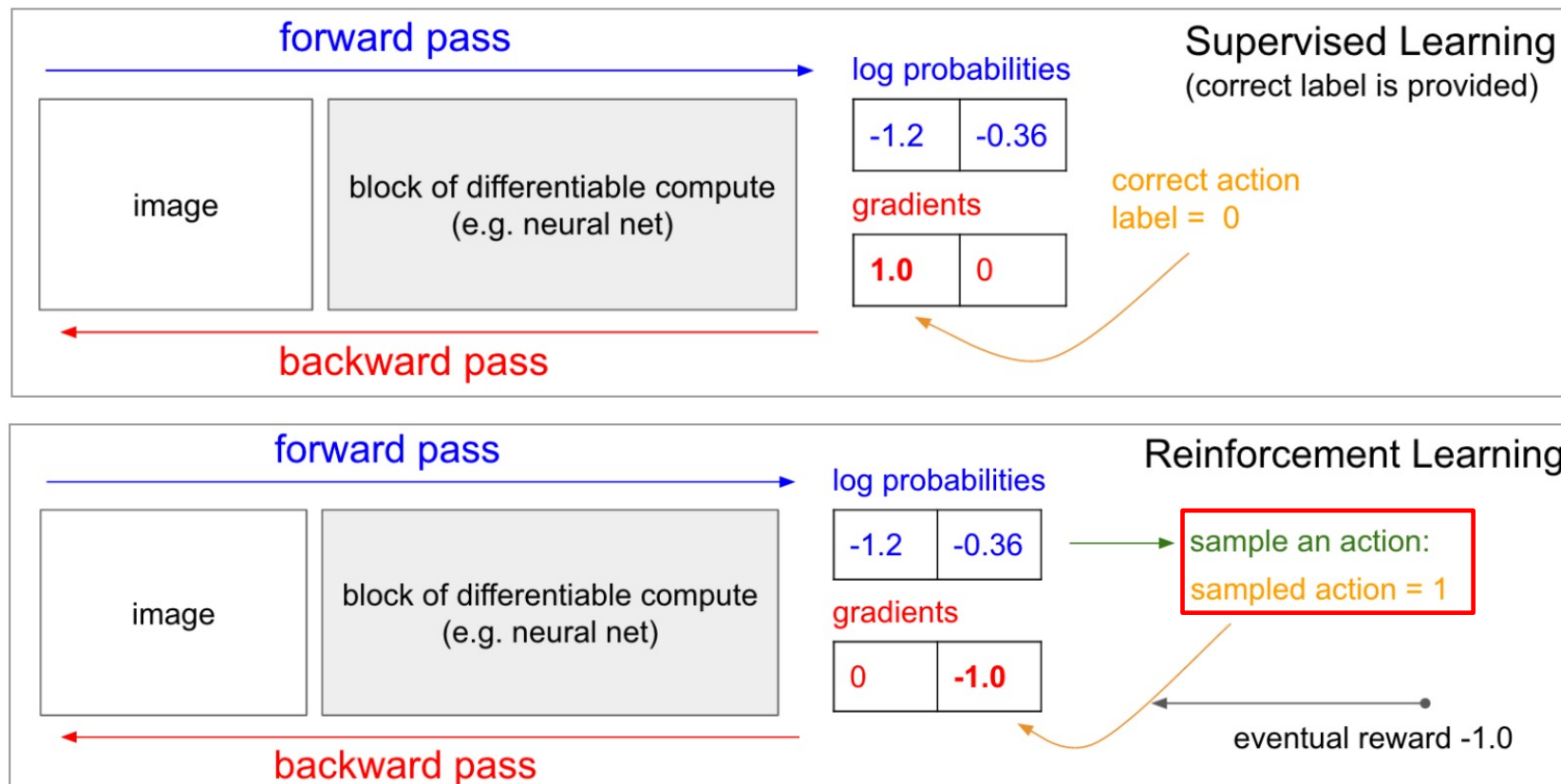


A 2-layer fully-connected policy network



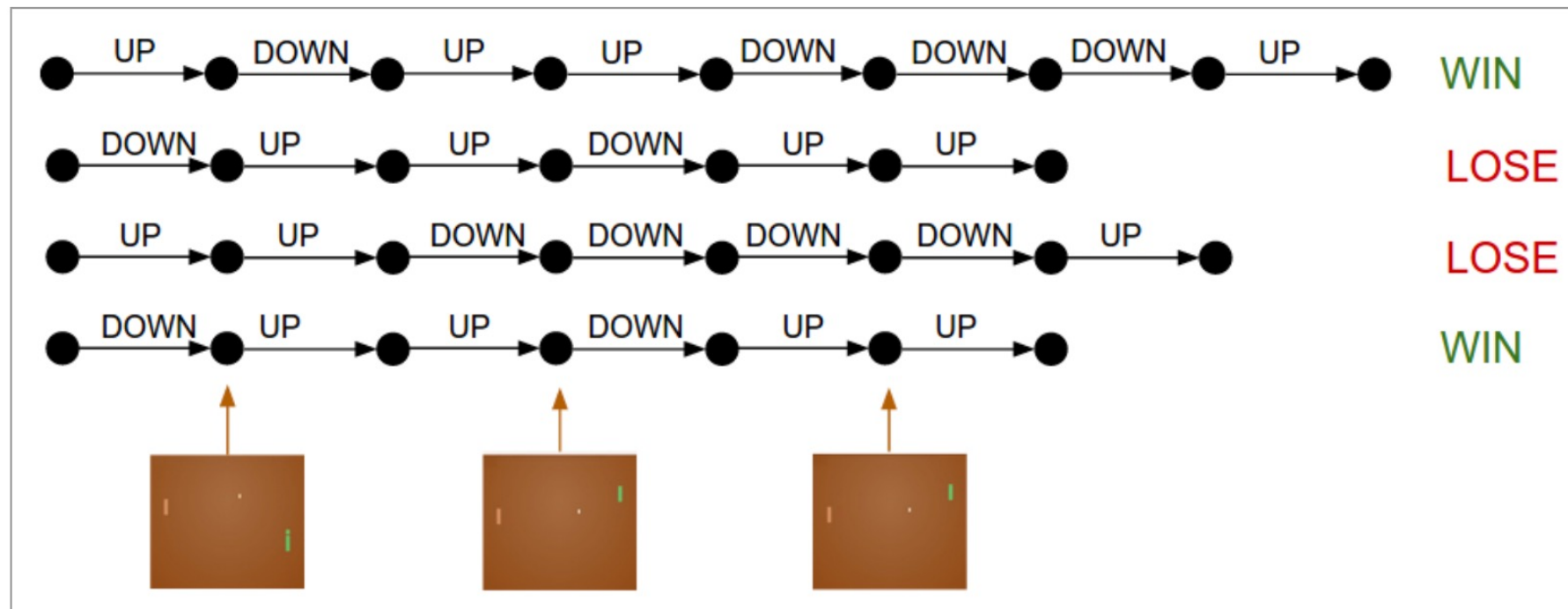
Policy Network vs Supervised Learning

- How to evaluate the action probabilities generated by our model?
 - Suppose at some moment, we choose DOWN, and finally we lose.



Policy Network

- Use a single final reward as the label for all previous actions, is it ok?
- Average over many iterations, we can eventually increase the probability of good actions and decrease the probability of poor actions.



REINFORCE Algorithm

Simple statistical gradient-following algorithms for connectionist reinforcement learning

RJ Williams - Machine learning, 1992 - Springer

This article presents a general class of associative reinforcement learning algorithms for connectionist networks containing stochastic units. These algorithms, called REINFORCE ...

☆ Save ↗ Cite Cited by 10332 Related articles All 19 versions

- Mathematically, we can write:

$$J(\theta) = \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau)]$$

$$p(\tau; \theta) = \prod_{t \geq 0} p(s_{t+1} | s_t, a_t) \pi_{\theta}(a_t | s_t)$$

where $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \dots)$ is the trajectory. $r(\tau)$ is the total reward in the trajectory.

- Now let's differentiate to calculate the gradients:

$$\nabla_{\theta} J(\theta) = \nabla_{\theta} \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau)]$$

- Can we do this?

Intractable! Gradient of an expectation is problematic when p depends on θ .



REINFORCE Algorithm

- Use some mathematical tricks help us move gradient into expectation.

$$\begin{aligned}\nabla_{\theta} J(\theta) &= \nabla_{\theta} \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau)] \\ &= \nabla_{\theta} \int_{\tau} r(\tau) p(\tau; \theta) d\tau \\ &= \int_{\tau} r(\tau) \nabla_{\theta} p(\tau; \theta) d\tau \\ &= \int_{\tau} r(\tau) \frac{\nabla_{\theta} p(\tau; \theta)}{p(\tau; \theta)} p(\tau; \theta) d\tau \\ &= \int_{\tau} (r(\tau) \nabla_{\theta} \log p(\tau; \theta)) p(\tau; \theta) d\tau \\ &= \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau) \nabla_{\theta} \log p(\tau; \theta)]\end{aligned}$$

Often refer to
“likelihood ratio trick”



REINFORCE Algorithm

$$\begin{aligned} & \nabla_{\theta} \log p(\tau; \theta) \\ &= \nabla_{\theta} \prod_{t \geq 0} p(s_{t+1} | s_t, a_t) \pi_{\theta}(a_t | s_t) \\ &= \nabla_{\theta} \sum_{t \geq 0} (\log p(s_{t+1} | s_t, a_t) + \log \pi_{\theta}(a_t | s_t)). \end{aligned}$$

- There's nothing to do with the first term. Therefore we can estimate the gradient with

$$\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t).$$

```
neg_log_prob = tf.reduce_sum(-tf.log(self.all_act_prob) *  
                             tf.one_hot(self.tf_acts, self.n_actions), axis=1)  
loss = tf.reduce_mean(neg_log_prob * self.tf_vt) # reward guided loss
```



REINFORCE Algorithm

function REINFORCE

Initialise θ arbitrarily

for each episode $\{s_1, a_1, r_2, \dots, s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**

for $t = 1$ to $T - 1$ **do**

$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$

end for

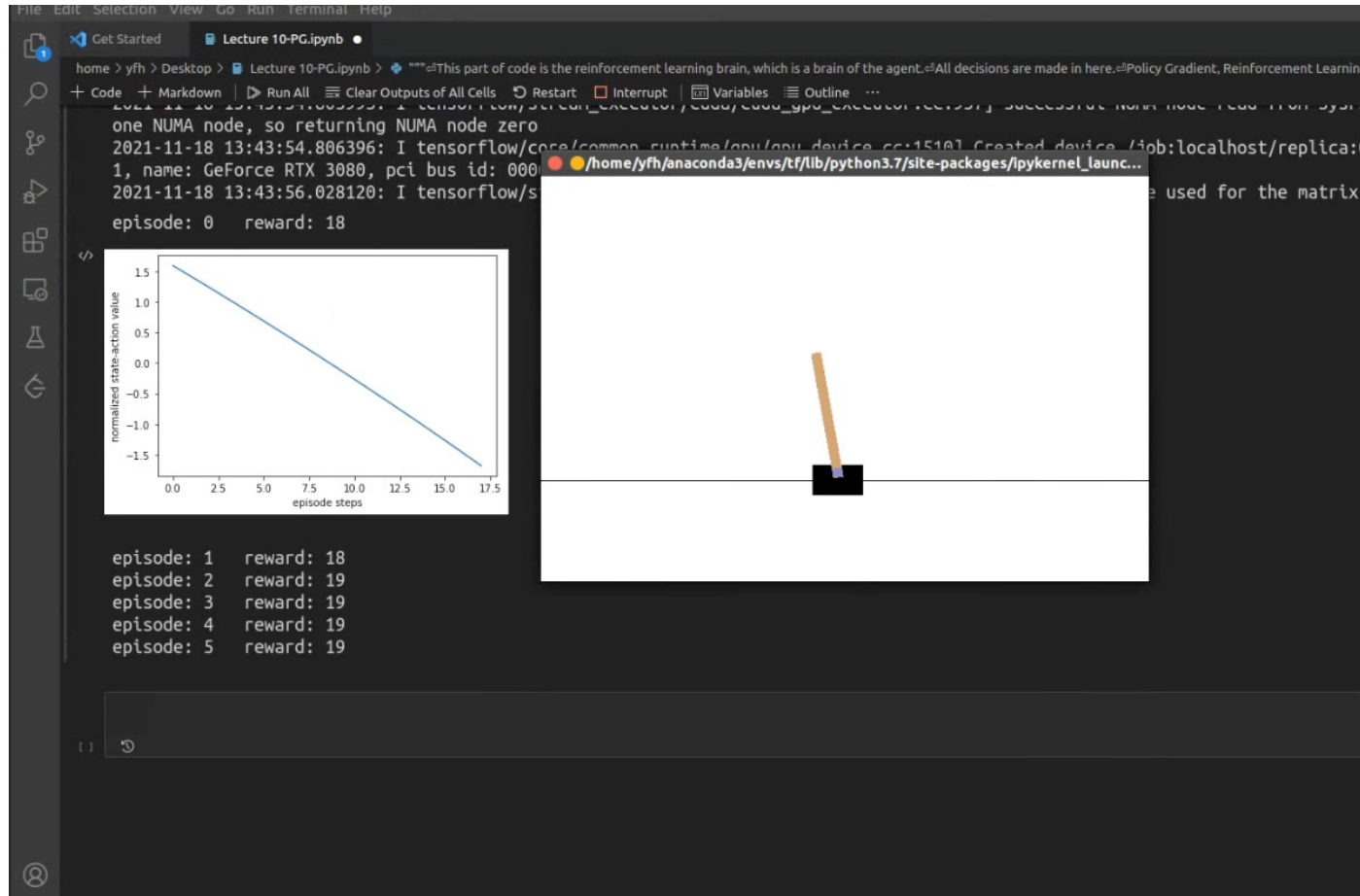
end for

return θ

end function



PG Demo



Run this demo by yourself in Lecture 10.ipynb



Problem of Gradient Policy

- What if we can only get a reward after thousands of actions?
- How can we know which action is good or poor?
- We call this the **credit assignment problem**.
- Although it averages out in expectation, policy gradient still suffers from high variance because credit assignment is really hard.
- Can we help the estimator to reduce variance?

$$\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$$

$r(\tau)$ has little relation with the early policy gradient



Problem of Gradient Policy

- What we want is to **estimate an immediate evaluation to the action a_t** .

$$\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$$

- What does this remind you of?
- Q-value function!
- $Q_{\pi}(s_t, a_t)$ is designed for estimating how a_t is on s_t .

$$\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} Q_{\pi_{\theta}}(s_t, a_t) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t).$$





ACTOR-CRITIC

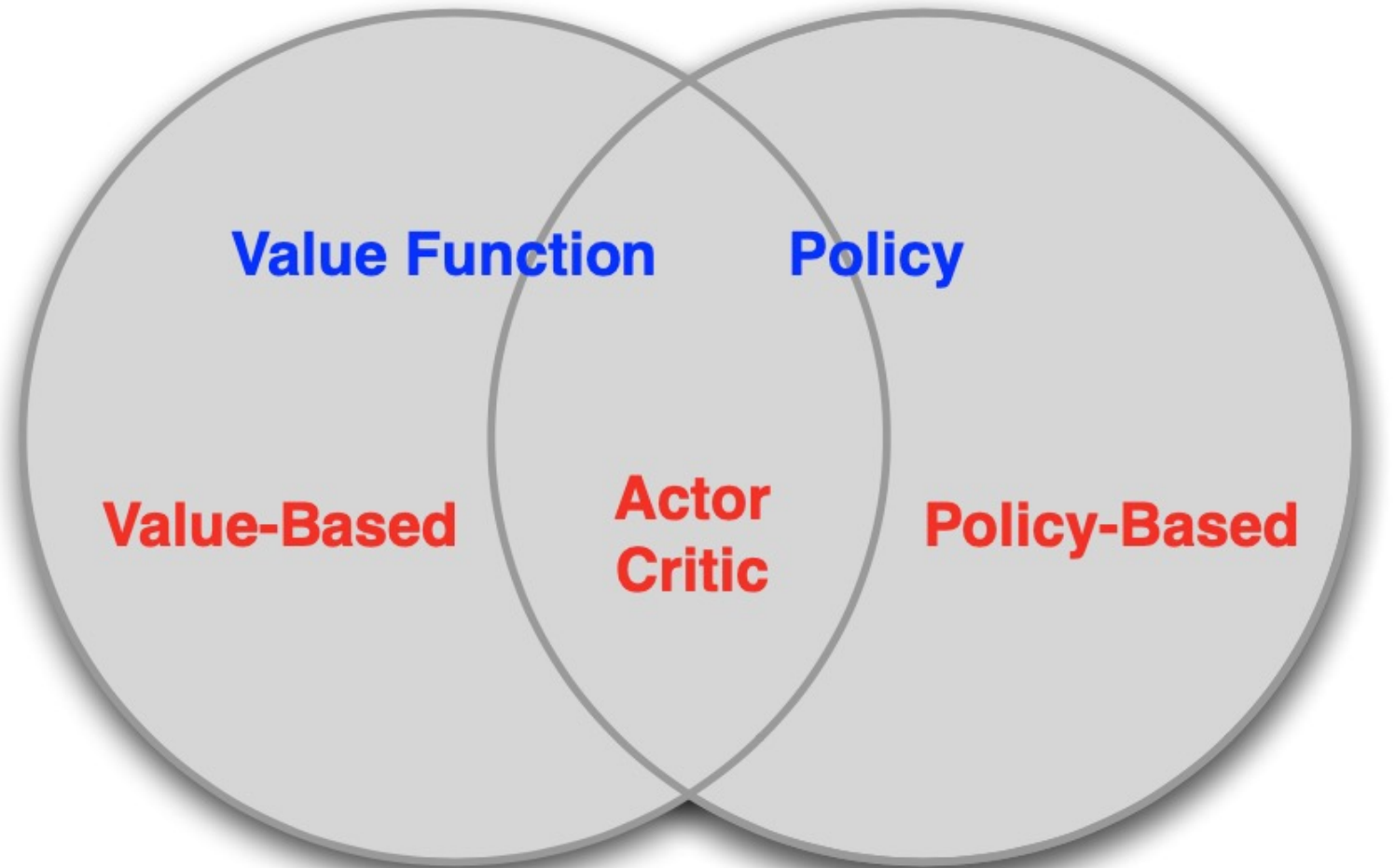
Q-Learning vs Policy Gradient

- Q-learning can't deal with continuous action but policy gradient can.
- Policy gradient usually uses episode-level update, but Q-learning can update at each step.

Why not together?

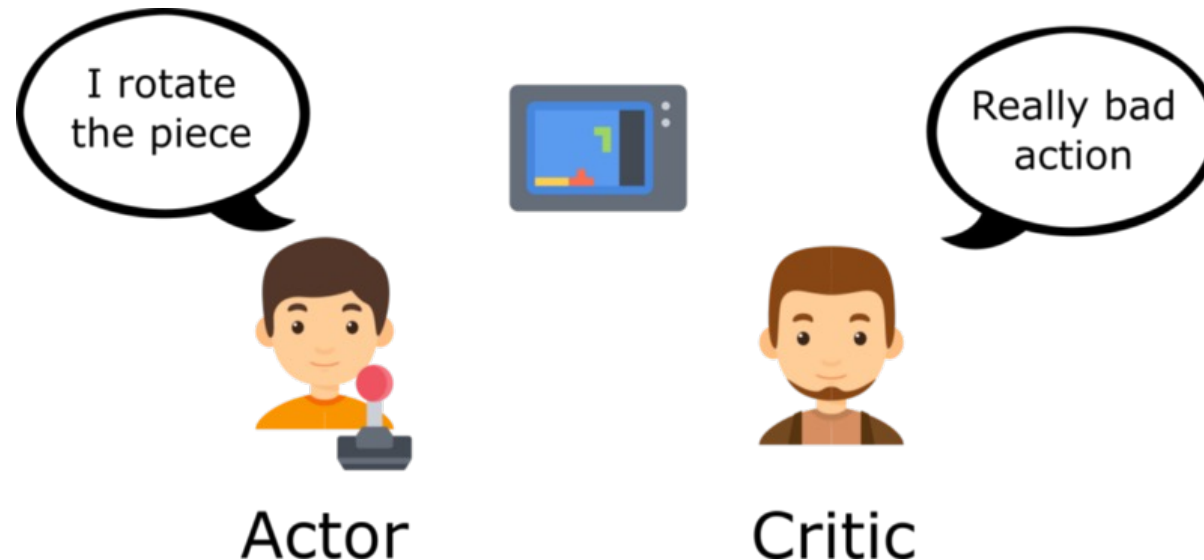
Value-Based and Policy-Based

- Value Based
 - Learn value function.
 - Implicit policy (e.g. ϵ -greedy).
- Policy Based
 - No value function.
 - Learn policy.
- Actor-Critic
 - Learn value function.
 - Learn policy.



Actor-Critic Algorithm

- The actor decides which action to take, and the critic tells the actor how good its action was and how it should adjust.
- Can also incorporate Q-learning tricks, e.g. experience replay.
- Advantage: Single step update, faster than pure PG.
- Disadvantage: Two networks, extremely hard to converge.



Actor-Critic Algorithm

We can combine policy gradient and Q-learning by training both an **actor** (the policy) and a **critic** (the Q-function).

- Actor update:

$$\Delta\theta = \alpha Q_\phi(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t | s_t)$$

- Critic update:

$$\Delta\phi = \beta \nabla_\phi (R_{t+1} + \gamma \max_a Q_\phi(s_{t+1}, a) - Q_\phi(s_t, a_t))^2$$

- Problem: How to take max on continuous actions?

We adapt the ideas underlying the success of Deep Q-Learning to the **continuous** action domain. We present an actor-critic, model-free algorithm based on the deterministic policy ...

☆ Save 📄 Cite Cited by 13953 Related articles All 16 versions ⇨

- Deep Deterministic Policy Gradient (DDPG) combines DQN and deterministic policy gradient.
- Before, we have the Q-value function:

$$\begin{aligned} Q_{\pi}(s, a) &= \mathbb{E}[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \\ &= \mathbb{E}[R_{t+1} + \gamma \mathbb{E}[Q_{\pi}(s_{t+1}, a)] | S_t = s, A_t = a] \end{aligned}$$

- When we have a **deterministic policy** $\mu: S \rightarrow A$, we can remove expectation and rewrite the formula as:

$$Q_{\pi}(s, a) = \mathbb{E}[R_{t+1} + \gamma Q_{\pi}(s_{t+1}, \mu(s_{t+1})) | S_t = s, A_t = a]$$

- For example, in DQN, we have $\mu(s) = \operatorname{argmax}_a Q(s, a)$:

$$\max_a Q_{\pi}(s_{t+1}, a) = Q_{\pi}\left(s_{t+1}, \operatorname{argmax}_a Q(s_{t+1}, a)\right).$$

- $\mu(s_t)$ is deterministic, e.g. $\mu(s_t) = \max_{a_t} \pi_{\theta}(a_t | s_t)$.
- Critic update can then be written as:

$$\Delta\phi = \beta \nabla_{\phi} \left(\underbrace{R_{t+1} + \gamma Q_{\phi}(s_{t+1}, \mu(s_{t+1}))}_{\text{real value}} - \underbrace{Q_{\phi}(s_t, a_t)}_{\text{estimation}} \right)^2$$

Not real “real value”, also an estimated value as the ground truth



Target Network

- Directly implementing Q-learning with neural networks proved to be unstable in many environments.
- Since the network $Q(s, a | \theta^Q)$ being updated is also used in calculating the target value, the Q update is prone to divergence.
 - Again, use prediction as label.

Target Network

- The solution is to use “soft” target updates, rather than directly copying the weights.
- Create a copy of the actor and critic networks, $Q'(s, a|\theta^{Q'})$ and $\mu'(s|\theta^{\mu'})$ respectively, that are used for calculating the target values, called **target networks**.
- Then, update them by soft target update:

$$\theta' = \tau\theta + (1 - \tau)\theta', \quad \tau \ll 1.$$

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ .

Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$

Initialize replay buffer R

for episode = 1, M **do**

 Initialize a random process \mathcal{N} for action exploration

 Receive initial observation state s_1

for $t = 1, T$ **do**

 Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise

 Execute action a_t and observe reward r_t and observe new state s_{t+1}

 Store transition (s_t, a_t, r_t, s_{t+1}) in R

 Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R

 Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$

 Update critic by minimizing the loss: $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$

 Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

 Update the target networks:

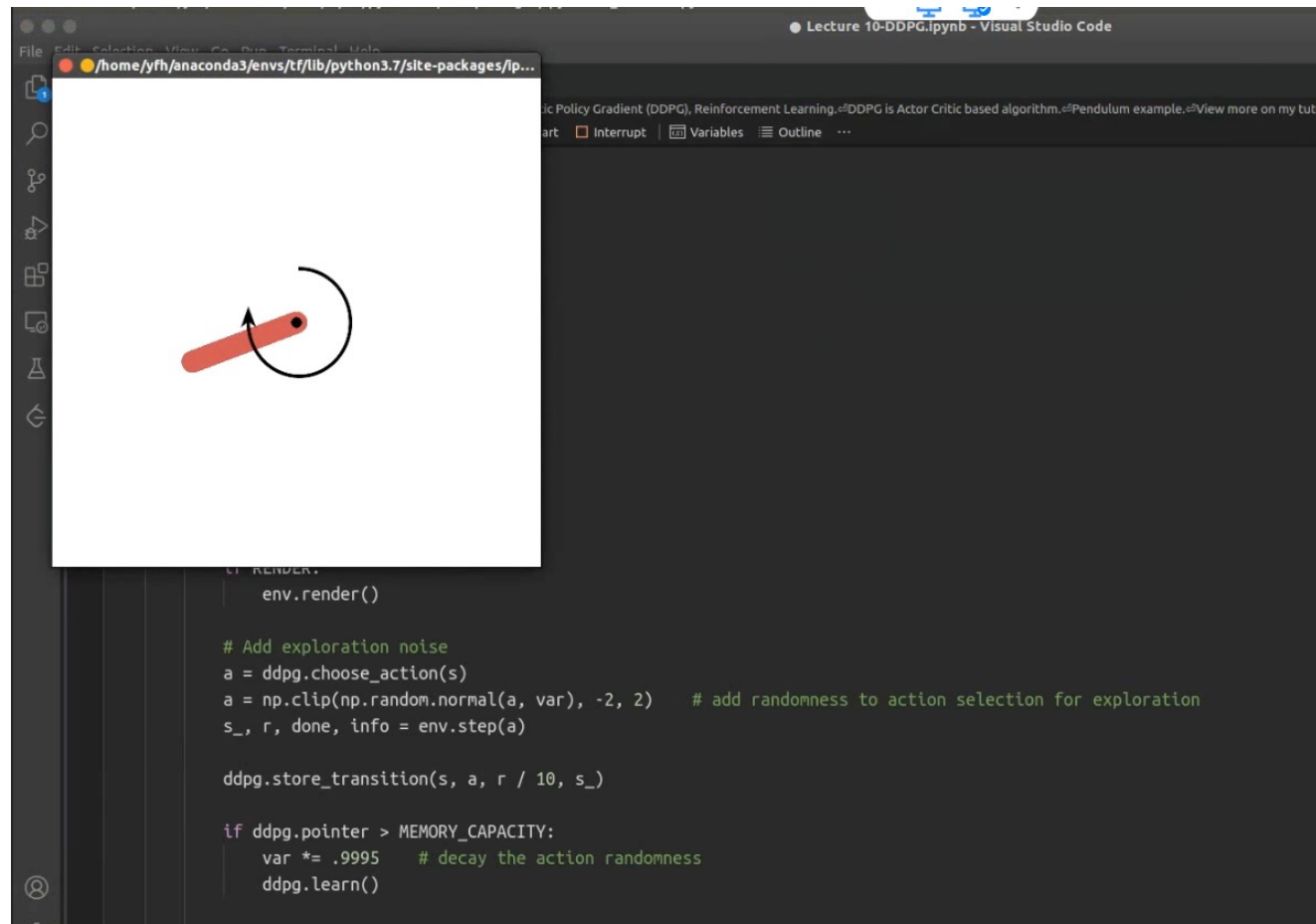
$$\theta^{Q'} \leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'}$$

end for

end for

DDPG Demo



The screenshot shows a Jupyter Notebook titled "Lecture 10-DDPG.ipynb" in Visual Studio Code. A white box highlights a diagram of a pendulum with a red rod and a black dot at the pivot, with a curved arrow indicating its rotation. Below the diagram, the notebook contains Python code for the DDPG algorithm:

```
env.render()

# Add exploration noise
a = ddpg.choose_action(s)
a = np.clip(np.random.normal(a, var), -2, 2) # add randomness to action selection for exploration
s_, r, done, info = env.step(a)

ddpg.store_transition(s, a, r / 10, s_)

if ddpg.pointer > MEMORY_CAPACITY:
    var *= .9995 # decay the action randomness
    ddpg.learn()
```

Run this demo by yourself in Lecture 10.ipynb



Actors in Parallel



Kage Bunshin no Jutsu in Naruto



- Instead of experience replay, asynchronously execute multiple agents in parallel, on multiple instances of the environment.
- This parallelism also **decorrelates** the agents' data into a more stationary process.
- At any given time-step the parallel agents will be experiencing a variety of different states.



Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

// Assume global shared parameter vectors θ and θ_v and global shared counter $T = 0$

// Assume thread-specific parameter vectors θ' and θ'_v

Initialize thread step counter $t \leftarrow 1$

repeat

Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.

Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$

Synchronize global parameters

$t_{start} = t$

Get state s_t

repeat

Perform a_t according to policy $\pi(a_t|s_t; \theta')$

Receive reward r_t and new state s_{t+1}

$t \leftarrow t + 1$

$T \leftarrow T + 1$

until terminal s_t **or** $t - t_{start} == t_{max}$

$R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t // \text{ Bootstrap from last state} \end{cases}$

for $i \in \{t - 1, \dots, t_{start}\}$ **do**

$R \leftarrow r_i + \gamma R$

Accumulate gradients wrt θ' : $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i; \theta')(R - V(s_i; \theta'_v))$

Accumulate gradients wrt θ'_v : $d\theta_v \leftarrow d\theta_v + \partial (R - V(s_i; \theta'_v))^2 / \partial \theta'_v$

Local parameters don't update

end for

Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.

Asynchronous update

until $T > T_{max}$

- The most commonly used policy gradient estimator:

$$\hat{\mathbb{E}}_t \left[\nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \hat{A}_t \right]$$

- Relaxation for PPO objective:

$$\underset{\theta}{\text{maximize}} \hat{\mathbb{E}}_t \left[\frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)} \hat{A}_t - \beta \text{KL}[\pi_{\theta_{\text{old}}}(\cdot | s_t), \pi_{\theta}(\cdot | s_t)] \right]$$

Training language models to follow instructions with human feedback

[L Ouyang, J Wu, X Jiang, D Almeida...](#) - Advances in ..., 2022 - proceedings.neurips.cc

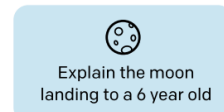
... **InstructGPT**. In human evaluations on our prompt distribution, outputs from the 1.3B parameter **InstructGPT** model ... Moreover, **InstructGPT** models show improvements in truthfulness and ...

☆ Save 📄 Cite Cited by 2748 Related articles All 10 versions 🔗

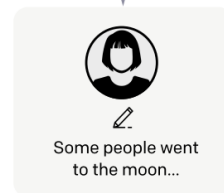
Step 1

Collect demonstration data, and train a supervised policy.

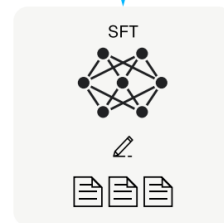
A prompt is sampled from our prompt dataset.



A labeler demonstrates the desired output behavior.



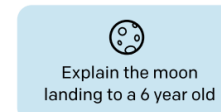
This data is used to fine-tune GPT-3 with supervised learning.



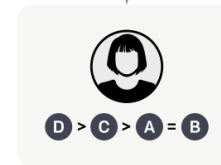
Step 2

Collect comparison data, and train a reward model.

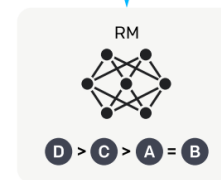
A prompt and several model outputs are sampled.



A labeler ranks the outputs from best to worst.



This data is used to train our reward model.



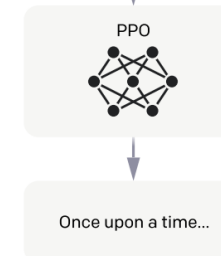
Step 3

Optimize a policy against the reward model using reinforcement learning.

A new prompt is sampled from the dataset.



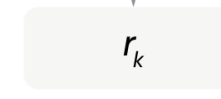
The policy generates an output.



The reward model calculates a reward for the output.



The reward is used to update the policy using PPO.



Reinforcement Learning vs Human

There are several differences or similarities between humans and reinforcement learning to play games:

- RL does not accept any rule input, and guesses the game rule through trial and error completely. From this perspective, RL will be much better than humans.
 - E.g. RL will not be influenced by shuffled pixels.
- Humans have a lot of prior knowledge. However, RL doesn't know anything, it's all by trial and error.
 - E.g. the ball will bounce, the popping angle is the same as the popping angle, the actions correspond to the movement of the racket, and so on.



Reinforcement Learning vs Human

There are several differences or similarities between humans and reinforcement learning to play games:

- Trial and error is a necessary condition for RL learning, but not a necessary condition for human learning.
 - E.g., humans don't need to experience car crashes hundreds of times in order to learn how to avoid car crashes.
- Prior knowledge determines that humans can quickly abstract the rules of the game (strategic network).
 - E.g., describe the rules of the game in one sentence.

Conclusion

After this lecture, you should know:

- What are states, actions, rewards?
- What is the Q-value function?
- How are deep neural networks incorporated into Q-Learning?
- How does policy gradient work?
- What is actor-critic?

Suggested Reading

- 莫烦强化学习
- Deep Reinforcement Learning: Pong from Pixels
- DQN paper
- DDPG paper



Reference

- Introduction to Reinforcement Learning with David Silver
- Lecture 17, cs231n, Stanford University

Assignment 4

- Assignment 4 is released. The deadline is **18:00, 18st November.**



Thank you!

- Any question?
- Don't hesitate to send email to me for asking questions and discussion. 😊