

清華大學 交叉信息研究院
Tsinghua University Institute for Interdisciplinary Information Sciences


ISR Lab
Intelligent Systems and Robotics Lab

Advanced Topics for Robotics
智能机器人前沿探究

Lecture 5: Reinforcement Learning

Jianyu Chen (陈建宇)

Institute for Interdisciplinary Information Sciences

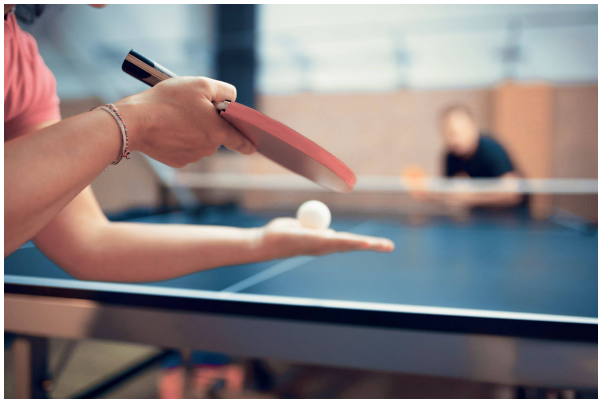
Tsinghua University

We are using reinforcement learning

- We grow up with continuous reinforcement learning

We are using reinforcement learning

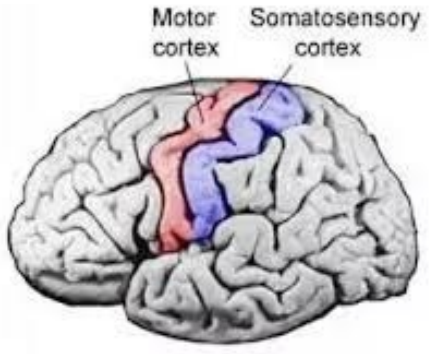
- We are still using reinforcement learning everyday

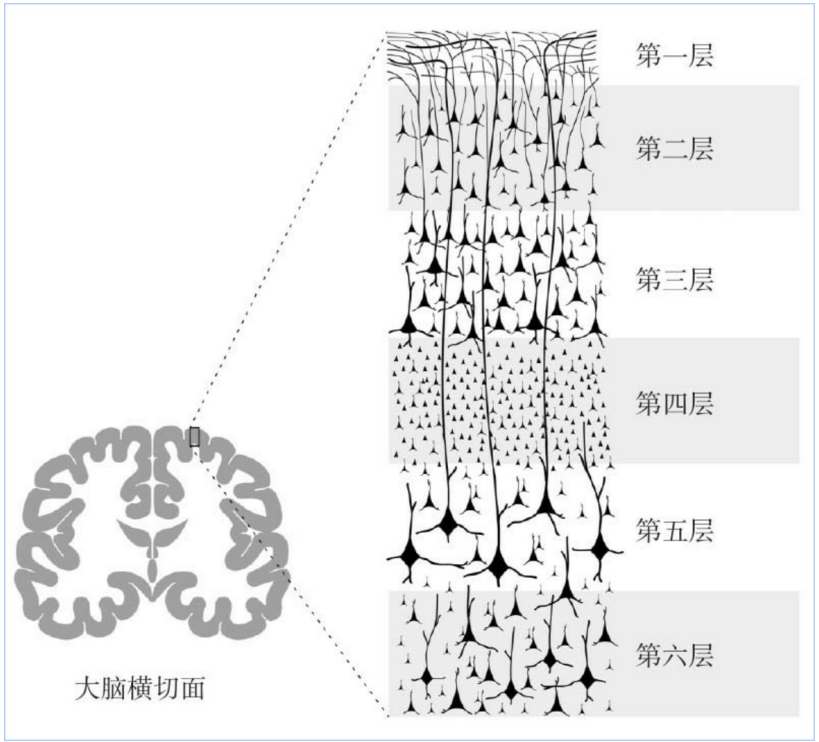


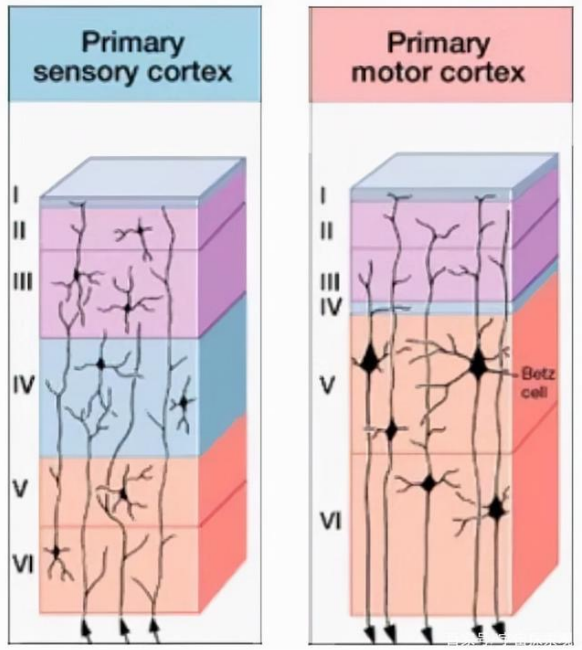



Reinforcement learning mechanism in brain

- The sensory motor cortex
- “End-to-end” control style

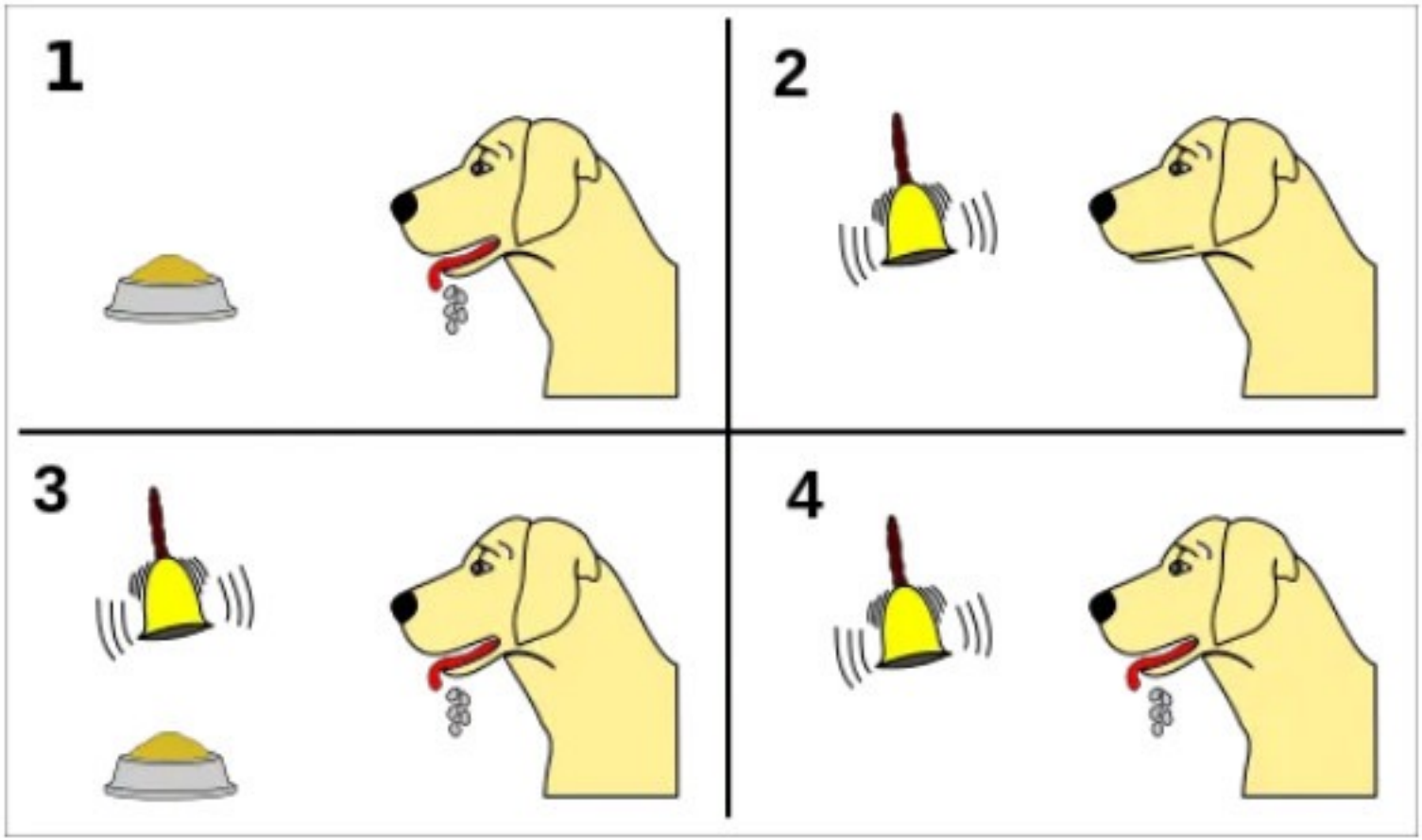






Reinforcement learning mechanism in brain

- Pavlov's dog

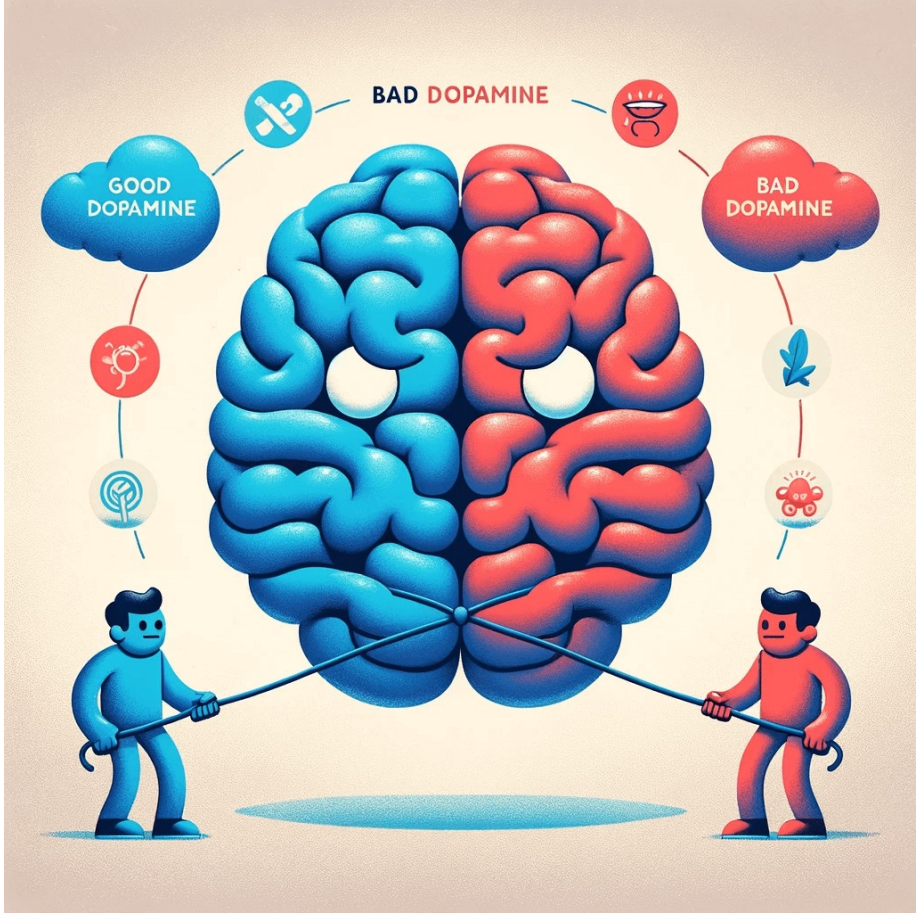


Reinforcement learning mechanism in brain

- Skinson's pigeon

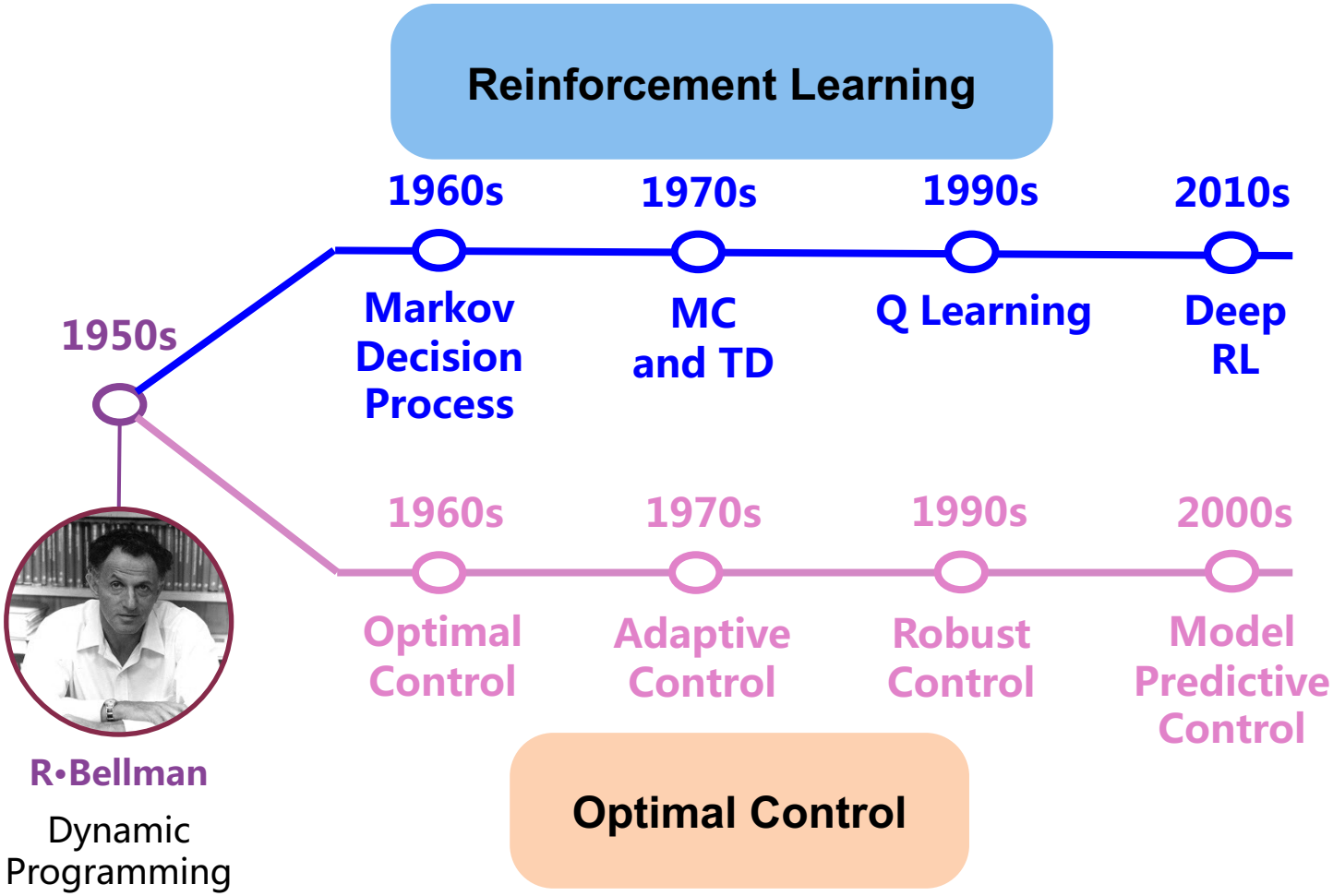
Reinforcement learning mechanism in brain

- Dopamine as the reward
- Learning connections between action and consequences



History of reinforcement learning

- Trace back to 1950s

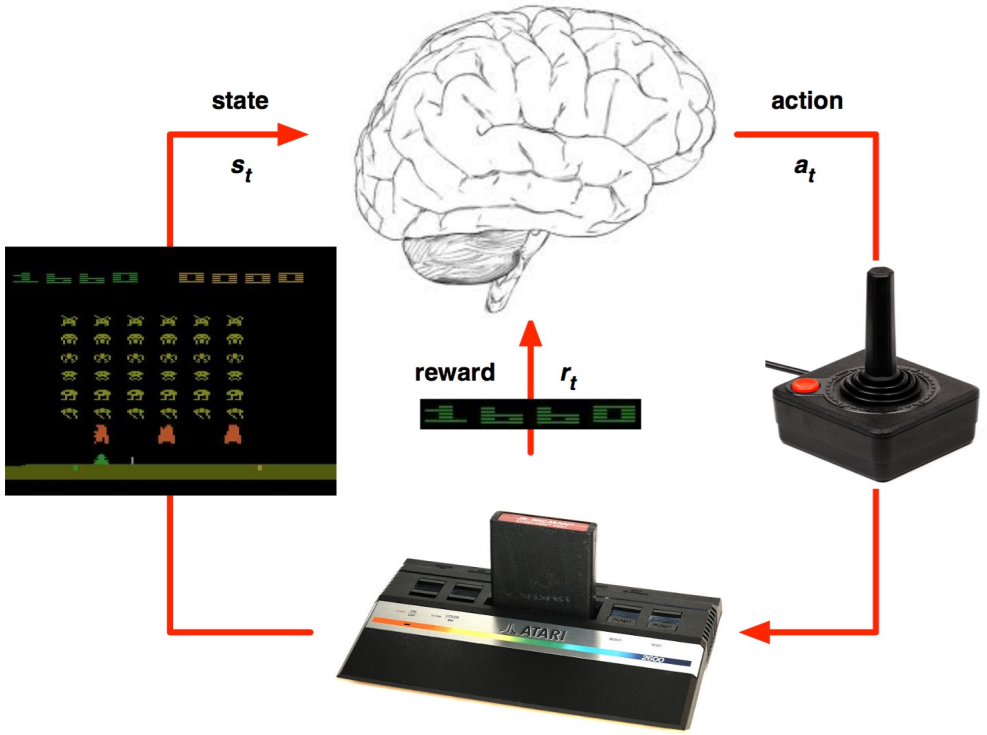


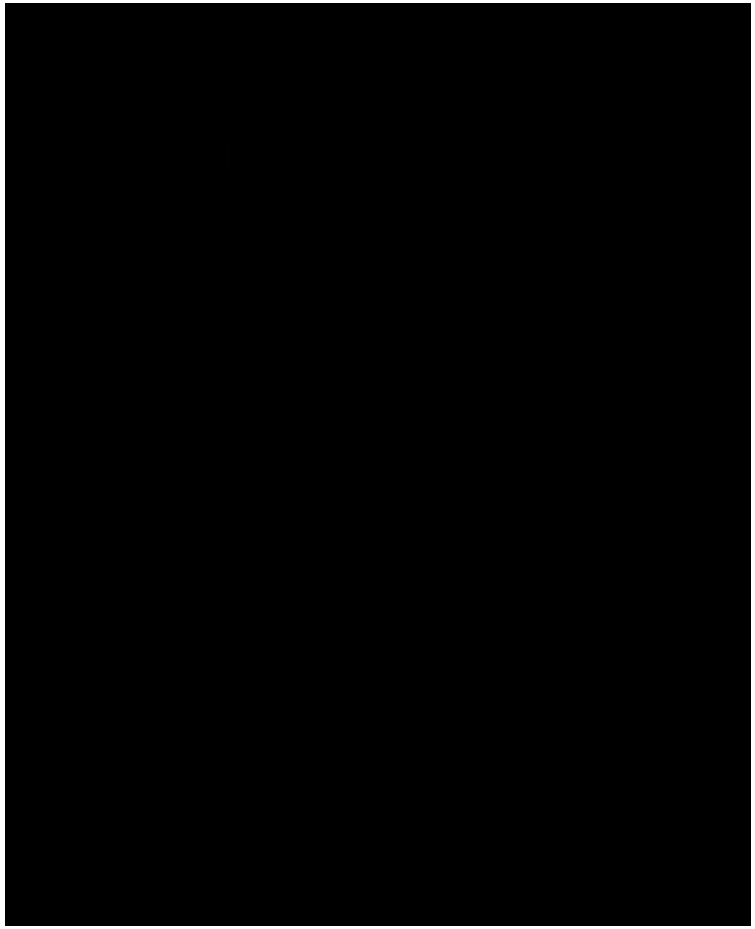
History of reinforcement learning

- Playing games

Deep reinforcement learning

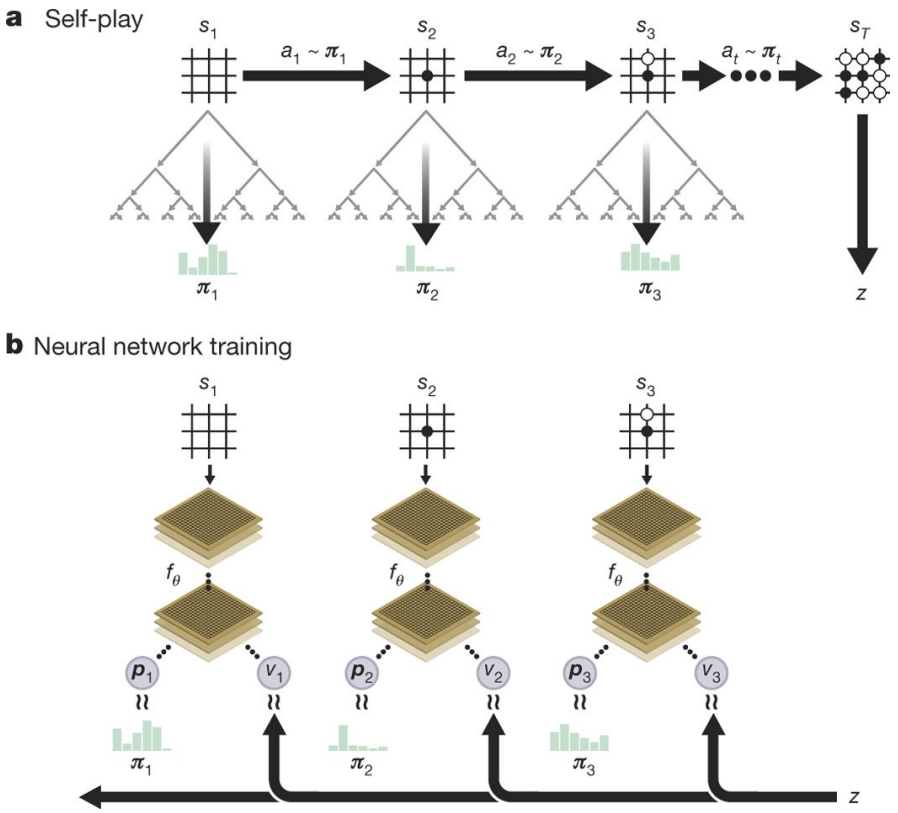
- Playing Atari game with DQN





Deep reinforcement learning

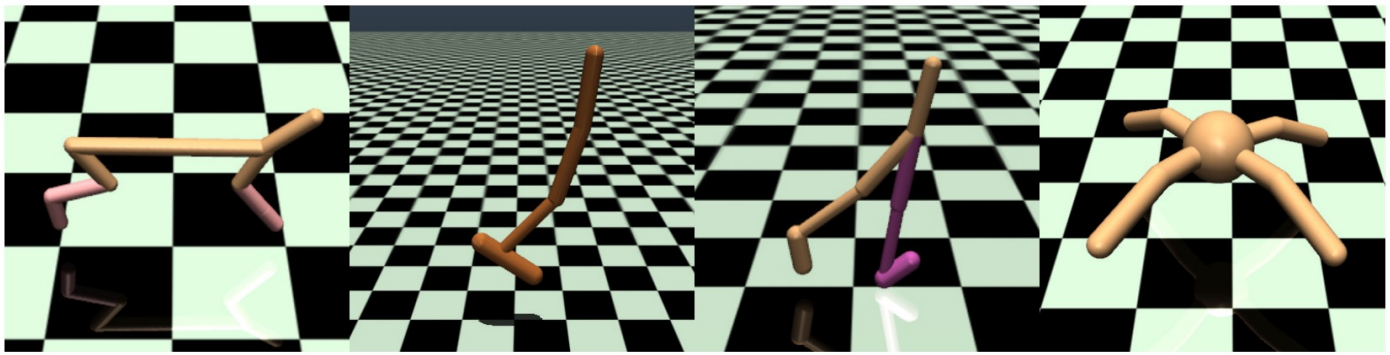
- Playing Go



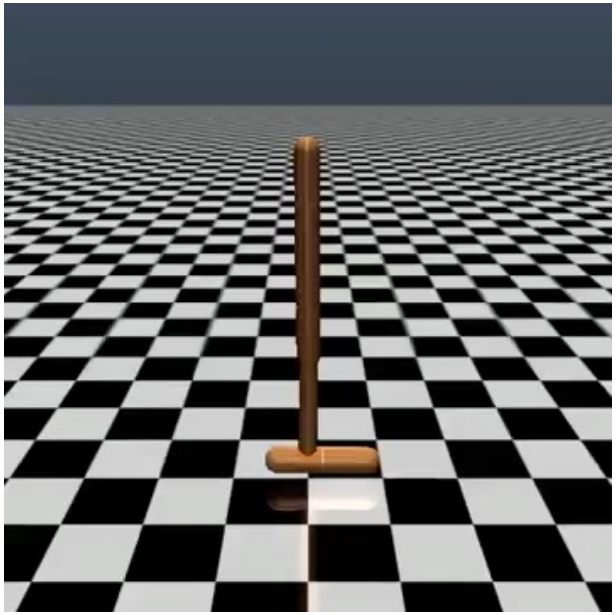
DeepMind (2016): Beat Go champion using reinforcement learning

Deep reinforcement learning

- Deep reinforcement learning for continuous systems

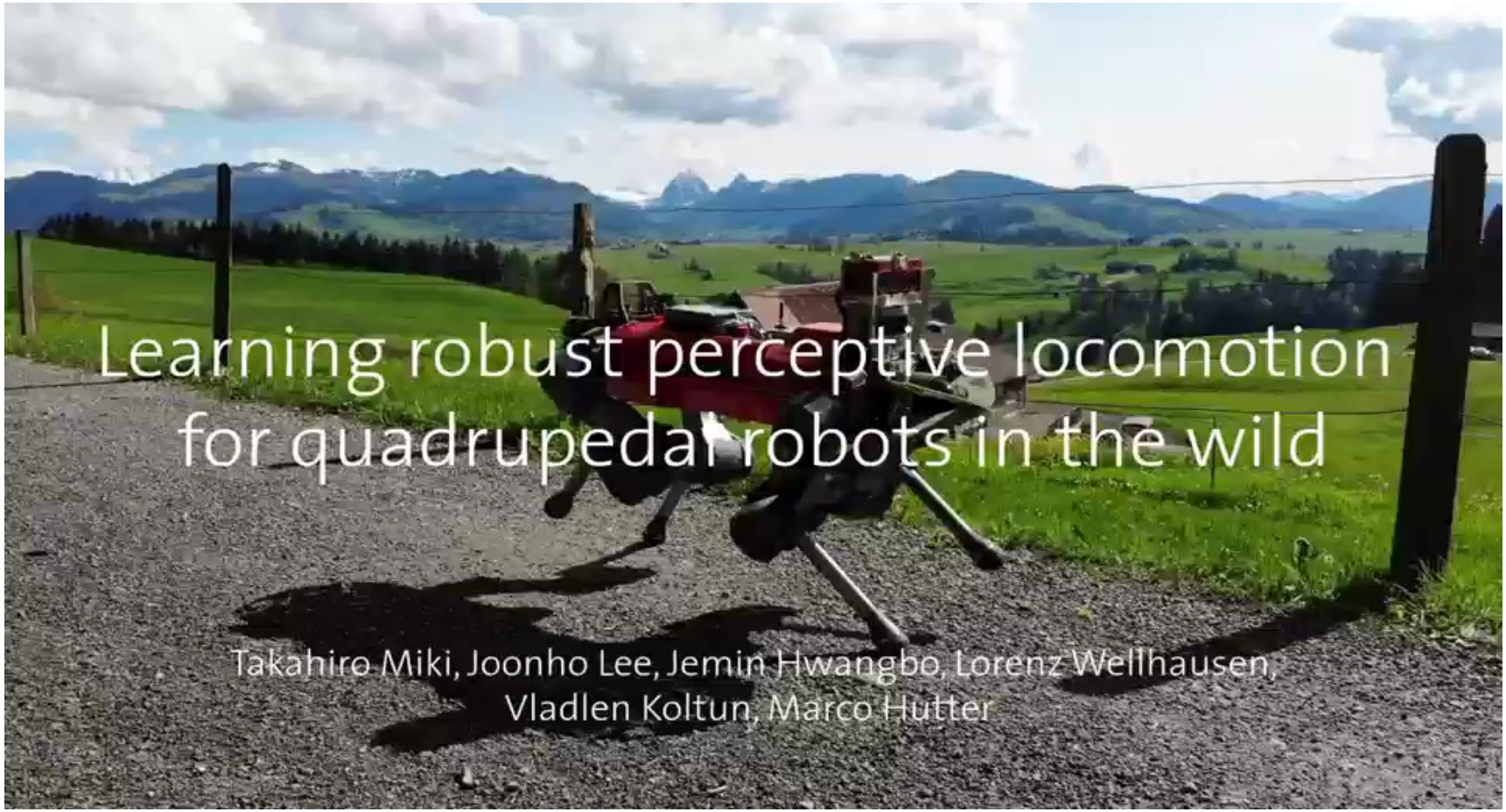


Mujoco simulation



Deep reinforcement learning

- Reinforcement learning for quadrupedal locomotion



ETH (2021): Climb Etzel

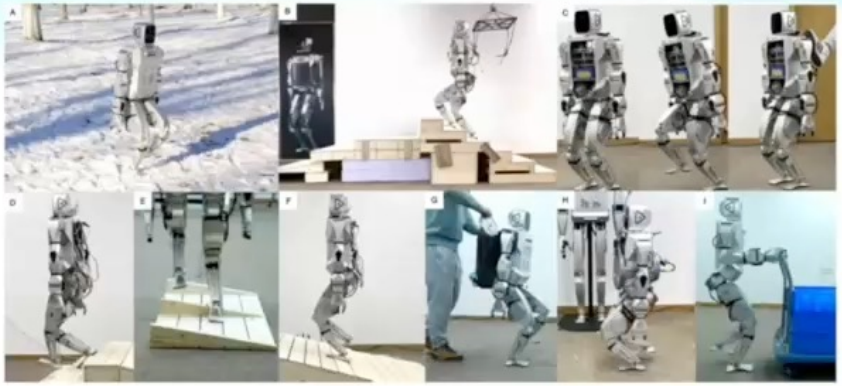
Deep reinforcement learning

- Reinforce learning for humanoid locomotion

Advancing Humanoid Locomotion: Mastering Challenging Terrains with Denoising World Model Learning

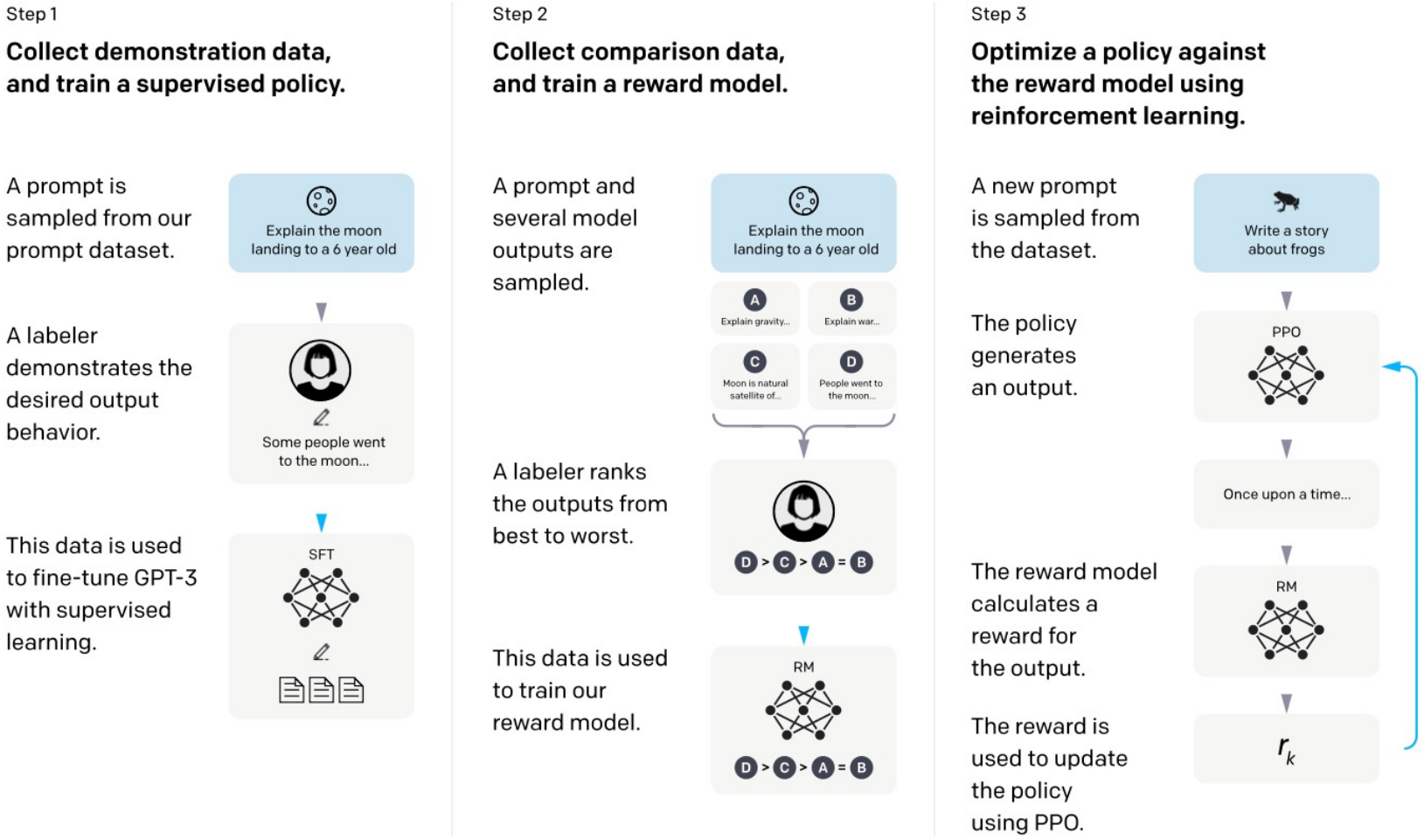
Supplementary Videos

The policy we trained was successfully deployed in all scenarios using a zero-shot sim-to-real approach, without any need for fine-tuning.



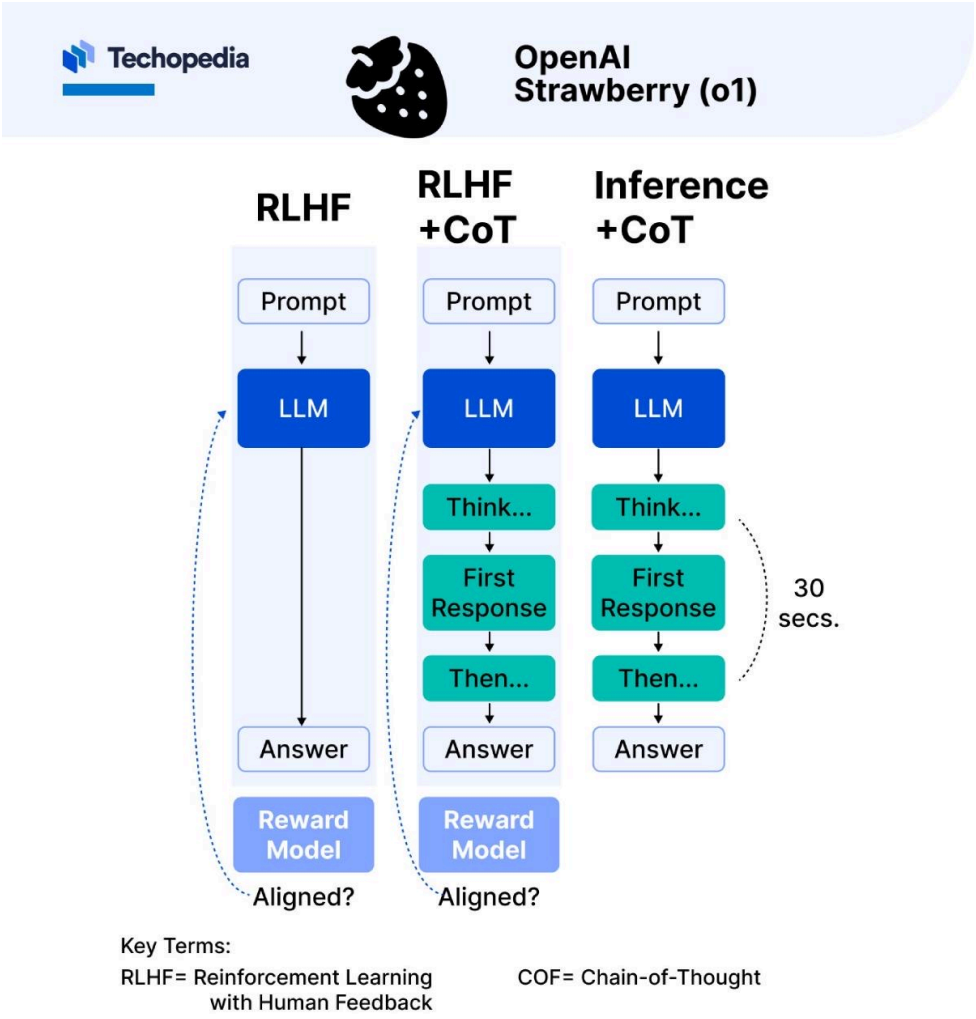
Deep reinforcement learning

Reinforcement learning in ChatGPT (RLHF)



Deep reinforcement learning

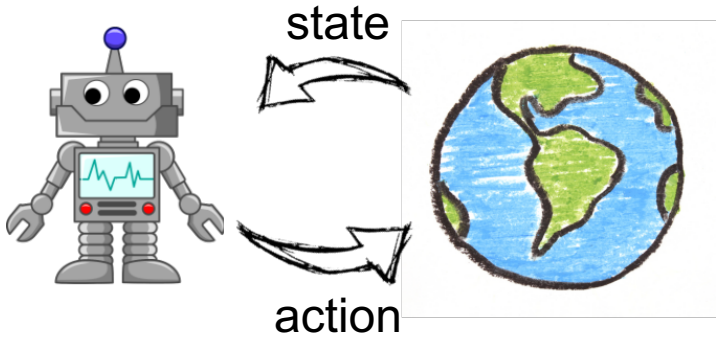
- Reinforcement learning in GPT o1



Contents

Markov Decision Process (MDP)

- Components of **Markov Decision Process (MDP)**
  - State $s \in \mathcal{S}$, action $a \in \mathcal{A}$
  - Environment model $p(s'|s,a)$
  - Reward function $r(s,a)$
  - Discount factor $\gamma \in [0,1]$



- Goal of **reinforcement learning (RL)**
  - Obtain an optimal policy that maximizes discounted accumulative reward under the Markov Decision Process

$$\pi^* \in \underset{\pi \in \Pi}{\operatorname{argmax}}\, \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right]$$

RL vs Optimal Control

Reinforcement Learning	Optimal Control
State-action (s, a)	State-control (x, u)
Transition model $s_{t+1} \sim p(s_{t+1}\|s_t, a_t)$	System dynamics $x_{k+1} = f(x_k, u_k, w_k)$
policy $a_t \sim \pi(s_t)$	Controller $u_k = u(x_k)$
Reward function $r(s, a)$	Cost function $l(x, u)$
Maximize accumulative reward $J = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]$	Minimize accumulative cost $J = \mathbb{E}_w \left[\sum_{k=0}^{\infty} l(x_k, u_k)\right]$

Value Function

Value functions

- State-value function: expected return starting from state s under a policy π

$$v^{\pi}(s) \stackrel{\text{def}}{=} \mathbb{E}_{\pi}\{G_t|s\} = \mathbb{E}_{\pi}\left\{\sum_{i=0}^{+\infty}\gamma^i r_{t+i}\,|s_t = s\right\}$$

- Action-value function: expected return under a policy π when taking action a in state s

$$q^{\pi}(s,a) \stackrel{\text{def}}{=} \mathbb{E}_{\pi}\{G_t|s,a\} = \mathbb{E}_{\pi}\left\{\sum_{i=0}^{+\infty}\gamma^i r_{t+i}\,|s_t = s, a_t = a\right\}$$

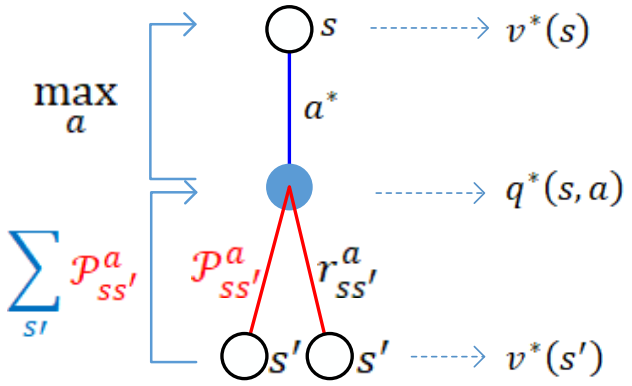
Bellman Equations

Bellman Equation for state value

$$v^*(s) = \max_{a \in \mathcal{A}} q_{\pi^*}(s,a)$$

$$q^*(s,a) = \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} \left(r^a_{ss'} + \gamma v^*(s') \right)$$

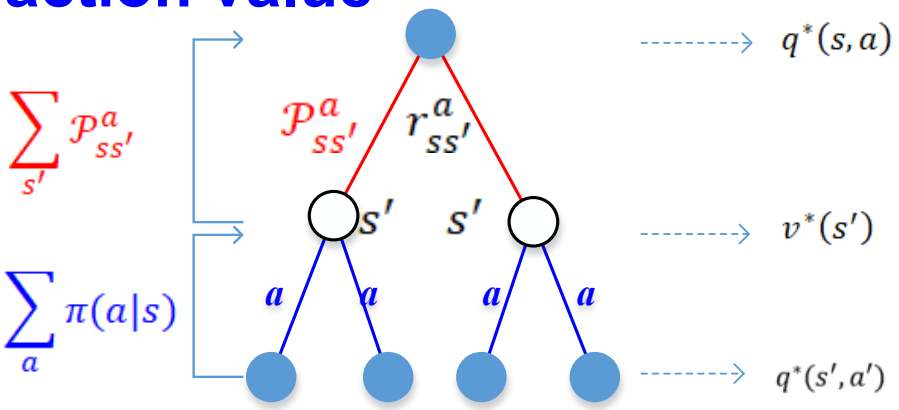
$$v^*(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} \left(r^a_{ss'} + \gamma v^*(s') \right)$$



Bellman Equation for state-action value

$$q^*(s,a) = \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} \left(r^a_{ss'} + \gamma v^*(s') \right)$$

$$= \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} \left(r^a_{ss'} + \gamma \max_{a' \in \mathcal{A}} q^*(s',a') \right)$$



Solving Bellman Equations

Value iteration

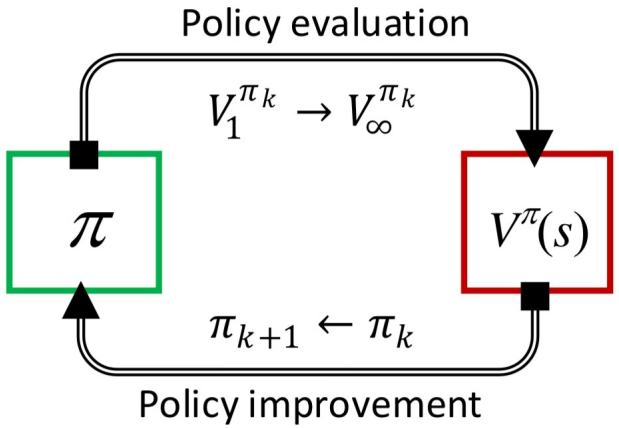
- Iteratively solve Bellman equation to its fixed point

$$v^*(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} \left(r^a_{ss'} + \gamma v^*(s') \right)$$

Numerical Iteration

Policy iteration

- Iteratively exploit value function to find a better policy



Value Iteration Algorithm

- The Bellman Equation

$$v^*(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \left(r_{ss'}^a + \gamma v^*(s') \right)$$

- **Value iteration** iteratively solve the fixed-point of the Bellman Equation

$$V_{j+1}(s) = \max_a \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \left(r_{ss'}^a + \gamma V_j(s') \right)$$

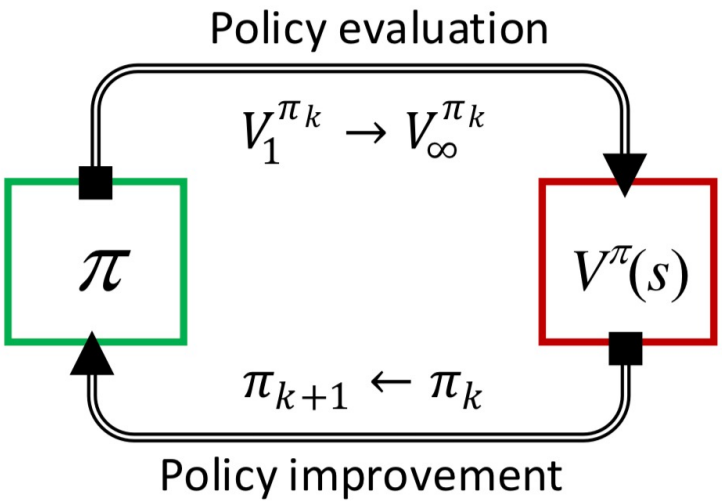
- Optimal policy

$$\pi^*(a|s) = \begin{cases} 1, & a = a^* \\ 0, & \text{otherwise} \end{cases}$$

where $a^* = \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \left(r_{ss'}^a + \gamma v^*(s') \right)$

Policy Iteration

- Policy Iteration
  - (1) Policy evaluation (PEV)
  - Find corresponding "true" value function for a *given* policy π
  - (2) Policy improvement (PIM)
  - Find a *better* policy according to "true" value function $v^{\pi}(s)$



"True" value function:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left\{\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right\}$$

"Optimal" value function:

$$V^{*}(s) = \max_{\pi \in \Pi} \mathbb{E}_{\pi}\left\{\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right\}$$

Policy Evaluation

- Use Bellman Expectation Equation to *estimate* true value function

$$v^{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left\{ \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^{a} \left(r_{ss'}^{a} + \gamma v^{\pi}(s') \right) \right\}, \forall s \in \mathcal{S}$$

  - Environment model $\mathcal{P}_{ss'}^{a}$, policy $\pi(a|s)$ and reward signal $r_{ss'}^{a}$ are known

- Iteratively solve

  - Repeat

$$V_{j+1}^{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left\{ \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^{a} \left(r + \gamma V_{j}^{\pi}(s') \right) \right\}$$

  - Until $V_{j}^{\pi}(s) \to V_{\infty}^{\pi}(s)$ as $j \to \infty$

Policy Improvement

- Goal of **policy improvement**
  - To find a better policy under the "true" value function
- Policy improvement step

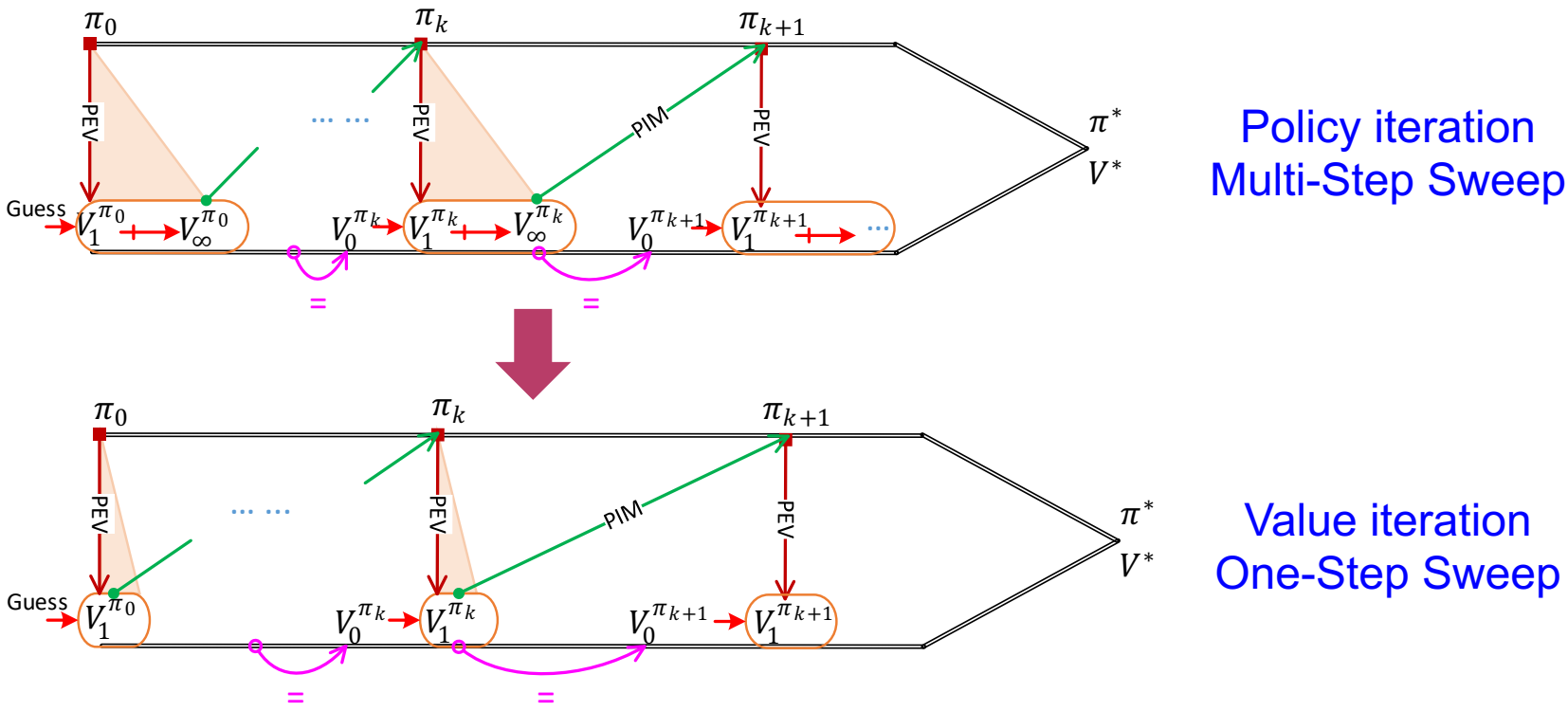
$$\pi'(s) = \arg\max_{a \in \mathcal{A}} Q^{\pi}(s,a) = \arg\max_{a \in \mathcal{A}} \sum_{a \in \mathcal{A}} \pi(a|s) \left\{ \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^{a} \left(r + \gamma V^{\pi}(s') \right) \right\}$$

- The new policy is strictly better than the previous one

$$Q^{\pi}\left(s, \pi'(s)\right) = \max_{a \in \mathcal{A}} Q^{\pi}(s,a) \geq Q^{\pi}\left(s, \pi(s)\right) = V^{\pi}(s)$$

$$\begin{aligned} V^{\pi}(s) &\leq Q^{\pi}(s, \pi'(s)) = \mathbb{E}_{\pi'}\left[r_t + \gamma V^{\pi}(s_{t+1}) | s_t = s \right] \\ &\leq \mathbb{E}_{\pi'}\left[r_t + \gamma Q^{\pi}(s_{t+1}, \pi'(s_{t+1})) | s_t = s \right] \\ &\leq \mathbb{E}_{\pi'}\left[r_t + \gamma r_{t+1} + \ldots | s_t = s \right] = V^{\pi'}(s) \end{aligned}$$

Relationship Between VI and PI

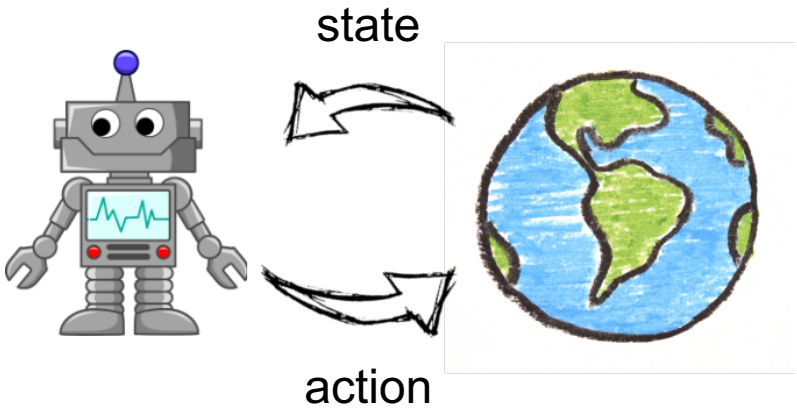


- Value iteration is a special case of policy iteration, where policy evaluation is stopped after just one sweep

Contents

Model-Free Reinforcement Learning

- Value Iteration and Policy Iteration can solve for the optimal value function and optimal policy for MDP
- However, they all need known environment models
- How to solve without knowing the model?
- We can use data obtained by interacting with the environment (This is also one of the essences of RL!)



Monte-Carlo Policy Evaluation

- **Return** is calculated as

$$G_t = \sum_{i=t}^{T-1} \gamma^{i-t} r_i$$

- **Value function** is the **expected** return

$$V^{\pi}(s) = \mathbb{E}_{\pi}\{G_{t:T} | s_t = s\}$$

- **Monte-Carlo policy evaluation** use **empirical** mean return instead of expected return
  - Increment episode count number $N(s) \leftarrow N(s) + 1$
  - Increment total return $S(s) \leftarrow S(s) + G_t$
  - Value estimated by $V(s) = S(s)/N(s)$
  - By law of large numbers, $V(s) \rightarrow V_{\pi}(s)$ as $N(s) \rightarrow \infty$

Incremental Monte-Carlo Updates

Incremental mean

- The mean $\mu_1, \mu_2, \ldots$ of a sequence $x_1, x_2, \ldots$ can be computed incrementally

$$\begin{aligned}
\mu_k &= \frac{1}{k} \sum_{j=1}^{k} x_j \\
&= \frac{1}{k} \left(x_k + \sum_{j=1}^{k-1} x_j \right) \\
&= \frac{1}{k} \left(x_k + (k-1)\mu_{k-1} \right) \\
&= \mu_{k-1} + \frac{1}{k} (x_k - \mu_{k-1})
\end{aligned}$$

Incremental Monte-Carlo Updates

- We have update $V(s)$, incrementally with a new episode

$$s_1, a_1, r_1, \ldots, s_T$$

- For each state s_t with return G_t, we have

$$N(s_t) \leftarrow N(s_t) + 1$$

$$V(s_t) \leftarrow V(s_t) + \frac{1}{N(s_t)} (G_t - V(s_t))$$

- More generally,

$$V(s_t) \leftarrow V(s_t) + \alpha (G_t - V(s_t))$$

Temporal-Difference Policy Evaluation

- **Incremental Monte-Carlo**
  - Update value $V(s_t)$ toward *actual* return G_t

$$V(s_t) \leftarrow V(s_t) + \alpha(G_t - V(s_t))$$

- **Temporal difference learning**
  - Update value $V(s_t)$ toward *estimated* return $r_t + \gamma V(s_{t+1})$

$$V(s_t) \leftarrow V(s_t) + \alpha(r_t + \gamma V(s_{t+1}) - V(s_t))$$

  - $r_t + \gamma V(s_{t+1})$ is called the *TD target*
  - $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$ is called the *TD error*

MC vs TD

- **Monte Carlo**
  - MC must wait until end of episode before return is known
  - MC can only learn from complete sequences
  - MC has high variance, zero bias
- **Temporal difference**
  - TD can learn online after every step
  - TD can learn from incomplete sequences
  - TD has low variance, some bias

Policy Iteration with MC PEV

- **Policy evaluation**
  - Monte-Carlo policy evaluation, $V^{\boldsymbol{\pi}} = V$

$$G_t = \sum_{i=t}^{T-1} \gamma^{i-t} r_i$$

$$V(s_t) \leftarrow V(s_t) + \alpha(G_t - V(s_t))$$

- **Policy improvement**
  - Greedy policy improvement

$$\pi'(s) = \arg\max_{a \in \mathcal{A}} \left\{ \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^{a} \left(r + \gamma V^{\pi}(s') \right) \right\}$$

Model-Free Policy Iteration

- Problem: greedy policy improvement **requires model**

$$\pi'(s) = \arg\max_{a \in \mathcal{A}} \left\{ \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^{a} \left(r + \gamma V^{\pi}(s') \right) \right\}$$

Transition model

- Idea: use $\boldsymbol{Q^{\pi}(\mathbf{s}, \mathbf{a})}$ for policy improvement

$$\pi'(s) = \arg\max_{a \in \mathcal{A}} Q^{\pi}(s, a)$$

- **Policy evaluation**
  - Monte-Carlo policy evaluation, $Q^{\boldsymbol{\pi}} = Q$
- **Policy improvement**
  - Greedy policy improvement over $Q^{\boldsymbol{\pi}}$

Problems of Greedy Policy

Is greedy policy a good choice?

$$\pi'(s) = \arg\max_{a \in \mathcal{A}} Q^{\pi}(s, a)$$

"Behind one door is tenure - behind the other is flipping burgers at McDonald's."

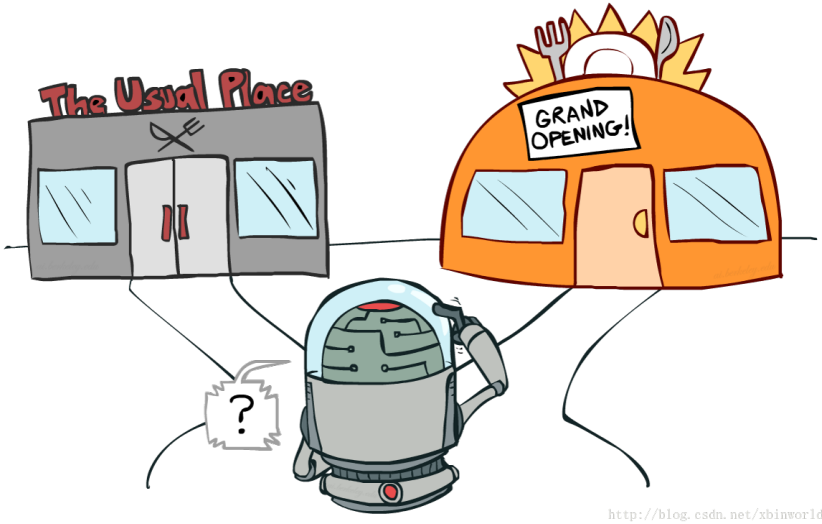
- There are two doors in front of you.
- You open the left door and get reward 0
  $V(left) = 0$
- You open the right door and get reward $+1$
  $V(right) = +1$
- You open the right door and get reward $+3$
  $V(right) = +2$
- You open the right door and get reward $+2$
  $V(right) = +2$

$\vdots$

- Are you sure you've chosen the best door?

Exploration and Exploitation

- **Exploration** is equally important to **Exploitation**



Go to your favorite restaurant?

Try a new restaurant?

Monte-Carlo RL

- For every episode
  - Policy evaluation
    - Monte-Carlo policy evaluation for Q

$$G_t = \sum_{i=t}^{T-1} \gamma^{i-t} r_i$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(G_t - Q(s_t, a_t))$$

  - Policy improvement
    - **ϵ-Greedy policy improvement**

$$\pi^{\epsilon}(a|s) \leftarrow \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}|}, & if\ a = \arg\max_{a \in \mathcal{A}} Q^{\pi}(s, a) \\ \frac{\epsilon}{|\mathcal{A}|}, & otherwise \end{cases}$$

SARSA

- **State-action-reward-state-action (Sarsa)**
- For every time step
  - Policy evaluation
    - **Temporal-Difference policy evaluation** for Q

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

  - Policy improvement
    - ϵ-Greedy policy improvement

$$\pi^{\epsilon}(a|s) \leftarrow \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathcal{A}|}, & if\ a = \arg\max_{a \in \mathcal{A}} Q^{\pi}(s, a) \\ \frac{\epsilon}{|\mathcal{A}|}, & otherwise \end{cases}$$

Off-Policy Learning

- The above RL methods are **on-policy**
  - PEV: Evaluate Q^{π} with target policy π
  - PIM: Improve π
  - π should be ϵ-greedy to maintain exploration
- We can also use **off-policy** learning
  - PEV: Evaluate Q^{π} with behavior policy b
  - PIM: Improve π
  - b can be arbitrary and π can be greedy
- Why Off-policy important?
  - Re-use experience from old policies
  - Learn the optimal policy with a different exploration policy

Q-Learning

- Now in off-policy TD, choose target policy π to be greedy:

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

- Choose **behavior policy** to be ϵ-greedy w.r.t. $\mathrm{Q}(s,a)$
- The Q-value update becomes

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

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

- This is the Q-Learning algorithm, which converges to the optimal action-value function $Q(s,a) \rightarrow Q^*(s,a)$

Contents

Motivation

- The above tabular RL algorithms are for discretized space
- But now can deal with model-free, MDP problem.
- Unable to deal with large-scale problems,
  - Game Go: 10^{170} states
  - Robotics: continuous space

- Tabular RL suffers from "Curse of dimensionality"

Value Function Approximation for RL

- Solution for large-scale/continuous RL problems:
  - Value function approximation
    - Replace tabular value by a parameterized function
    - $V(s|w) \approx V^{\pi}(s), \forall s \in \mathcal{S}$
    - $Q(s,a|w) \approx Q^{\pi}(s,a), \forall s \in \mathcal{S}$
  - Efficiently apply to all states, either large-scale or continuous
  - Generalize from seen states to unseen states

On-Policy Objective Function

Objective function

- Minimize the mean squared value error under state distribution

$$V(s|w) \approx V^{\pi}(s), \forall s \in \mathcal{S}$$

$$\min_{w} J(w) = \mathbb{E}_{s\sim d^{\pi}}\left\{\left(V^{\pi}(s) - V(s|w)\right)^2\right\}$$

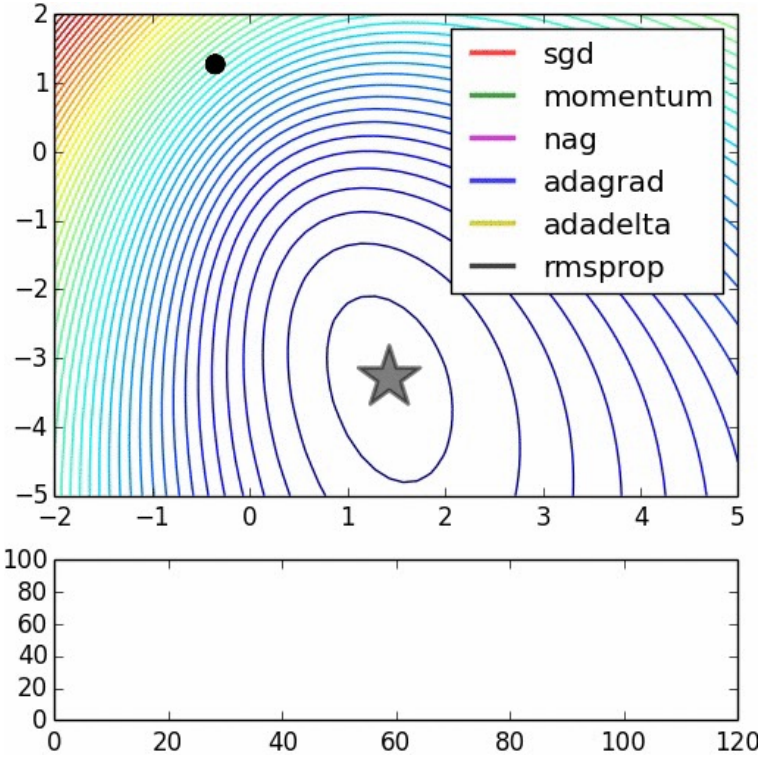
$$= \sum_{s} d^{\pi}(s) \cdot \left(V^{\pi}(s) - V(s|w)\right)^2$$

- where $d^{\pi}(s)$ is the state distribution under policy π
- $d^{\pi}(s)$ indicates how frequent the state s can be visited under π, and is used to weight the error term

Gradient Descent Algorithm

- Let $J(w)$ be a differentiable function of parameter w
- Gradient of $J(w)$

$$\nabla_w J(w) = \begin{bmatrix} \frac{\partial J(w)}{\partial w_1} \\ \cdots \\ \cdots \\ \frac{\partial J(w)}{\partial w_n} \end{bmatrix}$$



- Adjust w in direction of gradient

$$w \leftarrow w - \alpha \nabla_w J(w)$$

  - $\alpha < 0$ is a step-size parameter

Gradient Descent of On-Policy Objective

- Gradient of the on-policy objective function

$$J(w) = \sum_s d^\pi(s)\big(V^\pi(s) - V(s|w)\big)^2 = \mathbb{E}_{s\sim d^\pi}\Big\{\big(V^\pi(s) - V(s|w)\big)^2\Big\}$$

$$\nabla_w J(w) \propto \mathbb{E}_{s\sim d^\pi}\big\{\big(V(s|w) - V^\pi(s)\big)\nabla_w V(s|w)\big\}$$

- Approximate expectation by averaging a batch of samples

$$\nabla J(w) \propto \mathbb{E}_{s\sim d^\pi}\big\{\big(V(s|w) - V^\pi(s)\big)\nabla_w V(s|w)\big\}$$
$$\approx \frac{1}{T}\sum_{t=1}^{T}(V(s_t|w) - V^\pi(s_t))\nabla_w V(s_t|w)$$

  - where samples s_t come from the states rollout with π
  - If T=1 ⇨ Stochastic gradient descent (SGD)

Deep Q-Networks (DQN)

DQN uses experience replay and fixed Q-targets

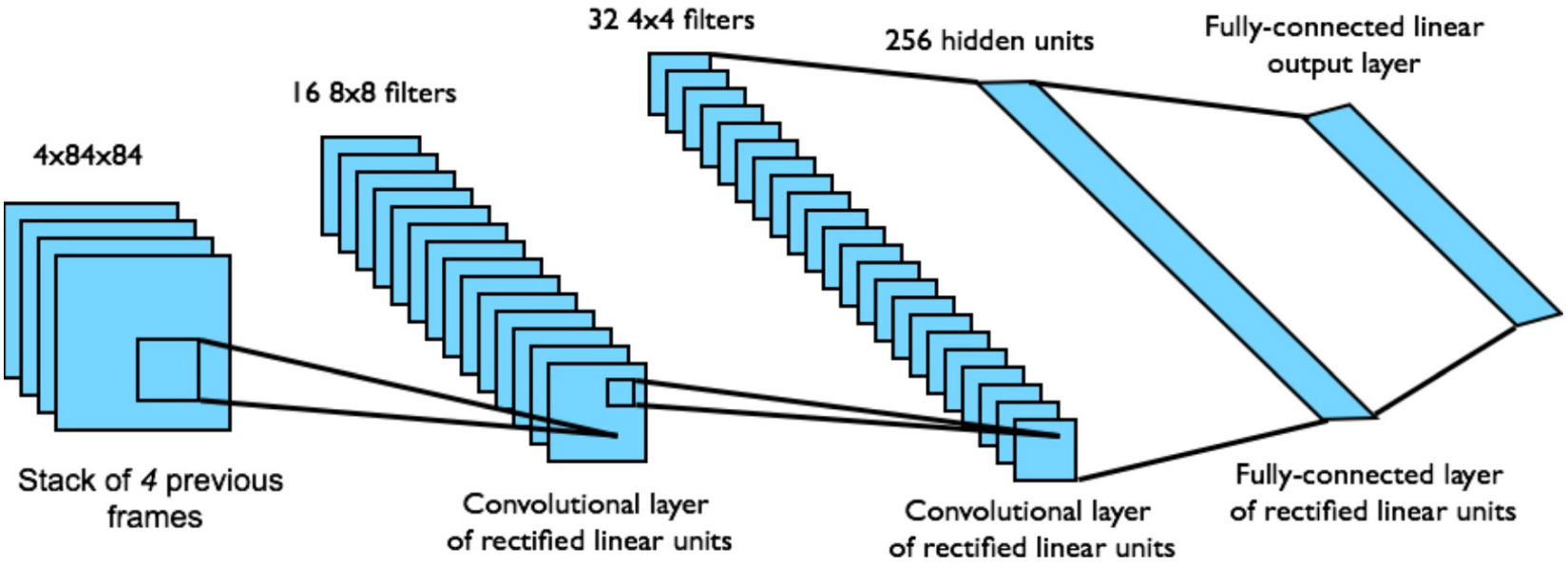
- Take action a_t according to ϵ-greedy policy
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory $\mathcal{D}$
- Sample random mini-batch of transitions (s, a, r, s') from $\mathcal{D}$
- Compute Q-learning targets w.r.t. old, fixed parameters w^-
- Optimise MSE between Q-network and Q-learning targets

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

- Using variant of stochastic gradient descent

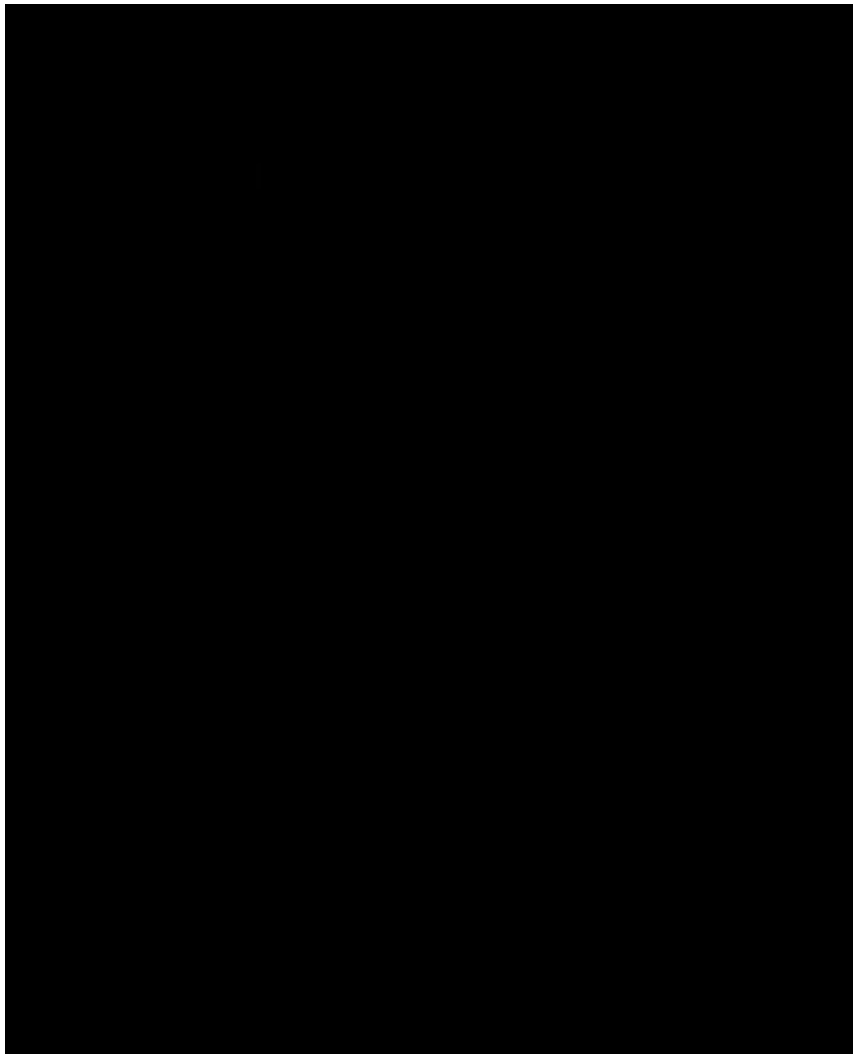
Deep Q-Networks (DQN)

- 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



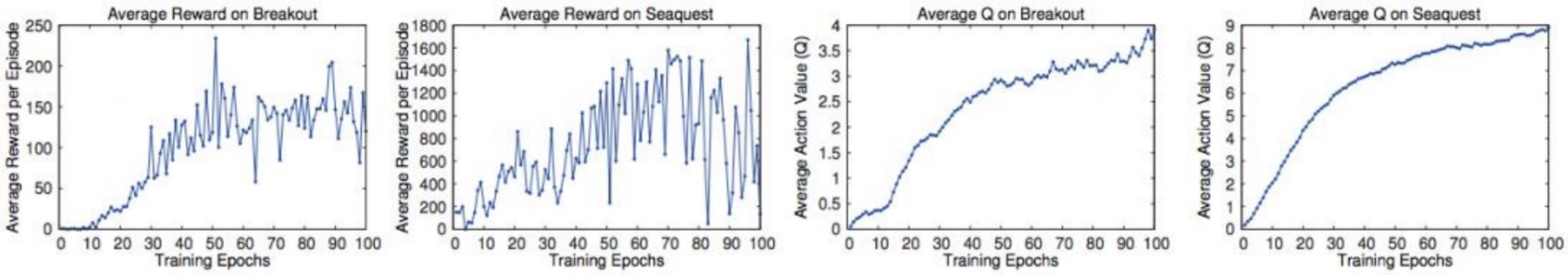
Deep Q-Networks (DQN)

- Playing Atari game with DQN

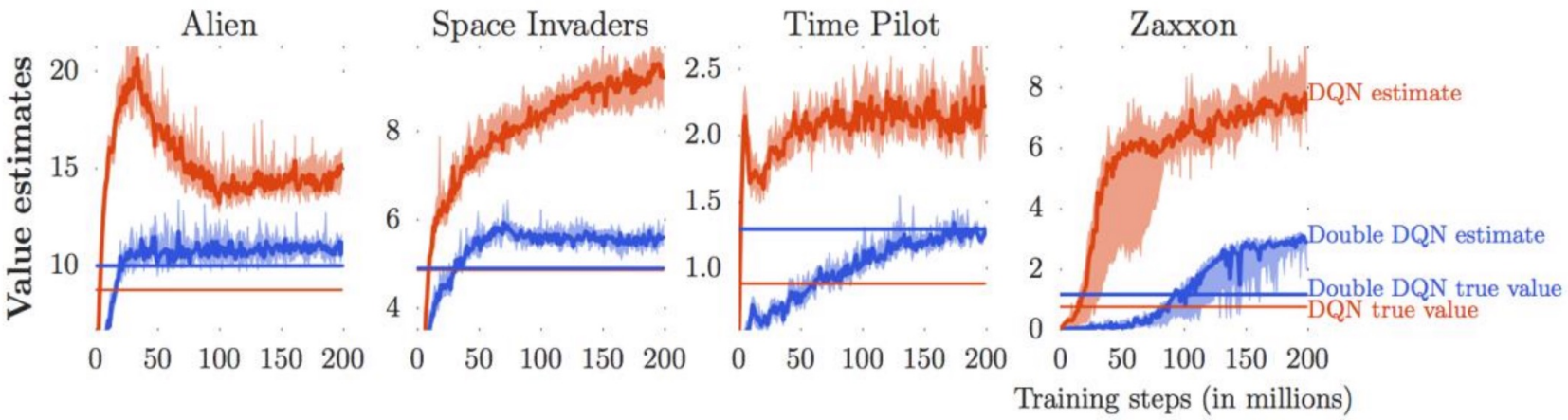


Are Q-Value Accurate?

- Learning curve of Q-values

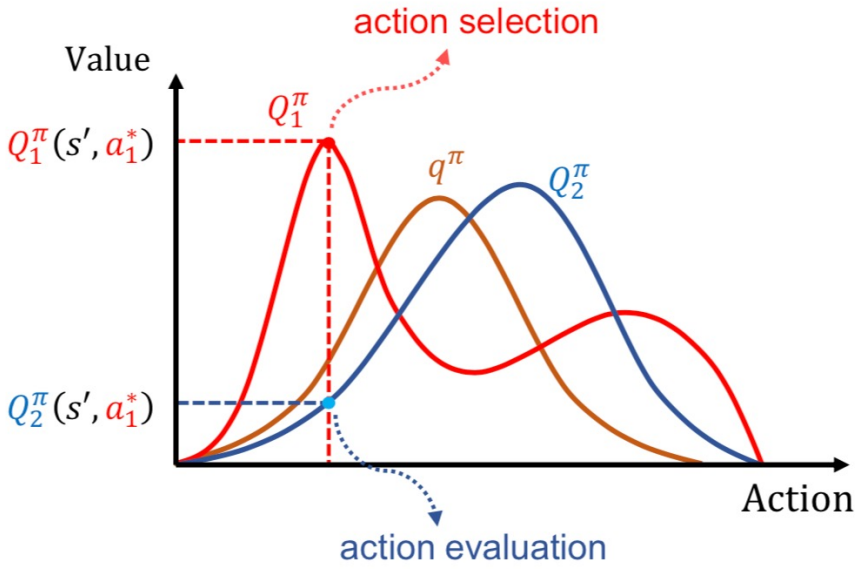


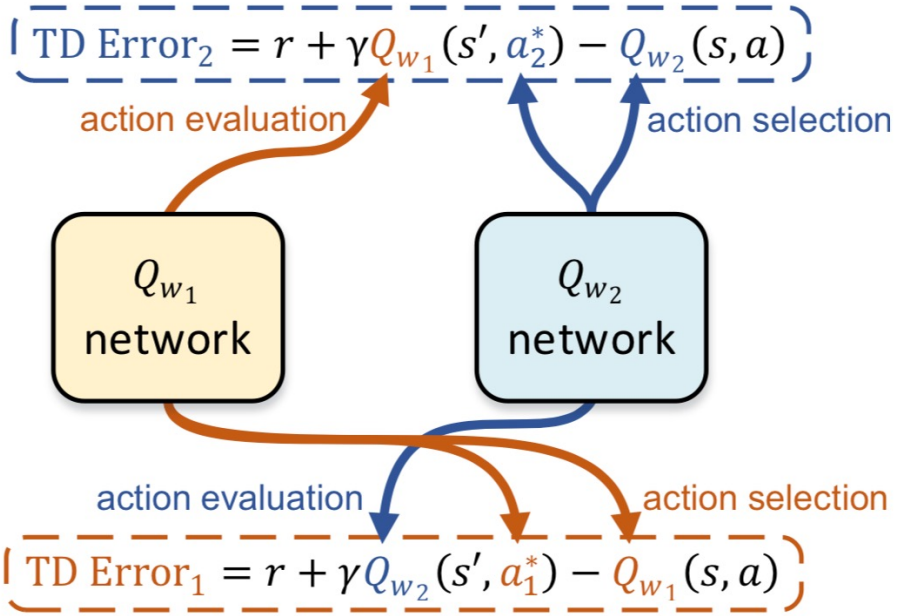
- Does the learned Q-value reflect the true Q-value?



Alleviate The Overestimation Problem

- The overestimated value $\max_{a'} Q_{\bar{w}}(s', a')$ is then used as the target to update the Q value, which leads to the overestimation of the Q value
- Idea: use an independent different Q network to calculate the learning target

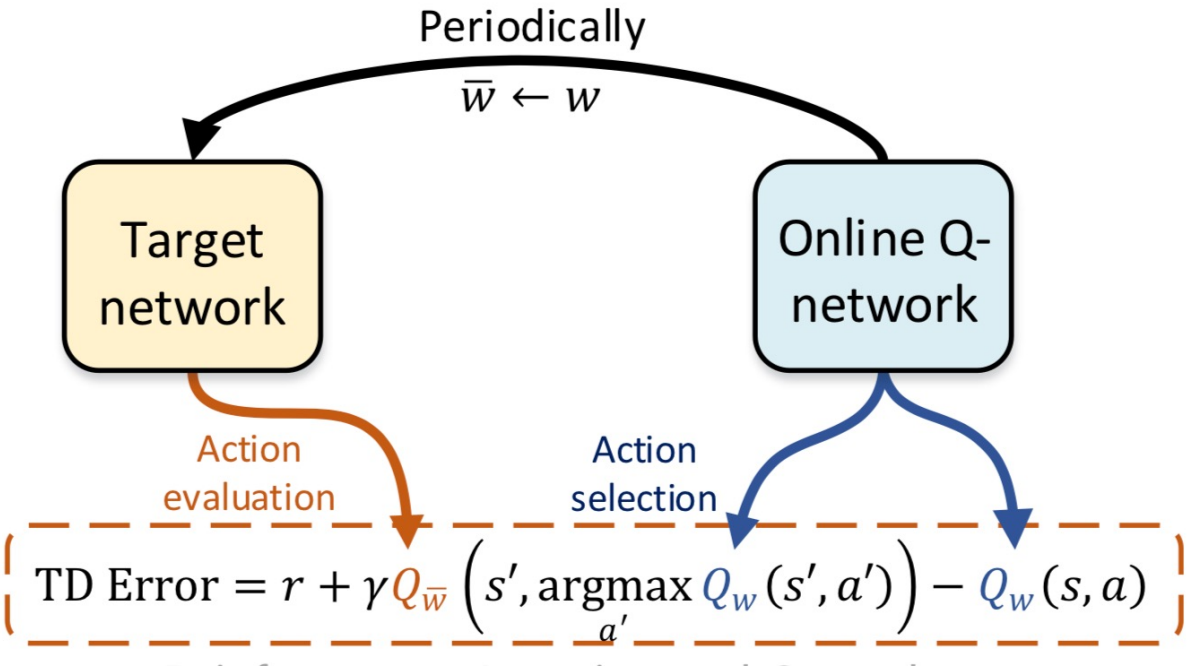




Double DQN (DDQN)

- In practice, just use the current and target networks as the two different Q networks
- **Double DQN (DDQN)**

$$J(w) = \mathbb{E}_{s,a,s' \sim \mathcal{D}_{\text{Replay}}} \left\{ \left(r + \gamma Q_{\bar{w}}\left(s', a^*(s')\right) - Q_w(s,a) \right)^2 \right\}$$



How About Continuous Actions?

- So far the policy is obtained by maximizing the Q value

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

- The action space is restricted to discrete
- Most robotics problems require continuous action space
  - Autonomous driving: steering angle, throttle…
  - Robotic arm: torques…
- However, how to maximize the Q value with continuous actions?

$$\text{TD Error} = r + \gamma Q_{\bar{w}}\left(s', \underset{a'}{\text{argmax}}\, Q_w(s', a')\right) - Q_w(s, a)$$

This is particularly problematic

Policy Approximation

- So far, we have only used function approximation for values, which enables continuous state space

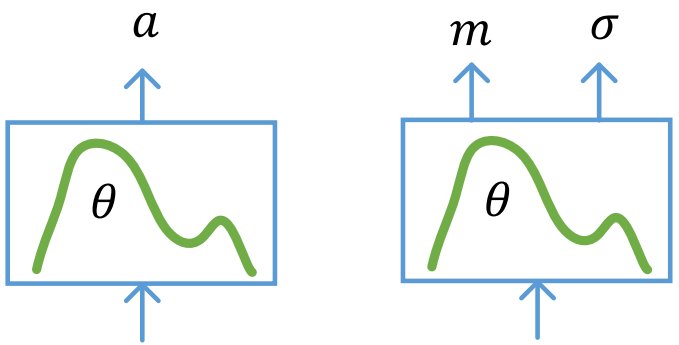
$$V(s|w) \approx V^{\pi}(s) \qquad Q(s,a|w) \approx Q^{\pi}(s,a)$$

- Similarly, we can use function approximation for policy
  - Deterministic policy:

$$\pi(a|s,\theta) = \delta(a) = \pi_{\theta}(s)$$

  - Stochastic policy:

$$\pi(a|s,\theta) \sim N(m_{\theta}(s), \sigma_{\theta}^{2}(s))$$



Q-Learning with Continuous Actions

- The policy $\pi(s) = \arg\max_a Q_w(s, a)$ is deterministic
- Idea: train a deterministic policy network to approximate the maximization policy:

$$\pi_\theta(s) \approx \arg\max_a Q_w(s, a)$$

- How to approximate?
- Just use gradient ascent to maximize the Q value:

$$J_{\text{Actor}}(\theta) = \mathbb{E}_{s \sim \mathcal{D}_{\text{Replay}}}\{Q_w(s, \pi_\theta(s))\}$$

- where

$$\frac{dQ_w}{d\theta} = \frac{d\mathbf{a}}{d\theta}\frac{dQ_w}{d\mathbf{a}}$$

Q-Learning with Continuous Actions

- The Q value is learned almost in the same way, but the maximization policy is replaced with the approximated policy

$$J_{\text{Critic}}(w) = \mathbb{E}_{s,a,s' \sim \mathcal{D}_{\text{Replay}}} \left\{ \left(r + \gamma Q_{\bar{w}} \left(s', \pi_{\bar{\theta}}(s') \right) - Q_w(s,a) \right)^2 \right\}$$

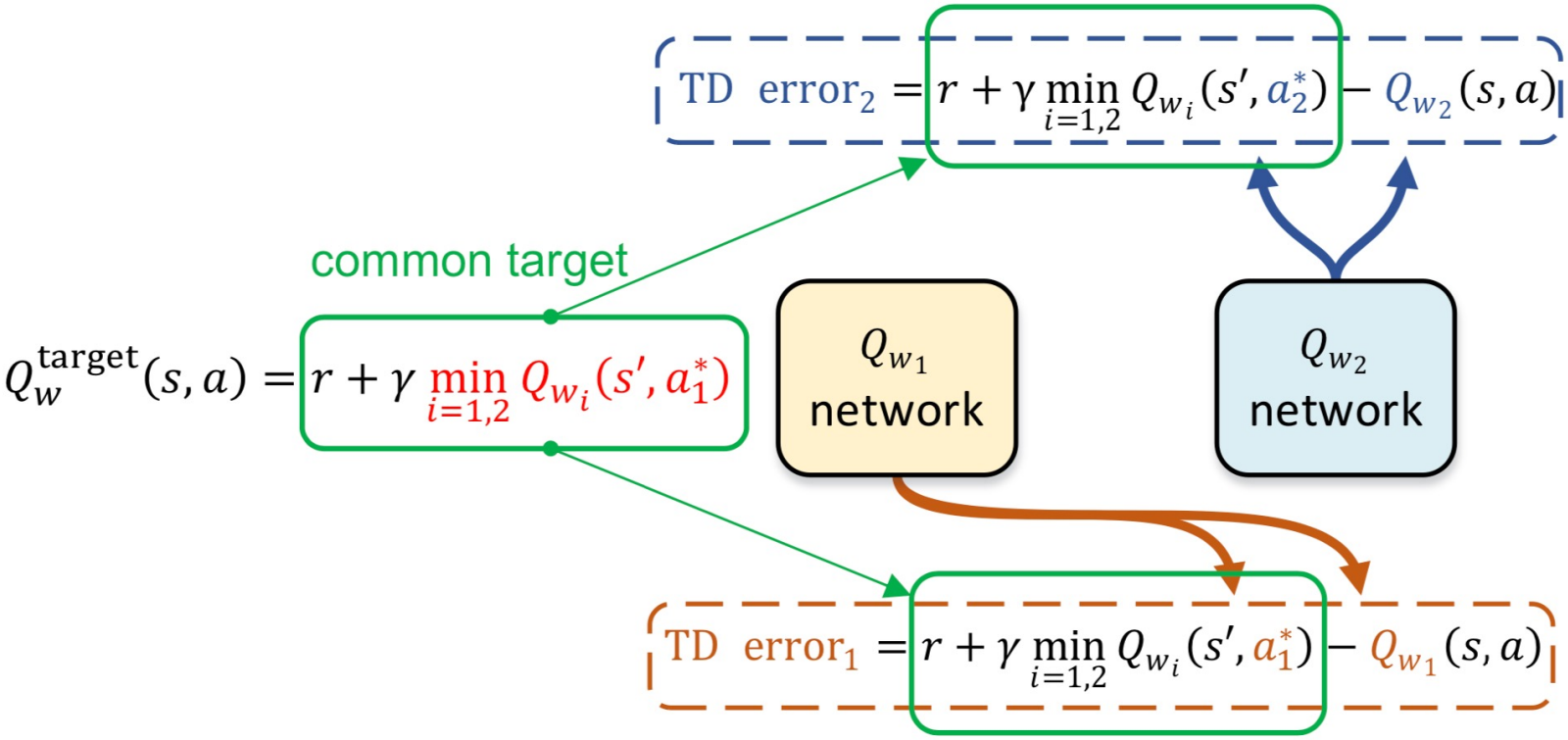
- Note that for the policy, we also keep a target network $\pi_{\bar{\theta}}(s)$, which is used to calculate the TD target
- This is the **Deep Deterministic Policy Gradient (DDPG)** algorithm
- There are two Q-networks (one training), two policy networks (one training)

Deep Deterministic Policy Gradient (DDPG)

- Solve reinforcement learning problems with continuous action space

Further Alleviate Overestimation

- Instead of use one value network to compute the target of another, we can choose the minimum of two value-functions as the common target to further alleviate the overestimation problem



Twin Delayed DDPG (TD3)

- This leads to the **Twin Delayed DDPG (TD3)** algorithm

$$Q_{\bar{w}}^{\text{target}}(s,a) = r + \gamma \min_{i=1,2} Q_{\bar{w}_i}\left(s', \pi_{\bar{\theta}}(s')\right)$$

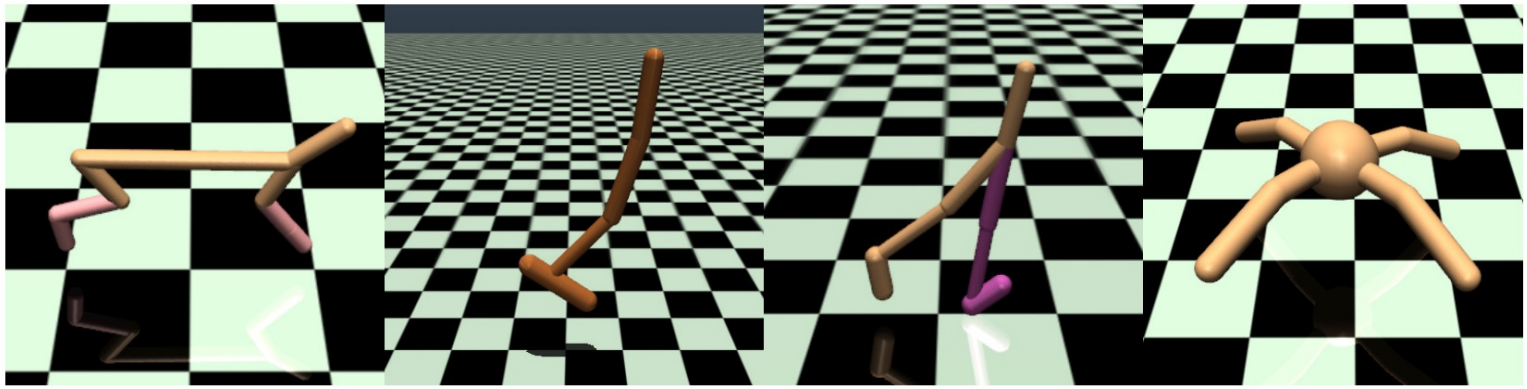
$$J_{\text{Critic}}(w_1) = \mathbb{E}_{s,a,s' \sim \mathcal{D}_{\text{Replay}}}\left\{\left(Q_{\bar{w}}^{\text{target}}(s,a) - Q_{w_1}(s,a)\right)^2\right\}$$

$$J_{\text{Critic}}(w_2) = \mathbb{E}_{s,a,s' \sim \mathcal{D}_{\text{Replay}}}\left\{\left(Q_{\bar{w}}^{\text{target}}(s,a) - Q_{w_2}(s,a)\right)^2\right\}$$

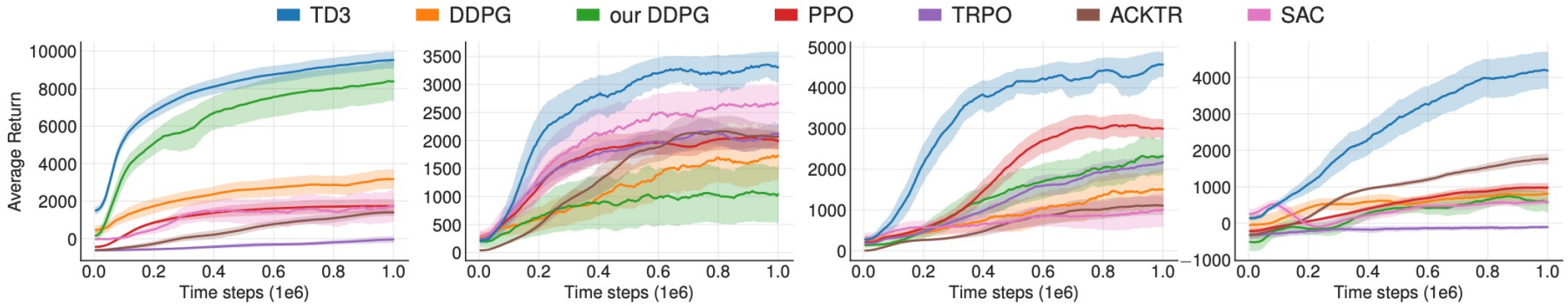
- Furthermore, TD3 delays the policy update, i.e., the weights of policy network change less frequently
- Waits until the value approximation error becomes smaller, which improves convergence
- There are four Q-networks (two training), two policy networks (one training)

Twin Delayed DDPG (TD3)

- Learn control policy on the MuJoCo Benchmark



- Learning curves of TD3



(a) HalfCheetah-v1 (b) Hopper-v1 (c) Walker2d-v1 (d) Ant-v1

Soft Value Function

- A big challenge for deep RL is its sample efficiency, which largely depends on a good strategy to balance exploration & exploration.
- Idea: include exploration into the RL objective
  - Augment the reward function with policy entropy

$$r_{\text{aug}}(s_t, a_t, s_{t+1}) = r(s_t, a_t, s_{t+1}) + \alpha\mathcal{H}\big(\pi(\cdot\,|s_t)\big)$$

  - Propose the maximum entropy RL framework

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left\{\sum_{i=0}^{\infty}\gamma^{i}\left(r_{t+i} + \alpha\mathcal{H}\big(\pi(\cdot\,|s_{t+i})\big)\right)\middle|\, s_t = s\right\}$$

Soft Actor Critic (SAC)

Soft Q-function

- Use the common target to update two Q-networks (like TD3)

$$J_{\text{Critic}}(w_i) = \mathbb{E}_{s,a,s' \sim \mathcal{D}_{\text{replay}}} \left\{ \left(Q_{\bar{w}}^{\text{target}}(s,a) - Q_{w_i}(s,a) \right)^2 \right\}, \forall i = 1,2$$

- Common target is calculated through soft Q-function

$$Q_{\bar{w}}^{\text{target}}(s,a) = r + \gamma \left(\min_{i=1,2} Q_{\bar{w}_i}(s',a') - \alpha \log \pi_\theta(a'|s') \right)$$

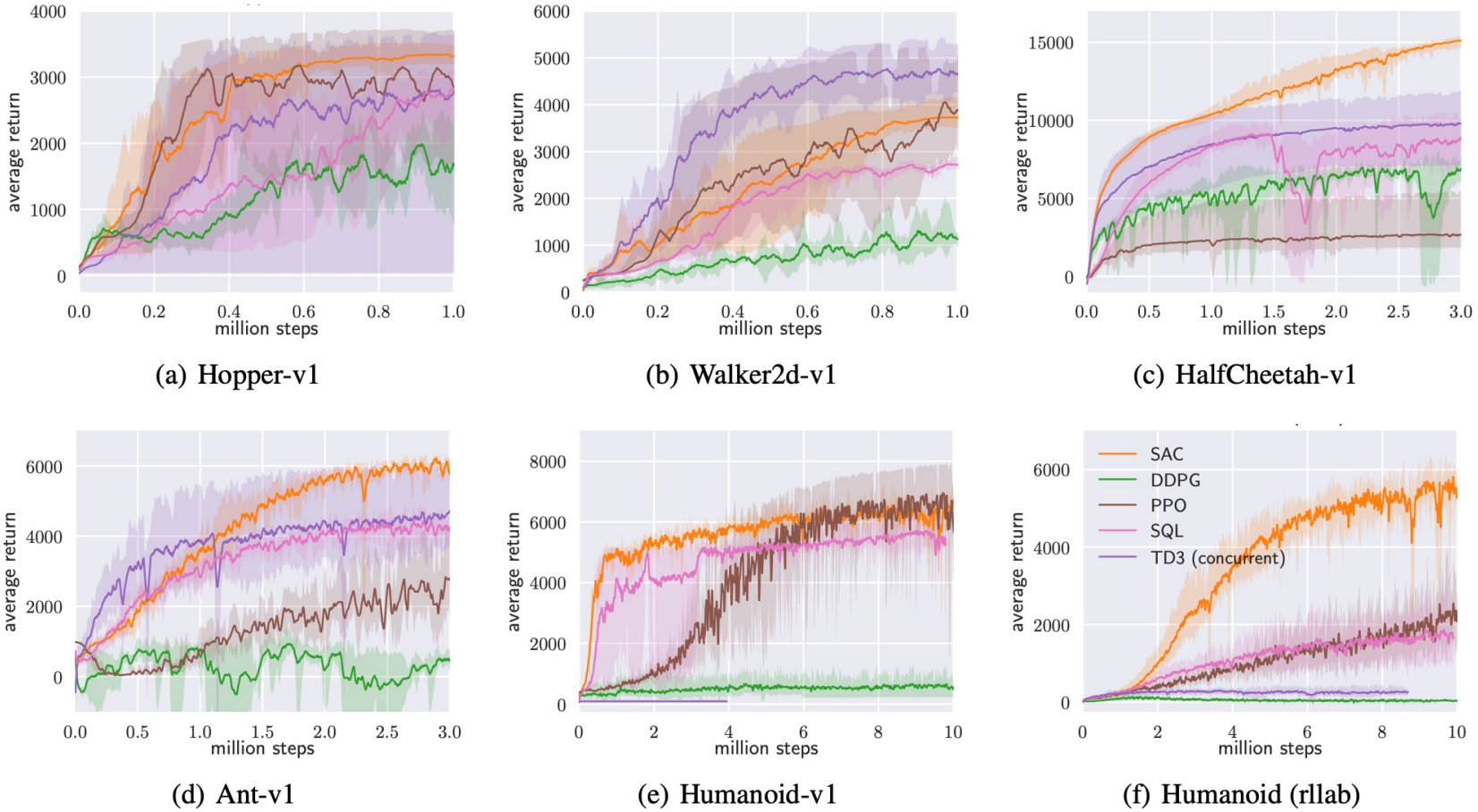
Stochastic policy

- Maximize the soft Q-function

$$J_{\text{Actor}}(\theta) = \mathbb{E}_{s \sim \mathcal{D}_{\text{replay}}, a \sim \pi_\theta} \{ Q^{\pi_\theta}(s,a) - \alpha \log \pi_\theta(a|s) \}$$

Soft Actor Critic (SAC)

- Compare the learning curve with TD3
  - TD3 might have better performance, but SAC is easy to tune and stable

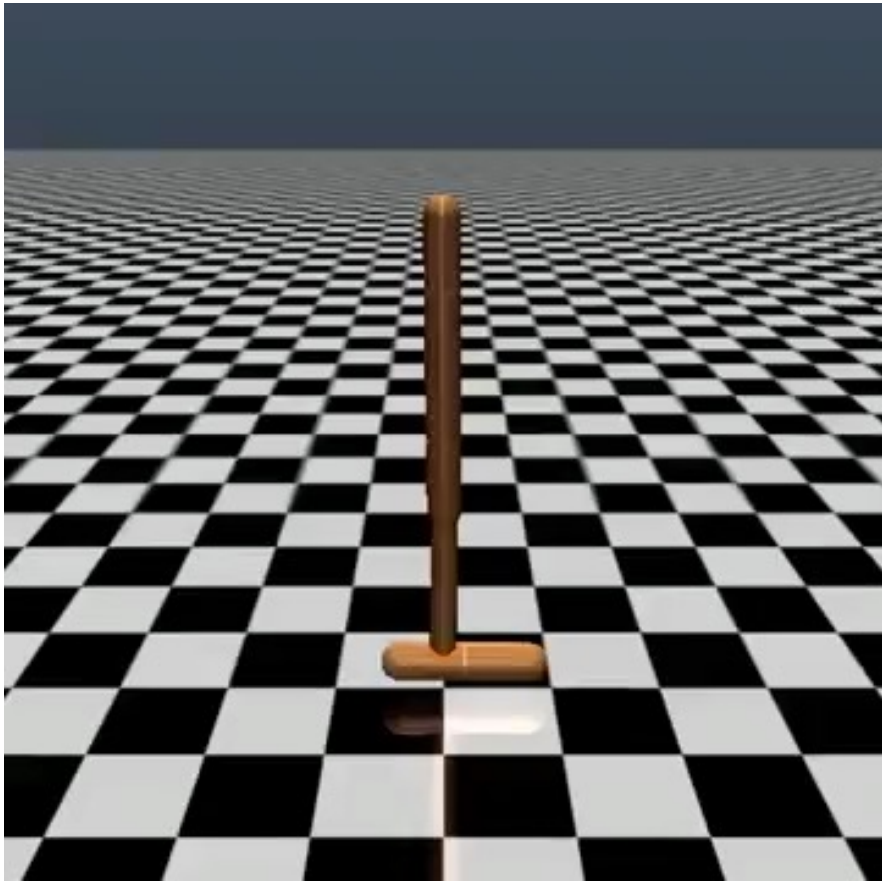


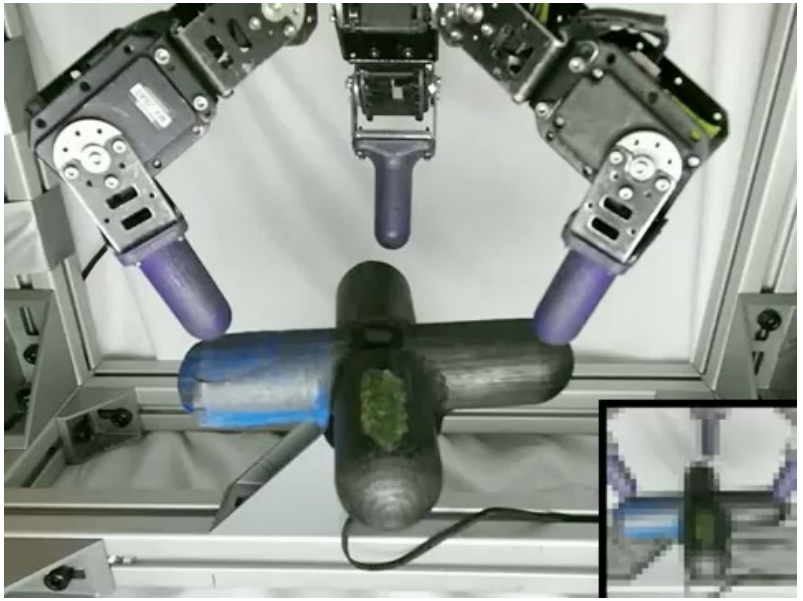
(a) Hopper-v1 (b) Walker2d-v1 (c) HalfCheetah-v1

(d) Ant-v1 (e) Humanoid-v1 (f) Humanoid (rllab)

Soft Actor Critic (SAC)

- Visualizations of Learned Policy





Revisit The Goal of RL

- Let's revisit the objective of RL
  - Obtain an optimal policy that maximizes discounted accumulative reward

$$\pi^* \in \underset{\pi \in \Pi}{\operatorname{argmax}}\, \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right]$$

Decision variable

Objective function

- This is actually an optimization problem
- Can we directly optimize the RL objective to obtain the optimal policy?

Reformulating The RL Problem

- Let's ignore the discount factor for now
- For a parametrized policy, the optimization problem becomes to optimize the policy parameters

$$\theta^* \in \arg\max_\theta \underbrace{E_{\pi_\theta}\left[\sum_t r(s_t, a_t)\right]}_{J(\theta)}$$

- Use gradient ascent: $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$
- How to calculate the gradient?
  - Cannot calculate objective and gradient analytically

Calculating The Policy Gradient

$$\theta^* \in \arg\max_\theta E_{\pi_\theta}\left[\sum_t r(s_t, a_t)\right]$$

What distribution is this?

- π_θ represents the distribution of the trajectories rolled out by π_θ. The RL objective becomes

$$\theta^\star = \arg\max_\theta E_{\tau\sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right]$$

- where

$$\underbrace{p_\theta(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)}_{p_\theta(\tau)} = p(\mathbf{s}_1)\prod_{t=1}^{T}\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

Calculating The Policy Gradient

$$\theta^\star = \arg\max_\theta \underbrace{E_{\tau\sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right]}_{J(\theta)}$$

- The objective can be rewritten as

$$J(\theta) = E_{\tau\sim p_\theta(\tau)}[\underbrace{r(\tau)}_{\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t)}] = \int p_\theta(\tau) r(\tau) d\tau$$

Calculating The Policy Gradient

- Now calculate the gradient of the objective

$$\nabla_\theta J(\theta) = \int \nabla_\theta p_\theta(\tau) r(\tau) d\tau = \int p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) r(\tau) d\tau$$

$$= E_{\tau \sim p_\theta(\tau)} [\nabla_\theta \log p_\theta(\tau) r(\tau)]$$

a convenient identity

$$p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) = p_\theta(\tau) \frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)} = \nabla_\theta p_\theta(\tau)$$

Calculating The Policy Gradient

$$\nabla_\theta J(\theta) = E_{\tau \sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau) r(\tau)]$$

- We need to calculate the log probability of the trajectory:

$$\underbrace{p_\theta(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)}_{p_\theta(\tau)} = p(\mathbf{s}_1) \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

log of both sides

$$\log p_\theta(\tau) = \log p(\mathbf{s}_1) + \sum_{t=1}^{T} \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) + \log p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

- The policy gradient is

$$\nabla_\theta J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) \right) \right]$$

REINFORCE

- Use Monte-Carlo estimation of the policy gradient

$$\nabla_\theta J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) \right) \right]$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$$

- The **REINFORCE** algorithm

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run the policy)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i) \right) \left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i) \right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

How to calculate?

REINFORCE

- Use stochastic policy with Gaussian approximation

$$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t) = \mathcal{N}(f_{\text{neural network}}(\mathbf{s}_t); \Sigma)$$

  - Neural network based mean
  - Usually a small constant variance

- The log probability becomes

$$\log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) = -\frac{1}{2}\|f(\mathbf{s}_t) - \mathbf{a}_t\|^2_\Sigma + \text{const}$$

- Its derivative can then be calculated as

$$\nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) = -\frac{1}{2}\Sigma^{-1}(f(\mathbf{s}_t) - \mathbf{a}_t)\frac{df}{d\theta}$$

Reducing Variance

- As we have already mentioned, the Monte-Carlo estimation has very large variance, which can harm the policy learning

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$$

- One way to reduce variance is to consider the causality
  - Policy at time t' cannot affect reward at time t when $t < t'$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \underbrace{\left(\sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right)}_{\text{“reward to go”}}$$

$\hat{Q}_{i,t}$

Adding Baselines

- But this is not enough, the variance is still high
- To further reduce variance, we can subtract a baseline

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log p_\theta(\tau)[r(\tau) - b]$$

- We have

$$E[\nabla_\theta \log p_\theta(\tau) b] = \int p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) b \, d\tau = \int \nabla_\theta p_\theta(\tau) b \, d\tau$$

$$= b \nabla_\theta \int p_\theta(\tau) d\tau = b \nabla_\theta 1 = 0$$

- Therefore, subtracting a baseline in the policy gradient remains unbiased

Minimizing Variance

- The goal now is to minimizing the variance

$$\mathrm{Var}[x] = E[x^2] - E[x]^2$$

- The random variable now is the policy gradient

$$\nabla_\theta J(\theta) = E_{\tau \sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau)(r(\tau) - b)]$$

- We have

$$\mathrm{Var} = E_{\tau \sim p_\theta(\tau)}[(\nabla_\theta \log p_\theta(\tau)(r(\tau) - b))^2]$$

$$- E_{\tau \sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau)(r(\tau) - b)]^2$$

this bit is just $E_{\tau \sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau) r(\tau)]$

(baselines are unbiased in expectation)

Minimizing Variance

- Now calculate the gradient of the variance over b

$$\begin{aligned}\frac{d\mathrm{Var}}{db} &= \frac{d}{db}E[g(\tau)^2(r(\tau)-b)^2] \\ &= \frac{d}{db}\left(E[g(\tau)^2 r(\tau)^2] - 2E[g(\tau)^2 r(\tau) b] + b^2 E[g(\tau)^2]\right) \\ &= -2E[g(\tau)^2 r(\tau)] + 2bE[g(\tau)^2] = 0\end{aligned}$$

- The optimal baseline is

$$b = \frac{E[g(\tau)^2 r(\tau)]}{E[g(\tau)^2]}$$

From Policy Gradient to Actor-Critic

- The policy gradient

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\left(\sum_{t'=t}^{T} r(\mathbf{s}_{i,t'},\mathbf{a}_{i,t'}) - b\right)$$

$$Q^\pi(\mathbf{s}_t,\mathbf{a}_t) \qquad V^\pi(\mathbf{s}_t)$$

- Define the advantage function

$$A^\pi(\mathbf{s}_t,\mathbf{a}_t) = Q^\pi(\mathbf{s}_t,\mathbf{a}_t) - V^\pi(\mathbf{s}_t)$$

From Policy Gradient to Actor-Critic

- We can further replace $Q^\pi(\mathbf{s}_t, \mathbf{a}_t)$ with TD target:

$$A^\pi(\mathbf{s}_t, \mathbf{a}_t) \approx r(\mathbf{s}_t, \mathbf{a}_t) + V^\pi(\mathbf{s}_{t+1}) - V^\pi(\mathbf{s}_t)$$

- The right hand-side is actually the TD error (without discount for now)
- Actor-critic is biased, but low variance

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) A^\pi(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$$

- Policy gradient is unbiased, but high variance

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \left(\sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) - b \right)$$

Learn The Baseline Function

- Learn $V^{\pi}(\mathbf{s}_t)$ with supervised learning

$$\mathcal{L}(\phi) = \frac{1}{2}\sum_i \left\| \hat{V}_{\phi}^{\pi}(\mathbf{s}_i) - y_i \right\|^2$$

- How to calculate the target?
  - Monte-Carlo:

$$y_{i,t} = \sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'})$$

  - Temporal Difference:

$$y_{i,t} \approx r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \hat{V}_{\phi}^{\pi}(\mathbf{s}_{i,t+1})$$

Reconsider The Discount Factor

- Now add the discount factor back

$$y_{i,t} \approx r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \gamma \hat{V}^{\pi}_{\phi}(\mathbf{s}_{i,t+1})$$

- For **actor-critic**

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \overbrace{\left(r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \gamma \hat{V}^{\pi}_{\phi}(\mathbf{s}_{i,t+1}) - \hat{V}^{\pi}_{\phi}(\mathbf{s}_{i,t}) \right)}^{\hat{A}^{\pi}(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})}$$

- For **policy gradient**

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \left(\sum_{t'=t}^{T} \gamma^{t'-t} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right)$$

Online Actor-Critic Algorithm

- An online actor-critic algorithm

1. take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
2. update $\hat{V}^\pi_\phi$ using target $r + \gamma \hat{V}^\pi_\phi(\mathbf{s}')$
3. evaluate $\hat{A}^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \hat{V}^\pi_\phi(\mathbf{s}') - \hat{V}^\pi_\phi(\mathbf{s})$
4. $\nabla_\theta J(\theta) \approx \nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s}) \hat{A}^\pi(\mathbf{s}, \mathbf{a})$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Direct Policy Improvement

- Previous methods are taking a policy gradient ascent to optimize the policy

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

- Can we directly improve the policy? E.g.:

$$\theta \rightarrow \theta' \qquad\qquad J(\theta') > J(\theta)$$

- We have the **Performance Difference Lemma**

$$J(\theta') - J(\theta) = E_{\tau \sim p_{\theta'}(\tau)} \left[\sum_t \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right]$$

Distribution Mismatch

- **Distribution mismatch**

$$J(\theta') - J(\theta) = E_{\tau \sim p_{\theta'}(\tau)} \left[\sum_t \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right]$$

expectation under $\pi_{\theta'}$ ← $E_{\tau \sim p_{\theta'}(\tau)}$

$A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t)$ → advantage under π_θ

- Change the behavior policy

$$E_{\tau \sim p_{\theta'}(\tau)} \left[\sum_t \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right] = \sum_t E_{\mathbf{s}_t \sim p_{\theta'}(\mathbf{s}_t)} \left[E_{\mathbf{a}_t \sim \pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)} \left[\gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right] \right]$$

$$= \sum_t E_{\mathbf{s}_t \sim p_{\theta'}(\mathbf{s}_t)} \left[E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)} \left[\frac{\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)} \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right] \right]$$

is it OK to use $p_\theta(\mathbf{s}_t)$ instead? ← $E_{\mathbf{s}_t \sim p_{\theta'}(\mathbf{s}_t)}$

Ignore Distribution Mismatch?

- Directly ignore the distribution mismatch?

$$\sum_t E_{\mathbf{s}_t \sim p_{\theta'}(\mathbf{s}_t)} \left[E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)} \left[\frac{\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)} \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right] \right]$$

$$\overset{?}{\approx} \underbrace{\sum_t E_{\mathbf{s}_t \sim p_\theta(\mathbf{s}_t)} \left[E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)} \left[\frac{\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)} \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right] \right]}_{\bar{A}(\theta')}$$

- $\bar{A}(\theta')$ is easier to calculate, and then

$$J(\theta') - J(\theta) \approx \bar{A}(\theta') \quad \Rightarrow \quad \theta' \leftarrow \arg\max_{\theta'} \bar{A}(\theta)$$

Trust Region Policy Optimization (TRPO)

- It can be shown that

$$\theta' \leftarrow \arg\max_{\theta'} \sum_t E_{\mathbf{s}_t \sim p_\theta(\mathbf{s}_t)} \left[E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)} \left[\frac{\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)} \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right] \right]$$

$$\text{such that } D_{\mathrm{KL}}(\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t) \| \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)) \leq \epsilon$$

- For small enough ϵ, this is guaranteed to improve $J(\theta') - J(\theta)$
- This is the **Trust Region Policy Optimization (TRPO)**
- How to solve this constrained optimization?

Trust Region Policy Optimization (TRPO)

- Use Lagrangian dual gradient ascent
  - Maximize $\mathcal{L}(\theta', \lambda)$ with respect to θ'

$$\mathcal{L}(\theta', \lambda) = \sum_t E_{\mathbf{s}_t \sim p_\theta(\mathbf{s}_t)} \left[E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)} \left[\frac{\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)} \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right] \right]$$

$$-\lambda(D_{\mathrm{KL}}(\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t) \| \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)) - \epsilon)$$

  - Update the Lagrange multiplier:

$$\lambda \leftarrow \lambda + \alpha(D_{\mathrm{KL}}(\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t) \| \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)) - \epsilon)$$

Trust Region Policy Optimization (TRPO)

- MuJoCo

Proximal Policy Optimization (PPO)

- TRPO is computationally expensive
  - Two-step optimization
  - calculation for KL divergence
- **Proximal Policy Optimization (PPO)**
  - Pure first order method
  - Maximize a lower bound of original objective function

$$\theta' \leftarrow \arg\max_{\theta'} \mathbb{E}_{s\sim d_{\pi_\theta}, a\sim\pi_\theta} \left\{ \min\left(\rho_{t:t} A^{\pi_\theta}(s,a), \rho_{\text{clip}} A^{\pi_\theta}(s,a) \right) \right\}$$

$$\rho_{\text{clip}} \stackrel{\text{def}}{=} \text{clip}(\rho_{t:t}, 1-\epsilon, 1+\epsilon)$$

Thank you!

清華大學 交叉信息研究院

Institute for Interdisciplinary Information Sciences, Tsinghua University

ISR Lab

Intelligent Systems and Robotics Lab