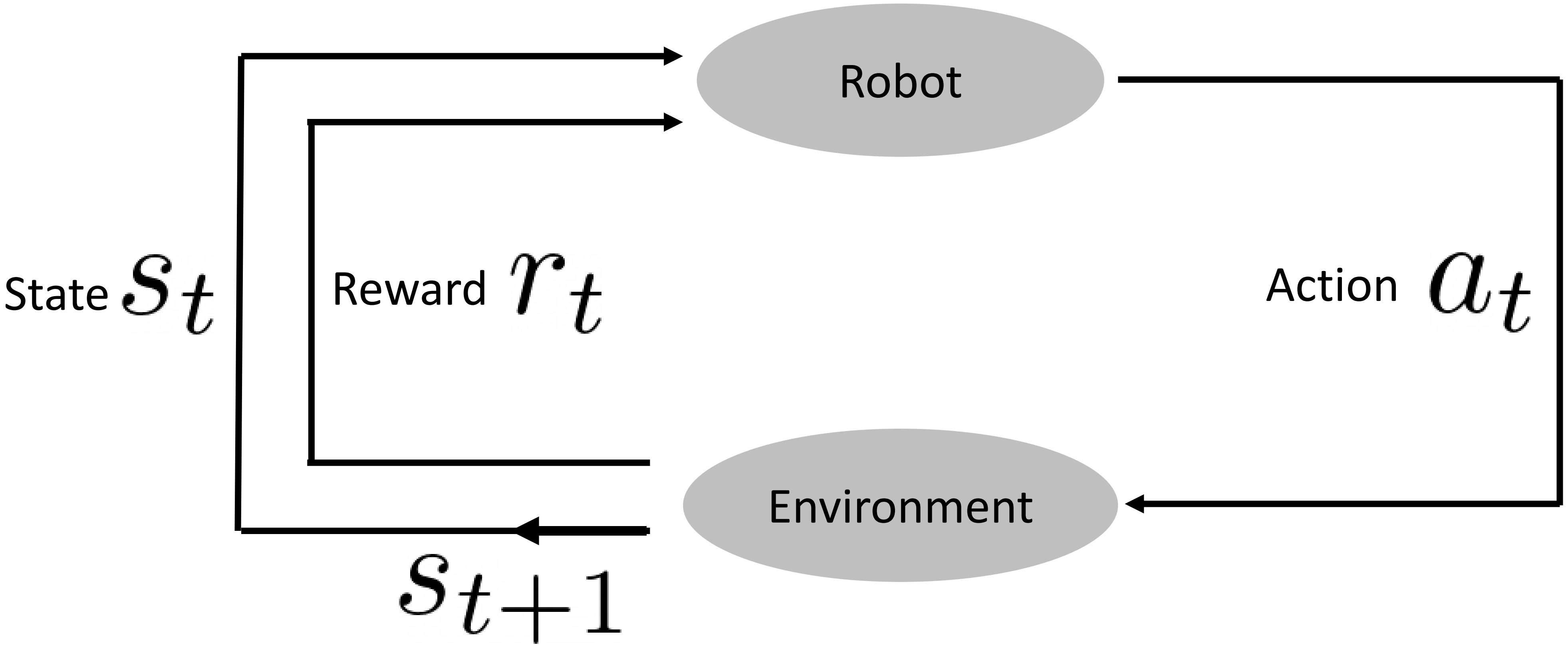
# Reinforcement Learning: Policy Optimization

CS 6341 Robotics

Professor Yu Xiang

The University of Texas at Dallas

# Reinforcement Learning




Reinforcement Learning:

Imitation Learning:

$$a_t = \pi(s_t)$$

# The RL Problem

- The goal of RL is to select a policy which maximizes expected return when the agent acts according to it
- Probability distribution over trajectories

$$P(\tau|\pi) = \rho_0(s_0) \prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t)\pi(a_t|s_t)$$

Transition model (no need in model-free RL)

$p(s' \mid s, a)$

Sample trajectories

- Expected return

$$J(\pi) = \int_{\tau} P(\tau|\pi)R(\tau) = \mathop{\mathrm{E}}_{\tau\sim\pi}[R(\tau)]$$

$R(\tau) = \sum_{t=0}^{T} r_t$ $\quad$ $R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$

$\gamma \in (0, 1)$

- The central optimization problem

$$\pi^* = \arg\max_{\pi} J(\pi)$$

Optimal policy

In practice $\quad \pi_\theta^* = \arg\max_{\theta} J(\pi_\theta)$

Learn the parameters of the policy

# Value Functions

- Value of a state or a state-action pair
  - The expected return if you start in that state or state-action pair, and then act according to a particular policy forever after
- On-policy Value Function $V^{\pi}(s) = \underset{\tau \sim \pi}{\mathrm{E}}\left[R(\tau) \left| s_0 = s\right.\right]$
- On-policy Action-Value Function $Q^{\pi}(s,a) = \underset{\tau \sim \pi}{\mathrm{E}}\left[R(\tau) \left| s_0 = s, a_0 = a\right.\right]$
- Optimal Value Function $V^{*}(s) = \max_{\pi} \underset{\tau \sim \pi}{\mathrm{E}}\left[R(\tau) \left| s_0 = s\right.\right]$
- Optimal Action-Value Function $Q^{*}(s,a) = \max_{\pi} \underset{\tau \sim \pi}{\mathrm{E}}\left[R(\tau) \left| s_0 = s, a_0 = a\right.\right]$

# Value Functions

- Connection

$$V^{\pi}(s) = \mathop{\mathrm{E}}_{a \sim \pi}\left[Q^{\pi}(s,a)\right] \qquad V^*(s) = \max_a Q^*(s,a)$$

- The optimal policy in $s$ will select whichever action maximizes the expected return starting in $s$

$$a^*(s) = \arg\max_a Q^*(s,a)$$

# Parametrized Value Functions

- On-policy Value Function $V^{\pi}(s) = \underset{\tau \sim \pi}{\mathrm{E}}\left[R(\tau) \left| s_0 = s\right.\right]$

- Parameterization (a network) $V_{\phi}(s)$

- Learning the value function
  - Sample trajectories
  - For each trajectory

$$G_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$$

  - Supervised learning $L(\phi) = \frac{1}{N} \sum_{t} \left(V_{\phi}(s_t) - G_t\right)^2$

# Bellman Equations

- The value of your starting point is the reward you expect to get from being there, plus the value of wherever you land next

On-policy

$$V^{\pi}(s) = \mathop{\mathrm{E}}_{\substack{a \sim \pi \\ s' \sim P}} \left[ r(s,a) + \gamma V^{\pi}(s') \right],$$

$$Q^{\pi}(s,a) = \mathop{\mathrm{E}}_{s' \sim P} \left[ r(s,a) + \gamma \mathop{\mathrm{E}}_{a' \sim \pi} \left[ Q^{\pi}(s',a') \right] \right]$$

Optimal policy

$$V^{*}(s) = \max_{a} \mathop{\mathrm{E}}_{s' \sim P} \left[ r(s,a) + \gamma V^{*}(s') \right],$$

$$Q^{*}(s,a) = \mathop{\mathrm{E}}_{s' \sim P} \left[ r(s,a) + \gamma \max_{a'} Q^{*}(s',a') \right]$$

# Advantage Functions

- How much better it is to take a specific action a in state s, over randomly selecting an action according to $\pi(\cdot|s)$

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

Q value for (s, a)

V value for s: random action from the policy

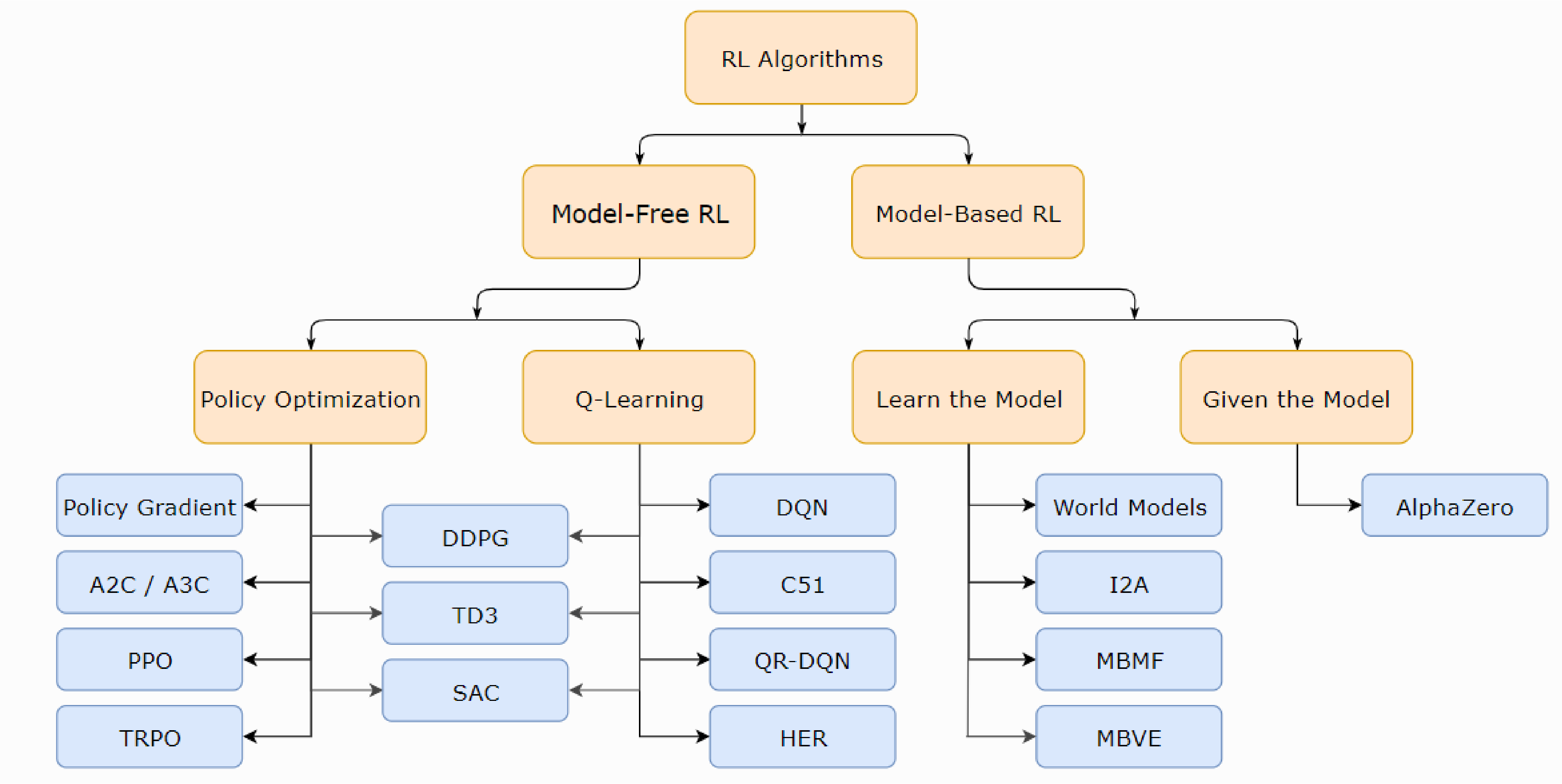
$$Q^{\pi}(s,a) = \mathop{\mathrm{E}}_{\tau \sim \pi}\left[R(\tau)\,|s_0 = s, a_0 = a\right]$$

$$V^{\pi}(s) = \mathop{\mathrm{E}}_{\tau \sim \pi}\left[R(\tau)\,|s_0 = s\right]$$

# Markov Decision Processes (MDPs)

- A MDP is a 5-tuple $\langle S, A, R, P, \rho_0 \rangle$
  - $S$ is the set of all valid states,
  - $A$ is the set of all valid actions,
  - $R : S \times A \times S \to \mathbb{R}$ is the reward function, with $r_t = R(s_t, a_t, s_{t+1})$,
  - $P : S \times A \to \mathcal{P}(S)$ is the transition probability function, with $P(s'|s, a)$ being the probability of transitioning into state $s'$ if you start in state $s$ and take action $a$,
  - and $\rho_0$ is the starting state distribution.

# A Taxonomy of RL Algorithms



# Model-Free vs. Model-based RL

- Whether the agent has access to (or learns) a model of the environment

- A model is a function which predicts state transitions and reward
  - Transition model $p(s' \mid s, a)$
  - Reward model $r(s, a)$

- A model allows the agent to plan by thinking ahead

- A ground-truth model of the environment is usually not available to the agent

# Model-Free RL: Policy Gradient

- Maximize expected return $J(\pi_\theta) = \underset{\tau \sim \pi_\theta}{\mathrm{E}}[R(\tau)]$

$$R(\tau) = \sum_{t=0}^{T} r_t$$

- Gradient ascent $\theta_{k+1} = \theta_k + \alpha \left. \nabla_\theta J(\pi_\theta) \right|_{\theta_k}$

Policy gradient

- How to compute the policy gradient?

$$\begin{aligned} \nabla_\theta J(\pi_\theta) &= \nabla_\theta \underset{\tau \sim \pi_\theta}{\mathrm{E}}[R(\tau)] \\ &= \nabla_\theta \int_\tau P(\tau|\theta) R(\tau) \\ &= \int_\tau \nabla_\theta P(\tau|\theta) R(\tau) \end{aligned}$$

**Probability of a Trajectory**

$$P(\tau|\theta) = \rho_0(s_0) \prod_{t=0}^{T} P(s_{t+1}|s_t, a_t) \pi_\theta(a_t|s_t)$$

# Policy Gradient

- The Log-Derivative Trick $\nabla_\theta P(\tau|\theta) = P(\tau|\theta)\nabla_\theta \log P(\tau|\theta)$

$$\log P(\tau|\theta) = \log \rho_0(s_0) + \sum_{t=0}^{T} \left( \log P(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t) \right)$$

$$\begin{aligned}\nabla_\theta \log P(\tau|\theta) &= \cancel{\nabla_\theta \log \rho_0(s_0)} + \sum_{t=0}^{T} \left( \cancel{\nabla_\theta \log P(s_{t+1}|s_t, a_t)} + \nabla_\theta \log \pi_\theta(a_t|s_t) \right) \\ &= \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t).\end{aligned}$$

No need to know the transition model

# Policy Gradient

$$
\begin{aligned}
\nabla_\theta J(\pi_\theta) &= \nabla_\theta \underset{\tau\sim\pi_\theta}{\mathrm{E}}[R(\tau)] & \\
&= \nabla_\theta \int_\tau P(\tau|\theta)R(\tau) & \text{Expand expectation} \\
&= \int_\tau \nabla_\theta P(\tau|\theta)R(\tau) & \text{Bring gradient under integral} \\
&= \int_\tau P(\tau|\theta)\nabla_\theta \log P(\tau|\theta)R(\tau) & \text{Log-derivative trick} \\
&= \underset{\tau\sim\pi_\theta}{\mathrm{E}}[\nabla_\theta \log P(\tau|\theta)R(\tau)] & \text{Return to expectation form}
\end{aligned}
$$

$$
\nabla_\theta J(\pi_\theta) = \underset{\tau\sim\pi_\theta}{\mathrm{E}}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)R(\tau)\right] \quad \text{Expression for grad-log-prob}
$$

# Policy Gradient

- Collect a set of trajectories using the policy $\pi_\theta$

$$\mathcal{D} = \{\tau_i\}_{i=1,\ldots,N}$$

- Estimate policy gradient

$$\hat{g} = \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau)$$

Categorical policy for discrete actions

$$\log \pi_\theta(a|s) = \log \left[P_\theta(s)\right]_a$$

Diagonal Gaussian policy

$$\log \pi_\theta(a|s) = -\frac{1}{2}\left(\sum_{i=1}^{k}\left(\frac{(a_i - \mu_i)^2}{\sigma_i^2} + 2\log\sigma_i\right) + k\log 2\pi\right)$$

# Policy Gradient

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathrm{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau)\right]$$

$$R(\tau) = \sum_{t=0}^{T} r_t$$

Agents should really only reinforce actions on the basis of their *consequences*.

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathrm{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1})\right]$$

$$\hat{R}_t \doteq \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1})$$

**reward-to-go**

# Vanilla Policy Gradient

- Key idea: push up the probabilities of actions that lead to higher return, and push down probabilities of actions that lead to lower return
- The expected finite-horizon undiscounted return of the policy $J(\pi_\theta)$

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathrm{E}}_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) A^{\pi_\theta}(s_t, a_t)\right]$$

Advantage function $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$

Stochastic gradient ascent $\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_{\theta_k})$

# Vanilla Policy Gradient

**Algorithm 1** Vanilla Policy Gradient Algorithm

1: Input: initial policy parameters $\theta_0$, initial value function parameters $\phi_0$
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4: Compute rewards-to-go $\hat{R}_t$.
5: Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function $V_{\phi_k}$.
6: Estimate policy gradient as

$$\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)|_{\theta_k} \hat{A}_t.$$

7: Compute policy update, either using standard gradient ascent,

$$\theta_{k+1} = \theta_k + \alpha_k \hat{g}_k,$$

or via another gradient ascent algorithm like Adam.
8: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left( V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.
9: **end for**

**Exploration vs. Exploitation**

- stochastic policy

**reward-to-go**

$$\hat{R}_t \doteq \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1})$$

**Advantage function**

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$$

$$= r + V^\pi(s') - V^\pi(s)$$

Bellman Equations

# Trust Region Policy Optimization (TRPO)

- TRPO update

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}(\theta_k, \theta)$$
$$\text{s.t. } \bar{D}_{KL}(\theta || \theta_k) \leq \delta$$

taking the largest step possible to improve performance

- *Surrogate advantage*

$$\mathcal{L}(\theta_k, \theta) = \mathop{\mathrm{E}}_{s,a \sim \pi_{\theta_k}} \left[ \frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a) \right]$$

A measure of how the policy performs related to the old policy

- *KL-divergence*

$$\bar{D}_{KL}(\theta || \theta_k) = \mathop{\mathrm{E}}_{s \sim \pi_{\theta_k}} \left[ D_{KL}\left(\pi_\theta(\cdot|s) || \pi_{\theta_k}(\cdot|s)\right) \right]$$

# Proximal Policy Optimization (PPO)

- PPO-clip updates

$$\theta_{k+1} = \arg\max_{\theta} \underset{s,a\sim\pi_{\theta_k}}{\mathrm{E}} \left[L(s,a,\theta_k,\theta)\right]$$

$$L(s,a,\theta_k,\theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s,a), \;\; \mathrm{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\epsilon, 1+\epsilon\right) A^{\pi_{\theta_k}}(s,a)\right)$$

Avoid stepping so far that we accidentally cause performance collapse

PPO methods are significantly simpler to implement, and empirically seem to perform at least as well as TRPO

- A simpler version

$$L(s,a,\theta_k,\theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s,a), \;\; g(\epsilon, A^{\pi_{\theta_k}}(s,a))\right)$$

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \geq 0 \\ (1-\epsilon)A & A < 0. \end{cases}$$

# Proximal Policy Optimization (PPO)

**Algorithm 1** PPO-Clip

Input: initial policy parameters $\theta_0$, initial value function parameters $\phi_0$

**for** $k = 0, 1, 2, \ldots$ **do**

  Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.

  Compute rewards-to-go $\hat{R}_t$.

  Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function $V_{\phi_k}$.

  Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left( \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t),\ \ g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

  typically via stochastic gradient ascent with Adam.

  Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left( V_\phi(s_t) - \hat{R}_t \right)^2,$$

  typically via some gradient descent algorithm.

**end for**

# PPO Example


https://rewind-reward.github.io/

# Summary

- Model-free RL
  - Vanilla Policy Gradient
  - Trust Region Policy Optimization (TRPO)
  - Proximal Policy Optimization (PPO)

# Further Reading

- OpenAI Spinning Up in Deep RL
  https://spinningup.openai.com/en/latest/index.html