RL - Function Approximations

Outline

1. Introduction
2. Bellman Equations
3. Temporal Difference (TD) Methods
4. Function Approximation for Value Functions
5. Actor-critic Methods
6. Deep Reinforcement Learning

Policy Function Approximation

Policy Gradient Methods

- Policy gradient methods learn a parameterized policy that can select actions without needing to compute a value function
- Policy π is parameterized with $\omega \in \mathbb{R}^n$

$$\pi(a \mid s, \omega) = p(a_t = a \mid s_t = s, \omega_t = \omega)$$

For instance, a Gibbs policy $\pi(a \mid s, \omega) = \dfrac{\exp(\omega^\top \psi(s, a))}{\sum_{a'} \exp(\omega^\top \psi(s, a'))}$ where $\psi(s, a)$ denotes the feature functions.

- Given a performance measure $J(\omega)$, the gradient is

$$\omega_{t+1} = \omega_t + \alpha \nabla_\omega J(\omega_t)$$

- $J(\omega)$ is typically $\displaystyle\sum_s d^\pi(s)\ V^{\pi(\omega)}(s)$.

Compared with Value-based Methods

- Pros: better convergence properties, effective in high-dim or continuous action space, can learn stochastic policies
- Cons: usually converge to local optimum, inefficient to evaluate, high variance

Policy Gradient Theorem

- Reward $J(\omega) = \sum_s d^{\pi_\omega}(s) V^{\pi_\omega}(s) = \sum_s d^{\pi_\omega}(s) \sum_a Q^{\pi_\omega}(s, a) \pi_\omega(a \mid s)$

where d^{π_ω} is the on-policy distribution under π, i.e., the stationary distribution of Markov chain for π_ω: $d^{\pi_\omega}(s) = \lim_{t \to \infty} P(s_t = s \mid s_0, \pi_\omega)$

Theorem $\nabla_\omega J(\omega) \propto \sum_s d^{\pi_\omega}(s) \sum_a Q^{\pi_\omega}(s, a) \nabla_\omega \pi_\omega(a \mid s)$

Provides a nice reformation of the derivative of the objective function to not involve the derivative of the state distribution or the Q

Policy Gradient Theorem: Proof Sketch

(1) A recursive form of derivative of the state value fn.

$$\nabla_w V^\pi(s) = \sum_{a\in A}\left(\nabla_w \pi_w(a|s) Q^\pi(s,a) + \pi_w(a|s)\sum_{s'} P(s'|s,a)\nabla_\theta V^\pi(s')\right)$$

(2) Unroll the recursive representation of $\nabla_w V^\pi(s)$

$$\nabla_w V^\pi(s) = \sum_{x\in S}\sum_{k=0}^{\infty} \rho^\pi(s\to x, k)\,\phi(x)$$

where $\phi(x) = \sum_{a\in A} Q^\pi(x,a)\,\nabla_w \pi_w(a|x)$

(3) A form excluding the derivative of Q-value fn

$$\nabla_w J(w) \propto \sum_s d^\pi(s)\,\phi(s)$$

Policy Gradient Theorem: Proof Details (1)

$$
\begin{aligned}
& \nabla_\theta V^\pi(s) \\
=& \nabla_\theta \Big(\sum_{a \in \mathcal{A}} \pi_\theta(a|s) Q^\pi(s,a) \Big) \\
=& \sum_{a \in \mathcal{A}} \Big(\nabla_\theta \pi_\theta(a|s) Q^\pi(s,a) + \pi_\theta(a|s) \nabla_\theta Q^\pi(s,a) \Big) && \text{; Derivative product rule.} \\
=& \sum_{a \in \mathcal{A}} \Big(\nabla_\theta \pi_\theta(a|s) Q^\pi(s,a) + \pi_\theta(a|s) \nabla_\theta \sum_{s',r} P(s',r|s,a)(r + V^\pi(s')) \Big) && \text{; Extend } Q^\pi \text{ with future state value.} \\
=& \sum_{a \in \mathcal{A}} \Big(\nabla_\theta \pi_\theta(a|s) Q^\pi(s,a) + \pi_\theta(a|s) \sum_{s',r} P(s',r|s,a) \nabla_\theta V^\pi(s') \Big) && P(s',r|s,a) \text{ or } r \text{ is not a func of } \theta \\
=& \sum_{a \in \mathcal{A}} \Big(\nabla_\theta \pi_\theta(a|s) Q^\pi(s,a) + \pi_\theta(a|s) \sum_{s'} P(s'|s,a) \nabla_\theta V^\pi(s') \Big) && \text{; Because } P(s'|s,a) = \textstyle\sum_r P(s',r|s,a)
\end{aligned}
$$

Policy Gradient Theorem: Proof Details (2)

★Def: k-step transition probability

the probability of transitioning from state s with policy π_w after k step: $\rho^{\pi}(s \to x, k)$

$$s \xrightarrow{a \sim \pi_w(\cdot|s)} s' \xrightarrow{a \sim \pi_w(\cdot|s')} s'' \quad \cdots\cdots$$

$$\rho^{\pi}(s \to x, k+1) = \sum_{s'} \rho^{\pi}(s \to s', k)\rho^{\pi}(s' \to x, 1)$$

Policy Gradient Theorem: Proof Details (2)

$$
\begin{aligned}
&\nabla_\theta V^\pi(s) \\
=&\phi(s) + \sum_a \pi_\theta(a|s) \sum_{s'} P(s'|s,a) \nabla_\theta V^\pi(s') \\
=&\phi(s) + \sum_{s'} \sum_a \pi_\theta(a|s) P(s'|s,a) \nabla_\theta V^\pi(s') \\
=&\phi(s) + \sum_{s'} \rho^\pi(s \to s', 1) \nabla_\theta V^\pi(s') \\
=&\phi(s) + \sum_{s'} \rho^\pi(s \to s', 1) \nabla_\theta V^\pi(s') \\
=&\phi(s) + \sum_{s'} \rho^\pi(s \to s', 1) [\phi(s') + \sum_{s''} \rho^\pi(s' \to s'', 1) \nabla_\theta V^\pi(s'')] \\
=&\phi(s) + \sum_{s'} \rho^\pi(s \to s', 1) \phi(s') + \sum_{s''} \rho^\pi(s \to s'', 2) \nabla_\theta V^\pi(s'') \text{ ; Consider } s' \text{ as the middle point for } s \to s'' \\
=&\phi(s) + \sum_{s'} \rho^\pi(s \to s', 1) \phi(s') + \sum_{s''} \rho^\pi(s \to s'', 2) \phi(s'') + \sum_{s'''} \rho^\pi(s \to s''', 3) \nabla_\theta V^\pi(s''') \\
=& \dots \text{; Repeatedly unrolling the part of } \nabla_\theta V^\pi(.) \\
=& \sum_{x \in \mathcal{S}} \sum_{k=0}^{\infty} \rho^\pi(s \to x, k) \phi(x)
\end{aligned}
$$

Policy Gradient Theorem: Proof Details (3)

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta V^\pi(s_0) && \text{; Starting from a random state } s_0 \\
&= \sum_s \sum_{k=0}^{\infty} \rho^\pi(s_0 \to s, k)\phi(s) && \text{; Let } \eta(s) = \textstyle\sum_{k=0}^{\infty} \rho^\pi(s_0 \to s, k) \\
&= \sum_s \eta(s)\phi(s) && \\
&= \left(\sum_s \eta(s)\right) \sum_s \frac{\eta(s)}{\sum_s \eta(s)}\phi(s) && \text{; Normalize } \eta(s), s \in \mathcal{S} \text{ to be a probability distribution.} \\
&\propto \sum_s \frac{\eta(s)}{\sum_s \eta(s)}\phi(s) && \textstyle\sum_s \eta(s) \text{ is a constant} \\
&= \sum_s d^\pi(s) \sum_a \nabla_\theta \pi_\theta(a|s) Q^\pi(s, a) && d^\pi(s) = \tfrac{\eta(s)}{\sum_s \eta(s)} \text{ is stationary distribution.}
\end{aligned}$$

Policy Gradient Theorem: Proof Details (3)

In the episodic case, the constant of proportionality ($\sum_s \eta(s)$) is the average length of an episode; in the continuing case, it is 1 (Sutton & Barto, 2017; Sec. 13.2). The gradient can be further written as:

$$\begin{aligned}
\nabla_\theta J(\theta) &\propto \sum_{s\in\mathcal{S}} d^\pi(s) \sum_{a\in\mathcal{A}} Q^\pi(s,a) \nabla_\theta \pi_\theta(a|s) \\
&= \sum_{s\in\mathcal{S}} d^\pi(s) \sum_{a\in\mathcal{A}} \pi_\theta(a|s) Q^\pi(s,a) \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} \\
&= \mathbb{E}_\pi[Q^\pi(s,a)\nabla_\theta \ln \pi_\theta(a|s)] & \text{; Because } (\ln x)' = 1/x
\end{aligned}$$

Where $\mathbb{E}_\pi$ refers to $\mathbb{E}_{s\sim d_\pi, a\sim\pi_\theta}$ when both state and action distributions follow the policy π_θ (on policy).

REINFORCE

- Policy Gradient Theorem $\nabla_\omega J(\omega) = \sum_s d^\pi(s) \sum_a Q^\pi(s,a) \nabla_\omega \pi(a \mid s, \omega)$ gives us an exact expression for the gradient. We need a sampling method whose expectation equals or approximates this expression.
- We know that

$$\mathbb{E}_{\Pr(s_{t+1}, r_{t+1} \mid s_t), \pi(a_t \mid s_t, \omega), s_t}\left[\gamma^t R_t \frac{\nabla_\omega \pi(a_t \mid s_t, \omega)}{\pi(a_t \mid s_t, \omega)}\right] = \mathbb{E}_{s_t}\left[Q^\pi(s_t, \pi(a_t \mid s_t, \omega)) \frac{\nabla_\omega \pi(a_t \mid s_t, \omega)}{\pi(a_t \mid s_t, \omega)}\right]$$

$$= \sum_s d^\pi(s) \sum_a Q^\pi(s,a) \frac{\nabla_\omega \pi(a \mid s, \omega)}{\pi(a \mid s, \omega)} \pi(a \mid s, \omega) = \nabla_\omega J(\omega)$$

- Therefore, $\nabla_\omega J(\omega)$ can be evaluated through $\mathbb{E}_\pi[\gamma^t R_t \frac{\nabla_\omega \pi(a_t \mid s_t, \omega)}{\pi(a_t \mid s_t, \omega)}]$ ———— REINFORCE

REINFORCE

- Evaluate $\nabla_\omega J(\omega)$ through

$$\mathbb{E}_\pi[\gamma^t R_t \frac{\nabla_\omega \pi(a_t \mid s_t, \omega)}{\pi(a_t \mid s_t, \omega)}] = \mathbb{E}_\pi[\gamma^t R_t \nabla_\omega \log \pi(a_t \mid s_t, \omega)]$$

- Update the policy through

$$\omega_{t+1} = \omega_t + \alpha \gamma^t R_t \nabla_\omega \log \pi(a_t \mid s_t, \omega)$$

- In case π is a Gibbs policy

$$\nabla \log \pi(a_t \mid s_t, \omega) = \psi(s_t, a_t) - \sum_{a'} \pi(a' \mid s_t, \omega) \psi(s_t, a')$$

REINFORCE

Algorithm 3 REINFORCE

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\omega})$, $\alpha > 0$
Initialise $\boldsymbol{\omega}$
repeat
    Generate an episode $s_0, a_0, r_1, \cdots, s_T, a_T$ following $\pi(\cdot|\cdot, \boldsymbol{\omega})$
    for each step $t = 0, \cdots, T$ **do**
        $R_t \leftarrow$ return from step t
        $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + \alpha\gamma^t R_t \nabla \log \pi(a|s_t, \boldsymbol{\omega})$
    end for
until convergence

- An episodic Monte-Carlo Policy-Gradient Method

Off-Policy Policy Gradient

1. The off-policy approach does not require full trajectories and can reuse any past episodes ("experience replay") for much better sample efficiency.
2. The sample collection follows a behavior policy different from the target policy, bringing better exploration.

Now let's see how off-policy policy gradient is computed. The behavior policy for collecting samples is a known policy (predefined just like a hyperparameter), labelled as $\beta(a|s)$. The objective function sums up the reward over the state distribution defined by this behavior policy:

$$J(\theta) = \sum_{s \in \mathcal{S}} d^\beta(s) \sum_{a \in \mathcal{A}} Q^\pi(s, a)\pi_\theta(a|s) = \mathbb{E}_{s \sim d^\beta} \Big[\sum_{a \in \mathcal{A}} Q^\pi(s, a)\pi_\theta(a|s) \Big]$$

where $d^\beta(s)$ is the stationary distribution of the behavior policy β; recall that $d^\beta(s) = \lim_{t \to \infty} P(S_t = s|S_0, \beta)$; and Q^π is the action-value function estimated with regard to the target policy π (not the behavior policy!).

Off-Policy Policy Gradient

Given that the training observations are sampled by $a \sim \beta(a|s)$, we can rewrite the gradient as:

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \mathbb{E}_{s\sim d^\beta}\Big[\sum_{a\in\mathcal{A}} Q^\pi(s,a)\pi_\theta(a|s)\Big] \\
&= \mathbb{E}_{s\sim d^\beta}\Big[\sum_{a\in\mathcal{A}} \big(Q^\pi(s,a)\nabla_\theta\pi_\theta(a|s) + {\color{red}\pi_\theta(a|s)\nabla_\theta Q^\pi(s,a)}\big)\Big] && \text{; Derivative product rule.} \\
&\overset{(i)}{\approx} \mathbb{E}_{s\sim d^\beta}\Big[\sum_{a\in\mathcal{A}} Q^\pi(s,a)\nabla_\theta\pi_\theta(a|s)\Big] && \text{; Ignore the red part: } {\color{red}\pi_\theta(a|s)\nabla_\theta Q^\pi(s,a)}\text{.} \\
&= \mathbb{E}_{s\sim d^\beta}\Big[\sum_{a\in\mathcal{A}} \beta(a|s)\frac{\pi_\theta(a|s)}{\beta(a|s)} Q^\pi(s,a)\frac{\nabla_\theta\pi_\theta(a|s)}{\pi_\theta(a|s)}\Big] \\
&= \mathbb{E}_\beta\Big[{\color{blue}\frac{\pi_\theta(a|s)}{\beta(a|s)}} Q^\pi(s,a)\nabla_\theta \ln\pi_\theta(a|s)\Big] && \text{; The blue part is the importance weight.}
\end{aligned}$$

where $\frac{\pi_\theta(a|s)}{\beta(a|s)}$ is the importance weight.

In summary, when applying policy gradient in the off-policy setting, we can simple adjust it with a weighted sum and the weight is the ratio of the target policy to the behavior policy,

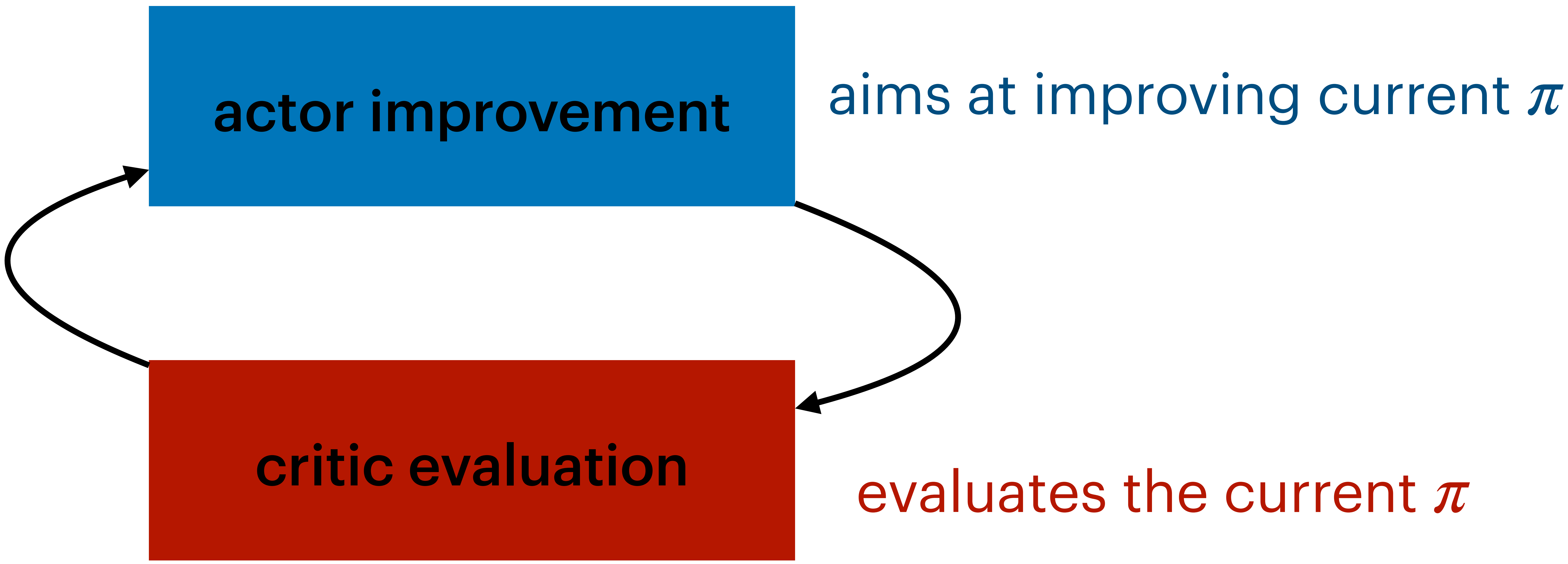
Outline

1. Introduction
2. Bellman Equations
3. Temporal Difference (TD) Methods
4. Function Approximation for Value Functions
5. Actor-critic Methods
6. Deep Reinforcement Learning

RL -Actor Critic & Intro to DRL

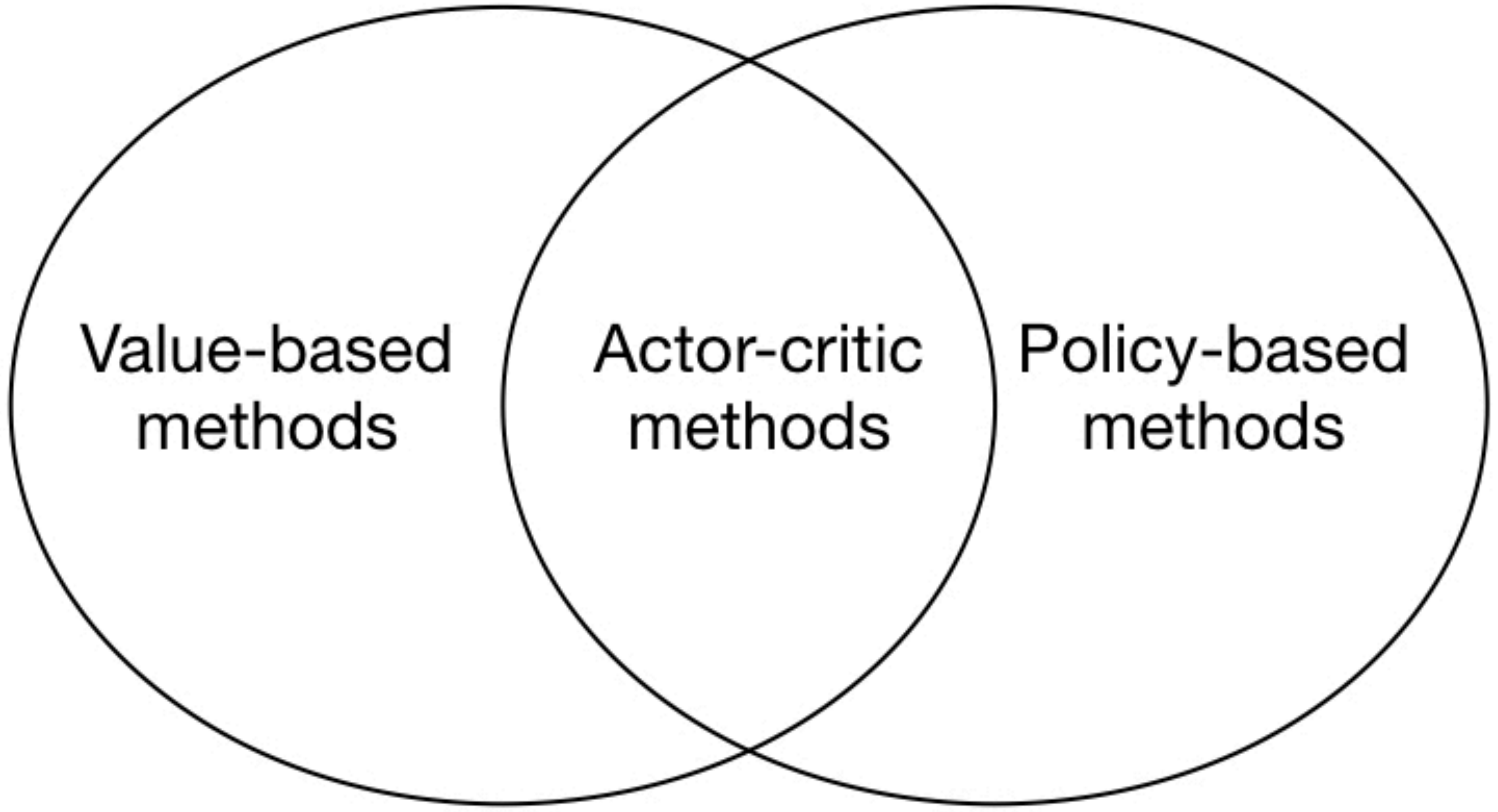
Actor-critic Methods

- A generalized policy iteration, **alternating** between a policy evaluation and a policy improvement step.



Relation to Other RL Methods

- Value-based methods
  - estimate the value function
  - policy is implicit (e.g., ϵ-greedy)
- Policy-based methods
  - estimate the policy
  - no value function
- Actor-critic methods
  - estimate the policy
  - estimate the value function



Implementing a Critic

- The critic estimates the value of the current policy - prediction problem
- Since the actor uses Q values to choose actions, the critic must estimate the Q function.
  - For **small state-spaces**, use tabular TD algorithms to estimate the Q function (SARSA, Q-learning, etc)
  - For **large state-spaces**, use LSTD to estimate the Q function.

Implementing an Actor

- Greedy improvement - move the policy toward the greedy policy underlying the Q function estimate obtained from the critic
  - For small state-action spaces: policy is greedy w.r.t. the obtained Q values
  - For large state-action spaces: policy is parameterized and the greedy action is computed on the fly
- Policy gradient - perform policy gradient directly on the performance surface $J(\omega)$ underlying the chosen parametric policy class

Variance Reduction in Policy gradient

- Instead of using $\nabla_\omega J(\omega) = \mathbb{E}_\pi[\gamma^t R_t \nabla_\omega \log \pi(a_t \mid s_t, \omega)]$ as in REINFORCE, we use $\nabla_\omega J(\omega) = \mathbb{E}_\pi[\gamma^t Q^\pi(s_t, a_t) \nabla_\omega \log \pi(a_t \mid s_t, \omega)]$ since we have estimated Q values in the Critic.

- Further we use an **advantage function** $A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$ instead of $Q^\pi(s_t, a_t)$ since $\mathbb{E}_\pi[\gamma^t V^\pi(s_t) \nabla_\omega \log \pi(a_t \mid s_t, \omega)] = 0$

$$
\begin{aligned}
\mathbb{E}_{\tau\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a_t|s_t) b(s_t)\right] &= \mathbb{E}_{s_{0:t},a_{0:t-1}}\left[\mathbb{E}_{s_{t+1:T},a_{t:T-1}}\left[\nabla_\theta \log \pi_\theta(a_t|s_t) b(s_t)\right]\right] \\
&= \mathbb{E}_{s_{0:t},a_{0:t-1}}\left[b(s_t)\cdot \underbrace{\mathbb{E}_{s_{t+1:T},a_{t:T-1}}\left[\nabla_\theta \log \pi_\theta(a_t|s_t)\right]}_{E}\right] \\
&= \mathbb{E}_{s_{0:t},a_{0:t-1}}\left[b(s_t)\cdot \mathbb{E}_{a_t}\left[\nabla_\theta \log \pi_\theta(a_t|s_t)\right]\right] \\
&= \mathbb{E}_{s_{0:t},a_{0:t-1}}\left[b(s_t)\cdot 0\right] = 0
\end{aligned}
$$

$$
\mathbb{E}_{a_t}\left[\nabla_\theta \log \pi_\theta(a_t|s_t)\right] = \int \frac{\nabla_\theta \pi_\theta(a_t|s_t)}{\pi_\theta(a_t|s_t)} \pi_\theta(a_t|s_t) da_t = \nabla_\theta \int \pi_\theta(a_t|s_t) da_t = \nabla_\theta \cdot 1 = 0
$$

Outline

1. Introduction
2. Bellman Equations
3. Temporal Difference (TD) Methods
4. Function Approximation for Value Functions
5. Actor-critic Methods
6. Deep Reinforcement Learning

Deep Reinforcement Learning

- Deep reinforcement learning refers to using a neural network to approximate the value function, the policy or the model.

- Nonlinear function approximate might be “rich”
- However, it may not give any interpretation or the estimate might be stuck at the local optima due to the non-convexity of the optimization landscape

Value-Based Algorithms

Policy Gradient Algorithms

Model-based Methods

A Brief Summary

Summary

- Neural networks can be used to approximate the value function, the policy or the model in reinforcement learning.
- Any algorithms that assumes a parametric approximation can be applied with neural networks
- However, vanilla versions might not always converge due to biased estimates and correlated samples
- With methods such as prioritised replay, double Q-network or duelling networks the stability can be achieved
- Neural networks can also be applied to actor-critic methods
- Using them for model-based method does not always work well due to compounding errors

Deep RL Methods

- Model-free
  - Value-based: DQN, Double DQN, Rainbow DQN, PER, Retrace, ...
  - Policy-based (usually actor-critic): A2C, A3C, DDPG, TRPO, PPO, ...
- Model-based
  - use dynamics model to simulate
  - use model to initialize model-free learners
  - use model to regularize learned representation