Reinforcement Learning Notes for Finance

Ruming Liu

Stevens Institute of Technology, School of Business

rumingliu1303@gmail.com

First version: May 29, 2026
Last update: June 1, 2026

Contents

1 Overview

In machine learning and optimal control, reinforcement learning (RL) is concerned with how an intelligent agent should take actions in a dynamic environment in order to maximize a reward signal. Reinforcement learning is one of the three basic machine learning paradigms, alongside supervised learning and unsupervised learning.

2 Markov decision process

Basic reinforcement learning is modeled as a Markov decision process $(\mathcal{S}, \mathcal{A}, \gamma, P_a, R_a)$:

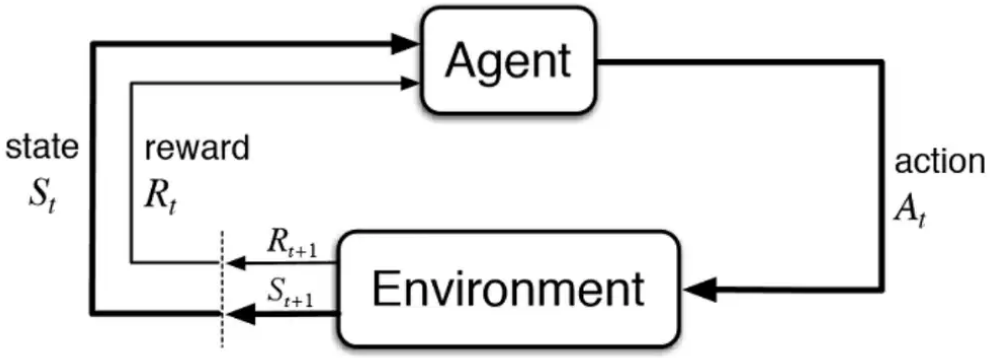
- A set of environment and agent states (the state space), $\mathcal{S}$;
- A set of actions (the action space), $\mathcal{A}$, of the agent;
- γ is the discount factor of reward;
- $P_a(s, s') = \Pr(S_{t+1}{=}s' | S_t{=}s, A_t{=}a)$, the transition probability at time t from state s to state s' under action a;
- $R_a(s, s')$, the immediate reward after transitioning from s to s' under action a.

A basic reinforcement learning agent interacts with its environment in discrete time steps. At each time step t, the agent receives the current state S_t and reward R_t. It then chooses an action A_t from the set of available actions, which is subsequently sent to the environment. The environment moves to a new state S_{t+1} and the reward R_{t+1} associated with the transition (S_t, A_t, S_{t+1}) is determined. The goal of a reinforcement learning agent is to learn a policy:

- $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$
- $\pi(s, a) = P(A_t = a | S_t = s)$

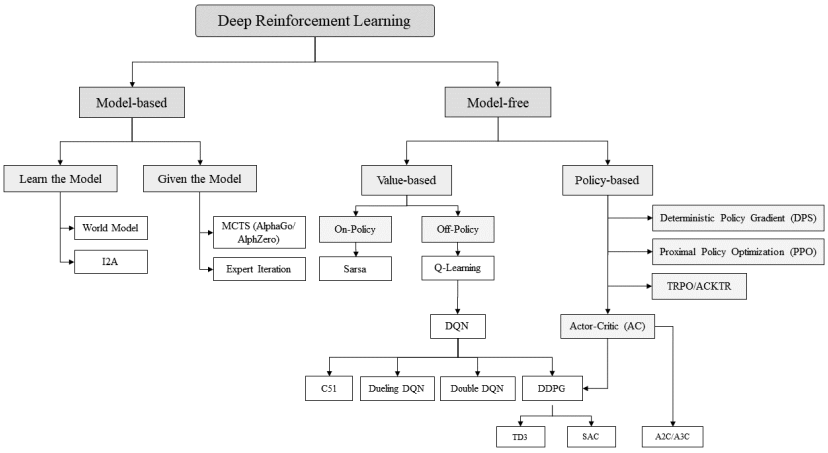
that maximizes the expected cumulative reward. π can be either deterministic or stochastic.[1]

[1] In economics game theory, we also call it pure strategy or mixed strategy.



3 Kinds of RL algorithms

- Model-based vs. model-free reinforcement learning methods.
- Value-based vs. policy-based reinforcement learning methods.
- On-policy vs. off-policy reinforcement learning methods.
- Monte-Carlo vs. temporal-difference update methods.
- Reference here: Young 2023



4 State value function and action value function

- The state value function $V^\pi(s)$ is the expected return when starting in state s and following policy π thereafter:

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_t \mid S_0 = s \right] \tag{1}$$

- The action value function $Q^\pi(s, a)$ is the expected return when starting in state s, taking action a, and thereafter following policy π:

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_t \mid S_0 = s, A_0 = a \right] \tag{2}$$

- The state value function and action value function are related by the following equation:

$$V^\pi(s) = \sum_a \pi(s, a) Q^\pi(s, a) \tag{3}$$

- Equation (1) and (2) are both value functions, if the method is value based, then we will estimate $V^\pi(s)$ or $Q^\pi(s, a)$; if the method is policy-based, then we will directly estimate $\pi(s, a)$. In many cases, the superscript π is omitted for simplicity, but it is important to note that the value functions are always defined with respect to a particular policy π.

5 Value & model based method: Dynamic Programming

5.1 Policy Evaluation (Prediction)

- The model is known as transition probability P_a and reward function R_a which are clearly defined.
- Given a policy π, policy evaluation computes the state value function V^π by iteratively applying the Bellman expectation equation until convergence.

Algorithm 1 Iterative Policy Evaluation, for estimating v^{π}[2]

Input: Policy π to be evaluated; small threshold $\theta > 0$

Initialize $V(s)$ arbitrarily for all $s \in \mathcal{S}$, with $V(\text{terminal}) = 0$
repeat
    $\Delta \leftarrow 0$
    for each $s \in \mathcal{S}$ **do**
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} P_a(s, s')[R_a(s, s') + \gamma V(s')]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
    end for
until $\Delta < \theta$

- But can we derive the optimal policy π^* from the estimated state value function V^{π}? No, because we only evaluate the state value function for one policy π but not the state value function for the optimal policy π^*. We will see in the next section Algorithm (2) that if we estimate the state value function for the optimal policy V^{π^*}, then we can derive the optimal policy π^* by choosing the action that maximizes the expected return for each state.

5.2 Value iteration (Control)

- If the game is infinite, we can solve the Bellman equation iteratively by dynamic programming.[3]
- Assume the optimal policy is π^*, then we have the value function:

$$V^{\pi^*}(s) = \max_a \sum_{s'} P_a(s, s')[R_a(s, s') + \gamma V^{\pi^*}(s')] := \max_a Q^{\pi^*}(s, a) \quad (4)$$

[4]

- Pseudo-code of value iteration of $V^{\pi^*}(s)$[5]:

[2]The algorithm is adapted from Sutton and Barto 2018, p. 75.

[3]If the game is finite, we can solve it directly by backward induction like the extensive form game in game theory.

[4]This denotion is consisitent with (3), but here we use $Q^{\pi^*}(s, a)$ to denote the expected return of taking action a in state s and thereafter following the optimal policy π^*.

[5]Here we use V to denote $V^{\pi^*}(s)$, and π^* is assumed to be the optimal policy but not known a priori.

Algorithm 2 Value Iteration, for estimating π^*[6]

Input: Small threshold $\theta > 0$ determining accuracy of estimation

Initialize $V(s)$ arbitrarily for all $s \in \mathcal{S}^+$, with $V(\text{terminal}) = 0$
repeat
  $\Delta \leftarrow 0$
  for each $s \in \mathcal{S}$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s'} P_a(s, s')[R_a(s, s') + \gamma V(s')]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  end for
until $\Delta < \theta$
Output a deterministic policy π^*:
  $\pi^*(s) = \arg\max_a \sum_{s'} P_a(s, s')[R_a(s, s') + \gamma V(s')]$

- Once we have the optimal value function $V^{\pi^*}(s)$, we can derive the unknown optimal policy π^* by choosing the action that maximizes the expected return for each state:

$$\pi^*(s) = \arg\max_a \sum_{s'} P_a(s, s')[R_a(s, s') + \gamma V^{\pi^*}(s')] = \arg\max_a Q^{\pi^*}(s, a) \tag{5}$$

- Under single-agent setting, the optimal policy is deterministic. However, in multi-agent setting, the optimal policy is usually stochastic.

- Why do we only estimate the state value function $V^{\pi^*}(s)$ but not the action value function $Q^{\pi^*}(s, a)$? Because we can derive $Q^{\pi^*}(s, a)$ from $V^{\pi^*}(s)$ if we know the model, i.e., transition probability P_a and reward function R_a. In the next section 6, we will see that if the model is unknown, then we have to estimate $Q^{\pi^*}(s, a)$ directly without estimating $V^{\pi^*}(s)$.

5.3 Policy iteration (Control)

- In section 5.2, we have introduced the value iteration method, which estimates the optimal state value function $V^{\pi^*}(s)$ and then derives the optimal policy π^* from $V^{\pi^*}(s)$.

- The policy iteration method is an alternative method to estimate the optimal policy π^*. It consists of two steps: policy evaluation and policy

[6]The algorithm is adapted from Sutton and Barto 2018, p. 83.

improvement. In policy evaluation, we compute the state value function V^π for the current policy π. In policy improvement, we update the policy π to be greedy with respect to the current value function V^π. We repeat these two steps until convergence, at which point we find the optimal policy π^*.

- The pseudo-code of policy iteration is as follows:

Algorithm 3 Policy Iteration (using iterative policy evaluation), for estimating π^*[7]

1. Initialization
Initialize $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$; $V(\text{terminal}) = 0$

2. Policy Evaluation
repeat
    $\Delta \leftarrow 0$
    for each $s \in \mathcal{S}$ **do**
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} P_a(s, s')[R_a(s, s') + \gamma V(s')]$[8]
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
    end for
until $\Delta < \theta$

3. Policy Improvement
policy-stable $\leftarrow$ **true**
for each $s \in \mathcal{S}$ **do**
    old-action $\leftarrow \pi(s)$
    $\pi(s) \leftarrow \arg\max_a \sum_{s'} P_a(s, s')[R_a(s, s') + \gamma V(s')]$[9]
    if *old-action* $\neq \pi(s)$ **then**
        policy-stable $\leftarrow$ **false**
    end if
end for
if *policy-stable* **then**
    stop and **return** $V \approx v^*$, $\pi \approx \pi^*$
else
    go to step 2
end if

[7]The algorithm is adapted from Sutton and Barto 2018, p. 80.

[8]I slightly change the notation here, but the meaning is the same as the original writing in the book.

[9]In other words, if there is an action a such that $Q^\pi(s, a) > Q^\pi(s, \pi(s))$. Then the

6 Value & model free based method: Monte Carlo

- We are unknown about the model, i.e., transition probability P_a and reward function R_a are not clearly defined.
- Monte Carlo methods learn directly from episodes of experience without knowledge of the environment's dynamics. The following algorithm estimates the state value function V^π using first-visit MC prediction for a given policy π.

Algorithm 4 First-visit MC prediction[10], for estimating V^π

Input: A policy π to be evaluated
Output: Value function V^π

Initialize $V(s) \in \mathbb{R}$ arbitrarily, for all $s \in \mathcal{S}$
Initialize $Returns(s) \leftarrow$ empty list, for all $s \in \mathcal{S}$
loop (for each episode)
    Generate an episode following π: $S_0, A_0, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    for each step $t = T-1, T-2, \ldots, 0$ **do**
        $G \leftarrow G + R_{t+1}$
        if $S_t \notin \{S_0, S_1, \ldots, S_{t-1}\}$ **then**
            Append G to $Returns(S_t)$
            $V(S_t) \leftarrow$ average$(Returns(S_t))$
        end if
    end for
end loop

- Can we extract the optimal policy π^* from the estimated state value function V^π? No, because we only evaluate the state value function for one policy π but not the state value function for the optimal policy π^*, also we don't know the model, we cannot extract the optimal policy like equation(5) which relies on the transition probabilities, reward function and the optimal state value function. None of these three are available

policy π can be strictly improved by setting $a \leftarrow \pi(s)$. This concept is very similar to the one-shot deviation principle in game theory, which states that a strategy profile is a Nash equilibrium if and only if no player can gain by unilaterally deviating from their strategy.

[10]The algorithm is adapted from Sutton and Barto 2018, p. 92. The first-visit MC method estimates $v^\pi(s)$ as the average of the returns following first visits to s, whereas the every-visit MC method averages the returns following all visits to s.

But this method is suitable for policy evaluation, i.e., estimating V^π for a given policy π.

6.1 MC with Exploring Starts

- Therefore, we have to estimate the action value function $Q^\pi(s, a)$ directly, and then we can derive the optimal policy π^* by choosing the action that maximizes $Q^\pi(s, a)$ for each state.

Algorithm 5 Monte Carlo ES (Exploring Starts)[11], for estimating π^*

Initialize $\pi(s) \in \mathcal{A}(s)$ arbitrarily, for all $s \in \mathcal{S}$
Initialize $Q(s, a) \in \mathbb{R}$ arbitrarily, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
Initialize $Returns(s, a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
loop (for each episode)
    Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly s.t. all pairs have probability > 0[12]
    Generate an episode from S_0, A_0 following π: $S_0, A_0, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    for each step $t = T-1, T-2, \ldots, 0$ **do**
        $G \leftarrow G + R_{t+1}$
        if $(S_t, A_t) \notin \{(S_0, A_0), \ldots, (S_{t-1}, A_{t-1})\}$ **then**
            Append G to $Returns(S_t, A_t)$
            $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
            $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$
        end if
    end for
end loop

- The step $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$ can be replaced by an incremental update, which avoids storing the full list of returns. Let Q_n denote the estimate after n visits to (s, a), and $G_1, \ldots, G_n$ the observed

[11] The algorithm is adapted from Sutton and Barto 2018, p. 99.

[12] The episodes start in a state–action pair, and that every pair has a nonzero probability of being selected as the start. This guarantees that all state–action pairs will be visited an infinite number of times in the limit of an infinite number of episodes. We call this the assumption of exploring starts.

returns. Then:

$$\begin{aligned} Q_n(s,a) &= \frac{1}{n}\sum_{i=1}^{n} G_i \\ &= \frac{1}{n}\left(G_n + \sum_{i=1}^{n-1} G_i\right) \\ &= \frac{1}{n}\left(G_n + (n-1)\,Q_{n-1}(s,a)\right) \\ &= \frac{1}{n}\left(G_n + n\,Q_{n-1}(s,a) - Q_{n-1}(s,a)\right) \\ &= Q_{n-1}(s,a) + \frac{1}{n}[G_n - Q_{n-1}(s,a)] \end{aligned} \tag{6}$$

Setting $n = N(S_t, A_t)$ after incrementing the counter gives the update rule:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)}[G - Q(S_t, A_t)] \tag{7}$$

More generally, replacing $\frac{1}{N(S_t,A_t)}$ with a fixed step size $\alpha \in (0, 1]$ gives a weighted update that places more weight on recent returns, which is better suited for non-stationary environments.

- The Monte Carlo ES method is an on-policy method.

6.2 MC without Exploring Starts

- Why do we need exploring starts? Because we need to guarantee that all state–action pairs will be visited an infinite number of times in the limit of an infinite number of episodes. However, in practice, the game sometimes starts from a fixed state. Monte Carlo Control without Exploring Starts learns an optimal policy by using an exploratory policy (e.g., ϵ-greedy) to ensure all state-action pairs are visited, instead of assuming every episode can start from any state-action pair.

- We first introduce On-policy first-visit MC control, which estimates the optimal policy π^* by using an ϵ-soft policy to generate behavior. By including some randomness in the policy, we ensure that all state–action pairs are visited without the need for exploring starts.

- The Monte Carlo control method without exploring starts is also an on-policy method, because the policy being evaluated and improved is the same as the policy used to generate behavior (both are ϵ-soft).

[13] The algorithm is adapted from Sutton and Barto 2018, p. 101.

Algorithm 6 On-policy first-visit MC control (for ε-soft policies), for estimating π^{*}[13]

Input: Small $\varepsilon > 0$

Initialize $\pi \leftarrow$ an arbitrary ε-soft policy
Initialize $Q(s,a) \in \mathbb{R}$ arbitrarily, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
Initialize $Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
loop (for each episode)
    Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    for each step $t = T-1, T-2, \ldots, 0$ **do**
        $G \leftarrow \gamma G + R_{t+1}$
        if $(S_t, A_t) \notin \{(S_0, A_0), \ldots, (S_{t-1}, A_{t-1})\}$ **then**
            Append G to $Returns(S_t, A_t)$
            $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
            $A^{*} \leftarrow \arg\max_a Q(S_t, a)$ (ties broken arbitrarily)
            for all $a \in \mathcal{A}(S_t)$ **do**
                $$\pi(a \mid S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^{*} \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^{*} \end{cases}$$
            end for
        end if
    end for
end loop

6.3 Off-policy MC Prediction & Control

- All learning control methods face a dilemma: They seek to learn action values conditional on subsequent optimal behavior, but they need to behave non-optimally in order to explore all actions (to find the optimal actions). How can they learn about the optimal policy while behaving according to an exploratory policy? The on-policy approach in the preceding section is actually a compromise—it learns action values not for the optimal policy, but for a near-optimal policy that still explores.

- A more straightforward approach is to use two policies, one that is learned about and that becomes the optimal policy, and one that is more exploratory and is used to generate behavior. The policy being learned about is called the target policy, and the policy used to generate behavior is called the behavior policy. In this case we say that learning is from data "off" the target policy, and the overall process is termed off-policy learning.

Algorithm 7 Off-policy MC prediction (policy evaluation), for estimating Q^{π}[14]

Input: An arbitrary target policy π

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
  $Q(s,a) \in \mathbb{R}$ arbitrarily
  $C(s,a) \leftarrow 0$
loop (for each episode)
  $b \leftarrow$ any policy with coverage of π
  Generate an episode following b: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  $W \leftarrow 1$
  for each step $t = T-1, T-2, \ldots, 0$, while $W \neq 0$ **do**
    $G \leftarrow \gamma G + R_{t+1}$
    $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \dfrac{W}{C(S_t, A_t)}[G - Q(S_t, A_t)]$
    $W \leftarrow W \cdot \dfrac{\pi(A_t \mid S_t)}{b(A_t \mid S_t)}$[15]
  end for
end loop

[14]The algorithm is adapted from Sutton and Barto 2018, p. 110.

[15]The importance-sampling ratio $\prod_{k=t+1}^{T-1} \dfrac{\pi(A_k \mid S_k)}{b(A_k \mid S_k)}$ gives the probability of the target

- After the off-policy prediction method, we now add one more step to derive the optimal policy π^*. The key differences between prediction and control are:
  - π **is fixed in prediction, but updated at every step in control.** In prediction, π is a given target policy and we only estimate Q^π. In control, π is greedily improved after each (S_t, A_t) update — this is an instance of Generalized Policy Iteration (GPI), where evaluation and improvement are interleaved within a single episode.
  - **The importance sampling weight** W **simplifies.** In prediction, $W \leftarrow W \cdot \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$. In control, π is a deterministic greedy policy, so $\pi(A_t|S_t) \in \{0, 1\}$: if $A_t = \pi(S_t)$ then $\pi(A_t|S_t) = 1$ and $W \leftarrow W \cdot \frac{1}{b(A_t|S_t)}$; if $A_t \neq \pi(S_t)$ then $\pi(A_t|S_t) = 0$, making $W = 0$ and all further updates useless — so we exit the inner loop immediately.
  - **The exit condition differs.** Prediction exits when $W = 0$; control exits when $A_t \neq \pi(S_t)$, which is equivalent but expressed directly in terms of the greedy action.

policy generating the episode, divided by the probability of the behavior policy generating the episode. For example, if the target policy has large probability of generating the episode, but the behavior policy has small probability of generating the episode, the update will be weighted more heavily.

Algorithm 8 Off-policy MC control, for estimating π^*[16]

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
  $Q(s,a) \in \mathbb{R}$ arbitrarily
  $C(s,a) \leftarrow 0$
  $\pi(s) \leftarrow \arg\max_a Q(s,a)$ (ties broken consistently)
loop (for each episode)
  $b \leftarrow$ any soft policy
  Generate an episode following b: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  $W \leftarrow 1$
  for each step $t = T-1, T-2, \ldots, 0$ **do**
    $G \leftarrow \gamma G + R_{t+1}$
    $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)}[G - Q(S_t, A_t)]$
    $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$ (ties broken consistently)
    if $A_t \neq \pi(S_t)$ **then**
      exit inner loop (proceed to next episode)
    end if
    $W \leftarrow W \cdot \frac{1}{b(A_t \mid S_t)}$
  end for
end loop

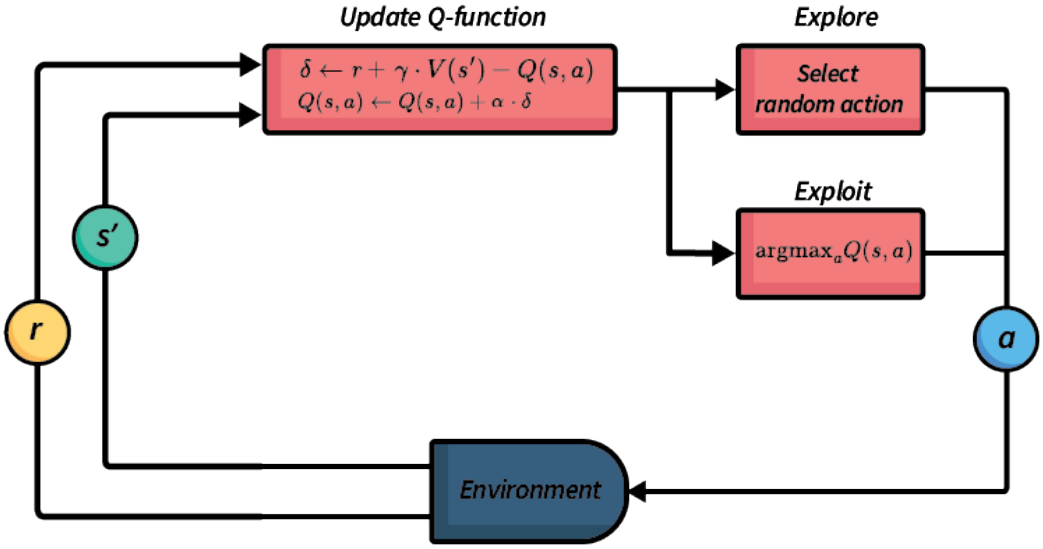
7 Value & model free based method: Temporal-Difference

- Temporal-Difference (TD) learning is a combination of Monte Carlo ideas and dynamic programming (DP) ideas. Like Monte Carlo methods, TD methods can learn directly from raw experience without a model of the environment's dynamics. Like DP methods, TD methods update estimates based in part on other learned estimates, without waiting for the final outcome (they bootstrap).

- TD methods update their estimates based in part on other estimates. They learn a guess from a guess—they bootstrap. Is this a good thing to do? What advantages do TD methods have over Monte Carlo and DP methods?

[16] The algorithm is adapted from Sutton and Barto 2018, p. 111.

- Obviously, TD methods have an advantage over DP methods in that they do not require a model of the environment, of its reward and next-state probability distributions. The next most obvious advantage of TD methods over Monte Carlo methods is that they are naturally implemented in an online, fully incremental fashion. With Monte Carlo methods one must wait until the end of an episode, because only then is the return known, whereas with TD methods one need wait only one time step.



- Monte Carlo methods must wait until the end of the episode to determine the increment to $V(S_t)$ (only then is Gt known), TD methods need to wait only until the next time step. At time $t+1$ they immediately form a target and make a useful update of $V(S_t)$ using the observed reward R_{t+1} and the estimate $V(S_{t+1})$. The simplest TD method makes the update

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{8}$$

- This TD method is called TD(0), or one-step TD. The pseudo-code of TD(0) is as follows:

Algorithm 9 Tabular TD(0), for estimating v^{π}[17]

Input: Policy π to be evaluated; step size $\alpha \in (0, 1]$
Initialize $V(s)$ arbitrarily for all $s \in \mathcal{S}^+$, with $V(\text{terminal}) = 0$
loop (for each episode)
    Initialize S
    for each step of episode **do**
        $A \leftarrow$ action given by π for S
        Take action A, observe R, S'
        $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
        $S \leftarrow S'$
    end for (until S is terminal)
end loop

- As we discussed in Algorithm(4), the prediction is suitable for policy evaluation, i.e., estimating V^{π} for a given policy π. The TD(0) method is also suitable for policy evaluation, but it can be easily extended to control problems. We will discuss two TD control methods: off-policy TD control (Q-learning) and on-policy TD control (SARSA).

7.1 On-policy TD: SARSA

- We now turn to the use of TD prediction methods for the control problem.
- SARSA stands for State-Action-Reward-State-Action, which reflects the fact that the update of $Q(S_t, A_t)$ depends on the quintuple $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$.
- The TD updates the action value function $Q(S_t, A_t)$ as follows:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \quad (9)$$

[17] The algorithm is adapted from Sutton and Barto 2018, p. 120.

Algorithm 10 SARSA (on-policy TD control), for estimating Q^{π^*}[18]

Input: Step size $\alpha \in (0, 1]$; small $\varepsilon > 0$

Initialize $Q(s, a)$ arbitrarily for all $s \in \mathcal{S}^+$, $a \in \mathcal{A}(s)$, with $Q(\text{terminal}, \cdot) = 0$
loop (for each episode)
    Initialize S
    Choose A from S using policy derived from Q (e.g., ε-greedy[19])
    for each step of episode **do**
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ε-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma\, Q(S', A') - Q(S, A)]$
        $S \leftarrow S'$; $A \leftarrow A'$
    end for (until S is terminal)
end loop

- When the ϵ is small, the output of SARSA is an ε-greedy policy with respect to the optimal action-value function Q^{π^*}.
- SARSA is an on-policy method: the same ε-greedy policy is used both to generate behavior (choose A') and to evaluate/improve the policy (update Q).

7.2 On-policy TD(n): n-step SARSA

[20]

- TD(0) is a one-step method, which updates the value estimate based on the observed reward and the estimate of the next state. We can generalize this to n-step TD methods, which update the value estimate based on the observed rewards and the estimate of the state n steps into the future. The n-step return is defined as:

$$G^{(n)}_{t:t+n} := R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n}) \quad (10)$$

  The n-step TD update is then:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[G^{(n)}_{t:t+n} - Q(S_t, A_t)] \quad (11)$$

[18] The algorithm is adapted from Sutton and Barto 2018, p. 130.

[19] The policy is ε-greedy with respect to Q, meaning that for each state s, the action a that maximizes $Q(s, a)$ is selected with probability $1-\varepsilon+\varepsilon/|\mathcal{A}(s)|$, and all other actions are selected with probability $\varepsilon/|\mathcal{A}(s)|$. $\pi(a|s) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(s)| & \text{if } a = \arg\max_{a'} Q(s, a') \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq \arg\max_{a'} Q(s, a') \end{cases}$

[20] And the off-policy TD(n) SARSA can be found in Sutton and Barto 2018, p. 149.

- The pseudo-code of n-step SARSA is presented in Sutton and Barto 2018, p. 147.

7.3 Off-policy TD: Q-learning

- As contrasted with SARSA, Q-learning is off-policy and bootstraps toward the greedy maximum action value $\max_{a'} Q(S', a')$ regardless of the policy used to generate the data.
- Q-learning updates the action value function $Q(S_t, A_t)$ as follows:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t)] \quad (12)$$

- Pseudo-code of Q-learning is as follows:

Algorithm 11 Q-learning (off-policy TD control), for estimating π^*[21]

Input: Step size $\alpha \in (0, 1]$; small $\varepsilon > 0$

Initialize $Q(s, a)$ arbitrarily for all $s \in \mathcal{S}^+$, $a \in \mathcal{A}(s)$, with $Q(\text{terminal}, \cdot) = 0$
loop (for each episode)
    Initialize S
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    for each step of episode **do**
        Take action A, observe R, S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_{a'} Q(S', a') - Q(S, A)]$
        $S \leftarrow S'$; $A \leftarrow$ action from S using policy derived from Q (e.g., ε-greedy)
    end for (until S is terminal)
end loop

- The policy extraction of Q-learning is straightforward: the optimal policy π^* is the deterministic greedy policy with respect to the optimal action-value function Q^{π^*}:

$$\pi^*(s) = \arg\max_a Q^{\pi^*}(s, a) \quad (13)$$

- Q-learning is an off-policy method: the policy used to generate behavior (e.g., ε-greedy) is different from the policy being evaluated and improved (the greedy policy with respect to Q).

[21] The algorithm is adapted from Sutton and Barto 2018, p. 133.

8 Intro of policy-based method

- As noted earlier, learning a policy directly has advantages, particularly for applications where the state space or the action space are massive or infinite. If the action space is infinite, then we cannot do policy extraction because that requires us to iterate over all actions and extract the one that maximises the reward. Learning the policy directly mitigates this.

9 Model free, policy based, & policy gradient method: MC

- As noted earlier in section 5.3, policy-based methods search for a policy directly, rather than searching for a value function and extracting a policy in sectuin 5.2. In this section, we look at a model-free method that optimises a policy directly. It is similar to Q-learning and SARSA, but instead of updating a action-state Q-function, it updates the parameters θ of a policy directly using gradient ascent.

- The goal of a policy gradient is to approximate the optimal policy $\pi_\theta(s, a)$ (e.g. normal distribution with parameters $\theta = \{\mu, \sigma\}$) via gradient ascent on the expected return. Gradient ascent will find the best parameters θ for the particular MDP.

- This section is heavily based on Levine 2023, Hui 2018b, and Achiam 2018a, which provide a more detailed introduction to policy gradient methods. Without loss of generality, we set discount factor $\gamma = 1$ in the following discussion.

- Our target function is the cumulative reward under policy π_θ:

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}[\sum_t r(s_t, a_t)] \tag{14}$$

  where $\tau = (s_1, a_1, \ldots, s_T, a_T)$ is a trajectory of states and actions generated by following the policy π_θ, and $p_\theta(\tau)$ is the probability of that trajectory under the policy π_θ:

$$p_\theta(\tau) = p_\theta(s_1, a_1, \ldots, s_T, a_T) = p(s_1) \prod_{t=1}^{T} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t) \tag{15}$$

9.1 Policy Evaluation (Prediction)

- Given a policy π_θ, we want to evaluate the target function $J(\theta)$, which is the expected return under the policy π_θ. We can estimate $J(\theta)$ by sampling trajectories from the policy and computing the average return:

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}[\sum_t r(s_t, a_t)] \simeq \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} r(s_t^i, a_t^i) \tag{16}$$

  where N is the number of sampled trajectories, and (S_t^i, a_t^i) is the state-action pair at time t in the i-th trajectory.

9.2 Policy Gradient & MC: REINFORCE

- Previous section is about policy evaluation, which estimates the expected return under a given policy. Now we want to optimize the policy parameters θ to maximize $J(\theta)$. We can do this by computing the gradient of $J(\theta)$ with respect to θ and performing gradient ascent:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}[(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t))(\sum_{t=1}^{T} r(s_t, a_t))] \tag{17}$$

- Proof of the equation (17):

  Setup. Let $\tau = (s_1, a_1, \ldots, s_T, a_T)$ denote a trajectory. The objective is the expected total reward under π_θ:

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}[r(\tau)] = \int p_\theta(\tau)\, r(\tau)\, d\tau, \quad r(\tau) = \sum_{t=1}^{T} r(s_t, a_t). \tag{18}$$

  The trajectory probability factorises as equation (15).

  Step 1: Differentiate the objective.

$$\nabla_\theta J(\theta) = \nabla_\theta \int p_\theta(\tau)\, r(\tau)\, d\tau = \int \nabla_\theta p_\theta(\tau)\, r(\tau)\, d\tau. \tag{19}$$

  Step 2: Log-derivative trick. Using the identity $p_\theta(\tau)\, \nabla_\theta \log p_\theta(\tau) = \nabla_\theta p_\theta(\tau)$, we substitute:

$$\nabla_\theta J(\theta) = \int p_\theta(\tau)\, \nabla_\theta \log p_\theta(\tau)\, r(\tau)\, d\tau = \mathbb{E}_{\tau \sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau)\, r(\tau)]\,. \tag{20}$$

Step 3: Expand $\log p_\theta(\tau)$ by equation (15).

$$\log p_\theta(\tau) = \log p(s_1) + \sum_{t=1}^{T} \log \pi_\theta(a_t \mid s_t) + \sum_{t=1}^{T} \log p(s_{t+1} \mid s_t, a_t). \quad (21)$$

Step 4: Isolate θ-dependent terms. Since $\log p(s_1)$ and $\log p(s_{t+1} \mid s_t, a_t)$ do not depend on θ, their gradients vanish:

$$\nabla_\theta \log p_\theta(\tau) = \nabla_\theta \left[\cancel{\log p(s_1)} + \sum_{t=1}^{T} \log \pi_\theta(a_t \mid s_t) + \cancel{\sum_{t=1}^{T} \log p(s_{t+1} \mid s_t, a_t)} \right] \quad (22)$$

$$= \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t). \quad (23)$$

Step 5: Final estimator. Substituting back:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \right) \left(\sum_{t=1}^{T} r(s_t, a_t) \right) \right]. \qquad \square$$

- If we consider the cause of the reward, agents should really only reinforce actions on the basis of their consequences. Rewards obtained before taking an action have no bearing on how good that action was: only rewards that come after. It turns out that this intuition shows up in the math, and we can show that the equation (17) can also be expressed by

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \left(\sum_{t'=t}^{T} r(s_{t'}, a_{t'}) \right) \right] \quad (24)$$

[22] Achiam 2018a call this form the "reward-to-go policy gradient," because the sum of rewards after a point in a trajectory, and it can help to reduce the variance of estimator.

- Proof of the equation (24):

$$
\begin{aligned}
&\mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t \mid s_t)\right)\left(\sum_{t=1}^{T} r(s_t,a_t)\right)\right] \\
=&\mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t \mid s_t)\left(\sum_{t'=1}^{T} r(s_{t'},a_{t'})\right)\right] \\
=&\mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t \mid s_t)\left(\sum_{t'=t}^{T} r(s_{t'},a_{t'})+\sum_{t'=1}^{t-1} r(s_{t'},a_{t'})\right)\right] \\
=&\mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t \mid s_t)\left(\sum_{t'=t}^{T} r(s_{t'},a_{t'})\right)\right]^{23}+\underbrace{0}_{\dagger}
\end{aligned}
$$

  † Let $h_t = (s_1, a_1, \ldots, s_t)$ denote the history up to but excluding a_t. By the law of iterated expectations:

$$
\mathbb{E}_\tau\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\cdot\sum_{t'=1}^{t-1} r_{t'}\right]=\mathbb{E}_{h_t}\left[\sum_{t'=1}^{t-1} r_{t'}\cdot\underbrace{\mathbb{E}_{a_t\sim\pi_\theta(\cdot|s_t)}[\nabla_\theta \log \pi_\theta(a_t \mid s_t)]}_{=\nabla_\theta\int \pi_\theta(a_t|s_t)\,da_t=\nabla_\theta 1=0^{24}}\right]=0. \tag{25}
$$

  The past rewards $\sum_{t'<t} r_{t'}$ are h_t-measurable and factor out of the inner expectation. □

- The pseudo-code of the REINFORCE algorithm:

Algorithm 12 REINFORCE: Monte-Carlo, for estimating π^*

Input: Step size $\alpha > 0$; differentiable policy $\pi_\theta(a_t \mid s_t)$

1: Initialize policy parameters θ arbitrarily
2: **loop** (for each episode)
3: Sample a trajectory $\tau^i = \{s_1^i, a_1^i, \ldots, s_T^i, a_T^i\}$ from $\pi_\theta(a_t \mid s_t)$
4: Estimate the policy gradient:

$$
\nabla_\theta J(\theta) \;\approx\; \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t^i \mid s_t^i)\left(\sum_{t\prime=t}^{T} r(s_{t\prime}^i, a_{t\prime}^i)\right)
$$

5: $\theta \leftarrow \theta + \alpha\,\nabla_\theta J(\theta)$
6: **end loop**

[23] Watch out the parentheses here very carefully!

[24] This is also called the Expected Grad-Log-Prob (EGLP) Lemma. For a full proof, see OpenAI Spinning Up: Extra Policy Gradient Proof.

- Sutton and Barto 2018 present a variant of REINFORCE that performs a separate θ update for each time step t within a single episode. Under our assumption $\gamma = 1$, the γ^t factor in Sutton's update reduces to 1, so the total parameter change accumulated over one episode is identical to the batch version with $N = 1$:

$$\Delta\theta = \alpha \sum_{t=0}^{T-1} G_t \, \nabla_\theta \log \pi(A_t \mid S_t, \theta).$$

  The only practical difference is that Sutton's version applies each per-step update sequentially (so later steps see a slightly updated θ), whereas the batch version accumulates all gradients first and then applies one update. For small α the two are essentially equivalent.[25]

Algorithm 13 REINFORCE: Monte-Carlo Policy-Gradient Control (episodic), for estimating π^*[26]

Input: A differentiable policy parameterization $\pi(a \mid s, \theta)$
Input: Step size $\alpha > 0$
Initialize policy parameter $\theta \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)
loop (for each episode)
    Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot \mid \cdot, \theta)$
    for each step $t = 0, 1, \ldots, T-1$ **do**
        $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
        $\theta \leftarrow \theta + \alpha \, \gamma^t G \, \nabla_\theta \ln \pi(A_t \mid S_t, \theta)$
    end for
end loop

9.3 Policy Gradient & MC: REINFORCE with baseline

- Still, the MC introduce high variance in the policy gradient estimator, which can lead to slow learning. Except the "reward to go" strategy in equation (24), another common technique to reduce the variance is to introduce a baseline function $b(s_t)$, which does not change the expected value of the estimator but can reduce its variance:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \left(\sum_{t=1}^{T} r(s_t, a_t) - b(s_t) \right) \right] \quad (26)$$

[25] Some discussions: What is the difference between Sutton's and Levine's REINFORCE algorithm?

[26] Adapted from Sutton and Barto 2018, p. 328.

- How can we guarantee subtracting a baseline is unbiased in expectation:

$$\mathbb{E}_{\tau\sim p_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau)\, b(s_t)] = 0. \tag{27}$$

  The proof is very similar to the proof of the equation (25) by the law of iterated expectations:

$$\begin{aligned}
&\mathbb{E}_{\tau\sim p_\theta(\tau)}[\nabla_\theta \log \pi_\theta(a_t \mid s_t) b(s_t)] \\
=&\mathbb{E}_{h_t}\left[b(s_t)\cdot \underbrace{\mathbb{E}_{a_t\sim\pi_\theta(\cdot|s_t)}[\nabla_\theta \log \pi_\theta(a_t \mid s_t)]}_{=0}\right] = 0.
\end{aligned}$$

  where $h_t = (s_1, a_1, \ldots, s_t)$ is the history up to but excluding a_t. □

- The derivation of the reward-to-go estimator in equation (24) can be viewed as using the past reward $b_t = \sum_{t'=1}^{t-1} r_{t'}$ as a baseline: since b_t is h_t-measurable (i.e., a function of $(s_1, a_1, \ldots, s_{t-1}, a_{t-1}, s_t)$, which does not depend on a_t), subtracting it is unbiased by the same EGLP argument. More generally, any h_t-measurable function is a valid baseline, which is a strictly weaker condition than the standard $b(s_t)$.

- The MC estimated target function including the baseline is:

$$\nabla_\theta J(\theta) \;\approx\; \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^i \mid s_t^i)\left(\sum_{t=1}^{T} r(s_t^i, a_t^i) - b(s_t^i)\right) \tag{28}$$

- The update of θ in REINFORCE with baseline is following equation (28). Or if we want to perform a separate θ update for each time step t within a single episode like Sutton and Barto 2018, the update is suggested as:

$$\theta \leftarrow \theta + \alpha\, \gamma^t (G - b(s_t))\, \nabla_\theta \ln \pi(a_t \mid s_t, \theta) \tag{29}$$

- The pseudo-code of REINFORCE with baseline is as follows:

Algorithm 14 REINFORCE with Baseline (episodic), for estimating π^*[27]

Input: Differentiable policy $\pi(a \mid s, \theta)$; differentiable baseline $\hat{v}(s, \mathbf{w})$[28]
Input: Step sizes $\alpha^\theta > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\theta \in \mathbb{R}^d$ and baseline weights $\mathbf{w} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)
loop (for each episode)
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot \mid \cdot, \theta)$
  for each step $t = 0, 1, \ldots, T-1$ **do**
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
    $\delta \leftarrow G - \hat{v}(S_t, \mathbf{w})$[29]
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \gamma^t \delta \nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$
    $\theta \leftarrow \theta + \alpha^\theta \gamma^t \delta \nabla_\theta \ln \pi(A_t \mid S_t, \theta)$
  end for
end loop

Remark.

Why do we update w in this way? The baseline $\hat{v}(s, \mathbf{w})$ is trained to approximate the expected return G from state s. Since G is a Monte-Carlo return that does not depend on $\mathbf{w}$, the update is a gradient descent step on the squared error $\frac{1}{2}(G - \hat{v}(S_t, \mathbf{w}))^2$:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha^{\mathbf{w}} \gamma^t \nabla_{\mathbf{w}} \frac{\delta^2}{2} = \mathbf{w} + \alpha^{\mathbf{w}} \gamma^t \delta \nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w}).$$

This is the same objective as the value function fitting step in Algorithm 15 (VPG), equation (35): both minimize the squared error between $\hat{v}$ and the MC return. The difference is purely implementational — Algorithm 14 applies online stochastic gradient steps per time step, whereas Algorithm 15 performs a batch regression over all collected trajectories $\mathcal{D}_k$.

9.4 Policy Gradient, MC, & REINFORCE with baseline: Vanilla Policy Gradient

- We can rewrite equation (24). Define $\tau_{:t} = (s_1, a_1, \ldots, s_t, a_t)$ as the trajectory up to time t, and $\tau_{t:}$ as the remainder of the trajectory after

[27] Adapted from Sutton and Barto 2018, p. 330.
[28] We learn two sets of parameters: θ for the policy and $\mathbf{w}$ for the baseline.
[29] You may notic that the reward to go and baseline techniques are combined in this algorithm. The reward to go is used to compute G, and the baseline is used to compute δ.

that. By the law of iterated expectations:

$$\begin{aligned}
\nabla_\theta J(\theta) =& \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t \mid s_t)\left(\sum_{t'=t}^{T} r(s_{t'}, a_{t'})\right)\right] \\
=& \sum_{t=1}^{T}\mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\left(\sum_{t'=t}^{T} r(s_{t'}, a_{t'})\right)\right] \\
=& \sum_{t=1}^{T}\mathbb{E}_{\tau_{:t}\sim p_\theta}\left[\mathbb{E}_{\tau_{t:}\sim p_\theta}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\left(\sum_{t'=t}^{T} r(s_{t'}, a_{t'})\right)\middle|\, \tau_{:t}\right]\right] \\
=& \sum_{t=1}^{T}\mathbb{E}_{\tau_{:t}\sim p_\theta}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\ \underbrace{\mathbb{E}_{\tau_{t:}\sim p_\theta}\left[\sum_{t'=t}^{T} r(s_{t'}, a_{t'})\middle|\, \tau_{:t}\right]}_{=Q^{\pi_\theta}(s_t, a_t)}\right] \\
=& \mathbb{E}_{\tau\sim p_\theta}\left[\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t \mid s_t)\, Q^{\pi_\theta}(s_t, a_t)\right] \qquad (30)
\end{aligned}$$

where the fourth line uses the fact that $\nabla_\theta \log \pi_\theta(a_t \mid s_t)$ is $\tau_{:t}$-measurable (it depends only on s_t, a_t), so it factors out of the inner expectation; and due to Markov property, $\mathbb{E}_{\tau_{t:}\sim p_\theta}[\sum_{t'=t}^{T} r(s_{t'}, a_{t'}) \mid \tau_{:t}] = \mathbb{E}_{\tau_{t:}\sim p_\theta}[\sum_{t'=t}^{T} r(s_{t'}, a_{t'}) \mid s_t, a_t] = Q^{\pi_\theta}(s_t, a_t)$ by definition of the action-value function in equation (2).

- In section 9.3, we introduce a baseline $b(s_t)$ to reduce the variance of the policy gradient estimator. Recall from equation (25) that any h_t-measurable function is a valid (unbiased) baseline, where $h_t = (s_1, a_1, \ldots, s_t)$ denotes the history up to but excluding a_t. A natural choice is the state value function $V^\pi(s_t)$.

  $V^\pi(s_t)$ is h_t-measurable because, by definition (1), it is a function of s_t alone and does not depend on a_t or future states. To see this explicitly, note that a_t is integrated out:

$$\begin{aligned}
V^\pi(s_t) &= \mathbb{E}_{\tau\sim p_\theta}\left[\sum_{t'=t}^{T} r(s_{t'}, a_{t'})\middle|\, s_t\right] \\
&= \mathbb{E}_{a_t\sim\pi_\theta(\cdot|s_t)}\left[\mathbb{E}_{\tau_{t:}\sim p_\theta}\left[\sum_{t'=t}^{T} r(s_{t'}, a_{t'})\middle|\, s_t, a_t\right]\right] \\
&= \mathbb{E}_{a_t\sim\pi_\theta(\cdot|s_t)}[Q^{\pi_\theta}(s_t, a_t)]
\end{aligned}$$

  The last line shows that a_t has been marginalised over by the expectation, so $V^\pi(s_t)$ depends only on $s_t \in h_t$, confirming h_t-measurability and hence its validity as a baseline.

- We define the advantage function:

$$A^{\pi}(s_t, a_t) = Q^{\pi}(s_t, a_t) - V^{\pi}(s_t), \tag{31}$$

  which measures how much better action a_t is compared to the average action at state s_t.

- Considering reward-to-go and substituting baseline $b(s_t) = V^{\pi}(s_t)$ into equation (26), we have:

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\tau \sim p_{\theta}} \left[\sum_{t=1}^{T} \nabla_{\theta} \log \pi_{\theta}(a_t \mid s_t) A^{\pi}(s_t, a_t) \right] \tag{32}$$

- We can rewrite the policy gradient estimator in equation (28) after considering reward-to-go and baseline $V^{\pi}(s)$ as:

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_{\theta} \log \pi_{\theta}(a_t^i \mid s_t^i) \left(Q^{\pi}(s_t^i, a_t^i) - V^{\pi}(s_t^i) \right) \tag{33}$$

- In practice, $V^{\pi}(s_t)$ cannot be computed exactly, so it has to be approximated. This is usually done with a neural network, $V_{\phi}(s_t)$, which is updated concurrently with the policy (so that the value network always approximates the value function of the most recent policy).

- The simplest method for learning $V_{\phi}(s_t)$, used in most implementations of policy optimization algorithms (including VPG, TRPO, PPO, and A2C), is to minimize a mean-squared-error objective:

$$\phi_k = \arg\min_{\phi} \mathbb{E}_{\tau_k \sim p_{\theta_k}} \left[\left(V_{\phi_{k-1}}(s_t) - \sum_{t'=t}^{T} r(s_{t'}, a_{t'}) \right)^2 \right], \tag{34}$$

  or in MC, we can write as:

$$\phi_k = \arg\min_{\phi} \frac{1}{N \times T} \sum_{i=1}^{N} \sum_{t=1}^{T} \left(V_{\phi_{k-1}}(s_t^{i,k}) - \sum_{t'=t}^{T} r(s_{t'}^{i,k}, a_{t'}^{i,k}) \right)^2, \tag{35}$$

  where $\tau_k^i = (s_1^{i,k}, a_1^{i,k}, \ldots, s_T^{i,k})$ is the i-th trajectory generated by the current policy π_{θ_k}, and ϕ_k is the updated value network parameters after fitting all the trajectories.

- The pseudo-code of the vanilla policy gradient algorithm is as follows:

Algorithm 15 Vanilla Policy Gradient Algorithm, for estimating π^*[30]

Input: Initial policy parameters θ_0, initial value function parameters ϕ_0

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

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

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

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

5: Estimate policy gradient as:

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

6: 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.

7: 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.

8: **end for**

> **Remark.**
> Algorithm 14 (REINFORCE with Baseline) and Algorithm 15 (VPG) are essentially the same algorithm. Both perform a Monte-Carlo policy gradient update with a learned value function V_ϕ as baseline to reduce variance. The difference is presentational: Algorithm 14 follows the Sutton and Barto 2018 episodic formulation with an explicit discount weight γ^t, while Algorithm 15 follows the Achiam 2018b batch formulation that collects a set of trajectories $\mathcal{D}_k$ per iteration and uses the advantage estimate $\hat{A}_t = \hat{R}_t - V_\phi(s_t)$ directly. Under $\gamma = 1$ and a single-trajectory batch, the two updates are identical.

- We should notice that VPG is still a Monte-Carlo policy gradient

[30] Adapted from Achiam 2018b.

method, it needs to wait the end of an episode to compute the reward-to-go function $\hat{R}_t = \sum_{t'=t}^{T} r(s_{t'}, a_{t'})$.

- Some people also call this method advantage actor critic (A2C), because it uses the advantage function $A^\pi(s_t, a_t)$ to update the policy. But notice that if we use MC to estimate the advantage function, it is still a policy gradient method, not an actor-critic method. The actor-critic method use TD learning to estimate the value function. Here is the difference of VPG and A2C [link]. And we will introduce A2C in section 10.1

10 Model free, policy based, & policy gradient method: TD learning

- In the previous section 9.3, we discuss the policy gradient method with Monte-Carlo estimation. We involve baseline to reduce the variance of the policy gradient estimator. However, the MC estimation still has high variance and can lead to slow learning. To address this issue, we can use temporal-difference (TD) learning to estimate the value function, which allows us to update our estimates based on incomplete trajectories and can lead to faster learning.

10.1 1-step TD learning: Advantage Actor Critic (A2C)

- As we discussed in equation (35), in MC, we have to wait until the end of an episode to compute the reward-to-go function $\hat{R}_t = \sum_{t'=t}^{T} r(s_{t'}, a_{t'})$ to fit the value function. Instead, we can use TD learning for estimation.

- We can rewrite equation (24) following the methodology in equation

(30) as:

$$\begin{aligned}
\nabla_\theta J(\theta) =& \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t \mid s_t)\left(\sum_{t'=t}^{T} r(s_{t'}, a_{t'})\right)\right] \\
=& \sum_{t=1}^{T}\mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\left(r(s_t, a_t) + \sum_{t'=t+1}^{T} r(s_{t'}, a_{t'})\right)\right] \\
=& \sum_{t=1}^{T}\mathbb{E}_{\tau_{:t+1}\sim p_\theta}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\left(r(s_t, a_t) + \mathbb{E}_{\tau_{t+1:}\sim p_\theta}\left[\sum_{t'=t+1}^{T} r(s_{t'}, a_{t'}) \,\middle|\, \tau_{:t+1}\right]\right)\right] \\
=& \sum_{t=1}^{T}\mathbb{E}_{\tau_{:t+1}\sim p_\theta}[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\left(r(s_t, a_t) + V^{\pi_\theta}(s_{t+1})\right)] \\
=& \mathbb{E}_{\tau\sim p_\theta(\tau)}\left[\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_t \mid s_t)\left(r(s_t, a_t) + V^{\pi_\theta}(s_{t+1})\right)\right]. \qquad (36)
\end{aligned}$$

- Selecting $b(s_t) = V^{\pi_\theta}(s_t)$ as the baseline, we have:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim p_\theta(\tau)}[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\left(r(s_t, a_t) + V^{\pi_\theta}(s_{t+1}) - V^{\pi_\theta}(s_t)\right)] \qquad (37)$$

- We call this equation (37) the 1-step TD policy gradient estimator, because it uses a 1-step TD target $r(s_t, a_t) + V^{\pi_\theta}(s_{t+1})$ to replace the reward-to-go function. Some people also call this method the advantage actor critic (A2C) method, its pseudo-code is as follows:

Algorithm 16 One-step Actor–Critic (episodic), for estimating π[31]

Input: Differentiable policy parameterization $\pi(a \mid s, \theta)$
Input: Differentiable state-value function parameterization $\hat{v}(s, w)$
Input: Step sizes $\alpha^\theta > 0$, $\alpha^w > 0$
1: Initialize policy parameter $\theta \in \mathbb{R}^{d'}$ and value weights $w \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)
2: **loop** (for each episode)
3: Initialize S
4: $I \leftarrow 1$
5: **while** S is not terminal **do**
6: $A \sim \pi(\cdot \mid S, \theta)$
7: Take action A, observe S', R
8: $\delta \leftarrow R + \gamma\, \hat{v}(S', w) - \hat{v}(S, w)$ (if S' terminal, $\hat{v}(S', w) \doteq 0$)
9: $w \leftarrow w + \alpha^w \delta\, \nabla \hat{v}(S, w)$
10: $\theta \leftarrow \theta + \alpha^\theta I \delta\, \nabla \ln \pi(A \mid S, \theta)$
11: $I \leftarrow \gamma I$
12: $S \leftarrow S'$
13: **end while**
14: **end loop**

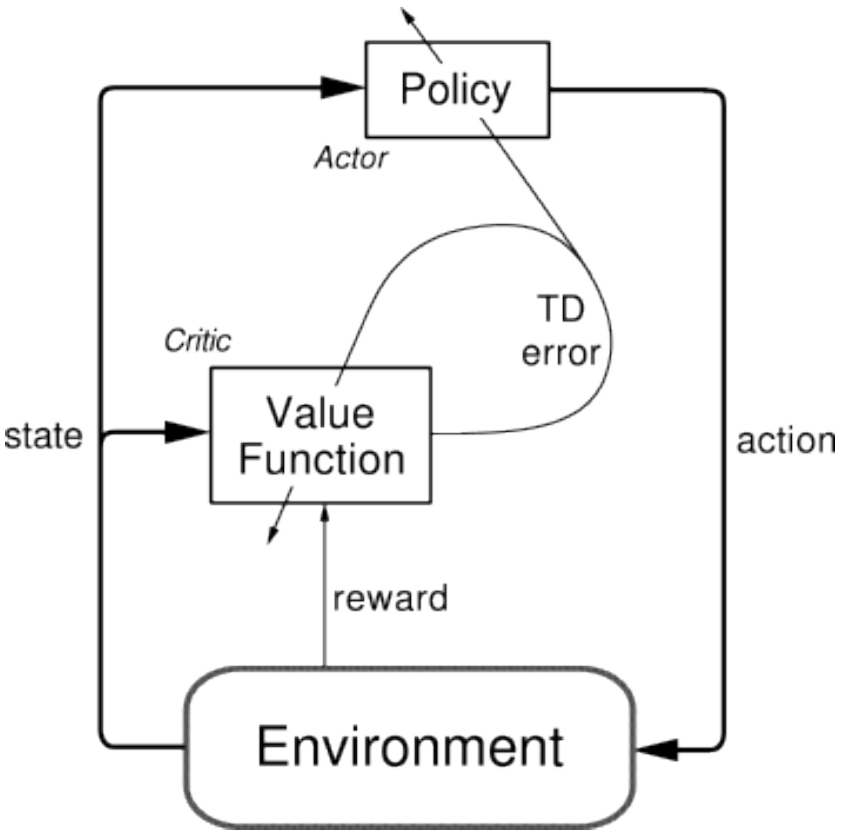
Remark.

Why do we use $w \leftarrow w + \alpha^w \delta\, \nabla \hat{v}(S, w)$ to update the value function parameters in row 9? This is the **semi-gradient** update. As in equation (35), we minimize the squared error between the estimated value and the TD target. Taking the gradient of $\frac{1}{2}(R + \gamma \hat{v}(S', w) - \hat{v}(S, w))^2 = \frac{1}{2}\delta^2$ with respect to w, and treating the TD target $R + \gamma \hat{v}(S', w)$ as a fixed constant (i.e., not differentiating through it), gives:

$$-\nabla_w \frac{1}{2}\delta^2 = \delta\, \nabla_w \hat{v}(S, w) \tag{38}$$

Hence the update $w \leftarrow w + \alpha^w \delta\, \nabla_w \hat{v}(S, w)$. It is called semi-gradient because the TD target itself depends on w but is treated as a constant — the full gradient would additionally include $-\gamma \nabla_w \hat{v}(S', w)$.

[31] Adapted from Sutton and Barto 2018, p. 332.



10.2 n-step TD learning

- We can also use n-step TD learning to estimate the value function. The n-step TD target is defined as:

$$G_t^{(n)} = r(s_t, a_t) + r(s_{t+1}, a_{t+1}) + \ldots + r(s_{t+n-1}, a_{t+n-1}) + V^{\pi_\theta}(s_{t+n}) \quad (39)$$

- The n-step TD policy gradient estimator is:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}\Big[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\left(G_t^{(n)} - V^{\pi_\theta}(s_t)\right)\Big] \quad (40)$$

10.3 Generalized Advantage Estimation (GAE): TD learning

- If we write the the n-step look ahead advantage function as:

$$A_t^{\pi_\theta,(n)} = G_t^{(n)} - V^{\pi_\theta}(s_t) \quad (41)$$

- Then we can further define the generalized advantage estimator GAE(λ) Schulman et al. 2016 as a weighted average of the n-step look ahead advantage functions:

$$A_t^{\pi_\theta,\lambda} = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}A_t^{\pi_\theta,(n)} \tag{42}$$

- Two special cases of GAE(λ) are:
- When $\lambda = 0$, GAE(λ) reduces to the 1-step TD advantage function:

$$A_t^{\pi_\theta,0} = A_t^{\pi_\theta,(1)} = r(s_t,a_t) + V^{\pi_\theta}(s_{t+1}) - V^{\pi_\theta}(s_t) \tag{43}$$

- When $\lambda = 1$, GAE(λ) reduces to the Monte-Carlo advantage function in equation (31):

$$A_t^{\pi_\theta,1} = \sum_{n=1}^{\infty}A_t^{\pi_\theta,(n)} = \sum_{t'=t}^{T} r(s_{t'},a_{t'}) - V^{\pi_\theta}(s_t) \tag{44}$$

- The online implementation of GAE(λ) is exactly the **Actor-Critic with Eligibility Traces** algorithm. The eligibility trace z^w accumulates $\nabla\hat{v}$ with decay λ^w, which is equivalent to computing the GAE-weighted critic update online; similarly z^θ does the same for the actor. Its pseudo-code is:

Algorithm 17 Actor–Critic with Eligibility Traces (episodic), for estimating π^*[32]

Input: Differentiable policy parameterization $\pi(a \mid s,\theta)$
Input: Differentiable state-value function parameterization $\hat{v}(s,w)$
Input: Trace-decay rates $\lambda^\theta \in [0,1]$, $\lambda^w \in [0,1]$; step sizes $\alpha^\theta > 0$, $\alpha^w > 0$
Initialize $\theta \in \mathbb{R}^{d'}$ and $w \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)
loop (for each episode)
  Initialize S
  $z^\theta \leftarrow \mathbf{0}, \quad z^w \leftarrow \mathbf{0}, \quad I \leftarrow 1$
  while S is not terminal **do**
    $A \sim \pi(\cdot \mid S,\theta)$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma\,\hat{v}(S',w) - \hat{v}(S,w)$ (if S' terminal, $\hat{v}(S',w) \doteq 0$)
    $z^w \leftarrow \lambda^w z^w + \nabla\hat{v}(S,w)$
    $z^\theta \leftarrow \lambda^\theta z^\theta + I\,\nabla \ln \pi(A \mid S,\theta)$
    $w \leftarrow w + \alpha^w \delta\, z^w$
    $\theta \leftarrow \theta + \alpha^\theta \delta\, z^\theta$
    $I \leftarrow \gamma I, \quad S \leftarrow S'$
  end while
end loop

Remark.

Algorithm 17 and GAE(λ) are the **same idea** in different implementations. Both exponentially weight past TD errors with λ: the eligibility trace accumulates online step-by-step, while GAE computes the equivalent quantity offline over a full batch. Setting $\lambda^w = \lambda^\theta = \lambda$:

- $\lambda = 0$**:** The trace resets each step, $z \leftarrow \nabla \hat{v}(S, w)$, so the update reduces to $w \leftarrow w + \alpha^w \delta \nabla \hat{v}$ and $\theta \leftarrow \theta + \alpha^\theta I \delta \nabla \ln \pi$ — exactly Algorithm 16 (A2C).
- $\lambda = 1$**:** The trace accumulates without decay, and the sum of TD errors telescopes: $\sum_{t=k}^{T} \delta_t = G_k - V(S_k)$, so the total update becomes $\sum_k \gamma^k (G_k - V(S_k)) \nabla \ln \pi$ — exactly Algorithm 14 (REINFORCE with Baseline).

Intermediate $\lambda \in (0, 1)$ interpolates between the two, trading bias for variance.

11 Further readings about actor-critic methods

Remark.

This section is written by generative AI and may contain errors. Please read with caution and verify with other sources.

11.1 Trust Region Policy Optimization (TRPO)

- TRPO aims to improve the policy while preventing excessively large policy updates that may collapse performance.
- Instead of directly maximizing the expected return

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)], \tag{45}$$

  TRPO optimizes a local surrogate objective around the old policy

$$\pi_{\theta_{\text{old}}}.$$

- Define the probability ratio

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}. \tag{46}$$

[32] Adapted from Sutton and Barto 2018, p. 332.

- The surrogate objective is

$$L(\theta) = \mathbb{E}_t \left[r_t(\theta) \hat{A}_t \right], \tag{47}$$

  where $\hat{A}_t$ is an estimator of the advantage function.

- The intuition is:
  - if $\hat{A}_t > 0$, increase the probability of action a_t,
  - if $\hat{A}_t < 0$, decrease the probability of action a_t.

- To avoid destructive policy updates, TRPO constrains the distance between the new policy and the old policy using KL divergence:

$$\bar{D}_{KL}\left(\theta_{\text{old}}, \theta\right) = \mathbb{E}_t \left[D_{KL}\left(\pi_{\theta_{\text{old}}} \,\|\, \pi_\theta\right) \right]. \tag{48}$$

- The constrained optimization problem becomes

$$\begin{aligned} \max_{\theta} \quad & L(\theta) \\ \text{subject to} \quad & \bar{D}_{KL}\left(\theta_{\text{old}}, \theta\right) \leq \delta. \end{aligned} \tag{49}$$

- Let

$$\theta = \theta_{\text{old}} + x,$$

  where x denotes the policy update step.

- Since the update is assumed to be small, perform a first-order Taylor expansion of the surrogate objective around θ_{old}:

$$L(\theta) \approx L(\theta_{\text{old}}) + g^T x, \tag{50}$$

  where

$$g = \nabla_\theta L(\theta)\Big|_{\theta=\theta_{\text{old}}}. \tag{51}$$

- Therefore, the local optimization objective becomes

$$\max_{x} \quad g^T x. \tag{52}$$

- Next, approximate the KL constraint using a second-order Taylor expansion.

- Since
$$\bar{D}_{KL}(\theta_{\text{old}}, \theta_{\text{old}}) = 0$$
and
$$\nabla_\theta \bar{D}_{KL}\Big|_{\theta=\theta_{\text{old}}} = 0,$$
the first-order term vanishes.

- Thus, the KL divergence is locally approximated by
$$\bar{D}_{KL}(\theta_{\text{old}}, \theta) \approx \frac{1}{2} x^T H x, \tag{53}$$
where
$$H = \nabla_\theta^2 \bar{D}_{KL}\Big|_{\theta=\theta_{\text{old}}} \tag{54}$$
is the Hessian of the KL divergence.

- The constrained optimization problem is therefore approximated as
$$\begin{aligned} \max_x \quad & g^T x \\ \text{subject to} \quad & \frac{1}{2} x^T H x \leq \delta. \end{aligned} \tag{55}$$

- Construct the Lagrangian
$$\mathcal{L}(x, \lambda) = g^T x - \lambda \left(\frac{1}{2} x^T H x - \delta \right). \tag{56}$$

- Taking the derivative with respect to x,
$$\nabla_x \mathcal{L} = g - \lambda H x. \tag{57}$$

- Setting the derivative to zero gives
$$g = \lambda H x. \tag{58}$$

- Solving for x,
$$x = \frac{1}{\lambda} H^{-1} g. \tag{59}$$

- This update direction corresponds to the natural gradient direction.

- To determine the step size, enforce the active constraint
$$\frac{1}{2} x^T H x = \delta. \tag{60}$$

- Substituting
$$x = \frac{1}{\lambda} H^{-1} g$$
into the constraint yields
$$\frac{1}{2} \frac{1}{\lambda^2} g^T H^{-1} g = \delta. \tag{61}$$

- Solving for λ,
$$\lambda = \sqrt{\frac{g^T H^{-1} g}{2\delta}}. \tag{62}$$

- Therefore, the final TRPO update step becomes
$$x^* = \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g. \tag{63}$$

- The parameter update rule is
$$\theta_{\text{new}} = \theta_{\text{old}} + x^*. \tag{64}$$

- In practice, the matrix H is too large to invert explicitly for neural network policies.

- The pseudo-code of TRPO is as follows:

Algorithm 18 Trust Region Policy Optimization[33]

Input: Initial policy parameters θ_0, initial value function parameters ϕ_0

Input: KL-divergence limit δ, backtracking coefficient α, maximum backtracking steps K

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

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

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

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

5: Estimate policy gradient as:

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

6: Use the conjugate gradient algorithm to compute:

$$\hat{x}_k \approx \hat{H}_k^{-1} \hat{g}_k,$$

where $\hat{H}_k$ is the Hessian of the sample average KL-divergence.

7: Update the policy by backtracking line search:

$$\theta_{k+1} = \theta_k + \alpha^j \sqrt{\frac{2\delta}{\hat{x}_k^T \hat{H}_k \hat{x}_k}} \hat{x}_k,$$

where $j \in \{0, 1, \ldots, K\}$ is the smallest value which improves the sample loss and satisfies the KL-divergence constraint.

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**

11.2 Proximal Policy Optimization (PPO)

- PPO is a policy gradient method designed to stabilize policy updates while avoiding the computational complexity of TRPO.

[33] Adapted from OpenAI Spinning Up: TRPO.

- Similar to TRPO, PPO starts from the policy gradient surrogate objective
$$L(\theta) = \mathbb{E}_t \left[r_t(\theta) \hat{A}_t \right], \tag{65}$$
where
$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \tag{66}$$
is the importance sampling ratio and $\hat{A}_t$ denotes the estimated advantage.

- The ratio $r_t(\theta)$ measures how much the new policy deviates from the old policy for the sampled action.

- If $\hat{A}_t > 0$, increasing $r_t(\theta)$ increases the objective, encouraging the policy to select action a_t more frequently.

- Conversely, if $\hat{A}_t < 0$, decreasing $r_t(\theta)$ reduces the probability of undesirable actions.

- However, directly maximizing Eq. (65) may lead to excessively large policy updates, causing unstable training.

- TRPO addresses this issue using a KL-divergence trust-region constraint, but solving the resulting constrained optimization problem requires second-order methods.

- PPO replaces the hard KL constraint with a clipped objective that approximately restricts the policy update within a local trust region.

- The clipped PPO objective is defined as
$$L^{CLIP}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t,\ \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t \right) \right], \tag{67}$$
where ϵ is a small clipping threshold, typically between 0.1 and 0.2.

- The clipping operation is defined as
$$\text{clip}(r, 1-\epsilon, 1+\epsilon) = \begin{cases} 1-\epsilon, & r < 1-\epsilon, \\ r, & 1-\epsilon \le r \le 1+\epsilon, \\ 1+\epsilon, & r > 1+\epsilon. \end{cases} \tag{68}$$

- The clipping mechanism prevents the importance ratio from moving too far away from 1, thereby limiting excessively large policy updates.

- Consider the case where
$$\hat{A}_t > 0.$$
- In this situation, the optimizer tends to increase $r_t(\theta)$ in order to increase the objective.
- However, once
$$r_t(\theta) > 1 + \epsilon,$$
the clipped term becomes constant, preventing the objective from growing further.
- Similarly, when
$$\hat{A}_t < 0,$$
the optimizer tends to decrease $r_t(\theta)$.
- The clipping mechanism prevents $r_t(\theta)$ from becoming excessively small, thereby avoiding overly aggressive suppression of actions.
- Therefore, PPO achieves stable policy optimization by softly constraining policy updates around the old policy.
- Unlike TRPO, PPO does not require:
  - Fisher information matrix inversion,
  - conjugate gradient optimization,
  - Hessian-vector products,
  - or line search procedures.
- PPO can therefore be optimized efficiently using standard first-order stochastic gradient descent methods.
- The parameter update rule becomes
$$\theta \leftarrow \theta + \alpha \nabla_\theta L^{CLIP}(\theta), \tag{69}$$
where α is the learning rate. The gradient $\nabla_\theta L^{CLIP}(\theta)$ has an explicit piecewise-analytic form. Let $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}$; then by the log-derivative trick $\nabla_\theta r_t = r_t \nabla_\theta \log \pi_\theta(a_t|s_t)$:
$$\nabla_\theta \min(r_t A_t,\ \mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\, A_t) = \begin{cases} r_t(\theta)\, A_t\, \nabla_\theta \log \pi_\theta(a_t|s_t) & \text{if } |r_t - 1| \leq \epsilon \text{ (unclipp} \\ 0 & \text{if } |r_t - 1| > \epsilon \text{ (clipped} \end{cases} \tag{70}$$

In the unclipped region the gradient reduces to the importance-weighted VPG gradient; in the clipped region the clip operation severs the dependence on θ, setting the gradient to zero. The clipping therefore acts as a **gradient mask**: it prevents excessively large updates by blocking gradient flow whenever the policy ratio strays outside $[1-\epsilon,\, 1+\epsilon]$.

- A key practical advantage of PPO is that the same batch of trajectories can be reused for multiple optimization epochs, significantly improving sample efficiency.
- PPO can therefore be interpreted as a first-order approximation to TRPO that preserves stable policy updates while greatly simplifying implementation and computation.
- The pseudo-code of PPO-Clip is as follows:

Algorithm 19 PPO-Clip[34]

Input: Initial policy parameters θ_0, initial value function parameters ϕ_0

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

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

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

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

5: Update the policy by maximizing the PPO-Clip objective:

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

typically via stochastic gradient ascent with Adam.

6: 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.

7: **end for**

[34] Adapted from OpenAI Spinning Up: PPO.

References

Achiam, Joshua (2018a). Introduction to RL — Part 3: Intro to Policy Optimization. https://spinningup.openai.com/en/latest/spinningup/rl_intro3.html. OpenAI.

— (2018b). Vanilla Policy Gradient. https://spinningup.openai.com/en/latest/algorithms/vpg.html. OpenAI.

Value Iteration (2024). https://gibberblot.github.io/rl-notes/single-agent/value-iteration.html.

Hui, Jonathan (2018a). RL — Actor-Critic Methods (A3C, GAE, DDPG, Q-Prop). https://jonathan-hui.medium.com/rl-actor-critic-methods-a3c-gae-ddpg-q-prop-e1c41f268541.

— (2018b). RL — Policy Gradients Explained. https://jonathan-hui.medium.com/rl-policy-gradients-explained-9b13b688b146.

— (2018c). RL — Policy Gradients Explained (Advanced Topic). https://jonathan-hui.medium.com/rl-policy-gradients-explained-advanced-topic-20c2b81a9a8b.

Levine, Sergey (2017). Deep Reinforcement Learning, Lecture 5: Actor-Critic Methods. http://rail.eecs.berkeley.edu/deeprlcourse-fa17/f17docs/lecture_5_actor_critic_pdf.pdf.

— (2023). Deep Reinforcement Learning, Lecture 5. https://rail.eecs.berkeley.edu/deeprlcourse-fa23/deeprlcourse-fa23/static/slides/lec-5.pdf.

Schulman, John et al. (2016). "High-Dimensional Continuous Control Using Generalized Advantage Estimation". In: International Conference on Learning Representation arXiv: 1506.02438.

Sutton, Richard S. and Andrew G. Barto (2018). Reinforcement Learning: An Introduction. 2nd ed. MIT Press. URL: http://incompleteideas.net/book/RLbook2020.pdf.

Wang, Hsin-Hung (2023). Intro to Reinforcement Learning: Monte Carlo to Policy Gradient. https://medium.com/@hsinhungw/intro-to-reinforcement-learning-monte-carlo-to-policy-gradient-1c7ede4eed6e.

Wikipedia contributors (2024a). Policy Gradient Method — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Policy_gradient_method.

— (2024b). Reinforcement Learning — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Reinforcement_learning.

Williams, Ronald J. (1992). "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning". In: Machine Learning 8.3–4, pp. 229–256.

Young, Brian (2023). Types of Reinforcement Learning Algorithms. https://wandb.ai/byyoung3/ML_NEWS3/reports/Types-of-reinforcement-learning-algorithms---VmlldzoxMzI4NjgzMw.

