Policy Gradients

Prof. Alfio Ferrara

Reinforcement Learning

Introduction

We discuss how, instead of estimating the value of actions in order to pick up the right one, we can learn a parametrized policy as a tool to select actions. In VFA, the policy is generated from the value function, for example using ϵ-greedy strategy. Now, we will directly parametrize the policy:

$$\pi_\theta(s,a) = \mathbb{P}(a \mid s;\theta) \tag{25}$$

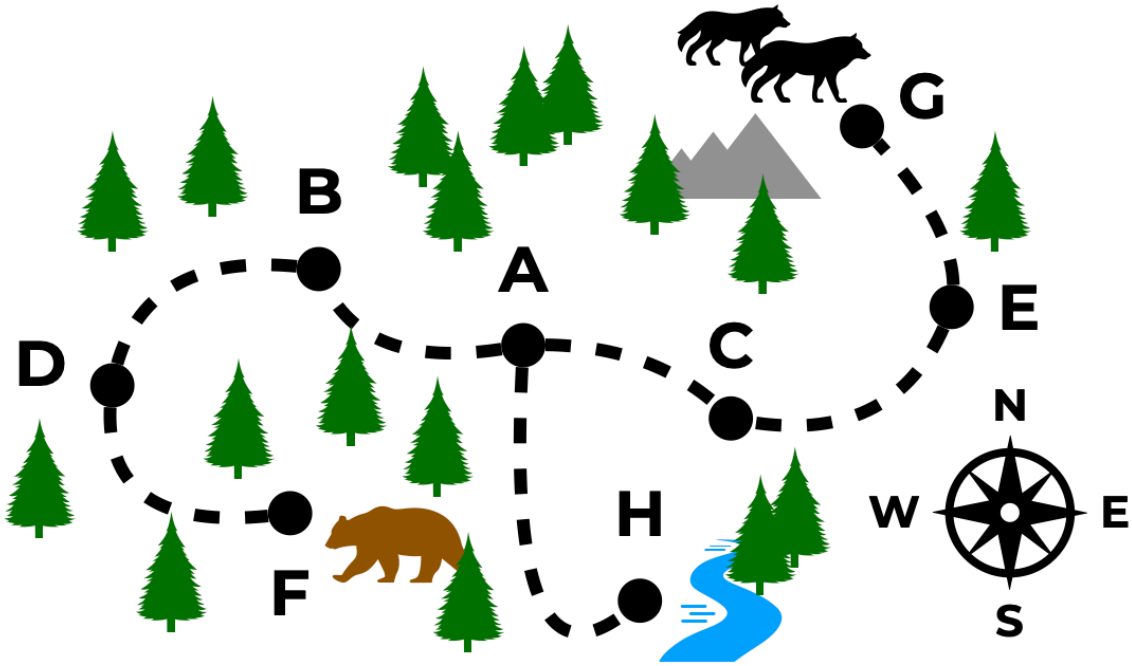
in order to find a policy π with the best V^π. Also in this case, we work on model-free algorithms.

One of the main advantages of policy gradient is that we can learn **stochastic policies** instead of deterministic policies. An example of a situation where a deterministic policy may not be optimal is when the environment is not **agnostic** with respect to the actions of the agent. This coul be the case of many games, like Tic-Tac-Toe. In this setting, a **deterministic policy can be exploited** by the opponent to learn an optimal strategy against the agent policy. Thus, we need to learn a stochastic policy that guarantee an equilibrium between the two opponents in the game.

Another example in which a deterministic policy is not optimal is when some states cannot be distinguished one from the other, but they lead to different outcomes.

Example and motivations

Let's take the case of an agent who is moving in a forest, looking for water.



Now suppose that each location from A to H is a state. Each state is described only by a binary vector $s = [N, S, W, E]$, where each dimension represents the availability of a path in one of the possible directions. We now have the following description (aka *perception*) of the environment states.

State	North	South	West	East	Reward
A	0	1	1	1	0
B	0	0	1	1	0
C	0	0	1	1	0
D	1	1	0	0	0
E	1	1	0	0	0
F	0	0	1	0	-1
G	0	1	0	0	-1
H	0	0	1	0	+1

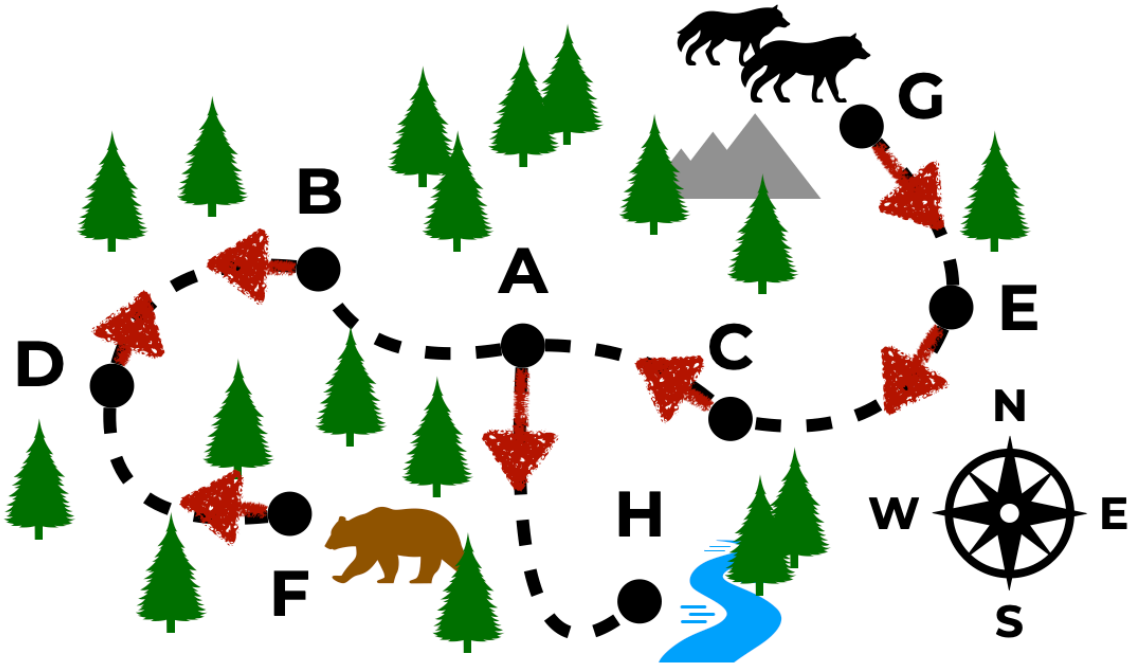
With tabular methods, states B and C are distinct states for wich our methods will learn different values. But in the approximate case, if we suppose that the only description available for the states in the binary vector s that provides the possible directions, it's easy to see that, from the Agent perspective, **states B and C are identical**.

Recall that the state-action function in the approximate case is:

$$Q(s, a; \mathbf{w}) = \mathbf{x}(s)^T \mathbf{w} \tag{26}$$

Thus, a VFA agent will eventually assign the same highest value to action `West` for both B and C, like in the following situation.

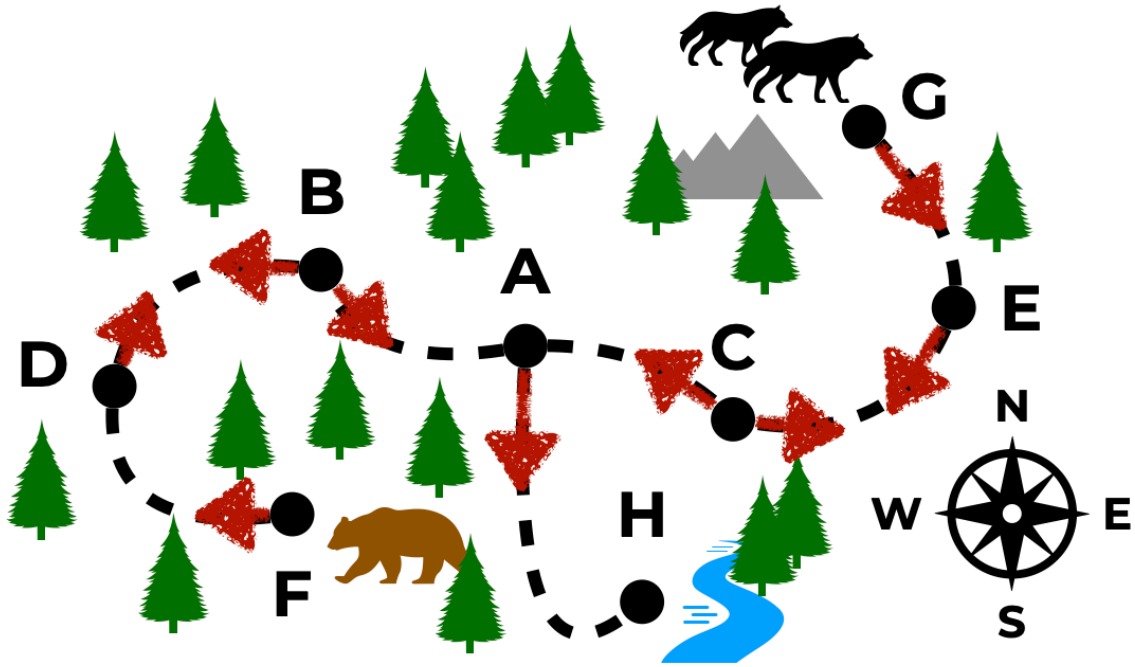
$$P(\pi(B) = \text{West}) = 1 \tag{27}$$
$$P(\pi(C) = \text{West}) = 1 \tag{28}$$



Note that for C this is actually the best policy, but for B, **with this policy, the agent enters a dead end** leading to the path from B to D and from D to B.

This can be avoided by a stochastic policy that makes it possible to escape the dead end with non-zero probability, such as.

$$P(\pi(B) = \text{West}) = 0.5 \tag{29}$$
$$P(\pi(B) = \text{East}) = 0.5 \tag{30}$$
$$P(\pi(C) = \text{West}) = 0.5 \tag{31}$$
$$P(\pi(C) = \text{East}) = 0.5 \tag{32}$$

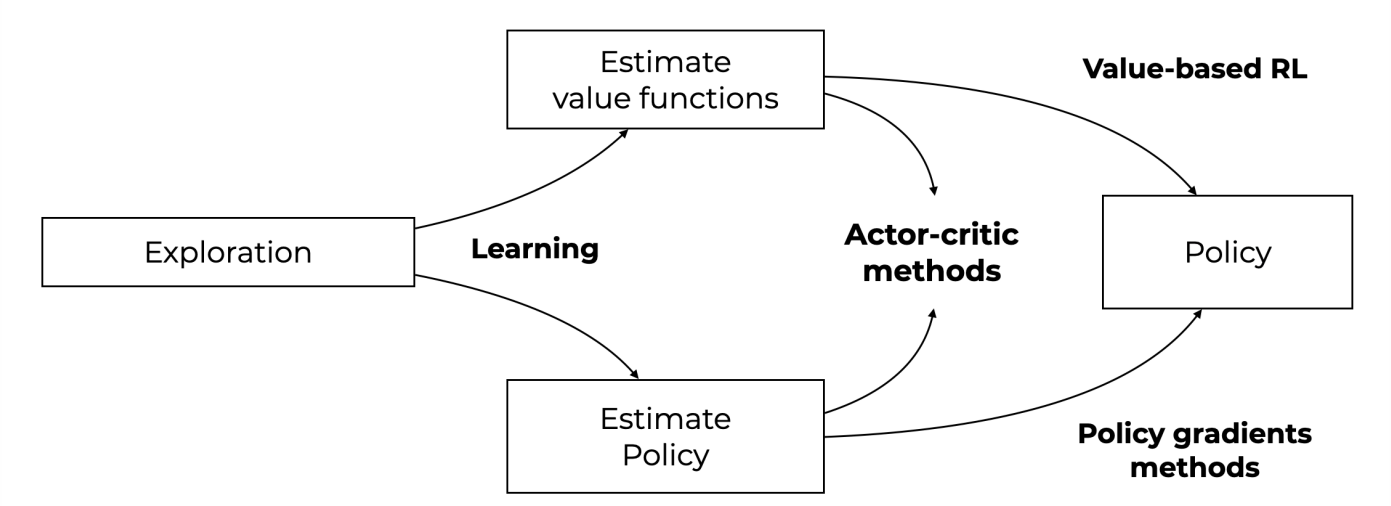


The general idea of policy gradient is to directly learn the policy π without exploiting the $Q(s, a)$ function. This is done by updating the parameters θ with respect to some performance measure $\mathcal{J}(\theta_t)$. Since we want to maximize the performances, we exploit gradient *ascent* as:

$$\theta_{t+1} = \theta_t + \alpha \nabla \mathcal{J}(\theta_t) \tag{33}$$

There are several methods that follow this schema, and their are called **policy gradient methods**.

Instead, methods that learn how to approximate both the policy and the value functions are called **actor-critic** methods, where **actor** represents the policy learning process, while **critic** represents the value function learning process.



Advantages

The main advantages of policy gradient methods are two:

- We enforce the policy to never become deterministic, so that we ensure always some exploration

Example: This is always an advantage in all the cases where we have almost identical states with different outcomes (see the forest example) or that a greedy or ϵ-greedy policy can be easily exploited by an opponent (for example in games like rock-paper-scissor).

- Policy-based methods are also useful for dealing with continuous action spaces

Example: Let's think to a autonomous driving car where the state is given by a set of features like (x_1, x_2, x_3, x_4). We can just simply model actions as $a_t = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$ to just learn the policy based on state features and parameters.

With discrete actions spaces, one natural parametrization for actions is to associate numerical scores based on parameters to each (s, a), $score(s, a, \theta)$, so that actions can be selected according to a distribution of those values, like exponential soft-max (*soft-max in action preferences*).

$$\pi(a \mid s, \theta) = \frac{e^{score(s,a,\theta)}}{\sum\limits_{a' \in A} e^{score(s,a',\theta)}} \tag{34}$$

The *score* function can then be learn arbitrary using a neural network or just as a lienar approximation of a based on a feature vector $score(s, a, \theta) = \theta^T \mathbf{x}(s, a)$.

Another advantage of policy learning is to have a smooth strategy for improving the policy during learning is a more stable way.

Let's take the previous example where $a_t = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$.

With greedy-like policy improvement strategy, we update the policy as follows:

$$\theta_{t+1} = \arg\max_{\theta} \hat{Q}^{\pi(s,a)} \tag{35}$$

This could lead to a situation where a small change in the estimation of $\hat{Q}^{\pi}(s, a)$ can generate a large change in the policy, which could in turn produce instability.

On the contrary, the direct upgrade of the policy could be mode stable:

$$\theta_{n+1} = \theta_n + \alpha \nabla G \theta_n \tag{36}$$

with $G = \sum_{t=0}^{\infty} \gamma^t r_t$.

Policy gradient

The goal of policy gradient methods is to learn a set of **parameters θ that maximizes the long-term reward**.

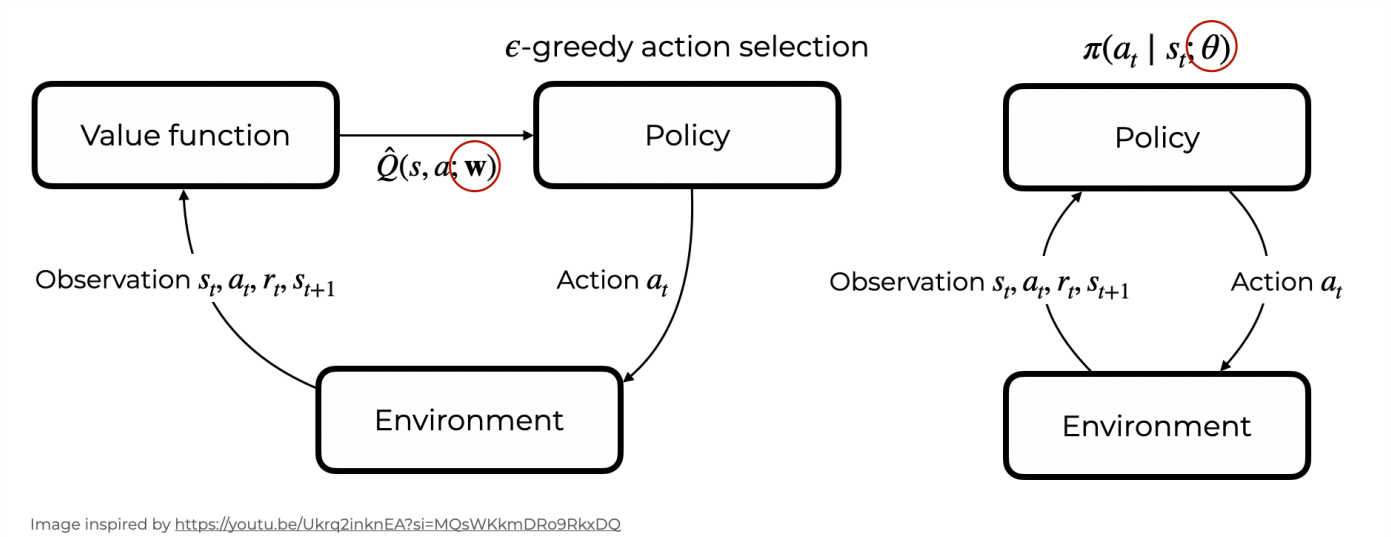


Image inspired by https://youtu.be/Ukrq2inknEA?si=MQsWKkmDRo9RkxDQ

Problem setting

Let's start by recap the general setting: first, we have a trajectory τ

$$\tau = (s_0, a_0, r_0, s_1, \ldots, s_{\tau-1}, a_{\tau-1}, r_{\tau-1}, s_{\tau}) \tag{37}$$

Where $r_i = R(s_i, a_i)$ and $G(\tau) = \sum_{t=0}^{\tau-1} \gamma^t r_t$.

In this setting we want to maximize the expected reward: $\max_{\theta} \mathbb{E}_{\pi_\theta} [G(\tau)]$.

In order to do so, we define an objective function $\mathcal{J}(\theta)$ that estimates $\max_{\theta} \mathbb{E}_{\pi_\theta}[G(\tau)]$ and we perform **gradient ascent**, which is the produce for searching a maximum for $\mathcal{J}(\theta)$.

$$\Delta_\theta = \alpha \nabla_\theta \mathcal{J}(\theta) \tag{38}$$

Expectation. If we define $P(\tau \mid \theta)$ as the probability of the trajectory τ under the policy π_θ, the expected total reward can be re-defined as:

$$\nabla_\theta \mathbb{E}_{\pi_\theta} G(\tau) = \nabla_\theta \sum_\tau P(\tau \mid \theta) G(\tau) \tag{39}$$

Now, we can reduce the formula as follows:

- $\sum_\tau \nabla_\theta P(\tau \mid \theta) G(\tau)$
- $\sum_\tau \frac{P(\tau|\theta)}{P(\tau|\theta)} \nabla_\theta P(\tau \mid \theta) G(\tau)$ by multiply and divide by $P(\tau \mid \theta)$
- $\sum_\tau P(\tau \mid \theta) \nabla_\theta \log P(\tau \mid \theta) G(\tau)$ since $\nabla_x \log f(x) = \frac{\nabla_x f(x)}{f(x)}$
- $\mathbb{E}_{\pi_\theta}\left(\nabla_\theta \log P(\tau \mid \theta) G(\tau)\right)$

Now we can address the problem of estimating the probability $P(\tau \mid \theta)$ of a trajectory τ under the policy parameters θ, that is

$$P(\tau \mid \theta) = P(s_0) \prod_{t=0}^{\tau-1} P(s_{t+1} \mid s_t, a_t) \pi_\theta(a_t \mid s_t) \tag{40}$$

Note that the issue here is that we cannot estimate such a quantity in a model-free environment, because we do not know the transition probability $P(s_{t+1} \mid s_t, a_t)$ that describes the dynamics of the MDP.

However, we can compute the gradients of the log-probability as shown before:

$$\nabla_\theta \log P(\tau \mid \theta) = \nabla_\theta \left(\log P(s_0) + \sum_{t=0}^{\tau-1} \log P(s_{t+1} \mid s_t, a_t) + \log \pi_\theta(a_t \mid s_t) \right) \tag{41}$$

Here, using the log, we substitute the product with the sum and we introduced the task of computing the gradients **with respect to** θ. Thus, note that sinche the only component that depends on θ is the policy π_θ, we succesfully eliminated the unknown $P(s_{t+1} \mid s_t, a_t)$ from our computation.

$$\nabla_\theta \log P(\tau \mid \theta) = \sum_{t=0}^{\tau-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \tag{42}$$

This demonstrates that the gradients of the probability of the trajectory under θ depends only by the gradients of the policy π_θ. We can now use this into the update formula:

$$\nabla_\theta \mathbb{E}_{\pi_\theta} G(\tau) = \nabla_\theta \log P(\tau \mid \theta) G(\tau) = \sum_{t=0}^{\tau-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t) G(\tau) \tag{43}$$

Which simply means that we want to increase probabilities for actions leading to positive returns and decrease the probability of actions leading to negative returns.

Monte Carlo sampling

One very simple strategy to approximate this estimation is to sample several trajectories:

$$\nabla_\theta \mathbb{E}_{\pi_\theta} G(\tau) \approx \frac{1}{\mid \tau \mid} \sum_\tau \sum_{t=0}^{\tau-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t) G(\tau) \tag{44}$$

This formula however implies some useless steps. In fact, when we are selecting action a_t, it makes sense to update its probabilities by taking into account only the total reward we cumulate from $t+1$ on. There's no reason for evaluating the action a_t by taking into account the reward cumulated before t. Thus, we can reformulate as:

$$\nabla_\theta \mathbb{E}_{\pi_\theta} G(\tau) \approx \frac{1}{\mid \tau \mid} \sum_\tau \sum_{t=0}^{\tau-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \sum_{t'=t}^{\tau-1} \gamma^{t'-t} R(s_{t'}, a_{t'}) \tag{45}$$

REINFORCE: Monte-Carlo Policy Gradient Control

Input: $\pi(a \mid s; \theta)$

- Init $\theta \in \mathbb{R}^d$
- Loop for episodes
  - Generate a episode $(s_0, a_0, r_0, s_1, a_1, \dots, s_{T-1}, a_{T-1}, r_{T-1}, s_T)$ using $\pi(\cdot \mid \cdot; \theta)$
  - Loop for each step $t \in T$:
    - $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k$
    - $\theta \leftarrow \theta + \alpha \gamma^t G \nabla_\theta \log \pi(a_t \mid a_t; \theta)$

Advantage functions

Some variants of comtrol with policy gradient are based on the idea of using **advantage functions** that model the specific advantage in taking soe actions with respect to others, by decoposing the state-value into a state-dependent component and an action-dependent component:

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s) \tag{46}$$

REINFORCE with Advantage functions

Input: $\pi(a \mid s; \theta)$ and $V(s; \mathbf{w})$

- Init θ and $\mathbf{w}$ to $\mathbb{R}^d$, with d action and state space size, respectively
- Loop for espisodes
  - Generate a episode $(s_0, a_0, r_0, s_1, a_1, \dots, s_{T-1}, a_{T-1}, r_{T-1}, s_T)$ using $\pi(\cdot \mid \cdot; \theta)$
  - Loop for each step $t \in T$:
    - $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k$
    - $\delta \leftarrow G - V(s_t; \mathbf{w})$ (TD error)
    - $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla_{\mathbf{w}} V(s_t; \mathbf{w})$

- $\theta \leftarrow \theta + \alpha^{\theta} \gamma^t \delta \nabla_{\theta} \log \pi(a_t \mid s_t; \theta)$

Actor-Critic Methods

The main idea in the Actor-Critic paradigm is to exploit two functions (eventually approximated by neural networks) that estimate two different components of the RL problem:

- **Actor**: estimates the policy $\pi(a \mid s; \theta)$ gradients and how to update the policy
- **Critic**: estimates $Q(s, a; \mathbf{w})$ and provides a evaluation of the actions taken by the Actor.

In practice this means that:

- **Actor**: updates θ by policy gradient
- **Critic**: updates $\mathbf{w}$ some way (such as by TD)

The main motivation for theese algorithms is that we want to use online TD updates instead of episodic return value in order to reduce the variance that tipically affects Monte-Carlo methods.

We are also going to use a straightforward advantage function $A^{\pi}(s_s, a_t)$ defined as:

$$A^{\pi}(s_t, a_t) = r + \gamma V^{\pi}(s_{t+1}) - V^{\pi}(s_t) \tag{47}$$

One-step Actor Critic

Input: a policy $\pi(a \mid s; \boldsymbol{\theta})$ and a parametrized value function $V(s; \mathbf{w})$

- Init $\boldsymbol{\theta}$ and $\boldsymbol{w}$ in $\mathbb{R}^d$ w.r.t. the action and space feature dimensions
- Loop for each episode:
  - Initialize $s = s_0$
  - $I \leftarrow 1$
  - Loop while s is not terminal:
    - $a \leftarrow \pi(\cdot \mid s; \boldsymbol{\theta})$
    - Take a, observe r, s'
    - $\delta \leftarrow r + \gamma \hat{V}(s', \boldsymbol{w}) - \hat{V}(s, \boldsymbol{w})$
    - $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha^{\boldsymbol{w}} \delta \nabla_{\boldsymbol{w}} \hat{V}(s, \boldsymbol{w})$
    - $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \boldsymbol{\theta} I \delta \nabla_{\boldsymbol{\theta}} \log \pi(a \mid s, \boldsymbol{\theta})$
    - $I = \gamma I$
    - $s = s'$

Actor Critic with eligibility traces

Input: a policy $\pi(a \mid s; \boldsymbol{\theta})$ and a parametrized value function $V(s; \mathbf{w})$

Input: eligibility trace decay factors $\lambda^{\boldsymbol{\theta}}$ and $\lambda^{\boldsymbol{w}}$ both $\in [0, 1]$

- Init $\boldsymbol{\theta}$ and $\boldsymbol{w}$ in $\mathbb{R}^d$ w.r.t. the action and space feature dimensions

- Loop for each episode:
  - Initialize $s = s_0$
  - Init $\boldsymbol{z^\theta}$ and $\boldsymbol{z^w}$ as eligibility trace vectors for policy and value-function
  - Loop while s is not terminal:
    - $a \leftarrow \pi(\cdot \mid s; \boldsymbol{\theta})$
    - Take a, observe r, s'
    - $\delta \leftarrow r + \gamma \hat{V}(s', \boldsymbol{w}) - \hat{V}(s, \boldsymbol{w})$
    - $\boldsymbol{z^w} \leftarrow \gamma \lambda^{\boldsymbol{w}} \boldsymbol{z^w} + \nabla_{\boldsymbol{\theta}} \hat{V}(s, \boldsymbol{w})$
    - $\boldsymbol{z^\theta} \leftarrow \gamma \lambda^{\boldsymbol{\theta}} \boldsymbol{z^\theta} + I \nabla_{\boldsymbol{\theta}} \log \pi(a \mid s; \boldsymbol{\theta})$
    - $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha^{\boldsymbol{w}} \delta \boldsymbol{z^w}$
      - $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \boldsymbol{\theta} \delta \boldsymbol{z^{theta}}$
    - $I = \gamma I$
    - $s = s'$

Dealing with Continuous Actions

Almost all the previous solutions can be applied to a a continuous action space. A common approach is to use a Gaussian policy, where the mean is some kind of (differentiable) state function $\mu(s)$. We can set up a constant variance σ^2 (of parametrize it).

According to this approach, the **policy** becomes a distribution over the states, like $a \sim \mathcal{G}(\mu(s), \sigma^2)$.

The gradient is then

$$\nabla_\theta \log \pi_\theta(s, a) = \frac{a - \mu(s)}{\sigma^2} \nabla \mu(s) \tag{48}$$