ECE433/COS435 Spring 2024 Introduction to RL

Lecture 7: Policy Gradient

1. Recap: Value Function

$$Q^\pi(s,a) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t) \mid s_0 = s, a_0 = a\right]$$

$$V^\pi(s) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t) \mid s_0 = s\right]$$

2. RL as Policy Optimization

The core objective of RL is to find a policy that maximizes cumulative reward in a given environment. This task is an optimization problem over the space of possible policies:

$$\max_\pi V^\pi(s_0) := \mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t) \mid s_0\right]$$

A policy π is essentially a mapping from states (or state-action pairs) to distributions of actions. The central challenge in RL is to search the policy space effectively which can be very large.

More generally, we often work with a parameterized policy, typically denoted as $\pi_\theta(a|s)$, where θ represents the parameters of the policy, s is the state, and a is the action. The policy defines the probability of taking action a in state s. A common choice is the softmax policy parametrization (similar to softmax classifier):

$$\pi_\theta(a|s) = \frac{e^{\phi(s,a)^\top\theta}}{\sum_{k=1}^N e^{\phi(s,a_k)^\top\theta}}$$

where ϕ is some feature map (for example a neural network).

Now the policy optimization problem becomes an optimization problem in the space of policy parameters θ (for example weights of the softmax policy network)

$$\max_\theta J(\theta) := \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t) \mid s_0\right]$$

where we use $J(\theta) = V^\pi(s_0)$ to denote the objective function which is also value of initial state.

Treating RL as Black-Box Optimization?

Black-box optimization is a domain of optimization, solving problem of the form $\max_x f(x)$ where the objective function does not have an explicit analytical form or its derivatives are not accessible. It implies a minimal assumption about the underlying function's structure, relying on evaluating the function at various points to find the optimum.

Can we treat RL as a black-box optimization problem? This would presents challenges:

- In the context of RL, black-box optimization would mean optimizing a policy based solely on the observed returns, without utilizing the rich structure of the environment or the temporal nature of decisions and rewards. Such an approach can be extremely sample-inefficient: It requires an immense number of trials to adequately explore the policy space, with A^S unique deterministic policies.
- A black-box approach would struggle to capitalize on the typical Markovian nature of RL problems, where leveraging the Markov Decision Process (MDP) framework can significantly reduce the complexity by breaking down the problem into more manageable sub-problems based on state transitions and rewards.

Next, what about gradient optimization?

Gradient optimization, on the other hand, is a cornerstone of many machine learning algorithms. It involves using the gradient of an objective function to guide the search for optima.

Let's review the notion of gradient, which is also known as the **steepest ascent direction**. Take a specific example:

$$f(x, y) = x^2y + 3xy^2$$

The gradient of this function, denoted as $\nabla f(x, y)$, is a vector of partial derivatives. For our function $f(x, y)$, the gradient is represented as a column vector:

$$\nabla f(x, y) = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

The partial derivatives are calculated as:

$$\frac{\partial f}{\partial x} = \frac{\partial}{\partial x}(x^2y + 3xy^2) = 2xy + 3y^2$$

$$\frac{\partial f}{\partial y} = \frac{\partial}{\partial y}(x^2y + 3xy^2) = x^2 + 6xy$$

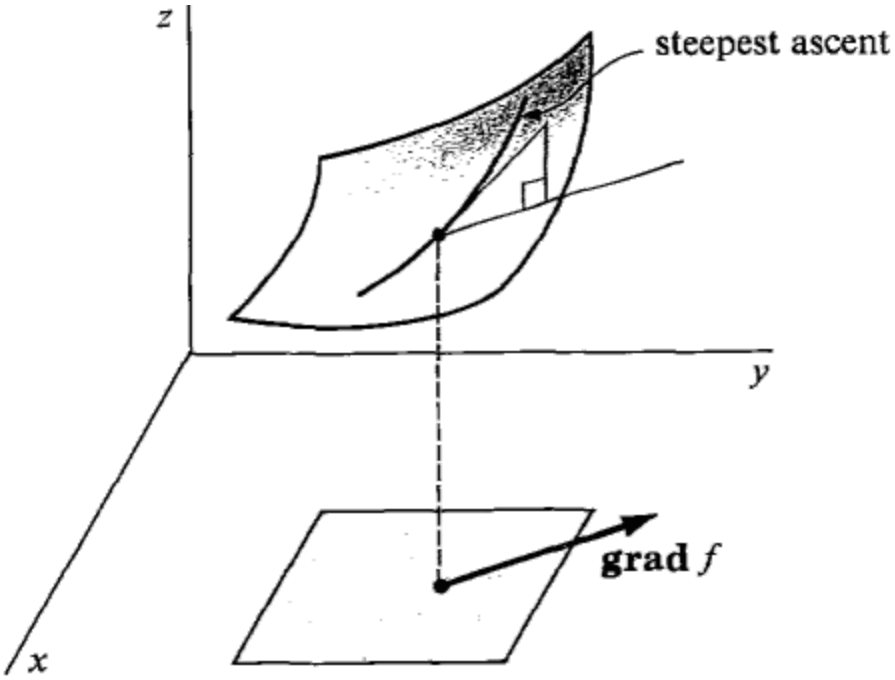
Therefore, the gradient of $f(x, y)$ is:

$$\nabla f(x, y) = \begin{bmatrix} 2xy + 3y^2 \\ x^2 + 6xy \end{bmatrix}$$

For instance, to find the gradient at a specific point, like $(1, 2)$, substitute $x = 1$ and $y = 2$:

$$\nabla f(1, 2) = \begin{bmatrix} 2 \cdot 1 \cdot 2 + 3 \cdot 2^2 \\ 1^2 + 6 \cdot 1 \cdot 2 \end{bmatrix} = \begin{bmatrix} 4 + 12 \\ 1 + 12 \end{bmatrix} = \begin{bmatrix} 16 \\ 13 \end{bmatrix}$$

This result indicates that at the point $(1, 2)$, the direction of the steepest ascent of the function is given by the vector $\begin{bmatrix} 16 \\ 13 \end{bmatrix}$.



Gradient optimization offers significant advantages. Primarily, it provides an efficient way to navigate the complex landscape of high-dimensional objective functions, steering towards the optimal solution by following the direction of steepest descent. This method is particularly effective for large-scale problems due to its scalability and adaptability. Moreover, gradient optimization techniques, such as gradient descent, can be easily combined with backpropagation in neural networks, enabling the effective training of deep learning models.

Algorithms like Gradient Descent and its variants (Stochastic Gradient Descent, Mini-batch Gradient Descent, Adam, etc.) are widely used in modern deep learning. They are particularly powerful for optimization in high-dimensional parameter space.

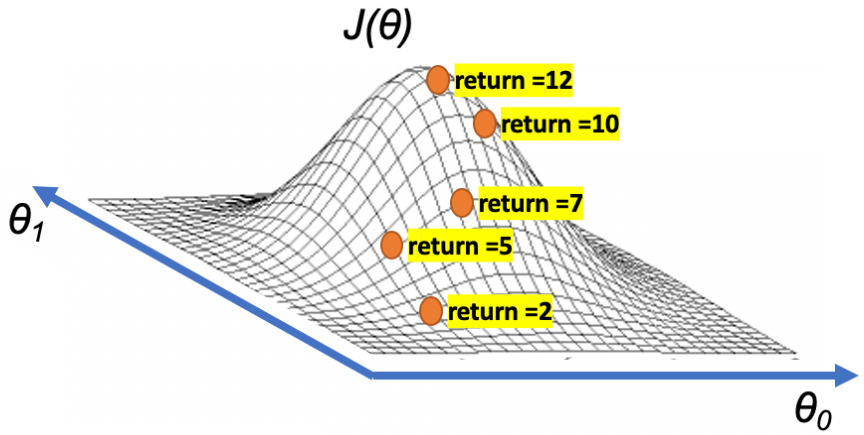
Policy Gradient Method

Motivated by gradient optimization, policy gradient method naturally emerges as a powerful tool in RL. By representing policies with parameterized functions (like neural networks), we can use gradient ascent to iteratively improve our policies.

$$\theta_{t+1} \leftarrow \theta_t + \alpha \nabla J(\theta_t)$$

where α is often a tunable learning rate.

In practice, policy gradient methods demonstrate remarkable versatility and robustness, handling challenges like delayed rewards and partial observability more gracefully than many alternative approaches. They excel in tasks where the quality of actions is continuously variable, such as robotic control, autonomous navigation, and game playing, where precision and adaptability are paramount.



This approach hinges on the **availability of gradient information**, which provides a direct indication of how to adjust the inputs (like model parameters) to improve the output (such as minimizing a loss function).

But what do we miss here?

- We need the policy gradient $\nabla_\theta J(\theta)$! (today's agenda)
- Make it efficient (we will work on this later in the course)

3. Finding Policy Gradient for MAB

A simple framework to introduce the concept of policy gradient is the Multi-Armed Bandit problem. In this context, the problem is to optimize over a distribution over action space to maximize expected return.

Let's formulate the optimization problem

Consider a bandit with N arms, each providing a stochastic reward. The policy π_θ is parameterized by θ, where $\pi_\theta(i)$ corresponds to the probability of choosing arm i. Our goal is to maximize the expected reward.

The objective function can be defined as:

$$J(\theta) = \mathbb{E}[R|i \sim \pi_\theta] = \sum_{i=1}^{N} \pi_\theta(i) R_i \tag{1}$$

where R is the stochastic reward and R_i is the expected reward for choosing arm i.

Find the gradient of expected reward with respect to θ

We need to compute the gradient of $J(\theta)$ with respect to θ. The gradient is given by:

$$\nabla_\theta J(\pi) = \nabla_\theta \left(\sum_{i=1}^{N} \pi_\theta(i) R_i \right) = \sum_{i=1}^{N} R_i \nabla_\theta \pi_\theta(i) \tag{2}$$

Thoughts:

- The previous formula is not directly computable because we do not know R_i.
- We hope to reformulate the gradient as a form of expectation over random rewards, making it easier to estimate by sampling.

Next, one key step is to use the **log likelihood trick**, which leverages the fact that

$$\nabla_\theta \log \pi_\theta(i) = \frac{\nabla_\theta \pi_\theta(i)}{\pi_\theta(i)}.$$

Thus, we can rewrite the gradient as:

$$\nabla_\theta J(\theta) = \sum_{i=1}^{N} R_i \pi_\theta(i) \nabla_\theta \log \pi_\theta(i) = \mathbb{E}[R \nabla_\theta \log \pi_\theta(i) | i \sim \pi_\theta] \tag{3}$$

Estimating the gradient

Notice that the policy gradient takes an expectation in the probability space associated with the policy π_θ. This is very nice! It means that one can estimate the PG easily by following policy π_θ repeatedly:

- Choosing actions $i_t \sim \pi_\theta$ independently and drawing reward observations r_t.
- Then estimate the PG by

$$\frac{1}{T} \sum_{t=1}^{T} r_t \nabla_\theta \log \pi_\theta(i_t).$$

This Monte Carlo estimate of policy gradient is easily computable: r_t is observed reward, and $\log \pi_\theta(i_t)$ is computable

4. Policy Gradient for Reinforcement Learning

The main idea of Policy Gradient Method is to adjust the parameters of the policy in a direction that increases the expected return, as indicated by the gradient of the return with respect to the policy parameters. This approach allows for efficient search in high-dimensional policy spaces, particularly in problems with continuous action domains or complex state representations.

4.1. Policy Gradient Theorem

First let's derive the form of policy gradient for Markov decision processs.
Consider a reinforcement learning problem with a policy $\pi_\theta(a|s)$ parameterized by θ, and a state-action value function $Q^{\pi_\theta}(s,a)$.
The objective is to maximize the expected return:

$$\max_\theta J(\theta) := \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t) \mid s_0\right]$$

Luckily, the policy gradient theorem comes to save the world! Woohoo! It provides a nice reformation of the derivative of the objective function to not involve the derivative of the state distribution $d^\pi(.)$ and simplify the gradient computation of $\nabla_\theta J(\theta)$ a lot.

Policy Gradient Theorem (Williams 1992)

$$\nabla J(\theta) = \mathbb{E}_{\pi_\theta}[Q^{\pi_\theta}(s,a)\nabla_\theta \log \pi_\theta(a|s)] \tag{4}$$

This equation provides a way to improve the policy by adjusting the parameters in the direction of the gradient of expected return.

4.2. Proof of Policy Gradient Theorem (Sutton & Barto, 2017; Sec. 13.2)

Note: This proof establishes the most elegant and important theory of RL. It established the foundation for all policy optimization algorithms that gave rise to powerful modern deep RL algorithms. Although we would not ask you to derive such a proof in homework or exams, we encourage you to understand the essence of derivation.

- We first start with the derivative of the state value function.

$$\begin{aligned}
&\nabla_\theta V^\pi(s) \\
=&\nabla_\theta\left(\sum_{a\in\mathcal{A}}\pi_\theta(a \mid s)Q^\pi(s,a)\right) \\
=&\sum_{a\in\mathcal{A}}(\nabla_\theta\pi_\theta(a \mid s)Q^\pi(s,a) + \pi_\theta(a \mid s)\nabla_\theta Q^\pi(s,a)) \\
=&\sum_{a\in\mathcal{A}}\left(\nabla_\theta\pi_\theta(a \mid s)Q^\pi(s,a) + \pi_\theta(a \mid s)\nabla_\theta\sum_{s',r}P\big(s',r \mid s,a\big)\big(r + V^\pi\big(s'\big)\big)\right) \\
=&\sum_{a\in\mathcal{A}}\left(\nabla_\theta\pi_\theta(a \mid s)Q^\pi(s,a) + \pi_\theta(a \mid s)\sum_{s',r}P\big(s',r \mid s,a\big)\nabla_\theta V^\pi\big(s'\big)\right) \\
=&\sum_{a\in\mathcal{A}}\left(\nabla_\theta\pi_\theta(a \mid s)Q^\pi(s,a) + \pi_\theta(a \mid s)\sum_{s'}P\big(s' \mid s,a\big)\nabla_\theta V^\pi\big(s'\big)\right)
\end{aligned}$$

where the second equality follows the **derivative product rule**, the third is from expanding Q^π with future state value, the fourth one uses the fact r is not a func of θ; and the last one uses $P(s' \mid s,a) = \sum_r P(s',r \mid s,a)$

- Now we have a recursive formula for policy gradients:

$$\nabla_\theta V^\pi(s) = \sum_{a\in\mathcal{A}}\left(\nabla_\theta\pi_\theta(a \mid s)Q^\pi(s,a) + \pi_\theta(a \mid s)\sum_{s'}P\big(s' \mid s,a\big)\nabla_\theta V^\pi\big(s'\big)\right)$$

In what follows, we will unroll the recursive formula.

- Let the probability of transitioning from state s to state x with policy π_θ after k step be denoted as $\rho^\pi(s\to x,k)$.

  For simplicity, we also let
  $$\phi(s) = \sum_{a\in\mathcal{A}}\nabla_\theta\pi_\theta(a \mid s)Q^\pi(s,a)$$
- Then we go back to unroll the recursive representation of $\nabla_\theta V^\pi(s)$ repeatedly:

$$
\begin{aligned}
&\nabla_\theta V^\pi(s) \\
=&\phi(s) + \sum_a \pi_\theta(a \mid s) \sum_{s'} P\big(s' \mid s, a\big) \nabla_\theta V^\pi\big(s'\big) \\
=&\phi(s) + \sum_{s'} \sum_a \pi_\theta(a \mid s) P\big(s' \mid s, a\big) \nabla_\theta V^\pi\big(s'\big) \\
=&\phi(s) + \sum_{s'} \rho^\pi\big(s \to s', 1\big) \nabla_\theta V^\pi\big(s'\big) \\
=&\phi(s) + \sum_{s'} \rho^\pi\big(s \to s', 1\big) \nabla_\theta V^\pi\big(s'\big) \\
=&\phi(s) + \sum_{s'} \rho^\pi\big(s \to s', 1\big) \left[\phi\big(s'\big) + \sum_{s''} \rho^\pi\big(s' \to s'', 1\big) \nabla_\theta V^\pi\big(s''\big)\right] \\
=&\phi(s) + \sum_{s'} \rho^\pi\big(s \to s', 1\big) \phi\big(s'\big) + \sum_{s''} \rho^\pi\big(s \to s'', 2\big) \nabla_\theta V^\pi\big(s''\big); \\
&\qquad\qquad \text{Use the Markov property } \sum_{s'} \rho^\pi\big(s \to s', 1\big) \sum_{s''} \rho^\pi\big(s' \to s'', 1\big) = \sum_{s''} \rho^\pi\big(s \to s'', 2\big) \\
=&\phi(s) + \sum_{s'} \rho^\pi\big(s \to s', 1\big) \phi\big(s'\big) + \sum_{s''} \rho^\pi\big(s \to s'', 2\big) \phi\big(s''\big) + \sum_{s'''} \rho^\pi\big(s \to s''', 3\big) \nabla_\theta V^\pi\big(s'''\big) \\
=&\ldots; \qquad\qquad \text{Repeatedly expanding into future time steps} \\
=&\sum_{x \in \mathcal{S}} \sum_{k=0}^\infty \rho^\pi(s \to x, k) \phi(x)
\end{aligned}
$$

- Define $\eta(s)$ as the **expected number of visits to state** s in a trajectory:

$$\eta(s) = \mathbb{E}^{\pi_\theta}\left[\sum_{k=0}^\infty 1(s_k = s) | s_0\right] = \sum_{k=0}^\infty \rho^\pi(s_0 \to s, k)$$

and define the visitation probability distribution as

$$d^\pi(s) = \frac{\eta(s)}{\sum_{s'} \eta(s')}$$

- Finally we have

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta V^\pi(s_0) \\
&= \sum_s \sum_{k=0}^\infty \rho^\pi(s_0 \to s, k) \phi(s) \\
&= \sum_{k=0}^\infty \sum_s \rho^\pi(s_0 \to s, k) \sum_a \nabla_\theta \pi_\theta(a \mid s) Q^\pi(s, a) \\
&= \sum_{k=0}^\infty \sum_s \sum_a \rho^\pi(s_0 \to s, k) \pi_\theta(a \mid s) \nabla_\theta \log \pi_\theta(a \mid s) Q^\pi(s, a)
\end{aligned}
$$

  where the last inequality applies the **log likelihood trick** as follows

$$\nabla_\theta \log \pi_\theta(a|s) = \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)}.$$

- Notice that $\rho^\pi(s_0 \to s, k) \pi_\theta(a \mid s)$ equals the probability of observing the state action pair (s, a) in k time steps following the policy π_θ. So we can rewrite the equation as

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \sum_{k=0}^\infty \mathbb{E}^{\pi_\theta}[\nabla_\theta \log \pi_\theta(a_k \mid s_k) Q^\pi(s_k, a_k)] \\
&= \mathbb{E}^{\pi_\theta}\left[\sum_{k=0}^\infty \nabla_\theta \log \pi_\theta(a_k \mid s_k) Q^\pi(s_k, a_k)\right]
\end{aligned}
$$

where the expectation is taken over a sequence of $s_0, a_0, s_1, a_1, \ldots$ of the MDP assuming that policy π_θ is being run.

(Note: in the previous analysis we didn't include discount factor.)

The policy gradient theorem has several forms:

- If the MDP always terminates in H time steps, we have
$$\nabla_\theta J(\theta) = \mathbb{E}^{\pi_\theta}\left[\sum_{k=0}^{H} \nabla_\theta \log \pi_\theta(a_k \mid s_k) Q^\pi(s_k, a_k)\right].$$
- If the MDP is undiscounted but has a stationary distribution d^π, we have
$$\nabla_\theta J(\theta) \propto \mathbb{E}[Q^\pi(s,a)\nabla_\theta \log \pi_\theta(a \mid s) | (s,a) \sim d^\pi]$$
- If there is a discount factor $\gamma < 1$, the policy gradient takes the form
$$\nabla_\theta J(\theta) = \mathbb{E}^{\pi_\theta}\left[\sum_{k=0}^{\infty} \gamma^k \nabla_\theta \log \pi_\theta(a_k \mid s_k) Q^\pi(s_k, a_k)\right].$$

denotes the visitation distribution of the MDP following policy π.

Remarks

- The analysis, although being quite long, is simply an application of the derivative product rule and properties of Markov chains/MDP.
- The PG Theorem expresses $\nabla_\theta J(\theta)$ as an expectation over trajectories $s_0, a_0, s_1, a_1, \ldots$ This makes it possible to evaluate such expectation via Monte Carlo sampling
- In the next lecture, we will discuss how to estimate PG and develop the first RL algorithm that iteratively refines policies and converges to the best one!