

# Quiz 4 Review: Understanding RL

[Chirayu Salgarkar]

Fall 2024

## Contents

<b>1</b>	<b>Policy Gradient</b>	<b>1</b>
1.1	Policy Objective functions . . . . .	2
1.2	Finite Difference Policy Gradient . . . . .	2
1.3	Monte-Carlo Policy Gradient . . . . .	2
1.4	One-step MDPs . . . . .	5
1.5	Actor-Critic Policy Gradient . . . . .	7
<b>2</b>	<b>Problems</b>	<b>8</b>

## 1 Policy Gradient

These are a popular class of practical algorithms to solve RL problems.

There are ways we've solved for action-value function before. We did approximation before, approximately the value function using parameters. Think function approximators. Now, we actually want to directly parametrize the policy. That is:

$$\pi_{\theta}(s, a) = \mathcal{P}[a|s, \theta]$$

We define a probability distribution that change the probabilities as we move to different states, allowing us to maximise reward. We care about model-free RL, and directly from experience, it can adjust its parameters of it's policy to maximize it's reward.

We care about this because it allows us to scale, where we have uncertain environment.

There are three major categories of parametrized policy. Two words: gradient descent. We follow the gradient in the direction that gets us to the most reward. We have value based and policy-based RL methods.

Are there advantages and disadvantages of policy-based vs value-based methods?

There are situations where it is more efficient to store policy as opposed to value function. Think atari games, where the learned value function may be complicated af. However, it's easier (compute-wise) to remember *leftgood*, or something similar. Policy can be more compact.

Policy also works because it converges better (some value-based methods can have oscillatory issues, for instance). If you directly follow policy, you're guaranteed to converge. They are also effective in high-dimensional or continuous action spaces. With value based methods, you need to find a max. That can be expensive. It can also learn stochastic policy.

Why would we ever want a stochastic policy? Consider the rock-paper-scissors analogy in class. If you just play this deterministically, you get exploited. If you play one choice often, your opponent will catch up. Similarly, if you have *partially observable environments*, as opposed to fully observable environments, the Markov Property may not hold. We only see certain features of the environment. In the Aliased Gridworld example, the agent can't differentiate the grey states. The two grey squares are aliased. They look basically the same. Your feature vector will be identical, which means that in a deterministic policy, you choose the *same action*! If you act greedily, you either go west all the time or east all the time! Which doesn't work. Stochastic policies work far better.

However, naive policy learning will typically converge to local as opposed to absolute minima. This is also inefficient, with high variance.

## 1.1 Policy Objective functions

What is the best  $\theta$  for policy  $\pi_\theta(s, a)$ ? First of all, how would you even measure that? In episodic environments, use the start value. That is, when I start the game at a start state, what policy ends with the best score?

This is represented as:

$$J_1(\theta) = V^{\pi_\theta}(s_1) = \mathbb{E}_{\pi_\theta}[V_1]$$

In continuing environments, use the average value. That is, consider the policy we are in any state times the value of all the states.

$$J_{\text{avgV}}(\theta) = \sum_s d^{\pi_\theta}(s)(V^{\pi_\theta}(s))$$

Or, we look at the average reward per time step. There is some probability i am in a state, there's some probability of a reward, and this is the immediate reward i get at each time step.

$$J_{\text{avgV}}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(s, a) \mathcal{R}_s^d$$

$d^{\pi_\theta}$  is the stationary distribution of the markov chain for  $\theta_0$ .

As you can tell, this is clearly an optimization problem. Some approaches don't use gradient, but greater efficiency is possible using gradient algorithms.

## 1.2 Finite Difference Policy Gradient

Policy gradient algorithms search for a local maximum in the policy objective function by ascending the gradient of the policy with respect to the parameters  $\theta$ .

$$\delta\theta = \alpha \nabla_\theta J(\theta)$$

Here,

$$\nabla_\theta J(\theta) = [(\frac{dJ(\theta)}{d\theta_1}) \dots (\frac{dJ(\theta)}{d\theta_\alpha})]^T$$

and  $\alpha$  is a step size parameter.

If you had no idea how to find the gradient, you can estimate by just perturbing  $\theta$ . Literally, the limit definition of the derivative:

$$\frac{dJ(\theta)}{d\theta_k} \approx \frac{J + \epsilon u_k - J(\theta)}{\epsilon}$$

where  $u_k$  is a unit vector with 1 in the  $k$ th component, and 0 elsewhere. It's simple, but sometimes effective, and works on nondifferentiable policies. It works on fast AIBO walk for RoboCup.

## 1.3 Monte-Carlo Policy Gradient

Now, we go to Monte-Carlo Policy Descent. We now compute the policy gradient analytically. We have some policy  $\pi_\theta$  which is differentiable when it is non-zero and we know the gradient  $\nabla_\theta \pi_\theta(s, a)$ . There are some likelihood ratios that basically show that:

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

This works because of calculus. We now have a *score function*  $\nabla_\theta \log \pi_\theta(s, a)$ . This allows us to take expectations. This is nice :).

**Derivation of the Score Function** We begin with the gradient of the policy  $\pi_\theta(s, a)$  with respect to the parameter vector  $\theta$ :

$$\nabla_\theta \pi_\theta(s, a)$$

We can express this gradient in terms of the log-probability using a simple property from calculus:

$$\nabla_\theta \pi_\theta(s, a) = \pi_\theta(s, a) \frac{\nabla_\theta \pi_\theta(s, a)}{\pi_\theta(s, a)}$$

Notice that  $\frac{\nabla_\theta \pi_\theta(s, a)}{\pi_\theta(s, a)}$  is just the gradient of the log-probability:

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

Thus, we can rewrite the original expression as:

$$\nabla_\theta \pi_\theta(s, a) = \pi_\theta(s, a) \nabla_\theta \log \pi_\theta(s, a)$$

This is a crucial result because it transforms the gradient of the probability  $\pi_\theta(s, a)$  into the probability itself multiplied by the gradient of the log-probability, which is known as the *score function*:

$$\nabla_\theta \log \pi_\theta(s, a)$$

**Why This is Useful** This result allows us to express expectations involving the gradient of the policy in a more tractable form. For example, for any function  $f(s, a)$ , we can take the expectation over actions under the policy  $\pi_\theta(s, a)$  as:

$$\mathbb{E}_{a \sim \pi_\theta(s)} [\nabla_\theta \log \pi_\theta(s, a) f(s, a)]$$

This is particularly useful in policy gradient methods, where we seek to optimize the expected return by adjusting  $\theta$ . The use of the score function simplifies the calculation of gradients for stochastic policies, as we can take expectations of the log-probabilities rather than directly computing derivatives of probabilities.

**Conclusion** By expressing the gradient in terms of the log-probability, we obtain a more convenient form that facilitates taking expectations over actions. This is a key technique in reinforcement learning algorithms like REINFORCE.

What does this look like?

**Softmax Policy with Linear Features** In a softmax policy, the probability of selecting an action  $a$  in a given state  $s$  is determined by the exponentiated linear combination of features. Let  $\phi(s, a)$  be the feature vector associated with the state-action pair  $(s, a)$ , and  $\theta$  be the parameter vector. The action probabilities are computed as follows:

$$\pi(a \mid s; \theta) = \frac{\exp(\phi(s, a)^T \theta)}{\sum_{a' \in A} \exp(\phi(s, a')^T \theta)}$$

where: -  $\pi(a \mid s; \theta)$  is the probability of taking action  $a$  given state  $s$  and parameter vector  $\theta$ . -  $\phi(s, a)$  is the feature vector for state  $s$  and action  $a$ . -  $\theta$  is the parameter vector to be learned. -  $A$  is the set of all possible actions.

The numerator,  $\exp(\phi(s, a)^T \theta)$ , represents the weight assigned to action  $a$  based on the features of the state-action pair  $(s, a)$  and the parameters  $\theta$ . The denominator normalizes these weights by summing over all possible actions  $a'$ .

**Score Function** The score function, which is the gradient of the log-probability of taking action  $a$  in state  $s$ , is given by:

$$\nabla_\theta \log \pi(a \mid s; \theta) = \phi(s, a) - \sum_{a' \in A} \pi(a' \mid s; \theta) \phi(s, a')$$

where: -  $\nabla_\theta \log \pi(a \mid s; \theta)$  is the gradient of the log-probability with respect to the parameters  $\theta$ . -  $\phi(s, a)$  is the feature vector for the selected action  $a$ . -  $\pi(a' \mid s; \theta)$  is the probability of selecting action  $a'$  under the policy. - The term  $\sum_{a' \in A} \pi(a' \mid s; \theta) \phi(s, a')$  is the expected value of the feature vector under the current policy, weighted by the action probabilities.

**Explanation** - The softmax policy ensures that actions with higher weights (i.e.,  $\phi(s, a)^T \theta$ ) have higher probabilities of being selected, while still assigning non-zero probabilities to all actions. - The score function is important for policy gradient methods, as it allows us to update the parameters  $\theta$  in a direction that improves the probability of good actions.

In continuous action spaces, you use Gaussian Policy.

**Gaussian Policy** In many reinforcement learning problems, especially those with continuous action spaces, the policy  $\pi_\theta(a | s)$  is modeled as a Gaussian distribution. For a state  $s$ , the policy outputs a mean  $\mu_\theta(s)$  and standard deviation  $\sigma_\theta(s)$ , parameterized by  $\theta$ . The action  $a$  is then drawn from a Gaussian distribution:

$$a \sim \mathcal{N}(\mu_\theta(s), \sigma_\theta(s)^2)$$

The probability density function (PDF) of a Gaussian policy can be written as:

$$\pi_\theta(a | s) = \frac{1}{\sqrt{2\pi\sigma_\theta(s)^2}} \exp\left(-\frac{(a - \mu_\theta(s))^2}{2\sigma_\theta(s)^2}\right)$$

where: -  $\mu_\theta(s)$  is the mean of the Gaussian distribution, which is a function of the state  $s$  and the parameters  $\theta$ . -  $\sigma_\theta(s)$  is the standard deviation of the Gaussian, which may also be parameterized by  $\theta$ . -  $a$  is the action sampled from this Gaussian distribution.

**Log-Probability of a Gaussian Policy** The log-probability of taking action  $a$  under the Gaussian policy is computed as follows:

$$\log \pi_\theta(a | s) = -\frac{1}{2} \log(2\pi\sigma_\theta(s)^2) - \frac{(a - \mu_\theta(s))^2}{2\sigma_\theta(s)^2}$$

This expression is the natural logarithm of the Gaussian PDF.

**Score Function for a Gaussian Policy** The score function is the gradient of the log-probability of taking action  $a$  with respect to the parameters  $\theta$ . Let's compute it for both the mean and the standard deviation.

1. **\*\*Gradient with respect to the mean\*\***  $\mu_\theta(s)$ :

$$\nabla_\theta \log \pi_\theta(a | s) = \frac{a - \mu_\theta(s)}{\sigma_\theta(s)^2} \nabla_\theta \mu_\theta(s)$$

This shows that the score function with respect to the mean is proportional to the error between the action  $a$  and the mean  $\mu_\theta(s)$ , weighted by the variance  $\sigma_\theta(s)^2$ .

2. **\*\*Gradient with respect to the standard deviation\*\***  $\sigma_\theta(s)$ :

$$\nabla_\theta \log \pi_\theta(a | s) = \left( \frac{(a - \mu_\theta(s))^2}{\sigma_\theta(s)^3} - \frac{1}{\sigma_\theta(s)} \right) \nabla_\theta \sigma_\theta(s)$$

This shows that the score function with respect to the standard deviation takes into account how far the action  $a$  is from the mean, normalized by the variance.

**Summary** For a Gaussian policy, the score function allows us to compute the gradient of the log-probability of selecting an action with respect to the policy parameters. Specifically:

- The gradient with respect to the mean  $\mu_\theta(s)$  is proportional to the difference between the action and the mean, scaled by the variance.
- The gradient with respect to the standard deviation  $\sigma_\theta(s)$  includes terms that capture the variability of the actions with respect to the mean.

These gradients are useful in policy gradient algorithms where the goal is to optimize the parameters  $\theta$  to improve the expected return.

## 1.4 One-step MDPs

**One-Step MDPs** A one-step MDP is a simplified version of a Markov Decision Process (MDP) in which the agent makes a single decision, receives a reward, and then terminates. This setup is useful for illustrating key concepts in reinforcement learning, as it eliminates the complexity of multi-step transitions.

In a one-step MDP, given a state  $s$ , the agent selects an action  $a$  according to a policy  $\pi(a | s)$ , receives an immediate reward  $R(s, a)$ , and then transitions to the terminal state. The value of taking action  $a$  in state  $s$  under policy  $\pi$ , denoted as  $Q^\pi(s, a)$ , is simply the expected reward:

$$Q^\pi(s, a) = \mathbb{E}[R(s, a)]$$

The state-value function  $V^\pi(s)$ , which represents the expected reward for the agent starting in state  $s$  and following policy  $\pi$ , is the expectation over all possible actions:

$$V^\pi(s) = \sum_a \pi(a | s) Q^\pi(s, a)$$

Substituting  $Q^\pi(s, a)$ , we have:

$$V^\pi(s) = \sum_a \pi(a | s) \mathbb{E}[R(s, a)]$$

### Policy Optimization in One-Step MDPs

**One-Step MDPs** A one-step Markov Decision Process (MDP) simplifies the typical MDP structure to a single decision step. The agent selects an action  $a$  from a state  $s$ , receives an immediate reward  $R(s, a)$ , and the process terminates. This simplified structure helps illustrate key reinforcement learning concepts.

For a one-step MDP, the objective is to maximize the expected reward  $J(\theta)$ , where the policy is parameterized by  $\theta$ . The value of taking action  $a$  in state  $s$ , denoted by  $Q^\pi(s, a)$ , is simply the expected reward for that action:

$$Q^\pi(s, a) = \mathbb{E}[R(s, a)]$$

The value of the state under policy  $\pi$ , denoted as  $V^\pi(s)$ , is the expected reward over all possible actions, weighted by the policy's action probabilities:

$$V^\pi(s) = \sum_a \pi(a | s) Q^\pi(s, a)$$

Substituting  $Q^\pi(s, a) = \mathbb{E}[R(s, a)]$ , we have:

$$V^\pi(s) = \sum_a \pi(a | s) \mathbb{E}[R(s, a)]$$

**Objective Function  $J(\theta)$  in One-Step MDPs** The goal is to optimize the policy  $\pi_\theta(a | s)$  to maximize the expected reward. This objective is captured by the function  $J(\theta)$ , which is defined as the expected reward under the current policy:

$$J(\theta) = \mathbb{E}_{a \sim \pi_\theta} [R(s, a)]$$

To optimize  $J(\theta)$ , we compute the gradient with respect to  $\theta$ . Using the score function  $\nabla_\theta \log \pi_\theta(a | s)$ , we derive the policy gradient as follows:

$$\nabla_\theta J(\theta) = \mathbb{E}_{a \sim \pi_\theta} [\nabla_\theta \log \pi_\theta(a | s) R(s, a)]$$

This formulation allows us to adjust  $\theta$  to improve the expected reward by updating the policy in the direction of the gradient.

To now do this for multistep MDPS, chance the instantaneous reward with the long-term value function.

**Multi-Step MDPs** In a multi-step Markov Decision Process (MDP), the agent interacts with the environment over several time steps. At each time step  $t$ , the agent takes an action  $a_t$  in state  $s_t$ , receives a reward  $R(s_t, a_t)$ , and transitions to a new state  $s_{t+1}$ . The goal is to maximize the expected long-term return, which is the cumulative sum of discounted rewards:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R(s_{t+k}, a_{t+k}),$$

where  $\gamma \in [0, 1)$  is the discount factor that determines how future rewards are weighted relative to immediate rewards.

The state-value function  $V^\pi(s)$  represents the expected return starting from state  $s$  and following policy  $\pi$ :

$$V^\pi(s) = \mathbb{E}_\pi[G_t \mid s_t = s].$$

Similarly, the action-value function  $Q^\pi(s, a)$  represents the expected return after taking action  $a$  in state  $s$  and following policy  $\pi$ :

$$Q^\pi(s, a) = \mathbb{E}_\pi[G_t \mid s_t = s, a_t = a].$$

**Policy Objective  $J(\theta)$**  In multi-step MDPs, the objective is to optimize the policy  $\pi_\theta(a \mid s)$  to maximize the expected long-term return. The objective function  $J(\theta)$  is defined as the expected return:

$$J(\theta) = \mathbb{E}_{s \sim \rho^\pi, a \sim \pi_\theta} [G_t],$$

where  $\rho^\pi(s)$  is the state distribution under policy  $\pi$ .

**Theorem 1** (Policy Gradient Theorem). *The gradient of the objective function  $J(\theta)$  with respect to the policy parameters  $\theta$  is given by:*

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim \rho^\pi, a \sim \pi_\theta} [\nabla_\theta \log \pi_\theta(a \mid s) Q^\pi(s, a)].$$

*This theorem states that the gradient of the expected return is proportional to the expected gradient of the log-probability of the policy, weighted by the action-value function  $Q^\pi(s, a)$ . In practice,  $Q^\pi(s, a)$  may be approximated using either real rewards or learned estimates of the action-value function.*

Using the score function  $\nabla_\theta \log \pi_\theta(a \mid s)$ , we can express the policy gradient as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim \rho^\pi, a \sim \pi_\theta} [\nabla_\theta \log \pi_\theta(a \mid s) G_t].$$

This form of the gradient is the foundation of policy gradient methods in reinforcement learning, allowing us to adjust the policy parameters  $\theta$  in the direction that maximizes the expected return.

**Monte Carlo Policy Gradient (REINFORCE)** The policy gradient theorem forms the basis for the Monte Carlo Policy Gradient, also known as the REINFORCE algorithm. In this algorithm, the parameters of the policy are updated by stochastic gradient ascent using samples from the environment.

From the policy gradient theorem, the gradient of the objective function  $J(\theta)$  is given by:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim \rho^\pi, a \sim \pi_\theta} [\nabla_\theta \log \pi_\theta(a \mid s) Q^\pi(s, a)].$$

Since the true action-value function  $Q^\pi(s_t, a_t)$  is typically unknown, we use the return  $G_t$  (the cumulative reward) as an unbiased sample of  $Q^\pi(s_t, a_t)$ . This is the foundation of the REINFORCE algorithm.

At each time step  $t$ , the agent observes a state  $s_t$ , takes an action  $a_t$ , and receives a reward. After completing an episode, the return  $G_t$  from each state-action pair can be computed as:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R(s_{t+k}, a_{t+k}),$$

where  $\gamma$  is the discount factor. The return  $G_t$  serves as an unbiased estimate of  $Q^\pi(s_t, a_t)$ , allowing us to compute the policy gradient for each episode.

**Stochastic Gradient Ascent** Using  $G_t$  as a sample of the action-value function, the gradient of the policy can be approximated as:

$$\nabla_{\theta} J(\theta) \approx \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) G_t,$$

where  $T$  is the length of the episode. The parameters  $\theta$  of the policy are updated using stochastic gradient ascent as follows:

$$\theta \leftarrow \theta + \alpha \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) G_t,$$

where  $\alpha$  is the learning rate. This update rule incrementally improves the policy by adjusting the parameters  $\theta$  in the direction that increases the expected return.

**Summary** The REINFORCE algorithm updates the policy parameters by performing stochastic gradient ascent using the sampled returns  $G_t$ . Since  $G_t$  is an unbiased estimate of  $Q^{\pi}(s_t, a_t)$ , this method optimizes the policy over time without requiring a model of the environment.

## 1.5 Actor-Critic Policy Gradient

However, Monte-Carlo policy gradients still have high variance. Now, we use a *critic* to estimate the action-value function, which updates actional value function parameters  $w$  which updates policy parameters  $\theta$  in direction suggested by the critic. This follows an approximate policy gradient.

**Reducing Variance with a Critic** While the Monte Carlo policy gradient (REINFORCE) algorithm provides a straightforward approach to policy optimization, it suffers from high variance due to the stochastic nature of the returns  $G_t$ . High variance can lead to unstable learning and slow convergence.

To mitigate this issue, we introduce a *critic*, which is a value function estimator that approximates the action-value function  $Q^{\pi}(s, a)$ . The critic uses its own parameters  $w$  to predict the expected return, allowing for more stable and lower-variance updates to the policy.

The critic is trained to minimize the mean squared error between its predictions and the actual returns:

$$L(w) = \frac{1}{2} \mathbb{E} \left[ (Q(s_t, a_t; w) - G_t)^2 \right],$$

where  $Q(s_t, a_t; w)$  is the estimated action-value function given state  $s_t$  and action  $a_t$ , and  $G_t$  is the return observed after taking action  $a_t$ .

**Approximate Policy Gradient** Once the critic provides an estimate of the action-value function, we can use this estimate to update the policy parameters  $\theta$ . The approximate policy gradient can be expressed as:

$$\nabla_{\theta} J(\theta) \approx \mathbb{E} [\nabla_{\theta} \log \pi_{\theta}(a_t | s_t) Q(s_t, a_t; w)].$$

In this formulation,  $Q(s_t, a_t; w)$  serves as a more stable estimate of the action-value function compared to using the return  $G_t$  directly. The parameters of the policy are then updated using:

$$\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta),$$

where  $\alpha$  is the learning rate.

By leveraging the critic, we perform policy updates that are more reliable and less susceptible to variance, leading to faster and more stable learning in reinforcement learning tasks.

**Summary** Incorporating a critic into the policy gradient framework allows us to reduce variance and improve the stability of policy updates. By estimating the action-value function using a separate set of parameters  $w$ , we achieve more reliable updates to the policy parameters  $\theta$ , following an approximate policy gradient approach. This methodology is a fundamental aspect of actor-critic algorithms in reinforcement learning.

## 2 Problems

Given the action preferences  $\{p_1, p_2, p_3\} = \{1.2, 0.5, 0.1\}$ , the probability of selecting each action using the softmax policy is calculated as follows:

The softmax policy assigns the probability for action  $i$  as:

$$\pi(a_i) = \frac{e^{p_i}}{\sum_{j=1}^3 e^{p_j}}$$

For the given preferences:

$$\pi(a_1) = \frac{e^{1.2}}{e^{1.2} + e^{0.5} + e^{0.1}} = \frac{e^{1.2}}{e^{1.2} + e^{0.5} + e^{0.1}}$$

$$\pi(a_2) = \frac{e^{0.5}}{e^{1.2} + e^{0.5} + e^{0.1}}$$

$$\pi(a_3) = \frac{e^{0.1}}{e^{1.2} + e^{0.5} + e^{0.1}}$$

Now calculating the values of  $e^{p_i}$ :

$$e^{1.2} \approx 3.3201, \quad e^{0.5} \approx 1.6487, \quad e^{0.1} \approx 1.1052$$

The denominator becomes:

$$3.3201 + 1.6487 + 1.1052 = 6.074$$

Thus, the probabilities are:

$$\pi(a_1) \approx \frac{3.3201}{6.074} \approx 0.5466$$

$$\pi(a_2) \approx \frac{1.6487}{6.074} \approx 0.2714$$

$$\pi(a_3) \approx \frac{1.1052}{6.074} \approx 0.1820$$

Therefore, the probabilities of selecting actions  $a_1$ ,  $a_2$ , and  $a_3$  are approximately 0.5466, 0.2714, and 0.1820, respectively.

### Problem 1: Policy Gradient Estimation

Given the following policy  $\pi_\theta(s, a)$  for state  $s$  and action  $a$ :

$$\pi_\theta(s, a) = \frac{e^{\phi(s, a)^T \theta}}{\sum_{a' \in A} e^{\phi(s, a')^T \theta}},$$

where  $\phi(s, a)$  is the feature vector for state-action pair  $(s, a)$ , and  $\theta$  is the parameter vector.

Given:

$$\phi(s, a_1) = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \quad \phi(s, a_2) = \begin{bmatrix} 0.5 \\ 1 \end{bmatrix}, \quad \theta = \begin{bmatrix} 0.3 \\ 0.7 \end{bmatrix},$$

compute the probability of taking action  $a_1$  in state  $s$ , i.e.,  $\pi_\theta(s, a_1)$ .



## Solution

First, calculate  $\phi(s, a_1)^T \theta$  and  $\phi(s, a_2)^T \theta$ :

$$\phi(s, a_1)^T \theta = 1 \cdot 0.3 + 2 \cdot 0.7 = 1.7,$$

$$\phi(s, a_2)^T \theta = 0.5 \cdot 0.3 + 1 \cdot 0.7 = 0.85.$$

Next, compute the action probabilities:

$$\pi_\theta(s, a_1) = \frac{e^{1.7}}{e^{1.7} + e^{0.85}} = \frac{e^{1.7}}{e^{1.7} + e^{0.85}}.$$

Using a calculator:

$$\pi_\theta(s, a_1) = \frac{5.4739}{5.4739 + 2.3396} = \frac{5.4739}{7.8135} \approx 0.7005.$$

Thus,  $\pi_\theta(s, a_1) \approx 0.7005$ .

## Problem 2: Gaussian Policy Log-Probability

In a Gaussian policy, the action  $a$  is sampled from a distribution  $\mathcal{N}(\mu_\theta(s), \sigma_\theta(s)^2)$ .

Given:

$$\mu_\theta(s) = 2, \quad \sigma_\theta(s) = 0.5, \quad a = 3,$$

compute the log-probability  $\log \pi_\theta(a|s)$ .

## Solution

The log-probability for a Gaussian distribution is given by:

$$\log \pi_\theta(a|s) = -\frac{1}{2} \log(2\pi\sigma_\theta(s)^2) - \frac{(a - \mu_\theta(s))^2}{2\sigma_\theta(s)^2}.$$

Substitute the given values:

$$\log \pi_\theta(a|s) = -\frac{1}{2} \log(2\pi(0.5)^2) - \frac{(3 - 2)^2}{2(0.5)^2}.$$

Calculate each term:

$$-\frac{1}{2} \log(2\pi(0.25)) = -\frac{1}{2} \log(1.5708) \approx -0.2857,$$

$$\frac{(3 - 2)^2}{2(0.5)^2} = \frac{1}{2 \cdot 0.25} = 2.$$

Thus, the log-probability is:

$$\log \pi_\theta(a|s) \approx -0.2857 - 2 = -2.2857.$$

## Problem 3: Expected Reward in One-Step MDP

In a one-step MDP, the agent takes an action  $a$ , receives a reward  $R(s, a)$ , and terminates. Given the policy:

$$\pi(a|s) = \begin{cases} 0.6 & \text{if } a = a_1, \\ 0.4 & \text{if } a = a_2, \end{cases}$$

and the rewards:

$$R(s, a_1) = 10, \quad R(s, a_2) = 5,$$

compute the expected reward  $V^\pi(s)$ .

## Solution

The expected reward is:

$$V^\pi(s) = \sum_a \pi(a|s)R(s, a).$$

Substitute the values:

$$V^\pi(s) = 0.6 \cdot 10 + 0.4 \cdot 5 = 6 + 2 = 8.$$

Thus, the expected reward  $V^\pi(s) = 8$ .

## Problem 4: Gradient with Respect to Policy Parameters

Consider a softmax policy with two actions. The probabilities of actions  $a_1$  and  $a_2$  are given by:

$$\pi(a_1|s; \theta) = \frac{e^{\theta_1}}{e^{\theta_1} + e^{\theta_2}}, \quad \pi(a_2|s; \theta) = \frac{e^{\theta_2}}{e^{\theta_1} + e^{\theta_2}}.$$

Compute the gradient  $\nabla_{\theta_1} \log \pi(a_1|s; \theta)$ .

## Solution

The gradient of the log-probability is given by:

$$\nabla_{\theta_1} \log \pi(a_1|s; \theta) = 1 - \pi(a_1|s; \theta).$$

Substitute  $\pi(a_1|s; \theta)$ :

$$\nabla_{\theta_1} \log \pi(a_1|s; \theta) = 1 - \frac{e^{\theta_1}}{e^{\theta_1} + e^{\theta_2}} = \frac{e^{\theta_2}}{e^{\theta_1} + e^{\theta_2}}.$$

Thus,  $\nabla_{\theta_1} \log \pi(a_1|s; \theta) = \pi(a_2|s; \theta)$ .

## Problem 5: Monte-Carlo Policy Gradient

In a Monte-Carlo policy gradient method, the return  $G_t$  at time step  $t$  is the cumulative discounted reward:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R(s_{t+k}, a_{t+k}),$$

where  $\gamma = 0.9$ . Given rewards  $R(s_t, a_t) = 5$ ,  $R(s_{t+1}, a_{t+1}) = 3$ ,  $R(s_{t+2}, a_{t+2}) = 2$ , compute  $G_t$ .

## Solution

Compute  $G_t$  using the given rewards:

$$G_t = 5 + 0.9 \cdot 3 + 0.9^2 \cdot 2 = 5 + 2.7 + 1.62 = 9.32.$$

Thus,  $G_t = 9.32$ .