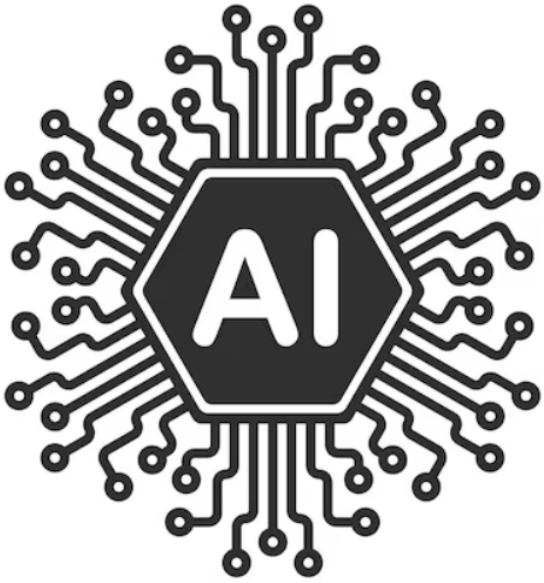

Deep Reinforcement Learning

Professor Mohammad Hossein Rohban

Recitation 3:

Policy-Based Methods

Designed By:

SeyyedAli MirGhasemi
sam717269@gmail.com

Spring 2025

1 Types of Reinforcement Learning Methods

Reinforcement Learning (RL) can be broadly classified into two main categories: *Model-Based* and *Model-Free* methods. These methods differ in how they approach the problem of decision making and learning in environments with uncertainty.

1.1 Model-Based Reinforcement Learning

In *Model-Based* RL, the agent attempts to learn or is provided with a model of the environment. This model represents the environment's dynamics, including the transition function (how the environment reacts to actions) and the reward function (the reward the agent receives for an action in a state). The agent can plan ahead by simulating possible future trajectories using this model.

Examples of Model-Based RL Methods:

- Dyna-Style Methods
- Model Predictive Control (MPC)

Advantages of Model-Based RL:

- Uses planning to reduce interaction with the environment.
- Converges faster when a reliable model is available.

Disadvantages of Model-Based RL:

- Requires an accurate model, which may be hard to obtain.
- Planning can be computationally expensive.

1.2 Model-Free Reinforcement Learning

In *Model-Free* RL, the agent does not learn a model of the environment. Instead, it directly learns the value function or policy through interaction with the environment. This is done by observing the outcomes of actions taken in various states and using these experiences to improve the agent's decisions.

1.2.1 Value-Based Methods

Value-Based methods focus on learning the value function, which estimates the expected return (or cumulative reward) from a given state or state-action pair. The agent typically learns this value function and selects actions based on it to maximize expected return.

Examples of Value-Based RL Methods:

- Q-Learning A
- SARSA
- DQN

Advantages of Value-Based Methods:

- Simple to implement and effective when the action space is discrete.
- Can work with both discrete and continuous state spaces.

Disadvantages of Value-Based Methods:

- Only applicable to discrete action spaces.
- May struggle in environments with complex dynamics or high-dimensional states.

1.2.2 Policy-Based Methods

In *Policy-Based* methods, the agent learns a policy directly, which is a mapping from states to actions. The goal is to optimize the policy in a way that maximizes expected return, rather than estimating values for state-action pairs.

Examples of Policy-Based RL Methods:

- REINFORCE
- Proximal Policy Optimization (PPO)
- Trust Region Policy Optimization (TRPO)

Advantages of Policy-Based Methods:

- Can handle continuous action spaces.
- Suitable for learning complex policies, especially in high-dimensional environments.

Disadvantages of Policy-Based Methods:

- Requires a large amount of data for effective training.
- Computationally expensive due to gradient estimation and optimization.

1.2.3 Actor-Critic Methods

Actor-Critic methods combine both value-based and policy-based approaches. These methods maintain two components:

- **Actor**: The policy function, which is responsible for deciding which action to take given the current state.
- **Critic**: The value function, which evaluates the actions taken by the actor based on their expected return (e.g., using a value function or action-value function).

In these methods, the critic evaluates the actions taken by the actor and provides feedback in the form of value estimates. The actor then updates its policy to take actions that are expected to lead to higher rewards, as guided by the critic's feedback.

Examples of Actor-Critic Methods:

- A2C (Advantage Actor-Critic)
- A3C (Asynchronous Advantage Actor-Critic)
- DDPG (Deep Deterministic Policy Gradient)

Advantages of Actor-Critic Methods:

- Combines value-based and policy-based approaches.
- Works well in continuous action spaces.
- Uses value-based feedback to improve policy stability.

Disadvantages of Actor-Critic Methods:

- More complex to implement and tune.
- Requires careful balancing between actor and critic to ensure stability.

1.3 Summary of RL Methods

- **Model-Based RL**: Learns or uses a model of the environment to plan and optimize actions.
- **Model-Free RL**:
  - **Value-Based Methods**: Learn a value function to guide action selection (e.g., Q-Learning, SARSA, DQN).
  - **Policy-Based Methods**: Learn a direct policy that maps states to actions (e.g., REINFORCE, PPO, TRPO).
  - **Actor-Critic Methods**: Combine both value-based and policy-based approaches (e.g., A2C, A3C, DDPG).

2 Policy-Based Methods

Policy-based methods are a class of reinforcement learning (RL) algorithms where the goal is to directly learn a policy that maximizes cumulative rewards, as opposed to learning a value function. These methods are advantageous in environments with large or continuous action spaces, where value-based methods (such as Q-learning) struggle.

2.1 Policy Search: Evolutionary Methods vs. Policy Gradient

Policy search methods aim to find an optimal policy by directly optimizing its parameters. There are two main approaches: evolutionary methods, such as Genetic Algorithms (GA), and gradient-based methods, such as Policy Gradient.

2.1.1 Evolutionary Methods (e.g., Genetic Algorithm)

Evolutionary methods optimize policy parameters by searching through a population of candidate policies. These methods do not rely on gradient information but instead evolve policies through selection, mutation, and crossover.

- A population of policies is initialized randomly.
- Each policy is evaluated based on its performance (fitness).
- The best-performing policies are selected and modified through mutation and crossover to create a new generation.
- This process continues for multiple generations, gradually improving the policy.

Since evolutionary methods do not use gradients, they are useful for optimizing policies in environments where the objective function is not differentiable or well-defined.

2.1.2 Policy Gradient Methods

Policy Gradient methods take a different approach by defining an objective function, which is the expected return, and optimizing it using gradient ascent. Instead of searching randomly, these methods directly adjust the policy parameters in the direction that increases the expected reward.

- The policy is parameterized (e.g., as a neural network).
- The expected return is used as an objective function.
- The gradient of this objective is computed with respect to the policy parameters.
- The parameters are updated using gradient ascent to maximize expected return.

Policy Gradient methods are efficient when the objective function is differentiable and provide a more structured way of reaching optimal policies compared to evolutionary methods.

2.1.3 Key Differences

- **Optimization Approach:** Evolutionary methods search for optimal policies using a population-based approach, while Policy Gradient methods optimize a defined objective function using gradients.A Genetic Algorithm (GA) evolves multiple walking policies by selecting, mutating, and recombining them. In contrast, a Policy Gradient method like REINFORCE continuously updates a single policy by computing gradients based on observed rewards.
- **Use of Gradients:** Evolutionary methods do not require gradients, making them useful in non-differentiable RL environments, whereas Policy Gradient methods rely on computing gradients to optimize the policy.
- **Convergence Behavior:** Policy Gradient methods typically converge more smoothly as they directly optimize an objective, whereas evolutionary methods rely on population-based exploration, which can lead to slower convergence.
- **Exploration vs. Exploitation:** Evolutionary methods naturally encourage more exploration by maintaining a diverse population of policies, while Policy Gradient methods refine a single policy through experience.

Both methods have advantages and are chosen based on the problem requirements. Evolutionary methods are useful for complex RL problems where gradient information is unavailable or unreliable, while Policy Gradient methods are more structured and efficient when gradients can be computed.

2.1.4 Sudo Code Example for GA-Based Policy Search

Below is an example pseudocode for a genetic algorithm applied to policy search:

Algorithm 1 Genetic Algorithm for Policy Optimization

```
Initialize a population of random policies:
  population = initialize_population(population_size)
for generation = 1 to num_generations do
    Evaluate fitness of each policy:
    for policy in population do
       fitness = evaluate_policy(policy)
       store fitness score
    end for
    Select top-performing policies:
      selected_policies = select_top_policies(population, fitness_scores)
    Generate next generation:
      next_generation = []
    while len(next_generation) < population_size do
        parent1, parent2 = select_parents(selected_policies)
        child1, child2 = crossover(parent1, parent2)
        child1 = mutate(child1)
        child2 = mutate(child2)
        next_generation.append(child1, child2)
    end while
    population = next_generation
end for
Return best-performing policy:
  best_policy = select_best_policy(population, fitness_scores)
```

3 Biased vs. Unbiased Estimation

The biased formula for the sample variance S^2 is given by:

$$S^2_{\text{biased}} = \frac{1}{n}\sum_{i=1}^{n}(X_i - \overline{X})^2$$

This is an underestimation of the true population variance σ^2 because it does not account for the degrees of freedom in estimation. Instead, the unbiased estimator is:

$$S^2_{\text{unbiased}} = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \overline{X})^2.$$

To see why the biased version underestimates the true variance, recall the expectation:

$$\mathbb{E}[S^2_{\text{biased}}] = \left(1 - \frac{1}{n}\right)\sigma^2 < \sigma^2.$$

This shows that the biased estimator systematically underestimates the true variance because of the factor $\left(1 - \frac{1}{n}\right)$.

To continue, we note that by subtracting μ from both sides of

$$\overline{X} = \frac{1}{n}\sum_{i=1}^{n} X_i,$$

we get

$$\overline{X} - \mu = \frac{1}{n}\sum_{i=1}^{n} X_i - \mu = \frac{1}{n}\sum_{i=1}^{n} X_i - \frac{1}{n}\sum_{i=1}^{n}\mu = \frac{1}{n}\sum_{i=1}^{n}(X_i - \mu).$$

Meaning, by cross-multiplication:

$$n \cdot (\overline{X} - \mu) = \sum_{i=1}^{n}(X_i - \mu).$$

Then, the previous expression becomes:

$$\mathbb{E}[S^2_{\text{biased}}] = \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}(X_i - \mu)^2 - \frac{2}{n}(\overline{X} - \mu)\sum_{i=1}^{n}(X_i - \mu) + (\overline{X} - \mu)^2\right].$$

Rewriting:

$$= \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}(X_i - \mu)^2 - \frac{2}{n}(\overline{X} - \mu)\cdot n \cdot (\overline{X} - \mu) + (\overline{X} - \mu)^2\right].$$

$$= \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}(X_i - \mu)^2 - 2(\overline{X} - \mu)^2 + (\overline{X} - \mu)^2\right].$$

$$= \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}(X_i - \mu)^2 - (\overline{X} - \mu)^2\right].$$

$$= \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}(X_i - \mu)^2\right] - \mathbb{E}\left[(\overline{X} - \mu)^2\right].$$

Since

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}(X_i - \mu)^2\right] = \sigma^2$$

and

$$\mathbb{E}\left[(\overline{X} - \mu)^2\right] = \frac{\sigma^2}{n},$$

we get:

$$\sigma^2 - \mathbb{E}\left[(\overline{X} - \mu)^2\right] = \left(1 - \frac{1}{n}\right)\sigma^2 < \sigma^2.$$

$$\mathbb{E}[S^2_{\text{unbiased}}] = \mathbb{E}\left[\frac{1}{n-1}\sum_{i=1}^{n}(X_i - \overline{X})^2\right] = \frac{n}{n-1}\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}(X_i - \overline{X})^2\right]$$

$$= \frac{n}{n-1}\left(1 - \frac{1}{n}\right)\sigma^2 = \sigma^2,$$

4 Variance of Estimation

Variance in estimation quantifies how much an estimator's output fluctuates across different samples from the same population. For an estimator $\hat{\theta}$ of a parameter θ, the variance is defined as:

$$\text{Var}(\hat{\theta}) = E\left[(\hat{\theta} - E[\hat{\theta}])^2\right]$$

Where:

- $\hat{\theta}$ is the estimator of the parameter θ,
- $E[\hat{\theta}]$ is the expected value of the estimator.

The variance measures the "spread" of the estimator's values around its expected value. A high variance means the estimator's results are less consistent, while a low variance implies more stable and reliable estimates.

4.1 Example: Variance in Sample Mean Estimation

Let's consider estimating the population mean μ using the sample mean $\bar{X}$:

$$\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$$

The variance of the sample mean $\bar{X}$ is given by:

$$\text{Var}(\bar{X}) = \frac{\sigma^2}{n}$$

Where:

- σ^2 is the population variance,
- n is the sample size.

Key Observations

- **For small** n: The variance is large, causing the estimate to fluctuate more.
- **For large** n: The variance is small, leading to a more stable and reliable estimate.

5 Monte Carlo Estimators in Reinforcement Learning

A Monte Carlo estimator is a method used to approximate the expected value of a function $f(X)$ over a random variable X with a given probability distribution $p(X)$. The true expectation is:

$$E[f(X)] = \int f(x)p(x)\,dx$$

However, directly computing this integral may be complex. Instead, we use Monte Carlo estimation by drawing N independent samples $X_1, X_2, \ldots, X_N$ from $p(X)$ and computing:

$$\hat{\mu}_{MC} = \frac{1}{N}\sum_{i=1}^{N} f(X_i)$$

This estimator provides an approximation to the true expectation $E[f(X)]$.

where $X_1, X_2, \ldots, X_n$ are independent and identically distributed (i.i.d.) samples. By the law of large numbers (LLN), as $n \to \infty$, we have:

$$\hat{X}_n \to \mathbb{E}[X] \quad \text{(almost surely)}$$

This method is commonly used in RL to estimate expected rewards, state-value functions, and action-value functions.

6 Policy Gradient in Reinforcement Learning

Policy gradient methods directly optimize the policy $\pi_\theta(a|s)$ by adjusting the parameters θ using gradient ascent on expected cumulative rewards:

$$J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[R(\tau)\right]$$

where $\tau = (s_0, a_0, s_1, a_1, \ldots)$ represents a trajectory following policy π_θ, and $R(\tau)$ is the cumulative reward. Using the log likelihood ratio trick, the policy gradient theorem gives:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R_t\right]$$

This gradient can be estimated using Monte Carlo methods by sampling multiple trajectories.

7 Unbiased Estimation and High Variance in Policy Gradient

Monte Carlo estimators of policy gradients are **unbiased**, meaning:

$$\mathbb{E}\left[\nabla_\theta J(\theta)\right] = \nabla_\theta J(\theta)$$

However, they often suffer from **high variance**, which makes learning unstable. The variance of the Monte Carlo estimate can be large due to randomness in the trajectory sampling process. To reduce variance, we commonly use:

- **Baseline**: Subtracting a baseline function $b(s)$, which does not affect the expectation but reduces variance:

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)(R_t - b(s_t))\right]$$

- **Actor-Critic Methods**: Using a learned value function $V(s)$ as a baseline.

- **Variance Reduction Techniques**: Techniques like reward normalization, advantage estimation, and bootstrapping help stabilize learning.

Despite these techniques, policy gradient methods remain sensitive to high variance, necessitating careful tuning and large batch sizes.

8 REINFORCE Algorithm

Algorithm 2 REINFORCE: Monte-Carlo Policy-Gradient Control (Episodic)

Input: Differentiable policy parameterization $\pi(a|s, \theta)$
Algorithm parameter: Step size $\alpha > 0$
Initialize: Policy parameters $\theta \in \mathbb{R}^d$ (e.g., to 0)
Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_T, A_{T-1}, R_T$ following $\pi(\cdot|\cdot, \theta)$
for each step $t = 0, 1, \ldots, T-1$ **do**
    Compute return:

$$G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$$

    Update policy parameters:

$$\theta \leftarrow \theta + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t, \theta)$$

end for

9 Using REINFORCE in Continuous Action Spaces

The REINFORCE algorithm is a policy gradient method that can be applied to continuous action spaces by parameterizing the policy as a continuous distribution over actions, rather than using discrete action choices. In continuous action spaces, the policy is typically represented as a probability distribution, such as a Gaussian distribution, where the mean and variance of the distribution are learned through the parameters θ.

9.1 Policy Representation

Let the policy $\pi_\theta(a_t|s_t)$ be represented by a parametric family of probability distributions. In the case of a continuous action space, one common choice is a Gaussian distribution:

$$\pi_\theta(a_t|s_t) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(a_t - \mu(s_t))^2}{2\sigma^2}\right)$$

where:

- $\mu(s_t)$ is the mean of the action distribution, which is a function of the state s_t parameterized by θ.

- σ^2 is the variance of the action distribution, which may also be a function of the state or treated as a fixed constant.

policy outputs a distribution over actions, and the agent samples actions a_t from this distribution during each step.

9.2 Gradient Estimation

For continuous action spaces, the gradient of the objective function (expected return) with respect to the policy parameters θ is given by:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} G_t \nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$

where:

- $\tau = (s_0, a_0, s_1, a_1, \dots)$ is a trajectory sampled from the policy π_θ.

- $G_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$ is the return (cumulative discounted reward) from time step t onward.

- $\log \pi_\theta(a_t|s_t)$ is the log probability of selecting action a_t given state s_t under policy π_θ.

The policy parameters θ are updated using gradient ascent:

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

For a Gaussian policy, where the action distribution is parameterized as:

$$\pi_\theta(a_t|s_t) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(a_t - \mu_\theta(s_t))^2}{2\sigma^2}\right),$$

the log-probability is:

$$\log \pi_\theta(a_t|s_t) = -\frac{1}{2}\log(2\pi\sigma^2) - \frac{(a_t - \mu_\theta(s_t))^2}{2\sigma^2}.$$

Taking the gradient of the log-probability with respect to θ results in:

$$\nabla_\theta \log \pi_\theta(a_t|s_t) = \frac{a_t - \mu_\theta(s_t)}{\sigma^2} \nabla_\theta \mu_\theta(s_t).$$

If σ is also learned, the gradient with respect to σ is:

$$\nabla_\theta \log \pi_\theta(a_t|s_t) = \frac{(a_t - \mu_\theta(s_t))^2 - \sigma^2}{\sigma^3} \nabla_\theta \sigma.$$

This shows how the gradient depends on the policy's mean and variance, influencing the updates in policy optimization.

9.3 Algorithm Overview in Continuous Action Space

The steps of the REINFORCE algorithm applied to a continuous action space are as follows:

1. Initialize the policy parameters θ (e.g., for a Gaussian policy, initialize the weights that determine the mean $\mu(s_t)$ and variance σ^2).
2. For each episode:
   - Initialize the state s_0.
   - For each time step:
     - Sample an action a_t from the policy $\pi_\theta(a_t|s_t)$.
     - Execute action a_t, observe reward r_t and next state s_{t+1}.
     - Compute the return G_t (e.g., the cumulative discounted reward).
     - Update the policy parameters using the gradient of the log-probability:

$$\theta \leftarrow \theta + \alpha G_t \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Repeat for multiple episodes to improve the policy.

9.4 Challenges in Continuous Action Spaces

- **Exploration:** In continuous action spaces, exploration can be more challenging because the agent must explore an infinite number of possible actions. One strategy to encourage exploration is to maintain some randomness in the policy (e.g., keeping the variance σ^2 large).

- **Variance of Gradient Estimates:** Policy gradient methods, including REINFORCE, can have high variance in the gradient estimates, especially in continuous action spaces. Techniques like baseline functions or actor-critic methods can help reduce this variance.