Policy Gradient Methods

Devesh Nath

February 9, 2025

Abstract

Policy gradient methods are a class of reinforcement learning algorithms that optimize the policy directly. These methods have gained popularity due to their ability to handle high-dimensional action spaces and stochastic policies. This document provides an overview of policy gradient methods, including their mathematical foundations, key algorithms, and practical considerations.

1 REINFORCE Algorithm

The REINFORCE algorithm is a Monte Carlo policy gradient method. The objective is to maximize the expected return $J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)]$, where τ denotes a trajectory and $R(\tau)$ is the return of the trajectory.
The objective function $J(\theta)$ is defined as:

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

The gradient of the objective function is given by:

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

The REINFORCE update rule is:

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

This can be connected to the principle of maximum likelihood estimation (MLE). In MLE, we aim to find the parameters that maximize the likelihood of the observed data. Similarly, in the REINFORCE algorithm, we adjust the policy parameters θ to maximize the likelihood of actions that lead to higher returns. The term $\nabla_\theta \log \pi_\theta(a_t|s_t)$ represents the gradient of the log-likelihood, which is a common component in MLE. By weighting this gradient with the return $R(\tau)$, we are effectively increasing the likelihood of actions that result in higher returns, thus drawing a parallel to the MLE approach.

2 Reducing Variance Using Baseline Subtraction

One of the challenges with the REINFORCE algorithm is the high variance of the gradient estimates. A common technique to reduce this variance is to use a baseline function. The idea is to subtract a baseline $b(s_t)$ from the return $R(\tau)$ in the gradient estimate. This does not introduce any bias but can significantly reduce the variance.
The modified gradient of the objective function with a baseline is given by:

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

A common choice for the baseline is the value function $V^\pi(s_t)$, which estimates the expected return from state s_t under the current policy π. Using the value function as the baseline, the gradient becomes:

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

Alternatively, the baseline can be the action-value function $Q^\pi(s_t, a_t)$, which estimates the expected return from state s_t after taking action a_t. The advantage function $A^\pi(s_t, a_t)$ is defined as the difference between the action-value function and the value function:

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$$

Using the advantage function as the baseline, the gradient becomes:

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

The REINFORCE update rule with advantage function is:

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

By subtracting the baseline, we reduce the variance of the gradient estimates, leading to more stable and efficient learning. The choice of baseline is crucial, and it is often learned alongside the policy to provide accurate estimates of the expected return. Using the advantage function as the baseline helps to focus the updates on actions that are better or worse than the average, further improving the learning process.
This approach is building up to look like an actor-critic method, where the policy (actor) is updated using the advantage function, and the value function (critic) is used as the baseline. The actor-critic method combines the benefits of policy gradient methods and value-based methods, leading to more stable and efficient learning.

3 Off-Policy Learning and Importance Sampling

Off-policy learning refers to learning a target policy π_θ while following a different behavior policy μ. This is useful in scenarios where we want to learn from data generated by a different policy, such as in batch reinforcement learning or when using experience replay.
To correct for the discrepancy between the target and behavior policies, we use importance sampling. The idea is to reweight the returns by the likelihood ratio of the target and behavior policies.
The importance sampling ratio is defined as:

$$\rho_t = \frac{\pi_\theta(a_t|s_t)}{\mu(a_t|s_t)}$$

Using this ratio, the gradient of the objective function with a baseline $b(s_t)$ is given by:

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

A common choice for the baseline is the value function $V^\pi(s_t)$, which estimates the expected return from state s_t under the current policy π. Using the value function as the baseline, the gradient becomes:

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

The off-policy REINFORCE update rule with baseline subtraction is:

$$\theta \leftarrow \theta + \alpha \sum_{t=0}^{T} \rho_t \nabla_\theta \log \pi_\theta(a_t|s_t)(R(\tau) - V^\pi(s_t))$$

Off-policy learning with importance sampling allows us to leverage data generated by different policies, making it a powerful technique in reinforcement learning. However, it is important to note that high variance in the importance sampling ratios can lead to instability, and techniques such as clipping or using truncated importance sampling are often employed to mitigate this issue.

4 Implementation Using Automatic Differentiation

In this section, we will discuss how to implement the REINFORCE algorithm using automatic differentiation. We will use PyTorch to build the computation graph and compute the gradients.
Given:

- `actions` - (N*T) x Da tensor of actions
- `states` - (N*T) x Ds tensor of states

- `q_values` – (N*T) x 1 tensor of estimated state-action values

We can build the computation graph as follows:

```
import torch
import torch.nn as nn
import torch.nn.functional as F

# Define the policy network
class PolicyNetwork(nn.Module):
    def __init__(self, state_dim, action_dim):
        super(PolicyNetwork, self).__init__()
        self.fc1 = nn.Linear(state_dim, 128)
        self.fc2 = nn.Linear(128, action_dim)

    def forward(self, states):
        x = F.relu(self.fc1(states))
        return self.fc2(x)

# Instantiate the policy network
policy = PolicyNetwork(state_dim=Ds, action_dim=Da)

# Compute the logits
logits = policy(states)  # (N*T) x Da tensor of action logits

# Compute the negative log-likelihoods
negative_likelihoods = F.cross_entropy(logits, actions, reduction='none')

# Weight the negative log-likelihoods by the estimated state-action values
weighted_negative_likelihoods = negative_likelihoods * q_values.squeeze()

# Compute the loss
loss = weighted_negative_likelihoods.mean()

# Compute the gradients
loss.backward()
gradients = [param.grad for param in policy.parameters()]
```

In this implementation, we define a simple policy network using PyTorch's 'nn.Module'. The network takes the states as input and outputs the logits for the actions. We then compute the negative log-likelihoods of the actions, weight them by the estimated state-action values, and compute the loss. Finally, we use PyTorch's automatic differentiation to compute the gradients of the loss with respect to the policy network's parameters.

5 Natural Policy Gradients

Natural policy gradients aim to improve the stability and efficiency of policy gradient methods by taking into account the geometry of the parameter space. Instead of using the standard gradient, natural policy gradients use the natural gradient, which is computed using the Fisher information matrix. The idea is to keep the updated policy close to the current one to prevent policy update to jump around.
The natural gradient is defined as:

$$\tilde{\nabla}_\theta J(\theta) = F(\theta)^{-1} \nabla_\theta J(\theta)$$

where $F(\theta)$ is the Fisher information matrix, which measures the curvature of the parameter space.
The Fisher information matrix is given by:

$$F(\theta) = \mathbb{E}_{s\sim d^\pi, a\sim\pi_\theta} \left[\nabla_\theta \log \pi_\theta(a|s) \nabla_\theta \log \pi_\theta(a|s)^T \right]$$

where d^π is the discounted state distribution under the policy π_θ.
The natural gradient can be interpreted as the direction that maximizes the expected return while taking the shortest path in the parameter space, as measured by the Kullback-Leibler (KL) divergence. The KL divergence between the current policy π_θ and a new policy $\pi_{\theta'}$ is given by:

$$D_{KL}(\pi_\theta \| \pi_{\theta'}) = \mathbb{E}_{s\sim d^\pi, a\sim\pi_\theta} \left[\log \frac{\pi_\theta(a|s)}{\pi_{\theta'}(a|s)} \right]$$

By using the natural gradient, we ensure that the updates to the policy parameters respect the underlying geometry of the parameter space, leading to more stable and efficient learning.
The natural policy gradient update rule is:

$$\theta \leftarrow \theta + \alpha \tilde{\nabla}_\theta J(\theta)$$

where α is the learning rate.
In practice, computing the exact Fisher information matrix and its inverse can be computationally expensive. Therefore, various approximations and techniques, such as the use of conjugate gradient methods, are often employed to make the computation tractable.
Natural policy gradients provide a principled way to improve the performance of policy gradient methods by incorporating information about the geometry of the parameter space, leading to more stable and efficient learning.

6 Proximal Policy Optimization

Proximal Policy Optimization (PPO) is a popular policy gradient method that aims to improve the stability and performance of policy optimization. PPO uses a surrogate objective function that penalizes large changes to the policy, ensuring that updates are more conservative and stable.
The PPO objective function is defined as:

$$L^{CLIP}(\theta) = \mathbb{E}_t \left[\min \left(\rho_t(\theta) A_t, \text{clip}(\rho_t(\theta), 1-\epsilon, 1+\epsilon) A_t \right) \right]$$

where $\rho_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio between the new and old policies, A_t is the advantage function, and ϵ is a hyperparameter that controls the clipping range.
The clipping term $\text{clip}(\rho_t(\theta), 1-\epsilon, 1+\epsilon)$ ensures that the probability ratio $\rho_t(\theta)$ stays within the range $[1-\epsilon, 1+\epsilon]$, preventing large updates that could destabilize the learning process.
The PPO update rule is:

$$\theta \leftarrow \theta + \alpha \nabla_\theta L^{CLIP}(\theta)$$

where α is the learning rate.