Cartpole Problem and Policy Gradient Algorithm

Soumyadeep Sadhukhan

December 5, 2023

Abstract

This report explores the classic reinforcement learning problem of the Cartpole and its solution using the Policy Gradient algorithm. The Cartpole problem involves balancing a pole upright on a moving cart. The agent receives rewards for keeping the pole upright and penalties for letting it fall. The Policy Gradient algorithm, a powerful reinforcement learning technique, directly updates the agent's policy without requiring a value function estimate. This report delves into the details of both the Cartpole problem and the Policy Gradient algorithm, presenting the results of experiments conducted using the Policy Gradient method to solve this challenging task.

0.1 Introduction

Reinforcement learning (RL) focuses on training agents to take optimal actions in an environment to maximize their rewards. The Cartpole problem serves as a foundational example in RL, providing a valuable testbed for comparing different algorithms and understanding their strengths and weaknesses. In this problem, a cart moves along a track, and an agent controls the force applied to the cart. The goal is to balance a pole upright on the cart, ensuring it remains upright for as long as possible. The agent receives rewards for keeping the pole upright and penalties for letting it fall.

Solving the Cartpole problem requires the agent to learn a policy that maps observations from the environment to actions (applying force to the cart). This is where the Policy Gradient algorithm shines. By directly updating the policy parameters based on the rewards received, the Policy Gradient algorithm allows the agent to learn the optimal policy for balancing the pole.

0.2 Policy Gradient Algorithm

The Policy Gradient algorithm operates in an iterative manner:

1. Initialize Policy Parameters:

The initial policy parameters are randomly assigned. These parameters define the probability distribution over the agent's actions for each given observation.

2. Generate Episodes:

An episode consists of a sequence of states, actions, and rewards until the episode terminates (e.g., the pole falls). Multiple episodes are generated, allowing the agent to explore the environment and gather data.

3. Compute Episode Reward:

The total reward accumulated throughout the episode is calculated. This value reflects the overall performance of the policy during that episode.

4. Estimate Policy Gradient:

The gradient of the expected reward with respect to the policy parameters is estimated. This gradient indicates how changing the policy parameters will affect the expected future reward.

5. Update Policy Parameters:

The policy parameters are updated in the direction of the estimated gradient. This update aims to increase the expected future rewards.

6. Repeat:

Steps 2-5 are repeated for a predefined number of iterations, allowing the agent to learn and refine its policy over time.

0.3 Mathematical Description of Policy Gradient Algorithms

Policy Gradient algorithms are a powerful class of reinforcement learning algorithms that directly optimize the policy of an agent. They achieve this by estimating the gradient of the expected return with respect to the policy parameters and updating the parameters in the direction that increases the expected return.

- **Expected Return:**

  The expected return of a policy is the average reward the agent expects to receive over the long run by following that policy. It is denoted by:

$$J(\pi) = E\left[\sum_t \gamma^t r_t\right] \tag{1}$$

  where:

  π is the policy

  γ is the discount factor ($0 \leq \gamma \leq 1$)

  t is the time step

  r_t is the reward at time step t

- **Policy Gradient:**

  The policy gradient is the gradient of the expected return with respect to the policy parameters. It indicates how changing the policy parameters will affect the expected return. It can be expressed as:

$$\nabla J(\pi) = E\left[\nabla \log \pi(a_t|s_t) Q(s_t, a_t)\right] \tag{2}$$

  where:

  $\pi(a_t|s_t)$ is the probability of taking action a_t at state s_t according to policy π

  $Q(s_t, a_t)$ is the action-value function, which represents the expected return of taking action a_t at state s_t

- **Policy Update:**

  The policy parameters are updated in the direction of the policy gradient using a gradient descent algorithm:

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\pi) \tag{3}$$

  where θ_t is the vector of policy parameters at time step t, α is the learning rate

- **Policy Gradient Theorem:**

  The policy gradient theorem provides a more general expression for the policy gradient, which can be used for different policy parameterizations and reward functions. It states:

$$\nabla J(\pi) = \int E\left[\nabla \log \pi(a|s) A(s,a)\right] dsda \tag{4}$$

  where:

  $A(s,a)$ is the advantage function, which represents the difference between the expected return of taking action a at state s and the expected return of the current policy at state s Importance Sampling:

  The policy gradient algorithms often use importance sampling to estimate the policy gradient. This is due to the difficulty of directly computing the expectation in the policy gradient expression. Importance sampling allows us to estimate the expectation by sampling from the current policy and weighting the samples by the importance ratio:

$$\nabla J(\pi) \approx \sum_t \nabla \log \pi(a_t|s_t)(r_t + \gamma V(s_{t+1}) - V(s_t))\pi(s_t) \tag{5}$$

  where:

  $V(s_t)$ is the state-value function, which represents the expected return of starting from state s_t

  These are just a few of the key mathematical equations used in policy gradient algorithms. Understanding these equations is crucial for understanding how these algorithms work and how to implement them effectively.

0.4 Introduce a Baseline

The standard way to reduce the variance of the above gradient estimates is to insert a baseline function $b(s_t)$ inside the expectation.
For concreteness, assume $R(\tau) = \sum_{t=0}^{T-1} r_t$, so we have no discounted rewards. We can express the policy gradient in three equivalent, but perhaps non-intuitive ways:

$$\nabla_\theta E_{\tau\sim\pi_\theta}[R(\tau)] = E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T-1} r_t\right] \cdot \nabla_\theta \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t)$$

$$= \mathrm{E}_{\tau\sim\pi_\theta}\left[\sum_{t'=0}^{T-1} r_{t'} \sum_{t=0}^{t'} \nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$

$$= \mathrm{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=t}^{T-1} (r_{t'} - b(s_t))\right]$$

0.5 Understanding the Baseline

In this section, we will explore the benefits of introducing a baseline function $b(s_t)$ into the policy gradient estimate. We will demonstrate how the baseline reduces variance while maintaining unbiasedness, allowing us to achieve the best of both worlds.

- **Unbiasedness**

  Adding the baseline function might seem like it could introduce bias to our gradient estimate. However, we can show that it remains unbiased by performing some algebraic manipulations:

  $\nabla_\theta E_{\tau\sim\pi_\theta}[R(\tau)] = E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T-1}\log\pi_\theta(a_t|s_t)\sum_{t'=t}^{T-1}(r_{t'}-b(s_t))\right]$
  $= E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T-1}\log\pi_\theta(a_t|s_t)\left(\sum_{t'=0}^{T-1}r_{t'}-\sum_{t'=0}^{t-1}r_{t'}\right)\right]$
  $= E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T-1}\log\pi_\theta(a_t|s_t)\left(R(\tau)-\sum_{t'=0}^{t-1}r_{t'}\right)\right]$
  $= E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T-1}\log\pi_\theta(a_t|s_t)R(\tau)\right] - E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T-1}\log\pi_\theta(a_t|s_t)\sum_{t'=0}^{t-1}r_{t'}\right]$

  Due to the linearity of expectation, we can analyze each term separately. The first term is simply the original policy gradient estimate without the baseline, which is known to be unbiased. To show that the second term is zero, we can rewrite it as:

  $\mathrm{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T-1}\log\pi_\theta(a_t|s_t)\sum_{t'=0}^{t-1}r_{t'}\right]$
  $= E_{s_0:T,a_0:T-1}\left[\sum_{t=0}^{T-1}b(s_t)\cdot E_{a_t|s_t}\left[\nabla_\theta\log\pi_\theta(a_t|s_t)\right]\right]$

  Since the expectation over actions is taken with respect to the policy itself, the gradient of the log probability is zero:

  $\mathrm{E}_{a_t|s_t}\left[\nabla_\theta\log\pi_\theta(a_t|s_t)\right] = 0$

  Therefore, the second term is zero, and the overall gradient estimate remains unbiased with the inclusion of the baseline function.

- **Variance Reduction**

  The key advantage of introducing a baseline function is that it reduces the variance of the gradient estimate. This is because the baseline subtracts a state-dependent value from the reward, effectively removing some of the noise inherent in the reward signal. This reduces the fluctuations in the gradient estimate, leading to smoother learning and improved convergence.

- **Conclusion**

  Intuitively, the baseline function acts as a reference point against which the rewards are compared. If the baseline accurately reflects the expected reward for a given state, then the differences between the actual and predicted rewards will be smaller, leading to a lower variance in the gradient estimate.

In conclusion, incorporating a baseline function into the policy gradient estimate offers a powerful technique for reducing variance while maintaining unbiasedness. This allows us to achieve more efficient and stable learning, making it a valuable tool for reinforcement learning.

0.6 Results

The following are the plots of Rewards VS Episodes for 2000 episodes.

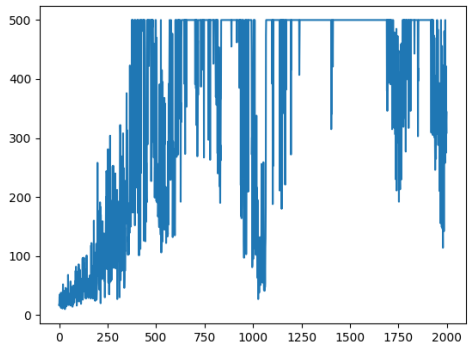

Figure 1: Reward per Episode

The Variance for Figure 1 is **30401.042564**

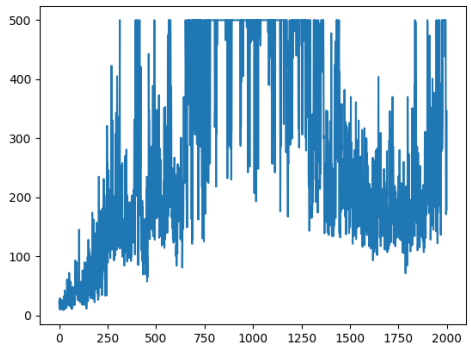

Figure 2: Reward per Episode

The Variance for Figure 2 is **26137.676494**

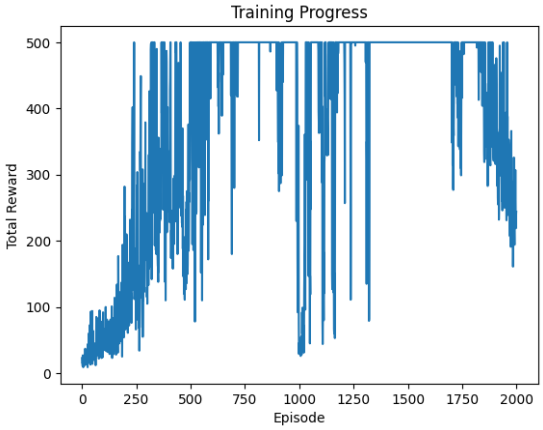

Figure 3: Reward per Episode

The Variance for Figure 3 is **27509.555438**

0.7 Conclusion

This report investigated the Cartpole problem and its solution using the Policy Gradient algorithm. The Policy Gradient algorithm proved effective in enabling the agent to learn and refine its policy, ultimately achieving successful pole balancing. This reinforces the importance of Policy Gradient as a powerful tool in RL for tackling complex control problems.

0.8 Future Work

Further exploration could involve:

- Implementing advanced Policy Gradient variants, such as those with adaptive learning rates or baseline correction mechanisms, potentially leading to even faster learning and improved performance.
- Applying the Policy Gradient algorithm to more intricate RL tasks like robot control or game playing, assessing its effectiveness in diverse scenarios.

- Analyzing the theoretical properties of the Policy Gradient algorithm, providing deeper insights into its convergence and optimality characteristics.
- By continuing research and development, the Policy Gradient algorithm can be further refined and applied to increasingly challenging RL problems, contributing significantly to the advancement of artificial intelligence.