Thompson Sampling for CartPole-v1

Project Overview

This project demonstrates the application of **Thompson Sampling**, an exploration-driven algorithm, to the classic reinforcement learning environment **CartPole-v1** from OpenAI's Gym. The goal is to balance a pole on a moving cart by applying forces to the cart, preventing the pole from falling over.

Why Thompson Sampling?

Thompson Sampling is a probabilistic algorithm used for balancing exploration and exploitation in decision-making problems, particularly in multi-armed bandit scenarios. It maintains a distribution over possible outcomes and selects actions based on the probability of being optimal, allowing for efficient exploration of the action space.

Prerequisites

Required Libraries

- `numpy` : For numerical computations.
- `gym` : For the CartPole-v1 environment.

Installation

Install the necessary libraries using pip:

```
pip install numpy gym
```

Code Description

The implementation consists of the following key components:

1. Importing Libraries

```
import numpy as np
import gym
```

2. Environment Setup

Initialize the CartPole-v1 environment:

```
env = gym.make('CartPole-v1')
```

3. Hyperparameters

Define the hyperparameters for Thompson Sampling:

```
alpha = 1  # Prior success count for Beta distribution
beta = 1   # Prior failure count for Beta distribution
gamma = 0.99  # Discount factor (not used in this specific implementation)
```

4. Initialization of Success and Failure Counts

Initialize matrices to keep track of successes and failures for each state-action pair:

```
successes = np.ones((env.observation_space.shape[0], env.action_space.n)) * alpha
failures = np.ones((env.observation_space.shape[0], env.action_space.n)) * beta
```

5. Thompson Sampling Policy

Define the policy for selecting actions based on Thompson Sampling:

```
def thompson_sampling_policy(state):
    samples = np.random.beta(successes[state], failures[state])
    return np.argmax(samples)  # Select action with the highest sample
```

6. Training Loop

Implement the training loop to update the success and failure counts based on the agent's interactions with the environment:

```
def thompson_sampling_q_learning(env, n_episodes=1000):
    for episode in range(n_episodes):
        state = env.reset()
        done = False
        while not done:
            action = thompson_sampling_policy(state)
            next_state, reward, done, _ = env.step(action)

            # Update Beta distributions based on success/failure
            if reward > 0:  # Assuming positive reward indicates success
                successes[state, action] += 1
            else:
                failures[state, action] += 1

            state = next_state
```

7. Running the Algorithm

Execute the training function:

```
thompson_sampling_q_learning(env)
```

Expected Outcomes

- **Policy Development**: The agent develops a policy to balance the pole by learning which actions lead to successful outcomes.
- **Exploration Efficiency**: Thompson Sampling efficiently balances exploration and exploitation, leading to effective learning in the CartPole-v1 environment.

Use Cases

- **Reinforcement Learning**: Applying exploration strategies in environments where the balance between exploration and exploitation is crucial.
- **Multi-Armed Bandit Problems**: Situations where decisions need to be made under uncertainty with the goal of maximizing cumulative rewards.

Future Enhancements

- **State Discretization**: Implement state discretization techniques to handle the continuous state space of CartPole-v1.
- **Reward Shaping**: Modify the reward structure to provide more informative feedback to the agent.
- **Comparison with Other Algorithms**: Compare the performance of Thompson Sampling with other exploration strategies like ?-greedy or Upper Confidence Bound (UCB).

References

- OpenAI Gym: CartPole-v1
- Thompson Sampling Explained with Python Code
