Aim : Implement a policy gradient algorithms like REINFORCE or Proximal Policy Optimization (PPO) to solve a continuous control task.

Objective :

In this experiment, we aim to solve the Pendulum-v0 environment using the Proximal Policy Optimization (PPO) algorithm. The Pendulum-v0 environment requires controlling a pendulum to stand upright by applying continuous actions, making it a suitable candidate for testing continuous control algorithms.

Methodology

Environment: We utilize the Pendulum-v0 environment provided by OpenAI Gym. This environment provides observations representing the pendulum's angle and angular velocity, and allows continuous actions to control the torque applied to the pendulum.

Policy Network: We employ a neural network as the policy function approximator. The neural network takes the environment's observations as input and outputs the mean and standard deviation of a Gaussian distribution representing the policy.

PPO Algorithm: We implement the Proximal Policy Optimization algorithm to update the policy network. PPO uses clipped surrogate objective to ensure stable policy updates and employs a value function to estimate the state-action value.

Training Procedure: The training process involves iteratively collecting trajectories by interacting with the environment using the current policy. We then use these trajectories to compute the policy gradient and update the policy network parameters using gradient ascent. Additionally, we update the value function parameters to improve the state-value estimation.

Hyperparameters: We carefully tune hyperparameters such as learning rate, discount factor, entropy coefficient, clipping parameter, etc., to ensure stable and efficient training.

Initialization: We initialize the policy network and value function parameters randomly or using a pre-trained model if available.

Training: We train the policy network and value function using the PPO algorithm. During training, we periodically evaluate the performance of the policy by running episodes with the current policy and recording the average reward.

Evaluation: After training for a sufficient number of iterations or episodes, we evaluate the final policy by running episodes and measuring its performance in terms of average reward, episode length, and stability.

Results

Training Progress: We plot the training progress, showing the change in average reward and other relevant metrics over training iterations or episodes. This provides insights into the algorithm's learning dynamics and convergence.

Evaluation Performance: We report the performance of the final trained policy on the Pendulum-v0 environment, including average reward achieved, episode length, and any observations regarding stability and robustness.

We analyze the results obtained from the experiment, discussing the effectiveness of the PPO algorithm in solving the continuous control task in the Pendulum-v0 environment. We also discuss any challenges encountered during training, potential improvements, and future directions.

Conclusion

In conclusion, we demonstrate the application of Proximal Policy Optimization (PPO) algorithm to solve the continuous control task in the Pendulum-v0 environment. The results indicate the effectiveness of the algorithm in learning a policy that successfully stabilizes the pendulum.

Code :

```
import torch

import numpy as np

import gym

from stable_baselines3 import PPO


# Define the Pendulum environment

env = gym.make('Pendulum-v0')


# Define the PPO agent

model = PPO("MlpPolicy", env, verbose=1)


# Train the agent

model.learn(total_timesteps=10000)


# Evaluate the agent

mean_reward, std_reward = evaluate(model, env, n_eval_episodes=10)

print(f"Mean reward: {mean_reward:.2f} +/- {std_reward:.2f}")
```

```
# Helper function to evaluate the agent
def evaluate(model, env, n_eval_episodes=10):
    rewards = []
    for _ in range(n_eval_episodes):
        obs = env.reset()
        episode_reward = 0
        done = False
        while not done:
            action, _ = model.predict(obs, deterministic=True)
            obs, reward, done, _ = env.step(action)
            episode_reward += reward
        rewards.append(episode_reward)
    mean_reward = np.mean(rewards)
    std_reward = np.std(rewards)
    return mean_reward, std_reward
```