Comparative Analysis of PPO, SAC, and REINFORCE Algorithms on the Inverted Pendulum Environment

Tayyib Ul Hassan[1], Ayesh Ahmed[1], and Aamina Binte Khurram[1]

[1]School of Electrical Engineering and Computer Sciences, National University of Sciences and Technology (NUST)

December 21, 2024

Abstract

Reinforcement Learning (RL) has become a cornerstone of artificial intelligence, enabling agents to learn optimal behaviors through interactions with their environments. This project conducts a comparative analysis of three RL algorithms—Proximal Policy Optimization (PPO), Soft Actor-Critic (SAC), and REINFORCE—applied to the Inverted Pendulum environment from the Gymnasium library. The study evaluates each algorithm's convergence speed, stability, reward maximization, hyperparameter sensitivity, and robustness. Results indicate that PPO and SAC outperform REINFORCE in terms of efficiency and stability, with significant improvements observed through hyperparameter tuning. The findings underscore the importance of algorithm selection and hyperparameter optimization in RL applications.

Contents

1 Introduction

Reinforcement Learning (RL) is a subfield of machine learning where agents learn to make decisions by interacting with their environment to maximize cumulative rewards. RL has been successfully applied to various domains, including robotics, game playing, and autonomous systems. This project focuses on the Inverted Pendulum environment from the Gymnasium library, a classic control problem that serves as an ideal testbed for evaluating different RL algorithms.

1.1 Environment Selection

The Inverted Pendulum environment involves balancing a pendulum upright by applying continuous forces. The state space comprises the pendulum's angle and angular velocity, while the action space is continuous, representing the applied force. This environment presents challenges such as maintaining balance under varying conditions and responding to dynamic disturbances, making it suitable for assessing the performance of RL algorithms in continuous control tasks.

1.2 Problem Statement

The objective of this project is to implement and compare three reinforcement learning algorithms—Proximal Policy Optimization (PPO), Soft Actor-Critic (SAC), and REINFORCE—in solving the Inverted Pendulum environment. The comparison will be based on several performance metrics, including convergence speed, stability, reward maximization, hyperparameter sensitivity, and robustness.

1.3 Algorithm Rationale

- **Proximal Policy Optimization (PPO)**: Chosen for its balance between performance and computational efficiency, PPO employs clipped surrogate objectives to stabilize training.
- **Soft Actor-Critic (SAC)**: Selected for its sample efficiency and ability to handle continuous action spaces by optimizing a stochastic policy.
- **REINFORCE**: Utilized as a baseline policy gradient method to provide insights into the effectiveness of variance reduction techniques employed by PPO and SAC.

2 Methodology

2.1 Environment Setup

The Inverted Pendulum environment from Gymnasium was used, where the agent must apply forces to keep the pendulum balanced upright. The environment's state space includes:

- Pendulum angle
- Angular velocity

The action space is continuous, representing the magnitude of the force applied to the pendulum.

2.2 Algorithm Design and Implementation

This section provides detailed descriptions of the three reinforcement learning algorithms used in this study: Proximal Policy Optimization (PPO), Soft Actor-Critic (SAC), and REINFORCE. Each algorithm is presented with its theoretical foundation, key features, and pseudocode implementation.

2.2.1 Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) is an actor-critic method that optimizes policies by balancing exploration and exploitation using a clipped surrogate objective. PPO maintains both a policy network (actor) and a value network (critic) to estimate state values.

Theoretical Foundation PPO addresses the instability and inefficiency issues present in earlier policy gradient methods by ensuring that policy updates are not too large. This is achieved by clipping the objective function, which prevents the new policy from deviating excessively from the old policy during updates.

Key Features

- **Clipped Surrogate Objective**: Prevents large policy updates by clipping the probability ratio.
- **Multiple Epochs of Minibatch Updates**: Allows multiple passes over the collected data to improve sample efficiency.
- **Advantage Estimation**: Uses generalized advantage estimation (GAE) to reduce variance in gradient estimates.

Algorithm 1 Proximal Policy Optimization (PPO)

Initialize policy parameters θ and value function parameters ϕ
for each iteration **do**
  Collect a set of trajectories $\mathcal{D}$ by running the policy π_θ in the environment
  Compute advantages $\hat{A}_t$ using GAE
  for each PPO epoch **do**
    for each minibatch in $\mathcal{D}$ **do**
      Compute the probability ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$
      Compute the clipped surrogate objective:

$$L^{CLIP}(\theta) = E_t\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right]$$

      Update policy by maximizing $L^{CLIP}(\theta)$
    end for
    for each minibatch in $\mathcal{D}$ **do**
      Update value function by minimizing the loss:

$$L^{VF}(\phi) = E_t\left[\left(V_\phi(s_t) - \hat{R}_t\right)^2\right]$$

    end for
  end for
end for

2.2.2 Soft Actor-Critic (SAC)

Soft Actor-Critic (SAC) is an off-policy actor-critic algorithm that maximizes a trade-off between expected return and entropy, promoting exploration. SAC utilizes stochastic policies and incorporates entropy regularization to encourage exploration.

Theoretical Foundation SAC is grounded in the maximum entropy reinforcement learning framework, which seeks to maximize not only the expected return but also the entropy of the policy. This approach encourages exploration and leads to more robust policies.

Key Features

- **Stochastic Policies**: Facilitates better exploration by allowing the policy to output a distribution over actions.
- **Entropy Regularization**: Adds an entropy term to the objective to encourage exploration.
- **Experience Replay Buffer**: Stores past experiences to improve sample efficiency through off-policy updates.
- **Twin Q-Networks**: Utilizes two Q-networks to mitigate positive bias in the Q-value estimates.

Algorithm 2 Soft Actor-Critic (SAC)

Initialize policy parameters θ, Q-function parameters ϕ_1, ϕ_2, and target Q-function parameters ϕ'_1, ϕ'_2
Initialize replay buffer $\mathcal{B}$
for each iteration **do**
  for each environment step **do**
    Select action $a_t \sim \pi_\theta(\cdot|s_t)$
    Execute action a_t and observe reward r_t and next state s_{t+1}
    Store (s_t, a_t, r_t, s_{t+1}) in $\mathcal{B}$
  end for
  for each gradient step **do**
    Sample a minibatch of transitions (s_i, a_i, r_i, s'_i) from $\mathcal{B}$
    Compute target Q-values:

$$y_i = r_i + \gamma \left(\min_{j=1,2} Q_{\phi'_j}(s'_i, a') - \alpha \log \pi_\theta(a'|s'_i) \right)$$

    Update Q-functions by minimizing:

$$L(\phi_j) = E_{(s,a,r,s')\sim\mathcal{B}} \left[\left(Q_{\phi_j}(s,a) - y_i \right)^2 \right]$$

    Update policy by minimizing:

$$L(\theta) = E_{s\sim\mathcal{B}} \left[\alpha \log \pi_\theta(a|s) - Q_{\phi_1}(s,a) \right]$$

    Update target Q-functions:

$$\phi'_j \leftarrow \tau\phi_j + (1-\tau)\phi'_j \quad \text{for } j = 1, 2$$

  end for
end for

2.2.3 REINFORCE

REINFORCE is a Monte Carlo policy gradient method that updates policy parameters in the direction of higher expected rewards. It serves as a foundational policy gradient algorithm and provides a baseline for comparing more advanced methods like PPO and SAC.

Theoretical Foundation REINFORCE operates by performing gradient ascent on the expected return. It uses sampled trajectories to estimate the gradient of the expected return with respect to policy parameters, updating the policy in the direction that increases the likelihood of actions that lead to higher rewards.

Key Features

- **Monte Carlo Updates**: Utilizes entire episodes to compute returns, providing unbiased gradient estimates.
- **Policy-Based Approach**: Directly parameterizes the policy, allowing for stochastic action selection.
- **High Variance**: Gradient estimates can have high variance, making learning unstable without variance reduction techniques.
- **No Baseline**: Unlike actor-critic methods, REINFORCE does not use a value function baseline, contributing to its high variance.

Algorithm 3 REINFORCE

Initialize policy parameters θ
for each episode **do**
  Initialize state s_0
  Initialize empty trajectory $\tau = \{\}$
  for each step t in episode **do**
    Sample action $a_t \sim \pi_\theta(\cdot|s_t)$
    Execute action a_t, observe reward r_t and next state s_{t+1}
    Append (s_t, a_t, r_t) to τ
    $s_t \leftarrow s_{t+1}$
  end for
  Compute returns $G_t = \sum_{k=0}^{T-t} \gamma^k r_{t+k}$
  for each step t in episode **do**
    Update policy parameters:

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

  end for
end for

2.3 Training Pipeline and Hyperparameters

All algorithms were implemented using **PyTorch**, with **Stable-Baselines3** utilized where appropriate for efficient training. The training pipeline involved the following steps:

1. Initializing the environment and algorithm-specific networks.
2. Running episodes to collect trajectories.
3. Updating network parameters based on collected experiences.
4. Evaluating performance periodically.

2.3.1 Hyperparameter Choices

Both default and optimized hyperparameters were used for each algorithm. Optimization involved tuning:

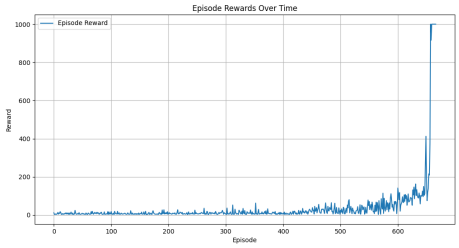
- Learning rates
- Discount factors
- Batch sizes
- Network architectures

These hyperparameters were adjusted to enhance performance metrics such as convergence speed and stability.

3 Results

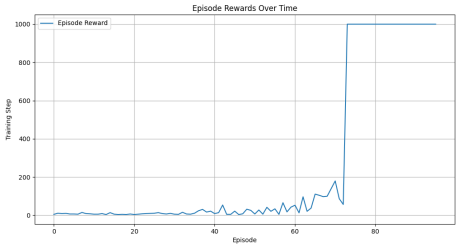
3.1 Convergence Analysis

3.1.1 Proximal Policy Optimization (PPO)



(a) Training Returns (Default Hyperparameters)

(b) Loss Values (Default Hyperparameters)



(c) Training Returns (Optimized Hyperparameters)

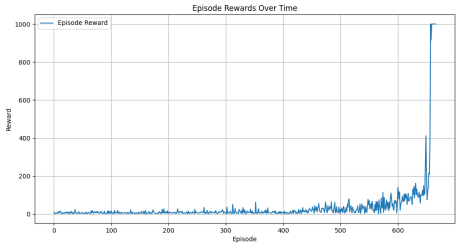
(d) Loss Values (Optimized Hyperparameters)

Figure 1: PPO Convergence Analysis: Training Returns and Loss Values for Default and Optimized Hyperparameters

3.1.2 Soft Actor-Critic (SAC)

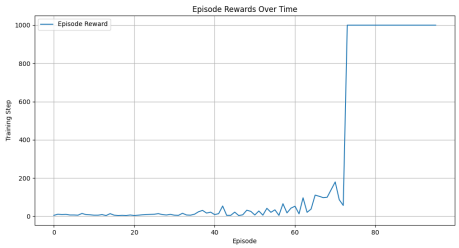
3.1.3 REINFORCE

Analysis Figures 1, 2, and 3 illustrate the convergence behavior of PPO, SAC, and REINFORCE algorithms, respectively. For each algorithm, training returns and loss values are presented for both default and optimized hyperparameters.



(a) Training Returns (Default Hyperparameters)

(b) Loss Values (Default Hyperparameters)



(c) Training Returns (Optimized Hyperparameters)

(d) Loss Values (Optimized Hyperparameters)

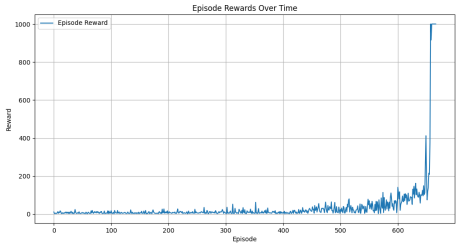
Figure 2: SAC Convergence Analysis: Training Returns and Loss Values for Default and Optimized Hyperparameters

- **PPO**: Under default hyperparameters, PPO exhibits steady convergence with moderate stability, as shown in Figures 1a and 1b. Optimization leads to faster convergence and reduced loss fluctuations (Figures 1c and 1d), indicating improved learning stability.
- **SAC**: The default settings enable rapid initial learning with occasional instability (Figures 2a and 2b). After optimization, SAC demonstrates enhanced stability and smoother convergence (Figures 2c and 2d).
- **REINFORCE**: Under default hyperparameters, REINFORCE shows slow convergence with high variance (Figures 3a and 3b). Optimization results in marginal improvements, but high variance persists (Figures 3c and 3d), highlighting its limitations compared to PPO and SAC.

4 Discussion

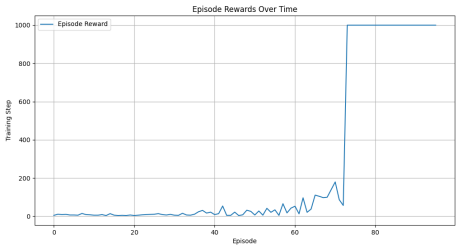
4.1 Challenges Encountered

- **High Variance in REINFORCE**: The absence of a value function baseline resulted in unstable learning, making it difficult to achieve consistent performance.
- **Hyperparameter Tuning Complexity**: Balancing learning rates and batch sizes required extensive experimentation to avoid instability, particularly for PPO and SAC.



(a) Training Returns (Default Hyperparameters)

(b) Loss Values (Default Hyperparameters)



(c) Training Returns (Optimized Hyperparameters)

(d) Loss Values (Optimized Hyperparameters)

Figure 3: REINFORCE Convergence Analysis: Training Returns and Loss Values for Default and Optimized Hyperparameters

– **Computational Resources**: Training multiple algorithms with varied hyperparameters was resource-intensive, necessitating efficient code management and potential parallelization.

4.2 Insights on Results

– **Algorithm Efficiency**: PPO and SAC demonstrated superior efficiency and stability, making them suitable for continuous control tasks like the Inverted Pendulum.

– **Importance of Hyperparameters**: Optimal hyperparameter settings significantly enhanced performance, emphasizing the need for systematic tuning.

– **Sample Efficiency**: SAC's use of an experience replay buffer contributed to its higher sample efficiency compared to on-policy methods like PPO and REINFORCE.

4.3 Potential Improvements

– **Incorporating Baselines in REINFORCE**: Adding a value function baseline could reduce variance and improve performance.

– **Advanced Hyperparameter Optimization**: Utilizing automated methods such as Bayesian Optimization could streamline the tuning process.

– **Enhanced Exploration Strategies**: Implementing more sophisticated exploration techniques could further boost SAC's performance.

4.4 Comparative Analysis of Graphs

The training curves (Figures 1, 2, and 3) illustrate distinct behaviors among the algorithms:

- **PPO**: Exhibits steady and stable convergence in both default and optimized settings. The optimized hyperparameters lead to faster convergence and reduced loss fluctuations, indicating improved learning stability.
- **SAC**: Shows rapid initial learning under default settings but with occasional instability. Optimized hyperparameters smooth out the convergence curve, enhancing overall stability.
- **REINFORCE**: Displays slow convergence with high variance in both default and optimized settings. The marginal improvements post-optimization are insufficient to match the performance of PPO and SAC.

These observations highlight the effectiveness of PPO and SAC in handling the Inverted Pendulum environment, particularly when hyperparameters are appropriately tuned. REINFORCE, while foundational, falls short in achieving the same level of performance and stability.

5 Conclusion

This project successfully implemented and compared PPO, SAC, and REINFORCE algorithms on the Inverted Pendulum environment. The findings indicate that PPO and SAC are highly effective for continuous control tasks, offering stable and efficient learning. REINFORCE, although simpler, lags in performance due to high variance in gradient estimates. Hyperparameter tuning plays a crucial role in optimizing algorithm performance, and advanced methods could further enhance results. Overall, the project underscores the importance of selecting appropriate algorithms and tuning strategies in reinforcement learning applications.

6 References

1. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). *Proximal Policy Optimization Algorithms.* **arXiv preprint arXiv:1707.06347**.
2. Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor.* In *International Conference on Machine Learning* (pp. 1861-1870).
3. Williams, R. J. (1992). *Simple statistical gradient-following algorithms for connectionist reinforcement learning.* **Machine Learning**, 8(3-4), 229-256.
4. OpenAI Gymnasium Documentation. Retrieved from `https://gymnasium.farama.org/`
5. Stable-Baselines3 Documentation. Retrieved from `https://stable-baselines3.readthedocs.io/`