Reinforcement Learning Assignment 3
Policy-based RL

Suju Li Ruiqing Sun Peiwen Xing

Abstract

This report presents a comprehensive study of policy-based reinforcement learning (RL) algorithms, with a particular focus on the REINFORCE and Actor-Critic algorithms. By leveraging entropy regularization, we aim to enhance exploration capabilities and prevent premature policy convergence to suboptimal deterministic strategies. Through a series of experiments, we assess the performance of the REINFORCE algorithm alongside Actor-Critic variants, providing a detailed analysis of their effectiveness in this benchmark setting. Results indicate significant insights into the dynamics of policy-based methods in RL, highlighting both strengths and limitations. The findings contribute to a deeper understanding of policy optimization in RL and suggest pathways for future enhancements.

1. Introduction

In this assignment, we will implement policy-based RL in the environment provided by OpenAI's Cartpole, which is a classic reinforcement learning problem that serves as a benchmark for various algorithms. In this environment, we aim to implement algorithms such as REINFORCE, Actor-Critic, and its variants. We will also use entropy regurization to encourage exploration by the learning agent, which is particularly useful in preventing the policy from prematurely converging to a suboptimal deterministic strategy. At last, we will conduct experiments to evaluate the performance of algorithms, interpret the results, discuss the limits and possible improvement in the future.

2. Methodology

2.1. REINFORCE

The main idea of REINFORCE is to learn a parameterized policy, which takes states as inputs and generates actions according to a certain probability distribution. The key idea is that the probability of producing actions with high rewards gradually increases while reducing the probability of producing actions with low rewards. In other words, it aims to make the policy distribution concentrate more on good actions. Therefore, through multiple training iterations, the probability distribution of actions generated by the parameterized policy will exhibit good performance.

Before diving into the details of REINFORCE, it's important to define several key concepts. First, policy π is defined as a function mapping states to actions. REINFORCE models the policy as a parameterized neural network, where the parameters are denoted by π. The parameterized policy is represented as π_θ, and the goal is to continuously improve these parameters θ so that the agent can maximize its reward by following the parameterized policy.

Therefore, next, we need to discuss what the objective of the agent is according to this policy, i.e., how to formalize the objective function.

Now, starting from the initial state s_0, using Monte Carlo (MC) sampling method, we obtain a trajectory throughout the entire probability space, denoted as τ_i.

$$\tau_i = \{s_0, a_0, r_0, s_1, a_1, \ldots, s_{T-1}, a_{T-1}, r_{T-1}, s_T\}$$

Next, we can define the cumulative discounted reward from the starting state s_0 until the termination state, denoted as $R(\tau_i)$, formally defined as:

$$R(\tau_i) = \sum_{t=0}^{T} \gamma^t r_t$$

Then, we want to know what the expected return of this reward is. A direct and unbiased estimation method is to sample multiple complete trajectories using Monte Carlo and take their average. The more samples we take, the more accurate the estimate will be, and the closer it will be to the true value. Under the policy π_θ followed by the agent, the expected return can be obtained by sampling as many trajectories as possible and summing them according to their probabilities. This is formally defined as:

$$\bar{R}(\pi_\theta) = \sum_{\tau_i}^{\infty} R(\tau_i) P_{\pi_\theta}(\tau_i) \approx \mathbb{E}_{\tau \backsim \pi_\theta}\left[R(\tau)\right]$$

From this, we derive the improvement target regarding the policy, which is the expected return of all complete trajec-

tories generated by the agent, denoted as $J(\pi_\theta)$, formally defined as:

$$J(\pi_\theta) = \mathbb{E}_{\tau \backsim \pi_\theta}\left[R(\tau)\right] = \mathbb{E}_{\tau \backsim \pi_\theta}\left[\sum_{t=0}^{T} \gamma^t r_t\right]$$

Based on the defined policy and objective function, REINFORCE can be formulated as an unconstrained optimization problem:

$$\max_\theta J(\pi_\theta) = \mathbb{E}_{\tau \backsim \pi_\theta}\left[R(\tau)\right]$$

This problem essentially aims to find a set of parameters θ in the parameter space that maximizes the expected return. The simplest way to solve this optimization problem is to use gradient ascent, where the parameter θ is iteratively updated according to the rule:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\pi_\theta)$$

Here, α is the step size, also known as the learning rate, which controls the magnitude of parameter updates in each iteration. The term $\nabla_\theta J(\pi_\theta)$ is referred to as the policy gradient. Here, we directly present the formal description of the policy gradient:

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

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

Here, $\pi_\theta(a_t|s_t)$ represents the probability of the agent taking action a_t at state s_t, and $R_t(\tau)$ represents the expected return from state s_t until the end. It can be observed that the policy gradient involves an expectation operation, which can cause significant computational challenges in implementation. Therefore, we still consider using sample averaging to obtain an unbiased estimate. Assuming N trajectories are sampled, the policy gradient can be approximated numerically as:

$$\nabla_\theta J(\pi_\theta) = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T} \big(R_t(\tau^n)\big) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

The above constitutes the main idea of the REINFORCE method, which is also a same-trajectory policy algorithm. Due to the high sample variance caused by Monte Carlo, one way to improve REINFORCE is to use the baseline REINFORCE, whose policy gradient is defined as follows:

$$\nabla_\theta J(\pi_\theta) = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T} \big(R_t(\tau^n) - b(s_t)\big) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

The three important components of REINFORCE include a parameterized policy π_θ, an objective function to be maximized, and an algorithm for updating policy parameters θ.

2.2. Entropy Regularization

Entropy regularization enhances exploration in reinforcement learning by preventing the policy from converging too quickly to suboptimal deterministic actions.

Entropy is calculated as:

$$H(\pi) = -\sum_a \pi(a \mid s, \theta) \log \pi(a \mid s, \theta)$$

The policy loss function incorporating entropy is:

$$L(\theta) = -\left(\sum_{t=0}^{T} \log \pi(a_t \mid s_t, \theta)(G_t - V(s_t)) + \lambda H(\pi)\right)$$

where λ is a coefficient that regulates the contribution of entropy to the total loss, promoting exploration.

2.3. Actor Critic

The Actor-Critic method employs a policy network (Actor) and a value network (Critic) to estimate both the policy and state values, respectively. The advantage function is computed as:

$$A(s, a) = G_t - V(s_t, w)$$

Policy and value network updates are performed using:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi(a_t \mid s_t, \theta) A(s_t, a_t)$$

$$w \leftarrow w - \beta \nabla_w (G_t - V(s_t, w))^2$$

Updates to both the Actor and Critic are executed simultaneously, with the Critic estimating the state value and the Actor updating the policy based on the calculated advantage.

2.3.1. Actor Critic with Bootstrapping

Bootstrapping in Actor-Critic methods involves updating the policy based on future reward estimates provided by the value network, rather than waiting for the completion of entire episodes. This approach allows for faster policy updates by leveraging immediate reward predictions.

Policy Network: Implements a neural network to model the policy, taking the state as input and outputting a probability distribution over actions. Actions are sampled according to this distribution at each step.

Value Network: A separate neural network approximates the state value function, providing baseline estimates for advantage computation.

Policy Update: During each training step, the policy gradient is computed using advantages, which are calculated by deducting the value network's baseline estimate from the returns.

Value Network Update: Updates are made via bootstrapping, where future state value estimates inform the training of the value network using Mean Squared Error (MSE) loss.

Advantage Computation: Advantages are computed as $A(s, a) = G_t - V(s)$, where G_t are the returns and $V(s)$ is the baseline estimate from the value network.

2.3.2. Actor Critic with Baseline Subtraction

Baseline Subtraction normalizes the rewards using the value estimates from the Critic network, which helps to reduce the variance in the policy gradient updates and stabilize training.

Policy Network: Uses a neural model to estimate the policy, outputting action probabilities based on input states.

Value Network: Employs a neural model to estimate the value of states, which is then used to normalize the rewards or bootstrap future state values.

Policy Update: Policy updates utilize advantages derived from either directly using the bootstrapped next state values or the current state's value estimate as baselines.

Value Network Update: Continues using bootstrapped future state values as targets for MSE loss optimization.

Advantage Computation: Advantages are computed using the formula $A(s, a) = G_t - V(s_{t+1})$, incorporating bootstrapped values for more dynamic adjustment.

2.3.3. Combination of Techniques

Integrating both bootstrapping and baseline subtraction allows for utilizing immediate rewards and adjusted advantages to update policy and value networks concurrently. This hybrid approach aims to balance learning efficiency with training stability.

Policy Updates:

$$policy\_loss = \sum_{t=0}^{T}(-\log \pi(a_t \mid s_t, \theta) \cdot adv_t)$$

Value Updates:

$$value\_loss = MSE(V(s_t, w), G_t)$$

These updates ensure that the agent's learning process is both rapid and robust, mitigating the variance while enhancing the convergence rate.

2.4. PPO

Proximal Policy Optimization (PPO) is a popular reinforcement learning (RL) algorithm that belongs to the family of policy gradient methods, which specifically falls under the category of actor-critic methods. PPO aims to improve upon the standard actor-critic method by addressing the problem of large policy updates, which can lead to performance collapse. It does this by ensuring that the updates are "proximal" (close) to the current policy, hence the name.

Clipped Surrogate Objective Function: The central component of PPO is its objective function, which modifies the typical policy gradient objective by incorporating a clipping mechanism to limit the policy update at each step. This clipping is defined by:

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

where:

1. $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)}$ is the ratio of the new policy to the old policy probabilities for action a_t at state s_t,
2. $\hat{A}_t$ is an estimator of the advantage at time t, which measures the benefit of taking action a_t over the average action at state s_t,
3. ϵ is a hyperparameter, typically set between 0.1 and 0.2, which defines the bounds for clipping to moderate the updates.

Advantage Estimation: PPO typically uses Generalized Advantage Estimation (GAE) to compute the advantage function $\hat{A}_t$, which helps in reducing the variance of the gradient estimates while retaining sufficient bias to guide effective learning. GAE combines temporal difference (TD) error estimates over multiple time steps, providing a balance between multi-step returns and bootstrapped values, controlled by a decay parameter λ.

3. Experiment

We designed 7 experiments including 2 for ablation, 2 for the impact of different entropy coefficients, 2 for network learning rates, 1 for the performance of PPO. The implementation and results could be found in our GitHub repository.

3.1. Neural Netwowrk Architecture

The *NeuralNet*, a fundamental component of our implementation, is a fully connected feed-forward neural network designed to map environmental states to action probabilities. This mapping is critical for making informed action decisions in various states observed during interaction with the environment.

Layers: The architecture includes an input layer corresponding to the environment's state dimensions, three hidden layers with $32, 32, 16$ neurons respectively, and a softmax output layer that converts the neural outputs to a probability distribution over actions.

Activation and Initialization: Each hidden layer employs the ReLU activation function for its ability to introduce non-linearity efficiently, while the Xavier normal initialization is used for setting up weights optimally to prevent issues related to gradient propagation.

3.2. Exploration and Exploitation Policy

Exploration is managed through a stochastic policy derived from the softmax probabilities output by the neural network, encouraging the agent to explore the environment effectively without sticking to suboptimal strategies.

3.2.1. Stochastic Policy:

Our model implements a stochastic policy by utilizing the softmax output of the *NeuralNet*, which provides a probability distribution over possible actions. This design inherently encourages exploration by enabling the agent to explore diverse strategies through the following sampling method:

Algorithm 1 Stochastic Policy Implementation

1: Initialize policy network *policy_net* with softmax output
2: **for** each state s_t in the environment **do**
3: Calculate probabilities: $p = \text{softmax}(\textit{policy\_net}(s_t))$
4: Sample action $a_t \sim p$
5: Execute action a_t, observe reward and next state s_{t+1}
6: **end for**

3.2.2. Entropy Regularization:

To prevent the policy from becoming overly deterministic and to encourage exploration, we employ entropy regularization. This method involves adding an entropy term to the loss function, which penalizes policies that exhibit too much certainty:

Algorithm 2 Entropy Regularization for Policy Exploration

1: Initialize policy network *policy_net*
2: **for** each training step **do**
3: Compute action probabilities p using *policy_net*
4: Calculate entropy: $H = -\sum p \cdot \log(p)$
5: Incorporate entropy into loss function: $\mathcal{L} = \mathcal{L}_{\text{original}} - \lambda H$
6: Update *policy_net* by minimizing loss $\mathcal{L}$
7: **end for**

The baseline setting for the entropy coefficient (*entropy_coef*) is 0.05, which represents the degree of exploration encouragement in the policy for trying new actions. Then we explored the trade-off between exploration and exploitation in our model's training dynamics:

1. To balance exploration with convergence, we adjusted the *entropy_coef* to 0.01, observing how a higher coefficient affects stability and learning speed.
2. To evaluate the impact of reduced exploration on the speed of convergence and the potential risk of premature convergence to suboptimal policies, we further adjusted the *entropy_coef* to 0.002.

3.3. Advanced Techniques for Policy Optimization

3.3.1. Baseline Subtraction

The value network acts as a baseline, estimating the state values which are then used to calculate the advantages during policy updates. This method, embedded within function *train_actor_critic*, enhances training efficiency by addressing the high variance associated with sampled returns.

Algorithm 3 Baseline Subtraction in Policy Learning

1: Initialize value network *value_net*, policy network *policy_net*
2: **for** each episode **do**
3: Collect set of states, actions, and rewards
4: Calculate predicted values $V(s)$ for each state using *value_net*
5: Compute advantages $A_t = R_t - V(s_t)$ for each timestep
6: Update *policy_net* using advantages to reduce variance
7: **end for**

3.3.2. Bootstrapping

By updating the value network at each timestep using predictions of future rewards, we integrate immediate feedback into the learning process:

Algorithm 4 Bootstrapping for Efficient Policy Learning

1: Initialize value network *value_net*, policy network *policy_net*
2: **for** each step in the episode **do**
3: Take action a_t, observe reward r_t and next state s_{t+1}
4: Estimate future value: $V_{\text{future}} = \textit{value\_net}(s_{t+1})$
5: Compute target $y_t = r_t + \gamma V_{\text{future}}$
6: Update *value_net* to minimize $(y_t - \textit{value\_net}(s_t))^2$
7: **end for**

3.3.3. Combination of Techniques

We utilize Torch's computational graph and optimization routines to update both the value and policy networks. Baseline subtraction reduces variance in policy updates, while

bootstrapping provides a mechanism for faster adaptation by using updated value estimates from subsequent states:

Algorithm 5 Bootstrapping and Baseline Subtraction

Initialize *policy_net* and *value_net*
Set optimizers for both networks
for each episode **do**
    Initialize state s
    while not episode end **do**
        Select action $a_t \sim \pi_\theta(a|s)$ from *policy_net*
        Execute a_t, observe reward r, next state s', and done d
        $V(s) \leftarrow$ *value_net*(s)
        $V(s') \leftarrow$ *value_net*(s')
        TD error $\delta \leftarrow r + \gamma V(s') \cdot (1 - d) - V(s)$
        Update *policy_net* using $\nabla_\theta \log \pi(a_t|s_t) \cdot \delta$
        Update *value_net* to minimize $(\delta)^2$
        $s \leftarrow s'$
    end while
end for

3.4. Other Hyperparameters

Learning Rate: We initially set both the policy (*policy_lr*) and critic (*critic_lr*) networks' learning rates to 0.0005 based on initial stability and performance. To explore the impact of learning rate adjustments to each network's sensitivity:

1. We held the critic's learning rate constant at 0.0005 and varied the policy network's learning rate through several trials $(0.0001, 0.0005, 0.00002, 0.0025)$, allowing us to assess how changes in the policy network's learning rate influence model performance.

2. Similarly, we fixed the policy network's rate at 0.0005 and adjusted the critic's learning rate using the same values, evaluating the training dynamics and stability.

Discount Factor::A discount factor (*gamma*) of 0.99 is used to prioritize immediate rewards over distant rewards, reflecting the typical preference for more immediate gains in reinforcement learning scenarios.

3.5. PPO[1]

We implement the PPO through the following steps:

1. Data Collection: Execute the current policy π_{old} to collect data about states, actions, and rewards over a fixed number of steps or episodes.

2. Advantage Calculation: Compute the advantage estimates $\hat{A}_t$ using GAE based on the data collected.

3. Optimization: Update the policy parameters θ by applying stochastic gradient ascent to maximize the clipped surrogate objective function. This step is typically repeated for several epochs over the same data batch to make efficient use of the collected experiences.

4. Critic Update: Simultaneously, update the parameters of the value function to minimize a suitable loss function, such as the squared-error loss between the predicted values and computed returns.

Some techniques are commonly used to improve the performance, such as normalizing advantage values to stabilize learning, adding entropy term in policy loss function to encourage exploration and using multiple episodes at each iteration to stablize learning. However, while designing the experiment, we decided not to apply any of these in order to observe the performance of the most simple version of PPO[2].

At last, We compare its performance with the best Actor-Critic algorithm at hands.

4. Results[3][4]

4.1. REINFORCE Ablation[5]

Figure 1 compares the effects of baseline subtraction and entropy regularization on the REINFORCE algorithm. It can be observed that the performance of the REINFORCE algorithm improved the most with the introduction of baseline subtraction. The high volatility observed during the training process of the REINFORCE algorithm without baseline subtraction indicates that the policy gradients used by REINFORCE suffer from high variance. In the REINFORCE algorithm, baseline subtraction defines the advantage function as the difference between the reward and a baseline value, thereby reducing the variance of the advantage function. This helps to more accurately estimate the quality of actions, leading to improved training efficiency and stability, as evidenced by the results shown in the figure.

From the figure, it can also be observed that the introduction

[1]We designed and experimented PPO for Actor-Critic, which could be our effort for bonus.

[2]We still implemented the code for the improved PPO, which could be found in our GitHub repository.

[3]Among all the graphs, the regions with some transparency represent the original reward data from 3 runs and the solid lines represent the smoothed average data.

[4]The solid lines in graphs indicate the average performance of the corresponding models shown in 1000 evaluation episodes.

[5]We had interests in the performance of REINFORCE without entropy regularization and REINFORCE with baseline substraction, thus designing an ablation experiment for it, which could be our effort for bonus.

of entropy regularization has slightly improved the performance of the algorithm. This is because entropy regularization increases the exploration level of the reinforcement learning algorithm, facilitating faster policy learning and reducing the likelihood of the policy getting stuck in poor local optima.

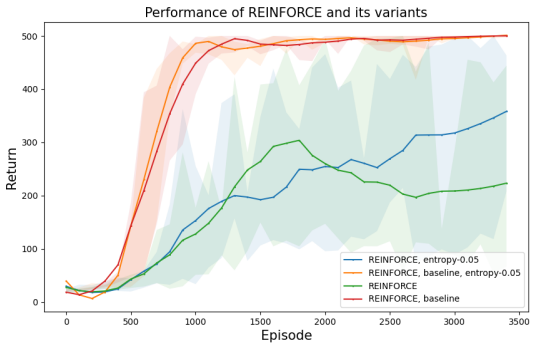

Figure 1. Effect of Baseline Subtraction and Entropy Regularization on REINFORCE Algorithm Performance

4.1.1. Adjustment of Entropy Regularization Coefficient

Figure 2 compares the effects of different entropy regularization coefficients on the performance of the REINFORCE algorithm with baseline subtraction. When the entropy regularization coefficient is small (e.g., 0.01 and 0.002), the agent may tend to exploit existing experience and reduce the selection of exploratory actions, resulting in lower performance during the exploration phase. Conversely, when the entropy regularization coefficient is large (e.g., 0.05), the agent is more inclined to take exploratory actions, facilitating the discovery of potential optimal strategies more quickly, thereby enhancing learning efficiency.

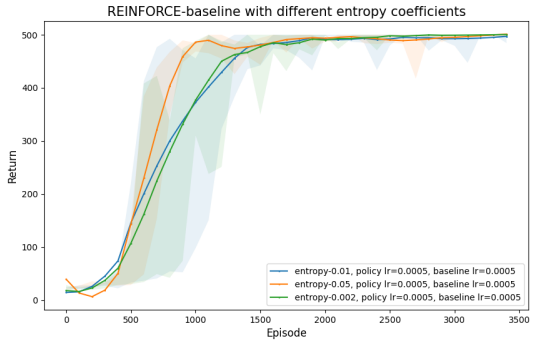

Figure 2. REINFORCE-baseline with different entropy coefficients

4.2. Actor-Critic Ablation

Figure 3 compares the effects of baseline subtraction and bootstrapping on the Actor-Critic algorithm. It can be observed that both bootstrapping and baseline subtraction can enhance the performance of the Actor-Critic algorithm. However, combining bootstrapping with baseline subtraction outperforms using either method alone. Comparing the variations of Actor-Critic with REINFORCE, it can be observed that Actor-Critic with Bootstrapping and baseline subtraction achieves the best performance.

Bootstrapping uses the current estimated value function to update both the policy and value function. This approach allows for updates within a single time step without waiting for the completion of full episodes or trajectories, thereby speeding up the learning process. However, Bootstrapping may lead to an increase in the variance of policy gradients because it utilizes the current estimated value function as the target value for the next state. This could introduce estimation errors, thereby increasing the variance of policy gradients. It can be observed that Bootstrapping exhibits significant volatility during the training process. Therefore, while Bootstrapping enhances learning efficiency, it also raises the risk of the algorithm getting trapped in local optima.

Baseline subtraction adjusts reward by subtracting a baseline value, thereby reducing the variance of the advantage function. This decreases the variance of policy gradients, resulting in more stable policy updates. Compared to Bootstrapping, Baseline subtraction provides greater stability in updating value functions but sacrifices some learning efficiency. By combining Bootstrapping with Baseline subtraction, their respective advantages can be leveraged to improve algorithm performance. Bootstrapping accelerates the learning process, enabling faster updates of policy and value functions, while Baseline subtraction reduces variance, enhancing stability.

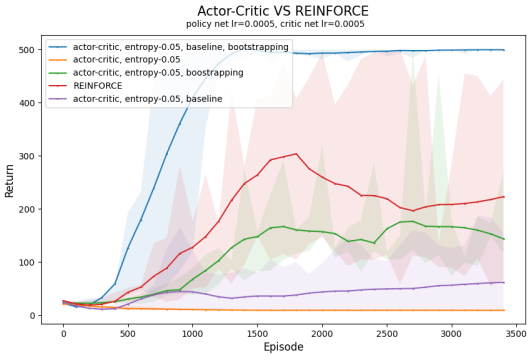

Figure 3. Effect of Baseline Subtraction and Bootstrapping on Actor-Critic Algorithm Performance

4.2.1. Adjustment of Entropy Regularization Coefficient

Figure 4 compares the effects of different entropy regularization coefficients on the performance of the Actor-Critic algorithm with baseline subtraction. It can be observed that when the entropy regularization coefficient is 0.01, the agent tends to converge to better policies more quickly during the learning process. This might be due to the fact that this coefficient provides sufficient exploratory actions without overly emphasizing exploration, thus better balancing the trade-off between exploration and exploitation.

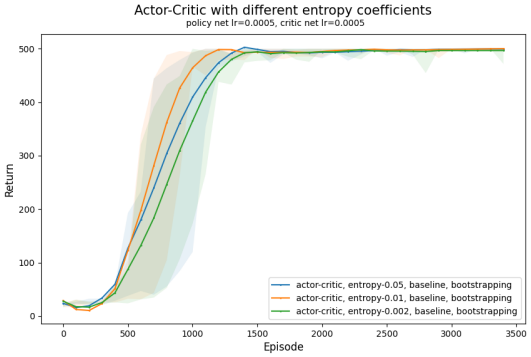

Figure 4. Actor-Critic with Different Entropy Coefficients

4.2.2. Adjustment of Learning Rate

Figure 5 compares the effects of different learning rates for the policy network parameters (with the value network parameters' learning rate set to 0.0005). Figure 6 compares the effects of different learning rates for the critic network parameters (with the value network parameters' learning rate set to 0.0005). It can be observed that as the learning rate increases, the algorithm converges to better policies more quickly. However, larger learning rates may lead to performance fluctuations. Selecting an appropriate learning rate can balance the trade-off between the convergence speed and stability of the algorithm.

4.3. Actor-Critic with PPO

Figure 7 displays the performance of the Actor-Critic algorithm utilizing PPO. While PPO exhibits faster convergence, its volatility is comparatively higher than that of the Actor-Critic algorithm incorporating baseline subtraction and bootstrapping. This could be attributed to the impact of PPO's clipping mechanism.

5. Analysis

In the comparative experiments, we observed different performances of reinforcement learning algorithms in solving

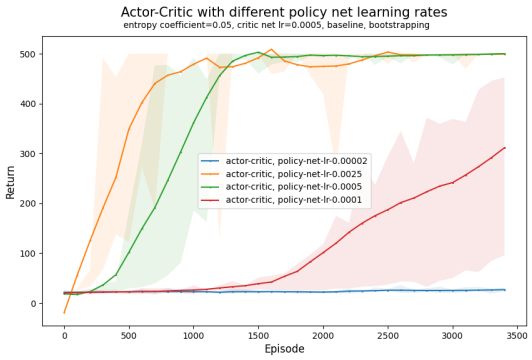

Figure 5. Actor-Critic with Different Policy Network Learning Rates

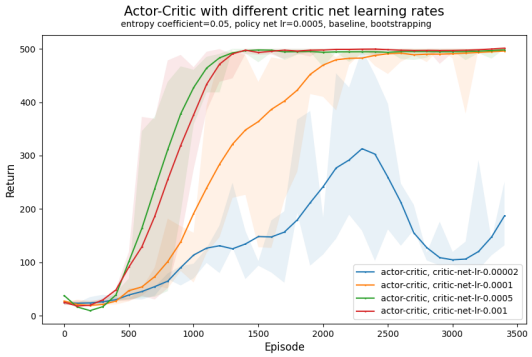

Figure 6. Actor-Critic with Different Critic Network Learning Rates

the Cartpole environment problem. The REINFORCE algorithm exhibited higher variance in the experiments, indicating that it may encounter significant fluctuations during training. The addition of baseline subtraction and entropy regularization significantly improved the performance of the REINFORCE algorithm. Baseline subtraction reduced the variance of the advantage function, thereby enhancing the stability and learning efficiency of the policy. Meanwhile, entropy regularization increased the possibility of exploration, accelerating the learning rate of the policy and reducing the risk of getting trapped in local optima.

In the Actor-Critic method, combining baseline subtraction with bootstrapping techniques can further enhance the algorithm's performance. Bootstrapping accelerates the learning process but may introduce some degree of estimation error, leading to an increase in the variance of policy gradients. On the other hand, baseline subtraction reduces the variance of the advantage function, improving the stability of policy gradients.

In the PPO algorithm, we observed that it converges faster but exhibits higher volatility, possibly due to the influence

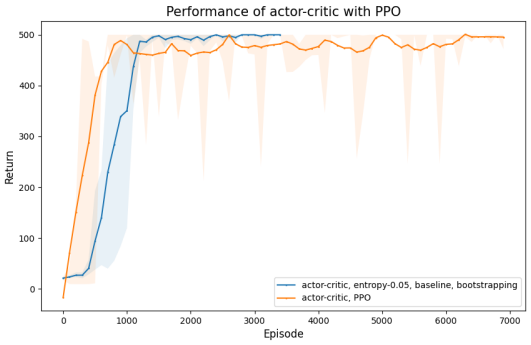

Figure 7. Performance of Actor-Critic with PPO

of the clipping mechanism in PPO.

Furthermore, we also observed that adjusting hyperparameters significantly impacts the algorithm's performance. In the experiments, we tried different learning rates and entropy regularization coefficient hyperparameters and observed their effects on algorithm performance. Proper hyperparameter selection can significantly improve the performance and stability of the algorithm.

6. Limitations

Environment Complexity: Our model heavily relies on the CartPole environment, which lacks the high degrees of uncertainty and complexity found in real-world scenarios. This may lead our models to overfitting on this specific task, which might not generalize well to more complex or varied environments.

Hyperparameter Sensitivity: The performance of the actor-critic method showed significant sensitivity to hyperparameters such as the learning rate and the entropy coefficient. Tuning these parameters required manual intervention and extensive experimentation, which may not be feasible in more complex scenarios or with limited computational resources.

7. Conclusion and Future Work

The experimental results from the implementation of the REINFORCE algorithm and Actor-Critic in the OpenAI Cartpole environment demonstrate the practical effectiveness and challenges of policy-based reinforcement learning methods.

While REINFORCE showed promising results in terms of learning stability and policy improvement, the addition of entropy regularization significantly enhanced exploration, mitigating the risk of early convergence to suboptimal policies. However, the study also uncovered limitations in scalability and sensitivity to hyperparameter settings, which could affect the broader applicability of these methods in more complex or dynamic environments.

As for Actor-critic algorithms, they stand out as a powerful tool in the arsenal of reinforcement learning techniques, capable of efficiently solving complex control tasks such as the Cartpole problem. The experimental results affirm their potential in handling environments where balancing exploration and exploitation is crucial. However, the effectiveness of these algorithms is heavily dependent on appropriate hyperparameter settings and the computational overhead involved in maintaining dual networks.

Future research should focus on several key areas to advance the understanding and capability of policy-based RL algorithms. First, investigating adaptive mechanisms for entropy regularization could provide more dynamic exploration strategies, potentially leading to faster convergence and improved policy robustness. Secondly, extending the current framework to multi-agent environments could offer insights into the cooperative and competitive dynamics that are prevalent in real-world scenarios. Additionally, the integration of meta-learning techniques could enhance the adaptability of policy-based methods, enabling them to perform well across a variety of tasks without extensive retraining. Finally, empirical studies involving larger and more diverse sets of environments would be invaluable in assessing the generalizability of the findings from this report and refining the algorithms for broad application.

8. Division of Labor

Three members of our group made equal contribution to this assignment.
Suju Li: Implemented the algorithms, conducted the experiments and contributed in 1, 2.4, and 7 of the report.
Ruiqing Sun: Contributed in 2.3, 3 and 6 of the report.
Peiwen Xing: Contributed in 2.1, 4 and 5 of the report.

References

T.Haarnoja, A.Zhou, K.Hartikainen, G.Tucker, S.Ha, J.Tan, V.Kumar, H.Zhu, A.Gupta, and P. Abbeel (2018). *Soft actor-critic algorithms and applications*, arXiv:1812.05905.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT Press.

Sutton, R. S., McAllester, D., Singh, S., & Mansour, Y. (2000). Policy Gradient Methods for Reinforcement Learning with Function Approximation. *Advances in Neural Information Processing Systems*.