EVALUATING ENVIRONMENTS WITH CONTINUOUS STATE SPACE WITH RL ALGORITHMS

Debanjan Mondal, Sriharsha Hatwar
(dmondal, shatwar) @ umass.edu

ABSTRACT

In this project, we have worked on identifying and implementing the baseline Reinforcement Learning algorithms such as Reinforce with baseline [Williams (2004)], One step actor critic method [Sutton & Barto (2018)] to evaluate the performances of environments with continuous state spaces such as Cartpole, Acrobot and Lunar lander using the OpenAI gym toolkit [Brockman et al. (2016)]. Additionally, we have also implemented the recently proposed policy gradient method, PPO (Proximal policy optimization)[Schulman et al. (2017b)] and have experimented on different environments and have explained our reasoning on the advantages of PPO over other previous methods.

1 INTRODUCTION

Research in reinforcement learning has become extremely important these days to automate and perform tasks without much of human supervision. Additionally, reinforcement learning is also being utilized in developing Language models that recently have taken the internet by storm. We in this project, explore the shortcomings, benchmark and discuss the findings of three various RL algorithms : Reinforce with baseline, One step actor critic method and PPO on environments with continuous state and discrete action spaces. The code for our project is available in this repo. We now will give a brief description about these environments.

1.1 CARTPOLE

Cartpole is one of the environments provide by the OpenAI gym. It consists of a cart with a pole that needs to be balanced. The goal of the agent is to balance the pole within a designated area and the pole should not have more than 12° to its starting point, else it would be considered as fallen . The action space of the environment involves moving left and right. The reward system is dense and for each time-step, the pole is balanced, the environment provides a +1 reward. As we are utlizing CartPole-V2, this is a finite horizon problem with a maximum time-step of 500. Hence, If the agent is able to get a reward of 500, it means the agent has won and the episode is terminated

1.2 ACROBOT

Acrobot is a Classic Control environments environment which was first introduced in Sutton & Barto (2018). The system consists of two links connected linearly to form a chain and the joint between the two is actuated. The goal of the agent is to make the free end of the actuated joint to reach a height by applying torque to the fixed point. Each of the torque application : -1 (-1 torque - towards left) 1 (0 torque to right) and 1 (1 torque - towards right) incurs a reward of -1 till the free end reaches the height. This is a finite horizon MDP where the number of steps in the episode is capped to 500. Hence, the minimum possible reward obtained per episode is -500. The goal of the RL algorithm would be maximize this reward by reaching the goal height with as minimum number of steps possible.

1.3 LUNAR LANDER

LunarLandar environment is a classic rocket trajectory optimization problem within the OpenAI Gym framework that simulates the task of landing a spacecraft on the moon's surface. The goal is to

land the module as gently as possible to earn points, balancing precise control with fuel efficiency. The lander starts at the top center of the viewport with a random initial force applied to its center of mass. The lander's state consists of its position, velocity, angular velocity along x and y axis. There are 4 possible actions: do nothing, fire main engine or fire any one of the side engines. For a successful landing, the agent achieves a +100 reward whereas crashing incurs a -100 reward. For each step, the agent receives a reward which is determined by proximity to lunar surface and penalized by firing engines. The episode terminates if the lander either lands or crashes, or goes outside of the viewport.

2 Intro to RL Algorithms

The algorithms in Reinforcement learning can be mainly distinguished between model based and model free algorithms. In the case of Model based, the algorithms builds an explicit model of the environment in order to plan and make decisions. Some of the examples involve Dynamic Programming, Monte Carlo tree search and the others. However, often times when the model of the environment is not available readily the algorithms sometimes rely on explicitly learning by simulating the environment, these are called Model free algorithms. The algorithms that we work on this project falls into the realm of Model free algorithms more particularly they belong to the special category of algorithms called as policy gradient methods.

3 REINFORCE with Baseline

3.1 Introduction

REINFORCE with baseline is a policy gradient method that learns a parameterized policy which helps in action selection instead of learning action values and then selecting actions depending on the value functions. The algorithm mainly depends on the fact that the expected return is proportional to its gradient. The learnt parameterized policy is advantageous where the state spaces are not restricted to be discrete for learning the optimal policy for an environment.

The Reinforce with baseline is a Monte carlo algorithm specifically applied for an episodic case. To reduce the variance and to improve the speed of convergence, the original REINFORCE algorithm has been generalized to include a comparison against a baseline. A baseline is any function that is independent of the action taken at a given time step and in this project, we are using the state value estimate $\hat{v}(S_t, w)$ as the baseline. The complete implementation of the algorithm is given below :

Algorithm 1 REINFORCE with Baseline

Input: a differentiable policy parametrization $\pi(a|s, \theta)$
Input: a differentiable state-value function parameterization $\hat{v}(s, \mathbf{w})$
Hyperparameters: Step sizes $\alpha^{\boldsymbol{\theta}}$, $\alpha^{\mathbf{w}}$, Discount factor γ
Initialize policy parameter $\boldsymbol{\theta}$ and state-value weights $\mathbf{w}$ arbitratrily.
for each episode **do**
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$ following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  for $t = 0$ **to** $T - 1$ **do**
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
    $\delta \leftarrow G - \hat{v}(s, \mathbf{w})$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla_{\hat{v}} \hat{v}(S_t, \mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\theta} \gamma^t \delta \nabla_{\theta} \log \pi(A_t|S_t, \theta)$
  end for
end for

3.2 Implementation Details

Implementation required us to chose the way we parameterize the policy and value evaluation and the clear choice for that was two different neural networks respectively. Reasons being that this helped in adapting the networks for the case of both continuous and discrete state spaces. Each of

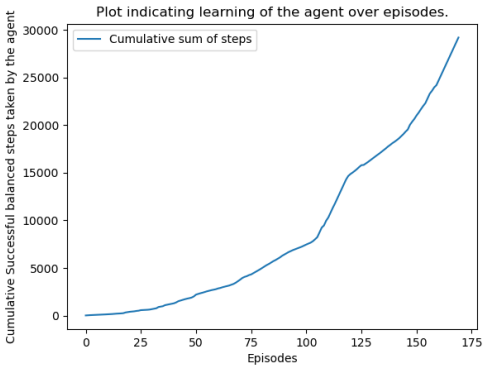


Fig1 : Cartpole - Steps per episode plot.

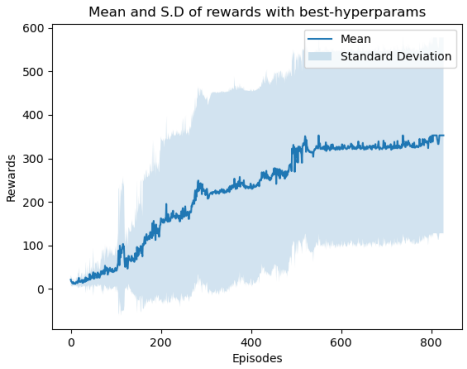


Fig2 : Cartpole - Mean and SD plots.

the neural network involved two intermediate layers with RELU activations. Adam optimizer was used for training the neural networks. The main difference between the two Neural networks is that the policy network will output "n" logits where n is the $|A|$ (Size of action space). However, the value network would output a single logit thereby trying to approximate the value function for a state representation. Except for Lunar lander (which took large number of episodes for convergence $\sim$ 20K), the hyper-parameter for the other two environments (Cartpole and Acrobot) has been tuned using Grid search. Now we will explain the implementation specifics for each of the environments.

3.3 CARTPOLE

For Cart-pole, both the value and policy network had two hidden layers with size 128 and 64 respectively. For the hyper-parameter tuning, we tuned with different learning rates. For policy network, the LR were tuned $(1e^{-2}, 1e^{-3}, 1e^{-}4)$ and for value network : $[0.1, 1e^{-3}, 0.01]$ with a fixed number of max episodes of 3k. It can be seen from the figures 1 that the reinforce with baseline was able to learn to balance as the episode progressed.

Below table is the best hyper-parameter that was used to train the model :

Parameters	**Values**
alpha_policy α^{θ}	1e-3
alpha_weights α^{w}	1e-2
Discount γ	0.99

After the best hyper-parameter was chosen, we ran the training 10 times and got the mean and Standard deviation of the reward per episode and it has been plotted in 2. The video of the model that was learnt being tested can be seen here : Cartpole-video

3.4 ACROBOT

For Acrobot, as the problem is similar in complexity of Cart-pole, we experimented with the same number of hidden layers and the sizes as of Cart-pole. For the hyper-parameter tuning, we tuned with different learning rates. For policy network, the LR were tuned $(1e^{-2}, 1e^{-3}, 1e^{-4})$ and for value network : $[0.1, 1e^{-3}, 1e^{-2}]$ with a fixed number of max episodes of 3k. However, due to the way the reward system has been built for acrobat, the agent was able to learn to achieve the goal / train properly 3/5 times when it was run with different seeds for the set of best hyper-parameters. The best hyper-parameters set is provided in the table below 3.4

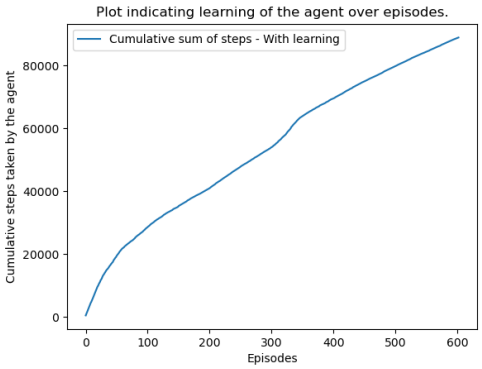


Fig3 : Acrobot - Steps per episode

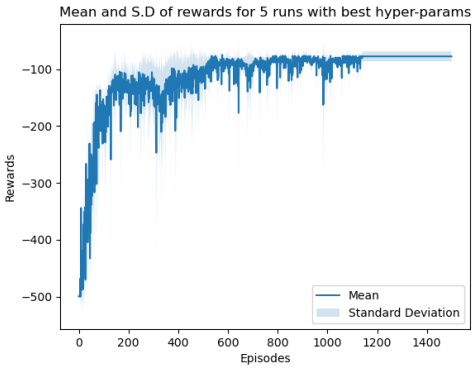


Fig4 : Acrobot - Mean and SD plots

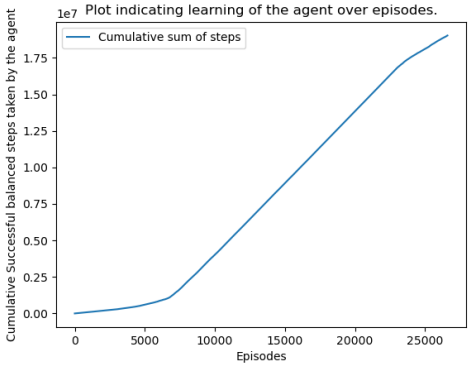


Fig5 : Lunar-lander: Steps per episode

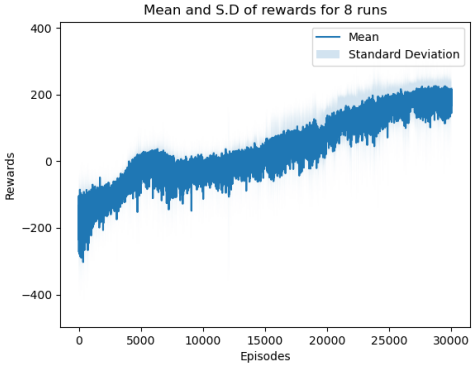


Fig 6 - Lunar-lander : Mean and SD plots

Parameters	Values
alpha_policy α^{θ}	1e-3
alpha_weights α^{w}	1e-2
Discount γ	0.99

Figure 3 depicts the way number of steps has been executed for a particular run of the acrobat environment. During the learning phase, as we can see, the number of steps taken per episode is reducing and hence the graph looks like its flattening, by this we can conclude that the networks were able to solve the acrobat problem as the training progressed. Figure 4 also depicts the rewards obtained as the training progresses for a set of 3 runs where the networks were able to get to a near optimal policy. The video of the learnt optimal policy can be seen here : Acrobot-link

3.5 LUNAR LANDER

For Lunar Lander, as the problem is slightly more complicated than the other two environments, we used a three layer neural network for value and policy representation. The layer sizes were 256, 128 and 64 respectively. The Learning rate was chosen to be $1e^{-4}$ for both policy and value networks. The maximum number of episodes were fixed to be 30K.

We chose a fixed set of hyper-parameter as stated in the table 3.5 due to the long duration of time taken by the Algorithm to learn the environment. Observing the figure, 5 we can state that as the training progressed, the number of steps started to increase and the agent was able to learn to utilize steps to land the lunar lander appropriately. Additionally, the figure 6 depicts the way rewards started to increase as the training progressed. The video of the agent being tested can be found here : Lunar-lander

Parameters	Values
alpha_policy α^{θ}	1e-4
alpha_weights α^{w}	1e-4
Discount γ	0.99

4 ONE-STEP ACTOR–CRITIC(EPISODIC) METHOD

The Actor-Critic methods represent a category of policy gradient approaches that extend beyond assigning state value only to the initial state; they also encompass the subsequent state in the transition. The key distinction between REINFORCE with a baseline and Actor-Critic lies in the substitution of the discounted return with the one-step return. This one-step return comprises the discounted estimated value of the second state, added to the reward. Similar to observations in TD learning, the one-step return often displays improved variance compared to the actual return, albeit it introduces a level of bias. Moreover, in this context, model updates can occur after each step without the necessity to wait until the episode's completion. When the state-value function is used to assess actions in this way it is called a *critic*, the policy network is deemed as an *actor*, overall the policy-gradient method is termed an *actor–critic* method.

Below is the outline of the algorithm used to learn One step actor critic method :

Algorithm 2 One-step Actor-Critic (episodic)

Input: a differentiable policy parametrization $\pi(a|s,\theta)$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Hyperparameters: Step sizes $\alpha^{\boldsymbol{\theta}}$, $\alpha^{\mathbf{w}}$, Discount factor γ
Initialize policy parameter $\boldsymbol{\theta}$ and state-value weights $\mathbf{w}$ arbitratrily.
for each episode **do**
  Initialize S (first state of episode)
  $I \leftarrow 1$
  while S is not terminal(for each timestep) **do**
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}}\delta\nabla_{\hat{v}}\hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\theta} I\delta\nabla_{\theta}\log\pi(A|S,\theta)$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$
  end while
end for

4.1 IMPLEMENTATION DETAILS

The *actor* network is similar to the policy network in 3.2 and the *critic* network is similar to value network. For both of them, we used fully connected layers with $(128, 64)$ hidden states and ReLU activations. Similar to 3.2 only the last fully connected layer is different between the two networks. Depending on the number of actions the output dimension of the last layer changes. We utilized Adam Optimizers with different learning rates based on the environment. Environment specific details are discussed in the subsequent sections. For hyperparameter tuning, empirically we observed that setting higher learning rate for critic results in better performance. We varied $\alpha^{\boldsymbol{\theta}}$ between 1e-2 and 1e-4 and $\alpha^{\mathbf{w}}$ between 1e-1 an 1e-3 with a fixed random seed and took the best set of hyperparameters for which the performance after 1000 episodes were maximized.

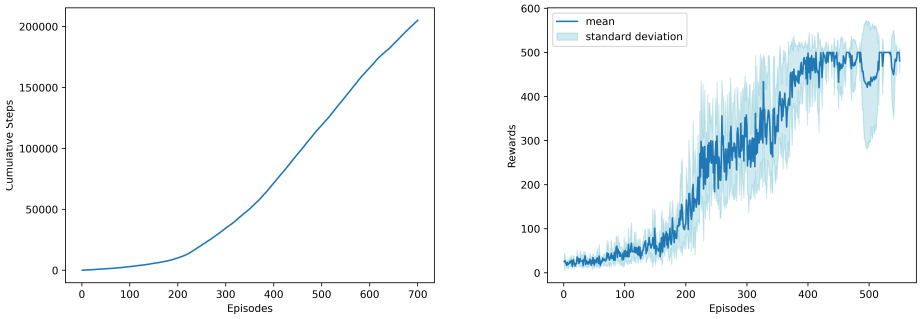
4.2 CARTPOLE

As presented in 2, we decay the learning rate for each step in the episode for the policy parameter $\alpha^{\boldsymbol{\theta}}$. For CartPole, we observed that decaying the value parameter $\alpha^{\mathbf{w}}$ in a similar manner helped stabilize the training performance. The learning curves have been presented in Figure 7. We notice

that the rewards exhibit lower variance, and the agent learns optimal behavior in fewer episodes(500) when compared to REINFORCE.

Here are best set of hyperparamaters we identified and used to train the model:

Parameters	Values
$\alpha^{\boldsymbol{\theta}}$	10^{-4}
$\alpha^{\mathbf{w}}$	10^{-3}
γ	0.99



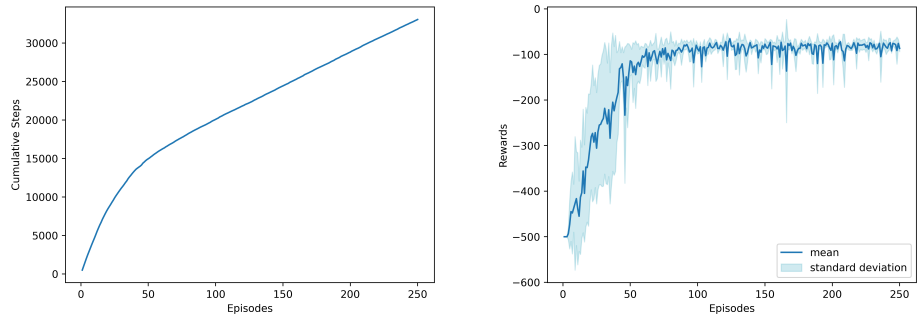
(a) Episodes vs Mean Cumulative Steps for 5 runs (b) Episodes vs Reward for 5 runs

Figure 7: Learning Curves for Actor Critic CartPole

4.3 ACROBOT

For Acrobot, the exact version of the algorithm worked pretty well. Remarkably, the actor-critic method achieved optimal behavior in the fewest number of episodes (250) compared to other algorithms. The optimal action in Acrobot is counterintuitive and requires the agent to cross the upper line randomly to learn about the desired action. The actor-critic approach, due to its parameter updates at each step, demonstrated swift adaptation to reward signals compared to alternative methods. The learning curves have been presented in Figure 8. Here are best set of hyperparamaters we identified and used to train the model:

Parameters	Values
$\alpha^{\boldsymbol{\theta}}$	10^{-4}
$\alpha^{\mathbf{w}}$	10^{-3}
γ	0.99



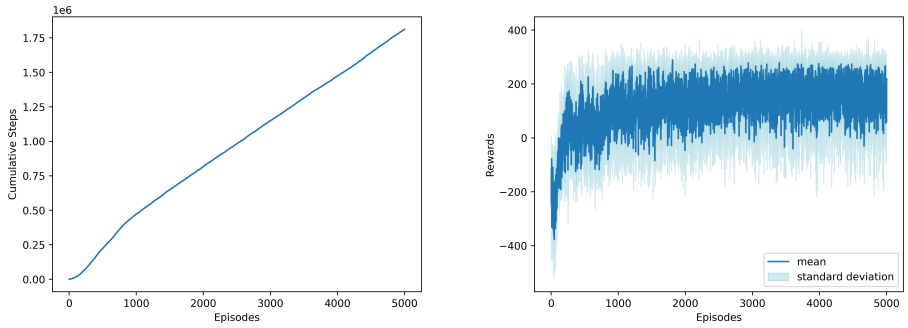
(a) Episodes vs Mean Cumulative Steps for 5 runs (b) Episodes vs Reward for 5 runs

Figure 8: Learning Curves for Actor Critic Acrobot

4.4 LUNAR LANDER

In the Lunar Lander environment, we observed improved outcomes by turning off the learning rate decay associated with steps, potentially influenced by the environment's larger horizon. Training the agent required a substantial duration, extending to 5000 episodes, yet the training process exhibited considerable noise, as evident in Figure 9. This noise could be attributed to the intricate and complex nature of the environment. The actor-critic method's frequent parameter updates in each step might lead to uncontrolled significant gradient steps, potentially causing instability in the learning process and directing it towards undesired paths. Here are best set of hyperparamaters we identified and used to train the model:

Parameters	Values
$\alpha^{\boldsymbol{\theta}}$	10^{-4}
$\alpha^{\mathbf{w}}$	10^{-3}
γ	0.99



(a) Episodes vs Mean Cumulative Steps for 5 runs (b) Episodes vs Reward for 5 runs

Figure 9: Learning Curves for Actor Critic Lunar Lander

5 EXTRA CREDIT: PROXIMAL POLICY OPTIMIZATION

Recently, Proximal Policy Optimization method has gained a lot of traction since its used to align large language models with human preference (Li et al., 2023). Policy Gradient methods are often unstable and could lead to either very large updates for some states or very small updates some other states. Trust Region Policy Approximation(TRPO) (Schulman et al., 2017a) addresses this issue by enforcing a strict KL-divergence constraint. Proximal Policy Optimization (PPO)(Schulman et al., 2017b) simplifies the process by adding a clipped surrogate objective function. We define $r_t(\theta) = \frac{\pi_\theta(s_t|a_t)}{\pi_{\theta_{old}}(s_t|a_t)}$. TRPO maximizes a "surrogate" objective:

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t \left[r_t(\theta) \hat{A}_t \right] \tag{1}$$

with a hard constraint on KL-divergence between $\pi(\theta)$ and $\pi(\theta_o ld)$ to avoid excessive policy update. PPO uses a simplified clipped surrogate objective:

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

This clipped objective prevents the model to take large catastrophic updates and stabilizes the training. The objective also includes the value function error term. This objective can further be augmented by adding an entropy bonus to ensure sufficient exploration. The overall objective becomes:

$$L_t^{CLIP+VF+S}(\theta) = \hat{E}_t[L_t^{CLIP}(\theta)] - c_1 \cdot L_t^{VF}(\theta) + c_2 \cdot S[\pi_\theta](s_t)$$

where c_1, c_2 are coefficients, and S denotes an entropy bonus, and L_t^{VF} is a squared-error loss In general, collecting a set of trajectories and then running batch gradient descent for some epochs

produces the best results. The PPO paper also discusses further optimization using Generalized Advantage Estimator (GAE), but due to time constraint, we used the non-truncated discounted return subtracted by the current state value estimate as the advantage estimator in our implementation. The implemented algorithm is presented in 3.

Algorithm 3 Proximal Policy Optimization (PPO)

Initialize policy parameters θ_0, initial value function parameters ϕ_0
Hypermarameters: number of epochs K, mini-batch size M, clipping parameter ϵ, discount factor γ, step sizes $\alpha^{\boldsymbol{\theta}}$, $\alpha^{\boldsymbol{\phi}}$
for $h = 0, 1, 2, \ldots$ **do**
  Collect set of trajectories $D_h = \{S_i, A_i, R_i, \pi_h(A_i|S_i)\}$ using $\pi_h = \pi(\theta_h)$ in the environment
  Calculate discounted return $G_i = \sum_{t=i+1}^{T} \gamma^{t-i-1} R_t$
  Create dataset $C = \{S_i, A_i, G_i, \pi_h(A_i|S_i)\}$
  for $k = 1$ **to** K **do**
    for every mini-batch B of size M in C **do**
      Compute advantages $A(s, a) = G - V_\phi(s)$
      $\pi_{\text{old}}(a|s) \leftarrow \pi_h(a|s)$
      Update policy by optimizing surrogate objective:
      $\theta \leftarrow \theta + \alpha^{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} \Bigg(\min \left(\frac{\pi_\theta(a|s)}{\pi_{\text{old}}(a|s)} A(s,a), \text{clip} \left(\frac{\pi_\theta(a|s)}{\pi_{\text{old}}(a|s)}, 1-\epsilon, 1+\epsilon \right) A(s,a) \right) + c_2 \cdot S[\pi_\theta](s_t) \Bigg)$
      $\phi = \phi + \alpha^{\boldsymbol{\phi}} A(s, a) \nabla_{\boldsymbol{\phi}} V_\phi(s)$
    end for
  end for
end for

5.1 Implementation Details

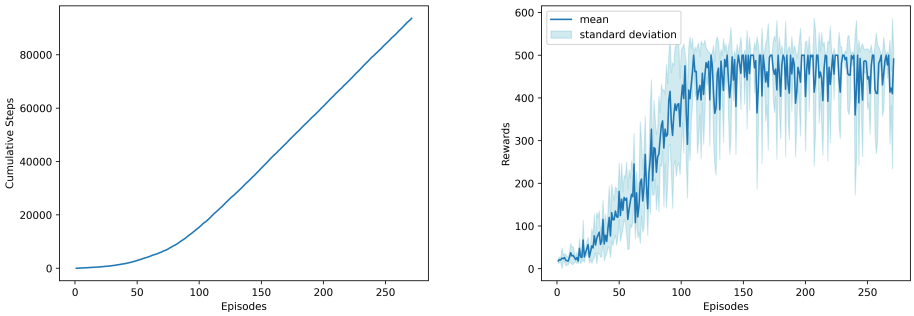
We utilized the same policy and value network as used in Section 4.1. Similar to REINFORCE and Actor-Critic, we applied Adam optimizers to update the parameters. Environment-specific details are discussed in the subsequent sections. One significant discovery was the stabilization of the training process by normalizing the advantage estimates, a technique we leveraged across all environments. However, hyperparameter tuning posed challenges due to the multitude of parameters involved. We set the clipping parameter $\epsilon = 0.2$, as mentioned in the original paper.

While collecting a set of trajectories, we ensured the minimum number of episodes reached a total timestep exceeding $4 \times$ horizon. Generally, maintaining a number of epochs $K = 10$ yielded satisfactory results. For the remaining parameters, we conducted a grid search for $\alpha^{\boldsymbol{\theta}} \in \left[10^{-2}, 10^{-4}\right]$ and $\alpha^{\boldsymbol{\phi}} \in \left[10^{-1}, 10^{-3}\right]$, along with a batch size $M \in \{2^5, 2^6, 2^7, 2^8\}$.

5.2 CartPole

PPO easily performs the best in this environment. It takes only 250 episodes to learn the optimal behavior. Furthermore, it demonstrates less variance when compared to alternative algorithms. The learning curves have been presented in Figure 10. Here are best set of hyperparamaters we identified and used to train the model:

Parameters	Values
$\alpha^{\boldsymbol{\theta}}$	10^{-4}
$\alpha^{\boldsymbol{\phi}}$	5×10^{-3}
γ	0.99
M	32
K	10
ϵ	0.2



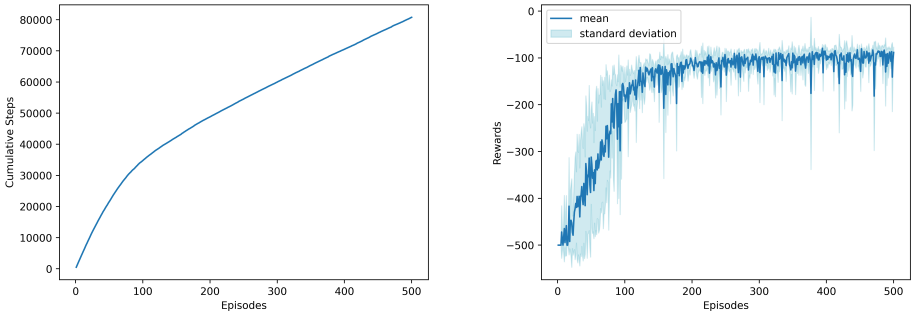
(a) Episodes vs Mean Cumulative Steps for 5 runs (b) Episodes vs Reward for 5 runs

Figure 10: Learning Curves for PPO CartPole

5.3 ACROBOT

In the case of Acrobot, PPO achieves optimal behavior in approximately 500 episodes, positioning itself between Reinforce and Actor Critic. However, PPO requires considerably less training time than Actor Critic due to its less frequent parameter updates. The learning curves have been presented in Figure 11. Here are best set of hyperparamaters we identified and used to train the model:

Parameters	**Values**
$\alpha^{\boldsymbol{\theta}}$	10^{-4}
$\alpha^{\boldsymbol{\phi}}$	5×10^{-3}
γ	0.99
M	256
K	10
ϵ	0.2



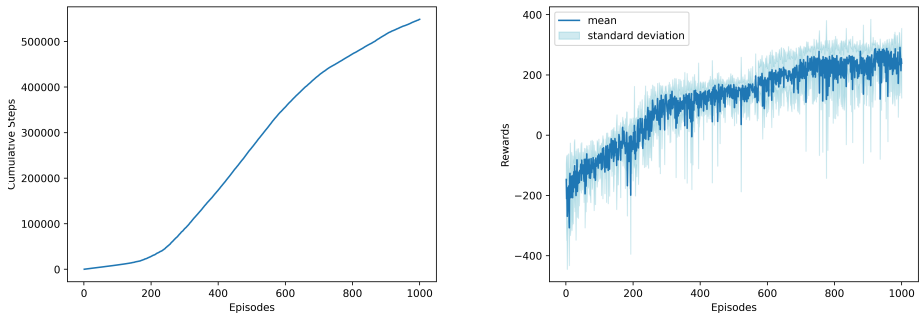
(a) Episodes vs Mean Cumulative Steps for 5 runs (b) Episodes vs Reward for 5 runs

Figure 11: Learning Curves for PPO Acrobot

5.4 LUNAR LANDER

Lunar Lander stands out as the most intricate environment in our exploration, and PPO surpasses other algorithms by a considerable margin within this setting. It achieves optimal behavior in just 1000 episodes, contrasting with the 5000 episodes required by the Actor Critic. Additionally, PPO demonstrates notably lower variance in rewards. The learning curves have been presented in Figure 12. Here are best set of hyperparamaters we identified and used to train the model:

Parameters	Values
$\alpha^{\boldsymbol{\theta}}$	10^{-4}
$\alpha^{\boldsymbol{\phi}}$	5×10^{-3}
γ	0.99
M	256
K	10
ϵ	0.2



(a) Episodes vs Mean Cumulative Steps for 5 runs

(b) Episodes vs Reward for 5 runs

Figure 12: Learning Curves for PPO Lunar Lander

6 Discussion and Comparison

Among the tested algorithms, Actor Critic achieved the best performance in the Acrobot environment, while PPO outperformed the others in all other environments. Tuning hyperparameters proved challenging for all algorithms, particularly for PPO. This hampered direct performance comparisons across different hyperparameter configurations due to both training time limitations and inherent randomness within the environments. Additionally, performance variations were observed across different random seeds, as visualized in the graphs. Notably, achieving the highest reward did not always translate to optimal behavior. In the case of Lunar Lander, while the agent achieved the optimal reward (200), it often continued firing engines even after landing successfully. Overall, all algorithms were successful in learning the optimal expected behavior for all environments.

References

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016. URL http://arxiv.org/abs/1606.01540.

Zihao Li, Zhuoran Yang, and Mengdi Wang. Reinforcement learning with human feedback: Learning dynamic choices via pessimism, 2023.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2017a.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017b. URL http://arxiv.org/abs/1707.06347.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 2004. URL https://api.semanticscholar.org/CorpusID:19115634.