COMPSCI 687 Final Project - Fall 2024

Bhanusree Ponnam and Sphurti Agarwal

Abstract

This report presents the implementation and evaluation of two reinforcement learning (RL) algorithms, REINFORCE with Baseline and One-Step Actor-Critic, on three Markov Decision Processes (MDPs): the Cat-vs-Monsters, Acrobot, and CartPole domains. Both algorithms were implemented from scratch in Python, utilizing the OpenAI Gym interface for the Acrobot and CartPole environments and Python for Cat-vs-Monsters. Experiments were conducted to tune hyperparameters, and performance was analyzed using learning curves.

1 Algorithms Implemented

1.1 REINFORCE with Baseline

The REINFORCE with Baseline algorithm is a Monte Carlo policy-gradient method that directly optimizes a parameterized policy for decision-making in episodic RL tasks. It builds upon the foundational REINFORCE algorithm (which uses complete return and updates are made after the episode is completed) by introducing a baseline, which is usually the state-value function. Baseline is added to reduce variance in the gradient estimates. This modification results in faster and more stable learning.

The main idea of the algorithm is to improve the efficiency of policy updates by subtracting a baseline value $V(s_t)$ from the return G_t, yielding $\delta_t = G_t - V(s_t)$. This value determines how much better or worse an action performed is, compared to the expected outcome. Importantly, the baseline does not introduce bias but significantly reduces variance in gradient estimates.

The flow of the Algorithm looks like:

1. The agent interacts with the environment, generating episodes that consist of sequences of states, actions, and rewards.
2. For each time step t in an episode, the total return G_t, which is the cumulative discounted reward, is computed.
3. The baseline $V(s_t)$, which our algorithm is learning using a value network, is subtracted from the return G_t to compute $\delta_t = G_t - V(s_t)$.
4. This value reflects the relative quality of an action compared to its expected value.
5. The policy is updated in the direction of the gradient of the logarithm of the policy, scaled by δ_t:

$$\theta \leftarrow \theta + \alpha_\theta \cdot \delta_t \cdot \gamma^t \cdot \nabla_\theta \log \pi_\theta(a_t \mid s_t).$$

6. This update encourages actions with higher δ_t, steering the policy towards actions yielding higher returns.
7. The baseline, which we have taken as the state-value function $V(s_t)$, is updated by minimizing the mean squared error between G_t and $V(s_t)$:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha_w \cdot \delta_t \cdot \nabla_{\mathbf{w}} V(s_t, \mathbf{w}),$$

where $\delta_t = G_t - V(s_t)$.

Pseudocode for the REINFORCE with Baseline Algorithm (1):

Algorithm 1 Reinforce with baseline

Require: a differentiable policy parameterization $\pi(a|s, \theta)$
Require: a differentiable state value function parameterization $\hat{v}(s, w)$
Algorithm Parameters: step sizes $\alpha_\theta \geq 0$, $\alpha_w \geq 0$
Initialize Policy Parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\boldsymbol{w} \in \mathbb{R}^d$
for each episode (Loop forever):
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 each step of the episode $t = 0, 1, 2, \ldots, T-1$:
$G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
$\delta \leftarrow G - \hat{v}(S_t, \boldsymbol{w})$
$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha_w \delta \nabla \hat{v}(S_t, \boldsymbol{w})$ ▷ gradient update for value network
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_\theta \gamma^t \delta \nabla \log \pi(A_t|S_t, \boldsymbol{\theta})$ ▷ gradient update for policy network
▷ (γ^t was dropped during implementation)
end for
end for

1.2 One-Step Actor-Critic

The **One-Step Actor-Critic Algorithm** is a policy-gradient method that leverages both the state-value function and the policy network to efficiently compute policy updates. Unlike the REINFORCE with Baseline algorithm, which uses the full return for episodic updates, this approach employs the *one-step return* for online and incremental learning.

The main idea of the algorithm is that it introduces a *critic*, represented by the state-value function $\hat{v}(s, w)$, which evaluates states and provides feedback to the *actor*—the policy network. The one-step return, computed as $G_{t:t+1} = R_{t+1} + \gamma \hat{v}(S_{t+1}, w)$, combines the immediate reward with the discounted estimate of the next state's value. Although this implies that this algorithm introduces bias, the one-step return here offers the benefits of reduced variance and greater computational efficiency. This return is then used to compute the temporal-difference (TD) error:

$$\delta_t = R_{t+1} + \gamma \hat{v}(S_{t+1}, w) - \hat{v}(S_t, w).$$

The TD error δ_t quantifies the difference between the expected and actual outcomes, guiding updates to both the actor and critic.

The flow of the Algorithm looks like:

1. The agent interacts with the environment in a sequential manner, observing states, selecting actions based on the policy, and receiving rewards.
2. For each state transition, the one-step return is computed as:
$$\delta_t = R_{t+1} + \gamma \cdot V(s_{t+1}, \mathbf{w}) - V(s_t, \mathbf{w}),$$
where $V(s_{t+1}, \mathbf{w})$ represents the estimated value of the next state.
3. This one-step return (δ_t) provides a measure of how good the action was compared to its expectation.
4. The policy is updated by adjusting the policy parameters θ in the direction of the gradient of the logarithm of the policy, scaled by δ_t:
$$\theta \leftarrow \theta + \alpha_\theta \cdot \delta_t \cdot \nabla_\theta \log \pi_\theta(a_t \mid s_t).$$
This update encourages the selection of actions that lead to higher rewards.
5. Simultaneously, the state-value function parameters $\mathbf{w}$ are updated using a TD(0) update rule to minimize the temporal difference error:
$$\mathbf{w} \leftarrow \mathbf{w} + \alpha_w \cdot \delta_t \cdot \nabla_{\mathbf{w}} V(s_t, \mathbf{w}).$$

6. These updates are performed incrementally and online, allowing the algorithm to process states, actions, and rewards in real-time as they are encountered.

The main distinction between One-Step Actor-Critic and REINFORCE with Baseline lies in the use of the one-step return. While the latter relies on the full return at the end of an episode, the former incorporates feedback from the next state S_{t+1}, enabling faster updates and improved stability at the cost of introducing bias.

Pseudocode for the One-Step Actor-Critic Algorithm (1):

Algorithm 2 One-step Actor-Critic

Require: a differentiable policy parameterization $\pi(a|s, \theta)$
Require: a differentiable state-value function parameterization $\hat{v}(s, w)$
  Parameters: step sizes $\alpha_\theta \geq 0$, $\alpha_w \geq 0$
  Initialize Policy Parameter $\theta \in \mathbb{R}^{d'}$ and state-value weights $w \in \mathbb{R}^d$
  for each episode (Loop forever):
    Initialize S (first state of episode)
    $I \leftarrow 1$
    for each time step; while S is not terminal state do:
      $A \sim \pi(\cdot|S, \theta)$
      Take action A, Observe S', R
      $\delta \leftarrow R + \gamma \cdot \hat{v}(S', \boldsymbol{w}) - \hat{v}(S, \boldsymbol{w})$
      $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha_{\boldsymbol{w}} \cdot \delta \cdot \nabla \hat{v}(S, \boldsymbol{w})$ ▷ gradient update for value or critic network
      $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_{\boldsymbol{\theta}} \cdot I \cdot \delta \cdot \nabla \ln \pi(A|S, \boldsymbol{\theta})$ ▷ gradient update for policy or actor network
      $I \leftarrow \gamma I$
      $S \leftarrow S'$
    end for
  end for

2 Cat-vs-Monsters Domain

2.1 Environment Description

Cat-vs-Monsters is a stochastic MDP where a cat navigates a grid to reach a food state while avoiding monsters and forbidden furniture. The environment dynamics and rewards are defined below:

- **States**: This domain consists of a 5×5 environment where each state $s = (r, c)$ describes the current coordinates/location of a cat. In particular, $r \in [0, 4]$ is the current row where the cat is located, and $c \in [0, 4]$ is the current column where the cat is located.
- **Actions**: There are four actions: AttemptUp (AU), AttemptDown (AD), AttemptLeft (AL), and AttemptRight (AR).
- **Dynamics**: This is a *stochastic* MDP:
  - With 70% probability, the cat moves in the specified direction.
  - With 12% probability, the cat gets confused and moves to the right with respect to the intended direction.
  - With 12% probability, the cat gets confused and moves to the left with respect to the intended direction.
  - With 6% probability, the cat gets sleepy and decides not to move.
  - The environment is surrounded by walls. If the cat hits a wall, it gets scared and does not move.

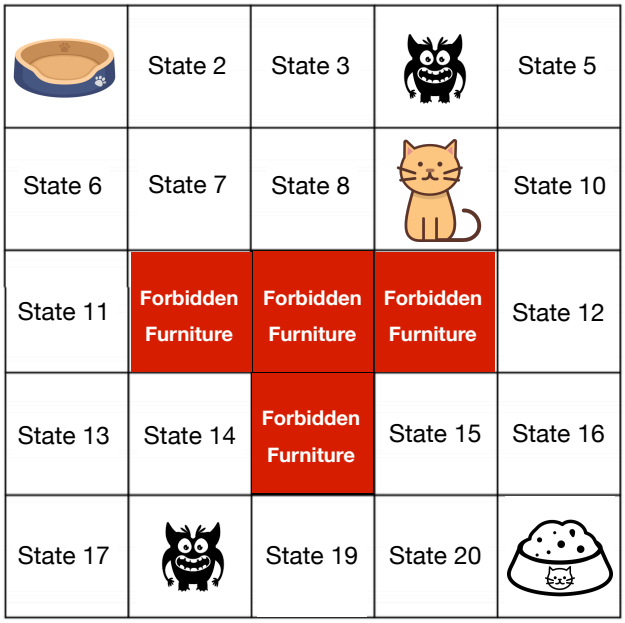


Figure 1: The Cat-vs-Monsters domain

  - There are four *Forbidden Furniture* locations in this environment: one in $(2, 1)$, one in $(2, 2)$, one in $(2, 3)$, and one $(3, 2)$. If the cat touches a Forbidden Furniture, it gets paralyzed and remains in its current state. The cat cannot go on the furniture.
  - There are two *Monsters*: one in $(0, 3)$ and one in $(4, 1)$.
  - There is a *Food state* located at $(4, 4)$.

- **Rewards**: The reward is always -0.05, except when transitioning to (entering) the Food state, in which case the reward is 10; or when transitioning to (entering) a state containing a Monster, in which case the reward is -8.
- **Terminal State**: The Food state is terminal. Any actions executed in this state always transition to s_∞ with reward 0.
- **Discount**: $\gamma = 0.925$.
- **Initial State**: The cat deterministically wakes up at the beginning of each episode on its bed; that is, $S_0 = (0, 0)$, always.

2.2 REINFORCE with Baseline

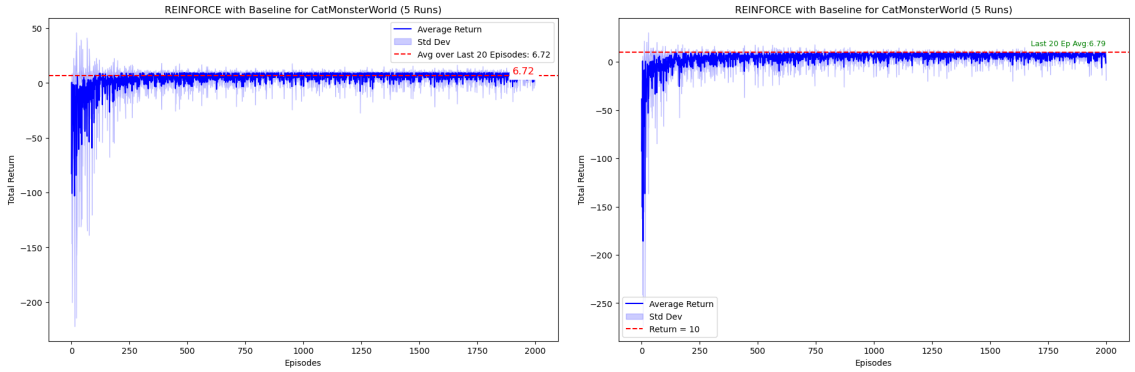
Experiments and Results

We ran experiments for 5 runs each with #episodes = 2000. Our plots and results report the total mean reward across those 5 runs for every episode. We utilized neural networks for both the policy and value (baseline) networks to implement the REINFORCE with Baseline algorithm. We used grid search for hyperparameter tuning and some of the learning curves for explanation are shown below.

Network Architecture: (Plots 2, 3)

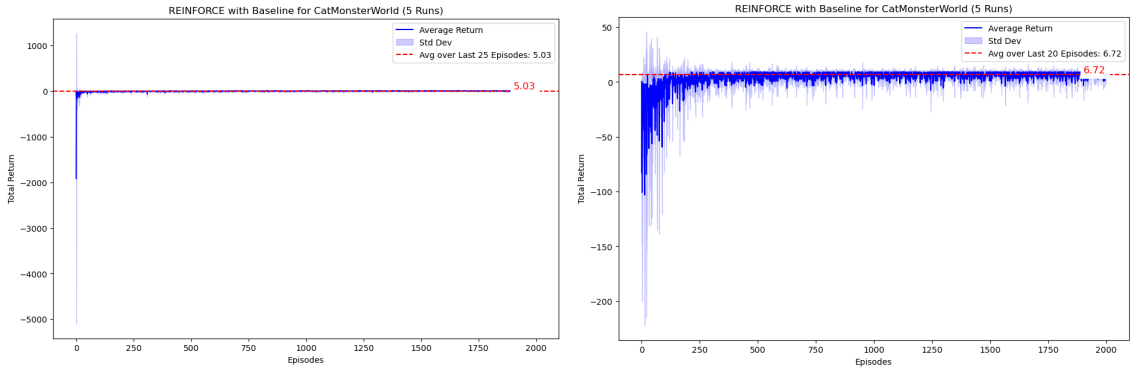
- We experimented with different numbers of **hidden units and layers** for both the **policy and value networks**. Initially, architectures with one and two hidden layers were tested.

- While the architecture with two hidden layers demonstrated slightly higher convergence values, the improvement was marginal. Considering the simplicity of the MDP, we opted for a single hidden layer to maintain simplicity and computational efficiency.
- Further experimentation with the number of hidden units in the single hidden layer revealed diminishing returns in the converged value as the number of units increased after a point. We observed that using 128 hidden units struck a good balance between performance and model complexity.
- The architecture with two hidden layers demonstrated slightly higher convergence values; however, the improvement was marginal. Additionally, using fewer hidden units resulted in lower variance in the returns but led to lower convergence values. Consequently, a higher number of hidden units was preferred to achieve better performance.



1 layer with 128 Neurons ($\alpha_\theta = 0.0001, \alpha_w = 0.01$) 2 layers with 128, 32 Neurons ($\alpha_\theta = 0.0001, \alpha_w = 0.01$)

Figure 2: # of Hidden Layers : 1 vs 2



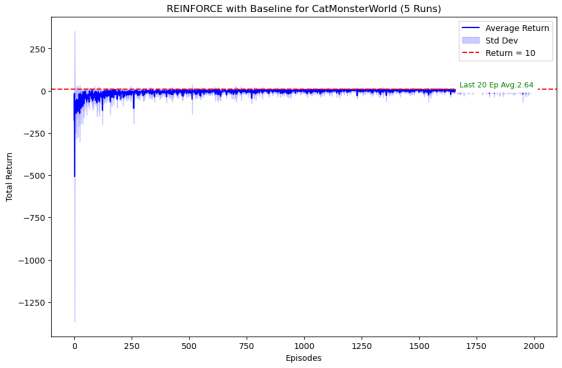
1 layer with 32 Neurons ($\alpha_\theta = 0.0001, \alpha_w = 0.01$) 1 layer with 128 Neurons ($\alpha_\theta = 0.0001, \alpha_w = 0.01$)

Figure 3: # of Hidden Units : 32 vs 128

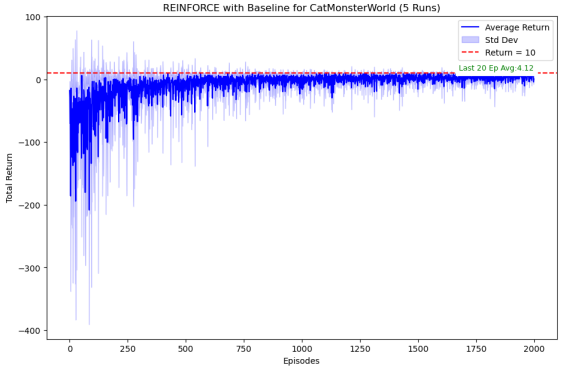
Learning Rates: (Plots 4, 5)

- Separate learning rates were tuned for the policy network (α_θ) and the value network (α_w).
- Through experimentation, we found that α_θ being smaller than α_w consistently yielded better performance and faster convergence.
- We explored α_θ in {0.001, 0.0001, 0.00001} and α_w in {0.1, 0.01, 0.001}.

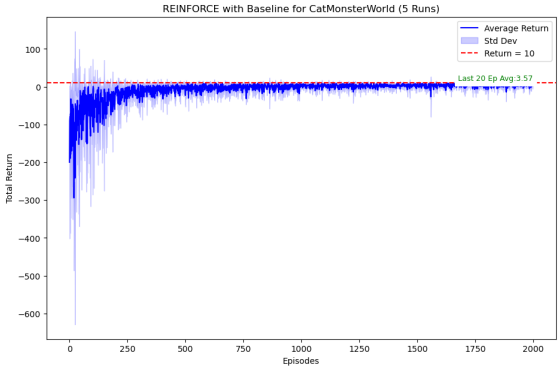
- Higher values of α_θ resulted in lower variance in the returns, but the variance was unstable. On the other hand, higher values of α_w led to lower variance but resulted in lower convergence values.
- The best convergence was achieved with $\alpha_\theta = 0.0001$ and $\alpha_w = 0.01$.



1 layer with 128 Neurons ($\alpha_\theta = 0.00001, \alpha_w = 0.1$)

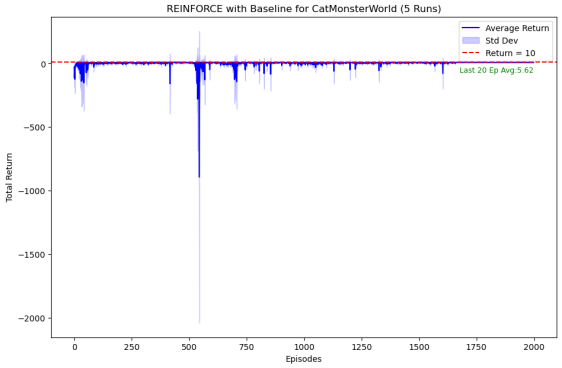


1 layer with 128 Neurons ($\alpha_\theta = 0.00001, \alpha_w = 0.01$)

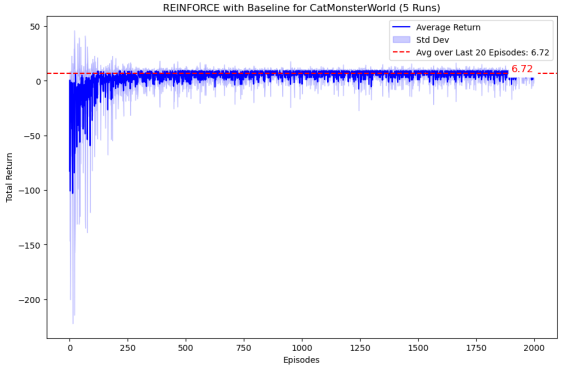


1 layer with 128 Neurons ($\alpha_\theta = 0.00001, \alpha_w = 0.001$)

Figure 4: $\alpha_w = 0.1\ vs\ 0.01\ vs\ 0.001$



1 layer with 128 Neurons ($\alpha_\theta = 0.001, \alpha_w = 0.01$)

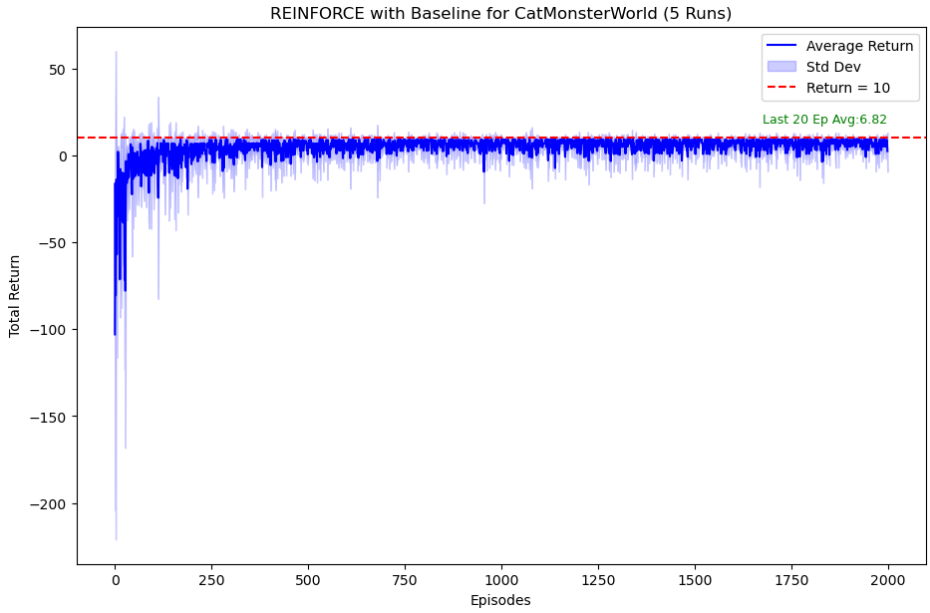


1 layer with 128 Neurons ($\alpha_\theta = 0.0001, \alpha_w = 0.01$)

Figure 5: $\alpha_\theta = 0.001\ vs\ 0.0001$

Overall:

- A single hidden layer with 120 hidden units provided a simpler yet effective architecture.
- Learning rate combinations of $\alpha_\theta = 0.0001$ and $\alpha_w = 0.01$ resulted in the most stable and efficient learning.



1 layer with 128 Neurons ($\alpha_\theta = 0.0001, \alpha_w = 0.01$)

Figure 6: Learning Curve for REINFORCE with Baseline for Cat vs Monster Domain

For Cat vs Monster Domain, the REINFORCE with Baseline algorithm converged to a policy with average return ˜ **6.82**.

2.3 One-Step Actor-Critic

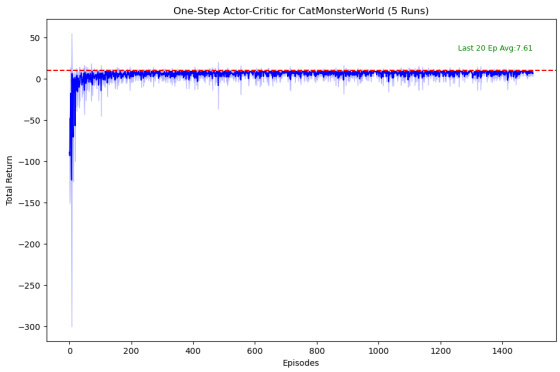
Experiments and Results for One-Step Actor-Critic

We ran experiments for 5 runs each with #episodes ranging 1000-2000. Our plots and results report the total mean reward across those 5 runs for every episode. We utilized neural networks for both the policy and value networks to implement the One-Step Actor-Critic algorithm. Hyperparameters were tuned using grid search and some of the learning curves for explanation are shown below.

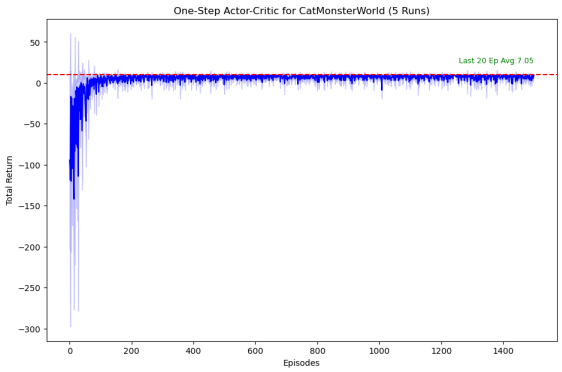
Network Architecture: (Plots 7, 8)

- Similar to REINFORCE with Baseline, we experimented with both one and two hidden layers for the policy and value networks. We observed that the one-layer architecture performed just as well, in fact slightly better in this case, than the two-layer architecture. This may have been due to the simplicity of the MDP and the computational benefits of using fewer layers. The performance was almost identical, so we opted for the simpler one-layer configuration.
- We also experimented with different numbers of hidden units: 32, 64, and 128. As with REINFORCE with Baseline, we found diminishing returns in performance as the number of hidden units increased after 128. So, we chose 128 hidden units, which provided a good balance between model complexity and performance.

- As in REINFORCE with Baseline, using fewer hidden units sometimes resulted in higher variance in the returns and sometimes lower, but the convergence value was lower. Hence, a higher number of hidden units, particularly 128, was preferred to achieve better performance.

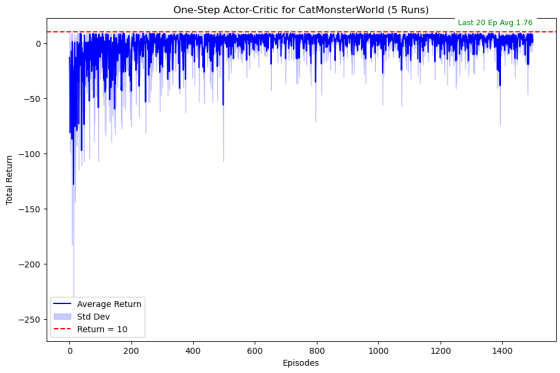


1 layer with 128 Neurons ($\alpha_\theta = 0.0001, \alpha_w = 0.01$)

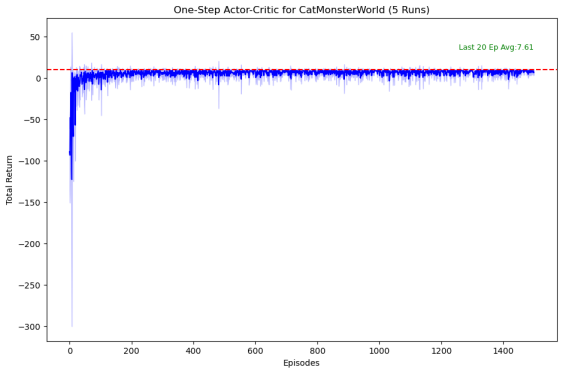


2 layers with 128, 32 Neurons ($\alpha_\theta = 0.0001, \alpha_w = 0.01$)

Figure 7: # of Hidden Layers: 1 vs 2



1 layer with 64 Neurons ($\alpha_\theta = 0.001, \alpha_w = 0.01$)

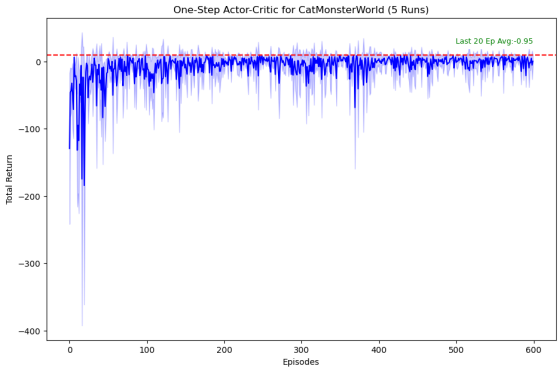


1 layer with 128 Neurons ($\alpha_\theta = 0.0001, \alpha_w = 0.01$)

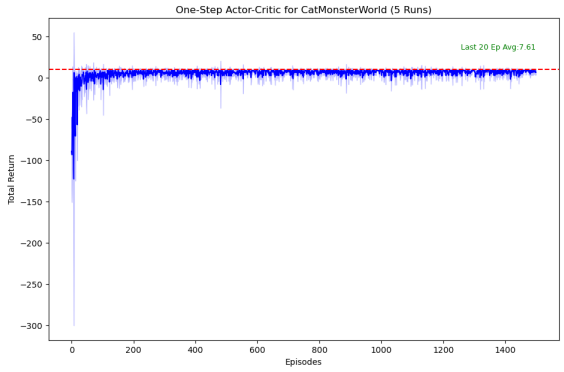
Figure 8: # of Hidden Units: 64 vs 128

Learning Rates: (Plots 10, 9)

- As in REINFORCE with Baseline, separate learning rates were tuned for the policy network (α_θ) and the value network (α_w).
- Because our experiments for REINFORCE with baseline algorithm revealed that α_θ smaller than α_w consistently resulted in better performance and faster convergence, so we set our search space for α_θ and α_w accordingly.
- We explored α_θ values in {0.001, 0.0001, 0.00001} and α_w values in {0.1, 0.01, 0.001}.
- Higher values of α_θ resulted in lower variance in returns but showed convergence to slightly lower returns. On the other hand, higher values of α_w led to high variance in this case and convergence to lower return values.
- The optimal convergence was achieved with $\alpha_\theta = 0.0001$ and $\alpha_w = 0.01$, consistent with our findings from REINFORCE with Baseline.

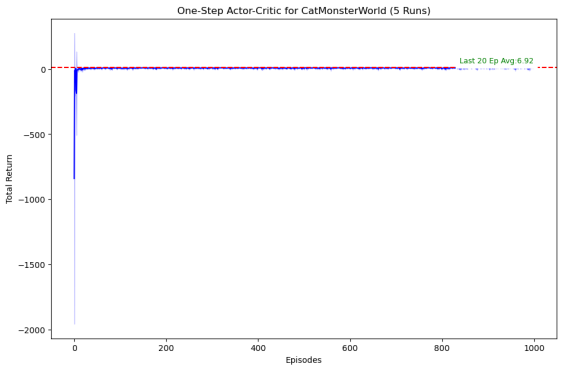


1 layer with 128 Neurons ($\alpha_\theta = 0.0001, \alpha_w = 0.1$)

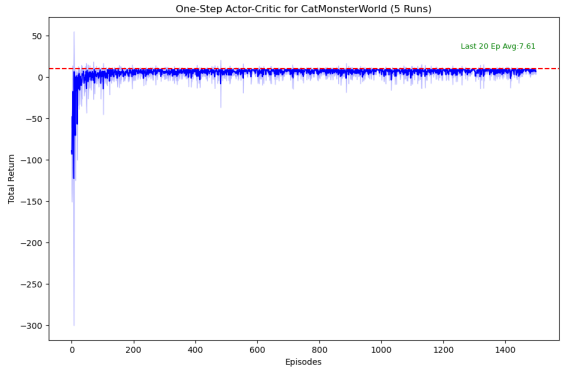


1 layer with 128 Neurons ($\alpha_\theta = 0.0001, \alpha_w = 0.01$)

Figure 9: Learning Rate Effect for $\alpha_w = 0.1\, vs\, 0.01$



1 layer with 128 Neurons ($\alpha_\theta = 0.001, \alpha_w = 0.01$)

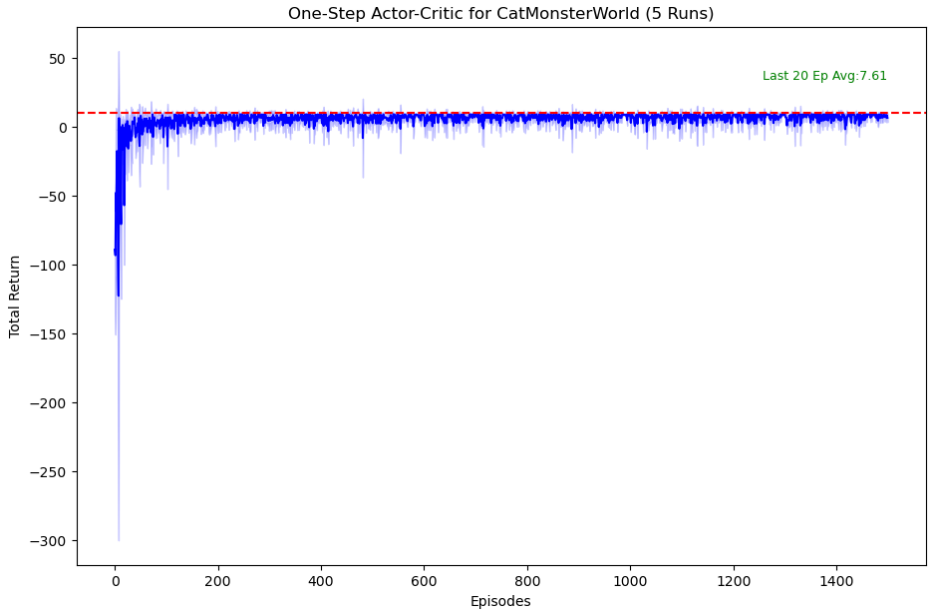


1 layer with 128 Neurons ($\alpha_\theta = 0.0001, \alpha_w = 0.01$)

Figure 10: Learning Rate Effect for $\alpha_\theta = 0.001\, vs\, 0.0001$

Overall:

- A single hidden layer with 128 hidden units was found to be the most effective architecture for the One-Step Actor-Critic algorithm.
- The best learning rate combination was $\alpha_\theta = 0.0001$ and $\alpha_w = 0.01$, providing the most stable and efficient learning curve.



1 layer with 128 Neurons ($\alpha_\theta = 0.0001, \alpha_w = 0.01$)

Figure 11: Learning Curve for One-Step Actor-Critic for Cat vs Monster Domain

For the Cat vs Monster Domain, the One-Step Actor-Critic algorithm converged to a policy with an average return of **7.61**.

Both algorithms achieved high returns, with One-Step Actor-Critic reaching a slightly higher return of 7.61 compared to REINFORCE with Baseline's 6.82 and with less variability. This could be due to One-Step Actor-Critic's use of Temporal Difference learning and more frequent updates, which may be leading to faster and more stable convergence. The delta function and smaller, more consistent updates possibly help reduce variance, contributing to a slightly higher return.

3 Acrobot Domain

3.1 Environment Description

The Acrobot problem is a classical Reinforcement Learning challenge designed to evaluate algorithms for learning complex motor control tasks. It is modeled as a two-link pendulum system connected by a joint, where the objective is to apply torque on the joint to swing the system until the free end reaches a specified target height (Figure 12) (Reference : (2)).

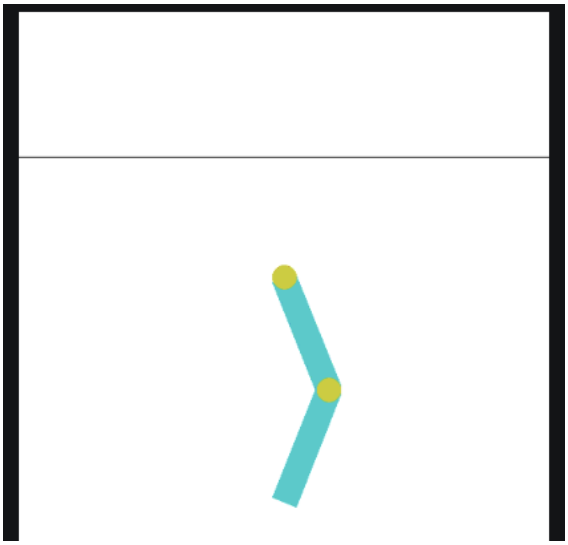


Figure 12: Acrobot

- **States**: The state of the system is represented by six variables, which describe the positions and velocities of the two links. Specifically:
  - θ_1: The angle of the first joint (measured relative to the downward vertical direction).
  - θ_2: The angle of the second joint relative to the first.
  - $\dot{\theta}_1$: The angular velocity of the first joint.
  - $\dot{\theta}_2$: The angular velocity of the second joint.

  These states are represented by the following six values: $\cos(\theta_1)$, $\sin(\theta_1)$, $\cos(\theta_2)$, $\sin(\theta_2)$, $\dot{\theta}_1$, and $\dot{\theta}_2$.

No.	Variable	Range
1	Cosine of θ_1	[-1, 1]
2	Sine of θ_1	[-1, 1]
3	Cosine of θ_2	[-1, 1]
4	Sine of θ_2	[-1, 1]
5	Angular velocity of θ_1	[-4π, 4π]
6	Angular velocity of θ_2	[-9π, 9π]

Table 1: State Representation of Acrobot

- **Actions**: The action space is discrete, consisting of three possible actions: applying torque values of -1, 0, or 1 on the joint connecting the two links.
- **Dynamics**: The objective is to apply torques to raise the free end of the chain to a specified height. At each step:
  - A constant reward of -1 is applied for steps that do not lead to the target height.
  - Reaching the target height results in a reward of 0, and the episode terminates.
  - The episode terminates after 500 steps if the target height is not reached.
- **Rewards**: The system is rewarded as follows:
  - A constant negative reward of -1 is applied for each step, except when the goal state is reached.
  - Reaching the target height provides a reward of 0 and terminates the episode.

- **Initial State**: The initial state parameters θ_1, θ_2, and their angular velocities are initialized randomly between -0.1 and 0.1, with both links initially pointing downward.
- **Terminal State**: The episode terminates when the free end reaches the target height, which is defined by the condition:
$$-\cos(\theta_1) - \cos(\theta_1 + \theta_2) \geq 1$$
or if the episode length reaches 500 steps.
- We have used OpenAI gym module to implement this environment.

3.2 REINFORCE with Baseline

Experiment Results and Analysis:

In our experiments on the Acrobot domain, we systematically tested different configurations of the policy and value networks, focusing on network architecture, hidden layer sizes, and learning rates to determine the most effective setup. The key findings are summarized below.

1. Network Architecture:

- **Number of Hidden Layers: 1, 2**
We experimented with both one and two hidden layers for the policy and value networks. Our results indicated that using a single hidden layer led to faster convergence. This could be because the simpler one-layer network avoided the complexity and instability that can arise with deeper architectures, which may struggle with the high-dimensionality of the Acrobot environment.
- **Hidden Unit Sizes: 64, 128, and 256**
We tested various hidden unit sizes: 64, 128, and 256. The network with 128 hidden units exhibited the smoothest and fastest convergence, outperforming both the smaller (64) and larger (256) configurations. This could possibly be because 128 hidden units provided a balance between the model's capacity to learn complex patterns and the computational efficiency, avoiding the risk of overfitting or underfitting seen in the other configurations.

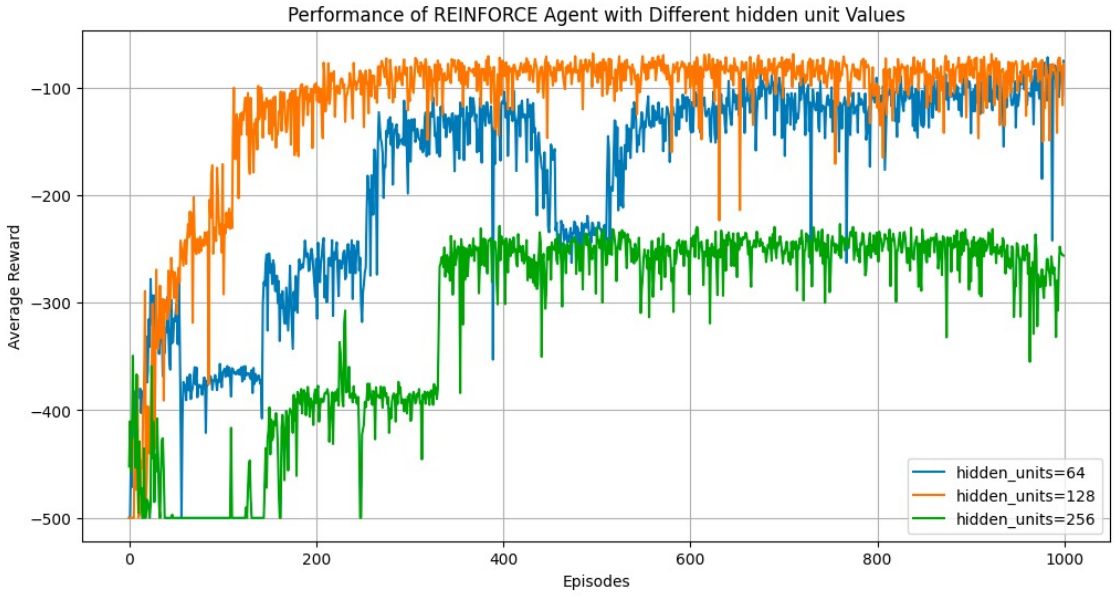


Figure 13: Hidden Unit Sizes: 64, 128, and 256

2. Learning Rate for α_θ (Policy Network)

We tested α_θ values of 10^{-3}, 10^{-4}, and 10^{-5}. The learning rate of $\alpha_\theta = 0.001$ resulted in the fastest convergence and the highest return with lower variance compared to the other rates. This indicates that a moderately higher learning rate enabled the model to efficiently update the policy weights without overshooting the optimal solution, striking the best balance between stability and learning speed.

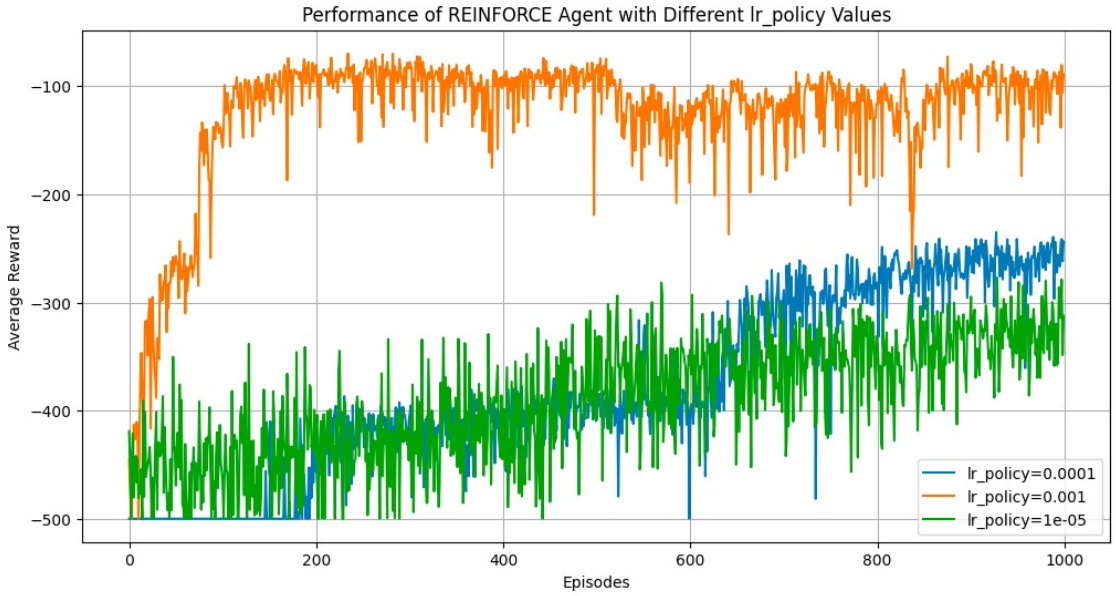


Figure 14: α_θ values $\in 10^{-3}, 10^{-4}, 10^{-5}$

3. Learning Rate for α_w (Value Network)

We tried values of α_w in $\{10^{-1}, 10^{-2}, 10^{-3}\}$. The learning rate $\alpha_w = 0.01$ achieved the highest return, although all values showed periods of convergence and high variance.

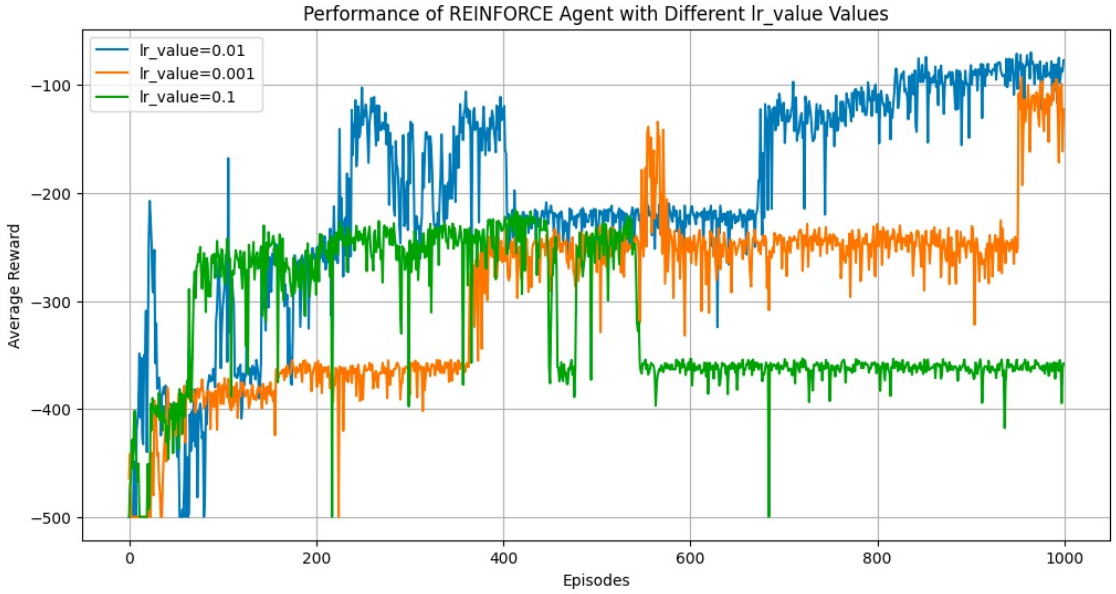


Figure 15: α_w values $\in 10^{-1}, 10^{-2}, 10^{-3}$

4. Final Configuration

Based on our analysis, the optimal configuration for REINFORCE with Baseline on the Acrobot domain was a network with **one hidden layer**, **128 hidden units**, $\alpha_\theta = 0.001$, and $\alpha_w = 0.01$. This setup resulted in the fastest convergence and the highest return, with manageable variance, making it the most robust choice for the problem.

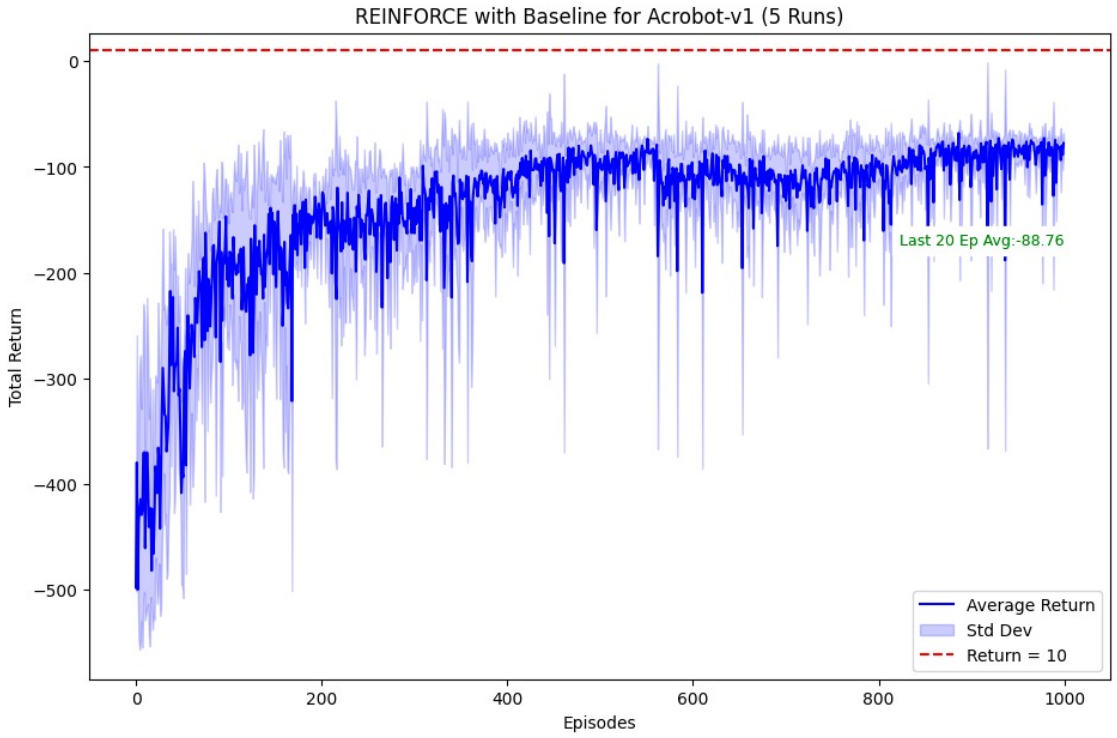


Figure 16: Learning Curve for REINFORCE with Baseline for Acrobot Domain

For Acrobot Domain, the REINFORCE with Baseline algorithm converged to a policy with average return ˜ **-88.76**.

3.3 One-Step Actor-Critic

Experiment Results and Analysis:

In our experiments on the Acrobot domain using the One-Step Actor-Critic algorithm, we conducted a series of tests varying the network architecture, hidden unit sizes, and learning rates to find the optimal configuration. The key observations are summarized below.

1. Network Architecture:

- **Number of Hidden Layers: 1, 2**
  We experimented with both one and two hidden layers for the policy and value networks. Similar to the REINFORCE with Baseline approach, using a single hidden layer led to faster convergence. The simpler architecture seemed more stable for the Acrobot problem, where deeper networks may not efficiently handle the high-dimensional state space.

- **Hidden Unit Sizes: 64, 128, and 256**
  We tested configurations with 64, 128, and 256 hidden units. The 256-hidden unit configuration

demonstrated the best performance in terms of convergence speed and stability, slightly outperforming the 128-unit setup. The larger network size allowed for more capacity to model the complex relationships in the Acrobot domain, while avoiding the computational inefficiencies that would arise from even larger configurations.

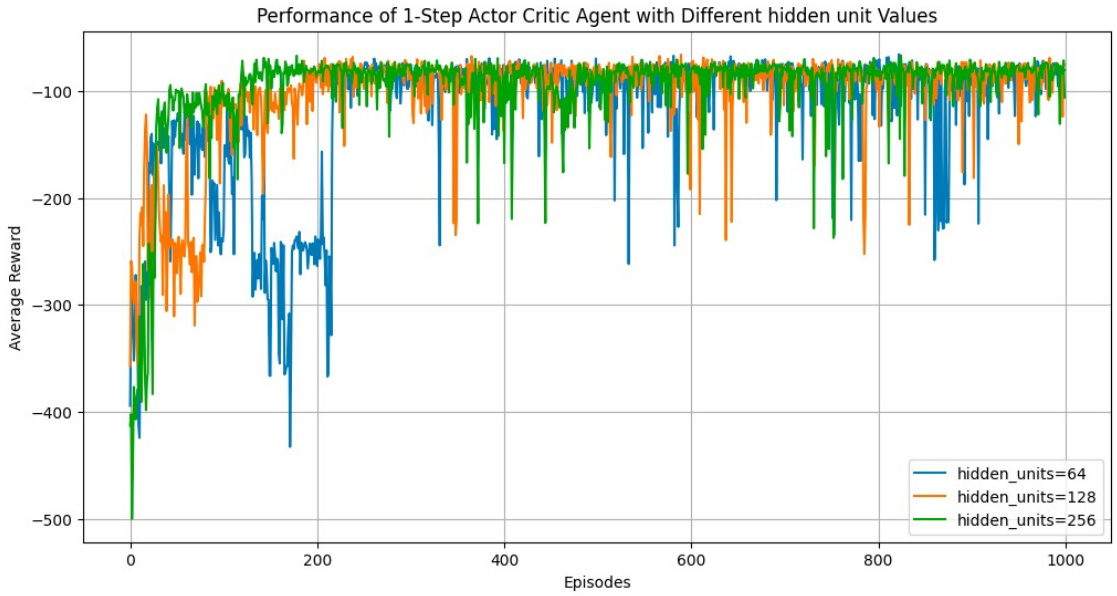


Figure 17: Hidden Unit Sizes: 64, 128, and 256

2. Learning Rate for α_θ (Policy Network)

We evaluated three values for α_θ: 10^{-3}, 10^{-4}, and 10^{-5}. A learning rate of $\alpha_\theta = 0.001$ resulted in the fastest convergence with the highest return and lowest variance. This suggests that a moderate learning rate allows for quicker learning without overshooting, offering a good trade-off between stability and learning speed.

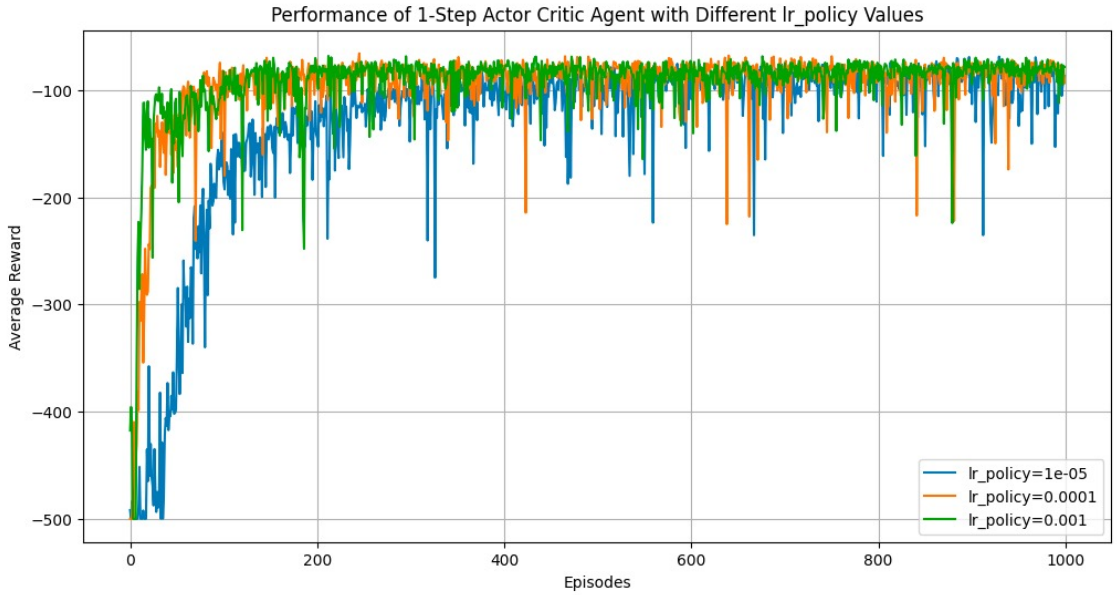


Figure 18: α_θ values $\in 10^{-3}, 10^{-4}, 10^{-5}$

3. Learning Rate for α_w (Value Network)

We tested values for α_w in $\{10^{-1}, 10^{-2}, 10^{-3}\}$. The learning rate $\alpha_w = 0.01 \,\&\, 0.001$ resulted in higher returns and smooth convergence. Ultimately, $\alpha_w = 0.001$ provided the best convergence quality and return.

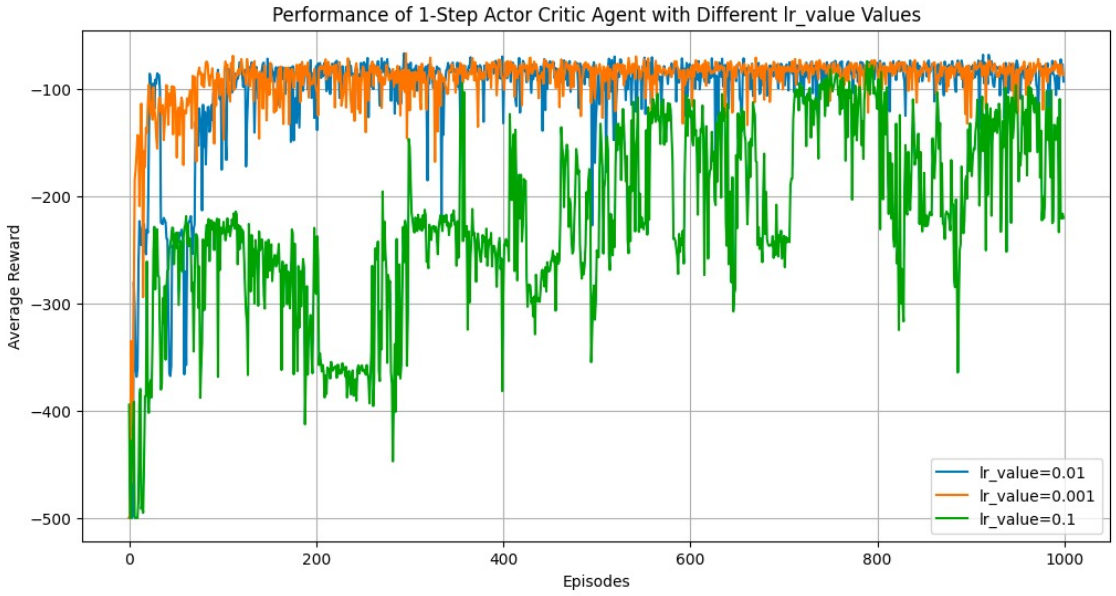


Figure 19: α_w values $\in 10^{-1}, 10^{-2}, 10^{-3}$

4. Final Configuration

Based on the results, the optimal configuration for One-Step Actor-Critic on the Acrobot domain was a network with **one hidden layer**, **256 hidden units**, $\alpha_\theta = 0.001$, and $\alpha_w = 0.001$. This configuration achieved the fastest convergence with the highest return while maintaining manageable variance, making it the most robust choice for solving the Acrobot task.

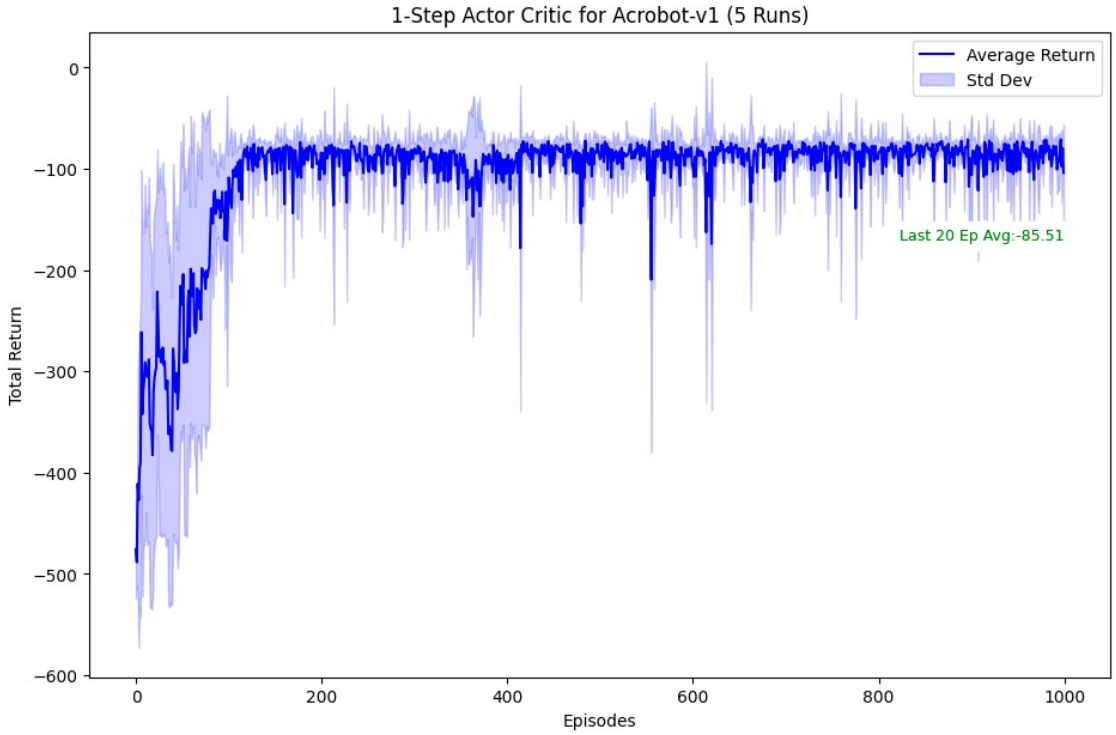


Figure 20: Learning Curve for One-Step Actor-Critic on Acrobot Domain

For the Acrobot domain, the One-Step Actor-Critic algorithm converged to a policy with an average return of **-85.51**, outperforming the REINFORCE with Baseline algorithm (**-88.76**) with lower variance. This can be attributed to the One-Step Actor-Critic's ability to update both the policy and value function more efficiently in each step, leading to faster learning and more stable convergence compared to the REINFORCE with Baseline, which relies on larger, less frequent updates.
Both algorithms achieved high returns, with One-Step Actor-Critic reaching a slightly higher return of **-85.51** compared to REINFORCE with Baseline's **-88.76**, along with lower variance. This could be due to One-Step Actor-Critic's use of Temporal Difference learning and more frequent updates, allowing for faster and more stable convergence. The delta function in the update rule, along with smaller, more consistent updates, likely helped reduce variance, contributing to a slightly higher return and more reliable performance in the Acrobot domain.

4 Cartpole Domain

4.1 Environment Description

The CartPole environment consists of a pole attached to a cart, and the goal is to balance the pole by applying a force to move the cart either left or right. The task involves controlling the cart's movement to prevent the pole from falling, which requires careful application of forces.

State Representation: The state is represented by a four-dimensional vector:

$$[x, v, \theta, \dot{\theta}]$$

where:

- x is the cart's position,

- v is the cart's velocity,
- θ is the pole's angle (with respect to the vertical),
- $\dot{\theta}$ is the pole's angular velocity.

Actions: The agent can choose from two discrete actions:

- Action 0: Apply a force to push the cart to the left.
- Action 1: Apply a force to push the cart to the right.

Rewards: The agent receives a reward of +1 for each time step the pole remains balanced, with the maximum possible reward being 500. The episode terminates at time step 500.
Initial State Distribution: The initial values for x, v, θ, and $\dot{\theta}$ are uniformly sampled from the range $(-0.05, 0.05)$.
Terminal State: The episode terminates when any of the following conditions are met:

- The pole's angle exceeds $\pm 12^\circ$ (0.209 radians),
- The cart position exceeds the boundaries of the track, i.e., $|x| \geq 2.4$,
- The episode length exceeds 500 time steps.

For this project, we utilized the OpenAI Gym's CartPole-v1 environment, which provides a standard implementation of this task.

Action Space: The action space is discrete with two possible actions:

- 0: Push the cart to the left,
- 1: Push the cart to the right.

State Space: The state space is continuous and consists of four values:

- Cart position: $x \in (-4.8, 4.8)$,
- Cart velocity: $v \in (-\infty, \infty)$,
- Pole angle: $\theta \in (-0.418, 0.418)$ radians,
- Pole angular velocity: $\dot{\theta} \in (-\infty, \infty)$.

Reward Function: A reward of +1 is given for each time step the pole is balanced, with the goal of maintaining balance for as long as possible. The episode is terminated either when the pole falls or when the maximum time step (500) is reached. The episode's reward threshold is set at 475.

Termination Conditions: The episode ends if:

- The pole angle is outside the range of $\pm 12^\circ$,
- The cart's position exceeds the track boundaries at ± 2.4,
- The episode length exceeds 500 steps.

This environment was implemented using OpenAI's Gym module.

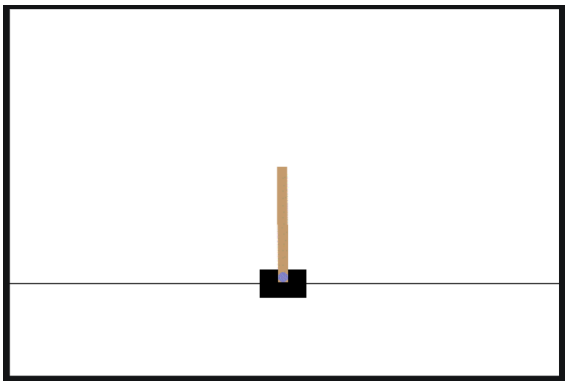


Figure 21: Cartpole

4.2 REINFORCE with Baseline

Experiment Results and Analysis:

In our experiments on the Cartpole domain, we systematically tested different configurations of the policy and value networks, focusing on network architecture, hidden layer sizes, and learning rates to determine the most effective setup. The key findings are summarized below.

1. Network Architecture:

- **Number of Hidden Layers: 1, 2**
  We experimented with both one and two hidden layers for the policy and value networks. Our results indicated that using a single hidden layer led to faster convergence.
- **Hidden Unit Sizes: 64, 128, and 256**
  We tested various hidden unit sizes: 64, 128, and 256. The network with 256 hidden units exhibited the smoothest convergence amongst all but still high variance.

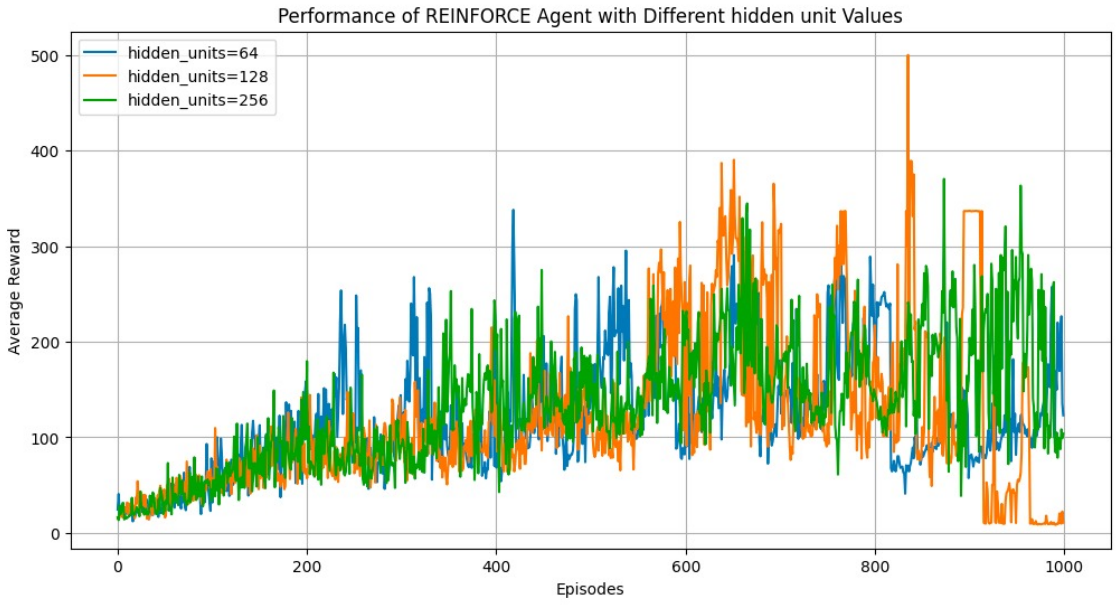


Figure 22: Hidden Unit Sizes: 64, 128, and 256

2. Learning Rate for α_θ (Policy Network)

We tested α_θ values of 0.001,0.0001,0.00001. The learning rate of $\alpha_\theta = 0.001$ resulted in the highest return but higher variance.

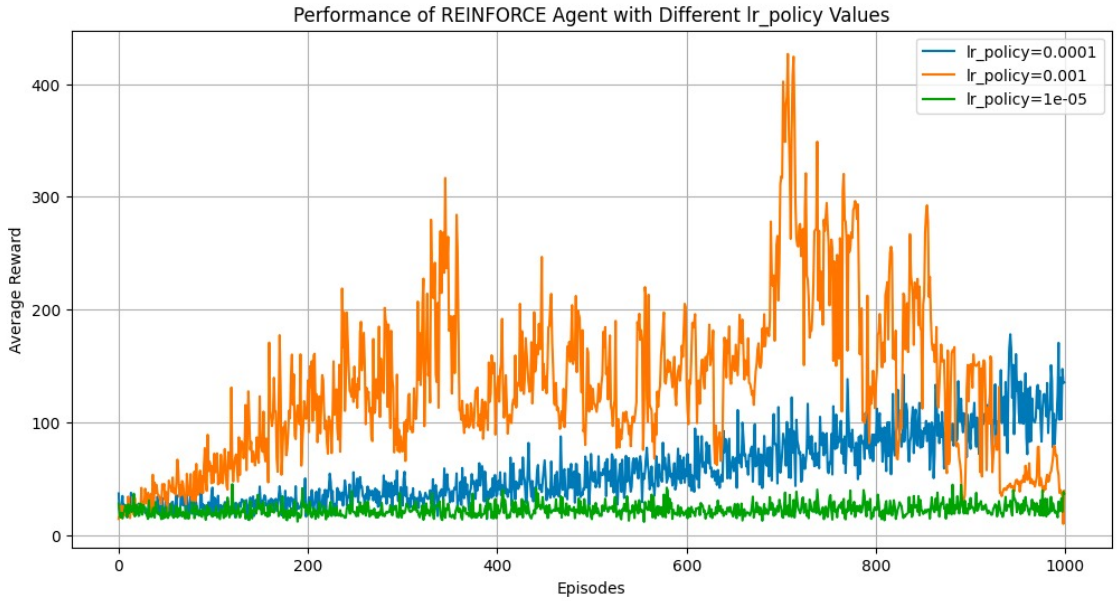


Figure 23: α_θ values $\in 10^{-3}, 10^{-4}, 10^{-5}$

3. Learning Rate for α_w (Value Network)

We tried values of α_w in $\{10^{-1}, 10^{-2}, 5 * 10^{-2}\}$. The learning rate $\alpha_w = 0.01$ achieved the highest return, although all values showed periods of convergence and high variance.

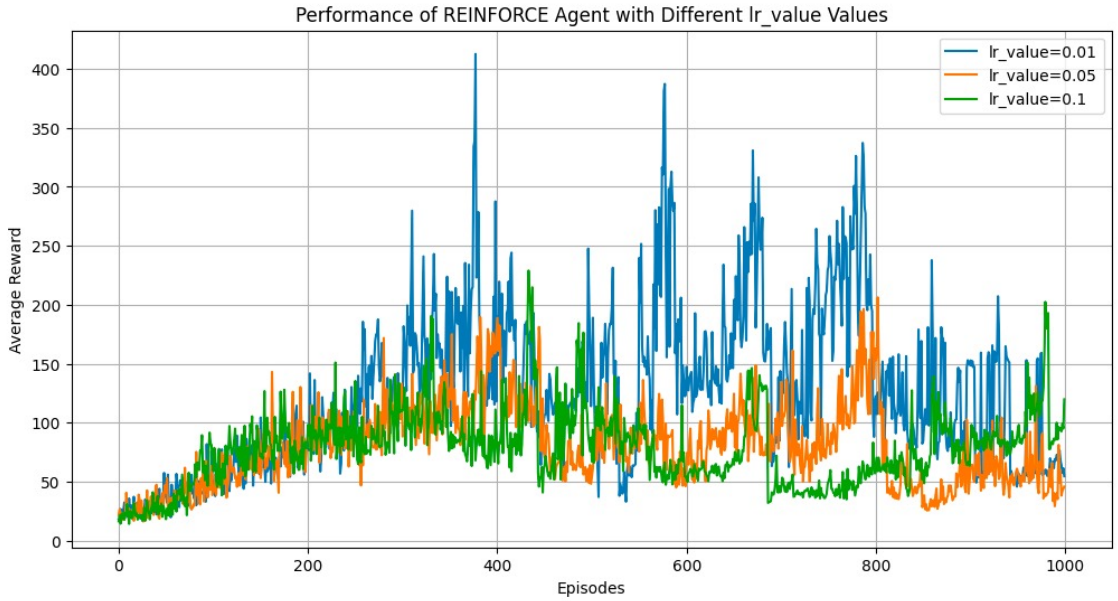


Figure 24: α_w values $\in 10^{-1}, 10^{-2}, 10^{-3}$

4. Final Configuration

Based on our analysis, the optimal configuration for REINFORCE with Baseline network with **one hidden layer**, **256 hidden units**, $\alpha_\theta = 0.001$, and $\alpha_w = 0.01$. This setup resulted in the fastest convergence and the highest return, with manageable variance, making it the most robust choice for the problem.

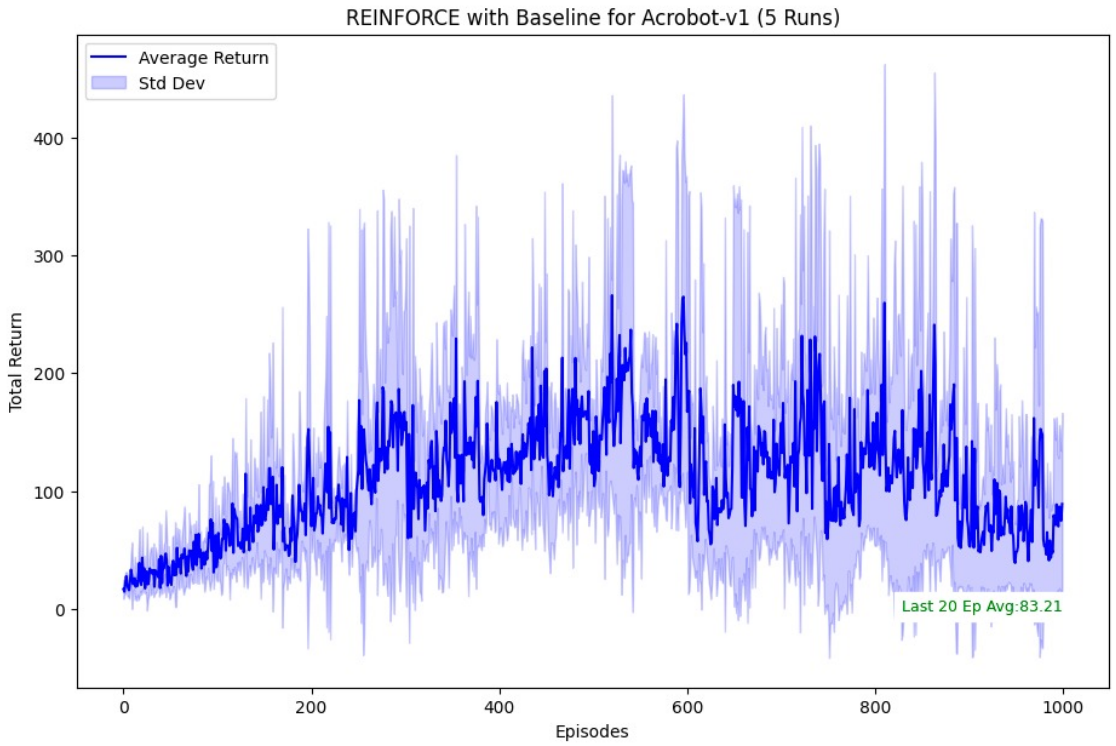


Figure 25: Learning Curve for REINFORCE with Baseline for Cartpole Domain

For cartpole Domain, the REINFORCE with Baseline algorithm converged to a policy with average return ~ **83.21**.

4.3 One-Step Actor-Critic

Experiment Results and Analysis:

In our experiments on the cartpole domain using the One-Step Actor-Critic algorithm, we conducted a series of tests varying the network architecture, hidden unit sizes, and learning rates to find the optimal configuration. The key observations are summarized below.

1. Network Architecture:

- **Number of Hidden Layers: 1, 2**
  We experimented with both one and two hidden layers for the policy and value networks. Our results indicated that using a single hidden layer led to faster convergence.
- **Hidden Unit Sizes: 64, 128, and 256**
  We tested various hidden unit sizes: 64, 128, and 256. The network with 256 hidden units exhibited highest return amongst all but still high variance.

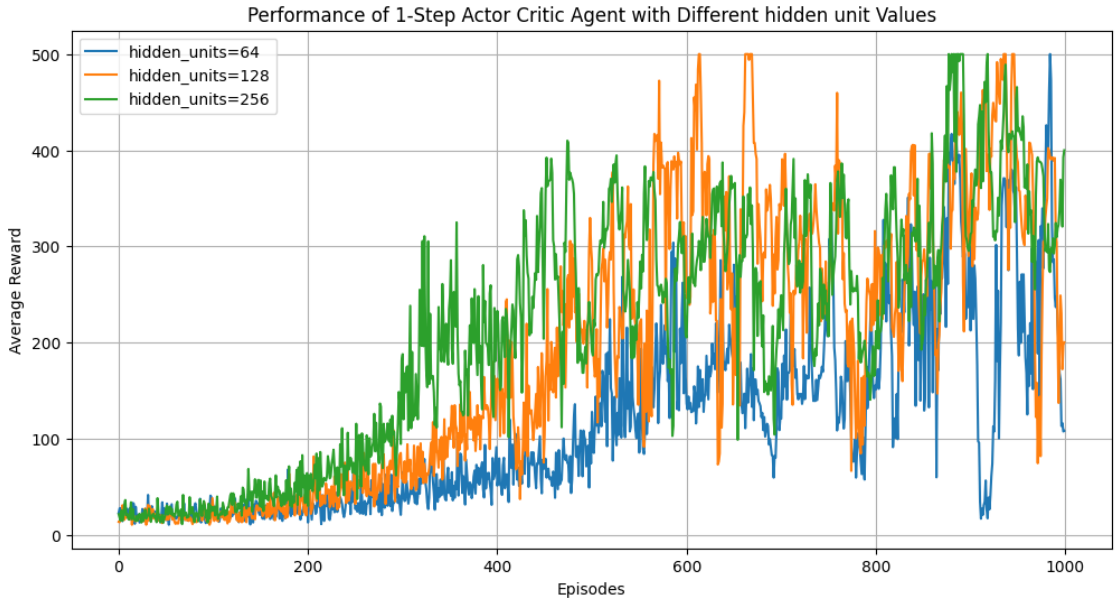


Figure 26: Hidden Unit Sizes: 64, 128, and 256

2. Learning Rate for α_θ (Policy Network)

We evaluated three values for α_θ: 10^{-3}, 10^{-4}, and 10^{-2}. A learning rate of $\alpha_\theta = 0.0001$ resulted in the highest return and lowest variance.

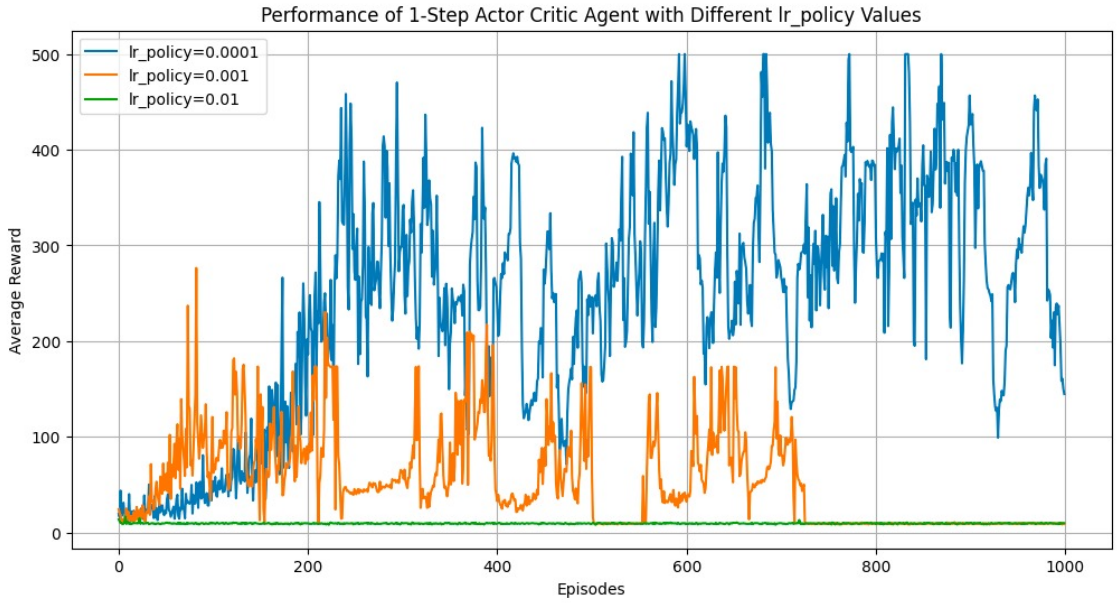


Figure 27: α_θ values $\in 10^{-3}, 10^{-4}, 10^{-2}$

3. Learning Rate for α_w (Value Network)

We tested values for α_w in $\{10^{-1}, 10^{-2}, 10^{-3}\}$. Ultimately, $\alpha_w = 0.0001$ provided the best convergence quality and return.

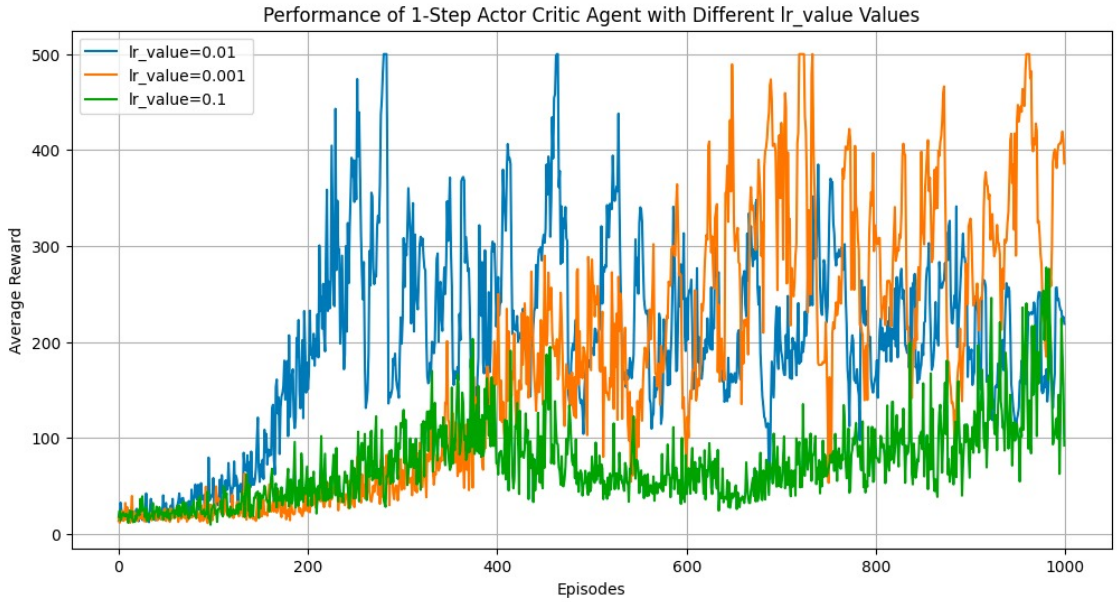


Figure 28: α_w values $\in 10^{-1}, 10^{-2}, 10^{-3}$

4. Final Configuration

Based on the results, the optimal configuration for One-Step Actor-Critic on the Cartpole domain was a network with **one hidden layer**, **256 hidden units**, $\alpha_\theta = 0.0001$, and $\alpha_w = 0.0001$. This configuration achieved the fastest convergence with the highest return while maintaining manageable variance, making it the most robust choice for solving the Acrobot task.

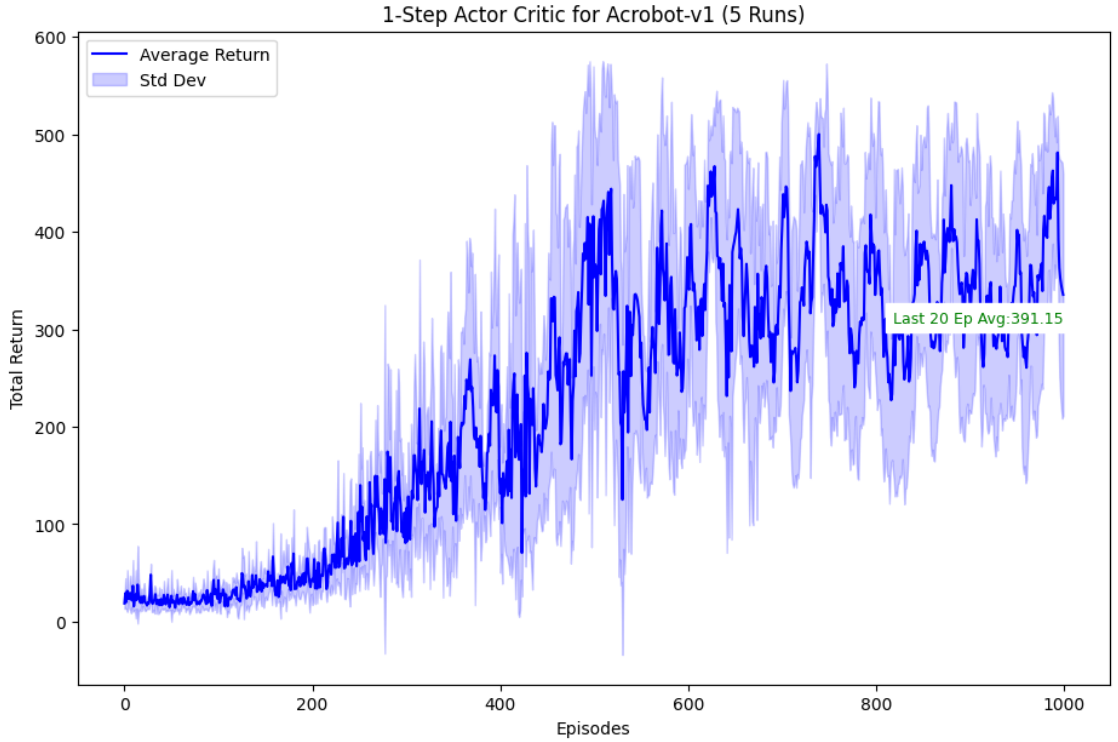


Figure 29: Learning Curve for One-Step Actor-Critic on Acrobot Domain

References

[1] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[2] OpenAI Gym Team. Acrobot environment documentation, 2024. Accessed: 2024-12-10.