

# Reinforcement Learning Project

Ishita Kumar, Aneri Rana

December 9, 2023

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Environment Definitions</b>	<b>1</b>
2.1	687-Gridworld . . . . .	1
2.1.1	States . . . . .	1
2.1.2	Actions . . . . .	2
2.1.3	Transition . . . . .	2
2.1.4	Rewards . . . . .	2
2.1.5	Terminal state . . . . .	2
2.1.6	Initial State . . . . .	2
2.1.7	Adaptation for training . . . . .	2
2.2	Cart Pole . . . . .	3
2.2.1	Environment Description . . . . .	3
2.2.2	States . . . . .	3
2.2.3	Action Space . . . . .	3
2.2.4	Rewards . . . . .	3
2.2.5	Terminal State . . . . .	3
2.2.6	Initial State . . . . .	3
<b>3</b>	<b>REINFORCE With Baseline 687-GRIDWORLD AND CARTPOLE</b>	<b>4</b>
3.1	Description . . . . .	4
3.2	Pseudocode and Algorithm design . . . . .	4
3.3	687-Gridworld . . . . .	5
3.3.1	Hyper parameters . . . . .	5
3.3.2	Experimental Results . . . . .	5
3.4	Cart Pole . . . . .	5
3.4.1	Hyper parameters . . . . .	5
3.4.2	Experimental Results . . . . .	7
<b>4</b>	<b>Actor Critic With 687-GridWorld And Cart Pole</b>	<b>8</b>
4.1	Description . . . . .	8
4.2	Pseudocode and Design choices . . . . .	8
4.3	Hyperparameter Tuning . . . . .	9
4.4	Experimental Results 687-GridWorld . . . . .	10
4.5	Experimental Results Cart Pole . . . . .	10
<b>5</b>	<b>Extra Credit</b>	<b>13</b>
5.1	Mountain Car . . . . .	13
5.2	Environment Design . . . . .	13
5.3	Actor critic with Mountain Car Environment . . . . .	13
5.4	REINFORCE with Baseline with Mountain Car Environment . . . . .	14
5.5	Cliff Walking . . . . .	15

5.6	Training Reinforce With Baseline on Cliff Environment . . . . .	16
5.7	Custom Environmnet . . . . .	17

# 1 Introduction

For this project, we implemented two algorithms:

1. REINFORCE With BASELINE We implemented this algorithm on the following environments
  - 697-GridWorld
  - Cartpole
2. ACTOR-CRITIC Algorithm We implemented this algorithm on the following environments
  - 687-GridWorld
  - CartPole
3. For **extra credit** we also implemented the REINFORCE With BASELINE and ACTOR-CRITIC algorithms on the following environments
  - Mountain Car
  - Cliff Walking
4. For **extra credit**, we also implemented a custom environment and trained it on REINFORCE With BASELINE
5. For **extra credit** we implemented ACTOR-CRITIC on Mountain Car

## 2 Environment Definitions

### 2.1 687-Gridworld

#### 2.1.1 States

We have implemented the 687-GridWorld from class for experimenting with both algorithms. The state space is a 5x5 grid and each state is the position in the grid (r,c), where r is the row index and c is the column index such that  $r \in [0,4]$  and  $c \in [0,4]$ . In figure, the state 1 is defined as (0,0). The obstacle state is located at (2,2) and (3,2). (4,2) is the water state and (4,4) is the goal state.



Start State 1	State 2	State 3	State 4	State 5
State 6		State 8	State 9	State 10
State 11	State 12	Obstacle	State 13	State 14
State 15	State 16	Obstacle	State 17	State 18
State 19	State 20		State 22	End State 23

Figure 1: The original 687-GridWorld.

### 2.1.2 Actions

The agent can take four actions namely, AttemptUp, AttemptDown, AttemptRight, and AttemptLeft.

### 2.1.3 Transition

Since it is a stochastic MDP the transition to a new state is non-deterministic and is sampled according to the following probabilities.

- With 80% probability, the agent moves in the specified direction.
- With 5% probability, the agent gets confused and veers to the right with respect to the intended direction.
- With 5% probability, the agent gets confused and veers to the left with respect to the intended direction.
- With 10% probability, the agent temporarily breaks and does not move.

If the agent hits a wall or obstacle, it stays in its current state

### 2.1.4 Rewards

The reward is always +10 for entering the goal state, -10 for entering the water state and zero everywhere else.

### 2.1.5 Terminal state

The episode terminates on reaching the goal state(4,4).

### 2.1.6 Initial State

The agent always starts in state 1

### 2.1.7 Adaptation for training

All state, action, and rewards are Markovian since at time  $t + 1$ , they do not depend on the history  $H_{t-1}$  given we know the values at time  $t$ . For implementation purposes, we represent the states using one hot encoding. If the grid space is flattened out there would be a total of 25 ( $5 \times 5$ ) states so each hot vector is of size 25. For example, state 6 in the figure would be represented as  $[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$ . For transitions given a state and action, we calculate the possible next state and sample from the probability distribution according to the transition function above.

## 2.2 Cart Pole

### 2.2.1 Environment Description

An un-actuated joint attaches a pole to a cart, which moves along a frictionless track. The pendulum is placed upright on the cart, and the goal is to balance the pole by applying forces in the left and right direction on the cart<sup>1</sup>. The environment is a Markov environment.

### 2.2.2 States

The state is represented using four tuples (Cart Position, Cart Velocity, Pole Angle, Pole Angular Velocity). The cart position is its position along the x-axis racing between -4.8 to 4.8, whereas Cart Velocity can be between -infinity to +infinity similar to Pole Angular Velocity. Finally, Pole Angle with respect to the cart ranges between -0.418 rad and 0.418 rad approximately. These four tuples provide enough information for the state to be Markovian.

### 2.2.3 Action Space

The action space is discrete with two possible actions:

- 0: left
- 1: right

### 2.2.4 Rewards

The agent gets a reward of +1 for every time step it is alive. The threshold for the environment is 475.

### 2.2.5 Terminal State

Termination occurs on the following conditions

- Pole Angle is greater than  $\pm 12^\circ$
- Cart Position is greater than  $\pm 2.4$
- Episode length is greater than 500

### 2.2.6 Initial State

The agent is assigned a uniform random value between (-0.05,0.05)

---

<sup>1</sup>[https://www.gymnasium.dev/environments/classic\\_control/cart\\_pole/](https://www.gymnasium.dev/environments/classic_control/cart_pole/)

## 3 REINFORCE With Baseline 687-GRIDWORLD AND CARTPOLE

### 3.1 Description

REINFORCE With Baseline is an extension to the REINFORCE algorithm as an episodic policy gradient learning method. It requires running trajectories to collect sample gradients that are proportional to the actual gradient as a function of the parameter.

$$\nabla J(\theta) \propto \mathbb{E}_{\pi} \left[ \sum_a q_{\pi}(S_t, a) \nabla \pi(a|S_t, \theta) \right]$$

REINFORCE With Baseline is a generalization of REINFORCE, such that we add a baseline to our equation to compare the action to this baseline. The baseline, which can be anything is not dependent on action, but the state. This results in the expected value of the update remaining unchanged but a large change in the variance of our learning algorithm. Therefore in general we get :

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a (q_{\pi}(s, a) - b(s)) \nabla \pi(a|s, \theta)$$

Adding a baseline can make our algorithm learn much faster.

### 3.2 Pseudocode and Algorithm design

The Pseudocode for the algorithm is as follows: 1

---

**Algorithm 1** REINFORCE with Baseline(episodic), for estimating  $\pi_{\theta} \approx \pi^*$

---

```
1: Initialize policy parameters  $\theta$  and baseline  $\hat{v}(s, w)$ 
2: for each episode do
3:   Generate trajectory  $\tau = (s_1, a_1, r_1, \dots, s_T, a_T, r_T)$  using policy  $\pi_{\theta}$ 
4:   for  $t = 1$  to  $T$  do
5:     Compute the return  $G_t = \sum_{k=t+1}^T \gamma^{k-t-1} R_k$ 
6:     Compute the advantage  $\delta \leftarrow G_t - b(s_t)$ 
7:     Update baseline:  $w \leftarrow w + \alpha^w \delta \nabla \hat{v}(s, w)$ 
8:     Update policy parameters:  $\theta \leftarrow \theta + \alpha \delta \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$ 
9:   end for
10: end for
```

---

For our algorithm design, we implemented two neural networks. The first neural network, the policy network, has two linear layers with the the input being the length of the observation space and the output being the number of actions. The size of the hidden layers is a hyperparameter. The baseline neural network has two linear layers with the input being the length of the observation space and the output being one. The size of the hidden layer is a hyperparameter.

### 3.3 687-Gridworld

The environment of 687-Gridworld is described above in Section 2.2. The input space of the networks is a one-hot vector of length 25, where the discrete position of the agent is represented by the value 1.

#### 3.3.1 Hyper parameters

We used grid search to find parameters for Grid World. The hyperparameters we trained were :

- $\alpha$ (learning rate) : 1e-3
- $\gamma$  : 0.99
- hidden layer size: 32
- number of episodes: 500

In our experiments, we found that reducing the learning rate decreased the variance, as the algorithm takes smaller steps. Higher gamma values generally performed better. Compared to other environments we experimented with, 687-Gridworld is a less complex environment, so we opted for a smaller model architecture.

#### 3.3.2 Experimental Results

We conducted 20 trials of the REINFORCE With Baseline Algorithm with the 687-GridWorld environment.

We plotted the average length of the episode, the MSE loss of the baseline algorithm, and the learning curve for the rewards obtained over 1000 episodes. As we can see, there is a steady decrease in both the loss and the number of episodes as our agent learns. The reduction in the number of episodes indicates that the agent is not merely roaming around and eventually stumbling into the reward state; instead, it is actively learning. Similarly, the increase in reward indicates that the agent is avoiding negative rewards.

We also plotted the final policy generated by our algorithm. It's important to note that the policy is not deterministic; rather, it represents the maximum of the action probabilities over a single trial, as provided by the Policy Network

### 3.4 Cart Pole

The environment variables for CartPole are described above in Section 2.2. Please note that we used the gym implementation for the environment. The observation space for this environment is 4 and the action space is 2.

#### 3.4.1 Hyper parameters

We used grid search to find parameters for Grid World. The hyperparameters space that we searched was :

- $\alpha$  : [0.0001,0.001,0.01]



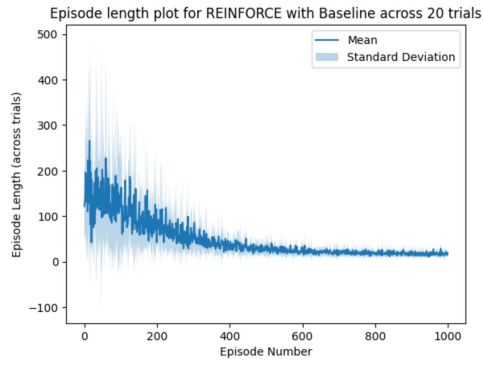


Figure 1: Average Length of Episode over 20 Trials.

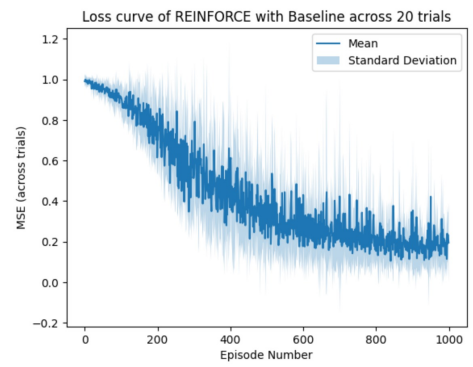


Figure 2: Average Loss of the network over 20 trials

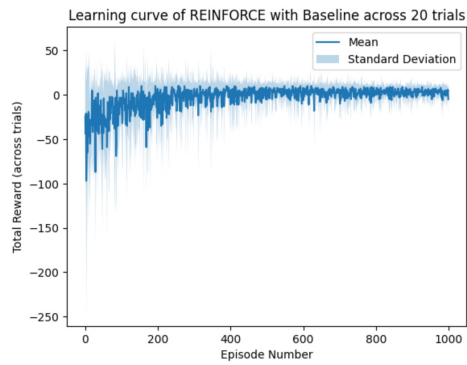


Figure 3: Average Learning Rate of the Network over 20 Trials.

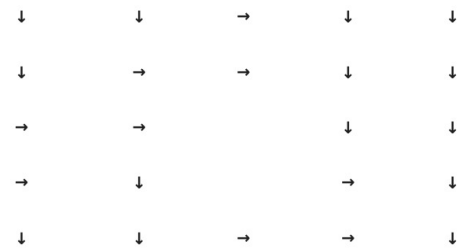


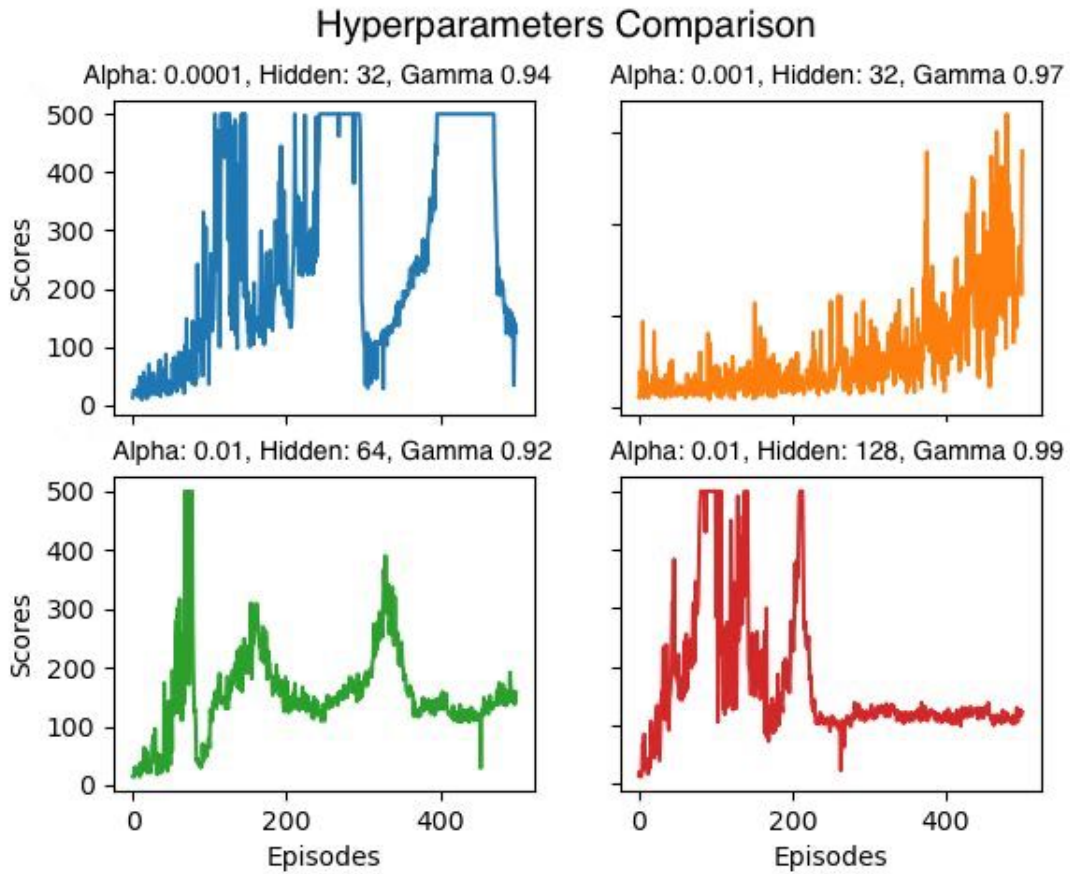
Figure 4: Policy of the network

- hidden layers : [32,64,128]
- $\gamma$ : [0.92, 0.95, 0.97, 0.99]

We ran the grid search using wandb. We found that multiple hyperparameter settings resulted in the graph to converge. We finally landed on the following settings:

- $\alpha$ (learning rate) :  $1e-3$
- $\gamma$  : 0.97
- hidden layer size: 32
- number of episodes: 1000

We found that the agent did not converge to its maximum reward with a smaller gamma.

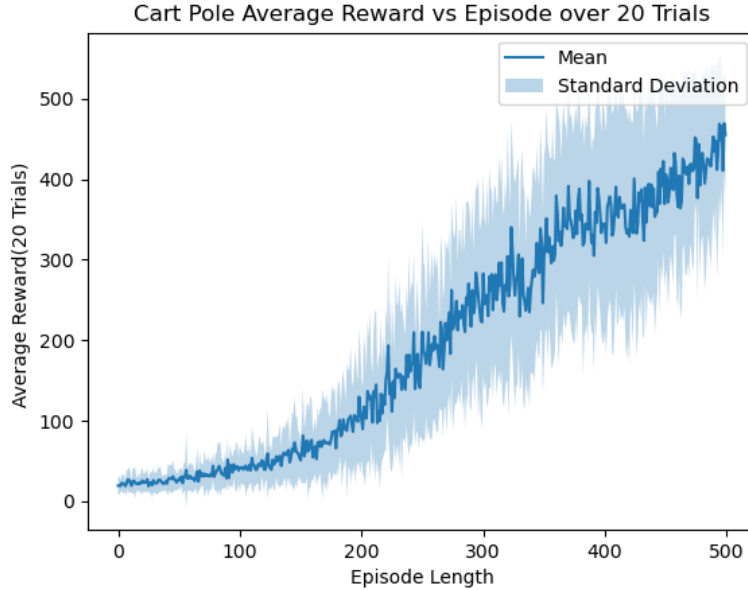


In our experiments, we found that reducing the learning rate decreased the variance, as the algorithm takes smaller steps. Higher gamma values generally performed better. Compared to other environments we experimented with, 687-Gridworld is a less complex environment, so we opted for a smaller model architecture.

### 3.4.2 Experimental Results

We conducted 20 trials of the REINFORCE With Baseline Algorithm with the Cartpole environment.

We plotted the average learning curve for the rewards obtained over 500 episodes. The reward keeps on steadily increasing as the agent learns to stay alive for longer.



## 4 Actor Critic With 687-GridWorld And Cart Pole

### 4.1 Description

Actor-Critic is a fully online and incremental algorithm. It is fully online because, unlike REINFORCE with a baseline, it updates the training model at each step of the episode. In the actor-critic method, the actions of the policy are assessed using an estimate of the return. This estimate is calculated as the sum of the reward and the discounted value of the next state by a model called the critic. The actor is responsible for selecting an action given a state and updating this policy according to the loss calculated on the estimate of the return as seen in equation:

$$\theta_{t+1} = \theta_t + \alpha (R_{t+1} + \gamma \hat{v}(S_{t+1}, w) - \hat{v}(S_t, w)) \frac{\Delta \pi(A_t | S_t, \theta_t)}{\pi(A_t | S_t, \theta_t)}$$

The one-step actor-critic method is an analog of the algorithms based on TD error. Although REINFORCE with baseline also uses an estimate of the baseline to learn the policy, comparatively actor-critic is superior in terms of variance and data efficiency. This is because the one-step TD error has less variance than the actual return calculated at the end of the episode using a noisy policy.

### 4.2 Pseudocode and Design choices

We implemented the actor-critic algorithm according to the pseudocode given in 5. To model the actor and the critic a neural network with linear layers and ReLU activation was employed. For both neural nets, the dimension of the input space is equal to the size of the state representation of the selected environment. In the case of discrete states, they were one hot encoded before feeding to the input layer. For the actor, the dimension of the output is the size of the action

**One-step Actor–Critic (episodic), for estimating  $\pi_\theta \approx \pi_*$** 

Input: a differentiable policy parameterization  $\pi(a|s, \theta)$   
 Input: a differentiable state-value function parameterization  $\hat{v}(s, \mathbf{w})$   
 Parameters: step sizes  $\alpha^\theta > 0, \alpha^\mathbf{w} > 0$   
 Initialize policy parameter  $\theta \in \mathbb{R}^{d'}$  and state-value weights  $\mathbf{w} \in \mathbb{R}^d$  (e.g., to  $\mathbf{0}$ )  
 Loop forever (for each episode):  
   Initialize  $S$  (first state of episode)  
    $I \leftarrow 1$   
   Loop while  $S$  is not terminal (for each time step):  
      $A \sim \pi(\cdot|S, \theta)$   
     Take action  $A$ , observe  $S', R$   
      $\delta \leftarrow R + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})$       (if  $S'$  is terminal, then  $\hat{v}(S', \mathbf{w}) \doteq 0$ )  
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha^\mathbf{w} \delta \nabla \hat{v}(S, \mathbf{w})$   
      $\theta \leftarrow \theta + \alpha^\theta I \delta \nabla \ln \pi(A|S, \theta)$   
      $I \leftarrow \gamma I$   
      $S \leftarrow S'$

Figure 5: Actor Critic pseudocode[1]

space with their probability distribution given an input state. The critic model outputs a single continuous value as the estimate of a given state. The neural nets are trained for a constant number of episodes. At each step of the episode, an action is taken and the agent transitions to the next state. Critic estimates the value of this state and the previous state. Based on this Mean Squared Error (MSE) is calculated and the critic network is updated using the Adam optimizer. Furthermore, the values are calculated again and the actor network is updated with the update rule and Adam Optimizer. This is a slight modification from the original pseudo-code for practical efficiency.

### 4.3 Hyperparameter Tuning

For training different environments with actor-critic algorithms we use a grid search to select the following hyperparameters.

- Neural network model - having different neural network architectures for actor and critic did not seem to have much effect on learning. Therefore, both were turned using a single set of parameters for efficiency.
  - Number of linear layers: [2, 3]
  - Dimension of the linear layers: [16, 32, 64, 128]
- Learning Rate - as per [2], having a higher learning rate for a critic than the actor is more efficient. This is because the actor is updated based on estimates from the critic and having the critic learn faster will further boost the learning of the actor. Hence a tuple was selected from the grid such that the critic always has a higher learning rate than the actor.
  - Actor learning rate: [0.5, 0.4, 0.3]

- Critic leaning rate: [0.4, 0.3, 0.2]
- Discount parameter: [0.9, 0.95, 0.99] Since having a high discount parameter has proven to be always better, we did not experiment with lower discount rates.
- Number of episodes - The grid for this depends on the environment as the selected environments have a different range of speed of convergence.
  - For grid world: [300, 500, 800]
  - For cart pole: [500, 1000, 1500]

#### 4.4 Experimental Results 687-GridWorld

687-GridWorld as described in section 2.1 was trained using the actor-critic algorithm and the following selected hyperparameters:

- Number of linear layers = 2
- Dimension of the linear layers = 32
- Actor learning rate = 1e-3
- Critic leaning rate = 1e-2
- Discount parameter = 0.99
- Number of episodes = 500

In 6 we can see the total reward increases from -50 to 0 as the number of episodes the agent has seen increases. The reward from each episode was plotted as the mean and standard deviation over 20 different training runs of the algorithm. In 687 GridWorld the maximum reward of an episode can earn is 10 and the above curve quickly plateaus at 10 after just 100 episodes. Also, the learned policy shown in 7 clearly directs the agent from the start state to the goal state while avoiding obstacles and water state. Even though our policy is non-deterministic, we selected the action with argmax of the probability distribution given by the actor for this plot.

The remaining figures show the curves for a single training session. In fig 10 as well the total reward plateaus to 10 within 100 episodes. Furthermore, we have plotted the episode length in 9. Since the agent gets reward 0 at every other state it is possible for the agent to trivially receive a total reward of 10 after roaming around the grid world for a long time. But we can see that episode length also decreases which indicates the agent is learning to reach the goal faster due to the discount factor in the loss function. Finally, we have also plotted the critic loss 8 which is decreasing as the training progresses since the value estimate improves on training the critic model.

#### 4.5 Experimental Results Cart Pole

Cart Pole was trained using an actor-critic algorithm and the following hyperparameters were selected:

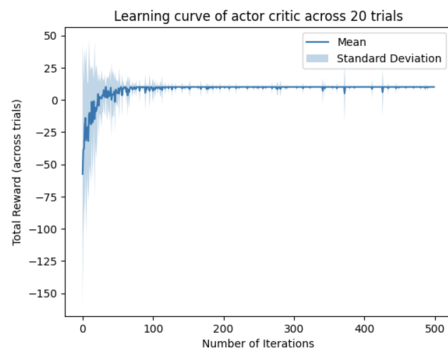


Figure 6: Average Rewards over 20 Trials.

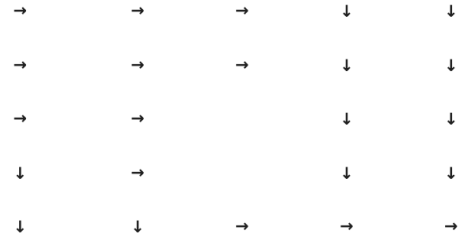


Figure 7: Policy given by the Network

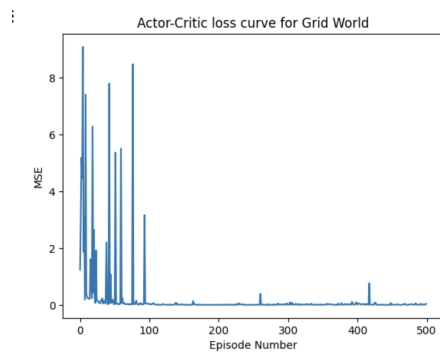


Figure 8: Average MSE the Network over 20 Trials.

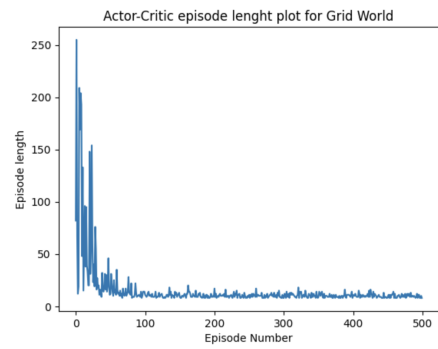


Figure 9: Average Episode length over 20 trials

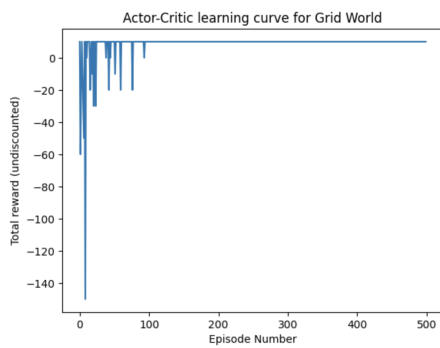


Figure 10: Average Rewards over 20 trials

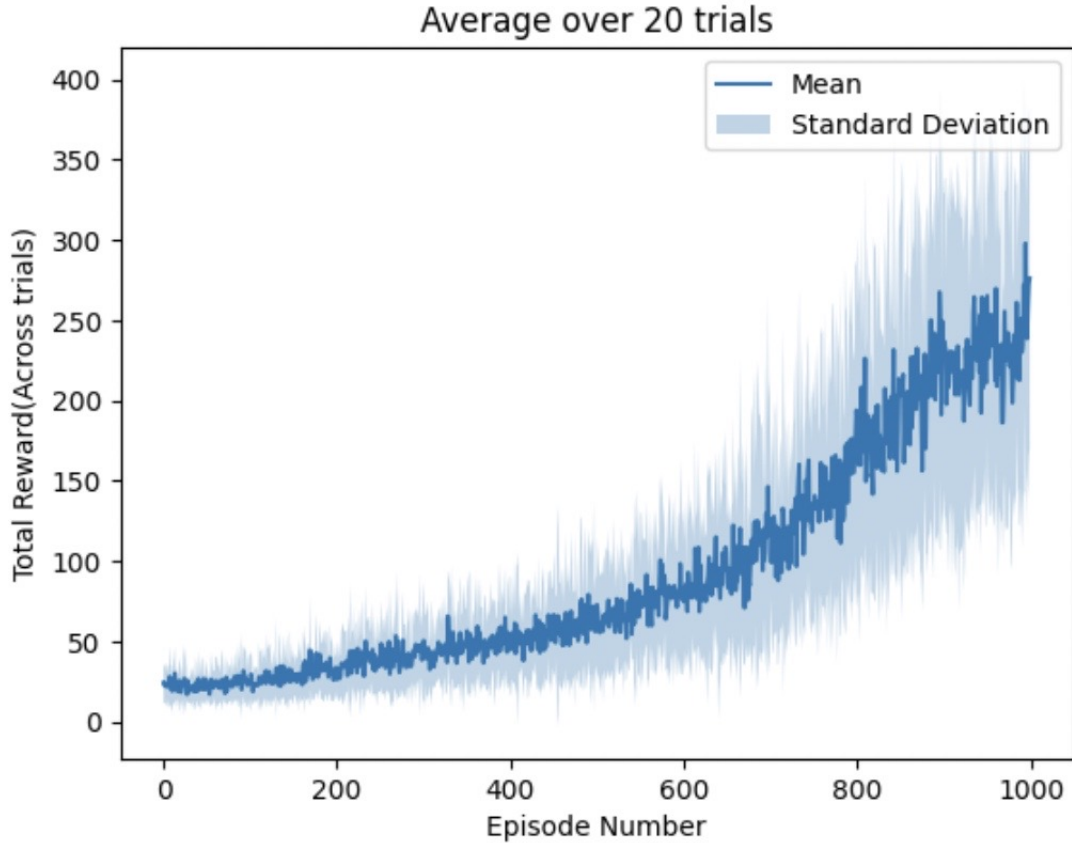


Figure 11: Average Reward Over 20 trials

- Number of linear layers = 2
- Dimension of the linear layers = 128
- Actor learning rate =  $1e-4$
- Critic learning rate =  $1e-3$
- Discount parameter = 0.99
- Number of episodes = 1000

In 11 we have plotted the total reward vs the number of episodes that the agent has seen during training. The environment was trained for 5 different trials and the mean and standard deviation of the total reward is shown. It takes the agent at least 1000 episodes to consistently earn the total reward of approximately 200. This learning curve has the potential to learn more given a more granular reward pattern and longer training. However, with the given CartPole gym environment and the actor-critic algorithm the total expected reward is 200.

## 5 Extra Credit

### 5.1 Mountain Car

#### 5.2 Environment Design

We performed extra experiments using multiple algorithms on the mountain car environment from the gym, as it was the most difficult environment to train among all the selected settings.

1. **State:** The state is represented by a 2-tuple position of the car along the x-axis in the range  $[-1.2, 0.6]$  and the velocity of the car in the range  $[-0.07, 0.07]$ .
2. **Actions:** Agent can take only 3 actions I.e., Accelerate to the left, Don't accelerate, and accelerate to the right
3. **Transition:** The agent deterministically transitions to the next state given a state and action. The next state is determined using the following formula:

- $velocity_{t+1} = velocity_t + (action - 1) * force - \cos(3 * position_t) * gravity$
- $position_{t+1} = position_t + velocity_{t+1}$

where  $force = 0.001$  and  $gravity = 0.0025$ .

4. **Rewards:** Agent is given a reward of -1 for every step until the episode is terminated.
5. Initial state: The position of the car is assigned a uniform random value in  $[-0.6, -0.4]$ . The starting velocity of the car is always assigned to 0.
6. Terminal state: The episode terminates when either the car reaches the goal position on top of the right hill or the episode is truncated. We have used a custom episode length according to how much each algorithm needs to explore in the beginning.

#### 5.3 Actor critic with Mountain Car Environment

Training Mountain Car using actor-critic required training on GPU and due to limited resources, we did not perform hyperparameter tuning. The following values were selected based on prior runs of this algorithm and manual adjustment.

- Number of linear layers = 2
- Dimension of the linear layers = 32
- Actor learning rate =  $1e-3$
- Critic leaning rate =  $1e-2$
- Discount parameter = 0.99
- Number of episodes = 500



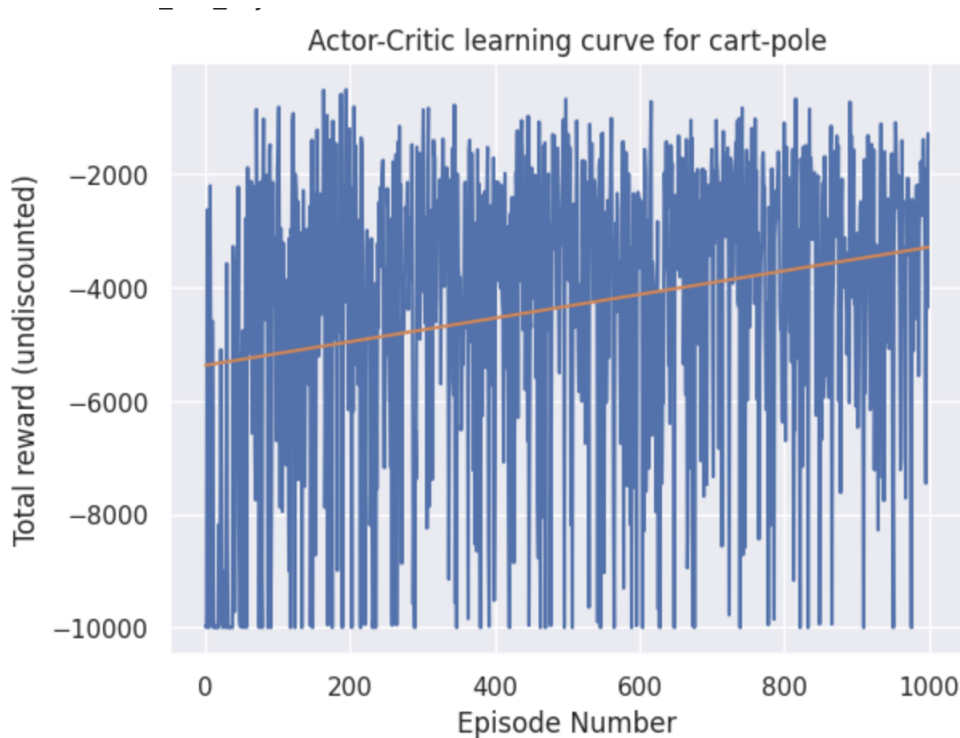


Figure 12: Learning Plot of Actor-Critic

Running the algorithm with the above setting required 4 hours on a T4 GPU and resulted in the curve shown in 12. It shows the total reward vs the number of episodes the agent has seen. Since the rewards are very noisy, we have plotted a regression line on top of it. We can see that the total reward increases but very slowly and is unable to cross -500. Also, in the beginning, more episodes were truncated at 10000 before reaching the goal state compared to later runs of the episode. Most of the time the agent was able to reach the goal state before truncation. However, it took a long time to reach the goal state of 3-4k. Current state-of-the-art agents can reach the goal state in 200 steps. However additional modifications are required to the environment to accomplish that, such as rewards for exploration or more granular rewards instead of a single reward at the end of the episode by dividing agent positions into bins. In the current environment settings agent has learned how to reach the goal but is unable to learn how to reach the goal state faster.

#### 5.4 REINFORCE with Baseline with Mountain Car Environment

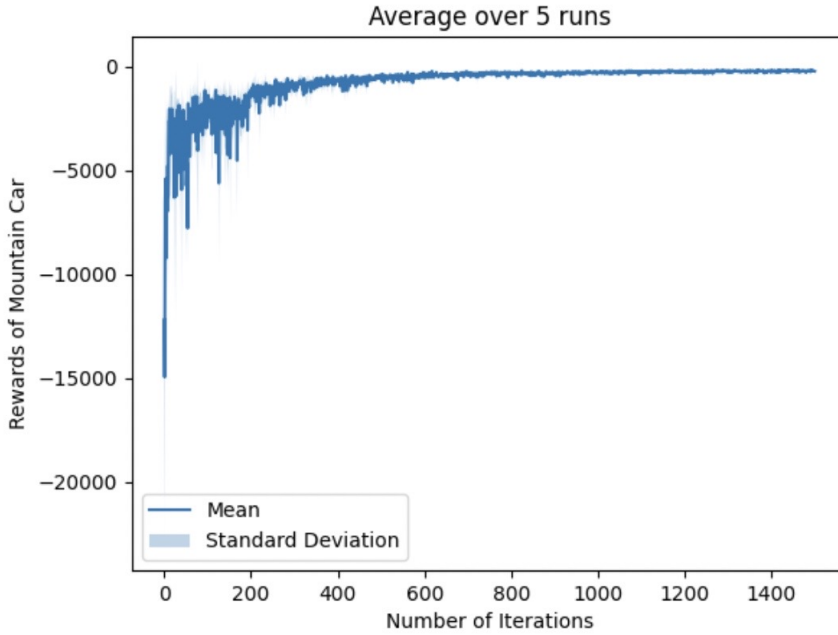
Mountain Car was trained for 5 trials using REINFORCE with Baseline Algorithm. As the environment is quite complex, with a low frequency of reward, we went with a bigger architecture for our neural network. I used a grid search for hyperparameter tuning on the following hyperparameters:

- Hidden layer : [64,120]
- $\gamma$  : [0.95,0.995]
- $\alpha$  : [0.01,0.001,0.0001]

The final hyperparameters of our network were:

- Hidden Layer: 120
- $\gamma$  : 0.99
- $\alpha$  : 0.001

It took the algorithm about 1500 steps to converge. The variance was quite high in the beginning with it stabilizing over time.



We also noticed when training the mountain car with a baseline versus without a baseline, the convergence was achieved faster with the baseline and it was also more stable.

## 5.5 Cliff Walking

We implemented a cliff walking environment, where the agent needs to navigate a 4x12 grid space from the initial to the goal state without falling into the cliff. Its dynamics are described as follows.

1. **States:** The state of an agent is its position on the grid given by  $[r, c]$  where  $r$  is the row index and  $c$  is the column index such that  $r \in 4$  and  $c \in 12$ . For implementation, we represent the state by flattening out the grid and using a one-hot encoded vector of size 48 (similar to grid world).
  - **Initial state** is always  $[3, 0]$
  - **Goal state** is  $[3, 11]$
  - **Cliff states** are in the row 3
2. **Actions:** The agent can take only 4 deterministic actions namely move up, move down, move right, and move left.

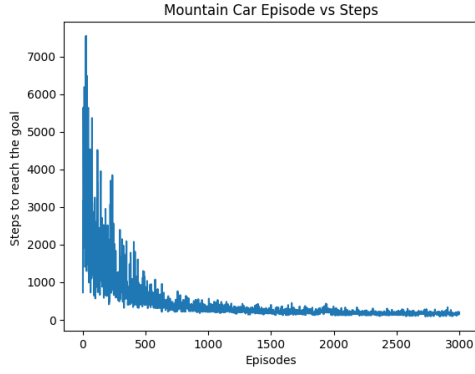


Figure 13: Mountain Car with baseline

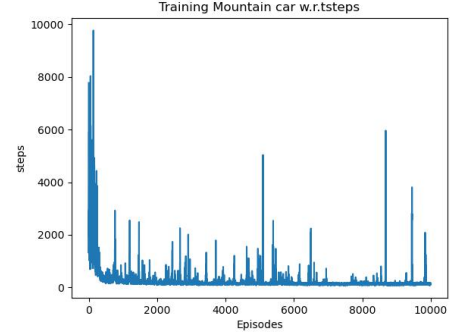


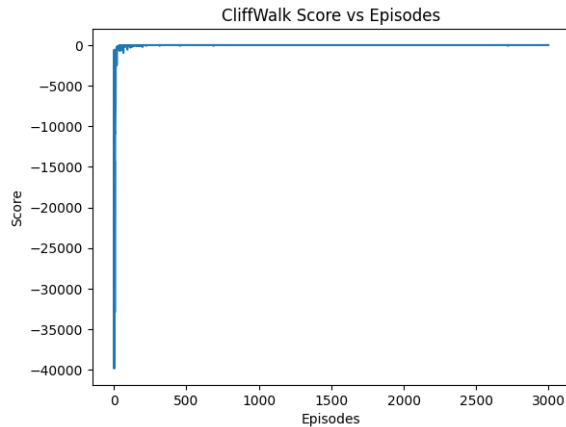
Figure 14: Mountain Car without Baseline

3. **Transition:** The transition is deterministic and the next state is decided based on the current state and action taken. For example, taking action to move right in  $[0,0]$  state will result in the next state  $[0,1]$ .
4. **Termination:** The episode terminates when either the agent reaches the goal state or enters a cliff state.

## 5.6 Training Reinforce With Baseline on Cliff Environment

We trained Reinforce with baseline on our implementation of the Cliff Walking Environment. The hyperparameters were as follows:

- Hidden State : 128
- $\alpha$  1e-3
- $\gamma$  0.99



## 5.7 Custom Environmnet

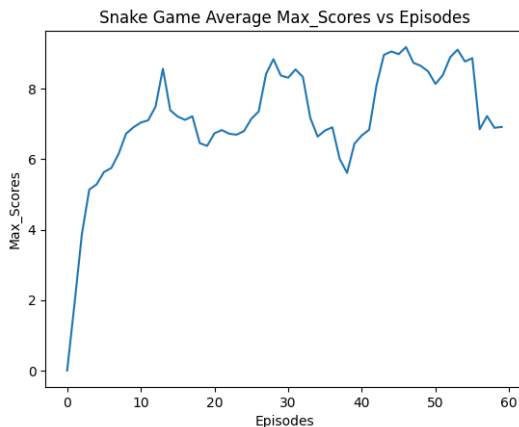
We also implemented a custom environment to train the agent to play the game of snake.

1. **State:** The boards was a 5\*5 matrix. There were in total 25 states. State representation to the network was a one-hot encoded vector of length 12. The first 4 represent the direction of the reward relative to the snake. The next four indices is the direction that the snake is moving in. Finally the last 4 indexes are the obstacles above, below, left or right to the snake. If there is an obstacle the index is 1, otherwise it is zero.
2. **Reward :** If the snake moves in the direction of the food, it gets a reward of 10. If it eats the food, it gets a reward of 500. If it dies it gets a reward of -50.
3. **food coordinates:** The food is randomly placed over the board.
4. **termination :** The game is terminated if the snake dies. The snake can die if it runs into a wall, runs into itself or becomes the same length as our state representation.
5. **Initial State :** The snake is randomly initialized.

The snake starts with the length 1. For every food item it eats the length of the snake increases by one. The score of the game is the length of the snake. Every time the snake eats a new food is spawn.

We trained a neural network with 2 hidden layers. The hyperparameters are as follows:

- board length: 5
- hidden layers: 128,64
- learning rate: 1e-3
- gamma: 0.95



Here we are plotting the average score of the game over 500 episodes. As we can see the snake rapidly learns at first and then it slows down, averaging around a score of 7. There are multiple reasons for observing this behavior. As the snake grows there are more terminal states. Also, our agent has incomplete information about the environment. It does not know if there is any terminal state diagonal to it, or the entire representation of the board.