Final Project Reinforcement Learning

Ishita Chakravarthy
`ichakravarth@umass.edu`
Srinanda Kishore Yallapragada
`syallapragad@umass.edu`

1 Problem Statement

The main motivation behind this problem statement is to identify and address a real world issue faced-of students commuting from home to campus during snow days. Finding the shortest path from home to campus can be impacted by a variety of events, including taking a bus to campus, walking to campus but feeling cold, taking detours into buildings to meet friends/escape from the cold. A student might also have different events to get to, which means their target is not necessarily fixed. This model can be seen as a more complicated version of gridworld which was discussed over assignments, with a bigger state space, and a variety of additional features.

1.1 Work and team members

We worked together on the code as well as the report. We met in person and over zoom and we used the VSCode live share code feature and worked on creating the environment and setting up the algorithms together. We separately ran experiments to finetune the hyperparameters, and also worked on the report together.

2 Assumptions

Various assumptions were made to model this problem statement. Firstly, the state space was considered to be a 10 by 10 grid. This assumption keeps the state space fairly large, while ensuring that the algorithms do not have a very high run time. The buildings, road locations, terminal state locations, crosswalks, and further details on the modeled state space are detailed below. These were arbitrary decisions, however, they were chosen to ensure that there is complexity in the MDP and that every incremental feature (road, crosswalk, buildings) required the model to learn more and explore more aspects of the state space.

The MDP was built on Python, using object oriented programming structures, and the OpenAI Gym implementation was used to run reinforcement learning algorithms. The environment was built in stages, and there are multiple versions of the environment present in the code. Each of these environments corresponds to the incremental updates in the environment and complexity.

3 MDP Formulation

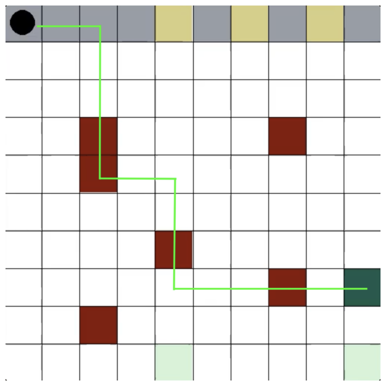

Figure 1: MDP

The MDP is a gridworld of size (10x10). There is a road present on the top row of the MDP, represented by grey. There are crosswalks replacing the road at states [8, 0], [6, 0],[4,0]- represented by yellow. There are buildings or warm spots present through the grid- located at [2, 3], [2, 4], [4, 6], [2, 8], [7, 7], [7, 3]- represented by maroon. The possible goal states (1) is located at [9, 9] (2) is located at [4, 9] and (3) is located at [9,7]. The active goal state for the episode is marked in dark green, while the remaining goal states are marked with light green. The agent is represented by the circle, and a run of an optimal policy after training on the algorithms in results is seen with the lines. Refer 1

States: [(x,y)]
States are defined as a pair of coordinates (x coordinate, y coordinate). They are 0 indexed.

Actions: AU, AD, AL, AR
At every state, there are 4 actions which are possible: Attempt Up (AU), Attempt Down (AD), Attempt Left (AL) and Attempt Right (AR). These actions are mapped as:

$$AR : [1, 0]$$

$$AD : [0, 1]$$

$$AL : [-1, 0]$$

$$AU : [0, -1]$$

Environment Dynamics:
At every state except cross walks, the action taken succeeds with probability 1 (unless the agent goes out of bounds after action in which case it remains in the same state). At the cross walks, with a probability of 0.2, the action succeeds, and with a probability of 0.8, the action fails and the agent remains in the same state.

d0 and goal states :
The initial coordinate of the agent is always at (0,0). However, there is a distribution over goal states g. At the beginning of the episode, a random goal state is chosen. If the agent makes it to the chosen goal state, it will receive a high reward and terminate, while making it to an unfavorable goal state will receive in a low reward.

$$Goalstate[9, 9] \text{ with probability } \frac{1}{3}$$

$$Goalstate[4,9] \text{ with probability } \frac{1}{3}$$

$$Goalstate[9,7] \text{ with probability } \frac{1}{3}$$

Rewards:

The rewards for the first row (road) of the MDP are -0.5, whereas other coordinates gives a reward of -2. The rewards for the warm buildings is -0.1. This models the road where walking on the road is easier than walking through the snow. At the beginning of the episode, one goal state is picked. If the agent makes it to the right goal state at the end of the episode, it gets a reward of +100. If the agent makes it to the wrong goal state, it gets a reward of +10. These are the rewards for transitioning (entering into) the states mentioned, and hence these are reward states of the form R(s,a,s'). These rewards were modified throughout the implementation of the algorithm to ensure that the agent learnt interesting policies.

Cold factor:

A cold factor is a penalty added to every timestep, a student is in the cold. As compared to reward, this is incremental. Initially, the agent has a cold factor of 1. After one timestep, considering it did not make it to a building, the new cold factor is 2. If the reward for entering the next state was x, with the cold factor, the new reward is $x - 2$. Again on the next timestep, if the reward was y, the new reward for for the state is $y - 3$. As the agent stays out in the cold for longer, it gets a higher penalty for being out in the cold. This is similar to the consequences of initially being in the cold might give you a headache, but being in the snow for long could mean you fall sick and have to miss assignments.

Discount rate and learning rate:

Discount rate gamma and learning rate are hyperparameters which can help finetune algorithms performance

4 Algorithms implemented from Stable Baselines 3

The algorithms which were implemented are PPO, DQN, and TRPO. The results were then plotted using tensorboard to get a better visualization of the data. With this, the plots for finetuning hyperparameters and good runs of the experiments are shown below. A brief summary of these algorithms is detailed below:

Proximal Policy Optimization (PPO): OpenAIdocs [b]

The main principle of PPO is how the policy can be improved in the best possible way using the current information, while ensuring the performance does not degrade. There are two variants of PPO which are often used in Reinforcement Learning- PPO Penalty and PPO Clip. For the experiment, PPO Clip from the OpenAI documentation is used. PPO clip adds a clip to the objective function to ensure that the new policy does not stray from the old policy. For this, the default clip range of 0.2 was used.

Trust Region Policy Optimization (TRPO): OpenAIdocs [c]

TRPO is an algorithm which similar to PPO, works by finding the best possible update by taking a step to improve the policy. It constrains on how far the new policy and old policy is with the help of KL divergence.

Deep Q Learning (DQN): OpenAIdocs [a]

This algorithm combines Q learning with deep neural networks, which enhances the capabilities of the model to perform and learn complex features about MDPs. Using Neural Networks to learn the q value of a state could be extremely useful as a method to scale vanilla Q learning, as the neural network reduces the complexity of calculating the q matrix as done in the homeworks (which grows proportionally to the size of states and actions).

5 Results

With the implementation of the MDP, the returns at the end of training were in the range of [35,42] with different goal states over a series of episodes, except for DQN which performed slightly poorer. The average episode lengths lied between 14 and 18, since the range for different return states was different. All three algorithms found a policy which led them to use the road initially, followed by trying to make it into the warm buildings which are on the way to the final terminal states. A sample episode run with the policy found is given in figure 1

5.1 PPO

The hyper parameter value of learning rate chosen for PPO was 0.0002. Hyperparameters were chosen by running grid search over learning rate and gamma (discount factor). Learning rate was varied in increments, from an initial list of [0.1,0.01,0.001,0.0001], followed by a run over [0.0001,0.0002,0.0005]. Similarly for the discount factor gamma, it was run over the list of [0.1,0.5, 0.9,0.99, 0.999].

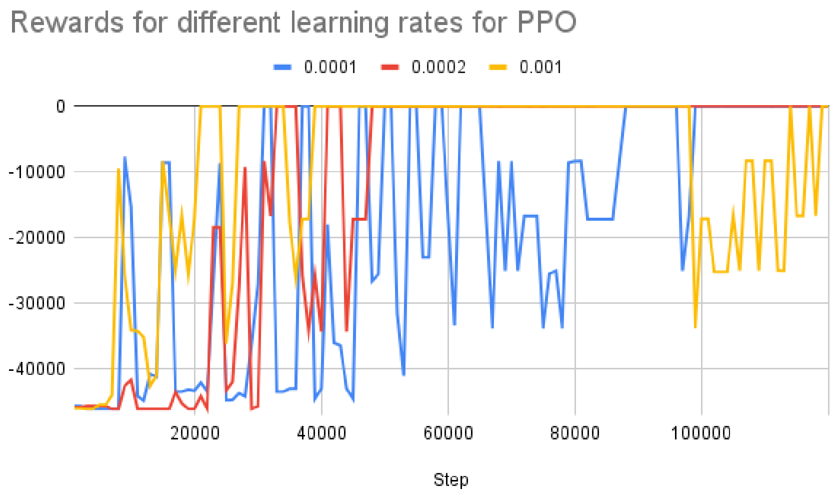

Figure 2: Rewards for different learning rates for PPO

Each parameter was run for 12k iteration, and based on performance, the value of 0.0002 was chosen for learning rate. The discount factor was set to 0.99. The plots 4 and 3 shows the curve between mean reward and iterations with hyperparameters of learning rate of 0.002 and discount factor of 0.99 and step size and iterations.

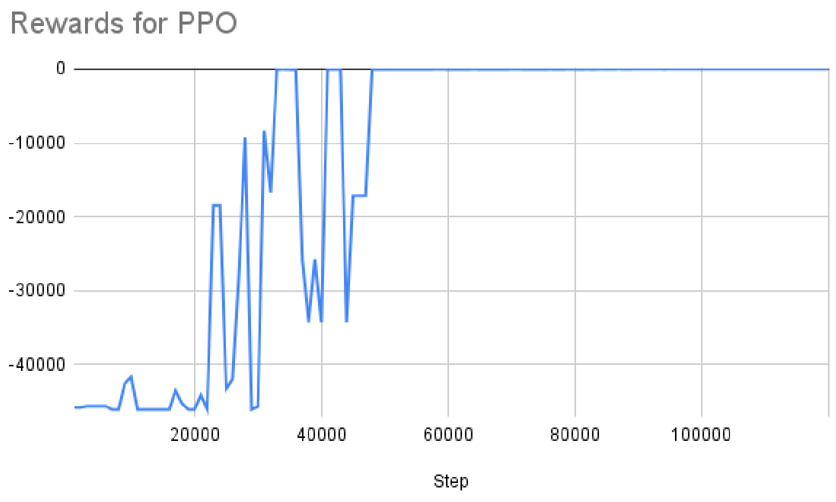

Figure 3: Rewards for PPO

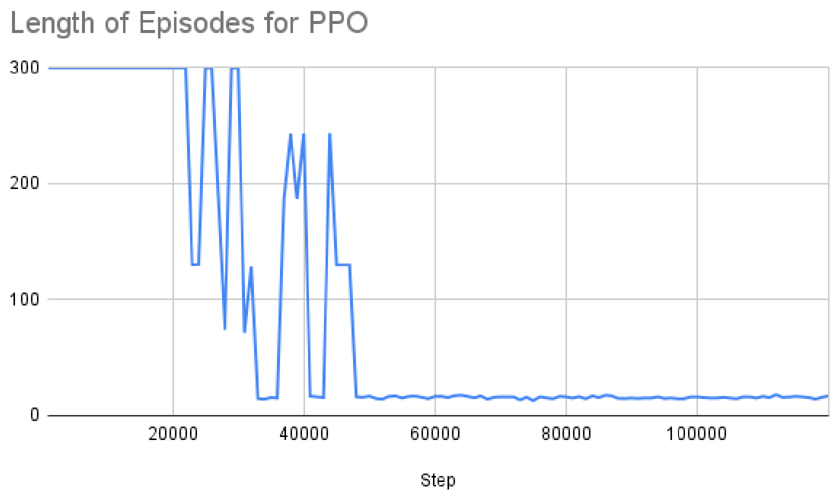

Figure 4: Length of episodes for PPO

5.2 TRPO

For TRPO, below is the learning curve for iterations vs mean reward. Hyperparameters were chosen by running grid search over learning rate and gamma (discount factor). Learning rate was varied in increments, from an initial list of [0.1,0.01,0.001,0.0001], followed by a run over [0.01,0.001,0.0001]. Similarly for the discount factor gamma, it was run over the list of [0.1,0.5,0.9,0.99].

For TRPO, it was noticed that the algorithm converged to high rewards fairly quickly, as compared to the other algorithms which were run. Each parameter was run for 12k iteration, and based on performance, the value of 0.0001 was chosen for learning rate. The discount factor was set to 0.99.

6 and 5 are the plots for the learning rate of 0.001, with different discount factors of gamma of [0.1,0.5,0.9,0.99]. It is clear to see that a high value of discount factor (i.e. the agent looks significantly into the future) and takes actions based on future events ensures that the algorithm does well. The two plots correspond to mean

reward vs iterations, and average episode length vs iteration. The average episode length maximum was set to 300, to keep check of runtime.

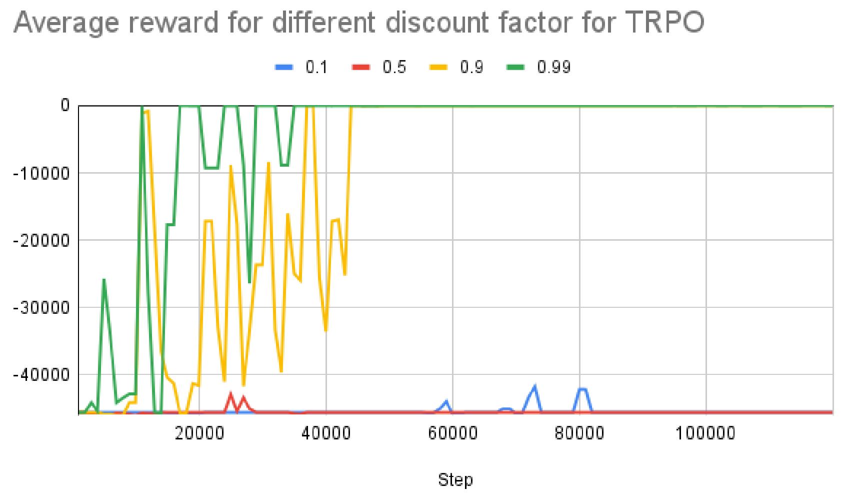

Figure 5: Rewards for different gammas for TRPO

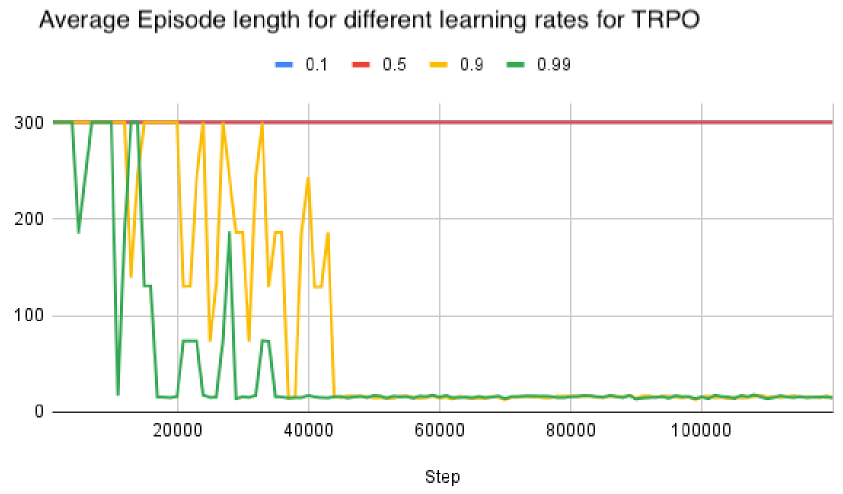

Figure 6: Length of episodes for different gammas for TRPO

The plots 8 and 7 shows the curve between mean reward and iterations with hyperparameters of learning rate of 0.001 and discount factor of 0.99 and step size and iterations.

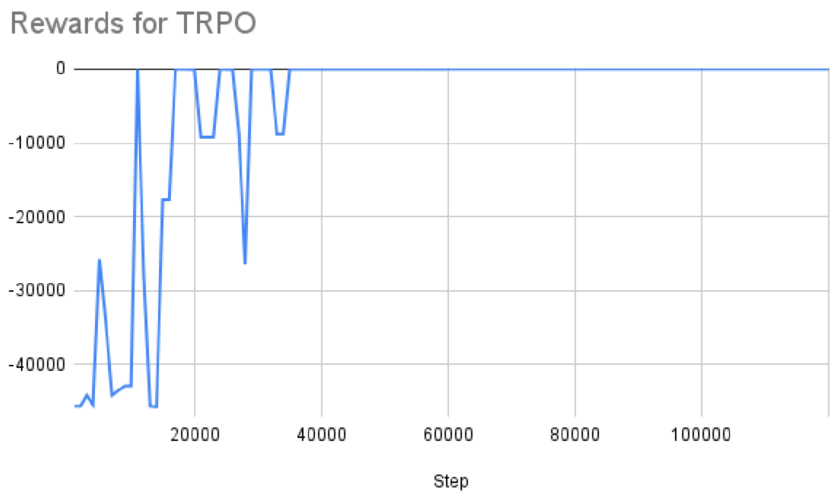

Figure 7: Rewards for TRPO

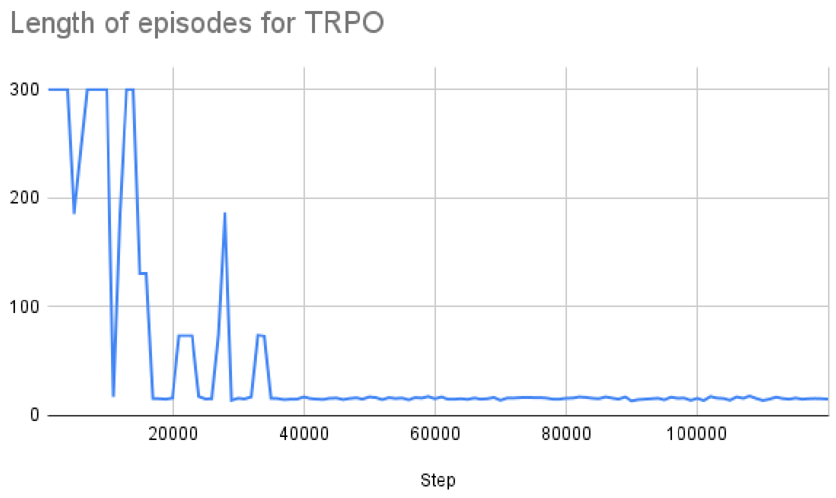

Figure 8: Length of episodes for TRPO

5.3 DQN

The DQN model was significantly harder to train, even with a wide variety of experiments done with hyperparameters, it did not perform as well as the other algorithms, with rewards around 20 instead of 30. The hyperparameters were tuned similar to the previous experiments, and the final learning curve for reward over iterations has been attached along with the loss over one trained model. Refer **??**

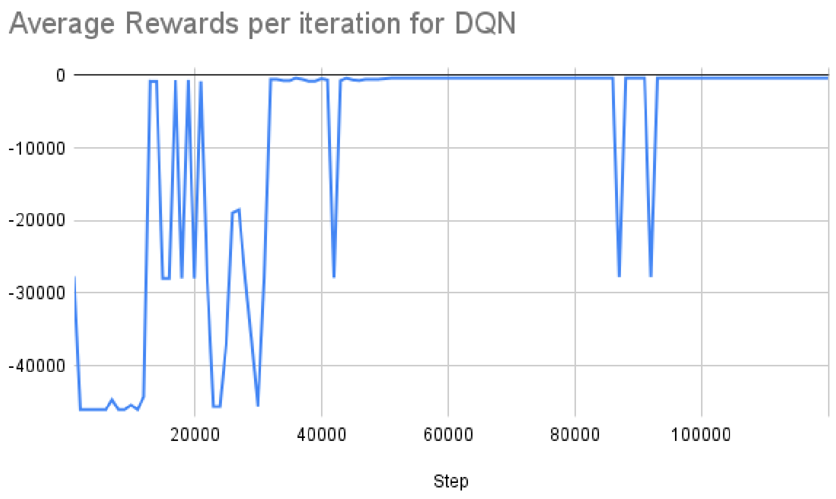

Figure 9: Rewards for DQN

References

OpenAIdocs. Deep q learning documentation, a. URL `https://openai.com/index/openai-baselines-dqn`.

OpenAIdocs. Proximal policy optimization - spinning up documentation, b. URL `https://spinningup.openai.com/en/latest/algorithms/ppo.html`.

OpenAIdocs. Trust region policy optimization - spinning up documentation, c. URL `https://spinningup.openai.com/en/latest/algorithms/trpo.html`.