COL333 Assignment - 3 Report

Aditya (2019CS50471)
Harshith Reddy (2019CS50450)

24^{th} November, 2021

Contents

1 Computing Policies

1.a Model Formulation

State Space:

1. No of possible positions for Taxi - 25 (5*5)
2. No of possible positions for Passenger - 26. Although, there are only 4 possible initial positions for passenger, it is mentioned that he can be dropped anywhere in the grid which gives 25 possible positions for passenger when he is not in Taxi. One more possibility is that passenger is in Taxi. So, total possibilities are 26
3. No of possible positions for destination - 4
4. End state - 1

So, total no of possible states are 25*26*4 + 1 = 2601. Each state is defined by a 3-tuple (Taxi Position, Passenger Position, Destination Position). Each of Taxi Position, Passenger Position, Destination Position have 2 coordinates (x, y). More precisely, each state is defined by ((Taxi X, taxi Y), (Passenger X, Passenger Y), (destination X, destination Y)). When passenger is in taxi, we defined passenger position as (-1, -1) to distinguish it from the states when passenger is not in taxi. End state is defined by a special 3-tuple ((-10, -10), (-10, -10), (-10, -10)).

Action Space:

Each of 2600 states (All except End state) have 6 actions: NORTH, SOUTH, EAST, WEST, PICK_UP, PUT_DOWN.

Transition and Reward Model:

Navigation Actions: If NORTH, SOUTH, EAST, WEST actions are attempted in any of the 2600 states, the probability that agent lands up in correct state is 0.85 and with a probability of 0.15, the agent moves randomly in other 3 directions. This implies that probability to land up in other 3 directions is 0.05 each. There will be no change in the state if agent tries to cross a wall/ move out of grid. The agent receives a reward of -1 for these actions.

PICK_UP: PICK_UP action is deterministic. If PICK_UP action is applied in a state where taxi position is same as passenger position, then agent moves to a state where passenger position is (-1, -1) (as defined above), taxi Position and Destination position remain the same. The agent receives a reward of -1 in this case. If PICK_UP action is applied in any other state, there will be no change in the state. In this case, A reward of -1 is received by the agent if passenger is in taxi and a reward of -10 if passenger is not in taxi.

PUT_DOWN: PUT_DOWN action is deterministic. If PUT_DOWN action is attempted when passenger is not in taxi, there there will be no change in state. In this case, if passenger and taxi are in same position, agent receives a reward of -1. If not, agent receives a reward of -10. If PUT_DOWN action is attempted at destination with passenger in taxi, the agent moves to the exit state. Agent receives a reward of +20 in this case. If PUT_DOWN action is attempted when passenger is in taxi (not at destination), the passenger will be dropped at that location, i.e., passenger position will change from (-1, -1) to taxi position and agent receives -1 reward.

1.b Value Iteration

No of iterations required for convergence = 62
epsilon = 10^{-12}

1.b.1 Discount Factor and Rate of Convergence

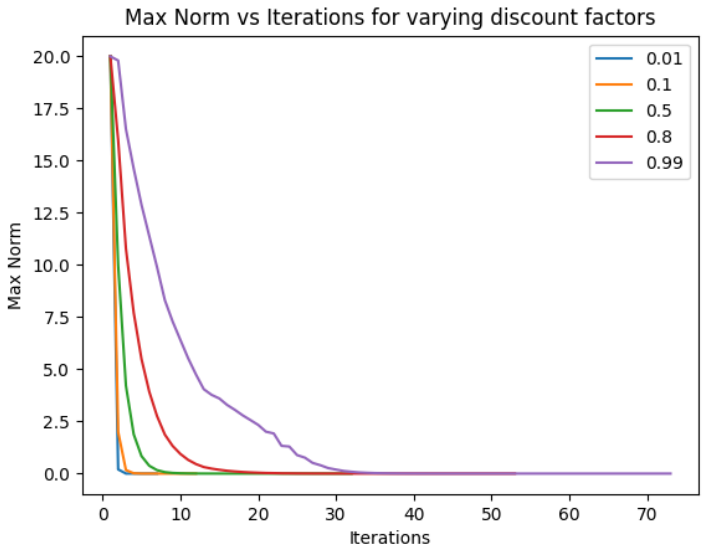

Figure 1: Max Norm vs Iterations for varying discount factors

Discount factor: 0.01 - No of iterations: 7
Discount factor: 0.1 - No of iterations: 12
Discount factor: 0.5 - No of iterations: 32
Discount factor: 0.8 - No of iterations: 53
Discount factor: 0.99 - No of iterations: 73

We can see that values are converging faster if we use a smaller discount factor. This is because, if discount rate is small then the dependence of the value of the state on its neighbouring states (North state, south state, east state, west state, pick up state, put down state) will be smaller. So, values converge faster. Where as if we use high discount rate, the value of a state depends more on the values of its neighbouring states, so it takes more iterations to converge.

1.b.2 Simulation of obtained Policy

As mentioned earlier, each state is defined by a 3-tuple: (Taxi Position, Passenger Position, Destination). The following are first 20 state-action pairs for discount rates 0.1, 0.99.

Initial Taxi Position: (4, 0); Initial Passenger Position: (0, 0); Destination: (4, 4)

Discount Rate - 0.1	Discount Rate - 0.99
((4, 0) (0, 0) (4, 4)) NORTH	((4, 0) (0, 0) (4, 4)) SOUTH
((4, 0) (0, 0) (4, 4)) NORTH	((3, 0) (0, 0) (4, 4)) SOUTH
((4, 0) (0, 0) (4, 4)) NORTH	((2, 0) (0, 0) (4, 4)) SOUTH
((4, 0) (0, 0) (4, 4)) NORTH	((1, 0) (0, 0) (4, 4)) SOUTH
((3, 0) (0, 0) (4, 4)) NORTH	((1, 0) (0, 0) (4, 4)) SOUTH
((4, 0) (0, 0) (4, 4)) NORTH	((0, 0) (0, 0) (4, 4)) PICK_UP
((4, 0) (0, 0) (4, 4)) NORTH	((0, 0) (-1, -1) (4, 4)) NORTH

((4, 0) (0, 0) (4, 4)) NORTH	((1, 0) (-1, -1) (4, 4)) NORTH
((4, 0) (0, 0) (4, 4)) NORTH	((2, 0) (-1, -1) (4, 4)) EAST
((4, 0) (0, 0) (4, 4)) NORTH	((1, 0) (-1, -1) (4, 4)) NORTH
((4, 0) (0, 0) (4, 4)) NORTH	((2, 0) (-1, -1) (4, 4)) EAST
((4, 0) (0, 0) (4, 4)) NORTH	((2, 1) (-1, -1) (4, 4)) EAST
((4, 0) (0, 0) (4, 4)) NORTH	((2, 2) (-1, -1) (4, 4)) NORTH
((3, 0) (0, 0) (4, 4)) NORTH	((3, 2) (-1, -1) (4, 4)) EAST
((4, 0) (0, 0) (4, 4)) NORTH	((3, 3) (-1, -1) (4, 4)) NORTH
((4, 0) (0, 0) (4, 4)) NORTH	((4, 3) (-1, -1) (4, 4)) EAST
((4, 0) (0, 0) (4, 4)) NORTH	((4, 4) (-1, -1) (4, 4)) PUT_DOWN
((4, 0) (0, 0) (4, 4)) NORTH	-
((4, 0) (0, 0) (4, 4)) NORTH	-
((4, 0) (0, 0) (4, 4)) NORTH	-

Initial Taxi Position: (0, 4); Initial Passenger Position: (4, 0); Destination: (4, 4)

Discount Rate - 0.1	Discount Rate - 0.99
((0, 4) (0, 0) (4, 4)) NORTH	((0, 4) (4, 0) (4, 4)) NORTH
((1, 4) (0, 0) (4, 4)) NORTH	((0, 3) (4, 0) (4, 4)) NORTH
((1, 3) (0, 0) (4, 4)) NORTH	((1, 3) (4, 0) (4, 4)) NORTH
((1, 4) (0, 0) (4, 4)) NORTH	((2, 3) (4, 0) (4, 4)) WEST
((2, 4) (0, 0) (4, 4)) NORTH	((3, 3) (4, 0) (4, 4)) WEST
((3, 4) (0, 0) (4, 4)) NORTH	((3, 2) (4, 0) (4, 4)) SOUTH
((4, 4) (0, 0) (4, 4)) NORTH	((2, 2) (4, 0) (4, 4)) WEST
((4, 4) (0, 0) (4, 4)) NORTH	((2, 1) (4, 0) (4, 4)) NORTH
((4, 4) (0, 0) (4, 4)) NORTH	((3, 1) (4, 0) (4, 4)) NORTH
((4, 4) (0, 0) (4, 4)) NORTH	((4, 1) (4, 0) (4, 4)) WEST
((4, 4) (0, 0) (4, 4)) NORTH	((4, 0) (4, 0) (4, 4)) PICK_UP
((4, 4) (0, 0) (4, 4)) NORTH	((4, 0) (-1, -1) (4, 4)) SOUTH
((4, 4) (0, 0) (4, 4)) NORTH	((3, 0) (-1, -1) (4, 4)) EAST
((4, 4) (0, 0) (4, 4)) NORTH	((3, 1) (-1, -1) (4, 4)) SOUTH
((4, 4) (0, 0) (4, 4)) NORTH	((2, 1) (-1, -1) (4, 4)) EAST
((4, 4) (0, 0) (4, 4)) NORTH	((2, 2) (-1, -1) (4, 4)) NORTH
((4, 4) (0, 0) (4, 4)) NORTH	((3, 2) (-1, -1) (4, 4)) EAST
((4, 4) (0, 0) (4, 4)) NORTH	((3, 3) (-1, -1) (4, 4)) NORTH
((4, 4) (0, 0) (4, 4)) NORTH	((4, 3) (-1, -1) (4, 4)) EAST
((4, 4) (0, 0) (4, 4)) NORTH	((4, 4) (-1, -1) (4, 4)) PUT_DOWN

From these tables, we can see that our agent is not following correct policy to reach destination if discount factor is 0.1. But agent is successful in reaching destination using discount factor of 0.99. This is because when discount factor is 0.1, there is less propagation of rewards between states so agent is not able to find correct path to destination. Whereas if discount factor is 0.99, values will be propagated between states and agent will successfully reach destination using prescribed policy. Although value iteration is converging fast if discount factor is 0.1, it is not successful in reaching destination. So, a smaller discount factor is not suitable for this problem.

1.c Policy Iteration

Policy Evaluation Step: Given a policy, values for each state are computed using this equation

$$V^{\pi}(s) = \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma . V^{\pi}(s')] \tag{1}$$

These equations are a system of linear equations, which can be solved using linear algebra methods. Approximate values for the states can be computed using iterative method.

1.c.1 Linear Algebra Method and Iterative Method

No of iterations to converge in Linear Algebra method: 6
No of iterations to converge in Iterative method: 26

Although, no of iterations required are less in linear algebra method, this method takes more time than Iterative method. In our case, for each iteration linear algebra method takes $O(|S|^3)$ time, where as Iterative method takes $O(|S|)$ time. Here $|S| = 2601$. So, for smaller state space linear algebra method is efficient but for larger state spaces this method takes more computation time. So, for larger state spaces Iterative method is preferable. For this problem, as the state space is large iterative method is efficient.

1.c.2 Policy Loss vs Iterations

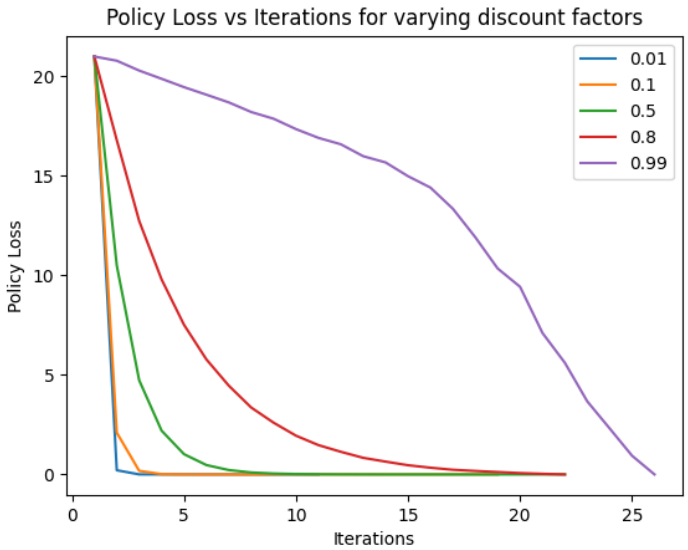

Figure 2: Policy Loss vs Iterations for varying discount factors

Discount Factor: 0.01 - No of iterations: 11
Discount Factor: 0.1 - No of iterations: 19
Discount Factor: 0.5 - No of iterations: 22
Discount Factor: 0.8 - No of iterations: 22
Discount Factor: 0.9 - No of iterations: 26

We can clearly see that policy loss is going to 0 faster than max-norm (in figure 1). For discount factor 0.9, policy iteration took 26 iteration whereas value iteration took 73 iterations. From this plot, we can observe that for smaller discount factors, policy loss is going to 0 quickly when compared to higher discount factors

2 Incorporating Learning

2.a Model Free Approaches

2.a.1 Q-Learning

In this algorithm, we ran 2000 episodes. In each episode, we select a random initial states for Taxi and Passenger (Destination must be same in all episodes because policy depends on destination). We selected an action using epsilon-greedy method. Optimal action at the state is chosen with 1-ϵ probability, random action is chosen at ϵ probability. ($\epsilon = 0.1$). We run each episode until the agent reaches exit state or length of the episode exceeds 500. We find the next state s' and reward R(s, a, s') using simulator. Update equation is given by :

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha[R(s,a,s') + \gamma \max_{a'} Q(s',a')] \tag{2}$$

2.a.2 Q-Learning with decaying exploration

Instead of constant exploration rate for all the iterations in Q-Learning, we decay the exploration rate, inversely proportional to the iteration number. $\epsilon = \frac{0.1}{iteration-number}$.
Agent explores more early but as number of iterations increase it is directed towards the more optimal states. This helps in faster convergence.

2.a.3 SARSA

This is similar to Q-Learning, but SARSA waits until an action is taken and backs up the Q-value for that action. SARSA learns the Q values from actual transitions. This algorithm is more useful if the policy is partly controlled by other agents. Update equation is given by:

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha[R(s,a,s') + \gamma Q(s',a')] \tag{3}$$

2.a.4 SARSA with decaying exploration

Instead of constant exploration rate for all the iterations in SARSA-Learning, we decay the exploration rate, inversely proportional to the iteration number. $\epsilon = \frac{0.1}{iteration-number}$.

2.b Convergence of Algorithms

For every 10 episodes, we calculated discounted reward averaged over 100 runs for each Algorithm.

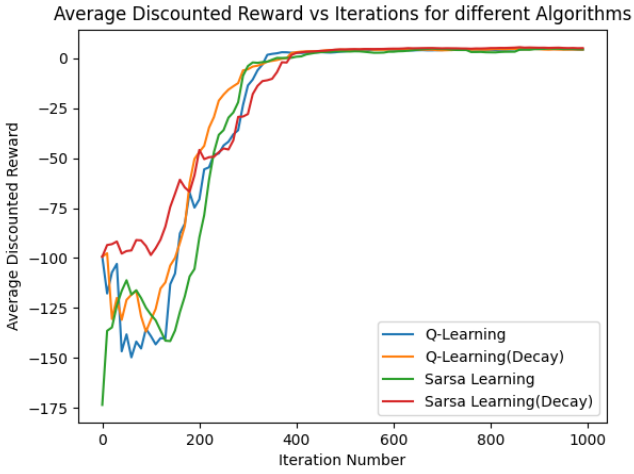

Figure 3: Discounted Reward vs Episodes for different Algorithms

We can see that Q-Learning is converging first. Then, Q-Learning(Decay), SARSA-Learning and SARSA-Learning(Decay) are converging, almost at the same iteration number. So, for this problem Q-Learning is performing better than SARSA-Learning.

2.c Varying Initial Taxi and Passenger Locations

From the above part, we can conclude that Q-Learning is performing better than other algorithms. So, we executed the policy for 5 instances for Q-Learning (learned 2000 episodes). We computed discounted reward averaged over 100 runs. Each instance is defined by (Taxi Position, Passenger Source, Passenger Destination). The 5 instances (varying initial locations of passenger and Taxi) we considered are:

Instance - 1: ((1, 3), (4, 4), (0, 3))
Instance - 2: ((3, 2), (0, 0), (0, 3))
Instance - 3: ((4, 1), (0, 3), (0, 3))
Instance - 4: ((0, 4), (4, 0), (0, 3))
Instance - 5: ((2, 0), (0, 3), (0, 3))

The rewards for these 5 instances are [5.79, 1.07, 9.6, -2.2, 8.15] respectively. The average of these rewards is 4.47.

2.d Varying Exploration Rate and Learning Rate

For every 10 episodes, we calculated discounted reward averaged over 100 runs for each Exploration rate and learning Rate.

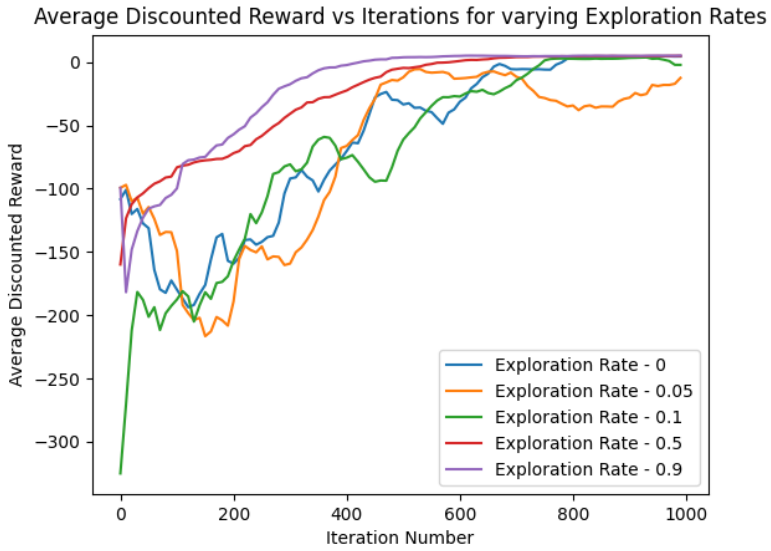

Figure 4: Discounted Reward vs Episodes for varying Exploration Rate

We can observe that Q-Learning with high exploration rate is converging faster than small exploration rates.

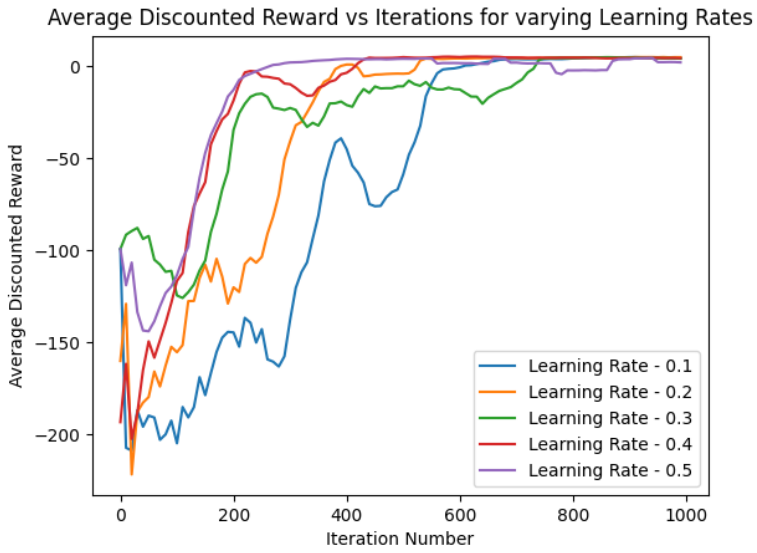

Figure 5: Discounted Reward vs Episodes for varying Learning Rate

We can observe that Q-Learning with high learning rate is converging fast. If learning rate is high, then the model will quickly adapt to the new Q values

2.e Extension to 10 X 10 grid

For 10 x 10 grid, we defined the state space similar to that of 5 x 5 grid, but here we have 100*101*8=80,800 states. Transition model and reward model are similar to that of 5 x 5 grid. We used Q-Learning with a learning rate of 0.25, exploration rate of 0.1 and executed it for 10,000 episodes. We created 5 instances and computed discounted rewards averaged over 100 runs. The 5 instances (varying initial locations of passenger, taxi and destination) we considered are:

Instance - 0: ((1, 3), (9, 5), (9, 0))
Instance - 1: ((3, 2), (6, 3), (9, 8))
Instance - 2: ((4, 1), (0, 4), (0, 9))
Instance - 3: ((0, 4), (9, 0), (5, 6))
Instance - 4: ((2, 0), (0, 9), (1, 0))

Average Discounted Rewards for 5 instances: [-11.46, -6.67, -8.26, -15.93, -24.68] Average Reward -13.40

3 Appendix and Animation

We also implemented an Animation (for both 5 x 5, 10 x 10 grid) of the taxi, where we can see the path taken by taxi to reach destination by picking up passenger. In animation, we pick a random position for passenger, destination, taxi and uses Q-Learning. Then, executes the obtained policy. Instructions to execute animation are given below:
Libraries Used: numpy, matplotlib, pandas, random, sys

3.a Instructions to Run Code

Run python3 A3.py <part-number>
The following are allowed part numbers:

1. A2b: Executes value iteration for varying discount factors and saves plot of max-norm vs Iterations as ValueIteration.png
2. A2c: Executes value iteration for the instance given in the problem statement and prints first 20 states for discount rates 0.1, 0.99
3. A3a: Executes iterative method and Linear algebra method for policy iteration and prints no of iterations taken in both the methods
4. A3b: Executes policy iteration for varying discount factors and saves policy-loss vs Iterations plot as PolicyIteration.png
5. B2: Implements Q-Learning, Decayed Q-Learning, SARSA, Decayed SARSA and saves Average discounted reward vs no of iterations plot as Convergence.png
6. B3: For 5 instances, executes Q-Learning and outputs Average discounted rewards for 5 states and an average of all these 5 values
7. B4: Executes Q-Learning for varying exploration rates and fixed learning rate and saves the plot as VaryingExplorationRate.png. Executes Q-Learning for varying learning rate and fixed exploration rate, saves the plot as VaryingLearningRate.png
8. B5: Executes Q-Learning for 10 x 10 grid, outputs Average discounted rewards for 5 states and an average of all these 5 values
9. Anim5: Shows animation for 5 x 5 grid
10. Anim10: Shows animation for 10 x 10 grid