REINFORCEMENT LEARNING PROJECT

SIT 215

Submitted By: Asim Arshad
Student ID: 219337467

Reinforcement Learning:

According to the (Sutton and Barto, 2018), Reinforcement Learning is an area of machine learning that focuses on teaching machine learning models to make a sequence of decisions in an environment by using rewards and penalties. The agent is taught to make optimal decisions that maximize the long-term rewards in environments that have very complex decision problems.

Taxi Problem:

In its paper, (Dietterich, 2000) says that in taxi problem there is an environment which is designed to train an agent to provide an autonomous taxi service. We need to create a model that teaches the agent to pick up and drop off the passenger by using the shortest path. The map that we have been given is set up on a 5x5 grid that contains pick up and drop off locations, roads and barriers. This means that the taxi has 25 possible locations at any time in the grid. The agent must pick up the passenger from any of the four pickup locations and drop them off at the drop off location, meanwhile avoiding all the barriers. The agent is shown as a yellow rectangle on the grid. The pickup and drop off locations are named as R, G, Y, B with pink as pickup and gray as drop-off locations. There are barriers or walls represented by "|" which the agent must avoid and change its path if encountered by them.

The total state space of the grid denotes the likely states our agent can be at any time. There are 4 destinations and 5 passenger locations (4 locations + 1 when the passenger is in the taxi) so, that makes it a total of 500 states.

5*5*5*4 = 500 (all possible states)

There are 6 movements or actions our agent can take in the state space so we can say that we have an action space size of 6. All the actions are given below:

1. South
2. North
3. East
4. West
5. Pickup
6. Dropoff

The environment will be initializing a random state and then out agent will try to take optimal decisions to pick up and drop off the passenger. There are rewards and penalties involved for the agent's action so that it would help It to train better and make optimal decisions that return the maximum possible reward.

The rewards and penalties of the environment are represented below:

Action	Reward/Penalties
Movements	-1
Successful Dropoff	20
Invalid pickup/Dropoff	-10
Valid Pickup	-1

We applied two methods to solve this problem i.e. Random Policy and Q learner algorithm. Both methods are listed below:

Random Policy:

In random policy, our agent makes random moves without any intelligence until they find the correct pickup or drop off locations. There is no learning method involved in random policy. Our program for random policy is an infinite loop which runs until the passenger reaches the drop off location and we get a reward of 20. One successful pick up and drop off is counted as one episode. The pseudocode for random policy is given below:

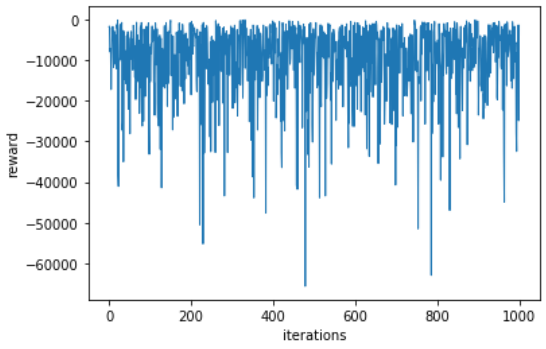
While (reward != 20):
    Take random action
    Change the current state to the new state

By using the random policy, we get the following stats:

Time steps taken: 1117

Penalties incurred: 363

After running over 1000 episodes we get the following graph:



This clearly shows that this isn't a feasible way to solve our problem as it's taking a lot of timesteps and penalties.

Q-learning Algorithm:

The developer and the author who gave the convergence proof the of Q-learning, (Watkins and Dayan, 1992), defines it as an approach of reinforcement learning in which the agent takes the most optimal action at any state in the environment. The Q learning algorithm learns from the environment's rewards and produces a Q-value that represents the optimal actions our agents can take at any state in order to maximize the total reward. Q learning observes the state of the environment and then based on some arbitrary rate of epsilon (ϵ), our agent will then choose to either explore the environment or exploit the present policy. If the agent chooses to explore, then it will act as a random agent and make a random move or if it chooses to exploit, the agent will refer to the Q- table choose the best action that can maximize the reward. This ensures that the agent doesn't take the same path every time and explores the state space.

The mathematical model of Q-learning is given below:

$$Q(state, action) \leftarrow (1-\alpha)Q(state, action) + \alpha\Big(reward + \gamma \max_{a} Q(next\ state, all\ actions)\Big)$$

Here, α is the learning rate ($0 < \alpha \leq 1$). α is the extent to which the Q-values are being updated in every iteration. γ is the discount factor ($0 \leq \gamma \leq 1$) which determines that how much importance should be given to the future reward. Higher value closer to 1 captures long term effective award while closer to 0 captures short term reward, thus it makes the agent greedy as it only cares about the immediate reward and not the long-term award. The arrow shows that we are updating the Q value of the agent's state and action at the left side by adding the weight of the old Q value to the learned value (i.e. The combined reward of the current state's action and the discounted reward of the next state). We store these Q values for every state and action in a Q table.

The Q table is matrix which has a row for every state and a column for every action. For the taxi problem, we have 500 rows and 6 Columns. All the values are first initialized to zero and then updated during training of the agent which then optimizes the agent's traversal. After implementing the Q learning algorithm on our taxi problem, we got significantly better results. The pseudocode for our Q learning algorithm is given below:

For 100000 episodes:

    While reward!=20:

If random value is smaller than epsilon value:

Explore the action space

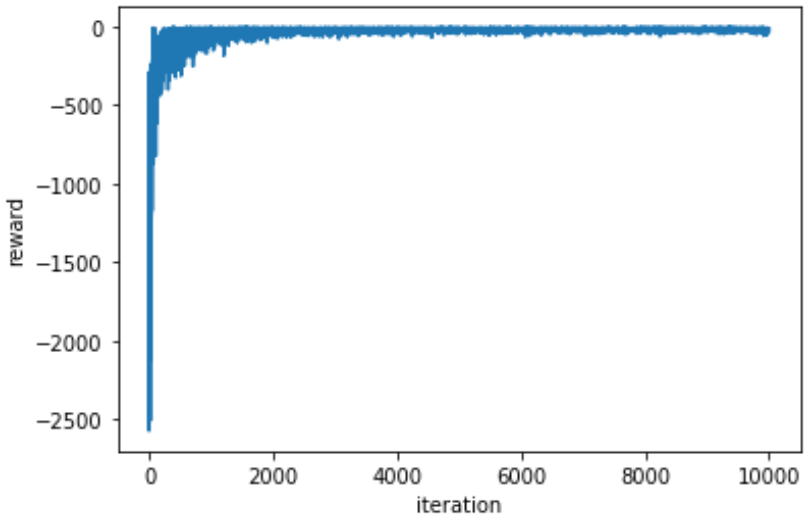
Else:

Use the learned values

Update the q value in the Q table

Change the current space to the new space

After learning through this method, we evaluated the agent's performance using the Q values and got significantly better results. As shown in the graph below, our agent took more optimal decisions over the iterations and learnt from its mistakes and in the end it started to get the maximum possible reward.



Cartpole Problem:

The cartpole problem is a continuous control problem in which there is a cart that moves on a frictionless track with a pole attached at the middle and we have to move the cart left(-1) or right(+1) to balance the pole. Our goal in this problem is to keep the poll from falling over.

The difference between the cartpole problem and taxi problem is that of state space. The cart pole problem has a continuous state space while the taxi problem has discrete

state space. The Taxi problem has clear tasks and goals to achieve while the cartpole problem only has one task and goal of balancing the pole.

To solve the problem, we use Markov decision process and break our problem into possible states, actions and rewards. The reward of +1 is provided for each timestep our pole remains standing. In the equation given below, the t represents one-time step.

$$Rt1 = Rt0 + 1$$

To represent the state of our cart, the environment provides following variables:

Observation	Min	Max
Cart Position	-4.8	4.8
Cart velocity	-∞	+ ∞
Pole Angle	-24 degrees	24 degrees
Pole velocity at the tip	-∞	+ ∞

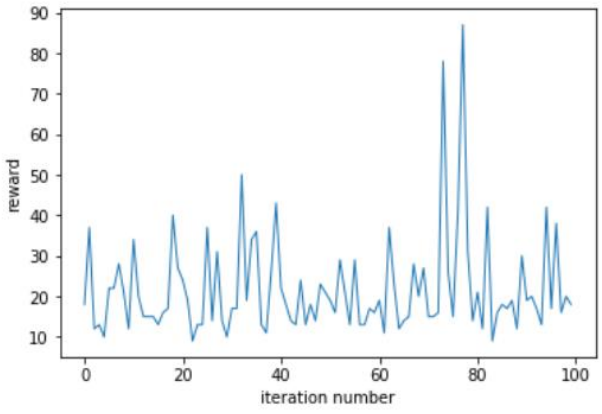
For solving the cartpole problem we used two methods, both are listed below:

Random Policy:

Just like the taxi problem, we implemented the same method for cartpole problem and made random moves. The agent failed badly to balance the pole and it kept on falling because it was moving without any intelligence and made random moves. The reward scores that we got were very low and our average timesteps for 10000 episodes is given below:

```
Results after 10000 episodes:
Average timesteps per episode: 22.1544
```

The graph for rewards after every iteration is given below and it clearly shows how badly the random agent performs:

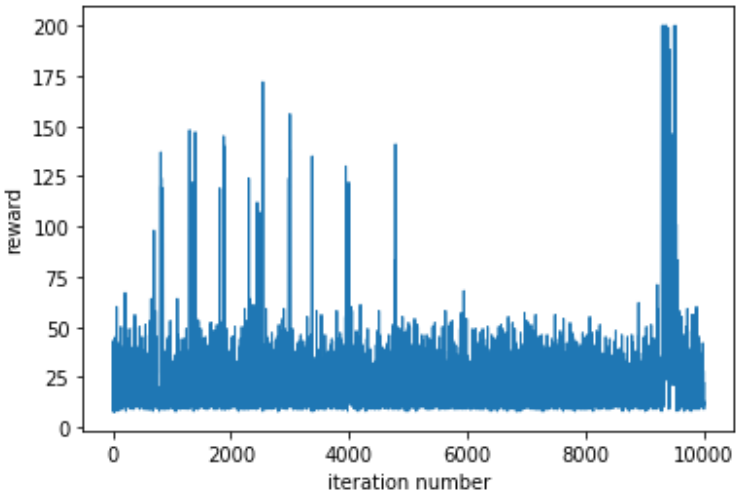


Q-learning Algorithm:

To implement the Q learning algorithm on the cartpole problem, we first need to convert the continuous variables of cart pole state space to discrete values as it is a requirement of Q tables. The two of our variables are infinite and we need to assign them limits as otherwise Q

learning can't be implemented. As we are forced to decrease the amount of information, we feed our agent, it can have negative effects and make our training process very inefficient.

We created discrete buckets for the representation of our variables. These buckets now become the hyper parameters for our algorithm. The algorithm was run over many episodes but the Q learning algorithm didn't run as better as it did for taxi problem. We got better rewards than random policy but still we couldn't get as good of results as expected. The learning of our algorithm stopped after a certain number of iterations. The reason for this failure can be the loss of information after discretization of our variables.



The Graph shows that our reward peaked near 9000 mark but then dropped, this shows that our algorithm failed to learn and keep the pole from falling over.

Frozen Lake Game Problem

The game consists of a rectangular grid wherein some tiles of the grid are walkable, and others lead to the agent falling into the water. Additionally, the movement direction of the agent is uncertain and only partially depends on the chosen direction. The agent is rewarded for finding a walkable path to a goal tile.

The surface is described using a grid:

SFFF (S: starting point, safe)

FHFH (F: frozen surface, safe)

FFFH (H: hole)

HFFG (G: goal)

The episode ends when you reach the goal or fall in a hole. You receive a reward of 1 if you reach the goal, and zero otherwise.

Let's say that *SH* is the state in which the agent falls in a hole; and

SG is a state in which the agents reach to the goal;

R0 is a reward of 0;

R1 is a reward of 1;

So mathematically:

SH → R0

SG → R1

A reinforcement learning agent will therefore learn to avoid the holes and to seek out the goal.

Solving Frozen Lake Game Problem using Q learning

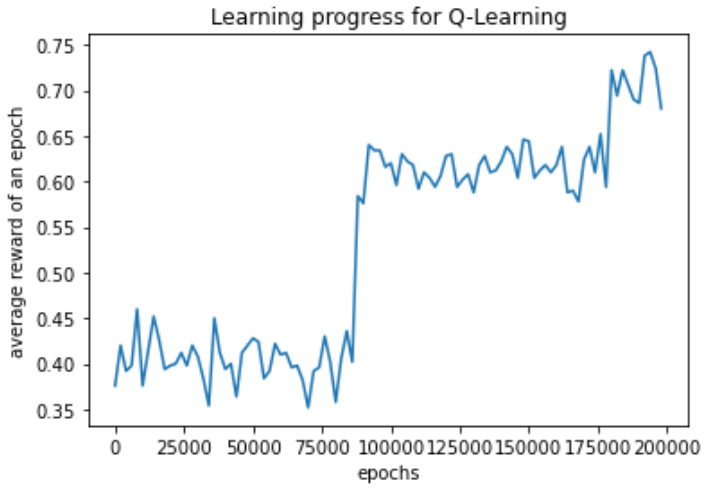
The q-learning implementation on the Frozen Lake Game can be rightly compared to the Taxi problem. Like Taxi Problem and unlike cart-pole problem it has discrete observation space. The State's discrete observations are mapped to four actions available. The q table where this mapping is stored become more accurate.

We ran the algorithm for 200000 episodes. The reason for such a high number is to evaluate the performance in a more precise way. The average reward for the 2000 steps has been calculated.

Now let's look at the results we got.

Average Timesteps Per Episode: **233.4129**

Success Percentage: **38.89949**

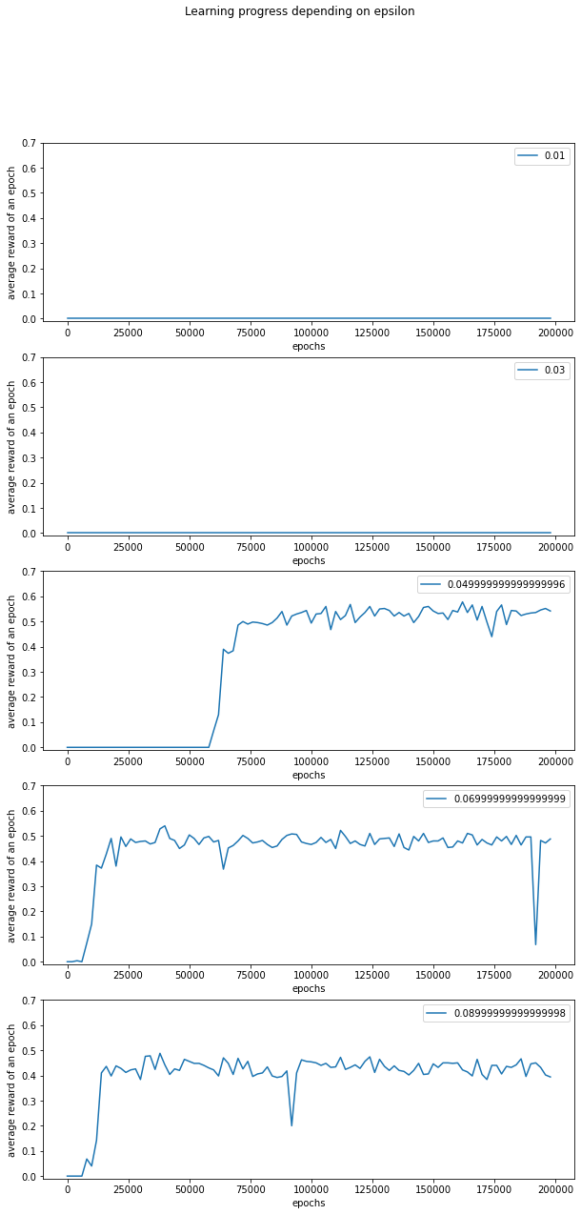


The Q-learning starts learning very quickly. With the passage of more episodes the average no of rewards is increasing. Somewhere near 90000 epochs, a sudden hike in average rewards can been seen. As the average no of rewards are nearing to 1 in the episodes after 100000, it can be said that q-learning can perform well on the frozen-lake problem if the no of epochs is kept larger. On average, it still performs good.

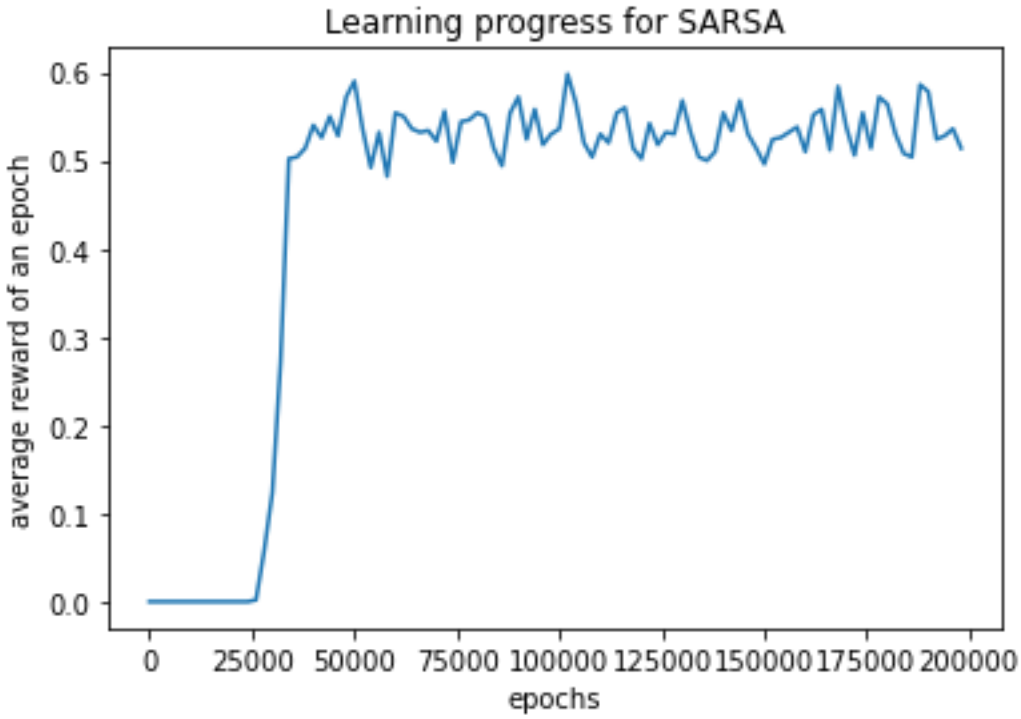
Temporal Difference Learning:

The question is bit confusing, because the Q-learning is itself a part of temporal learning, which is a broad category of modal-less Reinforcement Learning. Well to answer the question I comprehended the question as the comparison of Q-learning and SARSA (State, Action, Reward, State, Action). SARSA and Q-learning are two types of Temporal Difference Learning. The SARSA was proposed by the (Rummery and Niranjan, 1994). The SARSA is first implemented on the Frozen-Lake Problem, chosen environment for preceding question. Then the SARSA is implemented on the

At first, we tried to calculate the optimal epsilon value by implementing SARSA. Then a final SARSA was again implemented using the optimal epsilon value



Learning progress on SARSA depends on epsilon. For different epsilons learning progress was calculated. It was founded that greater of epsilon, the quicker it starts to learn. But the average reward value is also decreasing with the increasing epsilon. We chose the optimal value of epsilon as “0.05”



As it can be seen, unlike q-learning, SARSA was slow to start learning. For first 25000 or so episodes, it has not learned anything. The graph is quite flat for those epochs. However, just right after that, it has taken a sudden spike in the average rewards. Then the SARSA performed quite good, and results can be compared with the Q-learning. However, Q-Learning gained higher scores after 100000 episodes.

Let’s see average timesteps and success rate:

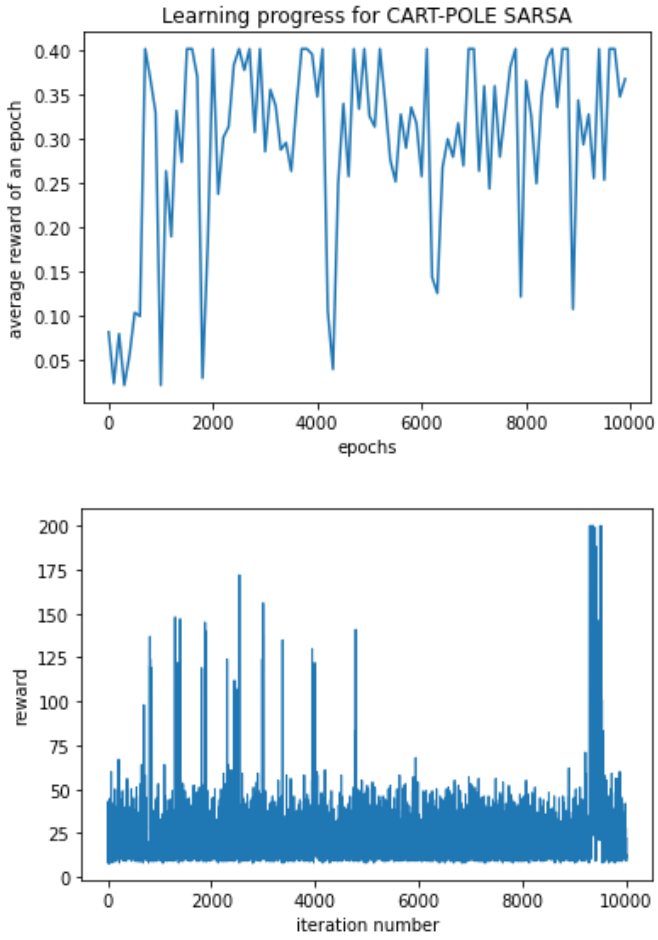
```
Average Timesteps Per Episode:  736.0885
Success Percentage:  42.5295
```

The success rate and the average timesteps per episodes, both are higher than the q-learning. The reason for higher success rate is elude.

With the above comparison, it can be concluded that the environments like Frozen Lake, Q-learning can be a better option. It started to learn quick and gave better results, apart from success rate.

SARSA on Cart-Pole

The SARSA was also implemented on the Cart-Pole problem. The findings of this implementation are given below. This was implemented for 10000 epochs. Just like our q-learning implementation on the cart-pole.



Q-Learning Implementation on Cart-Pole

The SARSA for Cart-pole is very quick to start learning quickly on Cart-pole, just like the q-learning. The result of SARSA is looking better than the Q-leaning on the Cart-pole. It is gaining not as good results as it gained on Frozen Lake Problem, but with comparison to Q-learning on Cart-pole, it seems better. However, this finding is not concrete because both of these approaches are not quite stable and not converging even after 7 to 8 thousand episodes. The SARSA, however, seems more stable than the q-learning here.

References

Dietterich, T. G. (2000) ‘An overview of MAXQ hierarchical reinforcement learning’, in *Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science)*. Springer Verlag, pp. 26–44. doi: 10.1007/3-540-44914-0_2.

Rummery, G. A. and Niranjan, M. (1994) ‘On-line q-learning using connectionist systems cued/f-infeng/tr 166’, *Update*, (September).

Sutton, R. and Barto, A. (2018) ‘Reinforcement learning: An introduction’.

Watkins, C. J. C. H. and Dayan, P. (1992) *Q-Learning*, *Springer*. Available at: https://link.springer.com/content/pdf/10.1007/BF00992698.pdf (Accessed: 3 October 2020).