Project 4: Reinforcement Learning

Sahil Suhas Pathak
Department of Computer Science and Engineering
University at Buffalo
Buffalo, NY, 14214
sahilsuh@buffalo.edu
UBITName: sahilsuh, Person Number: 50289739

Abstract

Our task is to teach the agent (Tom) to navigate in the grid-world environment and find the goal (Jerry). The report addresses the implementation of the same where a combination of reinforcement learning and deep learning, a Deep Q Network is used. The report also illustrates the necessary findings and the conclusions drawn from the execution.

Note: Solutions for the Writing Tasks, Question 1 & 2 included.

1 Implementation

1.1 3 – Layer Neural Network Using Keras Library

Overview:

This problem is a combination of Deep learning and Reinforcement Learning. In the default configuration, we have a three-layer neural network with two hidden layers.

1. The model's structure is: LINEAR - RELU - LINEAR - RELU - LINEAR.
2. Activation function for the first and second hidden layers is 'relu'.
3. Activation function for the final layer is 'linear', which returns the real values.
4. Input dimentions for the first hidden layer equals to the size of the observation space (state_dim).
5. Number of hidden nodes is 128.
6. Number of the output are of the same size as that of action_dim.

Following gives a fair bit of idea about the flow of execution:

1. First we initialize the environment by calling the reset() method
2. The initital state returned by the reset() method is then passes to the agent's act() method
3. Agent takes an action and the action is returned to the environment's step() method.
4. Step() method returns the next_state, reward and also a Boolean factor indicating whether the episode is over or not.
5. Then the agent's observe() method is used to save the observation tuple to memory.
6. Afterwards, training is done. This is where the Neural Network plays its role. Replay() method is responsible for training of an agent.
7. Finally, the render() method is invoked to display the possible action sequences.

Why are we using the Neural Network?

The general idea behind the Neural Network is to maintain the experience rather the behavior of the agent and the environment. Every time when an agent performs an action, we try to update the Q – Value for that particular state. If we do not maintain the updated information, we won't be able to use it in our future calculations. The main reason of having a neural network is to gather all the required information about the environment. Also, we are sending our experiences to the replay() method, which is used for training purpose.

Also, it is not feasible to have a Q – value entry for every state in Q – table when there are numerous states.

1.2 Exponential Decay Formula for Epsilon

Epsilon is also referred to as "Exploration Rate". Initially, we start with random values of epsilon. This random values will enforce agent to choose random actions at the very start. This is preferable because, we want our agent to choose random actions at the start so that it explores all the possible rewards of the surrounding states. As the training progresses, we want our agent to decrease the number of random actions and thus we intend to decrease epsilon. The idea behind how an agent will pick an action is, if the random value of an action is lower than epsilon, then it will pick the action randomly and if it is above epsilon, it will pick the best action. Thus, if we decrease epsilon, the randomness is decreased as the agent tends to choose the best possible action in every scenario.

1.3 Q - Function

Q - Function, Q – table is a look up table for reward associated with every state-action pair. Each cell in this table records a value called as Q – value. It is a representation of the long term reward an agent would receive when taking this action at the particular state.

How does an agent learn about this long-term Q – value reward?

- Q – learning forms the basis of learning long term reward. Reinforcement learning is basically learning about Q - values while taking actions.

Procedure of Q – Learning:

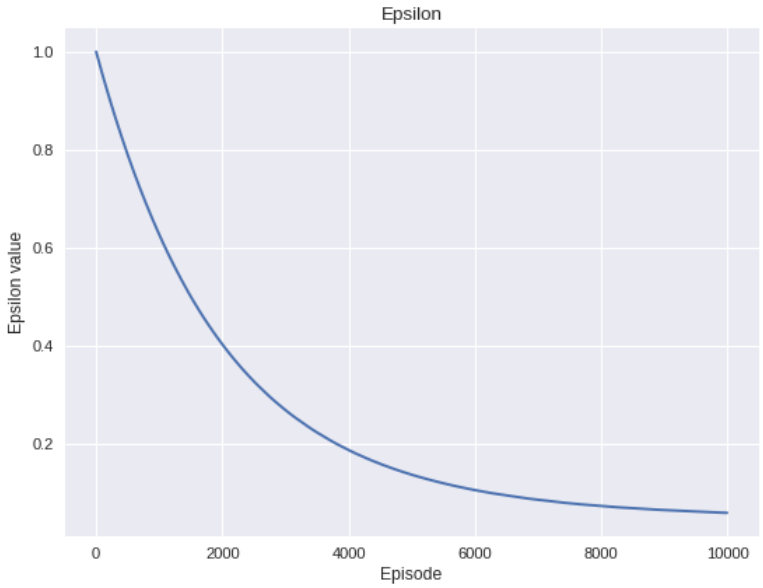
- At first, the agent initializes Q – values to random, for every state action pair. This is essentially saying that we have no or little information on long term reward for each state action pair.
- After the agent starts learning, it takes an action 'a' in a state 's' and receives reward 'r'. It also observes that the state has changed to a new state 's''. The agent then updates the Q – value entry as, $Q(S, a) = r + \gamma * \max(Q(S\_,a))$
- The new long term reward is the current reward 'r' plus all future rewards in the next state s' and later states.
- The future rewards are discounted by a discount rate between 0 and 1, meaning future rewards are not as valuable as the rewards now.
- In this updates, Q carries memory from the past experiences and takes into account all future steps.
- As the agent visits all the states and tries different actions, it eventually learns the optimal Q – values for all possible state-action pairs.

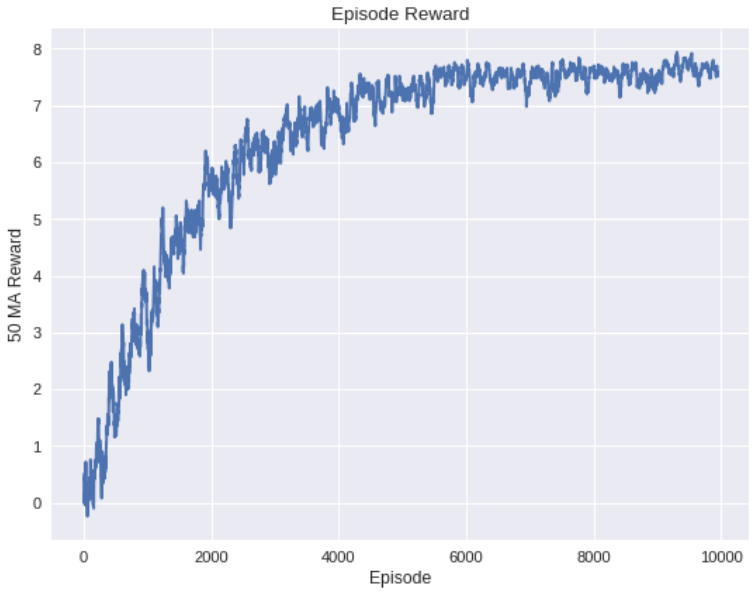
2 Tuning Hyper-parameters

2.1 Epsilon Min/ Max

Observation No. 1: (Default Configuration)
Epsilon_max: 1
Epsilon_min: 0.05

Episode: 9900
Time Elapsed: 972.01s
Epsilon: 0.06042766097583241
Last Episode Reward: 8
Episode Reward Rolling Mean: 6.3141516171819205

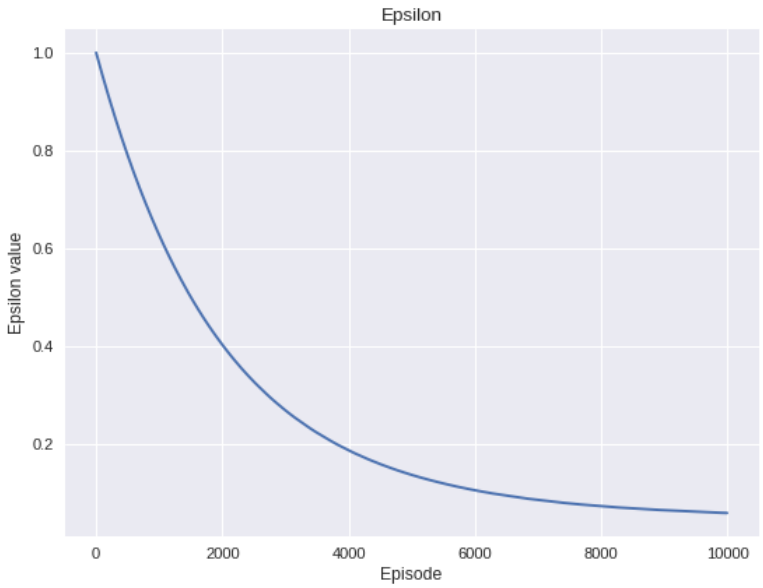


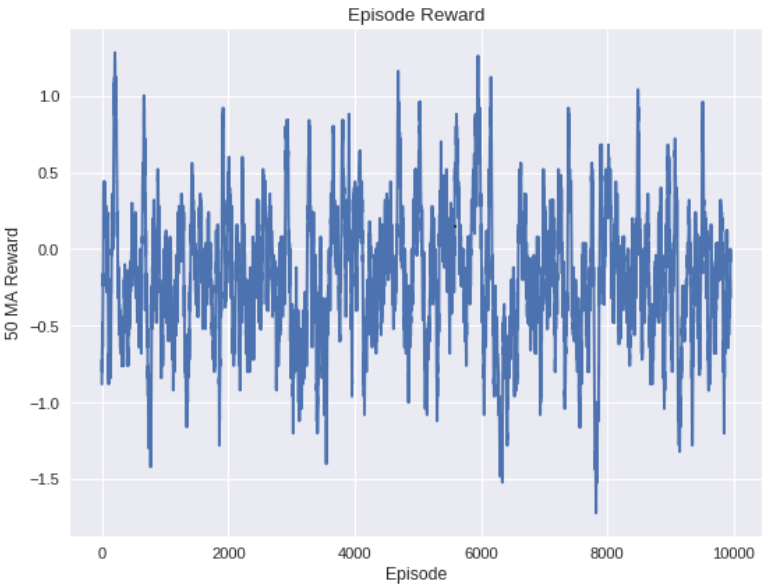


Observation No. 2:

Epsilon_max: 5
Epsilon_min: 1

Episode: 9900
Time Elapsed: 886.06s
Epsilon: 1.0283662381400829
Last Episode Reward: -2
Episode Reward Rolling Mean: -0.1868176716661565





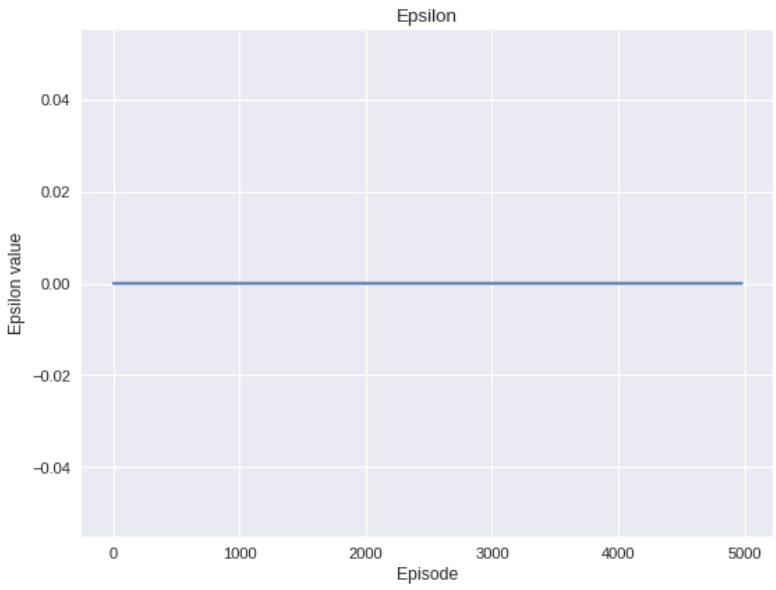
Inference: By keeping epsilon_max = 5 and epsilon_min = 1, the agent poorly performs in terms of finding its goal state. From the above graphs, we can observe that eventhough the epsilon value is exponentially decreasing over the number of episodes the reward is fairly distorted throughout the run of the episodes. It seems that the agent fails to maintain the balance between the exploration and exploitation.

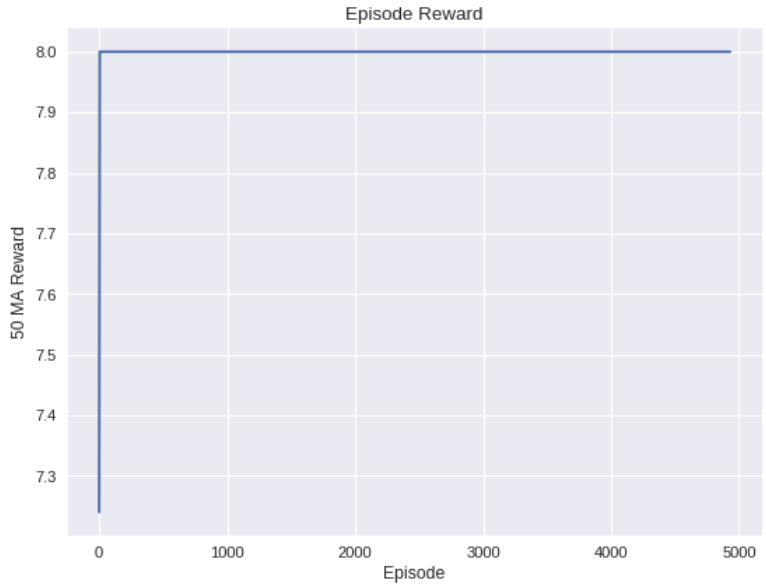
Observation No. 3:

Epsilon: 0
Number of Episodes: 5000

For all the episodes count ranging from 100 to 4900, the reward remained constant with the value 8, Also the rolling mean reward was 7.6237 at the very start of exploration i.e. 200 episodes and remained fairly constant at 7.99 from the episode count 3900 to 4900.

Episode: 4900
Time Elapsed: 544.75s
Epsilon 0
Last Episode Reward: 8
Episode Reward Rolling Mean: 7.9920849822953555

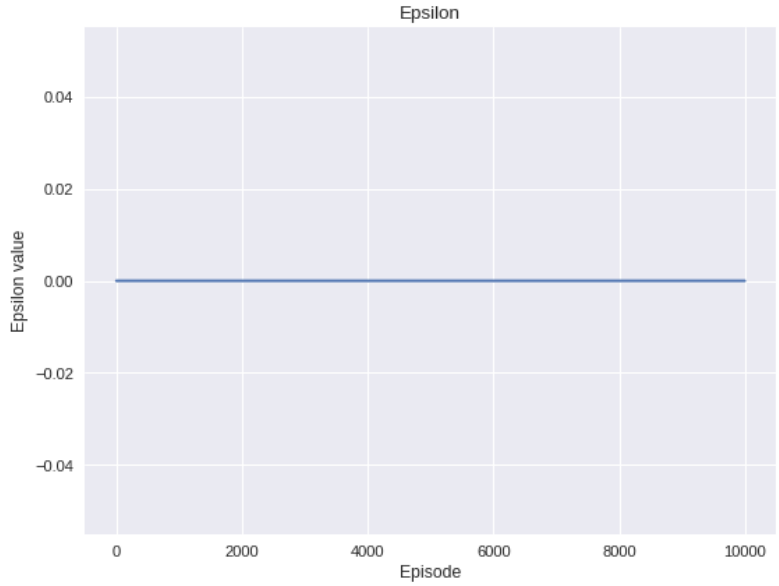


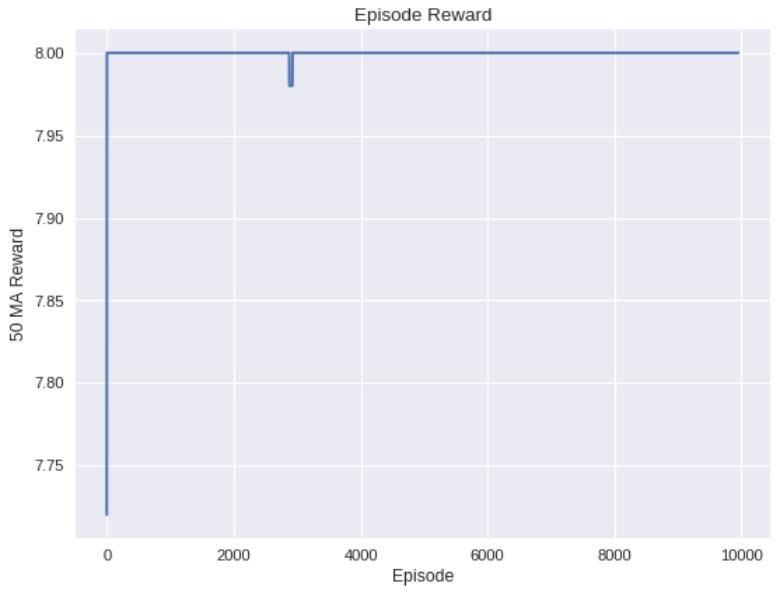


Observation No. 4:

Epsilon: 0

Episode: 9900
Time Elapsed: 725.33s
Epsilon: 0
Last Episode Reward: 8
Episode Reward Rolling Mean: 7.99846954392409



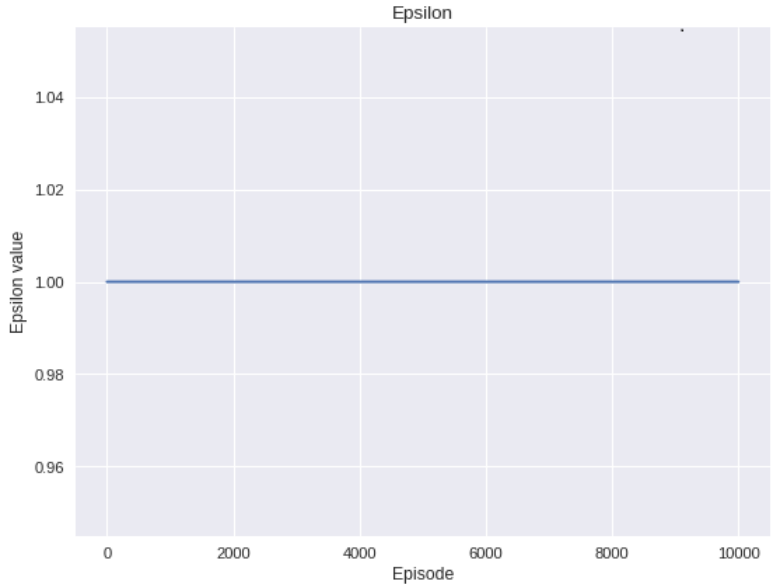


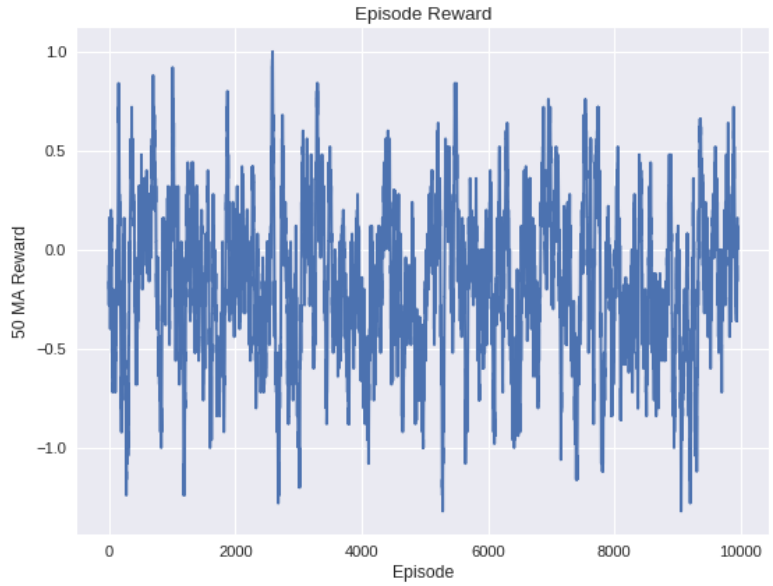
Inference: Considering Observation 3 and Observation 4, By looking at their respective animation, we can see that the agent (Tom) just directly goes towards left and then moves down towards the goal (Jerry). The agent fails to explore any other paths even though there are more than one paths for the agent to reach the goal. Here, in our example, there are actually 3 optimal paths for our agent (Tom) to reach Jerry. But still Tom preferred the same path. Probably it happened because tom just choose that path which gave him the best reward. He never took future rewards into consideration and thus failed to discover more paths that could have fetched more reward. The agent just considered one path which supposedly turned out to be the optimal path. Since our working environment (grid) was small, tom was able to find the optimal path to jerry but not always this would be the case. In complex environments, without exploring properly, the agent might choose sub optimal path or even a longer path towards goal just by relying on the rewards the agent is getting. We need some sort of exploration at the beginning so as to maintain balance between exploration and exploitation. We can define exploration as random fluctuations made by the agent between the states so as to explore the surrounding and get a fair bit of estimate of how rewards can be when particular action is taken. Setting epsilon = 0 turned out to be a scenario of exploitation where there was negligible exploration and the actions were taken just on the rewards obtained due to the current action without any impact of future rewards.

Observation No. 5:

Epsilon: 1

Episode: 9900
Time Elapsed: 845.71s
Epsilon 1
Last Episode Reward: 2
Episode Reward Rolling Mean: -0.1776349352106928





Inference: Considering the Observation No. 5, when epsilon is set to 1, out agent poorly performs when it comes to finding an optimal path to the goal. This is because, when Epsilon is set to 1, most of the times, the actions are chosen at random and because of this randomness, our agent is not able to formulate a correct path towards goal. Also, this randomness is affecting the computation of Q - Values. As mentioned earlier that it is expected for the agent to explore a bit at the very start and then it should converge to a point where it always chooses the optimal path towards the goal. The existence of randomness till the end is making it difficult for the agent to exploit the environment.

2.2 Discount Factor (γ)

Observation No. 1 (Default Configuration):

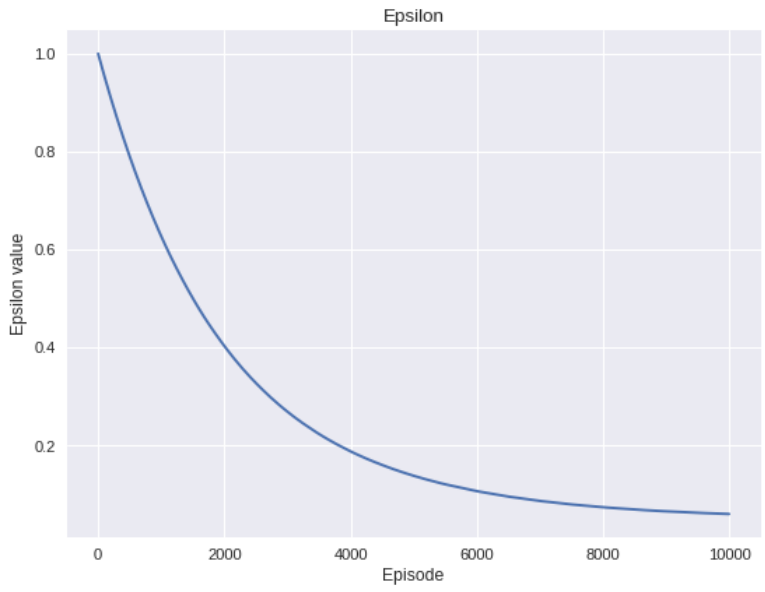
Gamma: 0.99

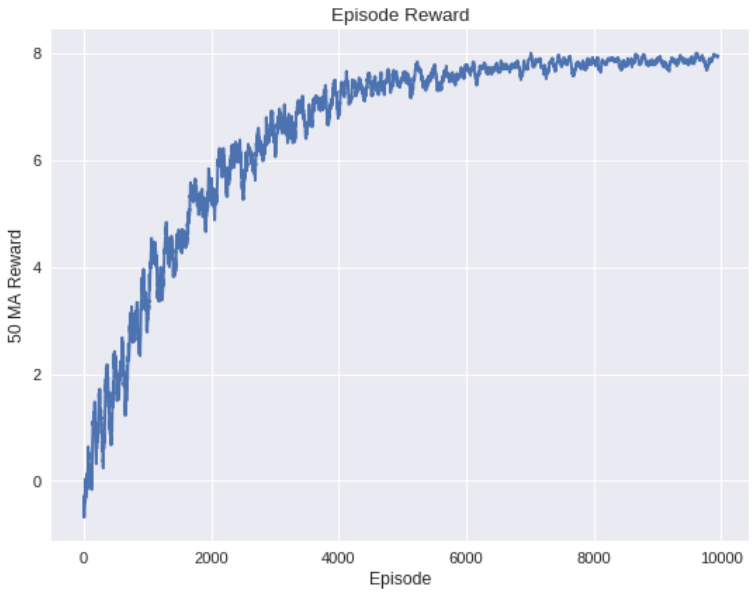
Episode: 9900
Time Elapsed: 1083.32s
Epsilon 0.060315648195071646
Last Episode Reward: 8
Episode Reward Rolling Mean: 6.29252117130905

Observation No. 2:

Gamma: 0.1

Episode: 9900
Time Elapsed: 1158.49s
Epsilon 0.06085702118577194
Last Episode Reward: 8
Episode Reward Rolling Mean: 6.483726150392817

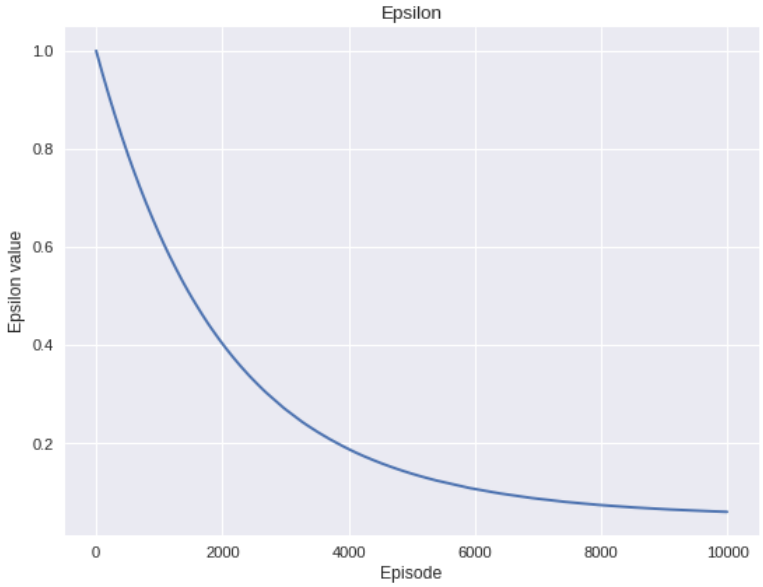


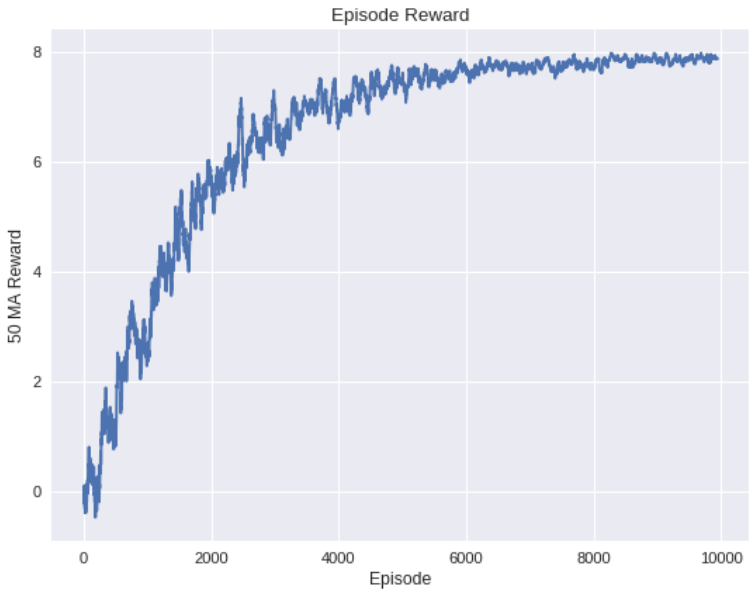


Observation No. 3:

Gamma: 0.5

Episode: 9900
Time Elapsed: 1186.14s
Epsilon 0.061794838076225184
Last Episode Reward: 8
Episode Reward Rolling Mean: 6.444745339027184

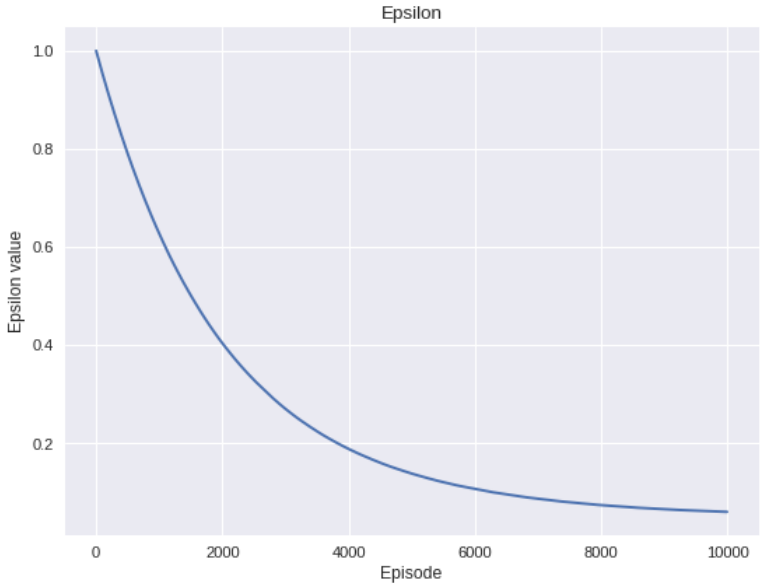


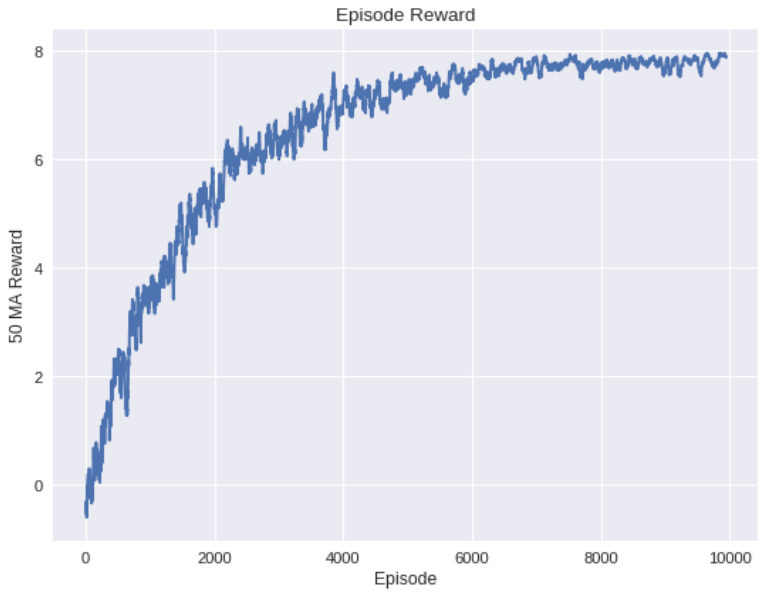


Observation No. 4:

Gamma: 0.7

Episode: 9900
Time Elapsed: 899.65s
Epsilon 0.06074093329836808
Last Episode Reward: 8
Episode Reward Rolling Mean: 6.41352923171105



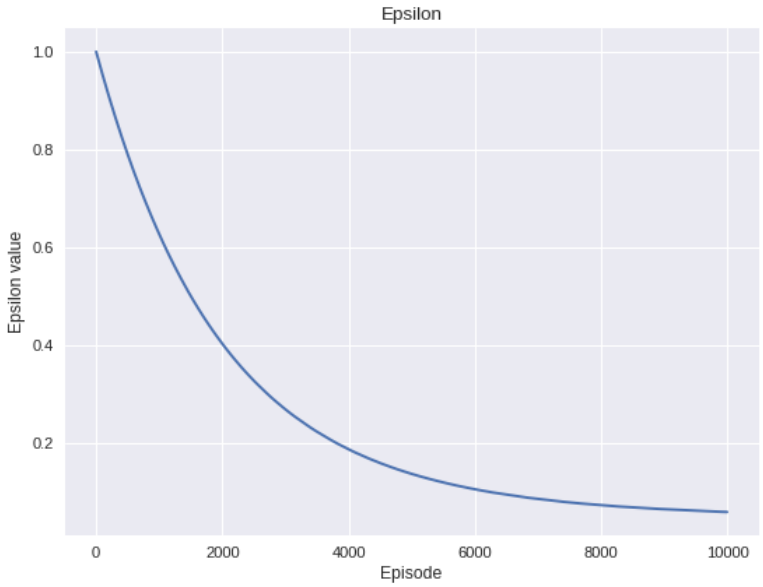


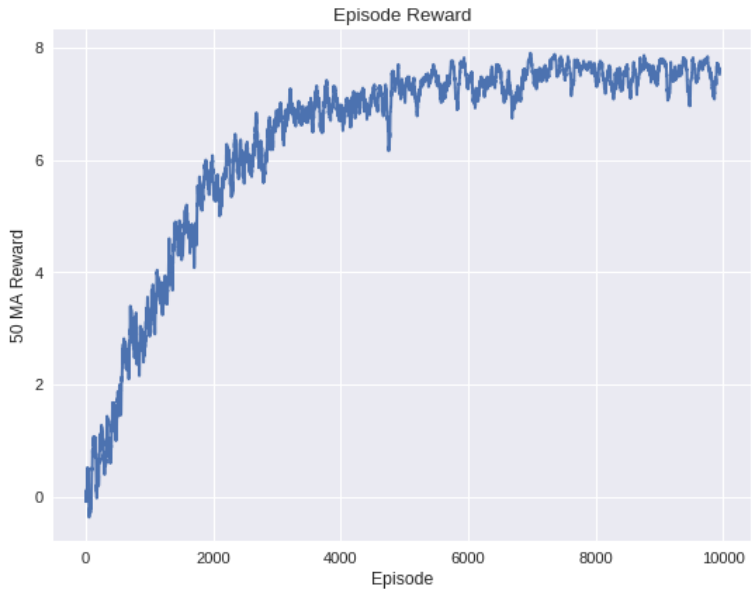
2.3 Varying lambda

Observation No. 1:

Lambda: 0.00005

Episode: 9900
Time Elapsed: 1131.18s
Epsilon 0.0602708725829849
Last Episode Reward: 6
Episode Reward Rolling Mean: 6.307621671258035



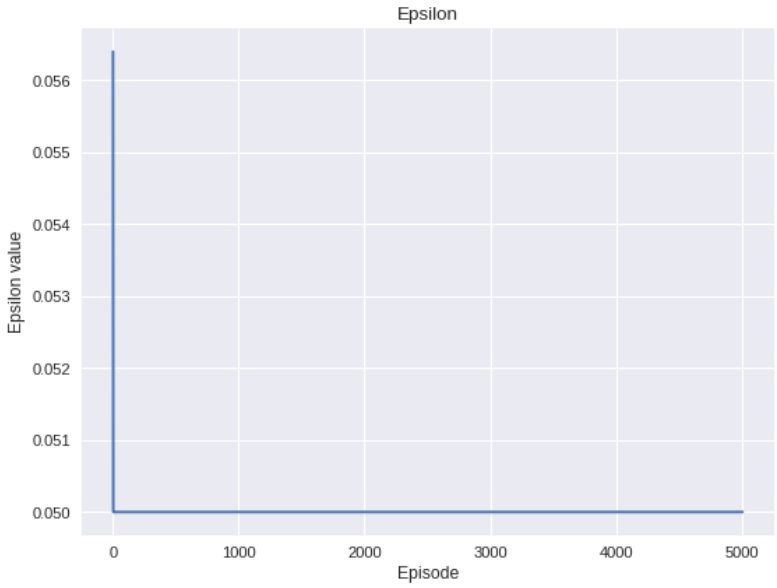


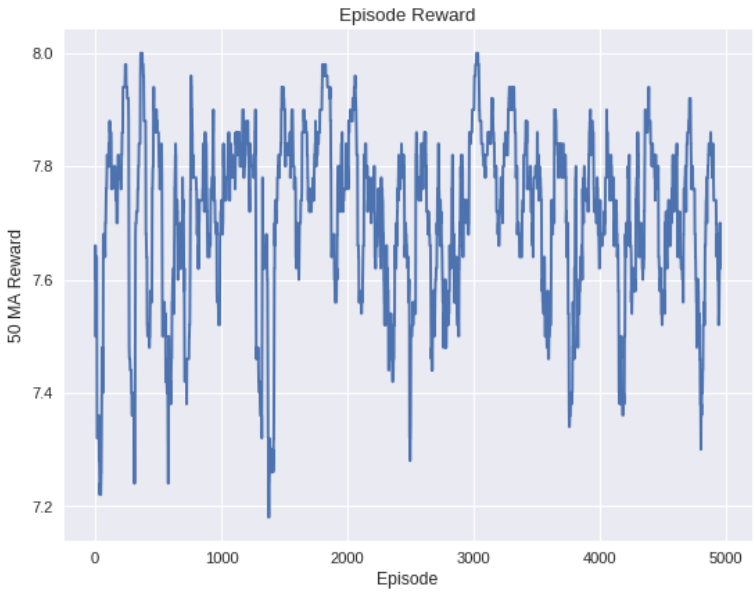
Observation No. 2:

Lambda: 0.5

Episode: 5000
Time Elapsed: 471.49s
Epsilon 0.05
Last Episode Reward: 8
Episode Reward Rolling Mean: 7.710671291573148

Note: Value of epsilon remained constant at 0.05 right from the 100 episode count. Also, the reward remained constant ~8 and rolling mean of 7.71 was constant from 2000 episodes count.

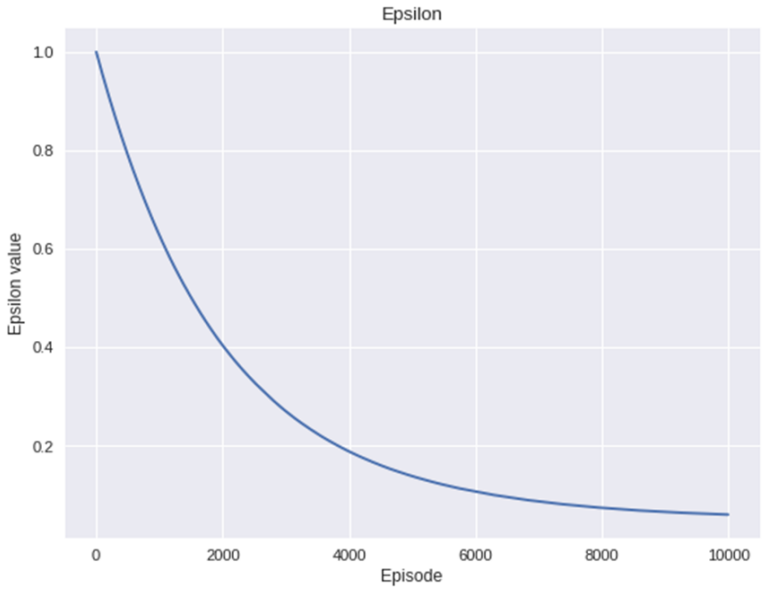


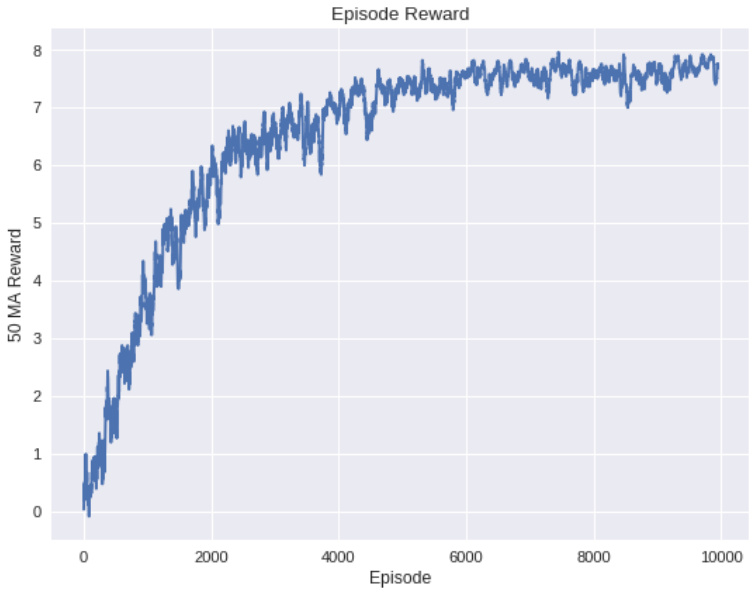

Episode Reward
8.0
7.8
7.6
7.4
7.2
50 MA Reward
0
1000
2000
3000
4000
5000
Episode

3 Experimenting Different Approaches

3.1 Implementing 4 – Layer Neural Network

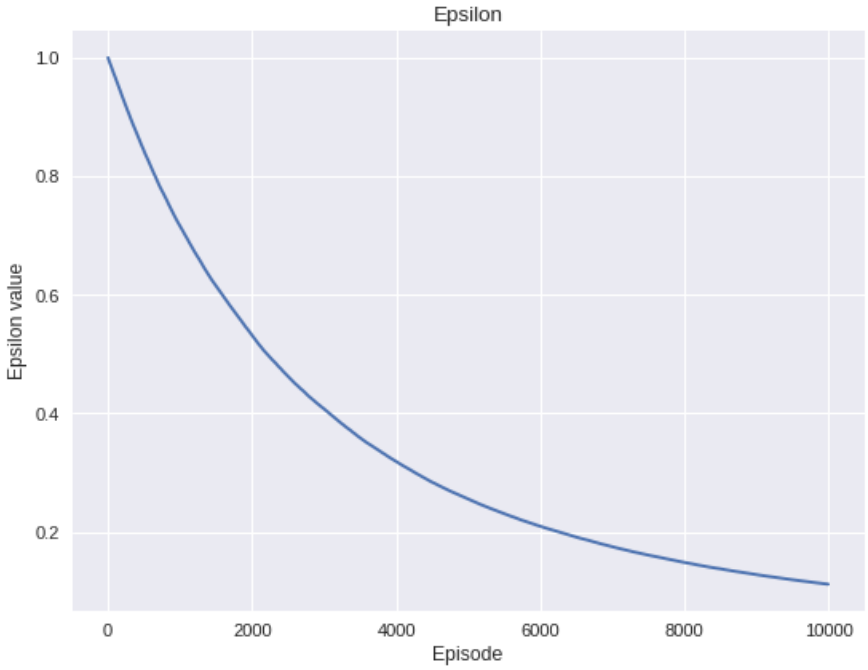
Observation No. 1:
Defaut configuration except 1 more hidden layer added with number of nodes = 128 and activation = sigmoid
Episode: 9900
Time Elapsed: 1173.14s
Epsilon 0.06052403811185667
Last Episode Reward: 8
Episode Reward Rolling Mean: 6.412815018875625

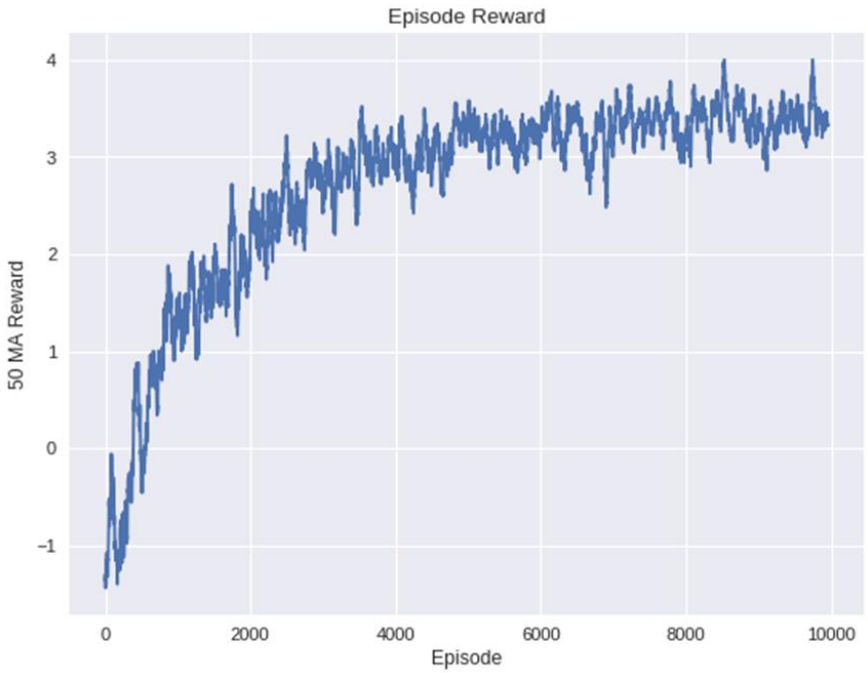




3.2 Random Orientation of Agent and the Goal

Episode: 9900
Time Elapsed: 487.90s
Epsilon 0.11371448842988056
Last Episode Reward: 3
Episode Reward Rolling Mean: 2.66727884909703





Inference: A small tweak in the code results in the random initial positions of Tom and Jerry. By setting Deterministic=False we can actually have random positions of our agent and goal. We then can follow computational steps so as to train our agent to reach the goal. From the above graphs, we can observe that agent does learn an optimal path even when the goal state changes. The epsilon decreases over time and the rewards do increase. By looking at the animation, it is evident that for smaller environments such as what we are working with, it is fairly easy to compute the optimal paths even when there are random positions of the agent and the goal.

4 Learning Process

How quickly the agent is able to learn?

- For illustration, let us consider the observations where epsilon was 0 and when epsilon was 1.
- When epsilon was forced to be 0, the agent just followed a greedy approach where only those actions were chosen that gave maximum reward.
- Our environment is limited and small, also it has no complexities involved that would obstruct the agent's path towards goal, thus the agent followed a specific path throughout the run and reached the goal.
- In our problem, the path followed by the agent turned out to be the optimal path.
- Thus, in just 100 episode count, the agent reached a maximum reward of 8.
- The reward remained constant as you can see the observations above.
- But, when epsilon was forced to 1, the agent performance degraded. More randomness was introduced by the epsilon and thus agent failed to follow an optimal path towards goal.
- For, optimum values of epsilon, agent gradually learns the optimal path, first my exploring and then by exploiting. Thus, balance is always maintained between exploration and exploitation which is very much essential.

5 Writing Tasks

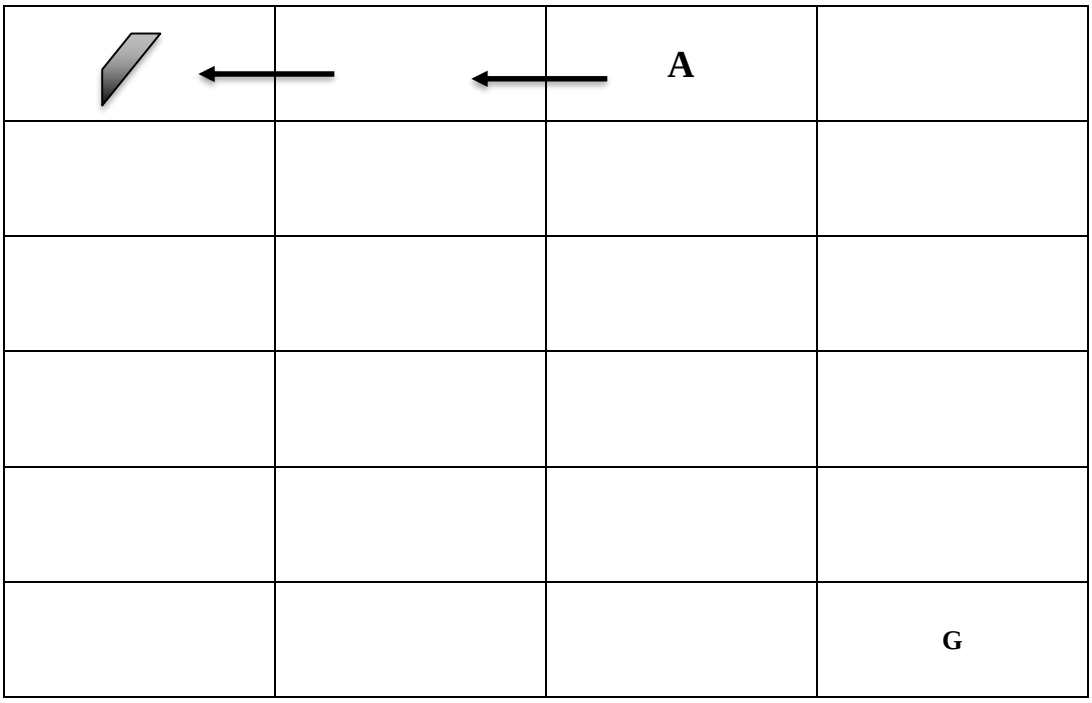
Task 1:

Question. Explain what happens in reinforcement learning if the agent always chooses the action that maximizes the Q-value. Suggest two ways to force the agent to explore.

Solution:

Q – values represent the optimal values when taking the best sequence of actions. This sequence of action is also called "policy". It is expected that the agent should choose the action that maximizes the Q – value. In the early stages of execution, when little is known about the world, it is important to explore and try unknown actions. There is a great deal to be learned and the agent does not have enough information to act well in any case. Later on, the agent may want to almost always choose the actions that is optimal, at that instant, there is very little knowledge left to attain.

What will happen if agent always chooses the action that maximizes the Q – value and where it would fail:



Let us consider above environment. Here, the agent acts greedy and always move towards left as it finds that action (moving left) maximize the Q – value. After moving couple of steps towards left, the agent will be blocked by an obstacle where it would get the negative reward. Thus, always choosing an action that would maximize the Q - value is not the optimum way. This strategy runs the risk of following those actions that are found during early training who have high Q – values. Exploration which is needed at the early stages is not performed and thus agent suffers.

Suggest two ways to force the agent to explore.

One way is to force the agent to **pick random actions** at the early stages so that the agent explores its surrounding.

Another way can be forcing the agent to choose actions with **minimum Q –values**, only at the start of the training so that again the agent can explore better.
Comprehensive approach to deal with this problem is to **use Probabilistic Approach of selecting** actions.
In Probabilistic Approach, actions with higher Q values are assigned higher probabilities. They are assigned as,

$$\mathrm{P}(a_i \mid S) = \frac{k^{Q(S_i, a_i)}}{\sum k^{Q(S_j, a_j)}}$$

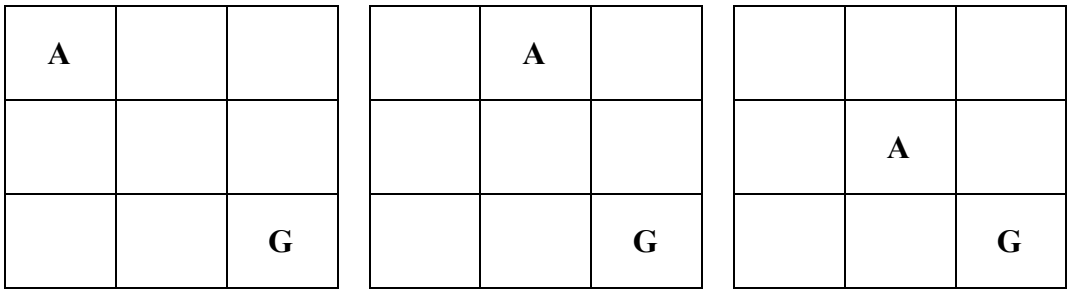
Where P(a | S) is the probability of selecting action a, given a state S and k>0 determines how strongly the selection favors action with high Q values.

Larger values of k will assign higher probabilities to actions with above average Q values, causing the agent to exploit what it has learned and will seek those actions who will maximize its reward.

In contrast, small values of k will allow higher probabilities for other actions, leading the agent to explore more.

Also, in some cases, k is varied with the number of iterations so that the agent favors exploration during early stages of learning, then gradually shifts towards a strategy of exploitation.

Task 2:



To find: Q – Values for the above mentioned state

Solution:

We will update the Q – Value table in every iteration. Let us consider calculating the Q – Values moving upwards i.e. computing Q – Values for S4 first then S3 and so on.

We will update the Q – Values as per the following formula: Q(S, a) = r + $\gamma * \max(Q(S,a))$
Also, the discount factor γ = 0.99

Starting from state S4, we can update the Q – Values for all actions as 0. Because, there is very little information that is needed to be gained and so the value of learning is negligible.

States	Actions			
	UP	DOWN	LEFT	RIGHT
S0				
S1				
S2				
S3				
S4	0	0	0	0

Now, Consider state S3,

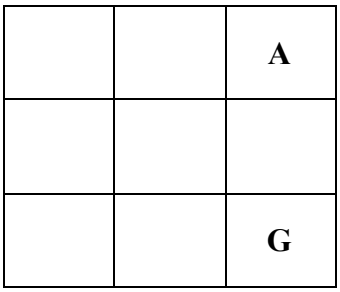
Q(S3, UP) = - 1 + (0.99 * max(Q(S5))

Q(S3, DOWN) = 1

Q(S3, LEFT) = - 1 + (0.99 * max(Q(S2))

Q(S3, RIGHT) = 0 + (0.99 * max(Q(S3))

In the above calculation, Q(S3, UP) discovered a new state, we renamed it as S5. It can be shown as:



S5

Now, Consider S2,

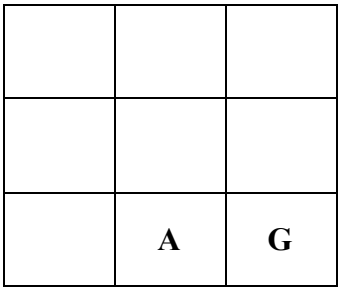
Q(S2, UP) = - 1 + (0.99 * max(Q(S1))

Q(S2, DOWN) = + 1 + (0.99 * max(Q(S6))

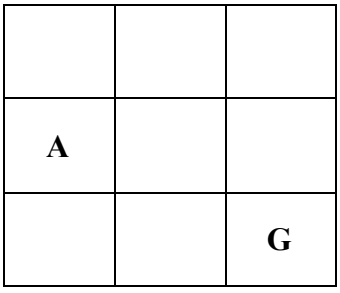
Q(S2, LEFT) = - 1 + (0.99 * max(Q(S7))

Q(S2, RIGHT) = + 1 + (0.99 * max(Q(S3))
Q(S2, RIGHT) = 1.99

In the above calculation, Q(S2, DOWN) and Q(S2, LEFT) discovered new states S6 and S7 respectively. They can be shown as:



S6



S7

Now, Consider state S1,

Q(S1, UP) = 0 + (0.99 * max(Q(S1))

Q(S1, DOWN) = + 1 + (0.99 * max(Q(S2))
Q(S1, DOWN) = + 1 + (0.99 * 1.99)
Q(S1, DOWN) = 2.97

Q(S1, LEFT) = - 1 + (0.99 * max(Q(S0))

Q(S1, RIGHT) = + 1 + (0.99 * max(Q(S5))

Now, Consider state S0,

Q(S0, UP) = 0 + (0.99 * max(Q(S0))

Q(S0, DOWN) = + 1 + (0.99 * max(Q(S7))

Q(S0, LEFT) = 0 + (0.99 * max(Q(S0))
Q(S0, LEFT) = 3.9

Q(S0, RIGHT) = + 1 + (0.99 * max(Q(S1))
Q(S0, RIGHT) = +1 + (0.99 * 2.97)
Q(S0, RIGHT) = 3.94

Now, since Q(S0, RIGHT) = 3.94, we can calculate,

Q(S0, UP) = 0 + (0.99 * max(Q(S0))
Q(S0, UP) = 0 + (0.99 * 3.94)
Q(S0, UP) = 3.9

Also,

Q(S1, LEFT) = - 1 + (0.99 * max(Q(S0))
Q(S1, LEFT) = - 1 + (0.99 * 3.94)
Q(S1, LEFT) = 2.9

We now have the max value in state S1. Thus,

Q(S1, UP) = 0 + (0.99 * max(Q(S1))
Q(S1, UP) = 0 + (0.99 * 2.97)
Q(S1, UP) = 2.94

Also,

Q(S2, UP) = - 1 + (0.99 * max(Q(S1))
Q(S2, UP) = - 1 + (0.99 * 2.97)
Q(S2, UP) = 1.94

We now have the max value in state S2. Thus,

Q(S3, LEFT) = - 1 + (0.99 * max(Q(S2))
Q(S3, LEFT) = - 1 + (0.99 * 1.99)
Q(S3, LEFT) = 0.97

We now have the max value in state S3. Thus,

Q(S3, RIGHT) = 0 + (0.99 * max(Q(S3))
Q(S3, RIGHT) = 0 + (0.99 * 1)
Q(S3, RIGHT) = 0.99

Till now, our Q – table looks like;

States	Actions			
	UP	DOWN	LEFT	RIGHT

S0	3.90	?	3.90	3.94
S1	2.94	2.97	2.90	?
S2	1.94	?	?	1.99
S3	?	1	0.97	0.99
S4	0	0	0	0
S5	?	?	?	?
S6	?	?	?	?
S7	?	?	?	?

Now, Consider state S5,

Q(S5, UP) = 0 + (0.99 * max(Q(S5))

Q(S5, DOWN) = + 1 + (0.99 * max(Q(S3))
Q(S5, DOWN) = + 1 + (0.99 * 1)
Q(S5, DOWN) = 1.99

Q(S5, LEFT) = - 1 + (0.99 * max(Q(S1))
Q(S5, LEFT) = - 1 + (0.99 * 2.97)
Q(S5, LEFT) = 1.94

Q(S5, RIGHT) = 0 + (0.99 * max(Q(S5))
Q(S5, RIGHT) = 0 + (0.99 * 1.99)
Q(S5, RIGHT) = 1.97

Also,

Q(S5, UP) = 0 + (0.99 * max(Q(S5))
Q(S5, UP) = 1.97

Now, Consider state S6,

Q(S6, UP) = - 1 + (0.99 * max(Q(S2))
Q(S6, UP) = - 1 + (0.99 * 1.99)
Q(S6, UP) = 0.97

Q(S6, DOWN) = 0 + (0.99 * max(Q(S6))

Q(S6, LEFT) = - 1 + (0.99 * max(Q(S8))

Q(S6, RIGHT) = 1

Thus,

Q(S6, DOWN) = 0 + (0.99 * max(Q(S6))
Q(S6, DOWN) = 0 + (0.99 * 1)
Q(S6, DOWN) = 0.99
Now, Consider state S7,

Q(S7, UP) = - 1 + (0.99 * max(Q(S0))
Q(S7, UP) = - 1 + (0.99 * 3.94)
Q(S7, UP) = 2.9

Q(S7, DOWN) = + 1 + (0.99 * max(Q(S8))

Q(S7, LEFT) = 0 + (0.99 * max(Q(S7))

Q(S7, RIGHT) = + 1 + (0.99 * max(Q(S2))
Q(S7, RIGHT) = + 1 + (0.99 * 1.99)
Q(S7, RIGHT) = 2.97

Thus,

Q(S7, LEFT) = 0 + (0.99 * max(Q(S7))
Q(S7, LEFT) = 0 + (0.99 * 2.97)
Q(S7, LEFT) = 2.94

In the above calculation, a new state was discovered, say S8,

A		**G**

S8

Q(S8, UP) = - 1 + (0.99 * max(Q(S7))
Q(S8, UP) = - 1 + (0.99 * 2.97)
Q(S8, UP) = 1.94

Q(S8, DOWN) = 0 + (0.99 * max(Q(S8))

Q(S8, LEFT) = 0 + (0.99 * max(Q(S8))

Q(S8, RIGHT) = + 1 + (0.99 * max(Q(S6))
Q(S8, RIGHT) = 0 + (0.99 * 1)
Q(S8, RIGHT) = 0.99

Thus,

Q(S0, DOWN) = + 1 + (0.99 * max(Q(S7))
Q(S0, DOWN) = 3.94

Q(S1, RIGHT) = + 1 + (0.99 * max(Q(S5))
Q(S1, RIGHT) = 2.97

Q(S2, DOWN) = + 1 + (0.99 * max(Q(S6))
Q(S2, DOWN) = 1.99

Q(S2, LEFT) = - 1 + (0.99 * max(Q(S7))
Q(S2, LEFT) = 1.94

Q(S3, UP) = - 1 + (0.99 * max(Q(S5))
Q(S3, UP) = 0.97

Thus, the final Q-Table looks like,

States	Actions			
	UP	DOWN	LEFT	RIGHT
S0	3.90	3.94	3.90	3.94
S1	2.94	2.97	2.90	2.97
S2	1.94	1.99	1.94	1.99
S3	0.97	1	0.97	0.99
S4	0	0	0	0

6 References

Teaching Assistants – CSE 574 – Introduction to Machine Learning

https://www.oreilly.com/ideas/reinforcement-learning-explained