Tom & Jerry in Reinforcement Learning

Ekta Katiyar

Person Number: ██████████

December 05, 2018

Abstract

In this experiment, our task is to teach the agent to navigate in the grid world environment using reinforcement learning. This experiment also helped in understanding various aspects of hyperparameters used in the reinforcement learning model. These are the parameters whose value is set before the learning process begins and tuning hyperparameters impacts the Learning process of the models. If hyperparameters are poorly chosen, then the agent will learn slowly or might not learn at all and end up affecting the accuracy of the model.

Introduction

Our task is to train Tom(Agent) to navigate in the grid world environment in order to find the shortest path to Jerry(Goal), given that the initial positions of both the agent and goal id deterministic using deep reinforcement learning algorithm – Deep Q-Network.

Problem Definitions:

1. First, we have to create a 3-layer neural network using Keras library.
2. Second, we have to implement exponential-decay formula for epsilon.
3. Third is to implement Q-function.
4. Fourth, code snippets of the implementation and description

Experiments

Below is the list of evaluations done on the implemented models:

1. Evaluate the performance of the brain implemented using neural network having 2 hidden layers on the basis of the next action taken by the agent considering the current state.
2. Evaluate the action on the basis of the highest Q value returned by the state chosen by the agent.
3. Performance of the epsilon curve and thus exponential decay to determine how quick our agent is learning by exploring the environment.

4. Evaluation of mean reward our agent has earned during the total number of episodes.

Explanations

1. **Reinforcement learning:** Reinforcement learning is an approach in machine learning in which an agent must interact with the environment in order to maximize the reward. The agent is not told, instead must discover which action yield the most reward by trying them. The overall idea behind reinforcement learning is that the agent will learn from the environment by interacting with it and receiving rewards for performing actions.
   It follows following steps:
   - Our agent receives S1 from the environment.
   - Based on the state S1, agent takes action A1.
   - Environment transitions to a new state S2.
   - Environment gives some reward R1 to the agent.

   That is the reason in Reinforcement Learning, to achieve the best behavior, we need to maximize the expected cumulative reward.
   Cumulative reward at each time step 't' can be described as:

$$G_t = R_{t+1} + R_{t+2} + \cdots$$

2. **Exploration/Exploitation tradeoff:** Since we aim to maximize the expected cumulative reward at each step in the reinforcement learning, there will be fairly good chances that our agent acts greedy. By greedy, it means that that it will always choose the step that has maximum rewards and therefore, fall into a trap because there can be case when there is relatively low reward at the first step but very high reward at the second step.
   But since our agent is following the greedy approach it won't end up choosing that state due to low rewards at the first step.
   Therefore, if our agent does **exploration**, we can gather more information about our environment that might lead us to better decisions in the future and then **exploit** the knowledge that it has found for the current state '**s**' by taking an action '**a**' in such a way that it maximizes Q[s,a].

3. **Epsilon:** One simple strategy for exploration for the agent is to take the best-known action for most of the time, but occasionally explore a new one, randomly selected direction even though it might be walking away from the known reward.
   This strategy is known as epsilon greedy strategy, where epsilon is the percentage of the time that agent took a randomly selected action instead of an action with maximum reward. We may obtain higher value of epsilon by exploring the environment extensively.
   Over the time, when our agent learns more about the environment and which action will generate the most long-term reward, it will steadily reduce the epsilon to very low as it settles into exploiting what it knows.
   Exponential decay formula for epsilon is: $\epsilon = \epsilon_{min} + (\epsilon_{max} - \epsilon_{min}) * e^{-\lambda |S|}$

   where $\epsilon_{min}, \epsilon_{max} \in [0,1]$

λ – hyperparameter for epsilon

$|S|$ – total number of steps
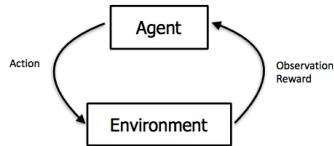
4. **Markov Decision Process:** Markov Decision Process is an approach based on Markov property which states that "future is independent of the past given the present."
The effects of an action taken in a state depends only on that state and not on a prior history.
A MDP model contains:
- A set of possible environment states S.
- A set of possible actions A.
- A real valued reward function for each state and action R(s,a)
- A description T of each action's effects in each state.
- A policy π is a mapping from S to A.
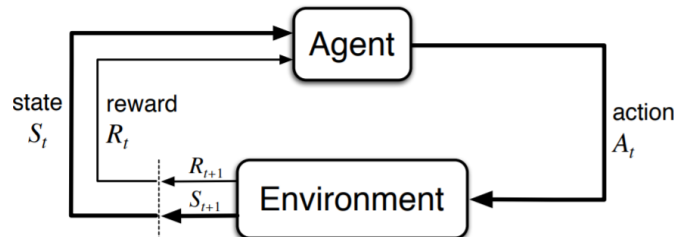
Therefore, the transition probability function is defined by

$$P[St+1 \mid St] = P[St+1 \mid S1, \,..... \,, St]$$

Above equation is known as **Bellman equation**. According to Bellman equation, the utility of a certain state is equal to the reward in the current state plus the discount from all the rewards, agent get from that point on, transitioning from the current state to a future state. We start with arbitrary utilities and then update them based on the neighbors and we repeat it until it converges.

5. **Discount Factor:** Discount factor γ determines how important future reward is. Value for gamma lies between 0 to 1.
Larger the value of gamma, smaller will be the discount which represent that agent is more concerned about the long-term rewards.
Smaller value of gamma means, greater value of discount which signifies that our agent is short sighted and only care about current rewards thus, greedy.

Discounted cumulative rewards is:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

where $\gamma \in [0,1)$

6. **Value Function:** Value function V(s) is used in the value based approach of reinforcement learning. Value function tells us the maximum expected future reward the agent will get at each state.
The value of each state is the total amount of the reward an agent can expect to accumulate over the future, starting at that state.

$$v_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \mid S_t = s \right]$$

Expected   Reward discounted   Given that state

Agent will select the state with biggest value at each step using value function.
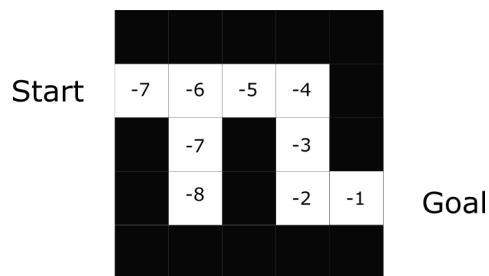
In the above example, agent will take the biggest value, -7, -6, -5, … till smallest value -1 to attain the goal.

7. **Q-function:** Since agent can't control what state is ends up in. Q-function Q*(s, a) accepts state and action as a parameter and returns the expected total reward. Therefore, it tells our agent how good of a choice is picking 'a' when state is 's'.
Relation between value function and Q function is given by:

$$V^*(s) = \max_a Q^*(s, a) \quad \forall s \in \mathbb{S}$$

Using Q*(s, a), optimal policy $\pi *$ can be determined by choosing the action as that gives maximum reward for Q*(s, a) for state 's':

$$\pi^*(s) = argmax_a Q(s, a)$$

8. **Deep Q-learning:** Deep reinforcement learning introduces deep neural network to solve the reinforcement learning and this for approximating the reward based on state q value.

Deep Q-learning can be represented as:
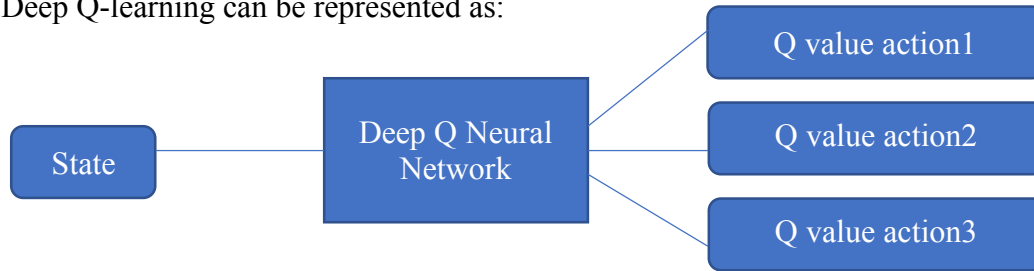
We want our agent to decrease the number of random action, as it goes we will observe an exponential decay epsilon, that eventually will allow our agent to explore the environment.

CODE IMPLEMENTATION

```
### START CODE HERE ### (≈ 3 lines of code)

model.add(Dense(128, input_dim=self.state_dim, activation='relu'))
model.add(Dense(128,  activation='relu'))

model.add(Dense(self.action_dim, activation='linear'))

### END CODE HERE ###
```

In the above snippet, I have implemented the 3-layer neural network using Keras library. We have trained the neural network to predict Q-values for each action in the given state. To develop it, the input needs to be the current state dimension, number of hidden nodes and activation function as 'relu'.
In case of linear activation function, we need to provide the dimension of action space and 'linear' activation function.

```
### START CODE HERE ### (≈ 1 line of code)
self.epsilon = self.min_epsilon + (self.max_epsilon - self.min_epsilon) * math.exp(-self.lamb * self.steps)

### END CODE HERE ###
```

Above snippet contains the implementation of exponential decay formula of epsilon, which is $\epsilon = \epsilon_{min} + (\epsilon_{max} - \epsilon_{min}) * e_{-\lambda|S|}$ to allow our agent to explore the environment.

```
### START CODE HERE ### (≈ 4 line of code)

if st_next is None:
  t[act]=rew
else:
  t[act]=rew+ self.gamma*np.amax(q_vals_next[i])

### END CODE HERE ###
```
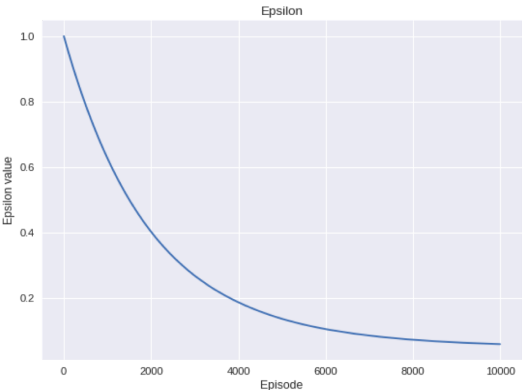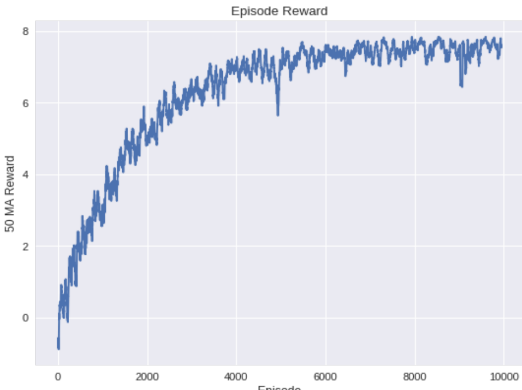
In the above implementation, I have implementation the condition for the Q-function which is

$$Q_t = \begin{cases} r_t, & \textit{if episode terminates at step } t+1 \\ r_t + \gamma max_a Q(s_t, a_t; \theta), & \textit{Otherwise} \end{cases}$$

Following the above implementations, Tom is taking 803.75 seconds to catch the Jerry mouse. This can be further improved by tuning the hyperparameters.

Hyperparameters Tuning

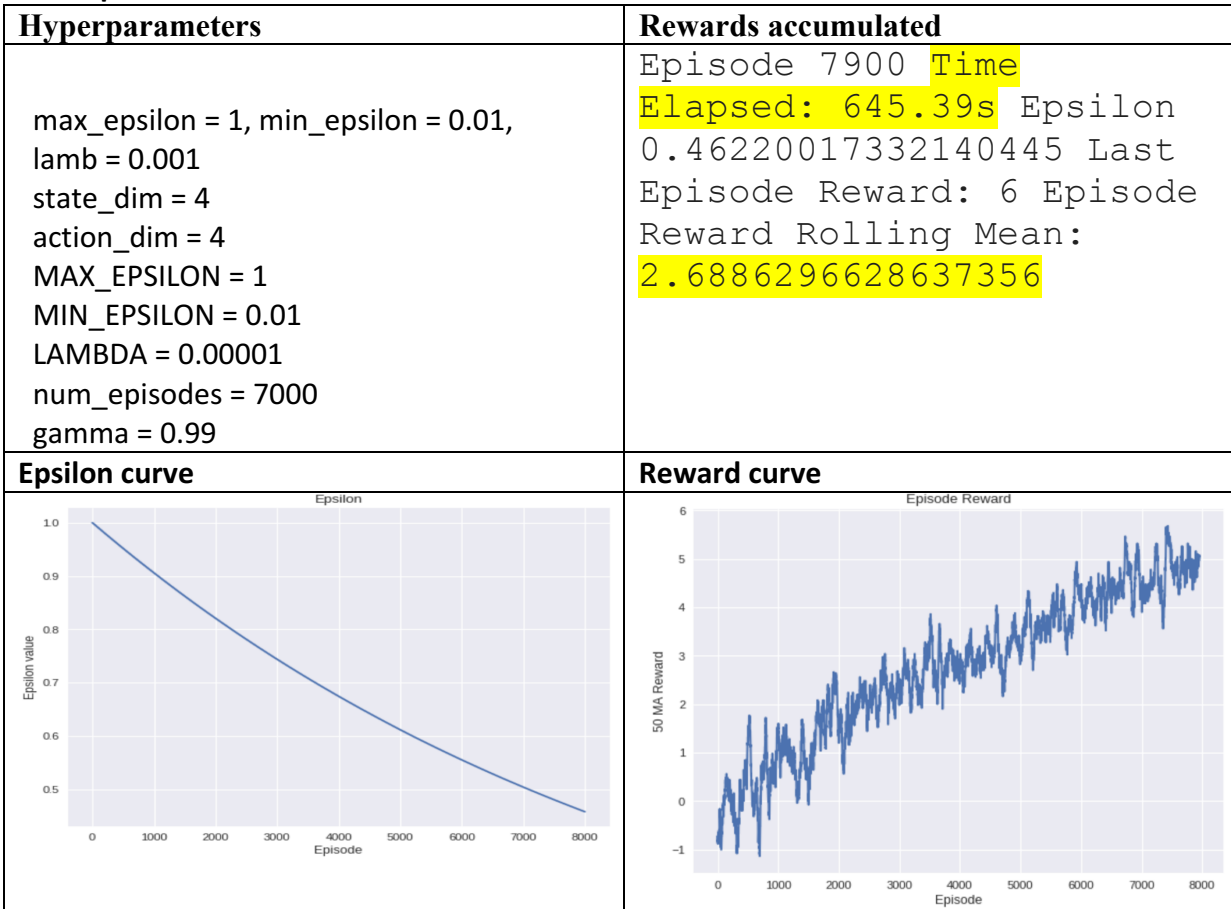1. **Hyperparameters provided with the project description.**

Hyperparameters	Rewards accumulated
max_epsilon = 1, min_epsilon = 0.01, lamb = 0.001 state_dim = 4 action_dim = 4 MAX_EPSILON = 1 MIN_EPSILON = 0.05 LAMBDA = 0.00005 num_episodes = 10000 gamma = 0.99	Episode 9900 Time Elapsed: 803.75s Epsilon 0.06021555935117375 Last Episode Reward: 8 Episode Reward Rolling Mean: 6.21181512090603
Epsilon curve	**Reward curve**

2. **Number of episodes has been decreased to 7000**
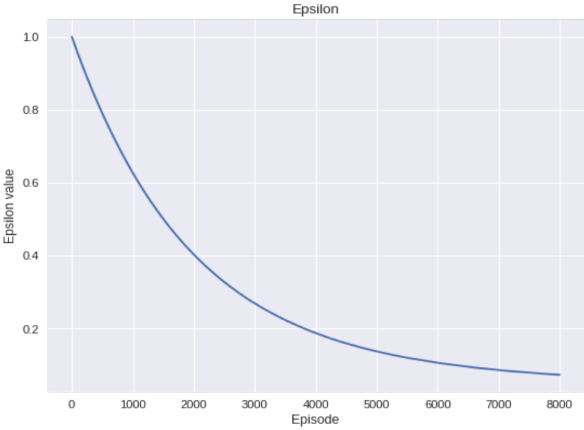
Hyperparameters	Rewards accumulated
max_epsilon = 1, min_epsilon = 0.01, lamb = 0.001 state_dim = 4 action_dim = 4 MAX_EPSILON = 1	Episode 7900 Time Elapsed: 637.07s Epsilon 0.07448013530012118 Last Episode Reward: 8 Episode Reward Rolling Mean: 5.91719010383842

MIN_EPSILON = 0.05 LAMBDA = 0.00005 num_episodes = 7000 gamma = 0.99	
Epsilon curve	**Reward curve**
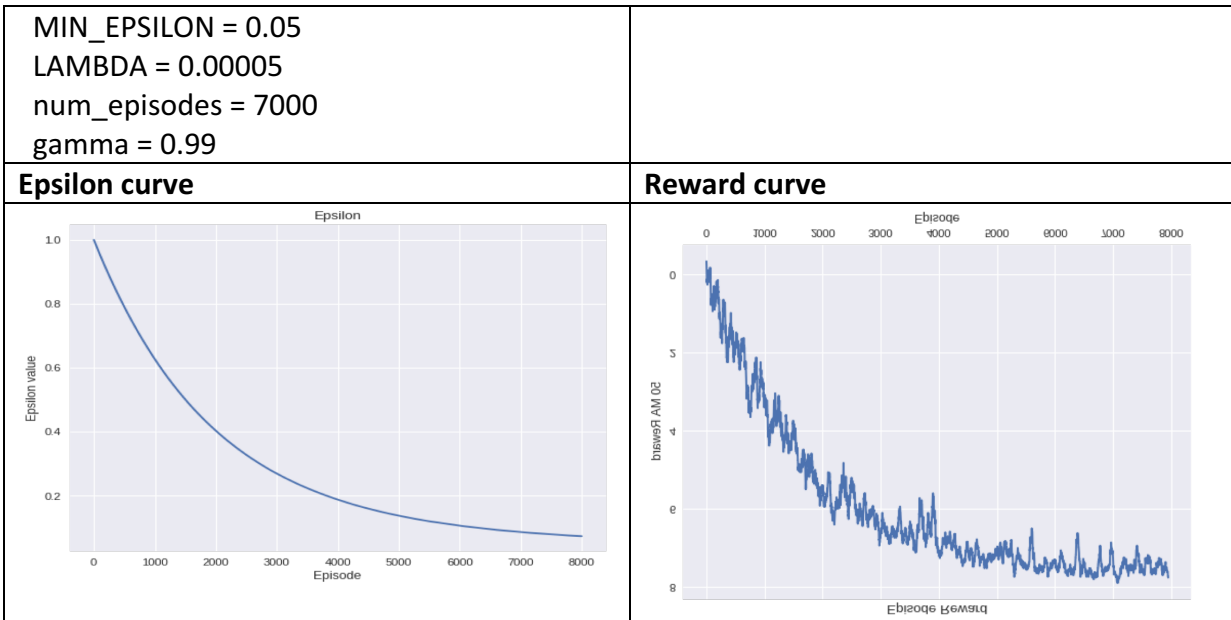
3. **Min epsilon is reduced to 0.01, lambda is reduced to 0.00001 along with number of episodes=7000**
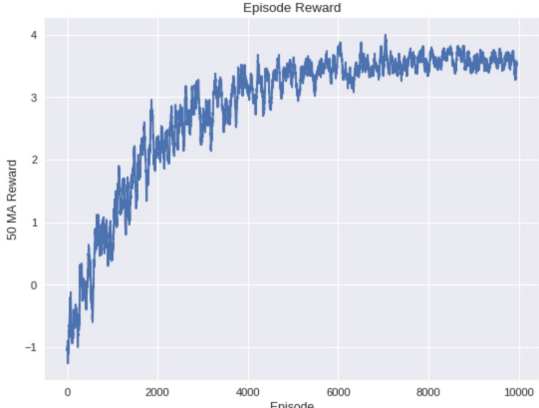
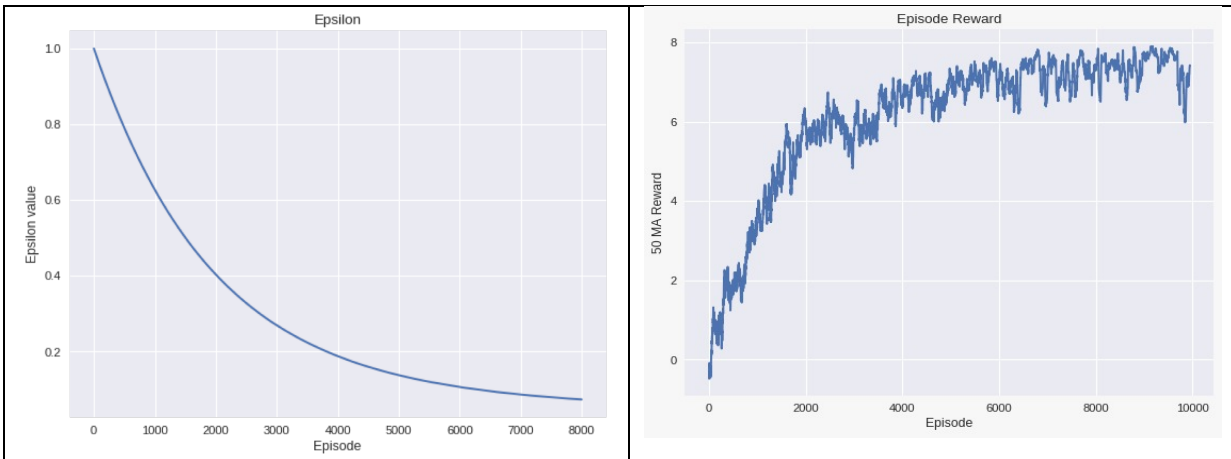| Hyperparameters	Rewards accumulated
max_epsilon = 1, min_epsilon = 0.01, lamb = 0.001 state_dim = 4 action_dim = 4 MAX_EPSILON = 1 MIN_EPSILON = 0.01 LAMBDA = 0.00001 num_episodes = 7000 gamma = 0.99	Episode 7900 Time Elapsed: 645.39s Epsilon 0.4622001733214045 Last Episode Reward: 6 Episode Reward Rolling Mean: 2.6886296628637356
Epsilon curve	**Reward curve**

4. **When discounting factor is reduced to 0, means the agent will follow the greedy approach while not learning about the environment through exploration.**

Hyperparameters	Rewards accumulated

max_epsilon = 1, min_epsilon = 0.01,
lamb = 0.001
state_dim = 4
action_dim = 4
MAX_EPSILON = 1
MIN_EPSILON = 0.05
LAMBDA = 0.00005
num_episodes = 10000
gamma = 0
	`Episode 9900`
	`Time Elapsed: 441.50s`
	`Epsilon`
	`0.12115865663975126`
	`Last Episode Reward: 4`
	`Episode Reward Rolling`
	`Mean: 2.7927762473217017`

Epsilon curve	**Reward curve**

5. **When discounting factor is reduced to 1, means the agent will learn through exploration/exploitation.**

Hyperparameters	Rewards accumulated

max_epsilon = 1., min_epsilon = 0.01,
lamb = 0.001
state_dim = 4
action_dim = 4
MAX_EPSILON = 1
MIN_EPSILON = 0.05
LAMBDA = 0.00005
num_episodes = 10000
gamma = 1
	`Episode 9900`
	`Time Elapsed: 795.55s`
	`Epsilon`
	`0.12115865663975126`
	`Last Episode Reward: 8`
	`Episode Reward Rolling`
	`Mean: 6.112776247321017`

Epsilon curve	**Reward curve**

Conclusion

Tom has successfully caught Jerry by learning through the environment by trading off the exploration and exploitation property by obtaining the maximum reward using reinforcement learning.

References

[1] Project 3 description posted on UB learns

[2] https://towardsdatascience.com/reinforcement-learning-demystified-markov-decision-processes-part-1-bf00dda41690

[3] https://medium.freecodecamp.org/an-introduction-to-reinforcement-learning-4339519/

[4] http://www.aispace.org/exercises/solutions/11a/4.shtml

[5] https://hackernoon.com/reinforcement-learning-part-3-38ca74a0bde4s

WRITTING TASKS

Task-1:
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:

If our agent always chooses the action that maximizes the Q-value, it will only be able to perform the exploitation but not the exploration thus it'll get stuck in non-optimal policy because it is not exploring the environment and simply picking the best action from each state.

In our case, if we change the discounting factor $\gamma(gamma) = 0$, agent will not explore the environment and start behaving short-sighted and will only seek for the current rewards while losing greater future rewards.

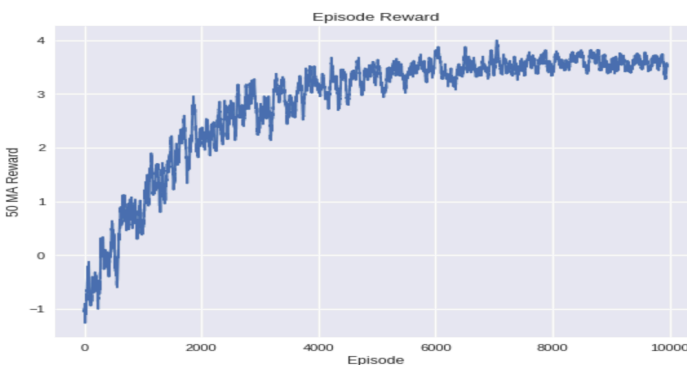For our gaming model, if we have done $\gamma(gamma) = 0$,

Total rewards earned by the agent are:

```
Episode 9900
Time Elapsed: 441.50s
Epsilon 0.12115865663975126
Last Episode Reward: 4
Episode Reward Rolling Mean: 2.7927762473217017
```

With epsilon curve, as:
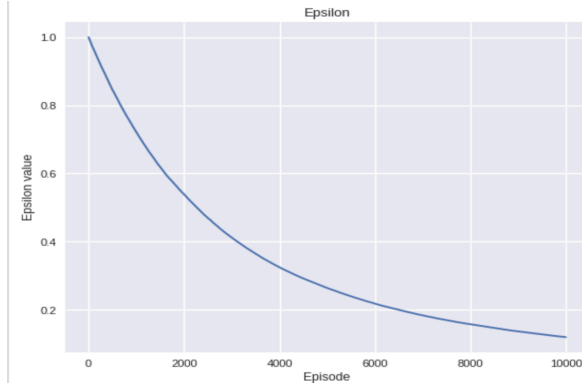
and Reward curve as follows:

Two ways to force the agent to explore:
1. One way to force the agent to explore the environment is by initializing the Q-values to the high values, that will encourage the agent to explore through the environment and seek for greater future rewards instead of best reward from the current state.
2. Second approach is 'ε-greedy strategy' for trading off exploration and exploitation where we can take decision at each step while learning to reach the goal and then opt to either take the random action or the agent's trusted action.

The probability of taking random action is known as Epsilon in this algorithm an it exploits the best with the probability of 1-Epsilon, whereas explores the best with the probability of Epsilon/n.

Task-2:
Calculate Q- value for the given states and provide the calculation steps.
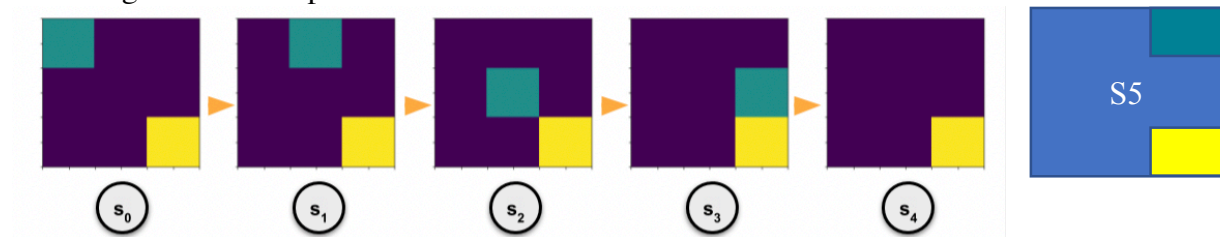
Solution:
Since, the agent is set to be in the upper-left corner and goal is in lower right corner. The agent receives a reward of:
  • When it moves closer to the goal: 1
  • When it moves away from the goal: -1
  • When it does not move at all: 0
  • $\gamma = 0.99$

Formula for calculating the Q function is given by-
$$Q(s_t, a_t) = r_t + \gamma * max_a Q(s_{t+1}, a)\text{------------------ equation 1}$$

And the grid states are provided as-

At state S4, our agent has reached to the goal and there will not be any further movement:
Q(S4, Up)=0
Q(S4, Down)=0
Q(S4, Left)=0
Q(S4, Right)=0

	Up	Down	Left	Right
S0				
S1				
S2				
S3				
S4	0	0	0	0

At state S3:
Q(S3, Down) = 1 (Because by taking 'Down' action, S3 will reach the goal)
Q(S3, Right) = By using equation 1
$$= r_t + \gamma * max_a Q(s_{t+1}, a) = 0+0.99*(1) \text{ [Since maximum value will be 1]}$$

$= 0.99$

$Q(S3, Left) \quad =$ By using equation 1

$= r_t + \gamma * max_a Q(S2, a) = (-1)+0.99*(1+0.99(1))$

$= 0.9701$

$Q(S3, Up) \quad =$ By using equation 1

$= r_t + \gamma * max_a Q(s_5, a)$ [In this case since we have not provided with state 5 we have considered a S5 to evaluate the value of Q(S3, Up)]

$=(-1)+0.99(1+0.99*max(Q(S3,a)))$

$= (-1)+0.99(1+0.99)$

$= 0.9701$

	Up	Down	Left	Right
S0				
S1				
S2				
S3	0.9701	1		0.99
S4	0	0	0	0

At state S2:

$Q(S2, Right) =$ By using equation 1

$= r_t + \gamma * max_a Q(S2, a) = (1)+0.99*(1+0.99(1))$

$= 1.99$

$Q(S2, Down) =$ By using equation 1

$= r_t + \gamma * max_a Q(S2, a) = (1)+0.99*(1+0.99(1))$

$= 1.99$

$Q(S2, Left) =$ By using equation 1

$= r_t + \gamma * max_a Q(S6, a) = (-1)+0.99*(1+0.99(1))$

$=$ we will assume a state S6 in order to calculate Q(S2, Left)

$= (-1)+0.99*(max_a Q(S1, a))$
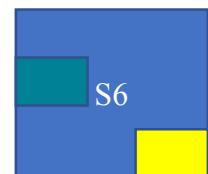
$= (-1)+0.99*(2.9701)$

$= 1.940399$

$Q(S2, Up) =$ By using equation 1

$= r_t + \gamma * max_a Q(S1, a) = (-1)+0.99*(2.9701)$

$= 1.940399$

	Up	Down	Left	Right
S0				

S1				
S2	1.940399	1.99	1.940399	1.99
S3	0.9701	1	0.9701	0.99
S4	0	0	0	0

At state S1:

Q(S1, Down) = By using equation 1
$= r_t + \gamma * max_a Q(S2, a)$
$= (1)+0.99*(1.99)$
$= 2.9701$

Q(S1, Right) = By using equation 1
$= r_t + \gamma * max_a Q(S5, a)$
$= (1)+0.99*(1+ 0.99)$
$= 2.9701$

Q(S1, Up) = By using equation 1
$= r_t + \gamma * max_a Q(S1, a)$
$= (0)+0.99*(2.9701)$
$= 2.940399$

	Up	Down	Left	Right
S0				
S1	2.940399	2.9701		2.9701
S2	1.940399	1.99	1.940399	1.99
S3	0.9701	1	0.9701	0.99
S4	0	0	0	0

At state S0:

Q(S0, Up) = By using equation 1
$= r_t + \gamma * max_a Q(S0, a)$
$= (0)+0.99*(3.940399)$
$= 3.400445$

Q(S1, Left) = By using equation 1
$= r_t + \gamma * max_a Q(S0, a)$
$= (-1)+0.99*(3.940399)$
$= 2.400445$

Q(S0, Left) = By using equation 1

$$= r_t + \gamma * max_aQ(S0, a)$$
$$= (0)+0.99*(2.9701)$$
$$= 3.400445$$

$Q(S0, Right) =$ By using equation 1
$$= r_t + \gamma * max_aQ(S1, a)$$
$$= (1)+0.99*(2.9701)$$
$$= 3.940399$$

$Q(S0, Down) =$ By using equation 1
$$= r_t + \gamma * max_aQ(S6, a)$$
$$= (0)+0.99*(2.9701)$$
$$= 3.940399$$

	Up	Down	Left	Right
S0	3.400445	3.940399	3.400445	3.940399
S1	2.940399	2.9701	2.400445	2.9701
S2	1.940399	1.99	1.940399	1.99
S3	0.9701	1	0.9701	0.99
S4	0	0	0	0