A Report on Teaching Agent Using Reinforcement and Deep Learning

Miki Padhiary
UBID: mikipadh Person Number: 50286289

December 6, 2018

Abstract

In this experiment, We studied how to apply reinforcement learning and deep learning to teach the agent to navigate in the grid-world environment. Our objective was to model a game using Tom and Jerry cartoon where Tom, a cat, is chasing Jerry, a mouse. The task for Tom (an agent) is to nd the shortest path to Jerry (a goal), given that the initial positions of Tom and Jerry. We applied ***Deep reinforcement learning algorithm - DQN (Deep Q-Network)****, to solve our task.*

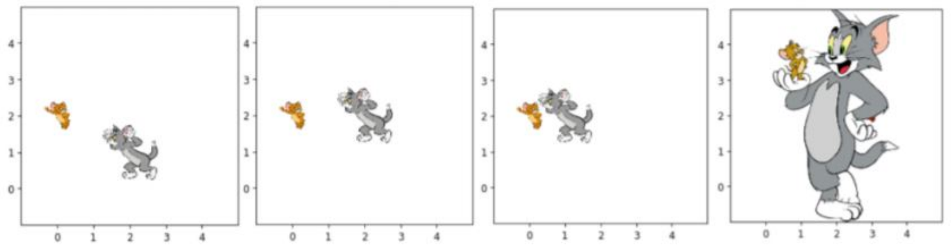
Keywords: Reinforcement Learning, Deep Learning, DQN, Machine Learning, Agent, Environment, Reward, State, Q-Learning, and Action

1 Introduction

In the project, Tom is the agent and Jerry is the goal. Our objective is to navigate to Jerry using the shortest path. The environment is designed as a grid-world 5x5. The agent has the flexibility to choose 100 steps to achieve as large a reward as possible. Tom and Jerry have the same position over each reset, thus the agent needs to learn a fixed optimal path. The program has been implemented under two circumstances.

- The agent have the same position over each reset and therefore the agent needs to learn a fixed optimal path
- At the start of each episode, agent and goal are randomly placed within a 5x5 grid-world.

A quick look of our goal is described in figure below:

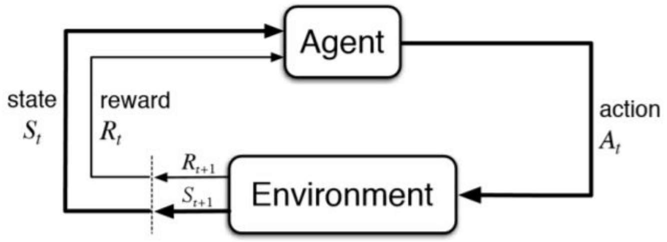


2 Reinforcement Learning

Reinforcement learning is a direction in Machine Learning where an agent learn how to behave in a environment by performing actions and seeing the results. In reinforcement learning, an agent learns from trial-and-error feedback rewards from its environment.The result is a map of states to actions to maximize the long-term total reward. Reinforcement learning has made significant progress recently. It solves the difficult problem of correlating immediate actions with the delayed returns they produce.

2.1 Markov Decision Process

Effectively in reinforcement learning, environments are functions that transform an action taken in the current state into the next state and a reward; agents are functions that transform the new state and reward into the next action.



Basic reinforcement is modeled as a Markov decision process which is a 5-tuple (S, A, Pa, Ra,), where

- S is a finite set of states
- A is a finite set of actions
- $\mathbf{Pa}(\mathbf{s}, \mathbf{s_0}) = \mathbf{Pr}(\mathbf{s_{t+1}} = \mathbf{s_0} | \mathbf{s_t} = \mathbf{s}, \mathbf{a_t} = \mathbf{a})$ is the probability that action a in state s at time t will lead to state $\mathbf{s_0}$ at $\mathbf{time + 1}$
- $\mathbf{Ra}(\mathbf{s}, \mathbf{s_0})$ is the immediate reward (or expected immediate reward) received after transitioning from state $\mathbf{s}$ to state $\mathbf{s_0}$ due to action $\mathbf{a}$
- $[\mathbf{0}, \mathbf{1})$ is the discount factor, which represents the difference in importance between future rewards and present rewards

2.2 Discounting Factor γ

The discounting factor $\gamma \in [\mathbf{0}, \mathbf{1}]$ penalize the rewards in the future.

3 Deep Q Learning

Experience replay will help us to handle three main things:

- Avoid forgetting previous experiences

- Reduce correlations between experiences
- Increases learning speed with mini-batches

The main idea behind the experience replay is that by storing an agent's experiences, and then randomly drawing batches of them to train the network, we can more robustly learn to perform well in the task. By keeping the experiences we draw random, we prevent the network from only learning about what it is immediately doing in the environment, and allow it to learn from a more varied array of past experiences.Each of these experiences are stored as a tuple of <**state,action,reward,next state**>. The Experience Replay buffer stores a fixed number of recent memories (memory capacity), and as new ones come in, old ones are removed. When the time comes to train, we simply draw a uniform batch of random memories from the buffer, and train our network with them.

3.1 Exploration vs Exploitation

Exploration:

- Discover better action selections.
- Improve the knowledge about the environment.

Exploitation:

- Maximize the reward based on what agent already knows.

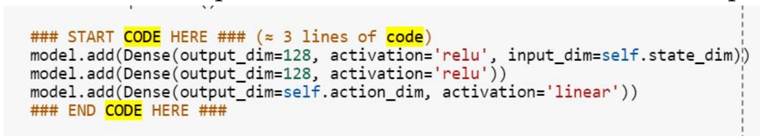
4 Explaination of Code Implementation

Replies to few questions asked in project.pdf:

4.1 What parts have you implemented?

I have implemented the following:

Neural Network with LINEAR → RELU → LINEAR → RELU → LINEAR. Activation function for the first and second hidden layers is **relu**. Activation function for the output layer is linear, that will return real values. Input dimensions for the first hidden layer equals to the size of the observation space (state_size). Number of hidden nodes is 128 for both hidden layers. Number of the output is same as the size of the action space (action_size).

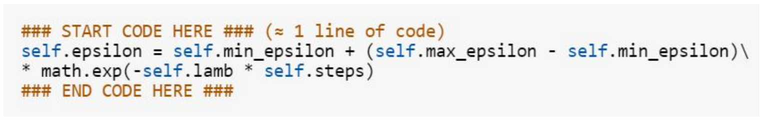


```
### START CODE HERE ### (≈ 3 lines of code)
model.add(Dense(output_dim=128, activation='relu', input_dim=self.state_dim))
model.add(Dense(output_dim=128, activation='relu'))
model.add(Dense(output_dim=self.action_dim, activation='linear'))
### END CODE HERE ###
```

Exponential-decay equation for epsilon has been implemented.

$$\epsilon = \epsilon_{min} + (\epsilon_{max} - \epsilon_{min}) * e^{-\lambda|s|} \tag{1}$$

where, $\mathbf{E_{max}}$ $\mathbf{E_{min}} \in [0,1]$
γ is a hyper-parameter for ϵ
and $|\mathbf{S}|$ is the total number of steps.

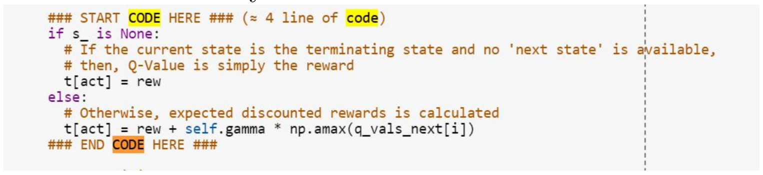


```
### START CODE HERE ### (≈ 1 line of code)
self.epsilon = self.min_epsilon + (self.max_epsilon - self.min_epsilon)\
* math.exp(-self.lamb * self.steps)
### END CODE HERE ###
```

Q - function formula has been implemented.

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

where, $\gamma \in [0,1]$ is the discounting factor,
$\mathbf{Q(s_t, a_t, \theta)}$ are the Q values of every action of next state out of which maximum value is considered.
and rt is the reward observed for any action.



```
### START CODE HERE ### (≈ 4 line of code)
if s_ is None:
  # If the current state is the terminating state and no 'next state' is available,
  # then, Q-Value is simply the reward
  t[act] = rew
else:
  # Otherwise, expected discounted rewards is calculated
  t[act] = rew + self.gamma * np.amax(q_vals_next[i])
### END CODE HERE ###
```

4.2 What is their role in training the agent?

1. **Neural Network:** To implement deep Q-learning, we have used neural network. For the fruit and the agent, a input of x and y co-ordinates is provided. The neural network predicts Q values of each action based on these inputs. The past experiences of the agent are stored as a tuple of ¡state,action,reward,next state¿ and is used for training the network. A uniform batch of random memories from the buffer is taken and used for training the network.

2. **Exponential-decay:** This is used for Exploration vs Exploitation. Our agent will randomly select its action at first by a certain percentage, called exploration rate or epsilon. This is because at first, it is better for the agent to try all kinds of things before it starts to see the patterns. When it is not deciding the action randomly, the agent will predict the reward value based on the current state and pick the action that will give the highest reward. We want our agent to decrease the number of random actions, as it goes, so we introduce an exponential-decay epsilon, that eventually will allow our agent to explore the environment.

3. **Q - function:** The goal of Q-learning is to learn a policy, which tells an agent what action to take under what circumstances and how to reach a goal faster. Discount factor γ determines the importance of future rewards. The Q-value of each cell is the maximum expected future reward for that given state and action.

4.3 Can implementation snippets be improved and how it will influence training the agent?

We can improve our implementation using the following techniques:

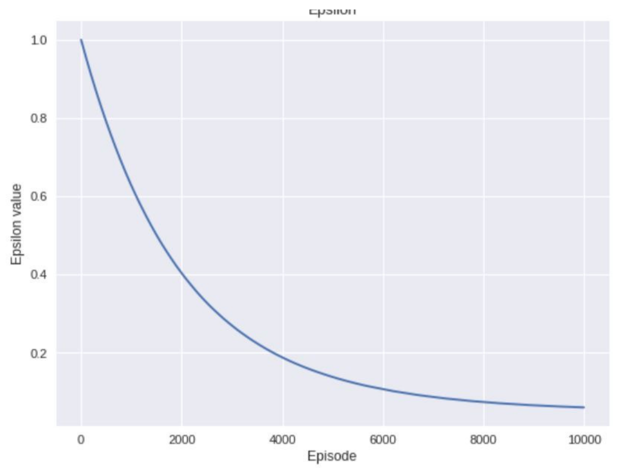
1. **Using Deeper Neural Network:** By applying Deep Q-Networks (DQN), the algorithm will store all of the agent's experiences and then randomly samples and replays these experiences to provide diverse and de-correlated training data, which will lead to a stable learning. Hence, in effect the agent will learn faster.

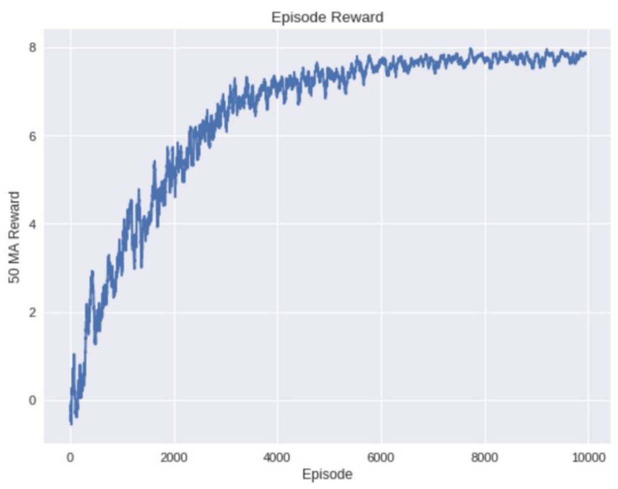
2. As this is a simpler 5*5 environment, we can stop the model by stopping the agent from exploring after certain number of episodes, as by that time the agent learns to navigate much faster. Lesser number of exploration will result in **Higher Mean Reward.**. This solution is only feasible for small environments.

3. Increasing the number of hidden layer had a positive effect on learning of agent. When the no of hidden layers was increased to 3, Tom was able to catch Jerry early.This increased the time taken per episode by the agent. This can help in complex environments as it can increase the performance significantly.

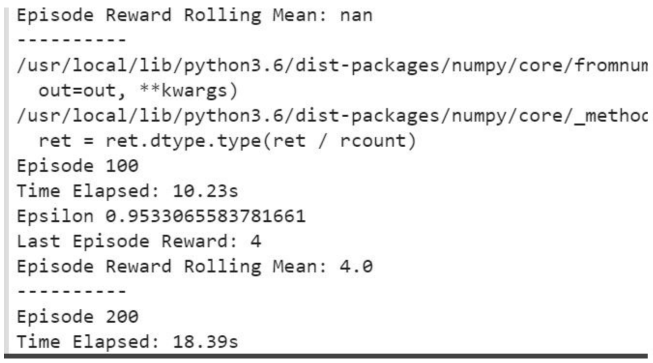
4.4 How quickly your agent were able to learn?

The best **Reward for the agent was 8** and the **Reward Rolling Mean was 6.38**. By viewing the graph below we can note that its around **3000th** episode where the reward curve starts stabilizing. The reward starts with a negative value which can be attributed to the fact that the agent takes random action to explore the environment. Once, it is able to catch the fruit it starts learning the best rewarding path to the fruit and takes actions accordingly. Below is a graph which describes the best reward and epsilon curve

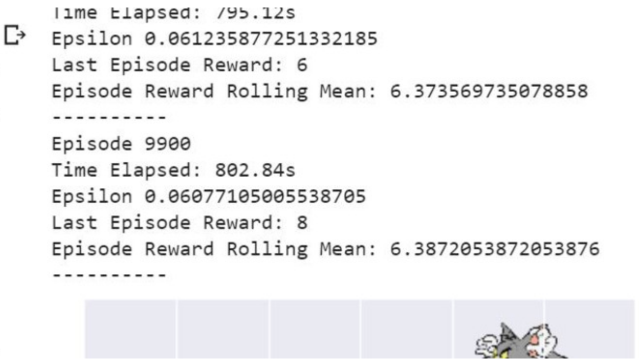




The time taken for the agent since start of exploration till it reaches the goal is depicted below:



```
Episode Reward Rolling Mean: nan
----------
/usr/local/lib/python3.6/dist-packages/numpy/core/fromnum
  out=out, **kwargs)
/usr/local/lib/python3.6/dist-packages/numpy/core/_methoc
  ret = ret.dtype.type(ret / rcount)
Episode 100
Time Elapsed: 10.23s
Epsilon 0.9533065583781661
Last Episode Reward: 4
Episode Reward Rolling Mean: 4.0
----------
Episode 200
Time Elapsed: 18.39s
```



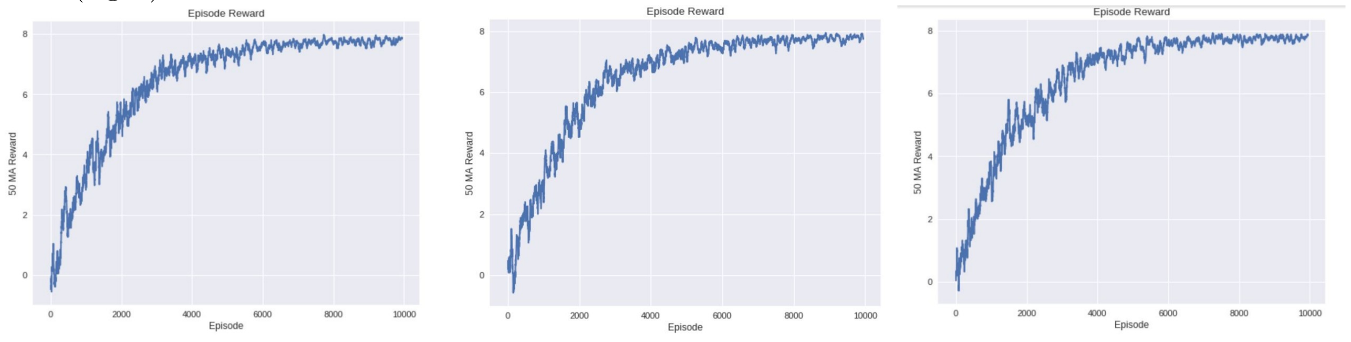
```
Time Elapsed: 795.12s
Epsilon 0.061235877251332185
Last Episode Reward: 6
Episode Reward Rolling Mean: 6.373569735078858
----------
Episode 9900
Time Elapsed: 802.84s
Epsilon 0.06077105005538705
Last Episode Reward: 8
Episode Reward Rolling Mean: 6.3872053872053876
----------
```

5 Hyperparameters Tuning

Following hyper-parameters tunings were done and the results are as described below:

5.1 Change in Hidden Nodes

A Hidden layer is one which transform the inputs into something that the output layer can use. Below results describes the rewards for number of hidden layer 128(left), 256(middle)and 64(right)



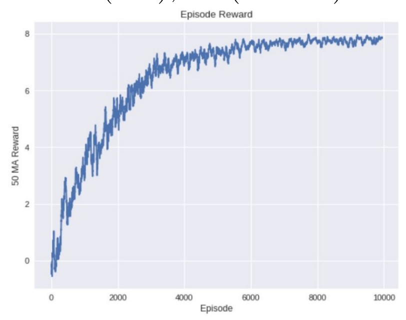
Hidden Nodes	No. of Episodes	Mean for First Reward	Mean for last Reward	Total Time taken	Best Reward
128	10000	4	6.38	802.84	8
256	10000	-2	6.37	919.67	8
64	10000	0	6.41	876.02	8

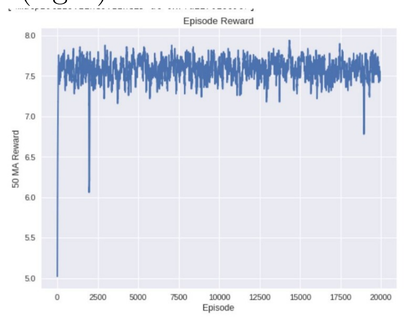
Table 1: Table for mean at episode 100, mean at last episode 9900 and total time taken

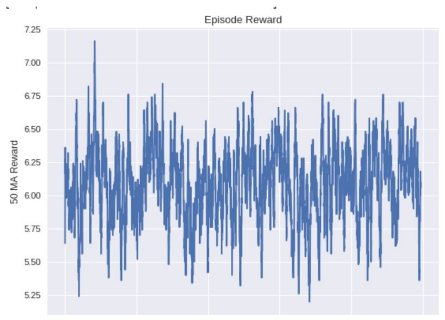
5.2 Change in $\epsilon_{\min}$

Exploration Rate or Epsilon is the rate by which our agent will randomly select its action at first by a certain percentage, called Below results describes the rewards when $\epsilon_{\min}$ was

0.05(left), 0.1(middle) and 0.3(right)





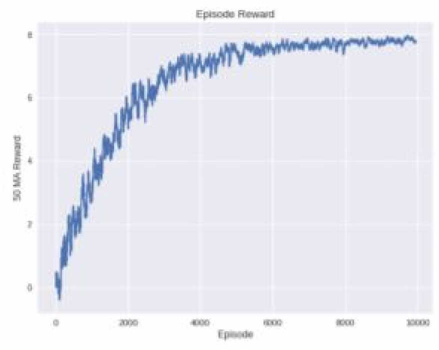


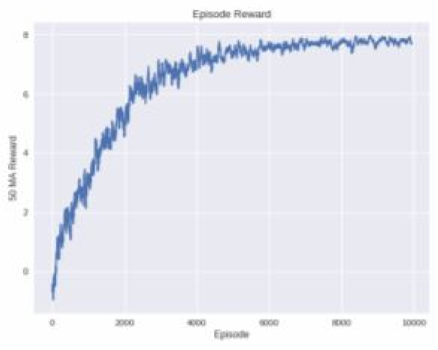
Hidden Nodes	No. of Episodes	Mean for First Reward	Mean for last Reward	Total Time taken	Best Reward
128	10000	4	6.38	802.84	8
128	20000	-2	6.19	1620.60	8
128	10000	0	5.89	880.44s	6

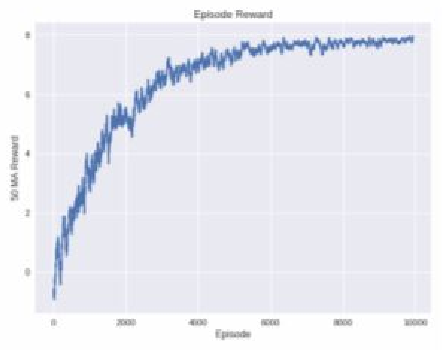
Table 1: Table for mean for first and last reward when $\epsilon_{\textbf{min}}$ changes

5.3 Change in Discounting Factor γ

Below **Reward** graph depicts the reward changes when γ value is considered to be 0.00005(left), 0.9(middle) and 0.3(right)





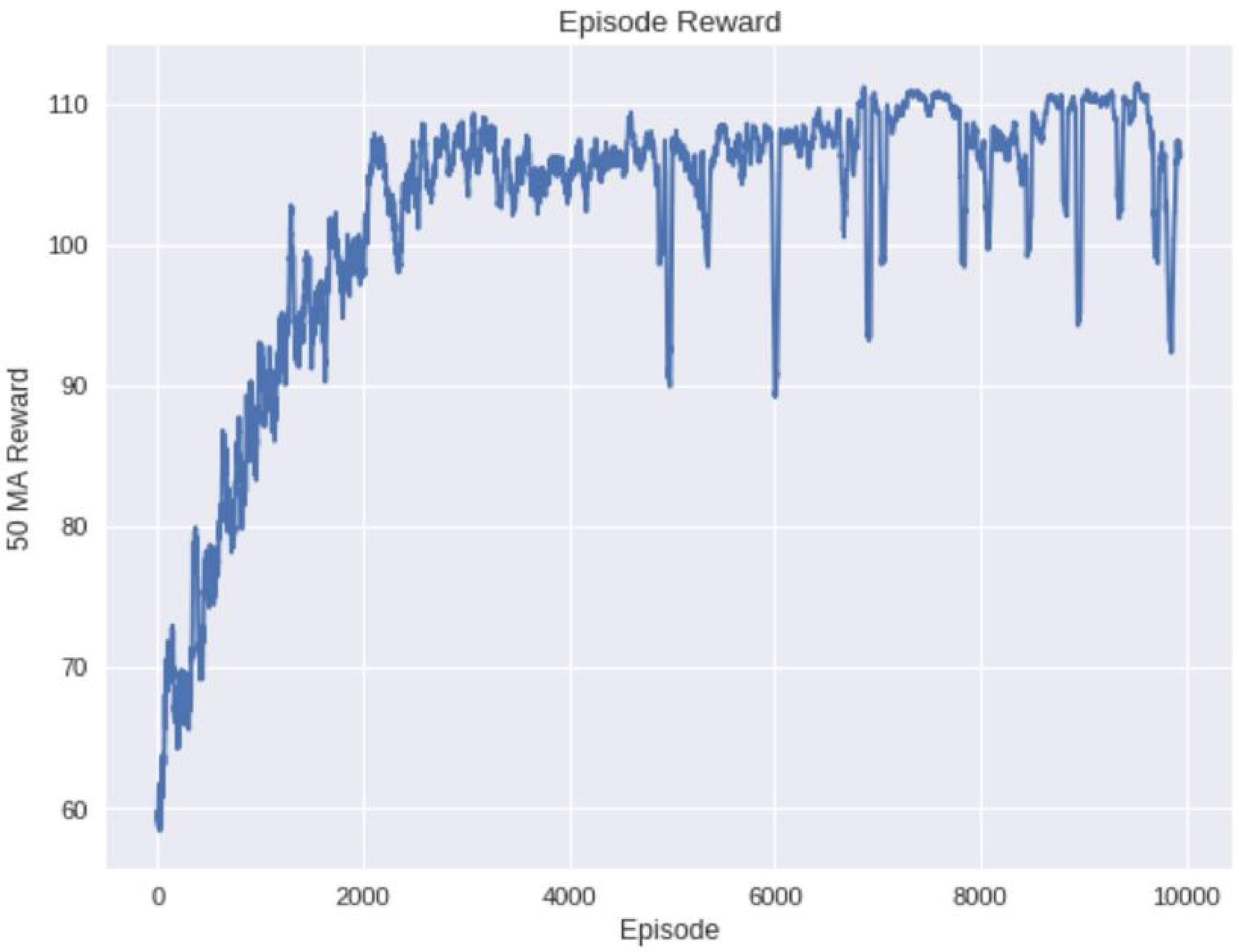


γ	Mean for First Reward	Mean for last Reward	Total Time taken	Best Reward
0.00005	4	6.38	802.84s	8
0.9	-2	6.30	819.36s	8
0.3	0	6.28	860.05s	6

Table 1: Table for mean for first and last reward when γ changes

5.4 Change in Rewards

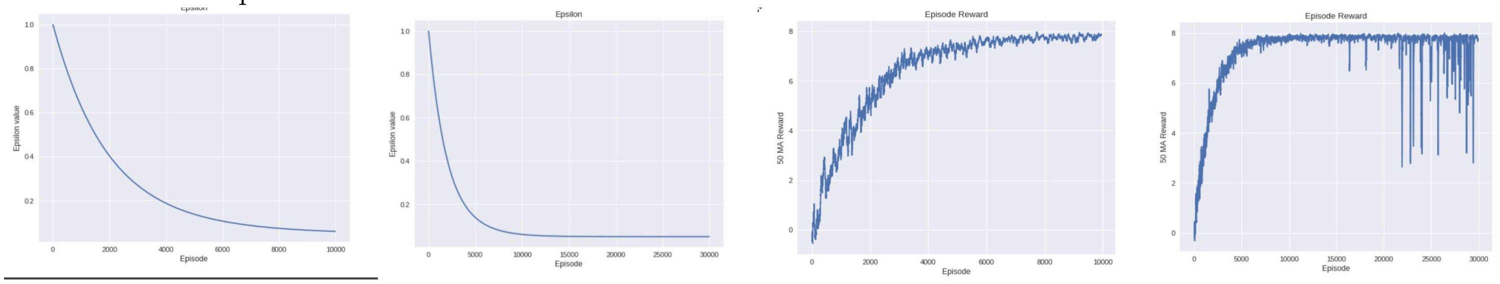
A reward of 15 was given if Tom successfully catches jerry and a reward of -5 if it moves away from Jerry. The reward graph for both is depicted below:



The learning curve as seen above fluctuates a lot and from the above graph it can be said that the maximum possible reward for this scenario is about 100.

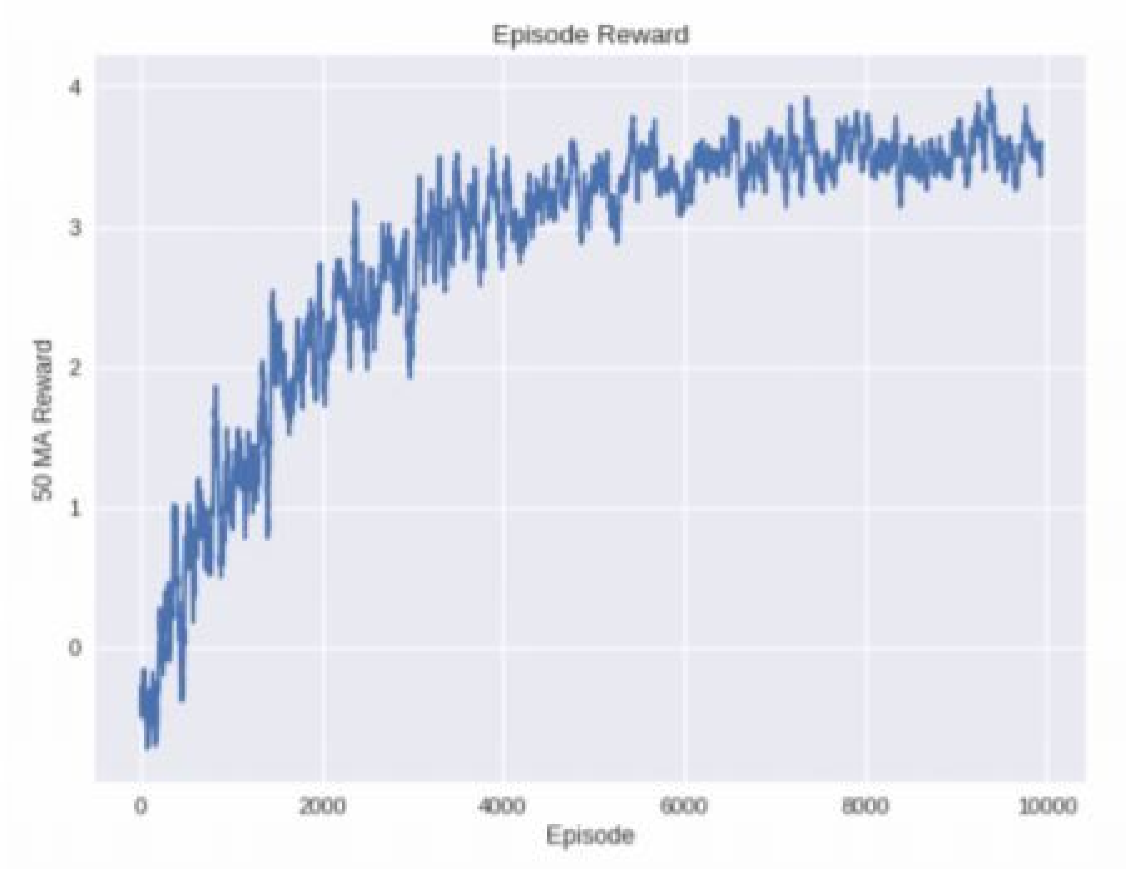
5.5 Change in Number of Episodes

The below figure depicts the graphs for **Epsilon(left) and Rewards(Right)** for 10000 and 30000 episodes:



5.6 Change in spawning positions of Tom and Jerry

The reward graph when Tom and Jerry will spawn at different places after every Environment reset is shown below.



As the position changes randomly, a lot of variation is observed. After few episodes the agent learns and is able to catch jerry successfully. This can be proved by looking at the graph above i.e after episode 6000 there is no negative fall.

6 Solutions - Writing Tasks

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.**

Answer If the agent always chooses the action which will maximize the Q-value then it gets stuck in non-optimal policies because it does not explore enough to find the best action from each state. As stated in project description the agent will land up in local maxima rather than finding the overall which is not intended. Any agent will end up in maximum rewards only if it keeps exploring in spite of the fact that it has already attained optimal policy. It is also possible that agent might discover the best optimal action sequence but will not be able to find the best optimal path if a change is introduced in the environment.

Following are the ways by which we could force the agent to explore:

1. Using **epsilon and exponential decay**, we can force our agent to take random choices at the beginning and then gradually decrease. Using this technique the agent will choose random actions at the start as epsilon value is higher but will gradually choose the path which will give the maximum value. As it is mentioned even if we find an optimal policy we should not stop exploring, so epsilon will never reach to 0.

2. It can pick random actions occasionally.

3. Initialize the initial values in Q-table with random high values, so that it will take maximum of those random values and will start exploring.

4. Use Exploration Functions and update Q-function formula. This takes a value estimate u and a visit count n, and returns an optimistic utility.
   The modified Q function would be:

$$Q = r_t + \gamma * max_a Q(s_{t+1}, a_t; \Theta) N(s_{t+1}, a_t) \tag{2}$$

Question **Calculate Q-value for the given states and provide all the calculation steps.**

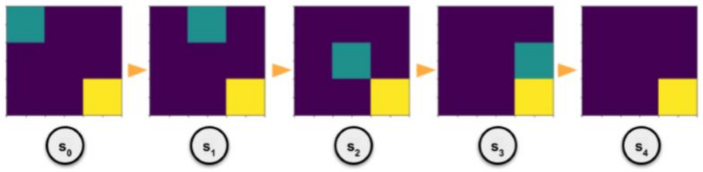


Figure 1: One of the possible optimal actions sequence

Answer Consider a deterministic environment which is a 3x3 grid, where one space of the grid is occupied by the agent (green square) and another is occupied by a goal (yellow square). The agents action space consists of 4 actions: UP, DOWN, LEFT, and RIGHT. The goal is to have the agent move onto the space that the goal is occupying in as little moves as possible. Initially, the agent is set to be in the upper-left corner and the goal is in the lower-right corner.
The agent receives a reward of:

- 1 when it moves closer to the goal
- -1 when it moves away from the goal
- 0 when it does not move at all (e.g., tries to move into an edge)

Calculation We will be using the following equation to calculate **Q-Table**

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

where value of γ is 0.99

	RIGHT	LEFT	UP	DOWN
$\mathbf{S_0}$	??	??	??	??
$\mathbf{S_1}$	??	??	??	??
$\mathbf{S_2}$	??	??	??	??
$\mathbf{S_3}$	??	??	??	??
$\mathbf{S_4}$	??	??	??	??

Table 1: Q-Table at Start

Step 1 We will start calculating the **Q-function** from the last state.
Let's consider the $\mathbf{State, S_4}$
As $\mathbf{State, S_4}$ is a self abosrbing state the values will be 0.

$$Q(S_4, RIGHT) = 0$$

$$Q(S_4, LEFT) = 0$$

$$Q(S_4, UP) = 0$$

$$Q(S_4, DOWN) = 0$$

	RIGHT	LEFT	UP	DOWN
$\mathbf{S_0}$	??	??	??	??
$\mathbf{S_1}$	??	??	??	??
$\mathbf{S_2}$	??	??	??	??
$\mathbf{S_3}$	??	??	??	??
$\mathbf{S_4}$	0	0	0	0

Q-Table after Step 1

Step 2: Consider the $\mathbf{State, S_3}$

$$\begin{aligned} Q(S_3, RIGHT) &= 0 + 0.99 * max_Q(s_{23}, a) \\ &= 0 + 0.99 * 1 = 0.99 \end{aligned}$$

$$\begin{aligned} Q(S_3, LEFT) &= -1 + 0.99 * max_Q(s_2, a) \\ &= -1 + 0.99 * (max_Q(1 + 0.99 * max_Q(s_3, a)) \\ &= -1 + 0.99 * (1 + 0.99 * 1) \\ &= 0.9701 \end{aligned}$$

$$\begin{aligned} Q(S_3, UP) &= -1 + 0.99 * max_Q(s_{13}, a) \\ &= -1 + 0.99 * (max_Q(1 + 0.99 * max_Q(s_3, a)) \\ &= -1 + 0.99 * (1 + 0.99 * 1) \\ &= 0.9701 \end{aligned}$$

since, max Q value for S_3 will be 1 (the agent reaching the reward)

$$\begin{aligned} Q(S_3, DOWN) &= 1 + 0.99 * max_Q(s_4, a) \\ &= 1 + 0.99 * 0 \\ &= 1 \end{aligned}$$

	RIGHT	LEFT	UP	DOWN
$\mathbf{S_0}$	??	??	??	??
$\mathbf{S_1}$	??	??	??	??
$\mathbf{S_2}$	??	??	??	??
$\mathbf{S_3}$	0.99	0.9701	0.9701	1
$\mathbf{S_4}$	0	0	0	0

Q-Table after Step 2

Step 3 Consider the $\mathbf{State, S_2}$

$$\begin{aligned} Q(S_2, RIGHT) &= 1 + 0.99 * max_Q(s_3, a) \\ &= 1 + 0.99 * 1 = 1.99 \end{aligned}$$

$$\begin{aligned} Q(S_2, LEFT) &= -1 + 0.99 * max_Q(s_{21}, a) \\ &= -1 + 0.99 * (max_Q(1 + 0.99 * max_Q(s_2, a)) \\ &= -1 + 0.99 * (1 + 0.99 * 1.99) \\ &= -1 + 0.99 * (2.9701) \\ &= -1 + 2.9403 \\ &= 1.9403 \end{aligned}$$

$$\begin{aligned} Q(S_2, UP) &= -1 + 0.99 * max_Q(s_{12}, a) \\ &= -1 + 0.99 * (max_Q(1 + 0.99 * max_Q(s_{13}, a)) \\ &= -1 + 0.99 * (max_Q(1 + 0.99 * (max_Q(1 + 0.99 * max_Q(s_3, a))) \\ &= -1 + 0.99 * (max_Q(1 + 0.99 * 1.99) \\ &= -1 + 0.99 * (2.9701) \\ &= -1 + 2.9403 \\ &= 1.9403 \end{aligned}$$

$$\begin{aligned} Q(S_2, DOWN) &= 1 + 0.99 * max_Q(s_{32}, a) \\ &= 1 + 0.99 * 1 = 1.99 \end{aligned}$$

In this case, S32 is symmetric to S3.

	RIGHT	LEFT	UP	DOWN
$\mathbf{S_0}$	??	??	??	??
$\mathbf{S_1}$	??	??	??	??
$\mathbf{S_2}$	1.99	1.9403	1.9403	1.99
$\mathbf{S_3}$	0.99	0.9701	0.9701	1
$\mathbf{S_4}$	0	0	0	0

Q-Table after Step 3

Step 4 Consider the $\mathbf{State, S_1}$

$$\begin{aligned} Q(S_1, RIGHT) &= 1 + 0.99 * max_Q(s_{13}, a) \\ &= 1 + 0.99 * 1.99 \\ &= 1 + 1.9701 \\ &= 2.9701 \end{aligned}$$

$$\begin{aligned} Q(S_1, LEFT) &= -1 + 0.99 * max_Q(s_0, a) \\ &= -1 + 0.99 * max_Q(1 + 0.99 * max_Q(s_1, a)) \\ &= -1 + 0.99 * max_Q(1 + 0.99 * max_Q(1 + 0.99 * max_Q(s_{13}, a))) \\ &= -1 + 0.99 * max_Q(1 + 0.99 * max_Q(1 + 0.99 * 1.99)) \\ &= -1 + 0.99 * max_Q(1 + 0.99 * 2.9701) \\ &= -1 + 0.99 * 3.940399 \\ &= -1 + 3.9009 \\ &= 2.9009 \end{aligned}$$

$$\begin{aligned} Q(S_1, UP) &= 0 + 0.99 * max_Q(s_1, a) \\ &= 0 + 0.99 * max_Q(1 + 0.99 * max_Q(s_{13}, a)) \\ &= 0 + 0.99 * max_Q(1 + 0.99 * 1.99) \\ &= 0 + 2.9403 \\ &= 2.9403 \end{aligned}$$

$$\begin{aligned} Q(S_1, DOWN) &= 1 + 0.99 * max_Q(s_2, a) \\ &= 1 + 0.99 * 1.99 \\ &= 1 + 1.9701 \\ &= 2.9701 \end{aligned}$$

	RIGHT	LEFT	UP	DOWN
$\mathbf{S_0}$	??	??	??	??
$\mathbf{S_1}$	2.9701	2.9009	2.9403	2.9701
$\mathbf{S_2}$	1.99	1.9403	1.9403	1.99
$\mathbf{S_3}$	0.99	0.9701	0.9701	1
$\mathbf{S_4}$	0	0	0	0

Q-Table after step 4

Step5 Consider the $\mathbf{State, S_0}$

$$\begin{aligned} Q(S_0, RIGHT) &= 1 + 0.99 * max_Q(s_1, a) \\ &= 1 + 2.9403 \\ &= 3.9403 \end{aligned}$$

$$\begin{aligned} Q(S_0, LEFT) &= 0 + 0.99 * max_Q(s_0, a) \\ &= 0 + 0.99 * max_Q(1 + 0.99 * max_Q(s_1, a)) \\ &= 0 + 0.99 * max_Q(1 + 2.9403) \\ &= 0 + 3.900897 \\ &= 3.9008 \end{aligned}$$

$$\begin{aligned} Q(S_0, UP) &= 0 + 0.99 * max_Q(s_0, a) \\ &= 0 + 0.99 * max_Q(1 + 0.99 * max_Q(s_1, a)) \\ &= 0 + 0.99 * max_Q(1 + 2.9403) \\ &= 0 + 3.900897 \\ &= 3.9008 \end{aligned}$$

$$\begin{aligned} Q(S_0, DOWN) &= 1 + 0.99 * max_Q(s_{21}, a) \\ &= 1 + 0.99 * max_Q(1 + 0.99 * max_Q(s_2, a)) \\ &= 1 + 0.99 * 2.9701 \\ &= 3.9403 \end{aligned}$$

since, S21 and S1 are symmetric cases.

The Q-table would be:

	RIGHT	LEFT	UP	DOWN
$\mathbf{S_0}$	3.9403	3.9008	3.9008	3.9403
$\mathbf{S_1}$	2.9701	2.9009	2.9403	2.9701
$\mathbf{S_2}$	1.99	1.9403	1.9403	1.99
$\mathbf{S_3}$	0.99	0.9701	0.9701	1
$\mathbf{S_4}$	0	0	0	0

Table 1: Final Q-Table

7 Conclusions

From this experiment, we learnt how to apply reinforcement learning to teach the agent to navigate in the grid-world environment.We saw the application of Q-learning so that a agent will learn quickly. We also implemented DQN for different Atari environments.Hyperparameters were tuned for model and the results obtained was observed. and plotted in the paper.

References

[1] Project Description
https://ublearns.buffalo.edu/webapps/blackboard/content/listContent.jsp?course_id=_156042_1&content_id=_4611149_1&mode=reset

[2] Project Description 2
https://ublearns.buffalo.edu/bbcswebdav/pid-4776954-dt-content-rid-21266552_1/courses/2189_24904_COMB/project4_task%5BUPDATED%5D%281%29.pdf

[3] Reinforcement Learning
https://skymind.ai/wiki/deep-reinforcement-learning

[4] Deep Learning
http://deeplearning.net/

[5] Markov Decision Process
https://en.wikipedia.org/wiki/Markov_decision_process

[6] Deep-reinforcement-learning
https://skymind.ai/wiki/deep-reinforcement-learning

[7] Q-Learning
https://en.wikipedia.org/wiki/Q-learning