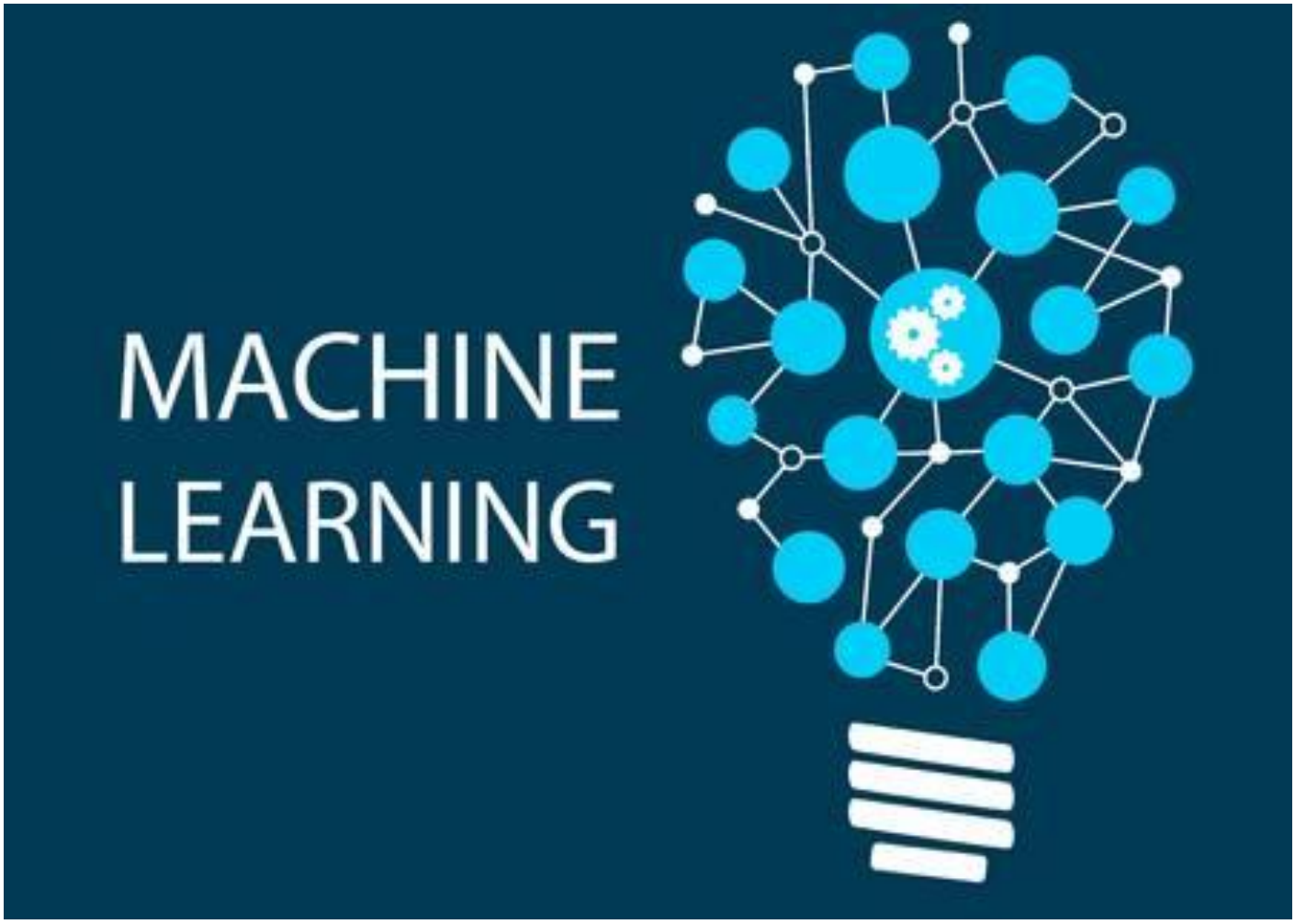
# Project 4: Tom and Jerry in Reinforcement learning




**Darshan Subhash Nevgi**
UBIT: darshann
Person Number : 50291068

Computer Science , University at Buffalo
12/05/2018

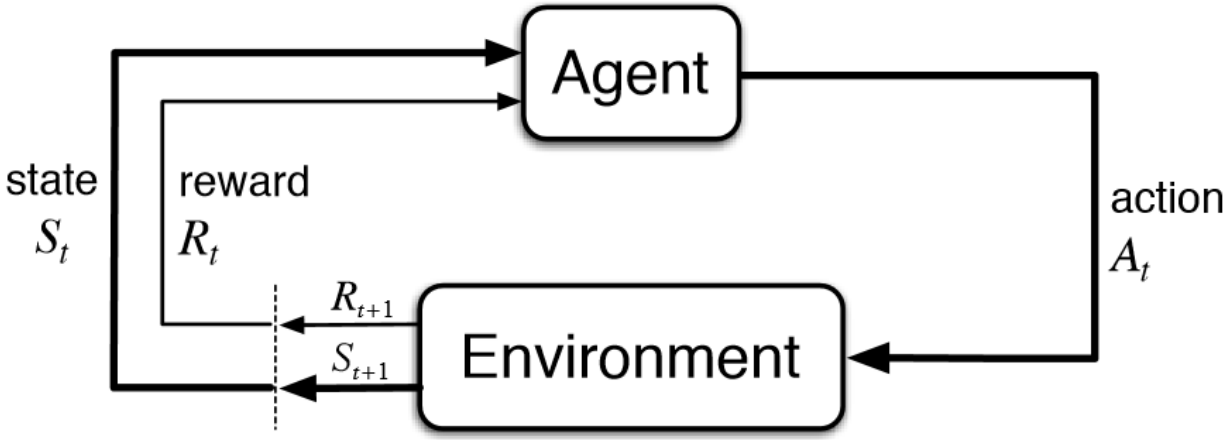
# Contents

## Reinforcement Learning and Project Topic Introduction

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, and results in a policy that maps states to actions to maximize the long-term total reward as a delayed supervision signal. Reinforcement learning combining with the neural networks has made great progress recently, including playing Atari games and beating world champions at the game of Go. It is also widely used in robotics.

### Markov Decision Process

The project combines reinforcement learning and deep learning.




### Goal

Our main goal is to let our agent learn the shortest path to the goal. In the environment the agent controls a green square, and the goal is to navigate to the yellow square (reward +1), using the shortest path. At the start of each episode all squares are randomly placed within a 5x5 grid-world. The agent has 100 steps to achieve as large a reward as possible. They have the same position over each reset, thus the agent needs to learn a fixed optimal path.

# Description of code-snippets implementations

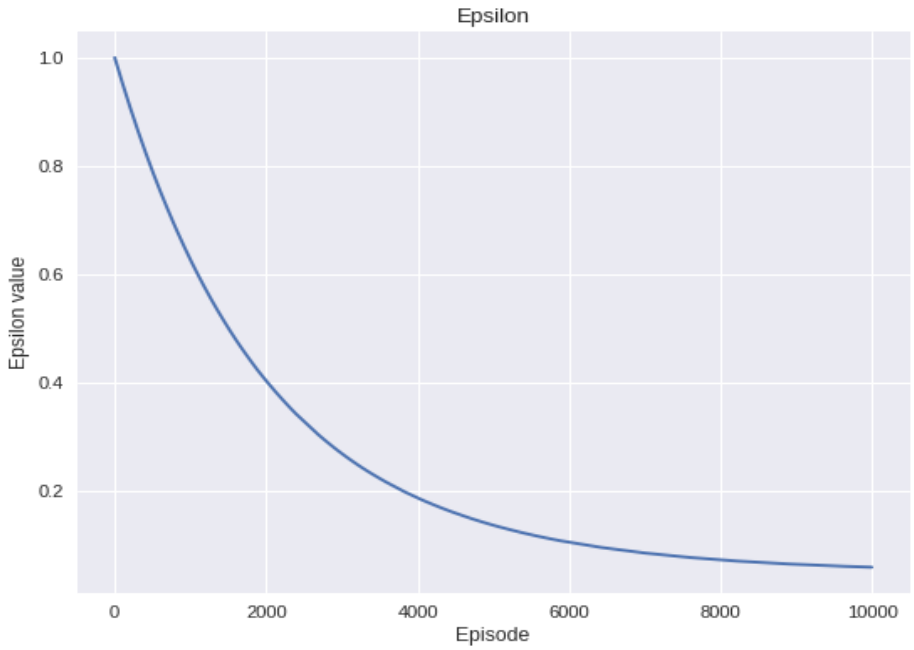
*I have*

***1) Implemented exponential-decay formula for epsilon.***

Epsilon is the exploration factor which denotes the frequency level of how much exploration to do. Exponential decay is used to balance between exploration and exploitation. Exponential decay ensures that we reduce random steps as Model learns through experience .

```
self.epsilon = self.min_epsilon + (self.max_epsilon-self.min_epsilon) * math.exp(-self.lamb*abs(self.steps))
```

Following is graph showing decay in value of Epsilon over Episodes




I tried working with linear and quadratic decay but received better result in exponential decay only.

***2) Built a 3-layer neural network, using Keras library in Brain***

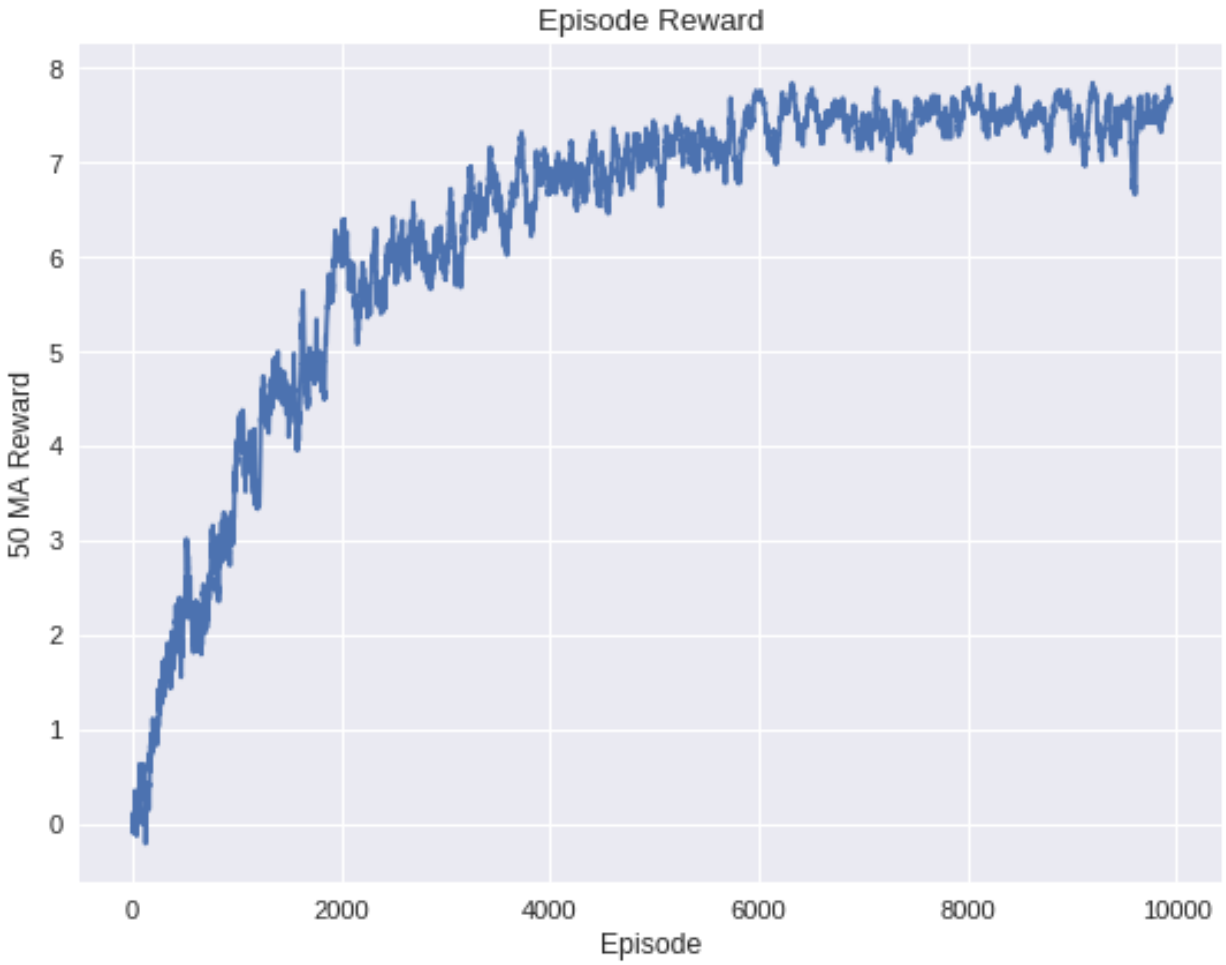
The 'brain' of the agent is where the model is created and held. I have used Neural network to implement brain functionality . Neural network accepts a state as input and outputs a vector q-values . so a trained neural network calculates the next action

```
model.add(Dense(128, input_dim=self.state_dim))
model.add(Activation('relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(output_dim=self.action_dim, activation='linear'))
```

I have tried to work with 4-hidden layers which increases running time and increases mean reward . I tried different activation functions like sigmoid and softmax but relu gave best results

*3) Implemented Q function*

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




the value Qt is equal to the immediate reward rt plus the maximal value of Qt+1, where t+1 is the next state and at is an action used to reach t+1.

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

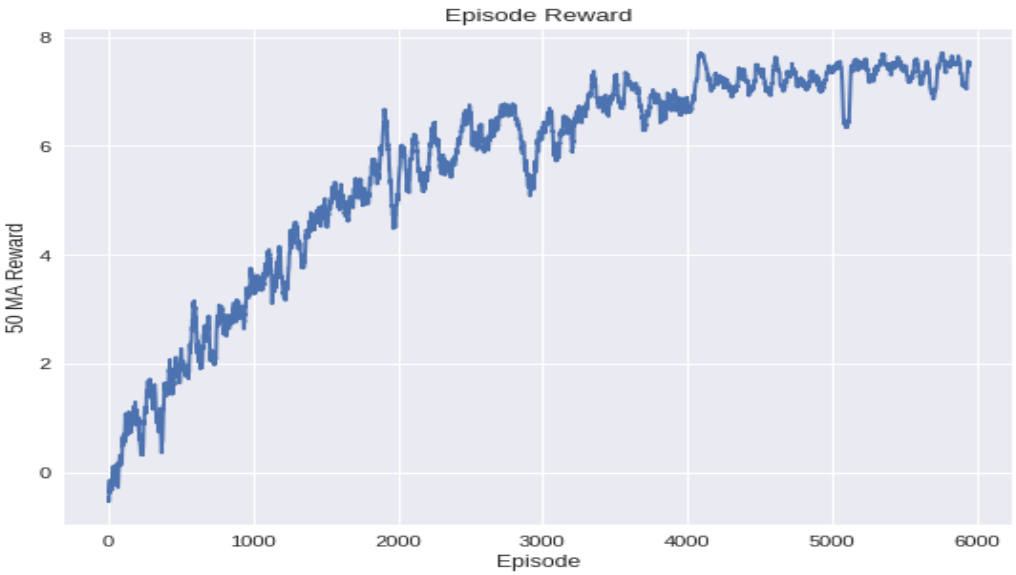
Q function is used to update value of Q recursively along episodes.

I though making changes in Q value but it does not do any good because then our model would not be considered as a DQN model

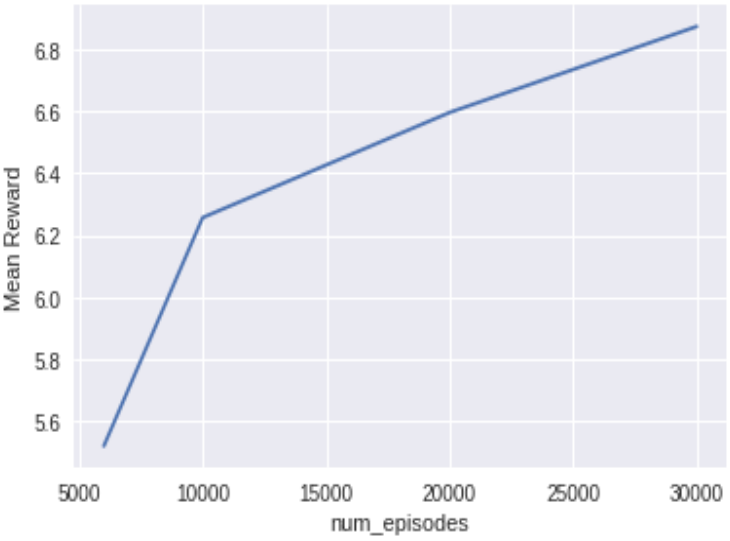
My agent was able to learn after 6000 episodes as values of Mean reward started converging

num_episodes = 6000

```
Episode 5900
Time Elapsed: 467.48s
Epsilon 0.10902800454476559
Last Episode Reward: 8
Episode Reward Rolling Mean: 5.520255128426133
```




After Increasing Number of Episodes I observed that time required for running increased but Mean Reward also increased in good quantity



# Writing tasks Solution

## 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.

In reinforcement learning if the agent always chooses the action that maximizes the Q-value, **Module will not be trained well with random actions and hence agent may not be able to reach to final outcome in optimum way.** Above approach will not allow Agent to explore various option to find optimum path hence solution may not be fully optimum.

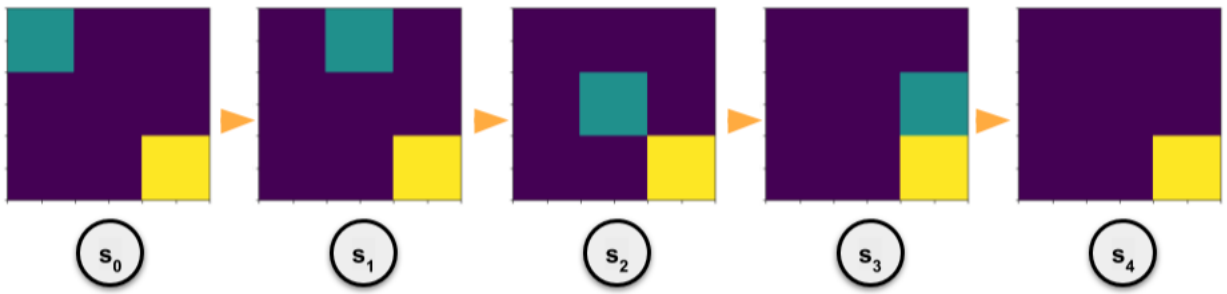
We can allow agent to Explore by having part of moves compulsorily random without referring to Max Q values this will allow agent to explore Rest of the moves can be done according to Max Q values .Below are the few sample approaches that can be used .% percentages can be changed.

- **Exploitation:** these are moves that policy $\pi$ dictates based on previous experiences. The policy function is used in about 90% of the moves before it is completed.

- **Exploration**: in about 10% of the cases, we take a completely random action in order to acquire new experiences (and possibly meet bigger rewards) which our strategy function may not allow us to make due to its restrictive nature. we can think of it as choosing a completely random new restaurant once in a while instead of choosing the routine restaurants that we already familiar with.

## 2. Calculate Q-value for the given states and provide all the calculation steps.

Given:
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)




Final Answer:

| | UP | DOWN | LEFT | RIGHT |
|---|---|---|---|---|
| **S0** | 3.900 | 3.940 | 3.900 | 3.940 |
| **S1** | 2.940 | 2.970 | 2.900 | 2.970 |
| **S2** | 1.940 | 1.990 | 1.940 | 1.990 |
| **S3** | 0.970 | 1 | 0.970 | 0.990 |
| **S4** | 0 | 0 | 0 | 0 |

The episode terminates as soon as the agent reaches the goal, The agent reaches the terminal state at s4, meaning it's the end of the episode and the agent can not make any moves**. Thus the Q-value function for s4 for any action is going to be 0.**

we will use below formula to calculate remaining values.

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

*Step By Step Calculation:*

For actions which are moving towards answer, reward is 1 and
Hence
*Step 1*
**(s3,DOWN) = 1.000**(rt: reward to move in correct direction)

because episode terminates at t+1 step for (s3,Down)

*Step 2*

(s2,RIGHT) = 1(rt:reward to move in correct direction) + 0.99(GAMMA)* 1(MAX of s3 calculated in step 1)

**(s2, RIGHT) = 1.990**

*Step 3*

(s1, DOWN) = 1(rt:reward to move in correct direction) + 0.99* 1.99((MAX of s2 calculated in step 2))

**(s1, DOWN) = 2.970**

*Step 4*

(s0, RIGHT) = 1(rt:reward to move in correct direction) + 0.99* 2.97((MAX of s1 calculated in step 3))

**(s0, RIGHT) = 3.940**

*Step 5*

(s0, DOWN) = 1(rt:reward to move in correct direction) + 0.99* 2.97

**(s0, DOWN) = 3.940**

*Step 6*

(s1, RIGHT) = 1(rt:reward to move in correct direction) + 0.99* 1.99

**(s1, RIGHT) = 2.970**

*Step 7*

(s2, DOWN) = 1(rt:reward to move in correct direction) + 0.99* 1

**(s2, DOWN) = 1.990**

For actions which are crossing edges, reward is 0 and next state is current state Hence

*Step 8*

**(s0, UP) = 0 + 0.99 * 3.94 = 3.900**

*Step 9*
**(s0, LEFT) = 0 + 0.99 * 3.94 = 3.900**

*Step 10*
**(s1, UP) = 0 + 0.99 * 2.97 = 2.940**

*Step 11*
**(s3,** RIGHT**) = 0 + 0.99 * 1 = 0.990**

For actions which are moving away from answer ,reward is -1

*Step 12*
**(s3, UP) = -1 + 0.99 * 1.99 = 0.970**

*Step 13*
**(s3, LEFT) = -1 + 0.99 * 1.99 = 0.970**

*Step 14*
**(s2, UP) = -1 + 0.99 * 2.94 = 1.940**

*Step 15*
**(s2, LEFT) = -1 + 0.99 * 2.94 = 1.940**

*Step 16*
**(s1, LEFT) = -1 + 0.99 * 3.94 = 2.900**

## References :

https://www.samyzaf.com/ML/rl/qmaze.html
https://stable-

baselines.readthedocs.io/en/master/modules/dqn.html
https://en.wikipedia.org