CSE 574: Introduction to Machine Learning

Amlan Gupta
#50288686
amlangup@buffalo.edu

Abstract

The project requires us to illustrate reinforcement learning, where we will teaching an agent to navigate in the grid-like environment. In the modeled game the task for the agent is to find the shortest path to the goal, given that the initial positions of agent and goal are deterministic. To solve the problem, we would apply deep reinforcement learning algorithm - DQN (Deep Q-Network).

1 Reinforcement Learning

Reinforcement learning is an special type of Machine Learning where an agent learn how to behave in a environment by performing actions and seeing the results. We will be explaining the jargons as a result the concept should be well-explained.

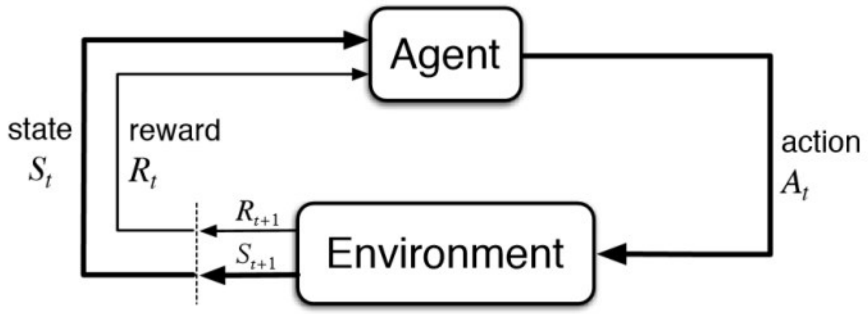


Figure 1: Reinforcement Learning

Agent: An agent takes actions; for example, in our case, Tom is the agent who acts to find Jerry through shortest possible path.

Action (A): A is the set of all possible moves the agent can make. An action is almost self-explanatory, but it should be noted that agents choose among a list of possible actions. In the grid-like environment like out example, Tom can either move left, right, up or down.

Environment: The world through which the agent moves. The environment takes the agent's current state and action as input, and returns as output the agent's reward and its next state. The environment sets some rules which needs be adhered to, it processes agent's actions and determine the consequences of it. In our case, the environment creates initial positions of Tom and Jerry, maintains a co-ordinate system for the current positions of them, calculate reward or a particular action, and decides if the goal has been reached.

State (S): A state is a concrete and immediate situation in which the agent finds itself; i.e. a specific place and moment, an instantaneous configuration that puts the agent in relation to other significant Object. It can the current situation returned by the environment, or any future situation. The co-ordinates of Tom and Jerry can be counted as a state for our use-case.

Reward (R): A reward is the feedback by which we measure the success or failure of an agent's actions. For one action we measure if Tom is closer to Jerry that he was in the previous. If it is closer we will give reward to reinforce this action, otherwise we will penalize for wrong action.

Discount factor (γ) : The discount factor is multiplied by future rewards as discovered by the agent in order to dampen these rewards' effect on the agent's choice of action. There could be situations where taking greedy path may not be the best possible solution, so we look ahead, and take future rewards into account.

our discounted cumulative expected rewards is = $R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+1}..$

Policy (π)**:** The policy is the strategy that the agent employs to determine the next action based on the current state. It maps states to actions, the actions that promise the highest reward.

Q-value or action-value (Q): The expected long-term return with discount, as opposed to the short-term reward R. Q π (s, a) refers to the long-term return of the current state s, taking action a under policy π. Q maps state-action pairs to rewards.

2 Implemented Snippets

2.1 3-layer Neural Network, using Keras Library

2.1.1 Implementation

```
model.add(Dense(128,activation='relu', input_shape=(self.state_dim,)))
model.add(Dense(128, activation='relu'))
model.add(Dense(self.action_dim))
```

2.1.2 Role

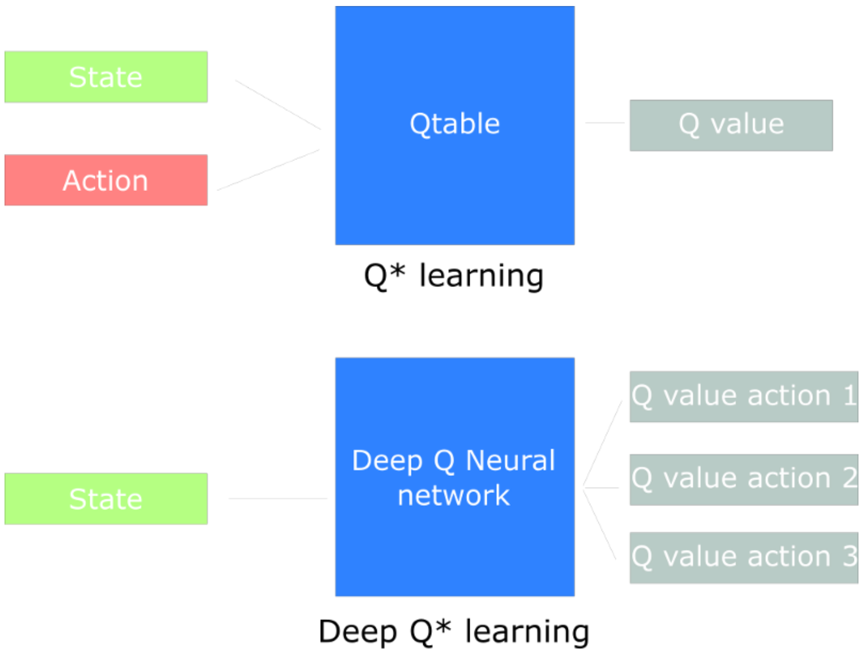


Figure 2: Deep Q neural Network

We can use Q-table to find the maximum expected future reward of an action, given a current state. However this is not scalable, the best idea in this case is to create a neural network that will approximate, given a state, the different Q-values for each action.

We are creating 3 layer neural network with input of 4 parameters, which are the co-ordinate of tom and Jerry. The neural network will predict Q value for each possible action. At the beginning, the prediction will be off the mark, however as the model keeps learning with new data and it will start giving better prediction.

2.1.3 Improvement

Increasing the nodes and add more layers in the neural network may increase the Reward Mean, hence stabilizing the prediction.

2.2 Exponential-decay formula for epsilon

2.2.1 Implementation

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

2.2.2 Role

Epsilon controls a very important concept in reinforcement learning, that is exploration/exploitation. It is important for the model to explore the possible outcomes to test how it's actions are getting rewarded or penalized. Once the model has enough data about it's actions only then it can take the best possible path to it's goal. For a higher value of epsilon random actions will be taken. With the reduction of epsilon, the number of random actions will be decreased and the agent will predict the reward value based on the current state and pick the action that will give the highest reward.

2.3 Q-function

2.3.1 Implementation

```
if i == batch_size-1:
    t[act] = rew
else:
    t[act] = rew + (self.gamma*np.max(q_vals_next[i]))
```

2.3.2 Role

Q(s, a) = r + γmax_a Q(s', a')

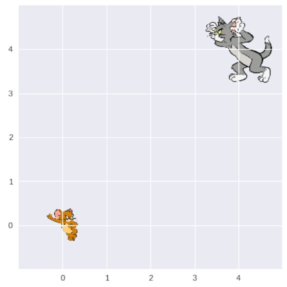
This equation, known as the Bellman equation, tells us that the maximum future reward is the reward the agent received for entering the current state s plus the maximum future reward for the next state s'. The gist of Q-learning is that we can iteratively approximate Q using the Bellman equation described above.

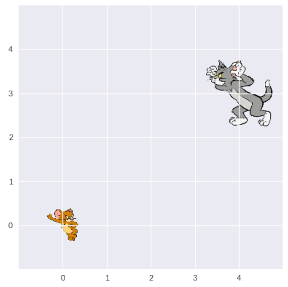
The initial approximations will most likely be completely random/wrong. However, as the agent explore more and more of the environment, the approximated Q values will start to converge to optimal Q-function.

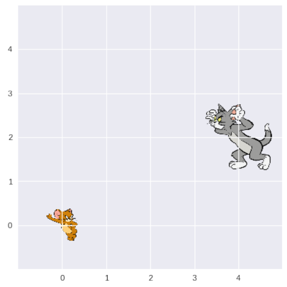
2.4 Observation

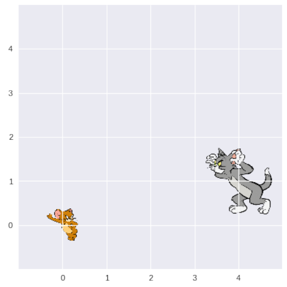
The model quickly learned how to predict the optimal Q function as we can see from the graph below, from 6000 episodes the model consistently start giving prediction. The rolling means stays near 6.

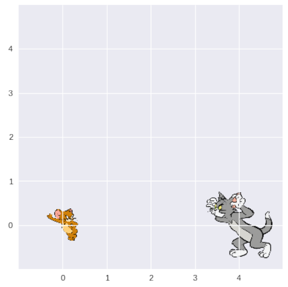
An optimal path rendered by the model:

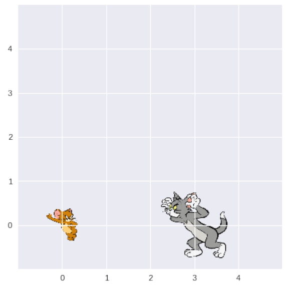

(a) State 0

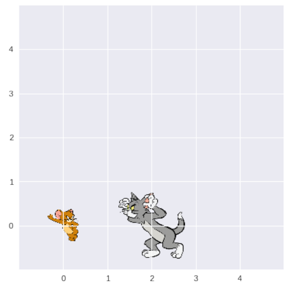

(b) State 1

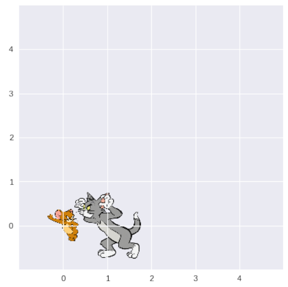

(c) State 2

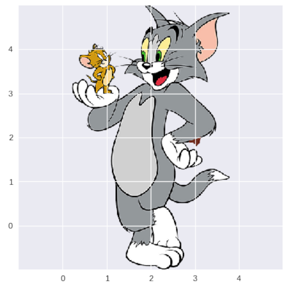

(d) State 3


(e) State 4


(f) State 5


(g) State 6


(h) State 7


(i) State 8

3 Hyper parameter Tuning

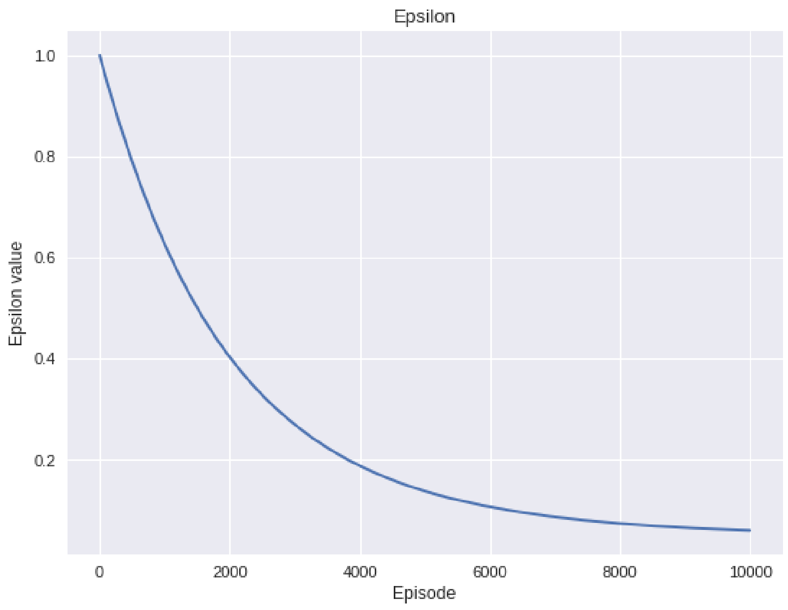


Figure 4: Changes of Epsilon over Episode: max epsilon = 1, min epsilon = 0.05

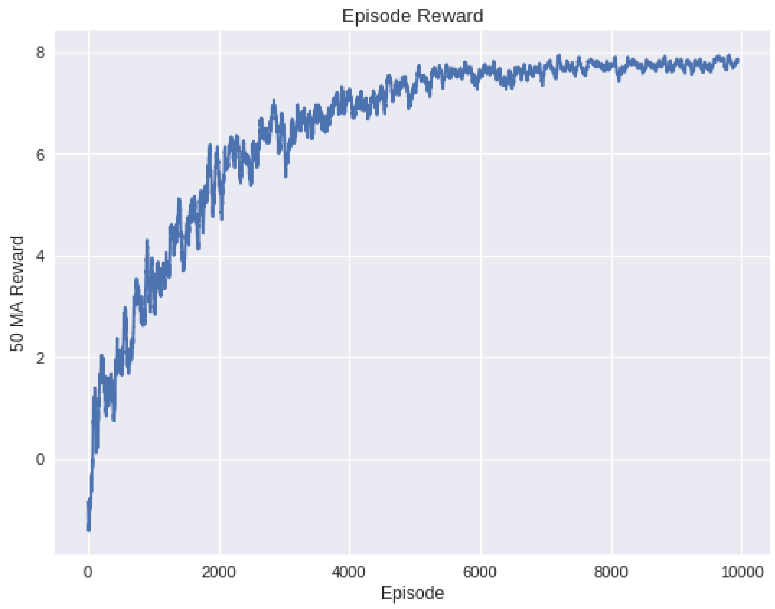


Figure 5: Gathered rewards over episodes

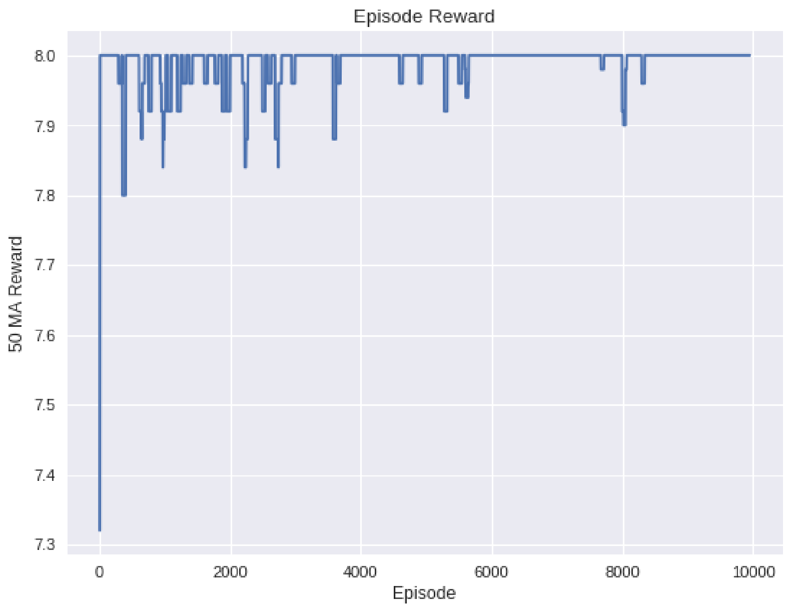


Figure 6: For epsilon = 0

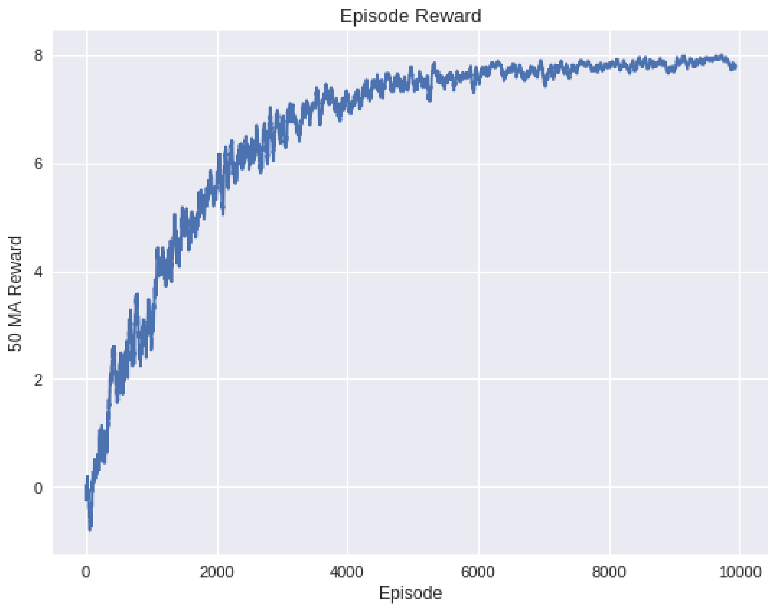


Figure 7: For Gamma = 0

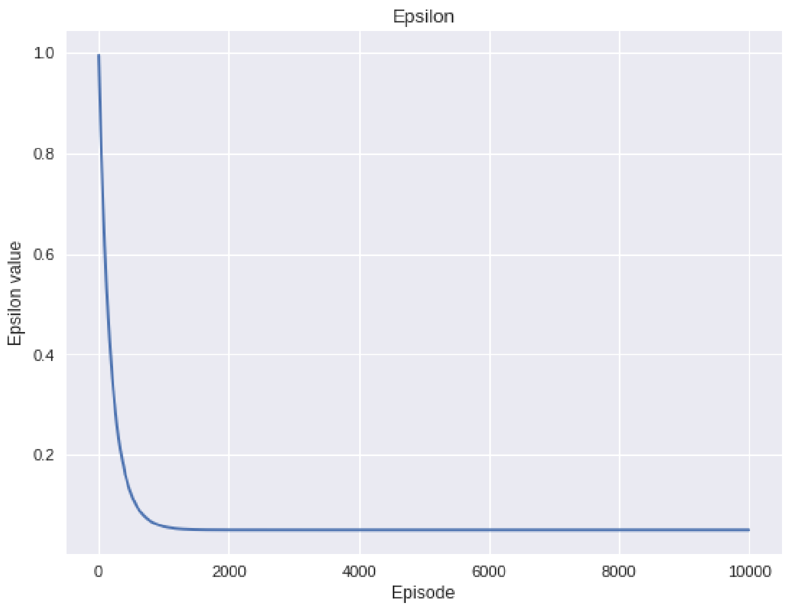


Figure 8: For lambda = .00055

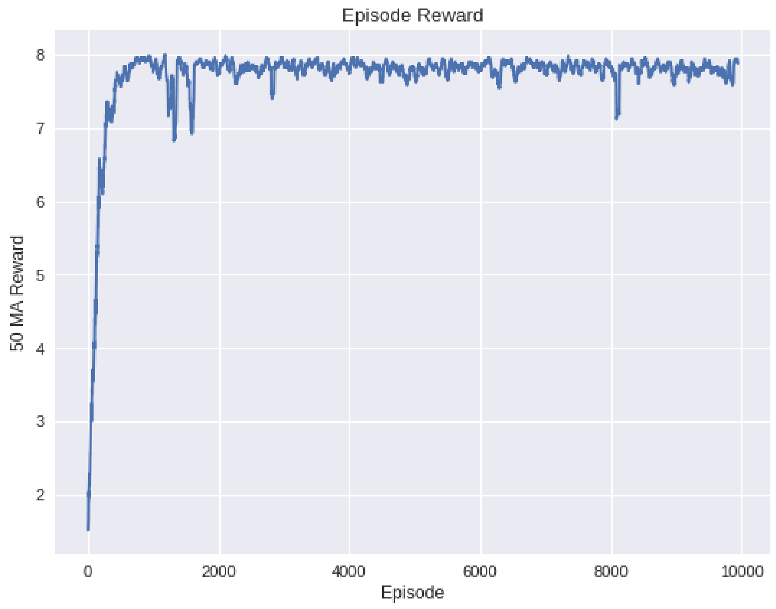


Figure 9: For lambda = .00055

4 Writing Tasks

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.**

   If the agent always chooses the action that maximized the Q-value that is basically working as Greedy algoithm. In a simple environment like ours, it greedy algorithm may work, however this is not a optimal solution. Just because the next state is the most rewarding path that does not mean it may have the most optimal solution.

For example, in our environment where Tom try to reach Jerry, if we introduce traps, just because the next most rewarded path is closer to Jerry does not mean it the most optimal, taking this step might be full of traps and Tom may have go around them.

Two ways to force the agent to explore are:

(a) R–max
(b) Explicit Explore or Exploit (E^3)

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

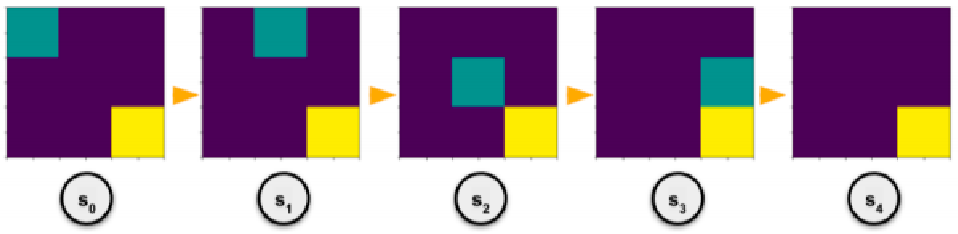


Figure 10: One of the possible optimal actions sequence

The agent's action space consists of 4 actions: UP, DOWN, LEFT, and RIGHT
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:

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

We have to fill out the Q-Table using the Q function Q(s_t, a_t) = r_t + γmax_a Q(s_{t+1} , a) for the above states where γ = 0.99
Since in S_4 we reach at the goal we have initialize all the values as zeros.
Step 1: $s_3(D, A)$ = 1 + 0.99*0 = 1
Step 2: $s_3(R, A)$ = 0 + 0.99*1 = 0.99
Step 3: $s_2(R, A)$ = 1 + 0.99*1 = 1.99
Step 4: $s_2(D, A)$ = 1 + 0.99*1 = 1.99
Step 5: $s_1(D, A)$ = 1 + 0.99*1.99 = 2.97
Step 6: $s_1(R, A)$ = 1 + 0.99*1.99 = 2.97
Step 7: $s_1(U, A)$ = 0 + 0.99*2.97 = 2.94
Step 8: $s_0(R, A)$ = 1 + 0.99*2.97 = 3.94
Step 9: $s_0(D, A)$ = 1 + 0.99*2.97 = 3.94
Step 10: $s_0(L, A)$ = 0 + 0.99*3.94 = 3.90
Step 11: $s_0(U, A)$ = 0 + 0.99*3.94 = 3.90
Step 12: $s_1(L, A)$ = -1 + 0.99*3.94 = 2.90
Step 13: $s_2(U, A)$ = -1 + 0.99*2.97 = 1.94
Step 14: $s_2(L, A)$ = -1 + 0.99*2.97 = 1.94
Step 15: $s_3(U, A)$ = -1 + 0.99*1.99 = 0.97
Step 16: $s_3(L, A)$ = -1 + 0.99*1.99 = 0.97

State	Up	Down	Left	Right
0	3.9	3.94	3.9	3.94
1	2.99	2.97	2.9	2.97
2	1.94	1.99	1.94	1.99
3	0.97	1	0.97	0.99
4	0	0	0	0

References

[1] Thomas Simonini. *An introduction to Deep Q-Learning*. Medium, 2018.

[2] *A Beginner's Guide to Deep Reinforcement Learning*. medium, 2014.

[3] Keras. Keras documentation. keras.io, 2015.