CSE 574 - Project 4: Tom and Jerry in Reinforcement learning

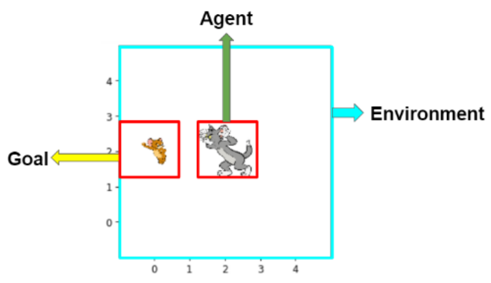
Baskar Adyar Krishnan
Person 50291475
UBIT: BASKARAD

Abstract

This project is to implement Reinforcement learning in a mini Tom and Jerry game. Eventually, learn about Reinforcement learning and its hyper-parameters like epsilon max/min,gamma, etc. and its influence in the total time of training and the mean reward.

1 Introduction

The process is to teach the agent to navigate in the grid-world environment. In our modeled game the task for Tom (an agent) is to find the shortest path to Jerry (a goal), given that the initial positions of Tom and Jerry are deterministic. 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.



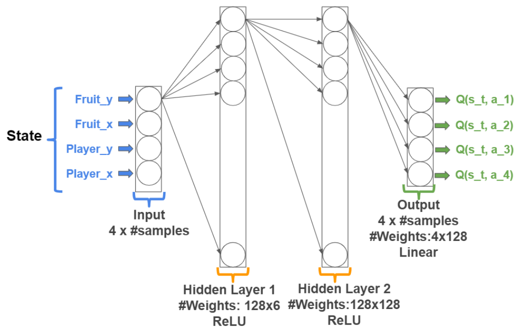
2 Coding Tasks

2.1 The Neural Network:

For this task we going to take a 3 layer neural network using keras library. In this task the Neural Network is setup in the class "Brain". The model's structure is: LINEAR -> RELU -> LINEAR -> RELU -> LINEAR. Activation function for the first and second hidden layers is 'relu'and its 'linear' for the final layer.

2.1.1 Code snippet:

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



2.2 Exponential-decay formula for epsilon :

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.

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

where $\epsilon_{min}, \epsilon_{max} \in [0, 1]$
λ - hyperparameter for epsilon
$|S|$ - total number of steps

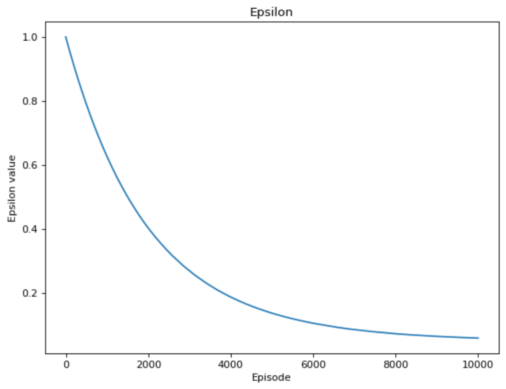
2.2.1 Code snippet:

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

2.2.2 Explanation:

We want our agent to explore as much as possible so we must have a higher epsilon at the beginning. As as agent learns more, in other words, as the agent gains more experience, we want our agent to act less randomly and predict based on the experience gained. For that we have to decrease the epsilon value as the training is done.

This process of decreasing the value of epsilon for every iteration of the training is done by the exponential-decay formula. Below is the graph of epsilon as the number of episode increases.



2.3 Implementation of Q-function:

Q - function gives the immediate reward (for taking that action) + discounted reward (for the future action) for all the possible actions that can be taken for a given state. Using Q - funtion/table we can

Deterministic policy (π) is a function that maps states to actions:

$$\pi(s) = a$$

Return (G_t):

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Value function $(V_\pi(s))$:

$$\begin{aligned} V_\pi(s) &\doteq \mathbb{E}_\pi[G_t | S_t = s] \\ &= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s\right], \text{for all } s \in S \end{aligned}$$

Action-value function $(Q_\pi(s, a))$ - how good to take an action at a particular state:

$$\begin{aligned} Q_\pi(s, a) &\doteq \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \\ &= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a\right] \end{aligned}$$

Objective is to find such a policy, that returns max Q-value.

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

find such a policy that returns max q-values. In other words, we can choose the action to perform in a state by analyzing the rewards we would get by taking that action with the help of Q - function.

2.3.1 Code snippet:

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

2.4 Analysing the Hyper-parameters:

2.4.1 Min/max epsilon:

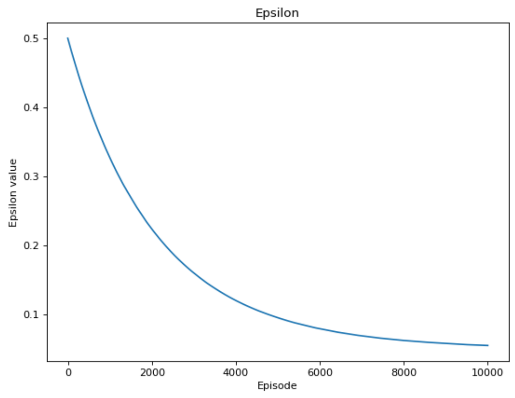


Figure 1: Epsilon decay

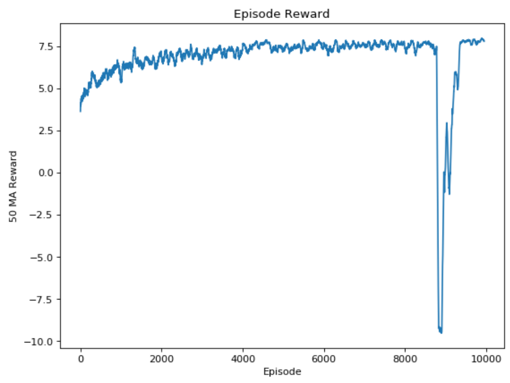


Figure 2: Rewards

Figure 3: with Max epsilon as 0.5

2.4.2 Number of episodes:

By decreasing the number of episodes, we are decreasing the number of iterations which in turn affects the model by stopping the iteration even before it gets to a optimal solution. Similarly, by increasing the number of episodes, the model doesn't seem to learn more than what it done for lesser number of iterations say 8000 iterations. Beyond 8000 episodes the rewards graph gets optimized, so it is waste of time to run for more than 10000 episodes for this agent in this environment.

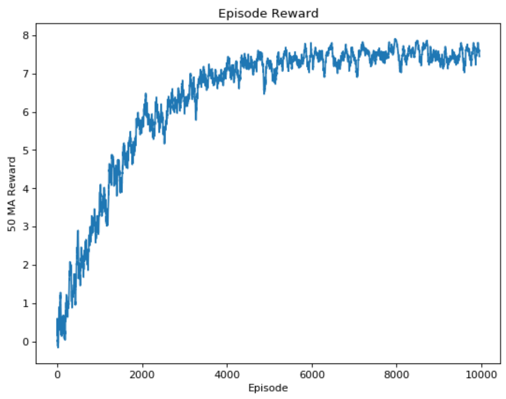


Figure 4: Rewards optimized after certain episodes

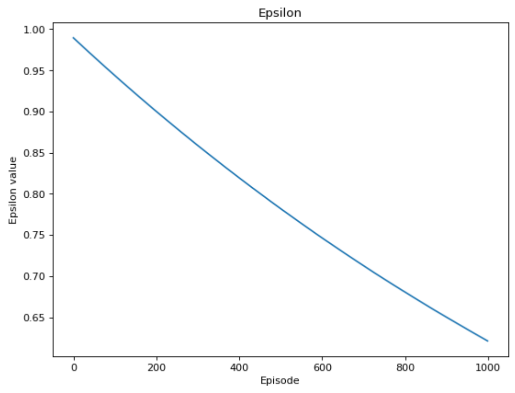


Figure 5: Epsilon not decayed completely

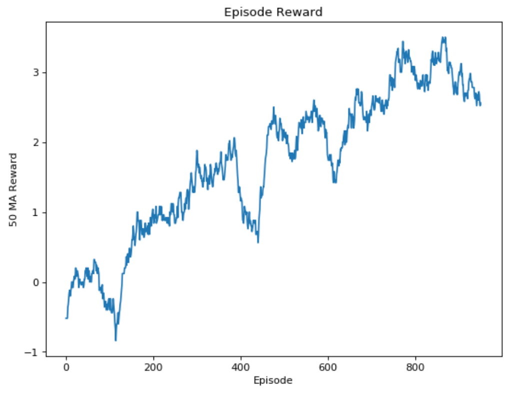


Figure 6: Max reward not obtained

Figure 7: With less number of iterations

2.4.3 Discount factor (Gamma):

The discount factor is a measure of how far ahead in time the algorithm looks. To prioritize rewards in the distant future, keep the value closer to one. A discount factor closer to zero on the other hand indicates that only rewards in the immediate future are being considered, implying a shallow look ahead.

Model is tested for various discount factors ranging from 0.1 to .99. The metrics of the models doesn't seem to vary a lot with the change in the discount factor. This is because this environment is very small. In a bigger environment the gamma values places a role in affecting the rewards and other performance of the agent.

3 Writing tasks:

3.1 What happens if the agent always chooses the action that maximizes the Q-value:

In order to answer this question we have to understand that in any reinforcement learning problem maintaining the balance between the exploration and exploitation is very important.

Exploitation: This phase is used to make the best decision in the given current information.
Exploration: phase is nothing but to gather more information/experience. This experience has to be used in the exploitation phase.

If we have the agent to always choose the action that maximizes the q-values it will be more like a **greedy approach**. It may be good short-term strategy but it is not good for long-term goals as this will end up in the local maxima without even exploring new possibilities in the environment and even it has the higher chance of failure of finding the optimal policy. Thus, we should not always choose the action that maximized the Q-values, but **we have to establish a balance between exploration and exploitation.**

3.1.1 Using Exponential-decay of epsilon:

One way of the most popular ways to maintain the balance between the exploration and exploitation is using "epsilon" and allowing it to decay exponentially. Epsilon is the factor that determines the exploration rate of the agent. If the epsilon is more then there will be more exploration, and exploitation will be more if the epsilon is less.

Initially we want out agent to explore more, so we will start with a higher epsilon value and as the agent increases the experience, the epsilon values is decreased. As discussed above, we have implemented the exponential decay of epsilon in the above task.

3.1.2 Using softmax:

A softmax operator applied to a set of values acts somewhat like the maximization function and somewhat like an average. In sequential decision making, softmax is often used in settings to stop an agent being greedy. The Boltzmann softmax operator is the most commonly used softmax operator. The Boltzmann distribution is given by:

$$\pi(a|s) = \frac{e^{\frac{Q(s,a)}{\tau}}}{e^{\sum_{a' \in \mathcal{A}} \frac{Q(s,a')}{\tau}}}$$

τ is a **"computational" temperature**:

- $\tau \to \infty$: $P = \frac{1}{|\mathcal{A}|}$
- $\tau \to 0$: greedy

3.2 Task 2:

The task to fill the Q-table with manual calculations is attached below:

4 Result and Inference

1. The coding part that was coded is explained along with code snippets.
2. Role of Epsilon is to maintain the balance between the exploration and exploitation.
3. The neural network code is to define the layers in the neural network using keras library
4. The role of Q-value code defines the formula for filling the Q-table.
5. The agent was able to learn mostly around 8000 episodes.
6. The above mentioned hyper parameters are tested my changing the values and the ideal set of hyper-parameters that would give better total time of training and the mean reward would be number of episodes = 8000, max epsilon = 1, min epsilon = 0.1 and gamma = 0.99
7. The given coding task and both the writing tasks are completed.

Initial table

	Action			
States	Up	Down	Left	Right
s0	0		0	
s1	0			
s2				
s3		1		0
s4	0	0	0	0

It is all zeros in s4 because our final destination is s4
Let us fill the table will all the actions possible and calculate the rewards

Gamma = 0.99

Q(s_t,a_t) = rew_t + gamma * max(Q(s_t+1 , a)

Sample calculation:
The possible action for s3 is up and left

Q(s3,up) = -1 + 0.99 * max(s2,a)

Now we have to calculate all the cells in the row s2 to find the max.
We continue the calculation from bottom to top for all the states, we get

	Action			
States	Up	Down	Left	Right
s0	?	3.9403	?	3.9403
s1	?	2.9701	2.9008	2.9701
s2	1.9403	1.99	1.9403	1.99
s3	0.9701	1	0.9701	?
s4	0	0	0	0

Now solving for the cells that are marked with question mark with the same formula stated abov

Q(s1,up) = 0 + 0.99 * max(Q(s1,a) = 0.99 * 2 = 2.9404

Now we get the final table as

	Action			
States	Up	Down	Left	Right
s0	3.9008	3.9403	3.9008	3.9403
s1	2.9404	2.9701	2.9008	2.9701
s2	1.9403	1.99	1.9403	1.99
s3	0.9701	1	0.9701	0.99
s4	0	0	0	0