PROJECT 4 : Tom and Jerry in Reinforcement Learning

Krishna Sehgal
ksehgal | 50291124

1. Introduction

The project combines reinforcement learning and deep learning. Our task is to teach the agent to navigate in the grid-world environment. We have modeled Tom and Jerry cartoon, where Tom, a cat, is chasing Jerry, a mouse. 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. To solve the problem, we would apply deep reinforcement learning algorithm - DQN (Deep Q-Network) using Neural Networks.

2. Software Used

- Python 3.6
- Google Colaboratory

3. Libraries Used

- **random** - generates pseudo-random numbers
- **math** - provides access to the a big variety of mathematical functions
- **time** - will be used to track how much time each computation takes
- **numpy** - is the main package for scientific computing with Python
- **keras** - is a high-level neural networks API, we will use to biuild a neural network for our agent
- **matplotlib** - is a plotting library
- **IPython** - is an interactive shell, that will help us to display our framework

4. Hyper-Parameters

- Epsilon
- Number of Episodes
- Gamma
- Batch Size

- Lambda

5. Terminologies

5.1 Reinforcement Learning

Reinforcement learning is the process of running the agent through sequences of state-action pairs, observing the rewards that result, and adapting the predictions of the Q function to those rewards until it accurately predicts the best path for the agent to take. That prediction is known as a policy. It can be understood using the concepts of agents, environments, states, actions and rewards.

- **Agent**: An **agent** takes actions; for example, a drone making a delivery, or Super Mario navigating a video game. The algorithm is the agent. Agents are functions that transform the new state and reward into the next action.

- **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. for example, In video games, the list might include running right or left, jumping high or low, crouching or standing still.

- **Discount Factor**: The **discount factor** is multiplied by future rewards as discovered by the agent in order to dampen thse rewards' effect on the agent's choice of action. It is designed to make future rewards worth less than immediate rewards; i.e. it enforces a kind of short-term hedonism in the agent expressed with gamma: γ. A discount factor of 1 would make future rewards worth just as much as immediate rewards.

- **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.

- **State (S)**: A **state** is a concrete and immediate situation in which the agent finds itself; i.e. a specific place and moment. From any given state, an agent sends output in the form of actions to the environment, and the environment returns the agent's new state as well as rewards, if there are any.

- **Reward (R)**: A **reward** is the feedback by which we measure the success or failure of an agent's actions. Rewards can be immediate or delayed. They effectively evaluate the agent's action.

- **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.

$$a = \pi(s)$$

- **Value (V)**: The expected long-term return with discount, as opposed to the short-term reward R. Vπ(s) is defined as the expected long-term return of the current state under policy π. We discount rewards, or lower their estimated value, the further into the future they occur.

- **Q-value(Q)**: **Q-value** is similar to Value, except that it takes an extra parameter, the current action a. Qπ(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.

$$Q(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

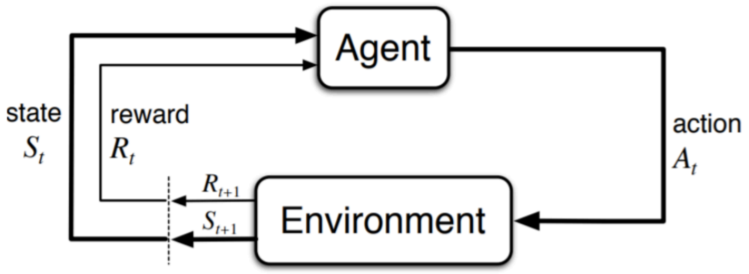
- **Trajectory**: A sequence of states and actions that influence those states.

5.2 Markov Decision Process

A Markov decision process (MDP) is a Markov reward process with decisions. It is an environment in which all states are Markov.

A Markov Decision Process is a tuple ⟨S, A, P, R, γ⟩ S is a finite set of states

- A is a finite set of actions
- P is a state transition probability matrix,
- $P^a_{ss'}$ =P[S_{t+1}=s′|S_t=s,A_t=a]
- R is a reward function, Ras = E[R_t+1 | S_t= s,A_t= a]
- γ is a discount factor γ ∈ [0, 1].



6. Questions

6.1 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, a suitable approach is for the agent to first explore the entire environment so that he gets to know actions that lead to rewards that may not be achieved instantly but at a later final state rather than choosing actions that maximizes the Q-value at that instant.

This approach where agent always chooses the actions that maximizes Q-value for the next step is a greedy approach. It works well only after the agent has complete idea of the environment else it results in a downfall this is because it does not allow agent to explore all the states rather looks for the immediate q value maxima, which generally let agent to follow a single path, which might have local maxima reward values, but there could be some other paths for which there might not be an immediate reward but a reward at a later state resulting in a much higher total reward as compared to the reward achieved through greedy approach which chooses path based on the action selection of maximize immediate Q values.

Two ways to force the agent to explore are :

1. ε-Greedy Approach
2. Boltzmann Approach

1. ε-Greedy

ε-greedy exploration is one of the most used exploration strategies. It uses 0 ≤ ε ≤ 1 as parameter of exploration to decide which action to perform using Qt (st , a). The agent chooses the action with the highest Q-value in the current state with probability 1 – ε, and a random action otherwise. A larger value for ε means more exploration actions are selected by the agent. Randomness is necessary for an agent navigating through a stochastic maze to learn the optimal policy.

2. Boltzmann Approach

Boltzmann exploration ideally like to exploit all the information present in the estimated Q-values produced by our network. Instead of always taking the optimal action, or taking a random action, this approach involves choosing an action with weighted probabilities. To accomplish this we use a soft-max over the networks estimates of value for each action. In this case the action which the agent estimates to be optimal is most likely to be chosen. The biggest advantage over e-greedy is that information about likely value of the other actions can also be taken into consideration. If there are 4 actions available to an agent, in e-greedy the 3 actions estimated to be non-optimal are all considered equally, but in Boltzmann exploration they are weighed by their relative value. This way the agent can ignore actions which it estimates to be largely sub-optimal and give more attention to potentially promising, but not necessarily ideal actions.

$$\pi(s_t, a) = \frac{e^{Q_t(s_t,a)/T}}{\sum_{i=1}^{m} e^{Q_t(s_t,a^i)/T}}$$

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

ACTIONS				
STATE	UP	DOWN	LEFT	RIGHT
0	3.90	3.94	3.94	3.90
1	2.94	2.97	2.97	2.90
2	1.94	1.99	1.99	1.94
3	0.97	1	0.99	0.97

ACTIONS				
4	0	0	0	0

Looking at each state individually,

S_0	S_{1b}	T_b
S_{1a}	S_2	S_{3b}
T_a	S_{3a}	Goal

S_{1a} and S_{1b} is same due to symmetry and therefore considered as S_1 only.
T_a and T_b are same due to symmetry and therefore considered as T only.
S_{3a} and S_{3b} is same due to symmetry and therefore considered as S_3 only.

Now for state S_4, going to any direction will yield zero reward and therefore, we fill the above table with zeros.

Now for state S_{3b},
Action UP

$Q_{t3} = -1 + 0.99 * \max Q_{tp} = -1 + 0.99*1.99 = -1 + 1.97 = 0.97$

Action DOWN

$Q_{t3} = 1 + 0.99 * \max Q_{tG} = 1$

Action LEFT

$Q_{t3} = -1 + 0.99 * \max Q_{t2} = - 1 + 0.99*1.99 = 0.97$

Action RIGHT

$Q_{t3} = 0 + 0.99 * \max Q_{t3} = 0.99$

Now for state S_2,
Action UP

$Q_{t2}= -1 + 0.99 * \max Q_{t1} = -1 + 0.99 *2.97 = 1.94$

Action DOWN

Q_{t2} = 1 + 0.99 * max Q_{t3} = 1.99

Action RIGHT

Q_{t2} = 1 + 0.99 * max Q_{t3} = 1.99

Action LEFT

Q_{t2} = -1 + 0.99 * max Q_{t1} = -1 + 0.99 *2.97 = 1.94

Now for state S_{1b},
Action UP

Q_{t1} = 0 + 0.99 * max Q_{t1} = 0.99 *2.97 = 2.94

Action DOWN

Q_{t1} = 1 + 0.99 * max Q_{t2} = 1+ 0.99*1.99 = 1+1.97 = 2.97

Action LEFT

Q_{t1} = -1 + 0.99 * max Q_{t0} = -1 + .99*3.94 = 2.90

Action RIGHT

Q_{t1} = 1 + 0.99 * max Q_{tP} = 1+ 0.99*1.99 = 1+1.97 = 2.97

Now for state T_b,
Action UP

Q_{tP} = 0 + 0.99 * max Q_{tP}= 1.97

Action DOWN

Q_{tP} = 1 + 0.99 * max Q_{t3} = 1.99

Action RIGHT

Q_{tP} = 0 + 0.99 * max Q_{tP} = 1.97

Action LEFT

Q_{tP} = -1 + 0.99 * max Q_{t1} = -1 + 0.99*2.97 = 1.94

Now for state S0,
Action UP

Q_{t0} = 0 + 0.99 * max Q_{t0} = 3.90

Action DOWN

Q_{t0} = 1 + 0.99 * max Q_{t1} = 1+ 0.99*2.97 = 3.94

Action RIGHT

Q_{t0} = 1 + 0.99 * max Q_{t1} =1+ 0.99*2.97 = 3.94

Action LEFT

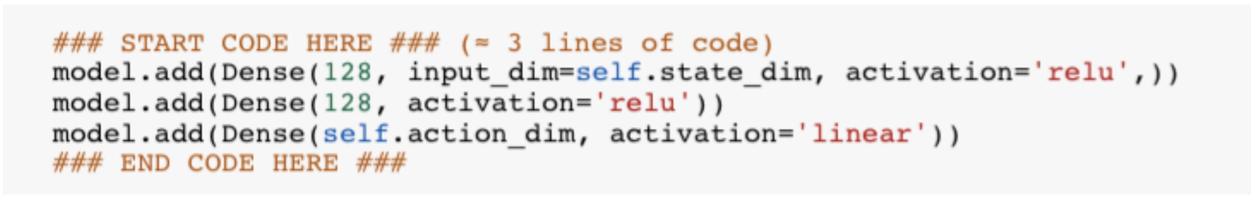
Q_{t0} = 0 + 0.99 * max Q_{t0} = 3.90

6.3 Code Snippets and Parts implemented

I have implemented Neural Network, Calculation of Epsilon and Implementation of Q - Function.

6.3.1 3-layer Neural Network, using Keras library

- A Three-layer neural network with two hidden layers
- The model's structure is: Linear -> Relu -> Linear -> Relu -> Linear
- Activation function for the first and second hidden layers is ‘Relu’.
- Activation function for the final layer is ‘linear’ (returning real values).
- Input dimensions for the first hidden layer equals to the size of your observation space(state_dim).
- Number of hidden nodes is 128.
- Number of the output should be the same as the size of the action space (action_dim).



```
### 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 ###
```

6.3.2 Epsilon Implementation

Exponential-decay formula for epsilon:

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

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

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

6.3.3 Q Function Implementation

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

```
### START CODE HERE ### (≈ 4 line of code)
if st_next is not None:
  t[act] = rew + self.gamma * np.amax(q_vals[i])
else:
  t[act]=rew
### END CODE HERE ###
```

6.4 Role in training the agent

Neural Network
Neural networks are the agent that learns to map state-action pairs to rewards. Like all neural networks, they use coefficients to approximate the function relating inputs to outputs, and their learning consists to finding the right coefficients, or weights, by iteratively adjusting those weights along gradients that promise less error. At the beginning of reinforcement learning, the neural network coefficients may be initialized stochastically, or randomly. Using feedback from the environment, the neural net can use the difference between its expected reward and the ground-truth reward to adjust its weights and improve its interpretation of state-action pairs.

Epsilon
The probability of taking random action is known as Epsilon. This means that when an action is selected in training, it is either chosen as the action with the highest q-value, or a random action. Choosing between these two is random and based on the value of epsilon, and epsilon is annealed during training such that initially, lots of random actions are taken (exploration), but as training progresses, lots of actions with the maximum q-values are taken (exploitation). Then, during testing, they also use this epsilon-greedy method, but with epsilon at a very low value, such that there is a strong bias towards exploitation over exploration, favoring choosing the action with the highest q-value over a random action. However, random actions are still sometimes chosen (5 % of the time).

Q - Function

Q maps state-action pairs to the highest combination of immediate reward with all future rewards that might be harvested by later actions in the trajectory. Having assigned values to the expected rewards, the Q function simply selects the state-action pair with the highest so-called Q value.

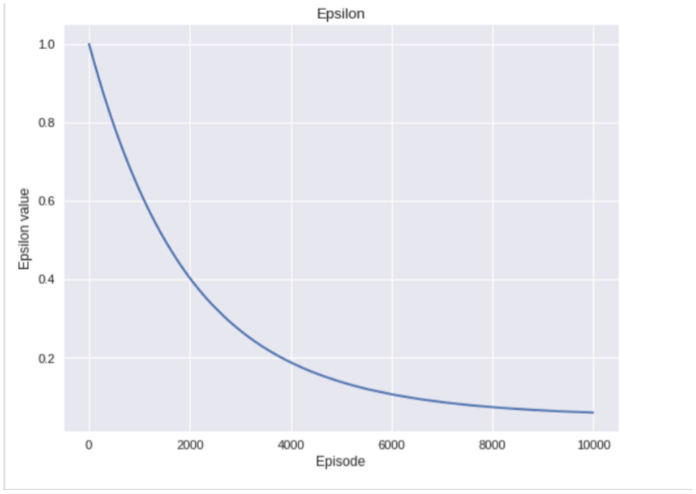
6.5 Can these snippets be improved and how it will influence in training the agent ?

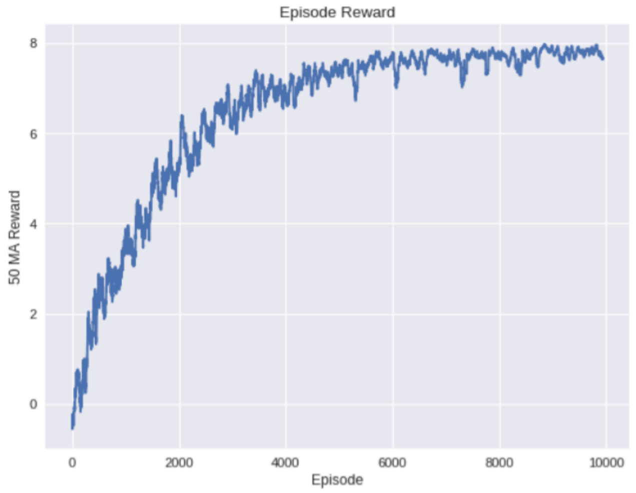
These snippets can be improved by tuning the Hyper-parameters such as Epsilon, Number of Episodes, Gamma, Batch Size, Lambda.

We can improve our model by using different activation functions such as Sigmoid, Relu and other functions but its better to use Linear at the end layer a s there are immediate rewards.

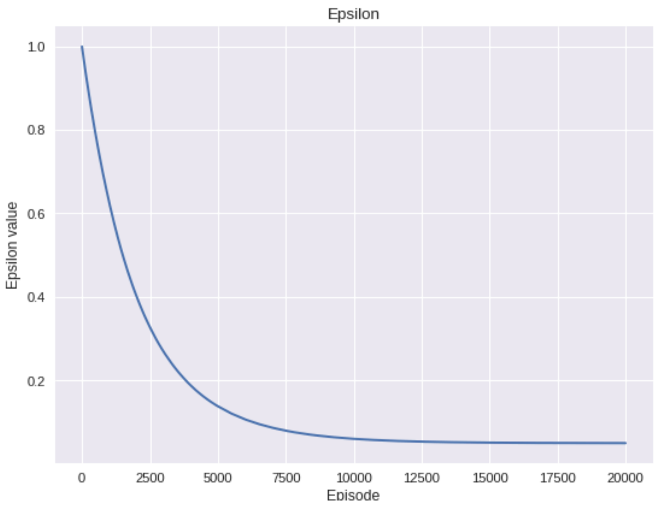
Hyper-Parameter Tuning

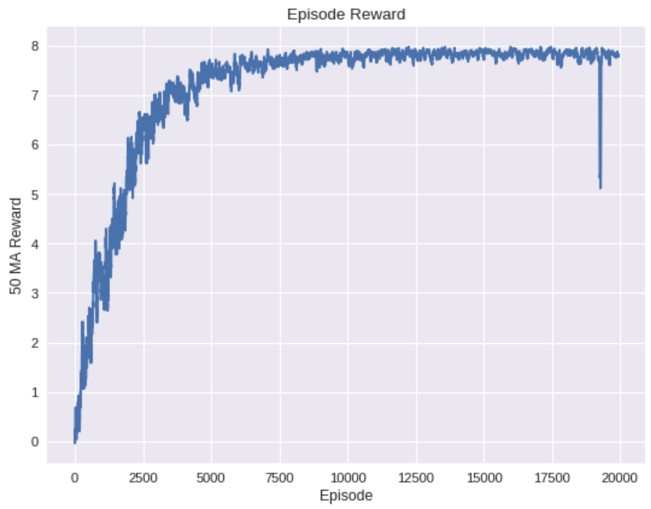
1. MAX_EPSILON = 1
   MIN_EPSILON = 0.05
   LAMBDA = 0.00005
   num_episodes = 10000



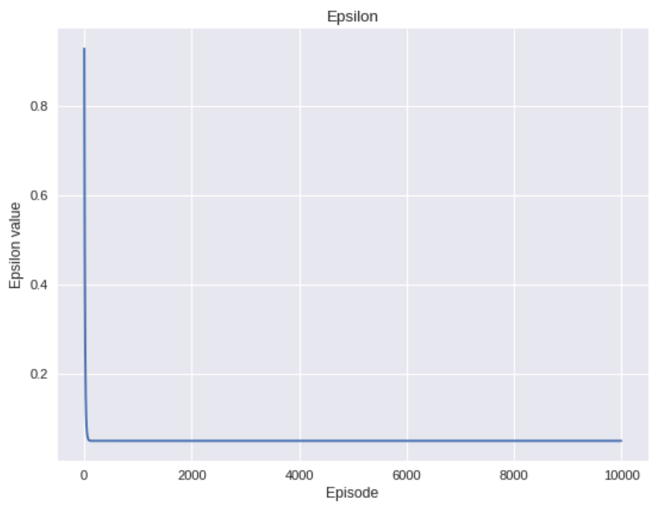


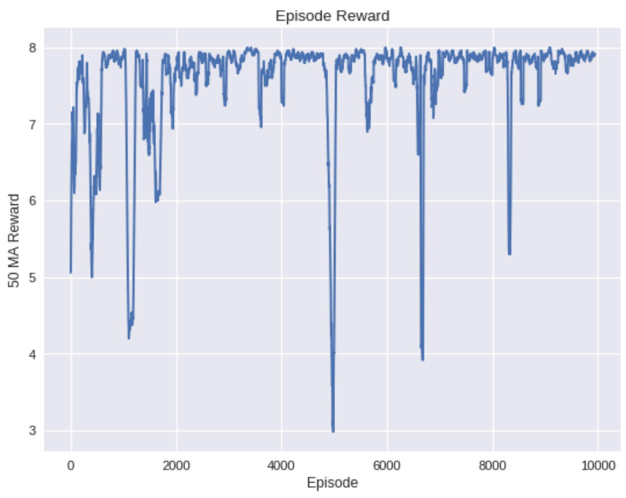
2. MAX_EPSILON = 1
   MIN_EPSILON = 0.05
   LAMBDA = 0.00005
   num_episodes = 20000





3. MAX_EPSILON = 1
   MIN_EPSILON = 0.05
   LAMBDA = 0.008
   num_episodes = 10000





6.6 How quickly agent was able to learn

```
Episode 6000
Time Elapsed: 9.56 mins
Epsilon 0.10682986527245786
Last Episode Reward: 8
Episode Reward Rolling Mean: 5.575156753092696
```

References

- https://skymind.ai/wiki/deep-reinforcement-learning#domain
- http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html
- https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-7-action-selection-strategies-for-exploration-d3a97b7cceaf
- https://www.textsheet.com/retort
- http://www.ai.rug.nl/~mwiering/GROUP/ARTICLES/Exploration_QLearning.pdf
- https://stats.stackexchange.com/questions/270618/why-does-q-learning-use-epsilon-greedy-during-testing