University at Buffalo

CSE 574: Introduction to Machine Learning
(Fall 2018)

Tom and Jerry in Reinforcement Learning

Author: Ameen M Khan
Person Number: 50288968

Problem Statement Implement a deep reinforcement learning algorithm, DQN (Deep Q-Network), to teach the agent to navigate the grid-world environment to reach the goal. In the provided problem statement, the environment is modeled such that Tom(agent) is tasked to find the shortest path to Jerry(goal). In the second part of the project, we answer individual questions (Writing Task) regarding reinforcement learning.

1 Methodology

In this project, we implement a neural network in Keras for the deep learning section and provide portions of implementation of custom environment i.e. exponential decay formula for epsilon and Q-Function and further hyperparameter tuning on epsilon, number of episodes, gamma, etc, and their further influence on the total time of training and the mean reward.

1.1 Reinforcement Learning and Markov Decision Process

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. Basic reinforcement is modeled as a Markov decision process a 5-tuple (S, A, P_a , R_a , γ), where S is a finite set of states, A is a finite set of actions, P_a is the probability that action a in state s at time t will lead to state s' at time t+1, R_a is the immediate reward (or expected immediate reward) received after transitioning from state s to state s' due to action a, γ ϵ [0, 1) is the discount factor, which represents the difference in importance between future rewards and present rewards. an environment, which handles state and reward, and an agent, which decides which action to take given a particular state. An environment starts with some initial state s_0 , which is passed to the agent. The agent passes an action a_0 , based on the state s_0 , back to the environment. The environment reacts to the action, then passes the next state, s_1 , along with the resulting reward for taking the action, r_0, back to the agent. Reinforcement Learning with MDP involves following definitions:

- **Policy** : π(s) = a
- **Return Function** $G_t = \sum \gamma^k * R_{t+k+1}$
- **Value Function** $(V_\pi)(s) = E_\pi[\sum \gamma^k * R_{t+k+1} | S_t = s, A_t = a]$, for all $s\epsilon$ S.
- **Action Value Function** (how good to take an action at a particular state) $Q_\pi(s, a) = E_\pi[\sum \gamma^k * R_{t+k+1} | S_t = s, A_t = a]$.

1.2 Implementation Details

1.2.1 Deep Neural Network (Coding Task 1)

The deep neural network is the place where actual actual model (DQN) is stored and computes the results for each action possible in the given state. The network is made up of an input layer (encoded input data), 2 hidden layer with RELU activation, and one output layer consisting of 4 units with linear activation. Hyperparameter tuning involved making network architecture decisions like selection of number of hidden layers, number of hidden nodes in each layer(125-1024). For our problem statement, I chose to stick to the provided guidelines and used *mean square error* with RMS optimizer.

```
        ### 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'))
```

Figure 1: Implementation of DNN

1.2.2 Exponential-Decay for Epsilon (Coding Task 2)

The task involves implementing an Exponential-decay for Epsilon *'Exploration Rate'* function which is used by the agent to randomly select its action by a certain percentage, to ensure that in initial stages of training, the agent tries all kinds of things before it starts seeing a pattern. The formula for exponential decay for decreasing epsilon:

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

where λ is the hyperparameter of epsilon. The above formula ensures slowly move from exploration (random actions) towards exploiting the environment.

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

Figure 2: Implementation of exponential decay of Epsilon

1.2.3 Q-Function in the Agent class (Coding Task 3)

The task involves implementing *Experience Replay*, which allows the agent to learn from recent observations and from previous observations. This implementation is inside the *replay* method in **Agent** class, which samples a set of observations from the agent's memory (**Memory** class), make a complete observation, and send it to the neural network (**Brain** class) to train it. The observation makes discounted reward for each action which is passed to it as an input, using the following equation :

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

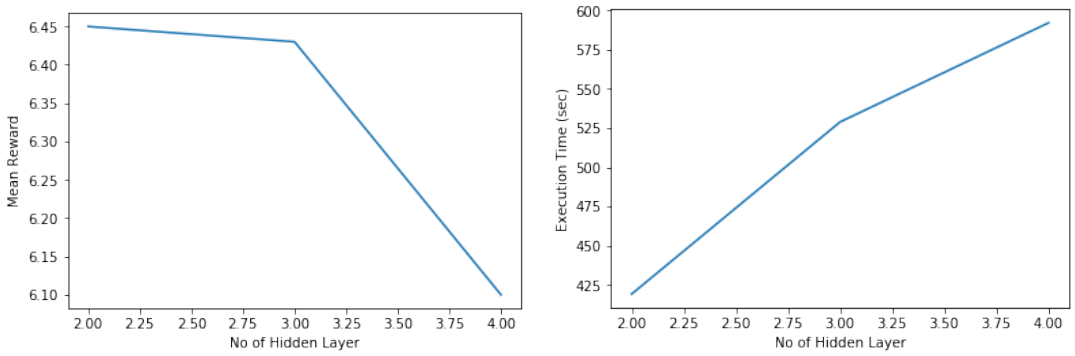
```
        ### START CODE HERE ### (≈ 4 line of code)
        if st_next is None:
            t[act] = rew
        else:
            t[act] = rew + self.gamma*np.amax(t)

        ### END CODE HERE ###
```

Figure 3: Implementation of Q Functions

2 Experiments and Observations

2.1 Tuning Neural Network

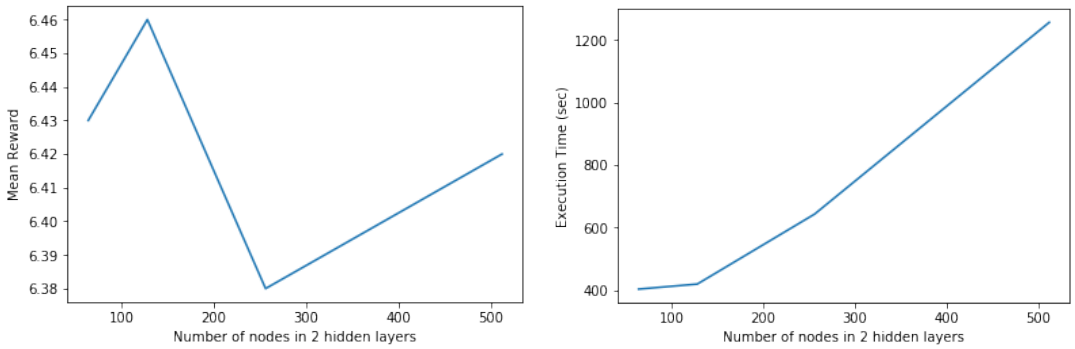
As suggested, I tried increasing the number of hidden layers. With the given 2 hidden layer architecture, the mean reward and elapsed time were 6.45 and 415 seconds respectively, which was the best combination, as indicated by following plots.



(a) Mean Reward vs Hidden Layers (b) Execution Time vs Hidden Layers

Figure 4: Tuning Hidden Layers

When tuning for an optimal architecture choice with different number of node in 2 hidden layer, the initial number of nodes were best, as indicated by following plots:



(a) Mean Reward vs Nodes in Hidden Layers (b) Execution Time vs Nodes in Hidden Layers

Figure 5: Tuning Nodes in Hidden Layers

Experiments were carried to try different activation functions, instead of Relu. I tried Sigmoid, Softmax and Linear in the same 2 hidden layer architecture, unsuccessfully, as indicated by table of execution time and mean reward with different activation functions:

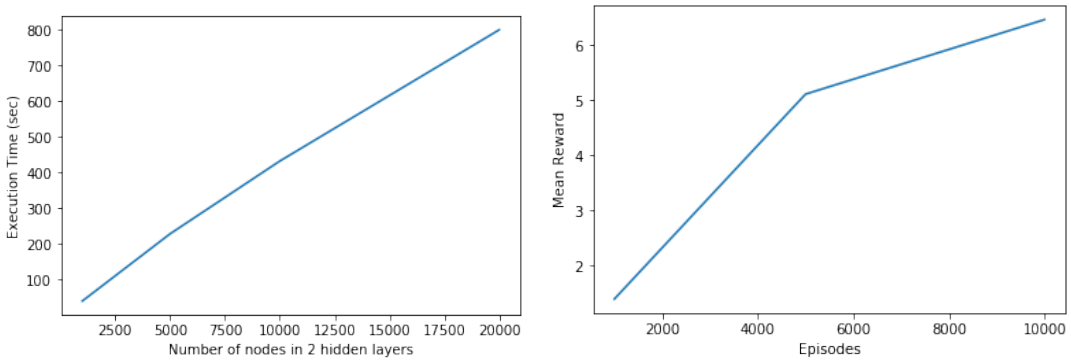
Activation Functions	Execution Time	Mean Reward
RELU	419.25	6.46
SOFTMAX	484.94	5.07
SIGMOID	426.94	6.35
LINEAR	428.19	5.8

2.2 Tuning Epsilon

I tried varying ϵ_{min}, by ranging values from 0.05 to 0.1, which indicate a trend that lower the ϵ_min value, earlier the model will converge, i.e. agent will learn faster with low values of ϵ_min.

2.3 Tuning Number of episodes

Tuning for number of episodes in the MDP in Reinforcement Learning model, 10000 episodes is optimal for training it in a reasonable amount of time, with best possible mean reward.



(a) Mean Reward vs Hidden Layers (b) Execution Time vs Hidden Layers

Figure 6: Tuning Hidden Layers

3 Conclusions

A Deep Q-Network reinforcement agent has been successfully implemented and optional variations of parts of the model have been explored and tuned for to give the following architecture:

- Neural Network : a 3 layer (2 hidden with RELU activation and 128 nodes) is optimal to get the best possible mean reward.
- Exponential Decay of Epsilon : The hyperparameter which decides
- Q-Function

4 References

1. Hands-On Machine Learning with Scikit-Learn and TensorFlow, OReilly Publication
2. Reinforcement Learning, An Introduction (Sutton, Barto).
3. Open AI Gym Documentation.

5 Writing Tasks

5.1 Exploration and Exploitation

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. An important question in reinforcement learning is how does the agent select actions during learning. That is, should the agent trust the learnt values of Q(s, a) enough to select actions based on it, or try other actions hoping this may give it a better reward also known as exploration vs exploitation.

The problem with the exploitation is that the agent will not explore the environment for better outcomes and look for short-term reward which might not be fruitful in the long run and generate a policy which is not globally optimal. In the case in the Coding Task of Tom and Jerry gridworld, the agent(tom) might initially select an action which has high Q-value and repeat this strategy to inadvertently follow a path which will not lead him to his goal(Jerry). Subsequently, in all future episode, the agent will repeat this path because of exploitation and never reach his goal but keep running to the boundaries of the grid environment.

To ensure that the agent tries some exploration, following methods can be used:

- **Epsilon Greedy** : One of the most widely used approach, it selects a values in range of 0,1 and using the value of $Q_t(s_t, a)$ decides which action to perform. This is the approach which has been implemented in the Coding portion of the assignment. A larger value for Epsilon means more exploration actions are selected by the agent. Initial randomness is necessary for an agent exploring a grid to try to learn an optimal policy. After initial exploration, it is necessary for the agent to start exploiting his leanings slowly and keep enhancing his approach with combined with small exploration attempts as we go on by decreasing the value of epsilon. In the project we use exponential decay for epsilon, but alternatively we can also apply linear, quadratic etc, but with experiments latter approaches might not be proved as proved as the exponential ones, for most environments.

- **Softmax (Boltzman) Exploration**: This approach, instead of selecting an action randomly for a set of possible action, uses Boltzmann distribution function to assign a probability $\pi(s_t, a)$ to the actions in order to create a graded function, where $\pi(s_t, a)$ denotes the probability the agent selects action 'a' in state $'s'_t$ and T ≥ 0 is the temperature parameter used in the Boltzmann distribution. Using this type of exploration with intermediate values for T, the agent will most likely selects the best action, but other actions are ranked instead of randomly chosen.

Reference : Comparing Exploration in Q Learning, University of Groningen.

5.2 Q-Table for given states

A Q-Table is lookup table where we calculate the maximum expected future rewards for actions at each state, which will guide us to the best action at each state.

States/Actions	UP	Down	Left	Right
0	3.90	3.94	3.90	3.94
1	2.94	2.97	2.90	2.97
2	1.94	1.99	1.94	1.99
3	0.97	1	0.97	0.99
4	0	0	0	0

$Q(S_3, UP) = -1 + \gamma * max(Q(S_2, a)) = -1 + 0.99(1.99) = 0.97$
$Q(S_3, DOWN) = 1$
$Q(S_3, LEFT) = -1 + \gamma * max(Q(S_2, a)) = -1 + 0.99(1.99) = 0.97$
$Q(S_3, RIGHT) = 0 + \gamma * max(Q(S_3, a)) = 0 + 0.99(1) = 0.99$

$Q(S_2, UP) = -1 + \gamma * max(Q(S_1, a)) = -1 + 0.99(2.97) = 1.94$
$Q(S_2, DOWN) = 1 + \gamma * max(Q(S_3, a)) = 1 + 0.99(1) = 1.99$
$Q(S_3, LEFT) = -1 + \gamma * max(Q(S_1, a)) = -1 + 0.99(2.97) = 1.94$
$Q(S_2, RIGHT) = 0 + \gamma * max(Q(S_3, a)) = 0 + 0.99(1) = 1.99$

$Q(S_1, UP) = 0 + \gamma * max(Q(S_1, a)) = 0 + 0.99(1.99) = 2.94$
$Q(S_1, DOWN) = 1 + \gamma * max(Q(S_2, a)) = 1 + 0.99(1.99) = 2.97$
$Q(S_1, LEFT) = -1 + \gamma * max(Q(S_0, a)) = -1 + 0.99(3.94) = 2.90$
$Q(S_1, RIGHT) = 1 + \gamma * max(Q(S_2, a)) = 1 + 0.99(1.99) = 2.97$

$Q(S_0, UP) = 0 + \gamma * max(Q(S_0, a)) = 0 + 0.99(3.94) = 3.90$
$Q(S_0, DOWN) = 1 + \gamma * max(Q(S_1, a)) = 1 + 0.99(2.97) = 3.94$
$Q(S_0, LEFT) = -1 + \gamma * max(Q(S_0, a)) = 0 + 0.99(3.94) = 3.90$
$Q(S_0, RIGHT) = 1 + \gamma * max(Q(S_1, a)) = 1 + 0.99(2.97) = 3.94$