Project 3: Reinforcement Learning

Krapi Vani
CSE474 Machine Learning
University at Buffalo
krapivan@buffalo.edu

Abstract

This project combines deep learning and reinforcement learning, we have a agent and we have to teach our agent to navigate a grid-world environment, here our agent (Tom) has to find shortest path to goal (Jerry). We use Deep learning and Q-learning for this task. We give our agent reward if he goes closer to goal and punishment if he doesnot and over the time (number of episodes) our agent will learn the pattern since the position of goal is deterministic.

1) Code:

1.1 Task 1 - Build 3-layer neural network using Keras library

```
model = Sequential()

### START CODE HERE ### (≈ 3 lines of code)

model.add(Dense(128, input_dim= state_dim))
model.add(Activation('relu'))

model.add(Dense(action_dim, input_dim=state_dim))
model.add(Activation('relu'))

model.add(Dense(action_dim, input_dim=state_dim))
model.add(Activation('linear'))
```

We use Keras library and build neural network, with 3-layers; 1st layer and 2nd layer has activation function "relu" and 3rd layer has "linear" activation function. We use neural network so as to not loose our data and instead use it to get general experience. We accumulate the experience and get generic idea of how environment behaves, we interact with environment and use neural network to get the general idea.

1.2 Task 2 - Implement exponential-decay formula for epsilon

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)*(math.exp(-1*(self.lamb)*self.steps))

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

We implement the exponential-decay formula for epsilon. At first our agent (Tom) will randomly select an action by certain percentage, called epsilon. This helps agent try all kinds of things before it sees a pattern.

1.3. Task 3 – Implement Q-function

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

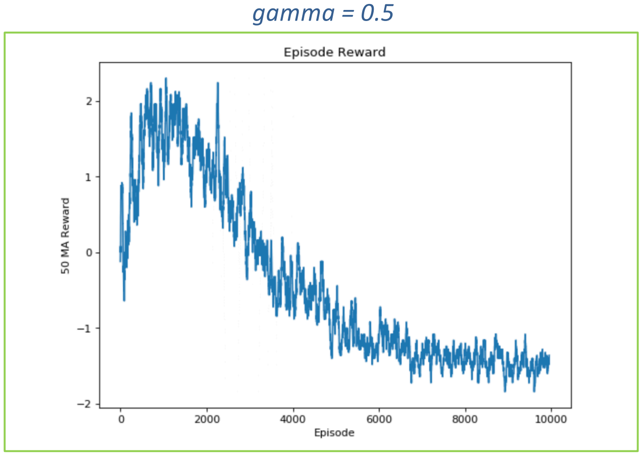
$$Q_t = \begin{cases} r_t, & \text{if episode terminates at step } t+1 \\ r_t + \gamma max_a Q(s_t, a_t; \Theta), & \text{otherwise} \end{cases}$$

Q-function is what helps our agent to learn. Q-function decides whether to give reward, punishment or nothing.

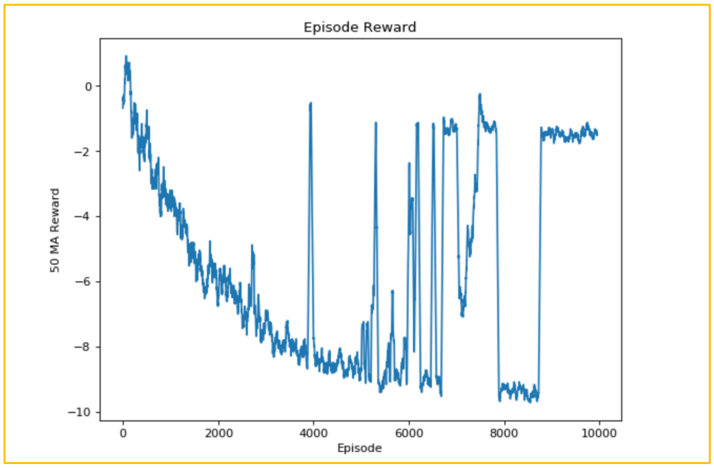
2) Tuning Hyper-parameters:

In the following section we will tune different hyper-parameters to see how they affect mean reward. The default values are as follows: gamma = 0.99, lambda = 0.01, max_epsilon = 1.0 and max_epsilon = 0.1. We will mention the changed parameter the rest will have default value.

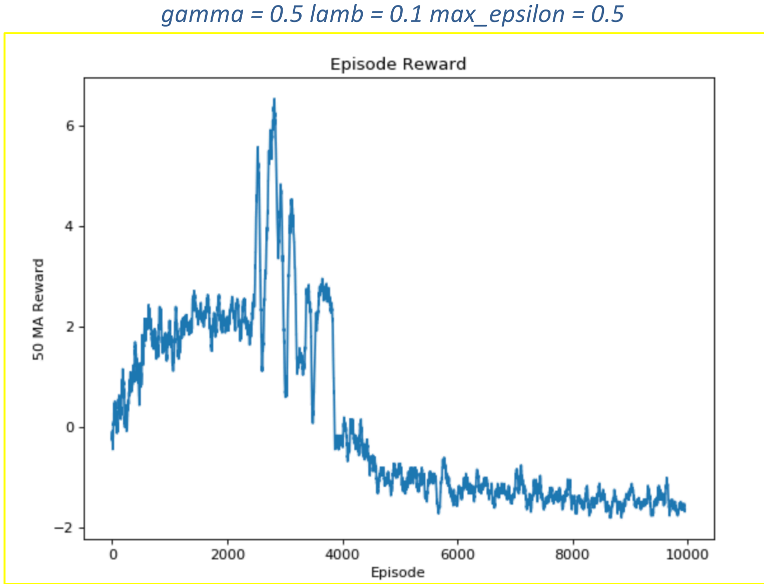
gamma = 0.5



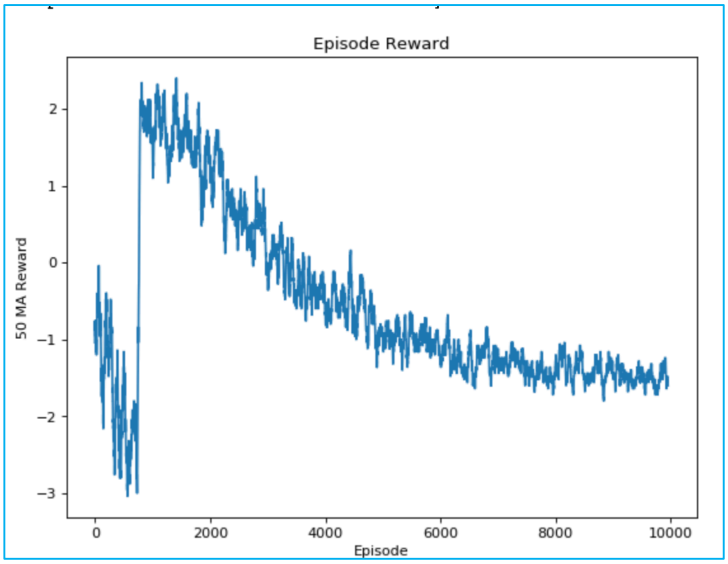
gamma = 0.5 lamb = 0.1



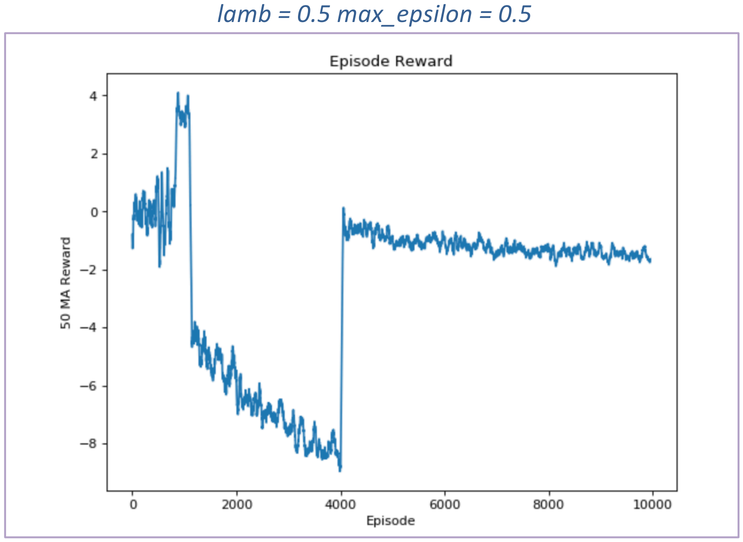
gamma = 0.5 lamb = 0.1 max_epsilon = 0.5



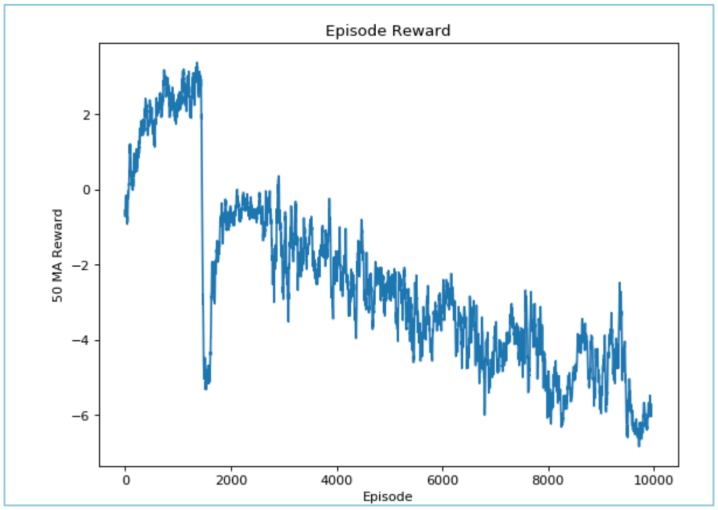
lamb=0.1 max_epsilon = 0.5



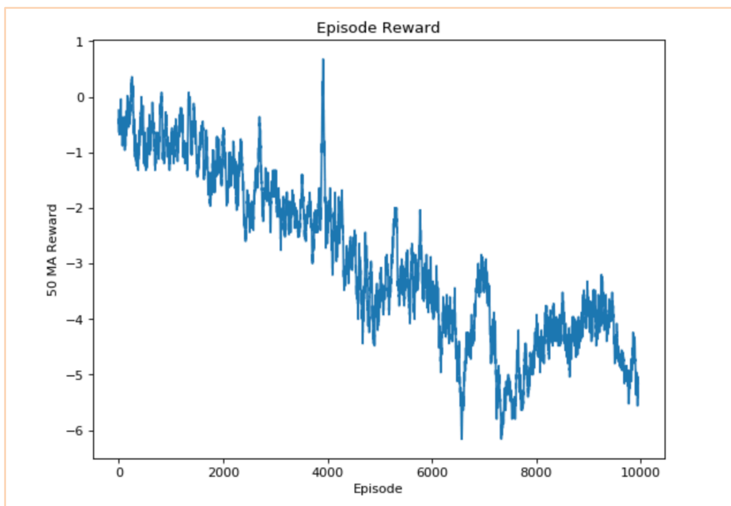
lamb = 0.5 max_epsilon = 0.5



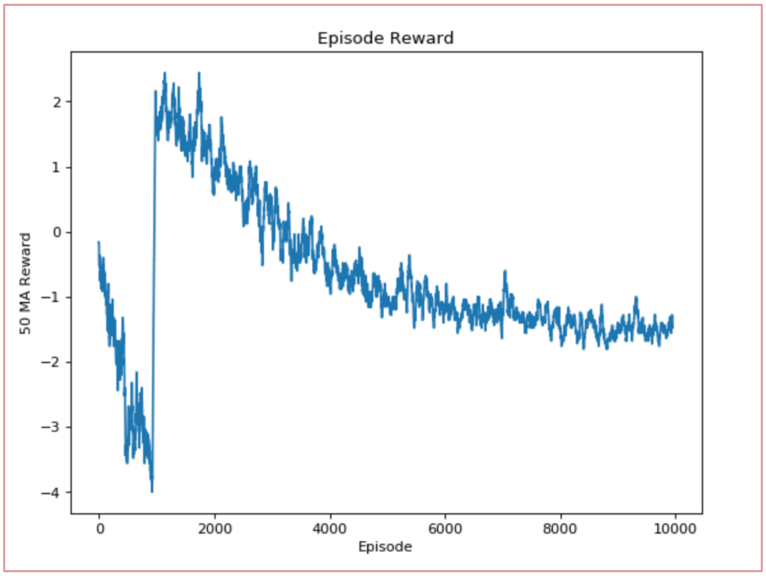
lamb = 0.5



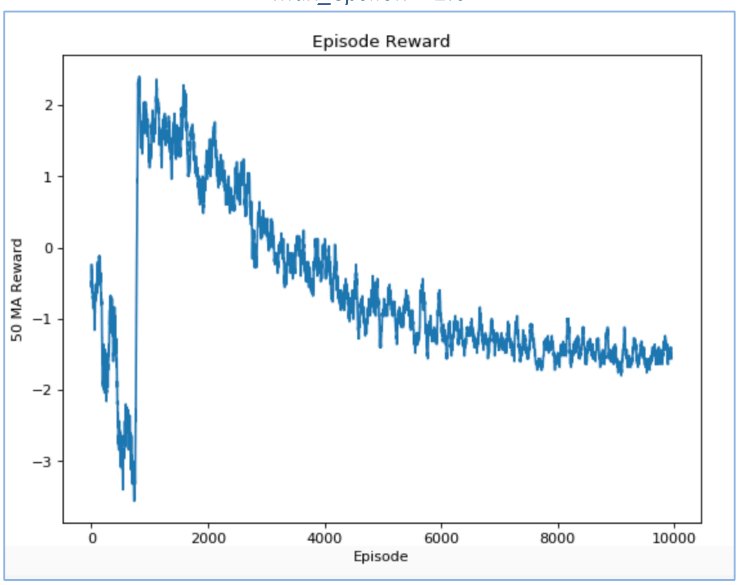
lamb = 0.001



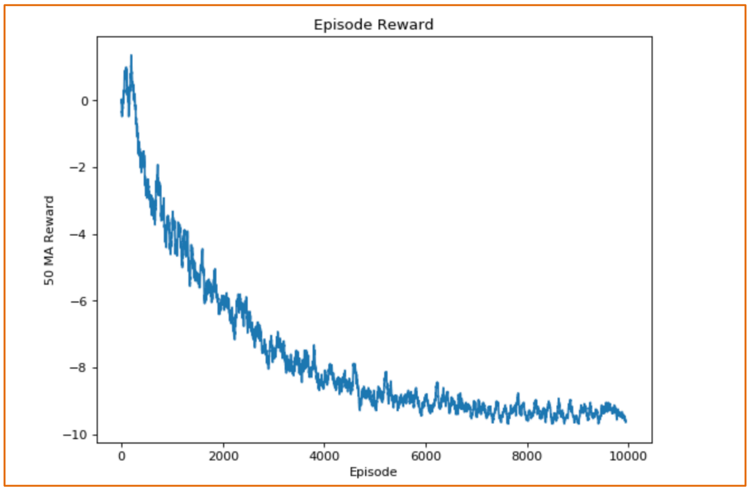
max_epsilon = 2.0



max_epsilon = 2.0



gamma = 1.5



Changing the gamma value to 0.5/1.5 gives lower mean rewards, changing lambda value doesnot change the rewards drastically; but changing max_epsilon (the randomness with which agent can choose action gives somewhat better mean rewards, it gives mean reward ~ 1 stable over the episodes =10,000.

3) Written Task:

3.1 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 agent always chooses the action that maximizes the Q-value then we will not get optimal path, always choosing the said action means robot will not explore the other available options (which might have been optimal). We need our agent to first explore everything before it sees the pattern and get the optimal (minimum) path.

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

State 4 is termination state so Q-values = respective reward values

Up = away from goal = -1
Down = into edge = 0
Left = away from goal = -1
Right = into edge = 0

State 3 is not terminating state so Q-values = reward + gamma*max(Q_next)

Up = away from goal = -1 + 0.99(0) = -1
Down = towards goal = 1 + 0.99(0) = 1
Left = away from goal = -1 + 0.99(0) = -1
Right = into in edge = 0

State 2 is not terminating state so Q-values = reward + gamma*max(Q_next)

Up = away from goal = -1 + 0.99(1) = -0.01
Down = towards goal = 1 + 0.99(1) = 1.99

Left = away from goal = = -1 + 0.99(1) = -0.01
Right = towards goal = 1 + 0.99(1) = 1.99

State 1 is not terminating state so Q-values = reward + gamma*max(Q_next)
Up = into edge = 0
Down = towards goal = 1 + 0.99(1.99) = 2.9701
Left = away from goal = -1 + 0.99(1.99) = 0.9701
Right = towards goal = 1 + 0.99(1.99) = 2.9701

State 0 is not terminating state so Q-values = reward + gamma*max(Q_next)
Up = into edge = 0
Down = towards goal = 1 + 0.99(2.9701) = 3.9403
Left = into edge = 0
Right = towards goal = 1 + 0.99(2.9701) = 3.9403

STATE	UP	DOWN	LEFT	RIGHT
0	0	3.9403	0	3.9403
1	0	2.9701	0.9701	2.9701
2	-0.01	1.99	-0.01	1.99
3	-1	1	-1	0
4	-1	0	-1	0