Project 4: Tom and Jerry in Reinforcement learning

Project 4

Pratik Appaso Vagyani

Graduate Student

Department of Computer Science & Engineering

SUNY Buffalo

pratikap@buffalo.edu

50288741

Overview

Goal of this project is to teach the agent to navigate in the grid-world environment. We have been provided a 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)

Task includes combination reinforcement learning and deep learning.

Machine Learning Methods

Reinforcement Learning

Reinforcement learning is a "less-supervised" version of supervised learning. Generally speaking, our task will be to teach an agent "what to do" in a particular environment. That environment could be as concrete as a real or virtual world—like a robot in a room, or a character in a video game—or as abstract as a sentence parser reading some text. In a fully supervised regime, the agent would be told, through some sort of labels, what the right decision was at every step of its path. In a reinforcement learning regime, the agent is not told what the right decision is; instead, it only knows when something good or bad happens (positive or negative reward, respectively). It's the job of the reinforcement learning

algorithm, then, to translate that perceived reward into the right decisions, given the other facts that it can perceive about its environment

Q-Table

Q-Table is a lookup table where we calculate the maximum expected future rewards for action at each state. Basically, this table will guide us to the best action at each state. There are four numbers of actions move up or down or right or left.So, let's model this environment in our Q-Table.In the Q-Table, the columns are the actions and the rows are the states.

Q-function

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

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. We want our agent to decrese the number of random action, as it goes, so we introduce an exponential-decay epsilon, that eventually will allow our agent to explore the environment.

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

where $\epsilon min, \epsilon max \in [0,1]$

λ - hyperparameter for epsilon

$|S|$ - total number of steps

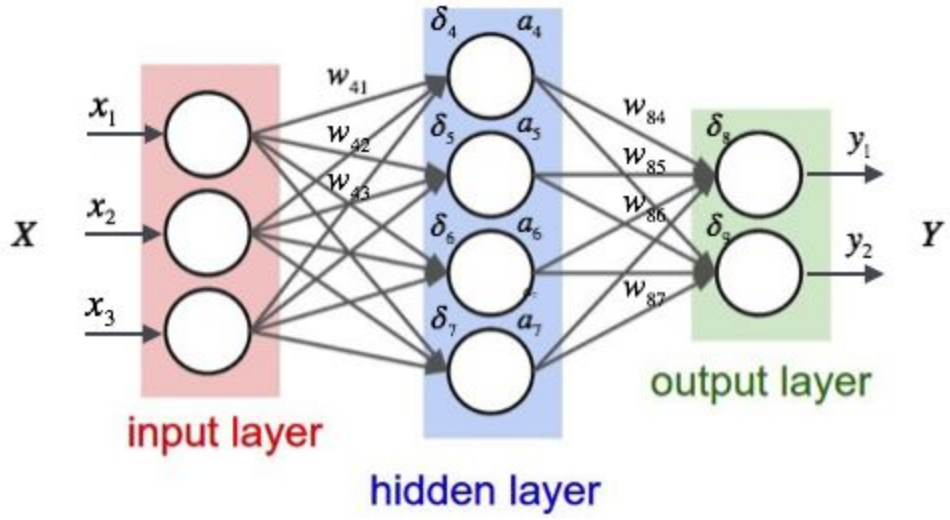
Neural Network

Neural networks are a class of machine learning algorithms used to model complex patterns in datasets using multiple hidden layers and non-linear activation functions. A neural network takes an input, passes it through multiple layers of hidden neurons (mini-functions with unique coefficients that must be learned), and outputs a prediction representing the combined input of all the neurons. Neural networks are trained iteratively

using optimization techniques like gradient descent. After each cycle of training, an error metric is calculated based on the difference between prediction and target. The derivatives of this error metric are calculated and propagated back through the network using a technique called backpropagation. Each neuron's coefficients (weights) are then adjusted relative to how much they contributed to the total error. This process is repeated iteratively until the network error drops below an acceptable threshold.

Input Layer

Holds the data your model will train on. Each neuron in the input layer represents a unique attribute in your dataset.



Hidden Layer

It is between the input and output layers. We can have multiple hidden layer as well. They apply an activation function before passing on the results to next layer. I used 2.

Output Layer

It receives input from last hidden layer and returns an output prediction.

Neuron

A neuron takes a group of weighted inputs, applies an activation function, and returns an output.

Activation function

They are inside neural network and modify input they receive and send further as output. It enables neural network to build complex relation between feature.

I used **relu** activation function in neural network model.

Implementation

1. What parts have you implemented?

   I implemented neural network with 2 hidden layers (128 neuron each) using keras.

   I also implemented exponential-decay formula for epsilon and Q-function.

2. What is their role in training the agent?

   Neural Network: It gets trained in exploration and used to predict best action in exploitation. When epsilon value decrease , then action is predicted by neural network.

   Exponential-decay formula: It decreases epsilon value with help of lambda. Initially for higher values of epsilon, agent will explore as there is more probability of epsilon is more than random.random value. Later for lower value of epsilon, agent will exploit as value of epsilon is more likely to be less than random.random. It uses neural network from brain for prediction.

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

   Q-function: Helps to build Q-table. It calculates max reward for a action from state.

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

3. Can these snippets be improved and how it will influence the training the agent?
   a. We can increase number of neuron or hidden layers to provide better prediction.
4. How quickly your agent were able to learn?

   In different scenario, agent learnt in different time. E.g. For large lambda, Agent learns a path in less time, and follows same. It doesn't try to find best path. For lower lambda, agent explores more states and learns more, while exploitation it chooses best path.

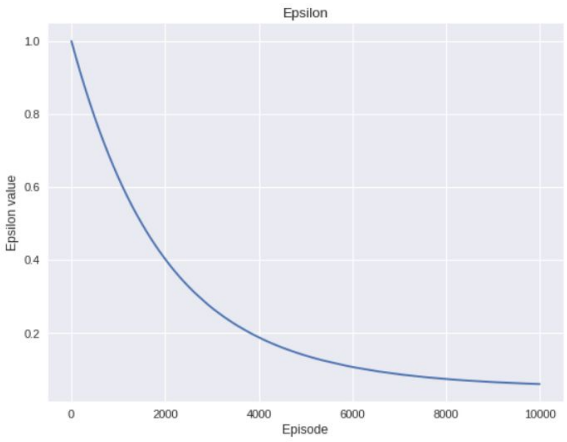
   In first observation, agent learnt in between 2k-3k episodes.

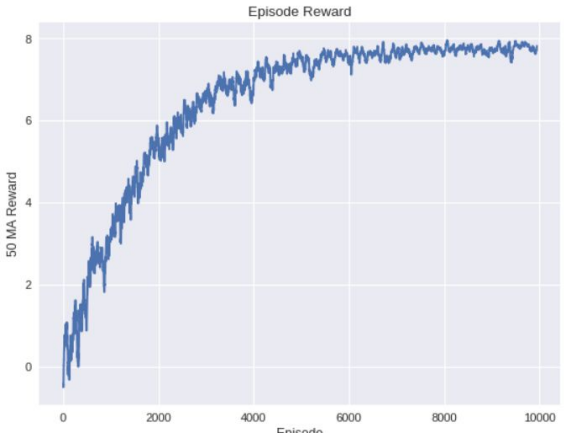
   In second observation, agent learnt in first 1k episodes and exploited the same path as lambda was high and epsilon decrease below 0.1 in first 1k episodes.

Observation:

1.

Episodes	Min Epsilon	gamma	lambda	catch
10k	0.05	0.49	0.00005	6



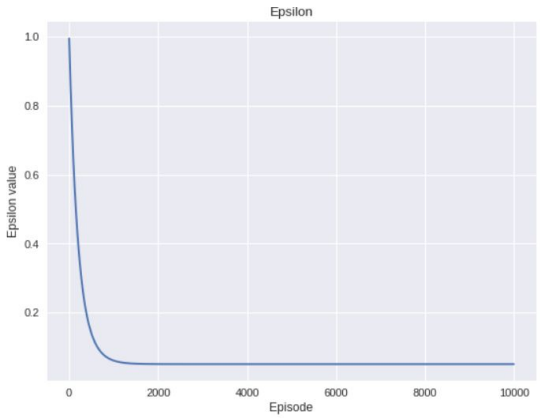


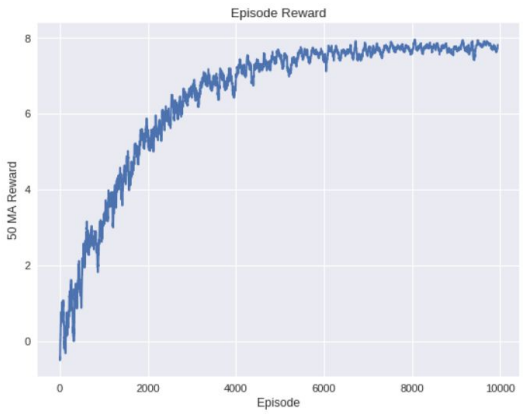
Since Gamma is 0.49, epsilon is decreasing slowly. Hence agent is getting more time to explore the world and model is getting trained on variety of data. When epsilon value decreases and model stars to exploit , model is able to predict better actions/steps. As rewards are increasing,

Epsilon graph is showing slow decay. Reward graph is showing consistent increase in rewards.

2.

Episodes	Min Epsilon	gamma	lambda	catch
10k	0.05	0.49	0.0005	5



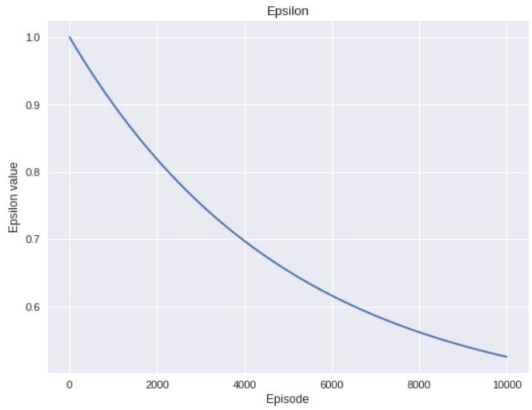


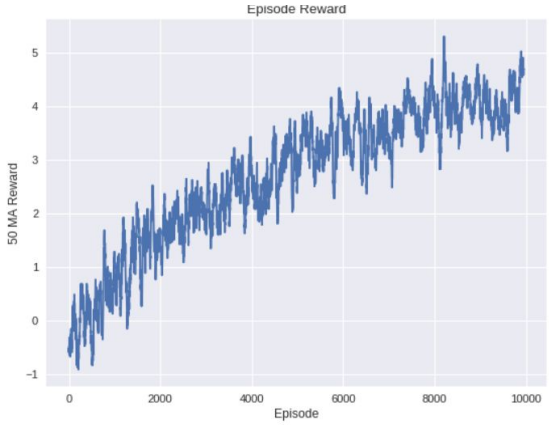
Increased lambda to 0.0005. Because of this epsilon decayed quickly, so agent got less time to explore. After that, Agent went into exploit and reached to goal via same route multiple times.

Epsilon is showing quick fall in epsilon value i.e. exploration finishes in less time.

3.

Episodes	Min Epsilon	gamma	lambda	catch
10k	0.5	0.49	0.00005	2



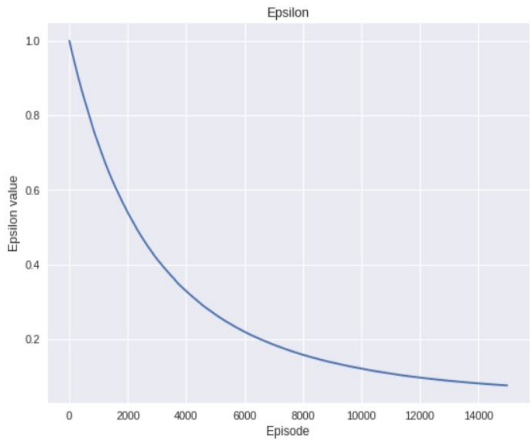


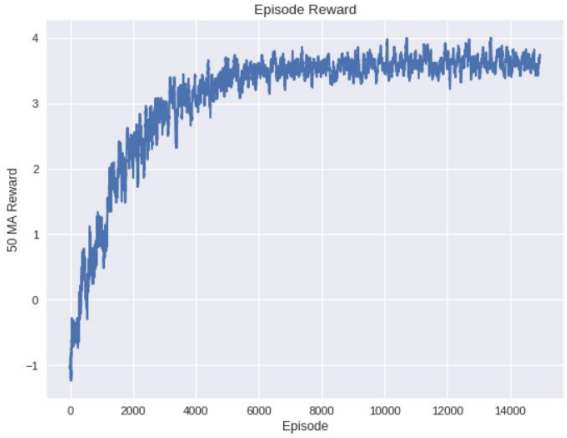
Since minimum epsilon is 0.5, agent is forced to explore there is less possibility that random value will be more than 0.5 . Hence agent will be exploring alot and it will exploit for very less time. Jerry got caught noly twice. Even after first detection, Agent was not able to detect jerry second for some time. Because agent was exploring a lot.

Epsilon graph is showing that epsilon value is decreased to 0.5. i.e agent is exploring and not exploiting. The reward graph is showing fluctuation in rewards (sudden increase and decrease in rewards).

4. Moving Jerry

Episodes	Min Epsilon	gamma	lambda	catch
15k	0.5	0.49	0.00005	6





Reward graph is showing that max reward reached is 4. Unlike previous observations where max rewards were 8, here max reward is 4. I.e tom is able catch gerry in less step and time.

Question:

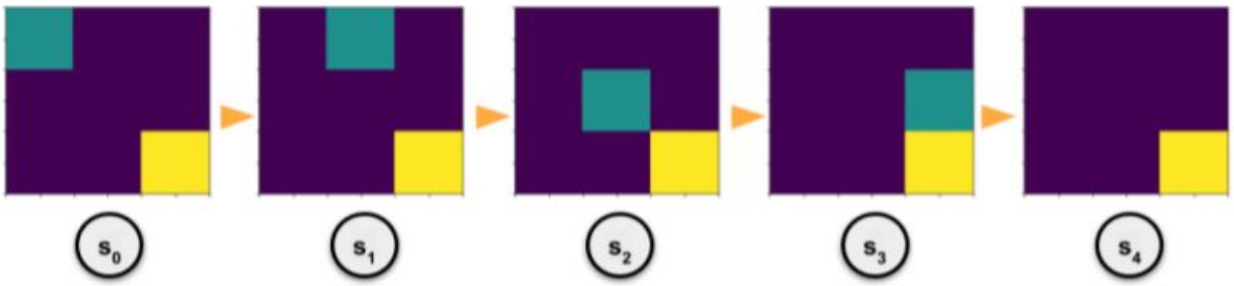
1. Explain what happens in reinforcement learning if the agent always chooses the action that maximizes the Q-value. Suggest two ways to force the agent to explore

   Answer: If agent always chooses the action that maximizes the Q-value, then it will exploit particular path only and won't explore the world much. Agent will follow particular route which gives maximum Q-value.

   We can force agent to explore by two ways:

   a. **Set lamba value very small eg. 0.00001,** so that epsilon value will decrease slowly and it will give more time to agent to explore. Explanation: agent will explore if random.radon value is less than epsilon. If epsilon value is in higher range for long time it means that agent will explore for long time.
   b. **Set large value to minimum_epsilon e.g. 0.6.** If lowest possible value of epsilon is minimum_epsilon. If minimum_epsilon is high like 0.6, there is more possibility of epsilon value will be greater than random.random value. It will make agent to explore more. I mentioned observation of this case in 3rd observation of this report.

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



Answer:

Q-function: formula for Q table

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

Starting calculation Q-function from the last state.

State-4 is goal state i.e. end of game hence Q values for all action is 0.

State-3

Q (S3, Down) = r (S3, Down) = 1

Q (S3, Up) = r (S3, Up) + γ*max{Q (*) } = -1 + 0.99*0 = -1

Since we have not explored state when agent is at upper of s3, we will consider max reward at that state is 0.

Q (S3, Left) = r (S3, Left) + γ*max{ Q (S2) } = -1 + 0.99*(?) = ?

We don't know max reward at S2. We will calculate Q(S3,left) later

Q (S3, Right) = r (S3, Right) + γ*max{ Q (S3) } = 0 + 0.99*1 = 0.99

State-2

We don't know states on Down & Left of S2, so we will consider max value in that state as 0.

Q (S2, Left) = r (S2, Left) + γ*max{ Q (?)} = -1 + 0.99*0= -1

Q (S2, Down) = r (S2, Down) + γ* max{ Q (?) } = 1 + 0.99*0= 1

Q (S2, Right) = r (S2, Right) + γ*max{ Q (S3) } = 1 + 0.99*1= 1.99

Q (S2, Up) = r (S2, Up) + γ*max{ Q (S1)} = -1 + 0.99*(?)= ?

We don't know max reward at S1, so we will calculate Q(S2, Up) later.

State-1

Q (S1, Right) = r (S1, Right) + γ*max{ Q (?) } = 1 + 0.99*0 = 1

Q (S1, Down) = r (S1, Down) + γ*max{ Q (S2) } = 1 + 0.99*1.99 = 2.97

Q (S1, Left) = r (S1, Left) + γ*max{ Q (S0) } = -1 + 0.99*(?) = ?

We don't know max reward at S0, so we will calculate Q(S1, left) later.

Q (S1, Up) = r (S1, Up) + γ*max{ Q (S1) } = 0 + 0.99*2.97 = 2.94

State-0

We don't know states on Down of S2, so we will consider max value in that state as 0.

Q (S0, Down) = r (S0, Down) + γ*max{ Q (?) } = 1 + 0.99*0 = 1

Q (S0, Right) = r (S0, Right) + γ*max{ Q (S1) } = 1 + 0.99*2.9 = 3.94

Q (S0, Left) = r (S0, Left) + γ*max{ Q (S0) } = 0 + 0.99*3.94 = 3.9

Q (S0, Up) = r (S0, Up) + γ*max{ Q (S0) } = 0 + 0.99*3.94 = 3.9

Let's calculate remaining rewards

Q (S3, Left) = r (S3, Left) + γ*max{ Q (S2) } = -1 + 0.99*1.99 = 0.97

Q (S2, Up) = r (S2, Up) + γ*max{ Q (S1) } = -1 + 0.99*2.97 = 1.94

Q (S1, Left) = r (S1, Left) + γ*max{ Q (S0) } = -1 + 0.99*3.94 = 2.9

Q-table

	UP	DOWN	LEFT	RIGHT
S0	3.9	1	3.9	3.94
S1	2.94	2.97	2.9	1
S2	1.94	1	-1	1.99
S3	-1	1	0.97	0.99
S4	0	0	0	0

Inference:

1. Epsilon decides probability of whether agent will explore or exploit. If epsilon is very small then agent will more likely exploit, if epsilon is high then agent will explore more.
2. Lambda plays significant role in decaying epsilon value which results in amount of time spend for exploration and exploitation.
3. Final rewards in Q-table dependents on how much agent have explored. If agent have explored less, values in Q-table may not be matured enough.
4. If gamma is higher, then agent gives more weightage to discounted rewards. This may mislead agent to take wrong action.

Reference:

1. https://ml-cheatsheet.readthedocs.io/en/latest/logistic_regression.html
2. https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/
3. https://medium.freecodecamp.org/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc