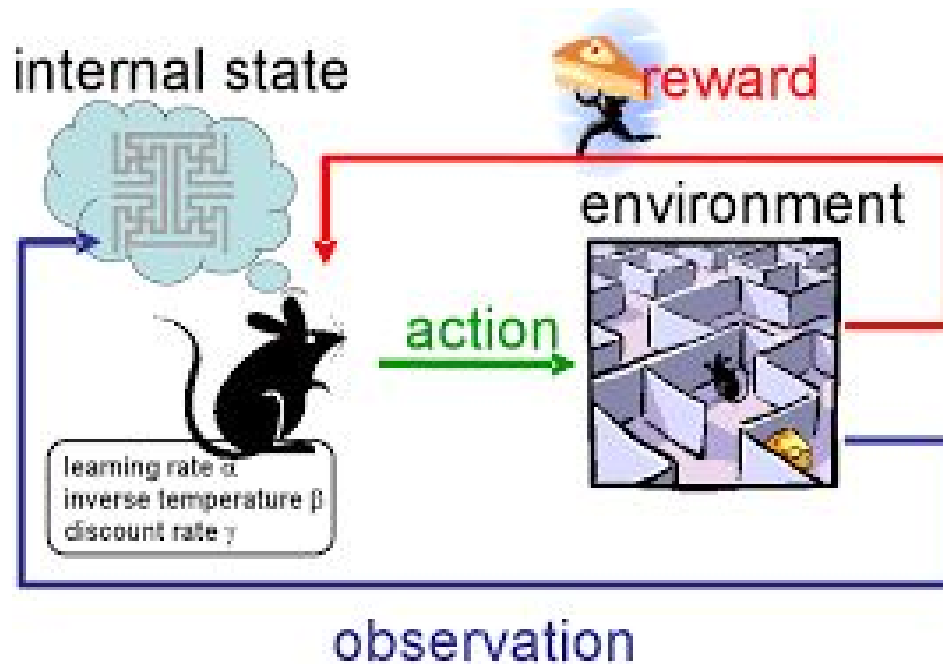


Project 3

Tom and Jerry in Reinforcement learning



Malini Anbazhagan

Person# 50289383

University at Buffalo

Buffalo, NY 14260

Contents

1. Introduction.....	3
2. Code explanation.....	4
3. Writing task.....	8
4. Summary.....	9
5. Reference.....	10

Introduction

The project combines reinforcement learning and deep learning. The task is to teach the agent to navigate in the grid-world environment.

“What is grid world?”

Grid World is a 2D rectangular grid of size (N_y, N_x) with an agent starting off at one grid square and trying to move to another grid square located elsewhere. Such an environment is a natural one for applying reinforcement learning algorithms to discover optimal paths and policies for agents on the grid to get to their desired goal grid squares in the least number of moves.

In the project, it is modeled after “Tom and Jerry” cartoon, where Tom, a cat, is chasing Jerry, a mouse. In the 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, deep reinforcement learning algorithm - DQN (Deep Q-Network) is applied.

“What is DQN?”

Algorithmically, the DQN draws directly on classic Q-learning techniques. In Q-learning, the Q-value, or “quality”, of a state-action pair is estimated through iterative updates based on experience. In essence, with every action we take in a state, we can use the immediate reward we receive and a value estimate of our *new* state to update the value estimate of our original state-action pair

Code Explanation

As usual, the required packages are imported such as 'random', 'math', 'time', 'numpy', 'keras', 'matplotlib', 'IPython'.

Environment:

This part of the code creates the environment. The code creates the image of tom and jerry, makes a grid for agent to move around. It has functions created for reward (which is collects reward if any), episode over (which is used for when the episode ends or the goal is reached), state update (which is used to move left, right, upward, or downward), reset (which is used to reset the start and start the episode again).

Random:

This part of the code creates random simulations. It chooses random action and applies random function.

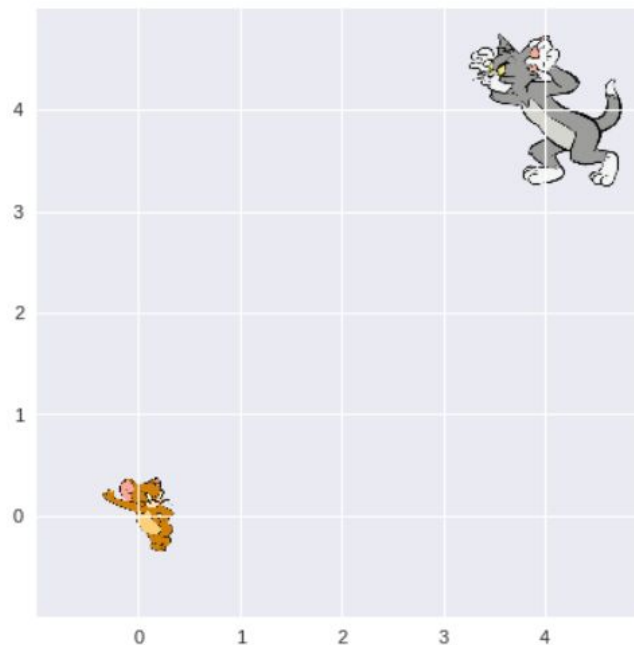


Fig 2.1

Brain:

The 'brain' of the agent is where the model is created and held. This part of the code created sequential keras model.

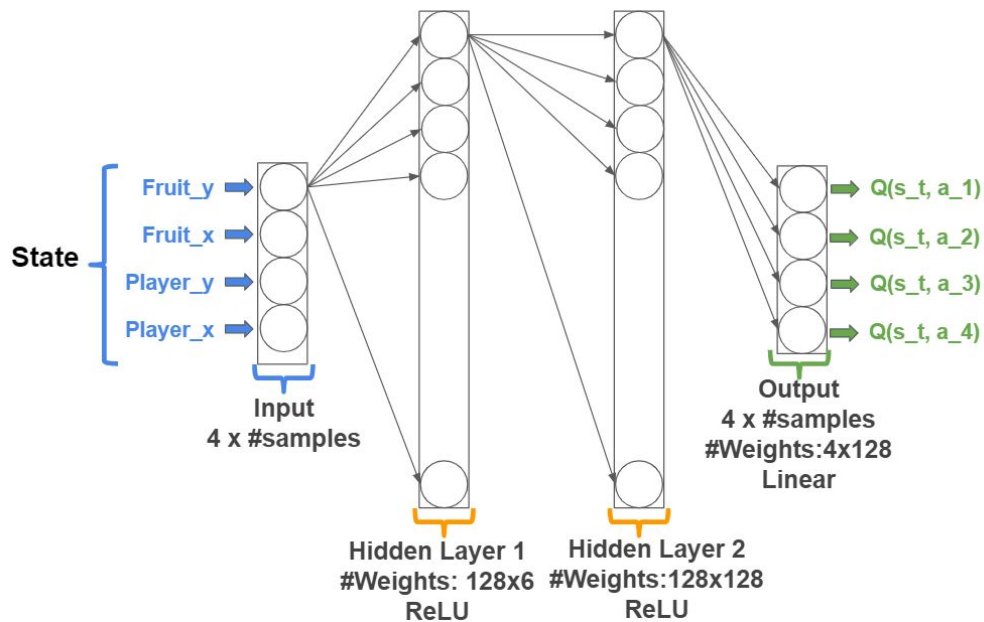


Fig 2.2

Memory:

In this part of the code, block the main functions that will be used to store the experiences of our agent are stored here.

Agent:

The agent will randomly select its action at first by a certain percentage, '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. To decrease the number of random action, as it goes, an exponential-decay epsilon is used, that eventually will allow the agent to explore the environment.

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

Running the game:

The algorithm, simply calls the methods in sequence for some given number of episodes. At the beginning of an episode, the environment is reset, and passes its return value, the initial state, to the agent's act method.

This returns an action, which is then passed to the environment's step method. This returns the next state, the reward, and the boolean indicating if the episode is over. Then the observation tuple is saved to the agent's memory via the agent's observe method, then run a round of training by calling the agent's replay method. Then the environment can be rendered.

If the episode is over, loop is broken, otherwise it continues with the next state being passed to the agent as the (now) current state.

Questions:

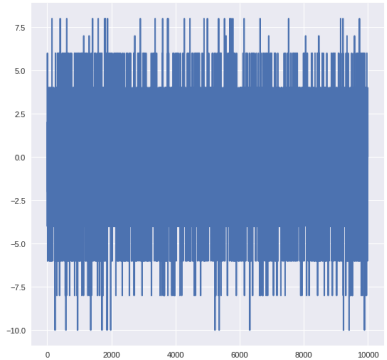
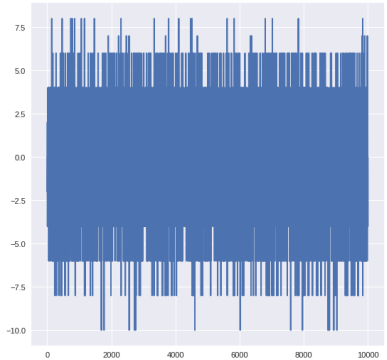
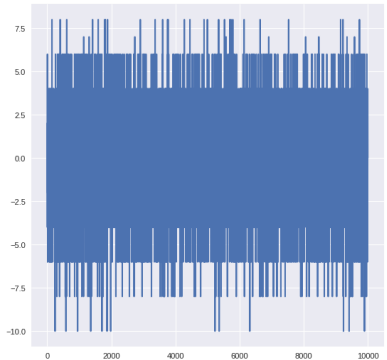
1. What parts have you implemented?

I implemented only few lines of code.

- creating a keras sequential model with 2 hidden layers with 128 density, with activation activation sequence as 'relu', 'relu' and 'linear' in the "Brain"
- Implementing exponential decay formula for epsilon and Q function

2. Can these snippets be improved and how it will influence the training the agent?

The code can be improved by tuning the hyperparameters.

Hyper parameter	Min and Max of rewards	Graph
Batch size: 64	Min:-6 Max:8	
Batch size: 54	Min:-10 Max:8	
Batch size: 74	Min:-6 Max:8	

Writing tasks

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.

The agent has to get all the possible values of Q values possible. The agent may initially think a particular Q value to be the maximum which might not be true. The maximum Q value path is not necessarily the best/shortest path.

- Temporal difference method
- Brute force

The above mentioned ways can be used to force the agent to explore

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

	Action			
State	Up	Down	Left	Right
s0	0	3.97	0	3.96
s1	-3.97	2.97	-3.94	2.99
s2	-2.97	1.99	-2.97	1
s3	-1.99	1	-1.99	0
s4	-1	0	-1	0

Summary

While doing this particular project, I got to learn DQN model. I got to learn how the dqn works, the advantages and disadvantages of it.

I also learned when to use methods like brute force, MPD, value function, Temporal difference method, direct policy search and so on.

I also learned the relation between each hyperparameter and parameter diffres for each algorithm and data set.

Reference

Grid world

<https://towardsdatascience.com/training-an-agent-to-beat-grid-world-fac8a48109a8>

Dqn

<https://towardsdatascience.com/advanced-reinforcement-learning-6d769f529eb3>

Reinforcement learning

https://en.wikipedia.org/wiki/Reinforcement_learning