ECE448 MP3 report 4 credits students

Yu\_Fan Fang, Chieh Hsu, Yu\_ming Huang

Video link: <https://drive.google.com/drive/folders/1YM-64vyDKZD9XTt3UH3V_vblmwmP-M5x?usp=sharing>

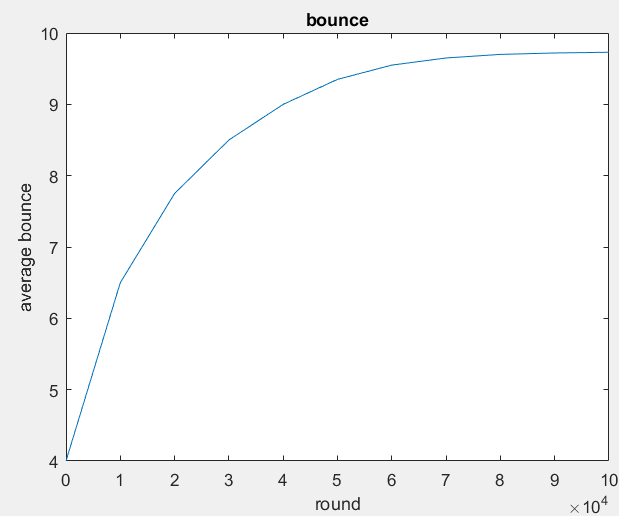
Part 1:Q-Learning Pong

1-1:

**Implementation:** Q-Learning

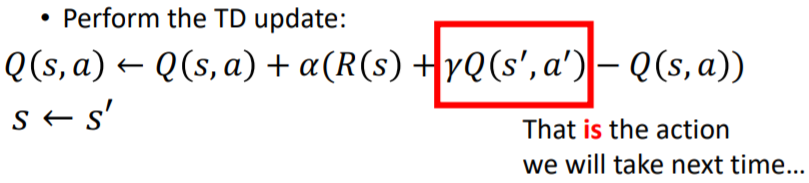
In this part of assignment, we create ball, paddle, QLearning, and Pong class so as to use QLearning to learn the weights for pong game playing. We store all the Qvalues and the number of occurrence for each state and action pair. The states should be discretized if QLearning is used. There are a total of 12\*12=144 positions. 2 signs for the ball horizontal velocity, 3 signs for ball vertical velocity, and 12 positions for the paddle. The new QValue after each update is calculated as follows:

Picture 1

For the overall training process, we store all the weightings and the training rewards into an additional file. Whenever the training is going to start, we utilize the stored training weightings to play. As the train phase progresses, the punishment for missing the ball will suppress the weightings from missing the balls

**Implementation:** Sarsa

In this part of assignment, the QLearning algorithm mentioned above is replaced with Sarsa algorithm. We first select the action that we are going to take next and perform the TD update.



We also store all the weightings and the training rewards into an addition file and the rewards during the process of training. And the resulted weightings are used in pong game playing.

1-2:reversing environment

For the reversing environment problem, we alter the initial state for the ball horizontal velocity. And the environment plot is also altered. The function that deals with the paddle hitting the ball is also changed according to the topology of the environments. After making the above mentioned alterations, we are able to keep bouncing the balls with the paddle.

1-3:extra credit

Requirement 1: Create a graphical representation of your Pong game. A GUI would be preferable, but a text-based console implementation is also acceptable, as long as the text characters are redrawn in place (the image of the playing court does not move around on the screen). Create and include an animation of your agent playing the game.

We modify the left paddle and let it be controlled by the player. The agent for the computer used the QLearning algorithm trained in part 1. The implementation process is almost the same. We import the keyboard module and enable the users to control the left paddle. If the users pressed “s”, the paddle will move up, and if they press “w”, the paddle will move down.

Requirement 2:  If you created either a GUI or a console-based implementation, allow for a human to play against the AI. Are you able to defeat it? What are its strengths and weaknesses? Describe your discoveries.

We are almost impossible to defeat the computer agent, the moves it makes are almost optimized during the course of training. However, if we manage to keep bouncing the ball toward 2 ends of the boundary, the computer agent will finally lose (it takes a lot of games for the human player to win).

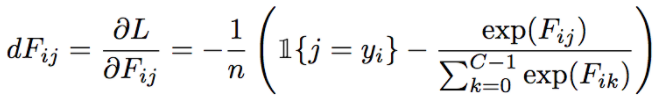
## Part 2: Behavioral Cloning and Deep Learning (Pong)

2-1: Policy behavioral cloning

**Implementation:**

In this part of assignment, we expect to use deep learning algorithm to clone the action of an expect policy. We use four neural network layers with 256 nodes per hidden layer along with 125 batch size, 0.01 learning rate, scaling weight 0.1. We use python as our developing environment with it’s helpful package “numpy” which can quickly calculate matrix multiplication. After each epoch, we upgraded the parameters w1, w2, w3, w4, b1, b2, b3, b4 by the cross entropy-loss function and go on to next epoch.

image3.png



Also, we decade the learning rate to 0.005 after 2000 epoch mutually. With 2500 epoch training we are willing to get the accuracy of 90% and then started to input the decision function to the game environment created by part 1 with some change from discrete to continuous environment.

What is the benefit of using a deep network policy instead of a Q-table ?

Comparing to Q-table, deep network policy used much smaller memory size since it only needs spaces for Ws and Bs which is the layer node sizes, in this case, 256. In contrast, Q-table store almost all the possibilities of five parameters. However, one benefit of Q-table is it only need little calculation to decide which action to make, but randomly choose when it saw a new state while testing where deep network policy don’t need to care about new states and could work in continuous environment.

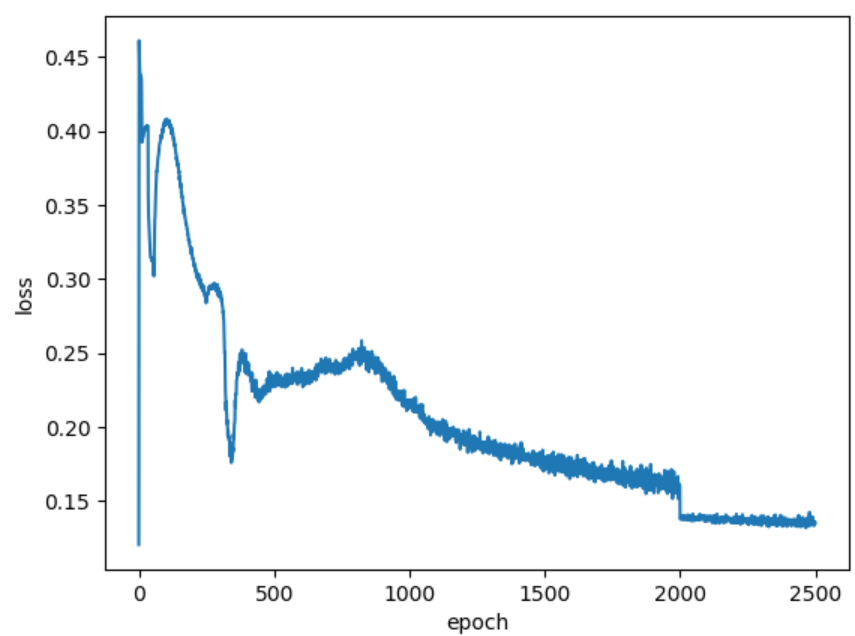
Confusion matrix:

|  |  |  |  |
| --- | --- | --- | --- |
| y\out | -1 | 0 | 1 |
| -1 | 2968 | 117 | 249 |
| 0 | 136 | 2066 | 141 |
| -1 | 206 | 145 | 3974 |

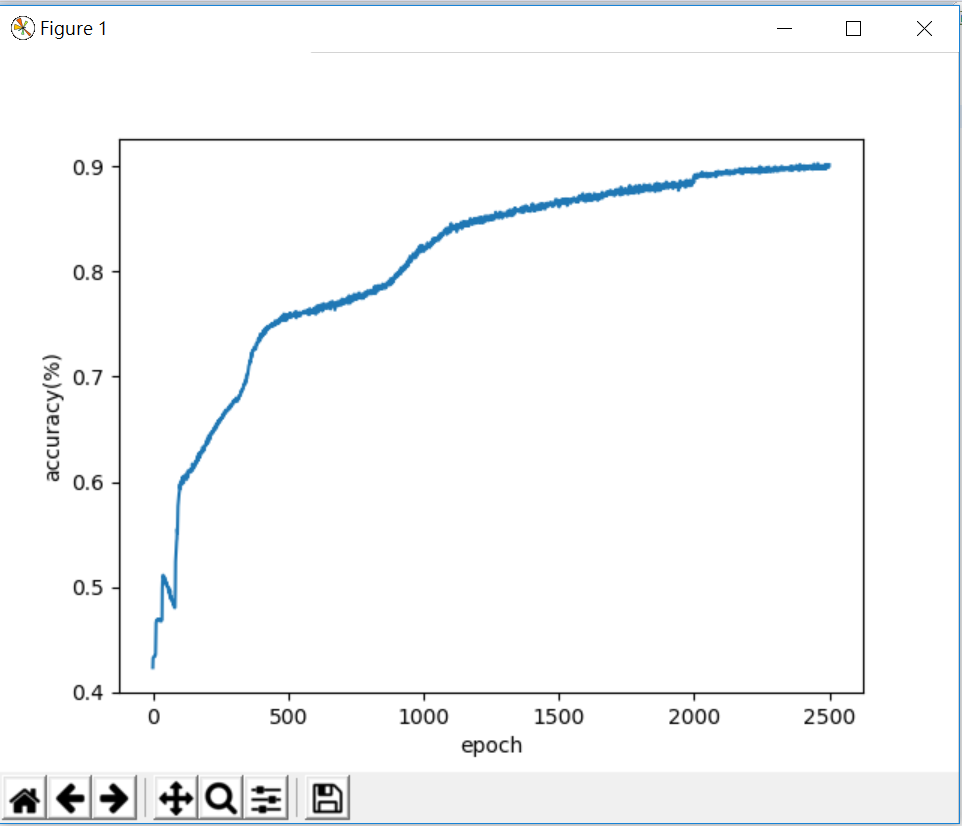
Misclassification:

|  |  |  |  |
| --- | --- | --- | --- |
| y\out | -1 | 0 | 1 |
| -1 | 0.890% | 0.035% | 0.075% |
| 0 | 0.058% | 0.882% | 0.060% |
| -1 | 0.048% | 0.033% | 0.919% |

Loss : 0.1348406063215956



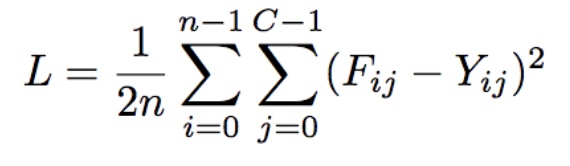
Accuracy : 0.902



Bounces: 35 in average with 300 games, maximum:126

2-2: Behavior Cloning Using Advantage Estimation

**Implementation:**

I construct a neural network with four layers and each layer contains 256 neurons except last layer. I normalize five features of every training data and split 10000 data into 80 batches meaning that the batch size is 125. Then feed data of each batch into forward network and do back propagation once for each batch. This process lasts 500 epochs and the all the weights and biases are trained well. In the testing phase, normalize this input state with mean and standard deviation of training data, then feed into forward network and pick the action with the largest advantage.

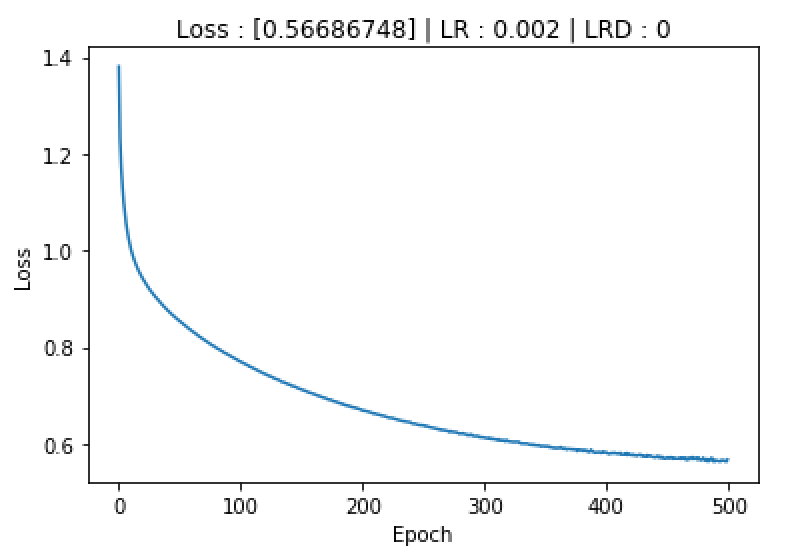
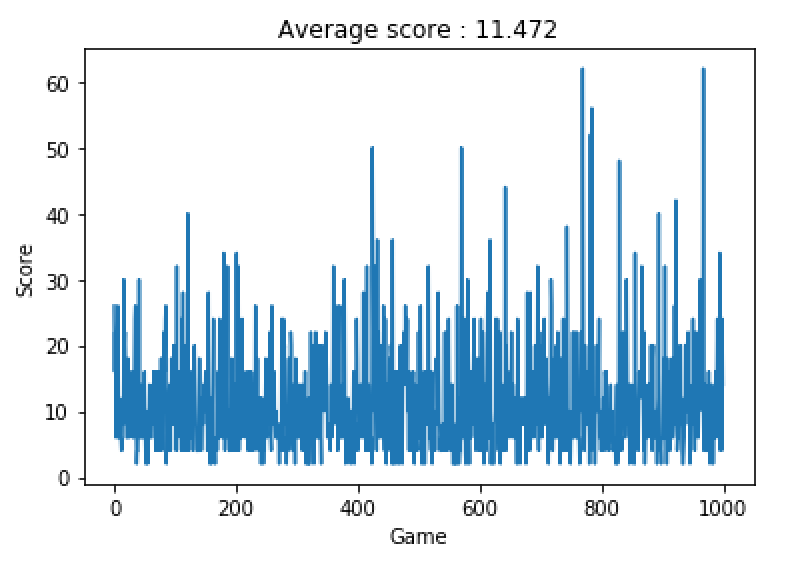
This is my loss function, which is provided on the webpage.

**Hyperparameter Setting:**

Fixed learning rate : 0.002, Weight initialization : -0.2 ~ 0.2

It takes me lots of time tuning these parameters. It seems that larger weight initialization can increase the upper bound of accuracy, but too large weight may cause numerical problem because of unstable loss function. On the other hands, learning rate can’t be too large or the loss may converge too fast and can’t go down.

**Results :**

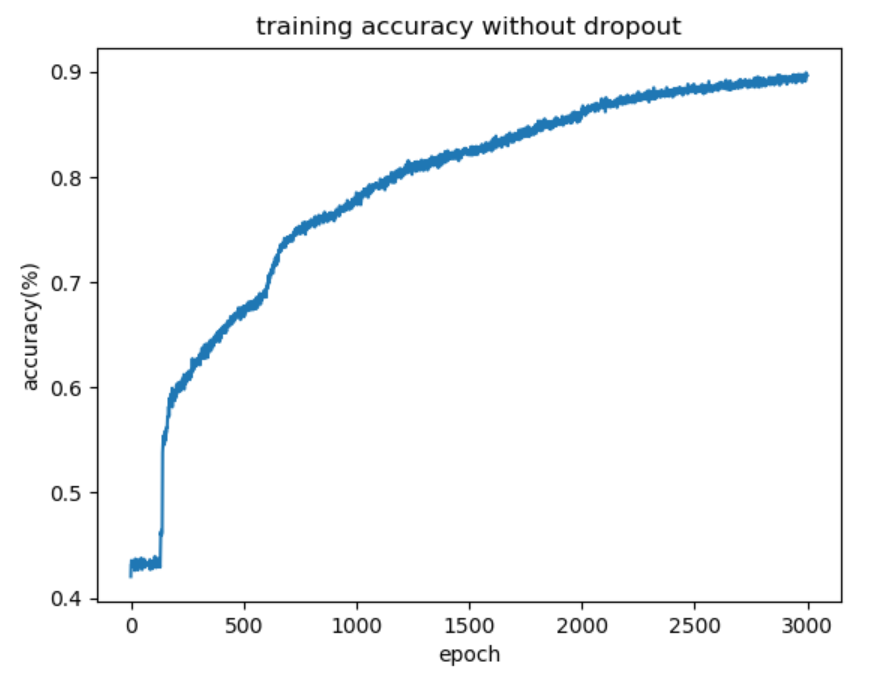
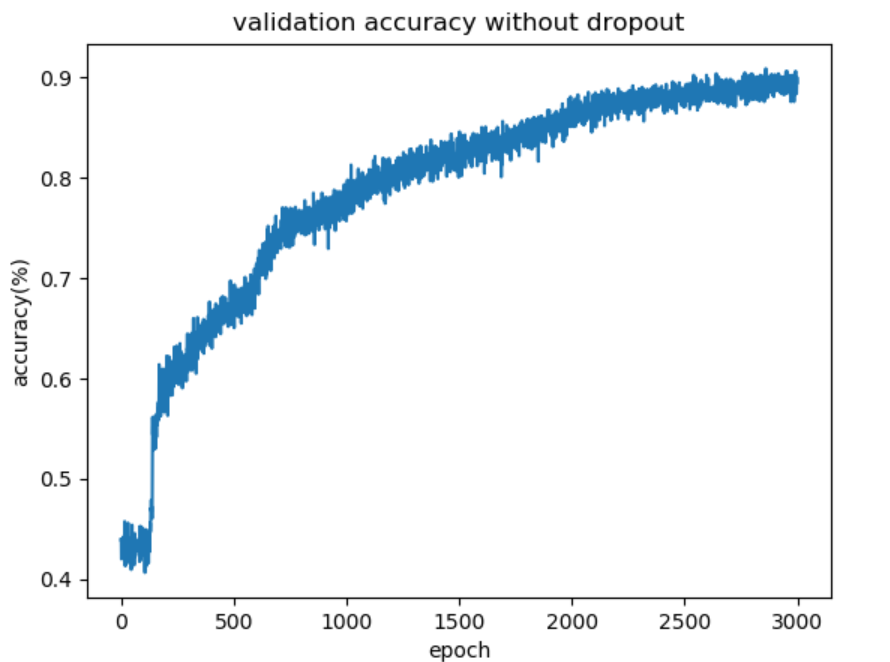
Loss Curve :   
Record of 1000 games :

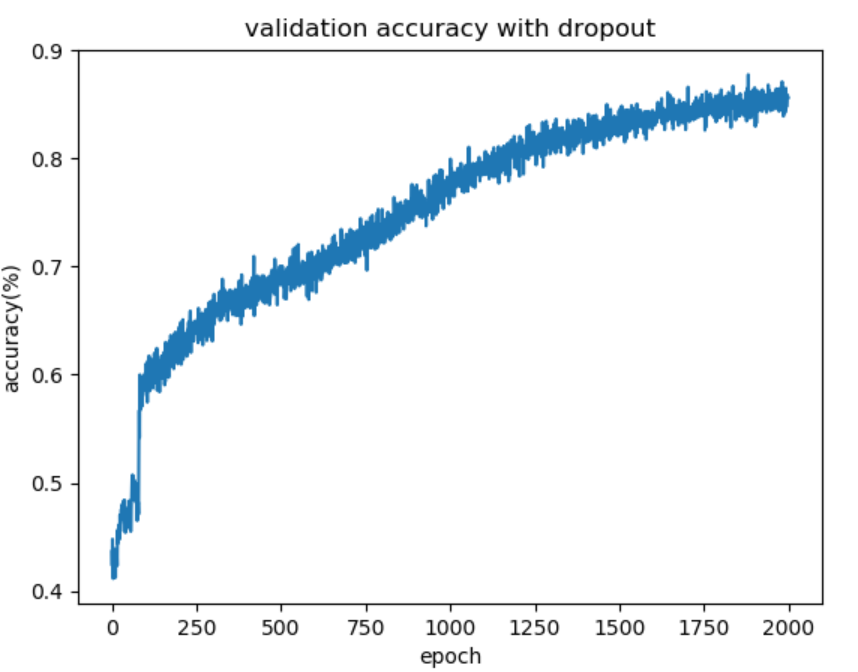
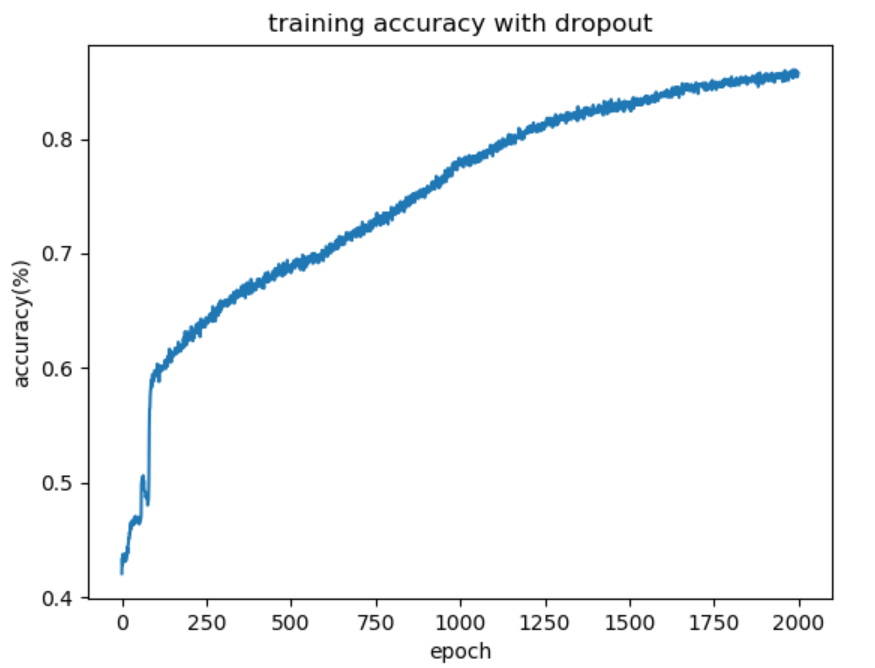
**Extra Credit :**

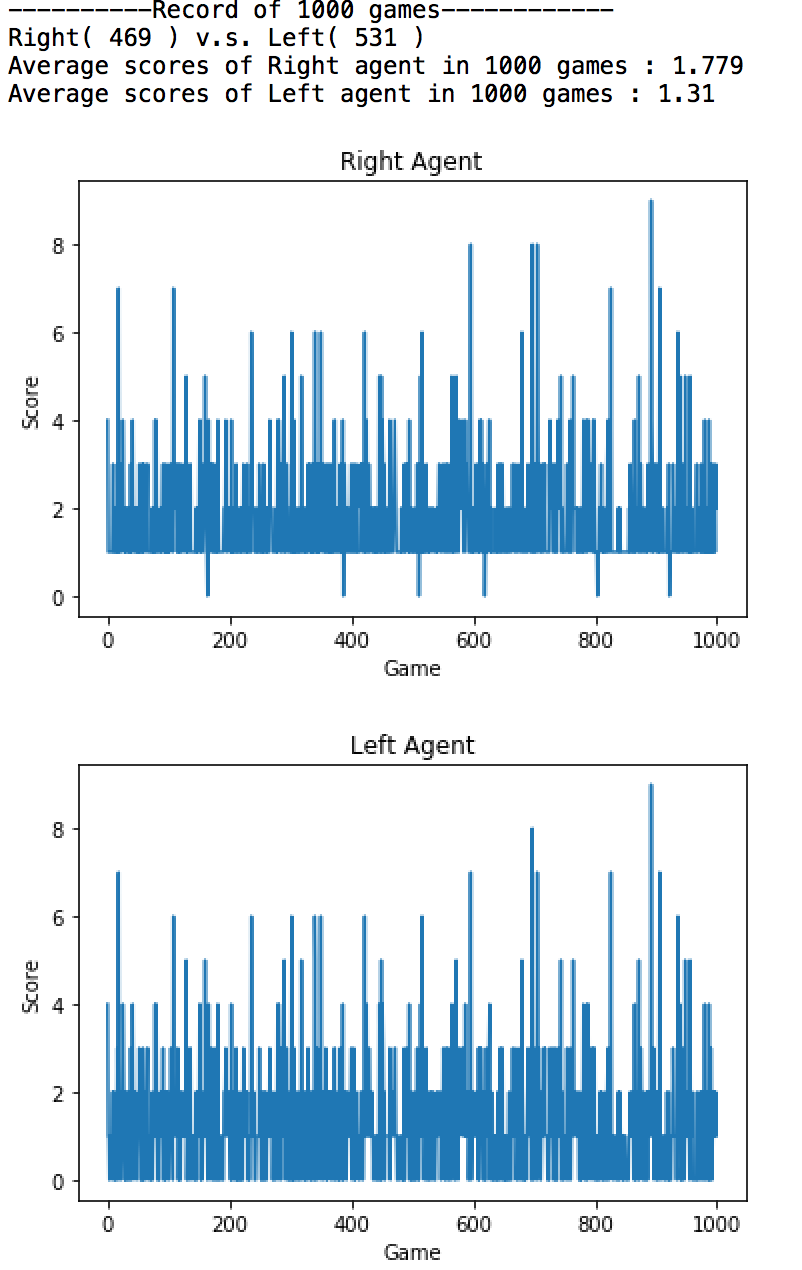
**2. Implement dropout and 80%-20% train-validation split**

**Implementation :**

1. Dropout : define a parameter p from 0 to 1 and randomly create an array from 0 to 1. If Aij < p then Fij = 0, else Fij = Fij \* 1.
2. Train-validation split : pass 8000 data to training and pass 2000 data to test after each epoch
3. We assign p = 0.2, learning rate = 0.001, weight = 0.01

**.**  



**3. Part 1 agent (Right) v.s. Part 2.2 agent (Left)  
**

**Implementation :**

For every state, discretize five features and feed into Q learning agent, also feed continuous features into Deep Q learning agent. I use the same weight trained in part2.2 for the left agent. Just change two features before feeding into Deep Q learning agent, ball\_x —> (1-ball\_x), ball\_velocity\_x —> -1\*ball\_velocity\_x.   
  
**Result :**

Part2.2 agent beats Part1 agent. Each game lasts shorter than the original ones because other side can’t reflect the ball all the time.

**Video :**

<https://drive.google.com/drive/folders/1YM-64vyDKZD9XTt3UH3V_vblmwmP-M5x?usp=sharing>