Prathik Sannecy

Machine Learning

Programming Assignment 2

3/11/2020

Running the Code

The code is written in Python3. It uses the numpy, matplotlib, and scipy libraries that need to be installed (not installed by default).

To run the code, go to the directory that contains Programming_Assignment2:

```
python3 -m Programming_Assignment2.Source.Run_TicTacToe [train]
```

Use the 'train' option to reset/retrain the Q Table

Otherwise, no arguments will allow the user to play Tic Tac Toe

Note: retraining the Q Table takes a long time to run (perhaps even an hour, depending on how many epochs)

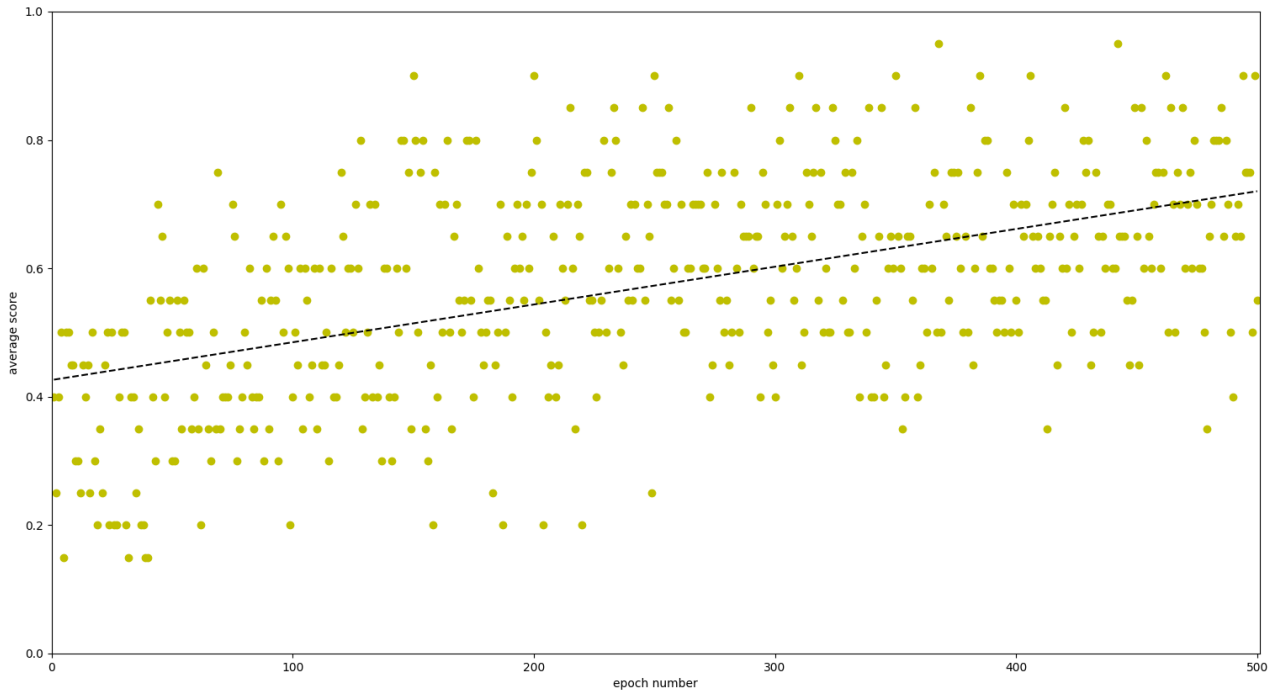
After training is complete, it prints a plot of the training progress for the agent with the epoch number on the horizontal axis and the (total score) / 10 against the baseline (random) opponent on the vertical axis.

Approach/Output

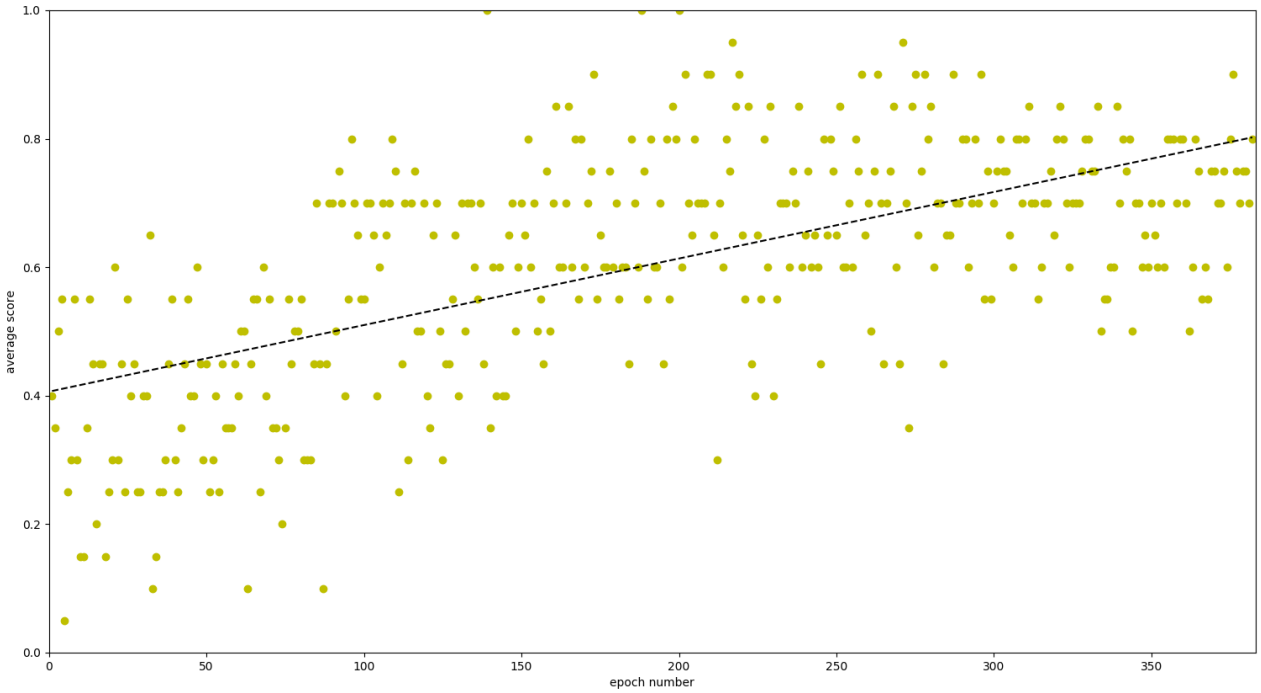
The approach I took was to train the agent with default values of ϵ =0.1, δ =0.01, and *m*=50. I ran 500 epochs (each epoch is 10 games) against a random player. For now, I decided to make the Q Learning agent only be the 'O', as 'O' is more likely to lose and thus it would be easier for me to notice the impact of my learning algorithm. If time permitted, I would have created the option of having the agent be either the 'X' player or the 'O' player, each with their own separate Q table.

I parameterized several different things for training the Q Learning agent. In the Q Learning formula, I parameterized the learning rate and the discount factor. In the training process, I parameterized the number of epochs, delta, m, and the initial randomization factor. I played around with changing the epochs, delta, m, and the initial randomization factor, and all their results seemed to converge within 500 epochs. The next step that I would have done would try experimenting with the Q Learning formula itself, rather than just use the default values.

Below are a few scatterplots produced that shows the training progress of the Q Learning agent against a random player during a couple of runs:



Training 1



Training 2

As shown above in both instances, the average score (+1 for win, +0.5 for tie, +0 for loss) steadily increases as the training runs.

Results and Conclusion

As shown by the scatter plots above, the Q Learner fared well against a random player that had no strategy. In most cases, if ended up either tying or winning (as indicated by the y-axis being greater than 0.5).

I ended up playing a 10 different games against it, of which the Q agent won 0, tied on 1, and lost on 9. During those 10 games I used several different strategies (e.g. choosing a corner square first, choosing the center square first, etc). The agent only tied against the simple strategy where I only planned 1 move ahead (e.g. continuously try to get three in a row). It fared badly when I tried a more complex strategy that requires looking 2 moves ahead, such as trying to pick 3 corner squares to force a win.

I think part of the reason that the agent lost so often is because I trained it against a random player. In most cases, the random player wouldn't make the ideal move, so the agent wouldn't have to "defend"; instead, it just played greedily and always tried to get 3-in-a-row as quickly as possible. When I played it, I had a strategy, and since the agent never learned to "defend", it couldn't stop me from winning. If I had more time to vary the hyperparameters, I would experiment changing the learning rate to see if this could help it learn better, but as as mentioned above, training the Q Learning agent takes a long time.

Therefore, while the agent achieved success of learning the fundamental idea Tic Tac Toe, it never really mastered it; in Tac Tac Toe, if both opponents played perfectly the result would be tie; ideally whenever I played the agent the game would end in a draw. However, it perform much better than a random player, and I believe with some finely tuned hyperparameters, it could have learned to play like the perfect player.