ENPM808F

HOMEWORK#4 REPORT

DINESH KADIRIMANGALAM

Section 1: Program a game

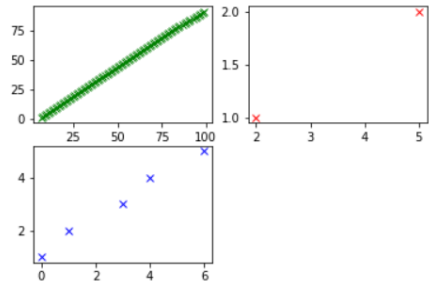
The game player takes turns drawing lines to complete the boxes. The game is completed when there are no boxes left to be drawn.

Section 2: 2*2 Grid, Q-Learning and Function Approximation

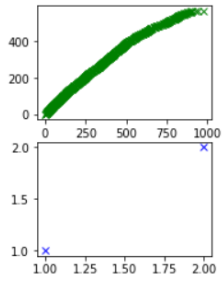
The game is started with 2*2 grid. For completion of one box a reward of +1 is assigned, +5 for a win and 0 otherwise. The game is trained itself to play through self-learning ang Q-learning. It was trained for 100,1000 and 10,000 times and the performance id recorded int the Q-value table. The parameters are, learning rate = 0.09, discount factor = 0.8 and epsilon = 0.9

Table for performance against computer while training

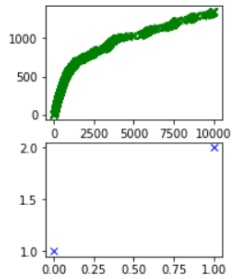
```
win 91
lose 2
tie 5
invalid 2
```



```
win 568
lose 0
tie 2
invalid 430
```



```
win 1354
lose 0
tie 2
invalid 8644
```



Win = Green; Lose = Red; Tie = Blue

The number of wins increases with the increase of trains and due to the invalid moves, it is learning not to repeat the same mistake again in the future. And the loss-ratio is also significantly decreasing per no-of trains.

Table for performance against Random Player for 100, 1000 and 10000 games played

```
Q_RandomPlayer(100,Q1)
```

```
win 36
lose 7
tie 57
not trained 600
```

```
Q_RandomPlayer(1000,Q1)
```

```
win 430
lose 95
tie 475
not trained 6000
```

```
Q_RandomPlayer(10000,Q1)
```

```
win 4285
lose 918
tie 4797
not trained 60000
```

The above results are when the computer is trained against itself while the other computer is player is also learning.

Table for performance against Random Player for 100, 1000 and 10000 games played with a Function approximation

NN_RandomPlayer(100,NN)	NN_RandomPlayer(1000,NN)	NN_RandomPlayer(10000,NN)
win 49 lose 7 tie 44 not trained 600	win 416 lose 88 tie 496 not trained 6000	win 4279 lose 883 tie 4838 not trained 60000

The chances of losing against a random player decreases as the number of raining level increases as it encounters many un trained state. The program couldn't perform better than random player because of the un trained states, and there are around 531441 states to be covered. Because of the random player moves the numbers shown vary each time the program is executed. CMAS is used as a function approximate in this case and it is trained 1000 times. When the state and action is given the CMAC gives the Q-Values and based on this the action is decided to get the maximum reward. And, as the number of trainers increases the performance of Q-Learning also increases. The CMAC will be trained very close to the Q-Table with increasing the number of training.

Section 3: 3*3 Grid, Q-Learning and Function Approximation

The procedure is very same as the 2*2* grid. As there are many untrained stated in the random player case the chances of win is less than the loose. To be precise there are only 1/10th of chance and there are 282429536481 states to be covered and the numbers vary each time when the program is executed.

Table for performance against computer while training

win 33 lose 58 tie 7 invalid 2	win 108 lose 385 tie 19 invalid 488	win 3119 lose 1164 tie 14 invalid 5703

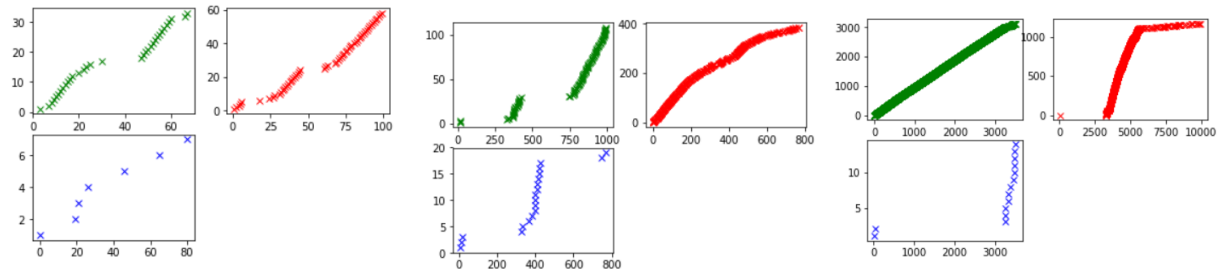


Table for performance against Random Player for 100, 1000 and 10000 games played

Q_RandomPlayer(100,Q1)	Q_RandomPlayer(1000,Q1)	Q_RandomPlayer(10000,Q1)
win 50 lose 44 tie 6 not trained 1102	win 443 lose 461 tie 96 not trained 11028	win 4495 lose 4579 tie 926 not trained 110209

To conclude, Performance of Q-Learning increases as number of trainers increases and Neural Network will be trained closer to the Q-Table with a higher number of trainings. And from the caparison of vales from the above tables, we can see the functional approximation is a better representation compared to the Q-Value representation.