Robot Learning Project 4

ENPM 808F

Date: May 28, 2019

Authors: Ashwin Varghese Kuruttukulam(115906518)

Description

This project is an implementation of a Q learning algorithm which trains itself through self-play to play the game of dots and boxes. The q learned player is then tested against a random move player.

Main Observations

- The 2x2 version of the game has smaller number of states and hence lesser number of training games are required for the algorithm to learn to win.
- High number of training games for the 2x2 game in the neural network causes the performance be worsen after a point. This may be due to overfitting. But this was not observed in the 3x3 case because the number of possible states are very large.

Results

The ratio of the number of games won for the different cases is shown below:

	2x2 grid size	3x3 grid size
Q Table	**6.45**	**1.36**
Neural Net	**1.30**	**2.67**

State Representation

The state is represented as a list of the state of all the edges(whether they are filled or not). The list is stored in row major form. This means that every even row will have 1 edge less than the odd rows. This is because the even rows

correspond to the horizontal lines and there is always 1 less horizontal line in comparison to the rows containing vertical lines.

Note: The state was flatten and represented as one dimensional array. This enable the algorithm to run much faster than if it was a 2-D list or array.

Q Table

The Q table has the all the states and the weights associated with possible actions of that state. The simple way to implement the qtable was to have a a numpy array with the number of rows as the number of all possible states and the number of columns as number of all possible actions in a state.

While this was feasible for a 2x2 game. The number of states in a 3x3 game is **16777216.** Maintaining a numpy array of this size was very inefficient. So instead the q table was represented by a dictionary, to which each state was the algorithm comes across was added.

Reward scheme

If a box was obtained => +1
Win => +5
otherwise => 0

Note: Draws were not awarded any positive reward

Neural Network Function Approximation

For simple cases when the number of possible states are low, the maintaining a Q table is feasible. But when the number of states are very large, it will be difficult to maintain a Q Table. Therefore, we train a neural network to give the same response as the q table. This can be done by initializing a single layer neural network with random weights and then updating the weights according to:

$$Q_{\text{old}} = w(1,a) + w(2,a)i.$$

$$Q_{\text{next}}(1) = w(1,1) + w(2,1)j; \quad Q_{\text{next}}(2) = w(1,2) + w(2,2)j.$$

$$\text{Now set } Q_{\text{next}} = \max\left\{Q_{\text{next}}(1), Q_{\text{next}}(2)\right\}.$$

$$Q_{\text{new}} \leftarrow (1-\alpha)Q_{\text{old}} + \alpha\left[r(i,a,j) + \lambda Q_{\text{next}}\right].$$

$$w(1,a) \leftarrow w(1,a) + \mu(Q_{\text{new}} - Q_{\text{old}})1; \quad w(2,a) \leftarrow w(2,a) + \mu(Q_{\text{new}} - Q_{\text{old}})i.$$

Where,
W is a weight matrix containing the weights connecting the two layers.
μ is the neural net learning rate
α is the q value learning rate
i is the current state
j is the next state
r is the reward
λ is the discount factor

Training

During the training part, two q learning agents play against each other and parallelly update the same q learning table. Initially I tested out the q learning player playing against a random player and updating the q learning. But I believe that did not work out because the learning was very slow. Then I created a script to make two q learning agents plays against each other but update two seperate q tables.

But the q table does not take into factor which player it is playing as. It only cares about what the current state is and what the best action to take is. Therefore I concluded that I could use the same q table for both the agents. This also enabled the table to be trained twice as quickly since it was learning from the moves of both the players and not just one.

Visualization

In order to realize how good one player is from the other, I used two metrics.

1. Number of wins
2. Sum of score

In order to easily visualize this metric, I plotted the scores of each of the player as a function of the game number.

Note: In order to make visualization better for games greater than 100 games easier, I have average close by scores such that only 100 values are plotted. And hence the x axis of all the plots have a range 0-100.

Testing

In order to test the q learning algorithm, I wrote a script to generate random legal moves. The results of the games between the q learning algorithm and the random player for different number of training games and grid sizes are given below.

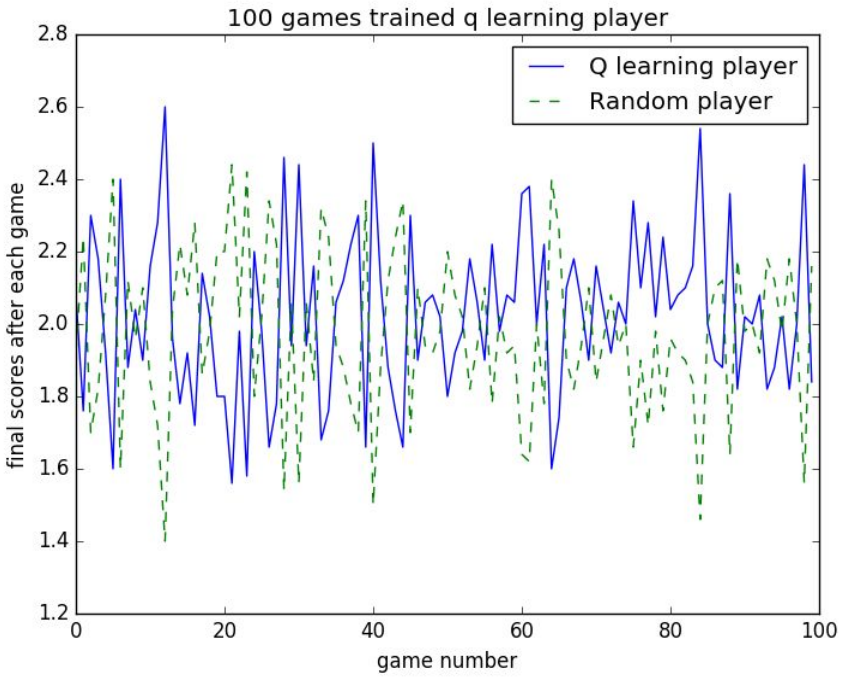
Note: Number of games for testing was 5000

2x2 grid (QTable)

<u>100 Training Games</u>
Q learning player win count : **2060**
Random player win count **: 2101**
Q learning/Random player win ratio: **0.9804854831**

Q learning player score sum : **9903**
Random player score sum **: 10097**
Q learning/Random player score sum ratio: **0.98078637219**

Observation: The q learning agent is not any better than a random player



<u>1000 Training Games</u>

Q learning player win count : **2299**

Random player win count **: 1568**

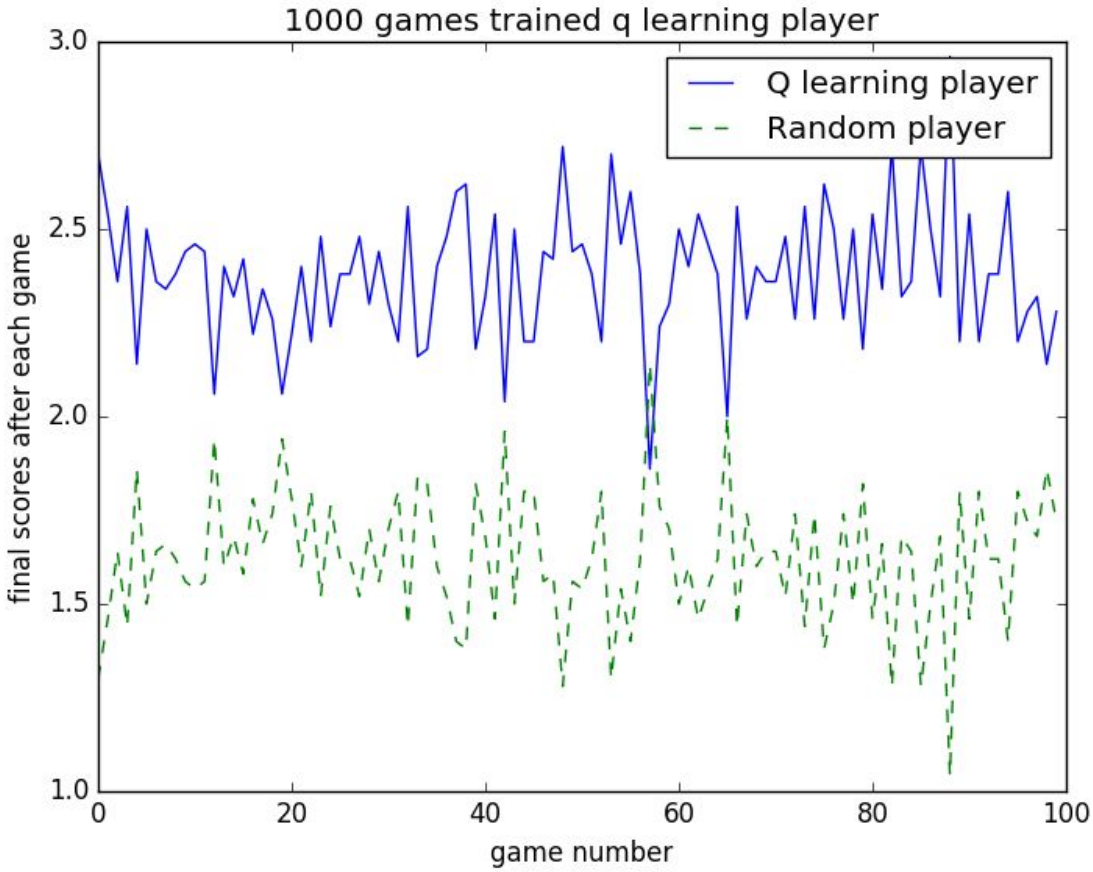
Q learning/Random player win ratio: **1.46619897959**

Q learning player score sum : **11130**

Random player score sum **: 8870**

Q learning/Random player score sum ratio: **1.25479143179**

Observation: The q learning agent is better than a random player by a small margin



<u>10000 Training Games</u>

Q learning player win count : **2808**

Random player win count **: 435**

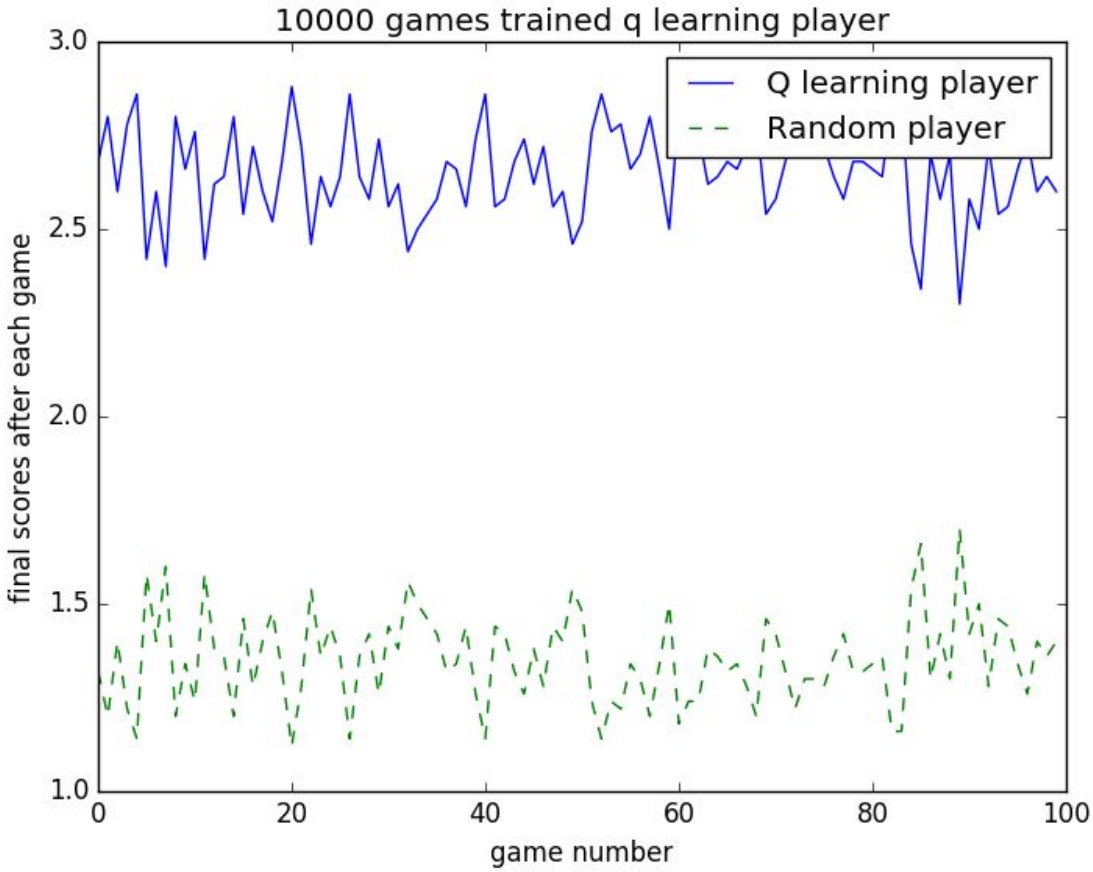
Q learning/Random player win ratio: **6.45517241379**

Q learning player score sum : **12747**

Random player score sum **: 7253**

Q learning/Random player score sum ratio: **1.75747966359**

Observation: The q learning agent is much better than a random player.



3x3 grid

<u>100 Training Games</u>

Q learning player win count : **2504**

Random player win count **: 2496**

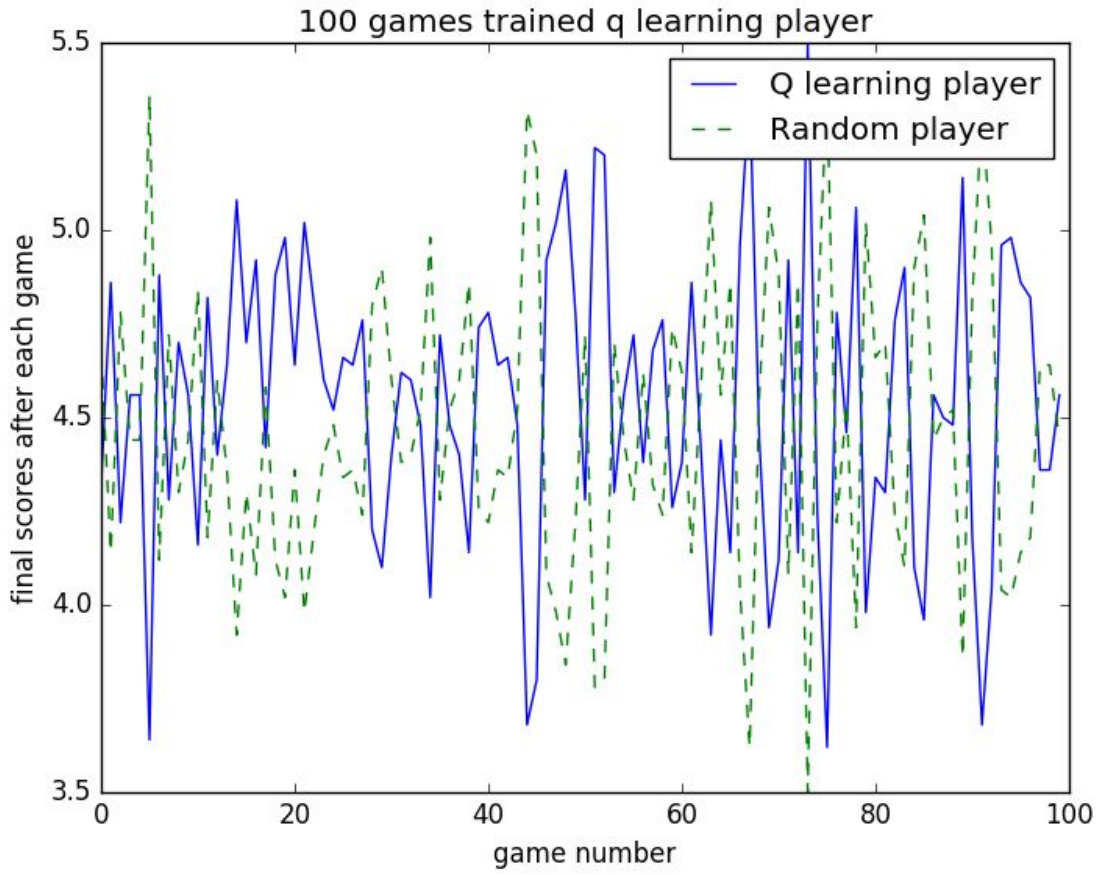
Q learning/Random player win ratio: **1.00320512821**

Q learning player score sum : **22528**

Random player score sum **: 22472**

Q learning/Random player score sum ratio: **1.00249199003**

Observation: The q learning agent is not any better than a random player

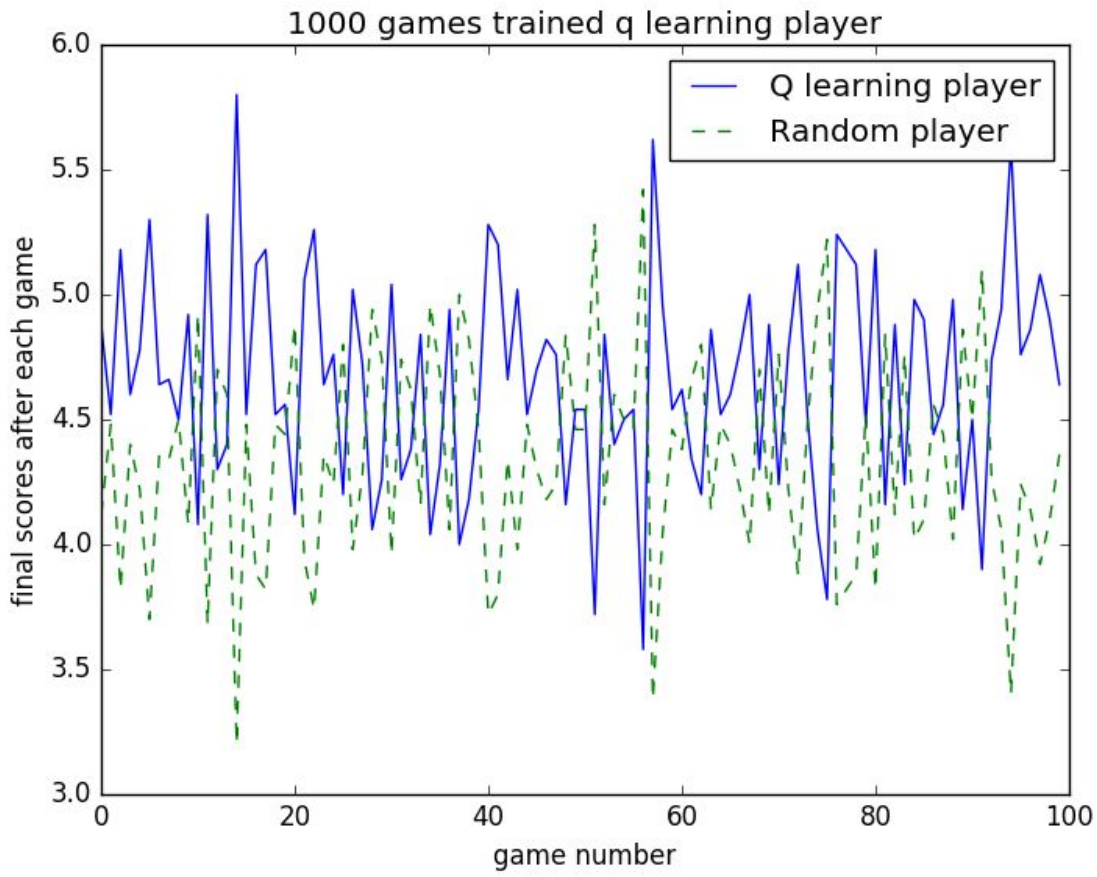


<u>1000 Training Games</u>

Q learning player win count : **2545**
Random player win count **: 2455**
Q learning/Random player win ratio: **1.0366598778**

Q learning player score sum : **22781**
Random player score sum **: 22219**
Q learning/Random player score sum ratio: **1.02529366758**

Observation: The q learning agent still doesn't seem to be much better than a random player

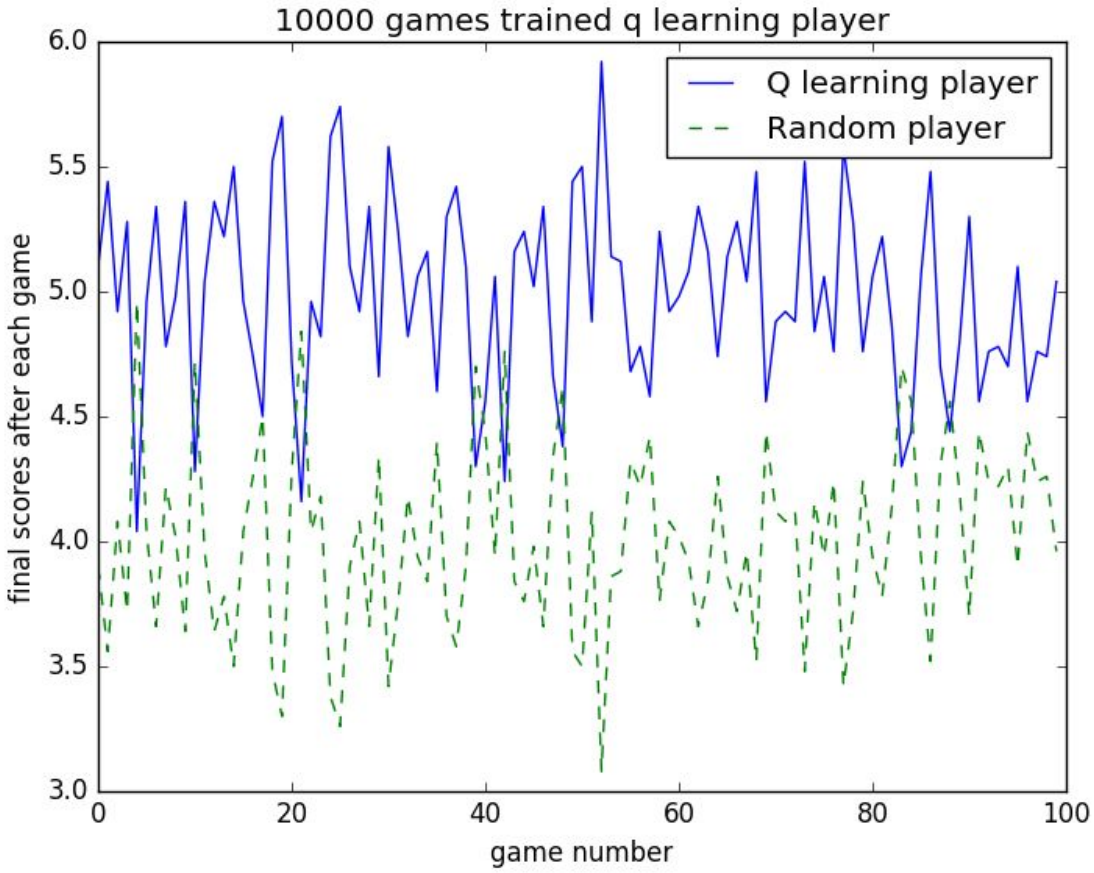


<u>10000 Training Games</u>

Q learning player win count : **2883**
Random player win count **: 2117**
Q learning/Random player win ratio: **1.36183278224**

Q learning player score sum : **24958**
Random player score sum **: 20042**
Q learning/Random player score sum ratio: **1.24528490171**

Observation: The q learning agent is visibly better than a random player



<u>100000 Training Games</u>

Q learning player win count : **3573**
Random player win count **: 1427**
Q learning/Random player win ratio: **2.50385423966**

Q learning player score sum : **28780**
Random player score sum **: 16220**
Q learning/Random player score sum ratio: **1.77435265105**

Observation: The q learning agent is much better than a random player

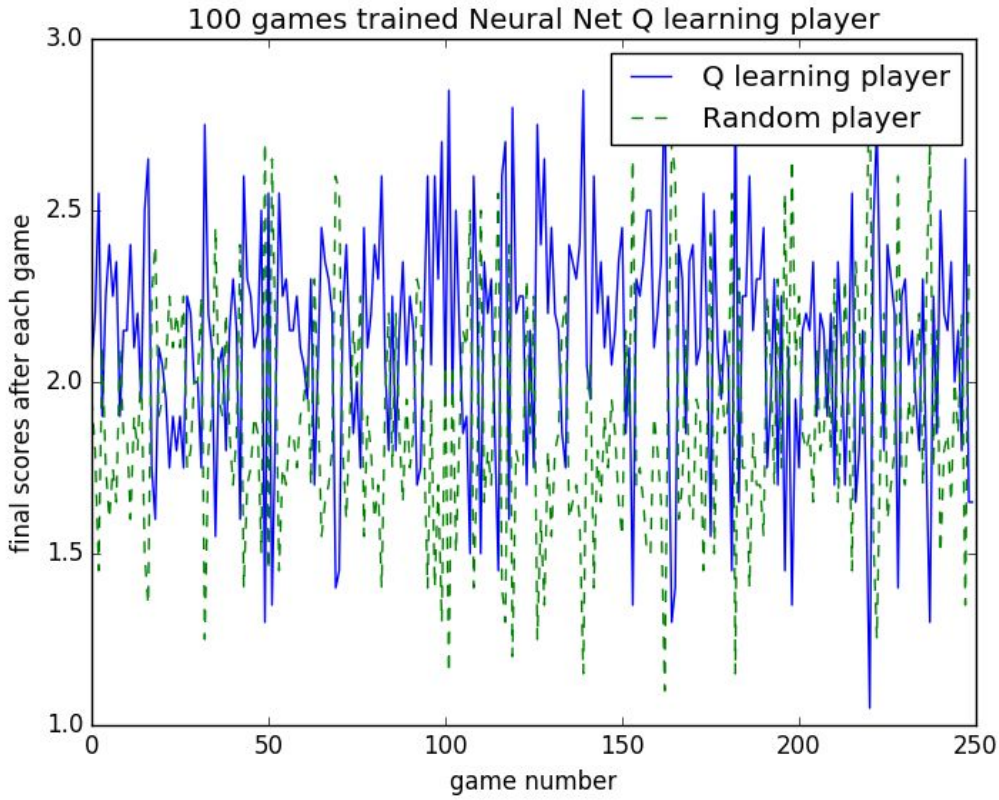
2x2 grid (Neural Network)

<u>100 Training Games</u>

Q learning player win count : **2246**
Random player win count **: 1722**
Q learning/Random player win ratio: **1.30429732869**

Q learning player score sum : **10581**
Random player score sum **: 9419**
Q learning/Random player score sum ratio: **1.12336766111**

Observation: The q learning agent is visibly better than a random player

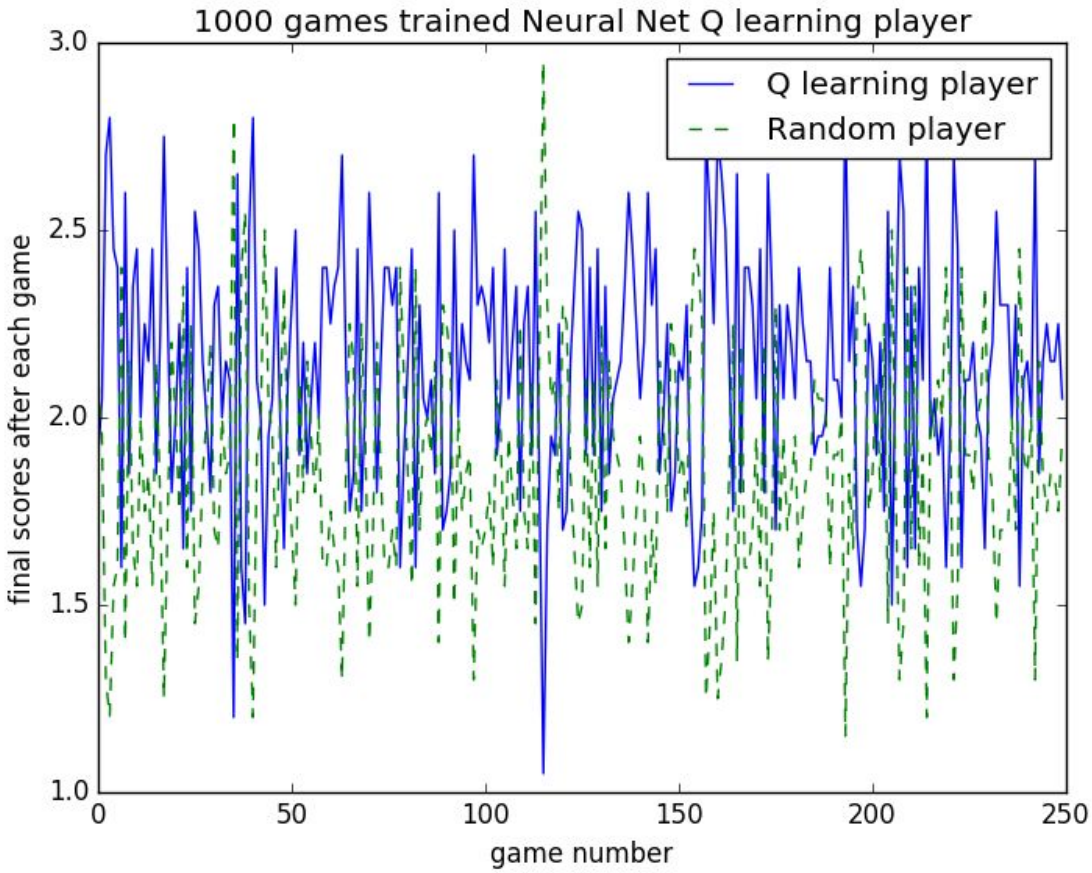


<u>1000 Training Games</u>

Q learning player win count : **2154**
Random player win count **: 2074**
Q learning/Random player win ratio: **1.03857280617**

Q learning player score sum : **10389**
Random player score sum **: 9611**
Q learning/Random player score sum ratio: **1.0809489127**

Observation: The q learning agent is better than a random player by a small margin

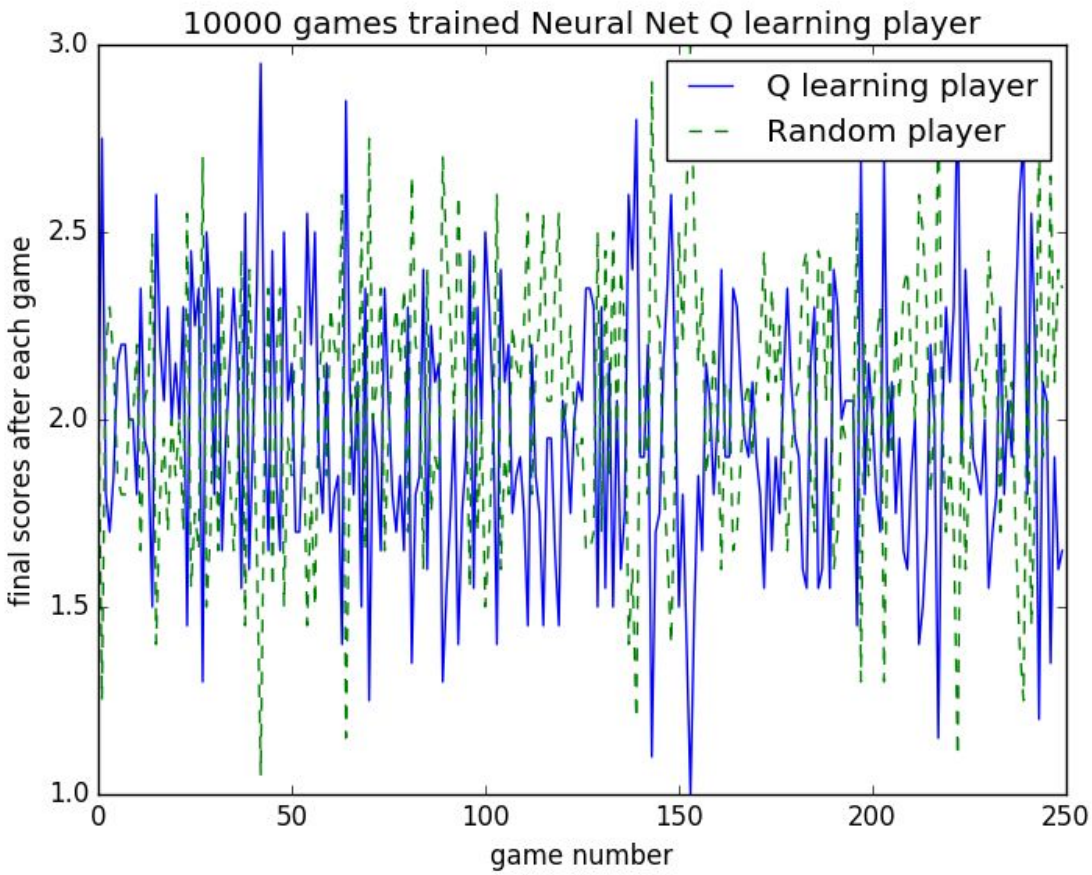


<u>10000 Training Games</u>

Q learning player win count : **2202**
Random player win count **: 2215**
Q learning/Random player win ratio: **0.9941309255**

Q learning player score sum : **9839**
Random player score sum **: 10161**
Q learning/Random player score sum ratio: **0.96831020568**

Observation: The q learning agent is no better than a random player.



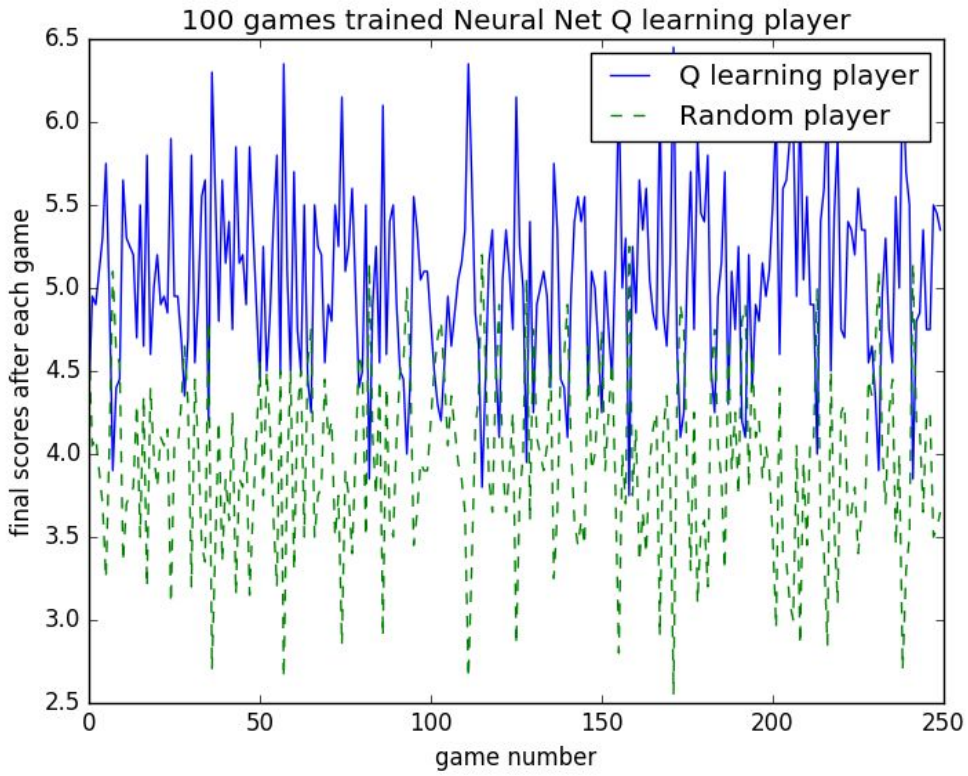
3x3 grid (Neural Network)

<u>100 Training Games</u>

Q learning player win count : **2698**
Random player win count **: 2302**
Q learning/Random player win ratio: **1.17202432667**

Q learning player score sum : **23685**
Random player score sum **: 21315**
Q learning/Random player score sum ratio: **1.11118930331**

Observation: The q learning agent is no better than a random player



<u>1000 Training Games</u>

Q learning player win count : **2844**

Random player win count **: 2156**

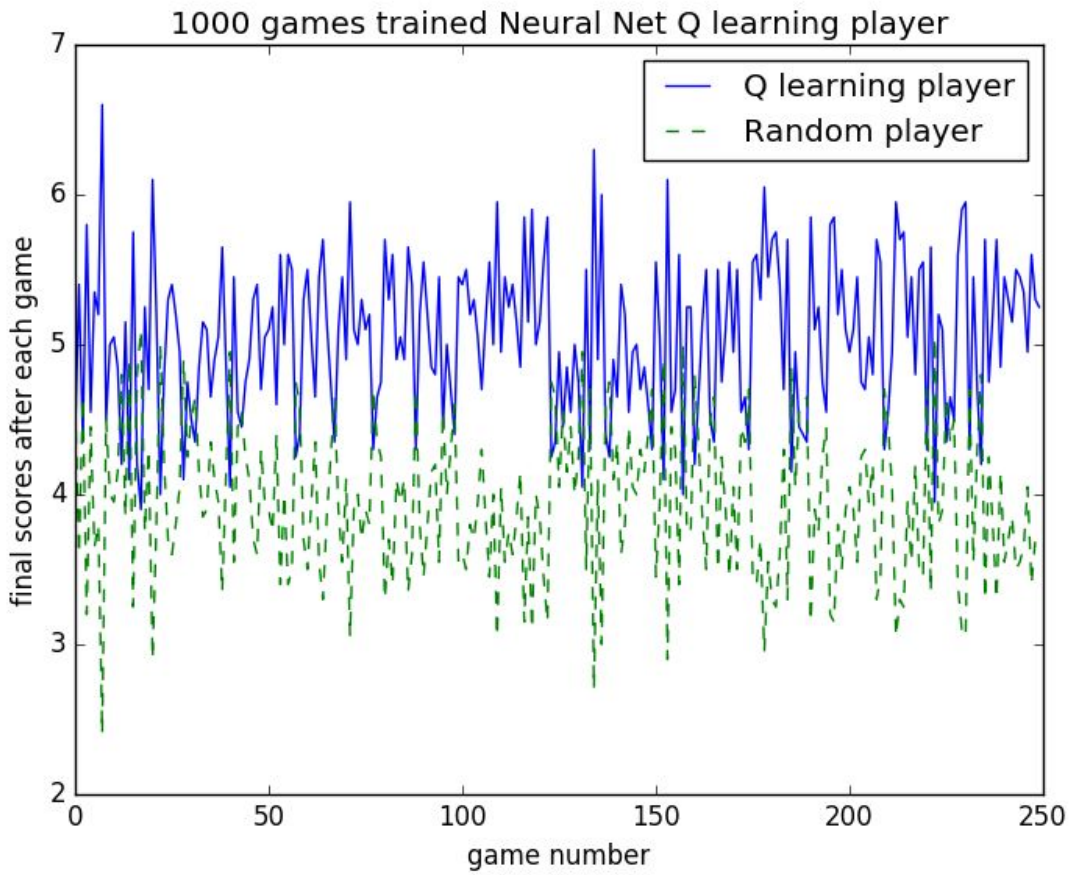
Q learning/Random player win ratio: **1.31910946197**

Q learning player score sum : **25251**

Random player score sum **: 19749**

Q learning/Random player score sum ratio: **1.27859638463**

Observation: The q learning agent is better than a random player by a small margin



<u>10000 Training Games</u>

Q learning player win count : **3545**

Random player win count **: 1325**

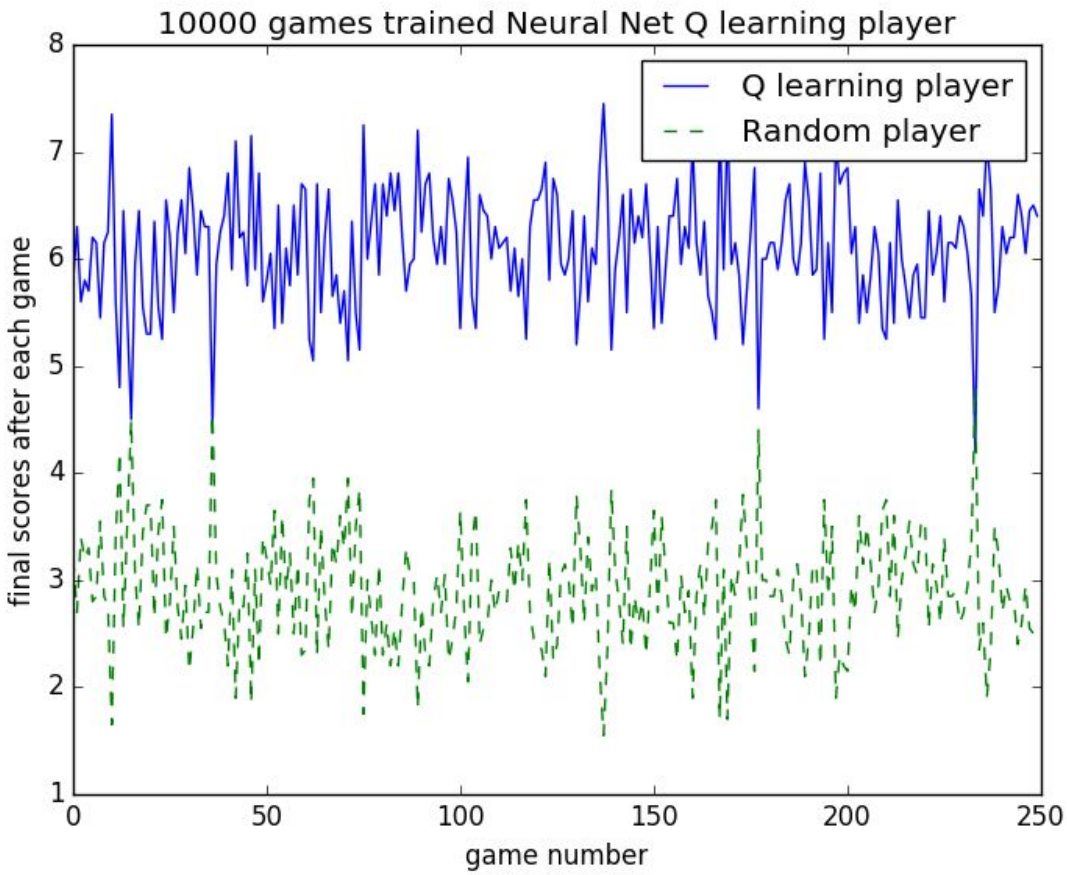
Q learning/Random player win ratio: **2.67547169811**

Q learning player score sum : **30483**

Random player score sum **: 14517**

Q learning/Random player score sum ratio: **2.09981401116**

Observation: The q learning agent is much better than a random player.



References

- NEURAL NETWORKS AND REINFORCEMENT LEARNING Presentation by Abhijit Gosavi
- https://gist.github.com/stephenroller/3163995