Deep Learning Carom Billiards with Reinforced Neural Networks

Presented by Mrinal Sourav towards the Machine Learning course at Northeastern University, Seattle – Spring 2018

1. Introduction

Carom Billiards is a simple looking game on a pool/billiards/snooker format but without pockets. Players are assigned a ball and take turns to strike that ball. The goal is to bounce off the players ball with two other balls within the same strike to get a score. On scoring, the player continues to get the next strike until they fail to score. The game continues for a finite number of rounds and the player with the maximum score wins the game.

The seeming simplicity of the game hides the complex game dynamics wherein reflection from the boundaries of the table and other balls play a key role. There is also the aspect of trying to win the next round while at the same time making it difficult for the other player if there is a miss. The main objective of the project is inspired by the automation of Atari game by Google DeepMind's Deep Q-learning. I want to train a Deep Neural Network to play a simulation of carom billiards and then observe "superhuman" strategies acquired, if any. If time permits, I would also change parameters of the game or scoring strategies and then observe how well the deep learning model can adapt to new/changing circumstances. Finally, I would also provide fixed circular obstacles on the board to train the model and observe if the model learns to avoid the obstacles or use them!

I hereby present a simple simulation of the game coded in python using **pygame**. For simplicity, some basic rules have been tweaked so that players are assigned the same "q-ball" and take turns to bounce it off the other two balls.

A q-learning approach will be implemented:

- The "state" would be defined by the position of the balls in the game display.
- The "action" will be defined in angles and discretized speeds of the q-ball.
- The "q-value", to be learned by the deep neural network model, would be represented by the rewards earned in each round (0 for no collision, 10 for collision with one ball and 100 for collision with both the other balls)

Although one popular approach to q-learning is to learn the probability distribution for the best action given a state, I would be using the alternate approach where a **state-action pair** is fed into the model as inputs and the **quality** associated with the action for the state will be learnt as the target for the model. This is done for the sole reason that the **action space for this problem is significantly higher than most Atari games referenced above**. Specifically, the action space includes each of the 360 angles for the q-ball and speeds ranging from 1 to 10; the total permutations of which is 3600. Most Atari games have an action space of "UP", "DOWN", "LEFT" and "RIGHT", which I believe is much easier for a model to converge to. Another reason this problem is interesting is its simplicity which can be built upon to test and observe Neural Network behavior for various cases. Perhaps, experiments like these may enable us to uncover laws if not theoretical explanations for "black-box" models.

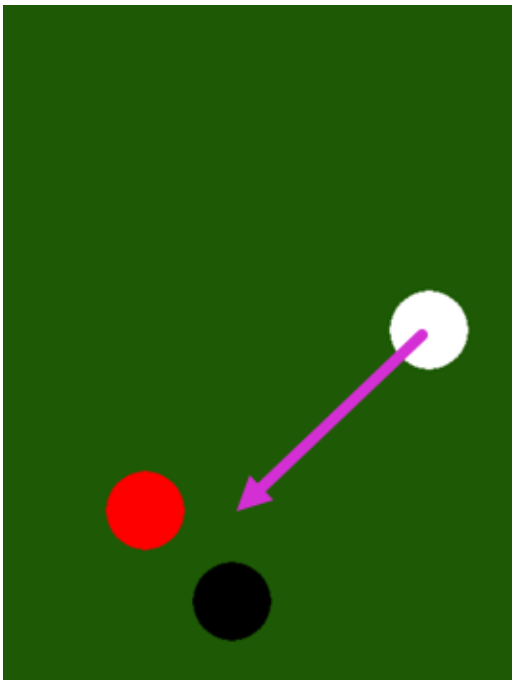
2. Problem Definition

2.1 Task Definition

The initial task was to create the game simulation itself and decide on simplified rules for this simulation. As per the screen-clip below, the main game area is modeled by the green board which can house 3 balls. The q_ball in white is the one users or the model can use to play the game by giving it an angle and a velocity. The game round starts and stops every time all three balls come to a stand-still. Within the game round, the player needs to achieve the maximum score/reward by collision of the balls. Rewards assigned are 0 for no collision, 10 for collision with one ball and 100 for collision with both other balls. So, the aim of the game is to use perfect reflection from ball collisions to bounce off one ball and then collide with the other.

Once we have the game simulation, we can then train a Machine Learning Model to select the best angle and velocity for the q_ball to earn the highest award within a game round by bouncing off both other balls.

For the q-learning of the model, we need to define the "state", "action" and "quality value".



The state here, is the set of x and y coordinates of each ball at the start of a game round. The action is the q_ball angle and speed to be chosen, and the quality value is the reward earned from collision of the balls. Although, q-learning is generally a classification task where given a state we classify the best possible action, I have considered an alternate approach as the action space consists of 3600 possible actions (360 angles and 10 speeds). Also, as the simulation has close to 700X600 possible coordinates (height and width of the green table), the total state space is a permutation of all possible locations each of the balls can take [~ (700X600) ^ 3 = ~ 74,088,000,000,000,000].

To simplify the q-learning approach, the **state and action pair are fed to the model as inputs and we train for the q-value as the output**. This is a regression task for the q_value. Once trained, we can input the model with all possible combinations of the action with a given state for it to predict the q-value of each action. The action with the maximum q_value (argmax) can then be selected as the action for the q_ball. Reinforcement can be done by noting which of the actions lead to the highest q_value for the state and then further training the model for that state-action pair and the actual reward.

The training for this problem needs to be done offline by generating training data from the simulation as online training would take a very long time. To achieve this, I have used a "para_simulator.py" code that simulates the math of the game without including "pygame".

2.2 Algorithmic Specifications

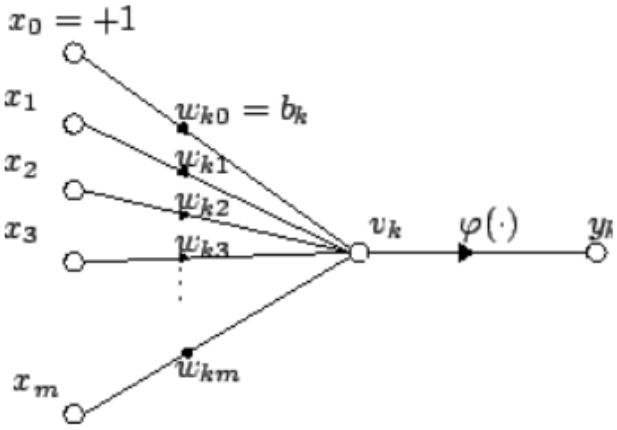
A Neural Network Model is chosen for the task as, given multiple states, there can be multiple coinciding angles which may give us the best reward. In other words, there is **a non-linear separation between the state-action pairs and the q_value to be regressed on**. Also, we need the model to estimate the "deeper" relationship between the game boundaries and the reward that can be earned by reflecting off the walls. Remember too, that the choice of the ball to collide with first is also not specified and this is left for the network to regress on q-values for either option given any state. The relatively simple environment coupled with a big action and state space, calls for a good generalizer which can learn the basic relation (hypotheses) from a small sample of the input space.

A Neural Network (hereto referred to as the "network"), is perhaps the best candidate model for the given task at hand. The flexibility to adapt to new data can also help us to add to the simulation and test the limits of how well a network can generalize and what counter-intuitive game plays it may implement.

The output of the *k*th neuron is:

$$y_k = \varphi\left(\sum_{j=0}^{m} w_{kj} x_j\right)$$

Where φ (phi) is the transfer function.



Artificial Neural Networks, as the name suggests are networks of interconnected neurons modeled on biological neurons first proposed by McCulloh-Pitts. Though Artificial Neurons have been shown to be far too simplistic when compared to its biological counterpart, it does inherit the essential ability to learn concepts from data with supervised learning.

The individual neuron aims to weight and sum inputs provided in order minimize the difference between the actual output and that sum (passed through a transfer function). In supervised learning known instances (input/output pairs) are provided to a model and the model is trained to map the relationship between them. A neuron looks at each such instance and adapts its weights by a small margin to converge to the relationship.

Logistic and Linear regression, Perceptron, and Winnow are models developed on the same principles with variations in the way the weights are updated. Each of these models have been established to perform well on tasks that have a linear separation between inputs and outputs. However, it is the "Network" in the Artificial Neural Network which contributes towards combining results from multiple linear separators and solve even non-linear classifications/regressions. The deeper the architecture of the network the more linearly separable concepts can be combined to provide the network with a deeper representation of the data generated by the game.

2.3 Expectations

The primary expectation from the network is that it learns to maximize its reward in the game as a player. The only context provided to it would be the x, y coordinates of each ball and I expect to observe the behaviors acquired by the network to maximize the score. By restricting the context of the game provided to the network we may expect the network to acquire counter-intuitive game plays that contrast with how a human would play.
By this experiment I wanted to answer questions like:

- Without specifying the locations of the walls (or any indication of their existence), would it be possible for the network to choose angles for the q_ball such that it uses the walls to secure the maximum reward?
- What would happen if we redesign rewards to, say, accommodate the distance travelled by the q_ball?
- What happens if we reward or penalize the network if it takes the longest path to both collisions?
- What happens when we add obstacles on the board? How well does the network adapt to changing the game parameters?
- If we negate the rewards, will a network learn to avoid collisions?

These and many other questions and creative situations could be tested by this experiment. The main goal is to provide a platform where such questions could be answered.

3. Experimental Methods and Results

3.1 Experimental Methodology

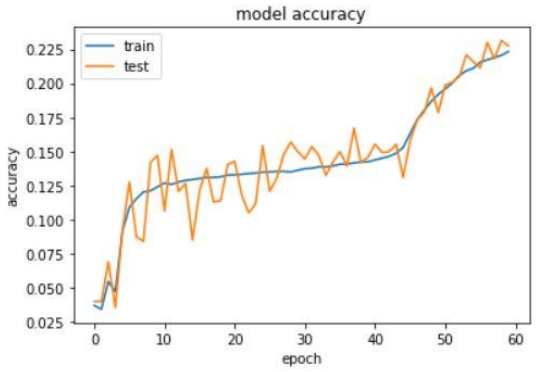
The first step towards training the network is to collect data. Because the input space to the network is big, we generate all the required data offline and train on them. A code which simulates just the math of the game is used to generate the required data for random states and actions.
One concern was the data imbalance for rewards. A larger ball size can even out the distribution as there would be a higher probability of collisions when the ball sizes are large given static walls. I first tried with ball size 70 and a small data of ~100,000 and found that a simple network could converge ever so slightly. I then proceeded to reduce the ball sizes to 40 and further to 30. Close to 6 million records were generated for ball size thirty and this became my go to training data set for further refining the network parameters and architecture. A Sequential model with dense layers provided with Keras on Tensorflow backend was used to build the network. Also, generating records and training would not take much time (~2 hours on my PC) for random data and that gave me a lot of room to get to the final topology:

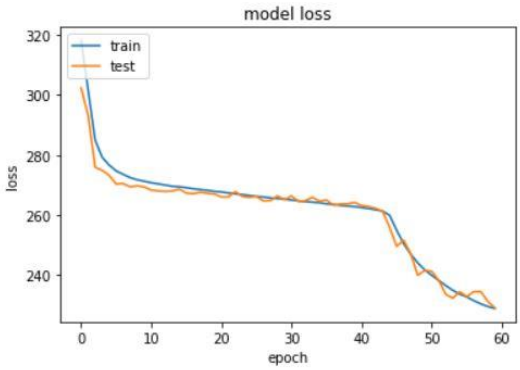
```
model = Sequential()
model.add(Dense(8, input_dim=8, activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(100, activation='relu'))
model.add(Dense(300, activation='relu'))
model.add(Dense(200, activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(20, activation='relu'))
model.add(Dense(5, activation='relu'))
model.add(Dense(1, activation='relu'))

model.compile(loss='mean_squared_error', optimizer='nadam', metrics=['accuracy'])
```

The 8 inputs are the positions of the balls and the angle and speed of the q_ball. A gradually widening network performed better during training than funnel architecture.

With the generated data I proceeded to tweak the network parameters for training (number of epochs, batch size etc.) and after many tries saw the following convergence:



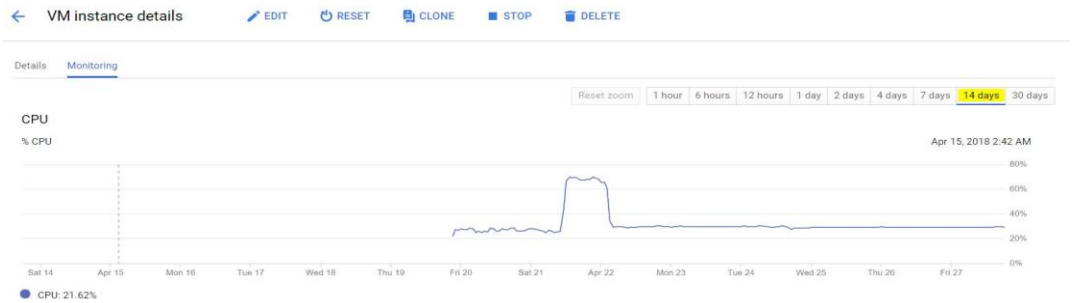


The training and validation split was chosen at 20 percent. Without dropout we can observe the network converge to a local optimum and stall a bit till the 40th epoch but soon after it breaks the local optima to converge better.

Further versions with one Dropout layer after the widest layer converged more smoothly towards the same loss seen here i.e. there was no stall or local optimum observed. This model was trained to 110 epochs and saved to a file called “ball_size_30_light_v2.h5”. This is the best performing model so far.

For the reinforcement this same model was used by the “reinforcer.py” to generate the data. We now need the model to generating data for reinforcement with q-learning. It is the model that is required to select actions to further train on positive rewards. In contrast, the “para_simulator.py” selects actions randomly.

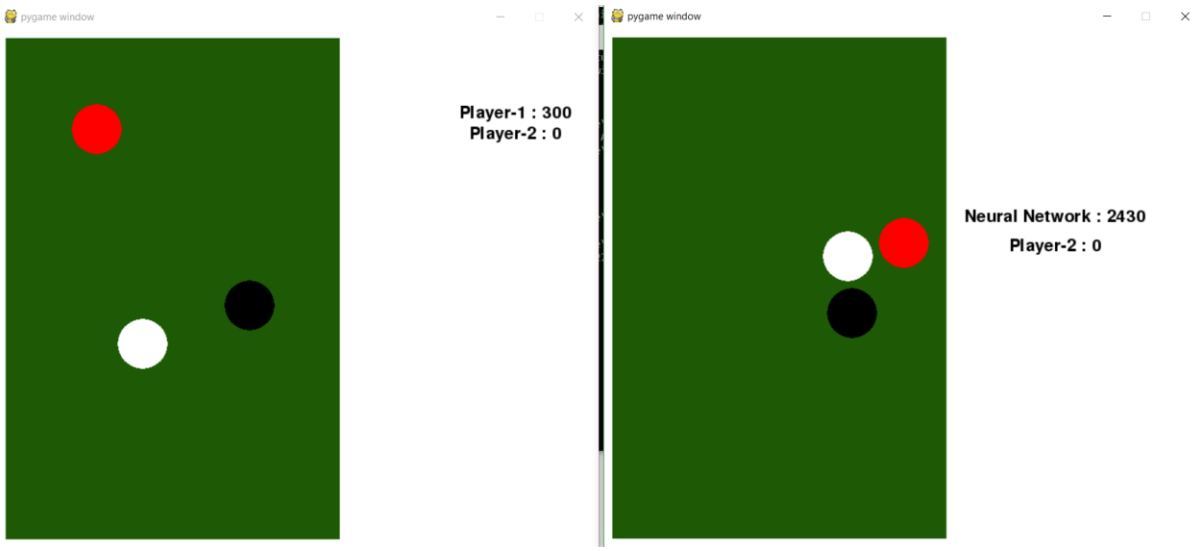
Because it is the network that generates data for reinforcement, it takes much longer to create a significant amount of positive training for reinforcement. It has been more than a week for my Google Compute engine I left to generate 1,000,000 records and it is still in progress! As such I have not been able to proceed with reinforcement for now.



3.2 Results and Analysis

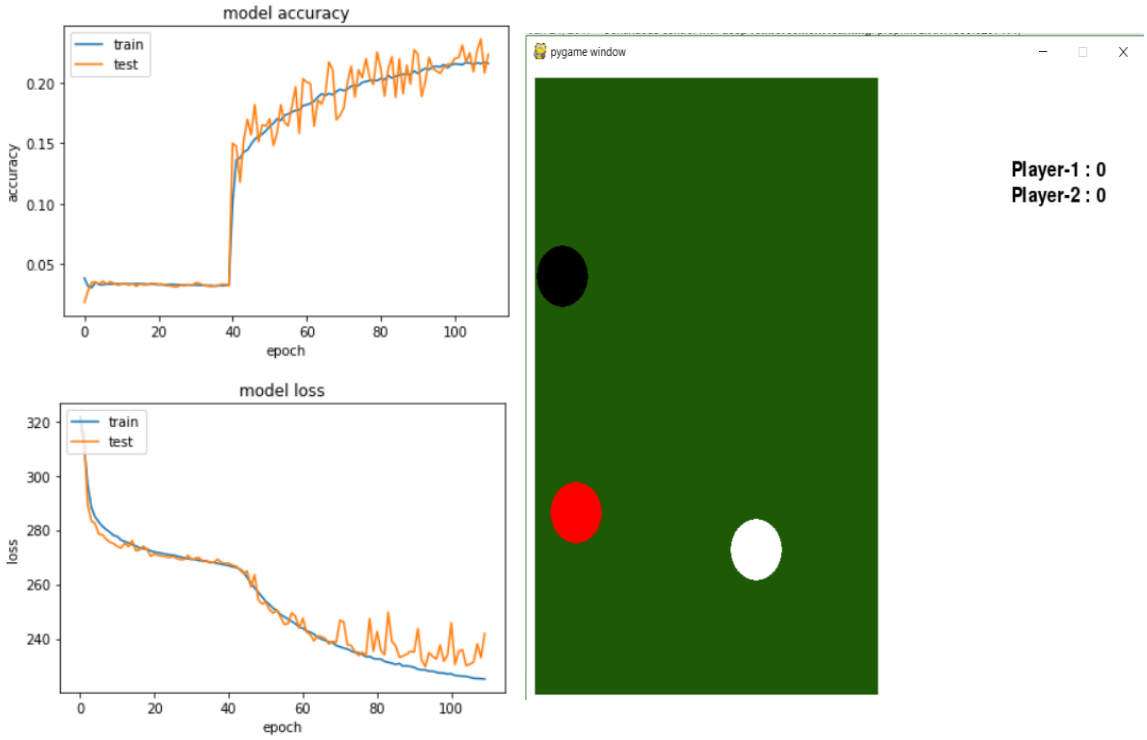
After training the network I proceeded to write another python code to test it called “nn_test.py”. This code is nothing but the game environment in which I load the model and use it to generate the action for a given state as soon as the balls are static. I also coded a random action generator to compare the results. The results are appealing even visually as one can see that the “nn_test.py” program manages to score a 10 with close to 99% accuracy and quite often also scores a 100.

I ran the network side by side with the random data generator to compare the total score accumulated after ~1 minute to find:



Another interesting result of the training is that the network almost always chooses a high speed for the q_ball. The explanation for this is that, according to the data, the higher the speed, the more area is covered by the q_ball, thereby increasing the probability of collisions.

Further, I also created a training dataset with negated rewards. So, now there is a -10 reward for collision with one ball and a -100 reward on collision with both balls. I data set contained ~3.5 million records and ReLU was removed from the last layer of the model. The training loss and accuracy look like:



As expected, the model now tries to avoid collisions instead (To the right: score after ~5 minutes of gameplay). This neural network can be tested by modifying the nn_test.py code to only contain speed range from 5 to 10 and loading the model named "reinforced_ball_size_30_neg.h5". [Or use nn_test_neg.py]

3.3 Discussion

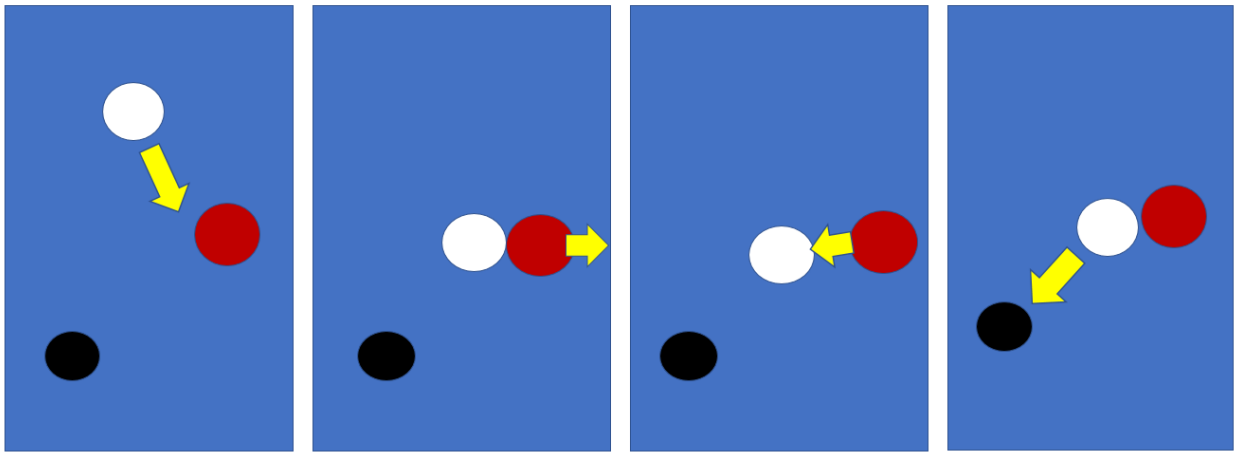
The qualitative aspects of how the network plays outweigh the quantitative results shown above. The network does make extensive use of the boundaries even though their presence was not encoded in the state on which it was trained. The network also plays rebound shots that seem counterintuitive to how a human would play the game. Since these results were achieved on a limited time with limited resources there can be vast improvements that can be made.

For instance, the data generated is highly imbalanced and skewed towards 0 reward as the probability of acquiring 100 from random states and actions are far less compared to scoring 0. This is perhaps the biggest hurdle for the network which prevents it from distinguishing between q_values for actions. The mean squared error loss for the data labeled 0, 10 and 100 is ~206 at its lowest during training over ~6,000,000 records. This can be improved by randomly skipping actions with 0 reward or removing them from the training data. However, there is the risk of underfitting with a small amount of data and it was hard to judge the quality of the data being trained on.

Some of the math of the game have been left imperfect. The angle acquired by balls after collision is erroneous for some angles of collision. This makes it even harder for a human to play as sometimes the angle after bounce is unpredictable. The network seems to have little problem as the training sample seems good enough to cover those edge cases. Some shots

played by the network correctly predict even erroneous angles after collision. Improving the math on the game may even out the distribution for both the network and humans.

Some shots where the q_ball nearly misses the second collision demonstrate the ability of the network to generalize. I feel these shots are selected even though the exact instance wasn't presented during training. One very surprising shot played by the network was to use two bounces from the first ball to then bounce with the second! I noticed this only once so far and it looked something like:



4. Theoretical Evaluation and Analysis

Even when results from reinforcement is not possible to show at this moment, with the results achieved so far, we may make some assumptions about the potential for further experiments. To answer each of the expected questions above it would take some time to generate the data for each case and test them but even then, we can see that they are quite doable and the network does perform well to reveal some interesting phenomenon. The proposed method of reinforcement too is simple as the states are presented to the network as independent. To model aspects of multiplayer game play and the fact that the next round needs to be won to acquire a higher total reward, we need to incorporate the aspect of reinforcement learning where the future states are dependent on past and current states and rewards can accumulate over game rounds. With state dependence, we may further enhance our scope of questions to ask from the experiment and learn deeper insights into the mechanisms of reinforcement.

Regardless, the claim the neural networks are best suited for this task still holds. As shown earlier, the total permutations of state are ~ 74,088,000,000,000,000. This combined with the total possible actions i.e. 3600, we have 266,716,800,000,000,000,000. Even though the input space is this big, the number of hypotheses that can generate good results are still comparatively small. There are only a small set of skills/concepts one needs to develop to score 100 no matter what the state. This calls for a model that can generalize well from small subsets of the total input space and the only good candidate stands to be a Neural Network.

5. Related Work

Although the deep-q learning approach developed by Deep Mind was an inspiration to this work my approach and methodology deviated slightly to theirs due the action space. Most of the ideas for this was developed from this post in Deep Learning 4J blog. Notice how the discussion here involves state action pairs unlike the approach used by Deep Mind. Since, this experiment was inspired by the real-world simulation of carom billiards I did not find much related work on this specific game.

6. Future Work

As noted earlier there is a big room for improvements. From the correction of the math to incorporating state dependence and reinforcement learning, one can use this same platform to demonstrate what can be done in the case of relatively big action space. Perhaps, we may even pitch two neural networks to play each other as adversaries to observe more interesting game plays. Also, adding obstacles may turn out to be like what happened with the walls in that the network may learn to use them unlike human players who would stick to avoid them for simplicity.

7. Conclusion

Simulating the game from scratch was a great learning experience: I had never developed a game before. Being able to see the results from the model was just as rewarding. I am currently going to progress with the work as much as I can. For now, most of the conclusions have already been mentioned in previous sections of this report along with discussions about the results.

It was surprising to find how easy it is to go from linearly separable problem to non-linear separation while modelling aspects from the real world. All I did was put some balls in a rectangle and defined a relationship between rewards and ball collisions. This stands to show the potential of optimization problems like genetic algorithms and neural networks as opposed to other models regardless of the intuition or its lack thereof. Just like the laws of gravity or motion are distilled from observation instead of intuition, perhaps one approach to probe Neural Networks would be to try and define its laws and discover patterns of behavior through experiments such as this.