CS 5891 Final Report

ANNs and RNNs to Predict NBA Gambling Outcomes

By Chris Hoogenboom

1. Introduction

The National Basketball Association, or NBA, is an American professional basketball league that brings in millions of fans and billions of dollars every year. In recent years data analytics have become more and more important to winning. Almost every team has an analytics department, and during games data is collected about every player on the court 25 times per second. NBA teams are not the only ones profiting from data science's applications to sports, however. Sportsbooks, the organizations that take bets on sports, also use data science to improve their predictions of games.

Given the massive amount of NBA data available, it makes sense that data science has become increasingly important to industries that value knowledge about player and team performances. In particular, people have found many different applications for machine learning algorithms for processing NBA data. Data scientists use methods such as random decision tree forests, collaborative filtering, neural networks, and Long Short-Term Memory networks to make predictions and evaluations based on NBA data.

The prospect of a machine learning algorithm that can outperform the ones that sports books is certainly an enticing one. Sportsbooks normally aim to take about 4.5% of all money gambled by taking a little less than 10% of all winnings, but if an algorithm could be designed that is good enough to beat that margin, then it could be considered an infinite money generator. Sportsbooks dedicate billions of dollars to making accurate predictions, so I cannot expect to beat them with a method of my own design, however the problem certainly piqued my curiosity.

First I was just interested in the ability of an Artificial Neural Network (ANN) to predict simple NBA outcomes. After doing some work with the data, I noticed that each teams results formed a time series. This led me to also try using a Long Short-Term Memory Network (LSTM) for predicting game outcomes. Then, after comparing the performance of both the LSTM and ANN, I wanted to see if I could design an ANN to classify betting opportunities into one of three categories: bet outcome A, bet outcome B, or do not bet.

1.1 Betting Background

To understand the rest of this project, it is necessary to have some background information on betting in the NBA. There are 3 basic types of bets: moneylines, spreads, and over/unders.

Moneyline bets: A bet on the win/loss outcome of a game that pays out more when an underdog wins and less when a favorite wins. The favored team will have negative odds

-xxx, which means that a winning bet of $xxx will return $100 dollars in winnings. The underdog team will have positive odds +xxx, which means that a winning bet on $100 will return $xxx in winnings.

Spread Bets: A bet on the score difference between teams. A bet of +/-x.x on team A pays out if team A's score +/-x.x beats is higher than the opponents score. The odds of winning this bet is 50%*.

Over/Under(O/U) Bets: A bet on the total score of the game. Given a total score xxx, you can choose to bet that the total score of the game will be under or over that number. The odds of winning this bet is 50%*.

*recall that sportsbooks take almost 10% of winnings

2. The Data

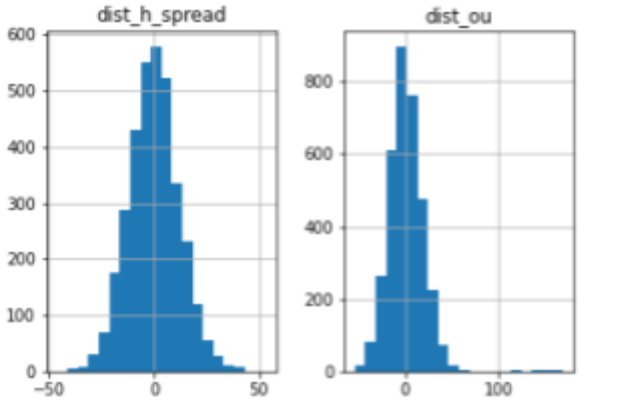
The NBA data and sportsbook data was downloaded from the NBA Historical Stats and Betting Data dataset on Kaggle (https://www.kaggle.com/ehallmar/nba-historical-stats-and-betting-data?select=nba_betting_spread.csv). This dataset contained a table with stats like points, assists, and rebounds as well as more advanced impact stats for each team in each game, as well as a table for each individual player's stats in each game. It also included tables for different betting types (spreads, moneylines, and over/unders) in each game. Data for each game in each table is indexed by game id so that it could be easily joined.

2.1 Feature Set Preparation

Once all the data tables were properly joined, it still needed more processing before it was ready to be used by a neural network. I wanted to use statistics from a team's most recent games as the inputs to the networks. To do this, I used Pandas sorting and shifting functions to very efficiently join data from each team's most recent 5 games. This provided me with 1538 input features to work with.

2.2 Label Preparation

To classify games into the categories bet outcome A, bet outcome B, and don't bet, I needed to prepare categorical labels for the data. To classify games into these categories for spread and over/under bets, I built a normal distribution of the differences between the actual game spread, and then used the z-score of each game to classify outcomes that were outside of the sportsbook's margin of error into outcomes to bet on, and outcomes that were within the margin into games not to bet on. Figure 2.2.1 shows the distribution for both spreads and over unders.



Spread distribution Over/Under Distribution

Figure 2.2.1

3. Methods

I used two different machine learning techniques to predict NBA game outcomes: Artificial Neural Networks (ANNs) and Long Short-Term Memory networks (LSTMs). I then used ANNs to classify games into one of the three betting categories. I used a random decision tree forest as a baseline to compare my methods to.

3.1 Artificial Neural Networks (ANNs)

Artificial neural networks (ANNs), also known as feed forward networks or multilayer perceptrons, use information learned from training examples in order to make predictions of an output given an input. ANNs are made up of layers of nodes, where each node in a layer takes in one or more inputs each multiplied by a weight, and then produces an output based on applying an activation function to the sum of the weighted inputs. The nodes in each layer take all of the outputs of the previous layer as inputs, with the first layer taking in some feature set as an input, and the last layer outputting the models prediction of the target variable based on the inputs. Figure 3.1.1 shows the structure of an ANN.

ANNs are trained by feeding in example input data for which the values of the target variables are known. Then, the difference in the output of the model and the real value of the target variable is evaluated using a loss function as the metric. Then the gradient of the loss function with respect to the weights is used to adjust the weights in a process called gradient descent. With enough training examples using properly predictive

inputs, we can train the weights of the network so that it can make very accurate predictions.

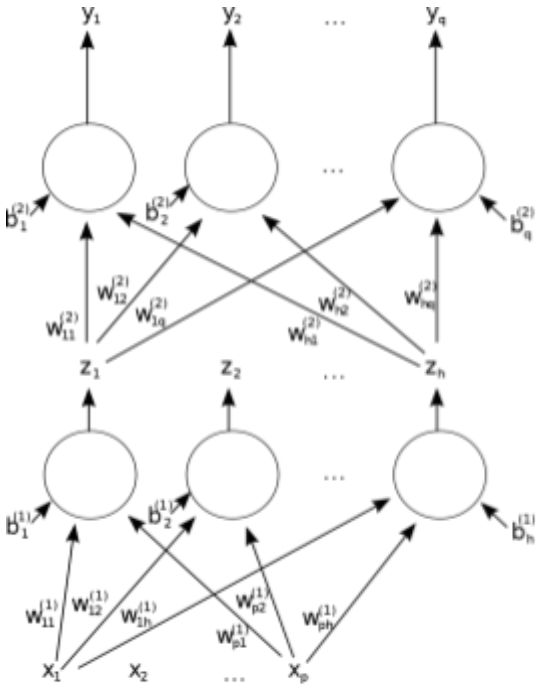


Figure 3.1.1

3.2 Long Short-Term Memory networks (LSTMs)

Recurrent Neural Networks (RNNs) are similar to ANNs, however RNNs have the ability to maintain memory based on recent inputs. Thus they can be more powerful than ANNs when fed sequential data. Long Short-Term Memory networks (LSTMs) are a type of RNN that can preserve both long term and short term memory, making them wonderfully powerful for taking on problems with sequential data.

LSTMs work by keeping a hidden state and 3 different gates that determine what data is kept and what data is forgotten. The input gate determines which values get to be stored in long term memory, the forget gate determines what data is forgotten from long term memory, and the output gate produces the output and the new hidden state. Figure 3.2.1 is a diagram of an LSTM cell.

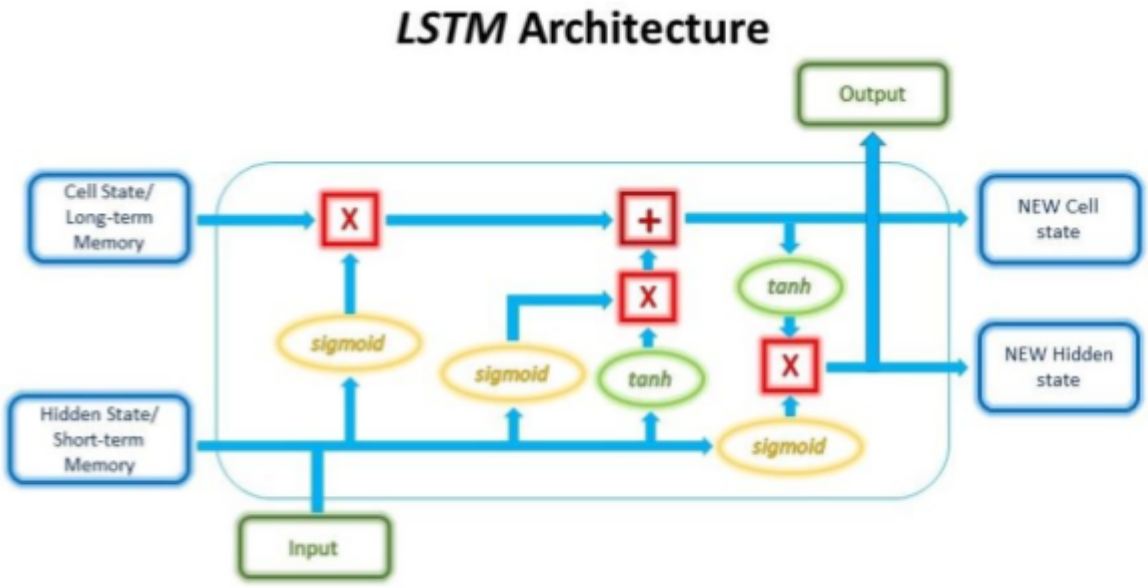


Figure 3.2.1

3.3 Random Forest

The random forest algorithm uses a series of decision trees, each trained separately, to come to a consensus that is used as the output of the model. Random forests generally don't perform as well as neural networks on these types of tasks, but still can make useful predictions.

4. The Models

In this section I give details on the specific models and their performance for each problem. I created two different ANNs for game outcome prediction, one that used all 1538 features in my feature set, and another that used the 100 features most correlated with the target variable. I also created an LSTM for game outcome prediction. I used an ANN for the betting classification problems.

	ANN1	ANN2	LSTM avg.
Loss	0.22583	0.23672	0.21725
Accuracy	62.3%	60.3%	57.1%

4.1 Random Forest for game outcome prediction and betting classification

The random forest baseline method had an accuracy of 53% when predicting game outcomes, and loses about 4.5% of money bet on both over/unders and spreads.

4.2 ANNs for game outcome prediction

The game outcome prediction problem is a binary classification problem, so each of these networks should have an outcome of 0 or 1. The best performing ANN for this task had the below configuration:

```
   (0): Linear(in_features=1544, out_features=463, bias=True)
   (1): Linear(in_features=463, out_features=138, bias=True)
   (2): Linear(in_features=138, out_features=41, bias=True)
   (3): Linear(in_features=41, out_features=12, bias=True)
   (4): Linear(in_features=12, out_features=3, bias=True)
  (out): Linear(in_features=3, out_features=1, bias=True)
(dropout): Dropout(p=0.4, inplace=False))
```

This ANN was trained using the Adam optimizer and mean squared error loss in 21 epochs after early stopping with learning rate of 0.001. The network used ReLU activation functions between each layer. You can find the performance results in figure 4.1 under ANN1.

I tried using another ANN to tackle this problem, but instead of using all 1538 of the available features, I used sklearn's feature_selection library to select the 100 best features for learning the problem. I call this network ANN2, and the structure for the network is below.

```
(0): Linear(in_features=100, out_features=50, bias=True)
   (1): Linear(in_features=50, out_features=25, bias=True)
   (2): Linear(in_features=25, out_features=12, bias=True)
   (3): Linear(in_features=12, out_features=6, bias=True)
   (4): Linear(in_features=6, out_features=3, bias=True)
  (out): Linear(in_features=3, out_features=1, bias=True)
(dropout): Dropout(p=0.4, inplace=False)
```

ANN2 was trained using the Adam optimizer and mean squared error loss in 38 epochs after early stopping with a learning rate of 0.001. The activation function was ReLU. The performance of this network is in figure 4.1 under ANN2.

4.3 LSTMs for game outcome prediction

The LSTM used for game outcome prediction used the 1000 best input features as determined by sklearns selectKBest. The LSTM network architecture is detailed below:

```
LSTM1(
  (lstm): LSTM(1000, 1, batch_first=True)
  (fc_1): Linear(in_features=1, out_features=128, bias=True)
  (fc): Linear(in_features=128, out_features=1, bias=True)
  (relu): ReLU())
```

The LSTM network has 1 hidden feature and was trained using mean squared error loss and the Adam optimizer, with ReLU as the activation function. This network trained in 5000 epochs with a learning rate of 0.0001. I trained LSTMs on data for each team, so

30 LSTMs were trained in total. The data in figure 4.1 represents the average performances across all 30 LSTMs.

4.4 ANN for Betting Classification

The betting classification problem is a multiclass classification problem. Each class was stored as a one-hot vector for the network to predict. I built one network for classifying spread bets and another for classifying over/under bets. Each one had the same structure so I will just describe the general model. The architecture of the classification network is below:

```
(0): Linear(in_features=100, out_features=130, bias=True)
   (1): Linear(in_features=130, out_features=150, bias=True)
   (2): Linear(in_features=150, out_features=130, bias=True)
   (3): Linear(in_features=130, out_features=100, bias=True)
   (4): Linear(in_features=100, out_features=50, bias=True)
   (5): Linear(in_features=50, out_features=20, bias=True)
 )
 (out): Linear(in_features=20, out_features=3, bias=True)
 (dropout): Dropout(p=0.4, inplace=False)
```

These networks trained in 100 epochs with a learning rate of 0.0001. They used cross entropy loss and Adam optimization, and there was a ReLU activation function between each layer. The network for classifying spreads had an accuracy of 39.49% and the network for classifying over/unders had an accuracy of 43.12%. Testing these networks revealed that the spread network loses 4.3% of all money bet and over/under network loses 4.0% of all money bet.

5. Conclusions

As expected, none of my methods were able to profit off of gambling. However, I was pleased that the ANNs were able to beat the random forest by 10% accuracy when classifying game outcomes, and the LSTM outperformed the random forest by almost 5% accuracy. Though I was disappointed by the performance of the LSTM, I believe that given more time to tweak the network parameters the performance of the LSTM could reach at least the same accuracy as the ANNs.

The most encouraging results from this work is the fact that my networks were able to lose just 4.3% of money bet on spreads and 4.0% of money bet on over/unders. The expected outcome of placing a bet is a loss of 4.5%, so these results indicate that my network's ability to predict game outcomes is comparable to that of sportsbooks.

Going forward, I would like to explore feature selection in greater detail, as the possible feature sets for this problem are huge but a lot of data has little correlation with the target variables. I would also like to explore using RNNs to process play-by-play data to make better predictions, using a custom loss function to classify moneyline bets, and using a regression network to predict game scores, and then using the z-scores of the

predicted scores to classify gambling outcomes. The most important conclusion however is that you SHOULD NOT GAMBLE