Predicting the Outcomes of NBA Games Using Deep Learning

By: Daniel Malerevich and Edan Davidai

Introduction

In the world of sports, the question of which factors lead to team success is one that is of great interest to many, including coaches, team managers, and bettors. As fans of basketball ourselves, we decided that it would be interesting to use this project to explore which factors are key to making a team great, and also to see how competitive neural networks are in this field when compared to other methods in machine learning. We also read that predicting NBA games is notoriously difficult, due to small sample size and high variance. We thought it would be interesting to attempt to solve this problem where others had mixed success.

Our final goal is to create a product which, when given a date and two teams, is able to extract the necessary data from our sources and accurately predict the team that won. We would also like to present a variable approximating how certain this team's victory is according to our model. Using this we could not only predict the outcomes of games on the schedule, but also the outcomes of hypothetical matchups which never occurred.

In order to reach this goal we came up with multiple different methods for predicting team success:

1. One approach was to use the advanced statistics of the players on each team (averages and in the last games) to predict which team would win.
2. The second was to use the statistics of both teams from that same season, including win percentages and counting stats.

The rationale behind these approaches and the conclusions drawn from them will be explored later on.

Prep Work

We began the project by creating a simple classifier that guessed that the team that would win is the one with the highest win percentage in that season. This would give us a general starting benchmark. After writing this code we saw that it achieved a 62.9% success rate in predicting who won a game, and we set this as our first goal to reach.

After this, we needed to find and extract data for our needs that was easily accessible. We found that there is a staggering amount of data online regarding the NBA, but that it is made relatively difficult to access in large quantities. This began a long process of data extraction, during which we drew mainly from online spreadsheets we found and from the nba_api interface, which (rather inefficiently) draws data from the NBA's official website. Due to the fact that in the end we are dealing with only a couple hundred thousand statistics, we managed to perform this

extraction without use of university resources, it simply required some workarounds to get past the uncomfortable methods we had at our disposal.

Once we knew what data we had, we began to outline a general plan for training our networks. We first decided that we would like our labels to represent a ‘level of certainty’ for winning the game. We decided to represent this certainty using the final score of the game, meaning if a team won by more, then it was a more ‘certain’ victory. This method is not perfect by any means, but it was good enough for our needs. The label we took in the end was linear between 0 and 1, with 1 meaning the away team won by more than x points (to be determined later) and 0 meaning the home team won by more than x points.

Now that we had our labels, we needed to decide which loss functions we were going to use. We decided on trying mse (mean squared error) and mae (mean absolute error) and seeing which of them would give us the best results during our trials. We would also end up trying binary cross entropy, with our loss function being zero-one loss, but this turned out to be worse.

Building the Network

After getting our data, labels and loss in order, we began a series of attempts to see which data would give us the best results. We decided to use the Keras module when building our network, because it is a convenient and comfortable way to build networks. We decided on a list of hyperparameters with which we wanted to define our network, (num. of layers, nodes in each layer, optimizer, batch size, etc.) In each attempt, our training process consisted mainly of trying to balance between these parameters in order to find a network that can learn our data most effectively without overfitting.

The way we did this in each attempt was by first training on a network with no regularization. This was done to make sure our model is complex enough, and can overfit on the training set. We then examined the resulting graph (including the validation loss) and used it to tweak the network (adding/removing layers, adding regularization, etc.). We tried many types of regularization, including l1, l2, drop out, and decreasing the batch size. This process would continue until we reached a network configuration that both learns and generalizes well (meaning relatively low training and validation loss). Once we had this configuration, we examined it’s loss graph and took the weights of the model at the epoch where we felt overfitting began.

In each attempt, we tried different deep learning approaches that were relevant to that specific data, which we will expand upon later.

ATTEMPT 1: Player Data

Input Data Used:

In the NBA, oftentimes a single good player can be the only difference between a championship team and a bad one. There have been many cases in which a superstar player or two have joined a struggling team and managed to completely turn it around. Due to this, as opposed to many other sports, the level of a team's best players seems to be one of the biggest factors in predicting team success. This brought us to a conclusion that perhaps team statistics are less relevant to our cause, and that we need to focus more on individual players in our research. To this end, we decided to begin by attempting to predict games based solely on recent data on each team's best players that are playing in that game. This gives an added advantage of taking into account injuries and players that aren't participating in that game.

For each game, we took each team and ordered its players based on Player Efficiency Rating (PER) for that season, an all encompassing stat that is a good estimate of who is most impactful on each team. We then took the top players from each team (we tried top 3,5,7 and 9) and gave their stats from previous games as the input. This was structured in a way in which the home team's players always came before the away team, due to the fact that home court advantage is a relevant factor in basketball (The home team wins around 60% of the time). Ordering it this way allows the network to factor in this advantage. After trying many combinations and researching, the data we settled on using was:

FGA (field goal attempts)	FGM (field goals made)
FG3A (3 point attempts)	FG3M (3 points made)
FTA (free throw attempts)	FTM (free throws made)
OREB (offensive rebounds)	DREB (defensive rebounds)
AST (assists)	STL (steals)
BLK (blocks)	TO (turnovers)

We averaged these stats for each player from the previous 8 games and this was our input to the network.

Techniques Used:

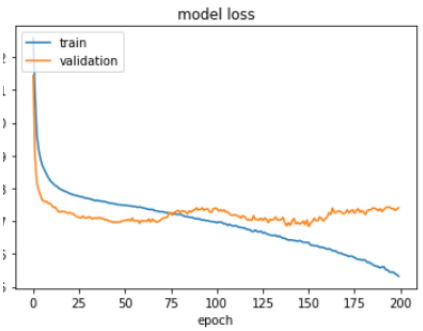
Our training process on this data was identical to what is written in the section 'Building the Network'. In addition to this process, we also decided that it might be effective in this case to use a convolutional neural network. This is because each input is divided into individual players, but in a fully connected network this does not translate. If we use a CNN, the stats of each player are passed through the same filter, and thus our first convolution layer will help weigh each player correctly. As we predicted, this approach did help, and our results improved.

We also tried adding a second convolution layer that was meant to differentiate between the two teams, but this worsened the results, and so we decided to get rid of it.

Results:

Test ZO Loss: `0.6192604632263308`

Graph and model parameters:




```
------------current model------------
CNN=True
TRAINING_FILE_NAME='player_training_9_8_25.csv'
FEATURES_LEN=234
POOLING='average'
REGULARIZATION=<keras.regularizers.L2 object at 0x7f8ba4376f90>
REGULARIZATION_CONST=0
DROPOUT=None
BATCH_NORMALIZATION=False
EPOCHS=200
SGD_EPOCHS=10
BATCH_SIZE=32
FILTERS1=7
FILTERS2=None
CNN_DENSE1=50
CNN_DENSE2=50
VALIDATION_SPLIT=0.1
OPTIMIZER='adam'
LOSS_FN='mean_squared_error'
-------------------------------------
```

Conclusions:

Despite what we initially thought, giving only player data, no matter how many players deep we went, was not able to give a good enough approximation of team success. It seems that factors such as team chemistry and players playing well together are too significant to ignore. This result shows us that individual players are of course a factor in winning games, but they alone do not necessarily make a team great.

In addition, from this attempt we concluded that player data is simply too volatile to be consistently reliable in our predictions. Players alone have high variance from game to game, and in order to get better results we would have to find a more stable input to our network.

Attempt 2: Team Data

Input Data Used:

After seeing the volatility of using only player data, we realized that we would have to find data with significantly less variance in order to accurately predict games. We decided our only real option was to use team statistics, and got to researching which stats are most impactful on a game. In our research, we found that there is a group of four statistics (EFG, RB, FT, TOV) called the ‘four factors’ [1], which are a very accurate predictor of who won a game. We decided to use mainly these factors (and a couple others) and see where this got us. The final data we ended up settling on was:

WPCT (winning percentage of the team for that season)
EFG (effective field goal percentage, generally measures offensive efficiency)
FTM (free throws made) FTA (free throws attempted)
OREB (offensive rebounds) DREB (defensive rebounds)
STL (steals) BLK (blocks) TO (turnovers)

This time, we decided that instead of taking an average, we could allow ourselves to give these statistics from each game separately. Like before, here we tested many options and found that 8 games back was a good benchmark to use.

Techniques Used:

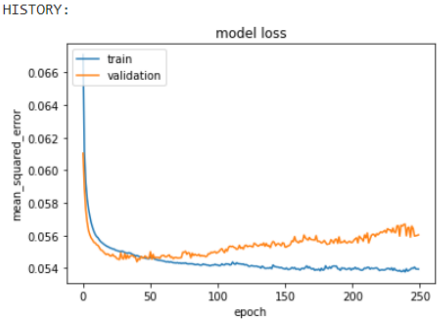
_____As in the previous case, here we also attempted to use both a convolutional and a fully connected network. Our goal in using the CNN was to correctly weigh each game, for example recent games may be more relevant than ones that are farther back, and so they should be weighed more. Unlike last time, here the convolution lowered the network's performance by almost 2 percent, and so we decided this was not the correct approach in this case.

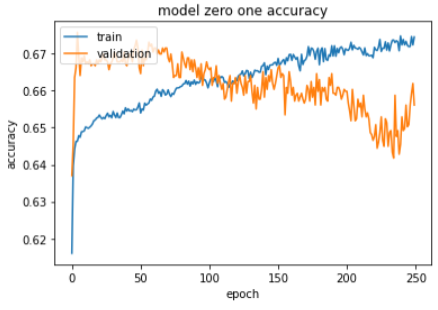
Despite the failure of the CNN, we still thought that the idea of recent games being more relevant was an important one. We decided to try and weigh the games ourselves manually, by multiplying each game in the vector by a factor that is dependent on how recent that game was. This approach worked far better than the CNN, but still did not seem to improve the best result we got using a simple fully connected network.

Results:

Test ZO Loss: `0.6677804295942721`
Graph and model parameters:

HISTORY:






```
------------current model------------
CNN=False
TRAINING_FILE_NAME='team_training_8_8_25_2004.csv'
FEATURES_LEN=130
REGULARIZATION=<keras.regularizers.L2 object at 0x7f8bba
REGULARIZATION_CONST=6e-05
DROPOUT=None
BATCH_NORMALIZATION=False
EPOCHS=250
SGD_EPOCHS=10
BATCH_SIZE=32
DENSE1=75
DENSE2=75
DENSE3=75
DENSE4=75
VALIDATION_SPLIT=0.1
OPTIMIZER='adam'
LOSS_FN='mean_squared_error'
-------------------------------------
```

Conclusions:

As expected after the weak results of the first attempt, this attempt gave us dramatically better results and even brought us on par with the other studies we had seen [2]. We suspect that this is due to what we said previously, that team results contain far less variance, and that they take into account how well the players fit with one another.

Attempt 3: Team Winning

Input Data Used:

While working on the second attempt we noticed something significant about the inputs we were putting in. It seemed that by far the most important statistic in our input vectors was winning percentage, to the degree where if this statistic was removed, the results became worse by several percent. This caused us to wonder if counting stats (points, rebounds, etc.) were even relevant, and if we could still get good results using only statistics relating to winning. We had seen a different study that attempted this approach [3] and so we took what they did as a baseline and expanded on it. The stats we ended up using this time were:

Winning percentage for that season
Home winning percentage/Away winning percentage based on the team
Win/loss record in the past 8 games
Point difference at the end of each of the last 8 games
Winning percentage of the last 8 teams played against
EFG of opponents in last 8 games
EFG in last 8 games

Meaning we barely used any traditional stats, and mainly paid attention to how much the team won in the last 8 games.

Techniques Used:

In this attempt we mainly stuck to a classic fully connected approach, because we had seen previously that the CNN does not work well on this sort of data.

We did however notice that as you got closer to the beginning of a season, the data that relates to winning in that season becomes far less reliable, due to a smaller sample size. Therefore, we attempted to create a new metric, 'adjusted season win pct', that handles this issue. This metric was calculated in the following manner:

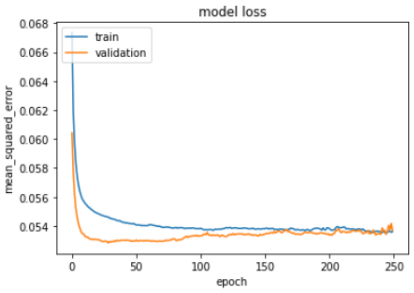
(win_pct*games_played + 0.5)/(games_played+1)

This way, an extreme record early on in the season would not be weighed as heavily as one later in the season. This approach gave slightly more consistent results, but did not significantly improve the overall success of the network.

Results:

Test ZO Loss: `0.6620525059665872`

Graph and model parameters:



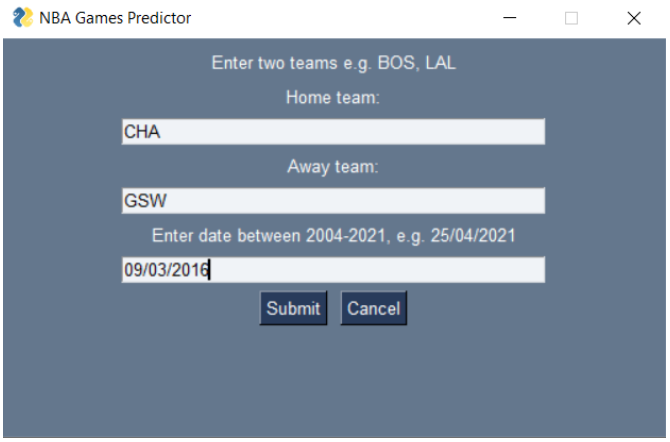

```
------------current model------------
CNN=False
TRAINING_FILE_NAME='team_training_8_5_25_2004_win_pct.csv'
FEATURES_LEN=84
REGULARIZATION=<keras.regularizers.L2 object at 0x7f6de45616
REGULARIZATION_CONST=0.0001
DROPOUT=None
BATCH_NORMALIZATION=False
EPOCHS=250
SGD_EPOCHS=10
BATCH_SIZE=32
DENSE1=50
DENSE2=50
DENSE3=50
DENSE4=None
VALIDATION_SPLIT=0.1
OPTIMIZER='adam'
LOSS_FN='mean_squared_error'
-------------------------------------
```

Conclusions:

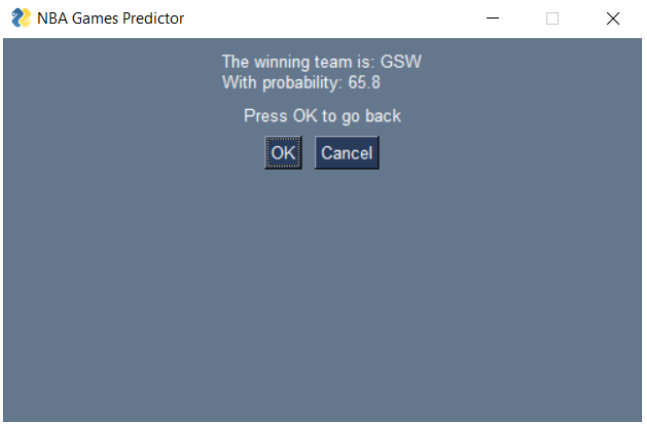
As we can see, not only was this model better than the first attempt, it was even almost competitive with the second attempt in which we included far more statistics. This was very surprising to us, because we were under the impression that the traditional counting statistics (points, rebounds, assists) were necessary for predicting game outcomes. This last attempt has led us to conclude that even these stats for the team as a whole barely affect the accuracy of the prediction. It seems that by far the most major factor in predicting wins is simply how much the team has won previously, with all other factors seemingly impacting the results a very small amount.

The Final Product

After our three attempts it seemed that attempt 2 had given us by far the best results. We saved the best network from that attempt and began to create our final product. The program ended up looking like this:



After the teams and date are entered, a team predicted to win is selected, including the level of certainty with which the model predicts this team will win. This percentage is chosen based on the label the network outputs, and on how accurate the network was for test samples with similar labels. This way a user can have a better idea of the real certainty of the outcome:



Due to the fact that our sample size is relatively limited, we decided to only use three possible ranges a game can fall into. The ranges are as follows:

```
CERTAINTY CHECK FOR TRESHOLD BETWEEN: 0 AND 0.08,PERCENTAGE: 0.30978520286396183, CERTAINTY: 0.588597842835131
CERTAINTY CHECK FOR TRESHOLD BETWEEN: 0.08 AND 0.12,PERCENTAGE: 0.3818615751789976, CERTAINTY: 0.65875
CERTAINTY CHECK FOR TRESHOLD BETWEEN: 0.12 AND 0.5,PERCENTAGE: 0.30835322195704057, CERTAINTY: 0.7585139318885449
```

Where the threshold is absolute distance of the label from 0.5, percentage represents the proportion of games in that range, and certainty is our success rate on games in that range.

Comparing to ML

After finishing all our experiments, the best success rate we had reached on the data was about 66.7% at predicting which team would win. We were curious to see how this result fared against other machine learning techniques, and so we used the sklearn library to run some other models on the data we had used. The best results we received from this (after trying on all the data we had used) were these:

```
LINEAR REGRESSION GOT ACCURACY OF 0.6572792362768496
SVM GOT ACCURACY OF 0.6568019093078759 WITH LINEAR KERNEL
SVM GOT ACCURACY OF 0.594272076372315 WITH POLY KERNEL
SVM GOT ACCURACY OF 0.6510739856801909 WITH RBF KERNEL
DESICION TREE GOT ACCURACY OF 0.5723150357995227
```

Which are worse than the results we received using deep learning. This implies that deep learning is a viable option for solving this problem, despite it seemingly not being widely used for this purpose.

Conclusion

Our many experiments over the course of this project allowed us to draw many conclusions, both about the NBA and about the applications of deep learning in this field.

As far as the NBA is concerned, we learned that player data is simply too volatile a metric to draw from in predicting games, and that we must look at teams as a whole during the season, both due to small sample size and to high variance in player statistics. In addition, we saw that most traditional statistics simply add noise to a prediction, and that an effective prediction can be reached using relatively few stats relating to general team success.

In addition, most importantly, we managed to prove that in this field, deep learning is at least competitive with some other top of the line machine learning techniques. This is a testament to the power of deep learning, especially considering the low sample size and high variance and difficulty this problem presents.

Future Work

During our project we noticed that around a third of the games in our sample set end with a point differential of less than 5 points. These types of games will always naturally be difficult to predict, and predicting them accurately is nearly impossible. If we use the close games less in the training process, we may be able to create a network that is incredibly accurate when guessing the other two thirds of the data.

In addition, one could create a mixture of the attempts we did, in an effort to include team statistics and winning percentages, while also paying attention to player injuries and performances.

References/Related Work

[1] Justin Jacobs (2015) “Intro to Oliver’s Four Factors”, https://squared2020.com/2017/09/05/introduction-to-olivers-four-factors/

[2] Eric Scot Jones (2016) “Predicting the Outcomes of NBA Games” https://library.ndsu.edu/ir/bitstream/handle/10365/28084/Predicting%20Outcomes%20of%20NBA%20Basketball%20Games.pdf?sequence=1&isAllowed=y

[3] Renato Torres (2013) “Prediction of NBA Games Based on Machine Learning Methods” https://slidetodoc.com/university-of-wisconsin-madison-prediction-of-nba-games/

[4] Matthew Beckler “NBA Oracle”, https://www.mbeckler.org/coursework/2008-2009/10701_report.pdf