CS230

Beating the Odds – NBA Analytics

Bodhi Nguyen
ICME – Stanford University
bodhin@stanford.edu

Matteo Santamaria
ICME – Stanford University
msantama@stanford.edu

Abstract

In this paper, we attempt to improve traditional models of NBA game predictions by augmenting box score data with player-specific RAPTOR data. This project employs two models. First, a novel log-two layer, fully connected neural network and secondly, a transformer model with self-attention, which treats each player as a vector of their statistics in order to learn relationships between players. We find that all three models perform similarly, with an average smoothed L1 loss of 10.10 in testing. We introduce a measure of the amount of confidence required in our model in order to bet against the bookies, and find that the transformer models are both more confident and slightly better in performance measured by overall model loss and win percentage against the house odds.

1 Introduction

In May of 2018, the U.S. Supreme Court ruled that individual states had the right to legalize sports betting, overturning a longstanding law. Since then, dozens of states have allowed betting sites to operate within their borders, and the cumulative value of the American sports betting market is expected to soar to $8 billion by 2025 [9].

Anecdotally, betting lines are remarkably accurate, but as an aggregation of human opinions, they are inherently biased. We believe we can improve on these group-think predictions by taking a data-driven approach towards odds making. Traditional NBA forecast models use team-based statistics to predict the outcome of games. Here, we've designed a deep learning system that utilizes individualized player data in addition to conventional game statistics to generate more accurate forecasts of team point production. In particular, the input to our algorithm is a vector of historical numerical statistics for the top six players on each team for a particular game. We then use a neural network to output a predicted score for each team in a particular game. By calculating a predicted score for each team, we are able to compare our predicted outcomes against actual outcomes as well as outcomes implied by betting lines.

A quick review of betting terminology: there are three basic types of bets one can make on a sporting event, `over/under`, `spread`, and `moneyline`. We focus mainly on the `over/under` and `spread` in this project. Betting on the `over/under` is wagering on the cumulative number of points scored by both teams; you can either pick `over` or `under`. Alternatively, betting on the `spread` is betting on a team after accounting for a point differential. In essence, the favorite is handicapped by subtracting off a pre-determined number of points. You are betting on the outcome of the match after accounting for the handicap.

2 Related work

Our initial inspiration came from the paper "The Bank is Open: AI in Sports Gambling" [3]. This was a project for CS229, in which one method they used was a fully connected, four layer Neural

CS230: Deep Learning, Fall 2020, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

Network on several player statistics for NBA games. They found that their model correctly chooses the Over or Under 51.5% of the time. We chose to investigate a similar method with additional player-level data. Another paper, "Predicting NBA Games Using Neural Networks" [1], used manual feature selection by experts to put into a variety of neural networks, such as feed-forward, radial basis, probabilistic and generalized regression networks. These researchers found they were able to predict the winning team 74.33% of the time. However, their top 5 neural network architectures all performed similarly, with an average accuracy of 71%. This suggests that feature engineering was particularly important in their case. In "Neural Network Quarterbacking" [10], Purucker attempts to predict NFL outcomes using an MLP and achieves a 64.3% winner pick rate with only five features. In this model, scores are not predicted but their results suggest that only a few important features may be necessary to predict a winner, in the case of the NFL at least. Lastly, in "Artificial Intelligence in Sports Prediction" [4], McCabe and Trevathan use a form of multi-layer perceptron applied to rugby, Australian rules football, and soccer. They find an average performance of around 67.5% with a network that has only one hidden layer of half the size of the input.

In an attempt to try new models on this existing problem, we drew inspiration from the paper "Attention is All You Need" [12]. There are several other papers with mixed results on our problem with RNNs and complicated models, but this paper shows that in translation tasks, disposing of those models entirely and using attention mechanism had advantages in both performance as well as computational cost. Together, the previous literature suggests that feature selection may be important in our task, and that complicated models may not be necessary with this type of data due to the relative scarcity of observations. Nevertheless, we build both a similar multi-layer feed-forward neural network, as well as a neural network utilizing “self-attention" layers, which we describe further in the model section.

3 Data & Preprocessing

The raw data was sourced from three primary stores: NBA.com [8], FiveThirtyEight.com [7], and sportsbookreviewsonline.com [6].

Historical boxscore data was scraped from NBA.com and was used to compute "live" player statistics at each day throughout the season. This allows us to use current player statistics as an input to our model, as opposed to retroactively applying seasonal averages. The boxscore data was then used to select the five starting players per game by position, along with the `S1` position (the non-starting player with the most minutes played). This simplifies the problem of having to represent entire teams, of possibly different sizes, to identifying the six most impactful players.

RAPTOR is a derived dataset produced by FiveThirtyEight and used in all their NBA modeling [11]. It is a plus-minus statistic that measures the number of points a player contributes to his team’s offense and defense per 100 possessions, relative to a league-average player. Altogether, our data is on the team and game-level, where each row consists of player-level historical statistics as features, and the output label is the individual team score (two rows per game).

Lastly, we downloaded odds data from Sports Book Review, an online archive of historical line data [6]. This dataset contains information about historical over/unders and spreads (both opening and closing), which is useful in evaluation of our model.

In total, our game data was for the seasons 2015-2020, split into four seasons for training, one season for development, and one season to test on. The training set has 9,330 observations, the dev set has 2,330 observations, and the test set has 1,994 observations. Since one would be using this model on sequential seasons in practice, it seemed sensible to isolate the 2020 season as the test, the 2019 season as the dev, and the remainder as training data.

4 Methods

4.1 Log Two Layer, Fully Connected NN

Our first model employed a fully-connected artificial neural network (ANN) model. ANN models are particularly well-suited for non-linear regressions. One weakness, however, is their need for copious amounts of data. Due to RAPTOR data deficiencies, our analysis was restricted to the years from 2015 through 2020 (6 total seasons). With approximately 2,500 games per season, this gives

us roughly 15,000 matches. Although this may seem restrictive, key shifts in playing styles and rules over time (higher scoring, players coming into and leaving the league) would make it so that older observations are much less relevant for predicting scores in our test set (2020). Since we do predictions by team, each match results in two observations, for a total of 30,000 observations.

In our fully connected model, the number of units in each successive hidden layer decreases by a half, so that layer l has $|X|/2^l$ units where $|X|$ is the number of input features. The current iteration of our model has $6 * 4 + 6 * 9 + 2 * 6 * 10 = 198$ features. These are the four "defensive" player statistics for each of the six players on the opposing team, plus the nine "offensive" statistics for the team we're predicting, plus the 10 RAPTOR statistics for all 12 players across both teams.

4.2 Attention Layers and Transformers

As a secondary model, we chose to alter our base model using "attention" layers. In the natural language processing context, they are used to associate context between two sequences through the use of weights on the dot product of two vectors, for example. In our project, we are attempting to learn relations between individual players and their player statistics. To do this, this model treats each player as a vector of their individual statistics. We will attempt to make use of "self-attention", in which a vector learns weights over itself. The intuition behind this is that we are trying to learn how important players are to the team score in relation to one another. We don't use the full transformer since our output is not a sequence, and we actually don't have a sequential nature to our input.

Since our output is a scalar, we use a similar architecture without decoding to a sequence at the end. Our architecture consists of 3 transformer blocks stacked together, where each transformer block is structured as follows, using a suggested stack from Peter Bloem [2]. The output of these transformer blocks are chained together, and lastly put through a single fully connected layer with an output dimension of 1.

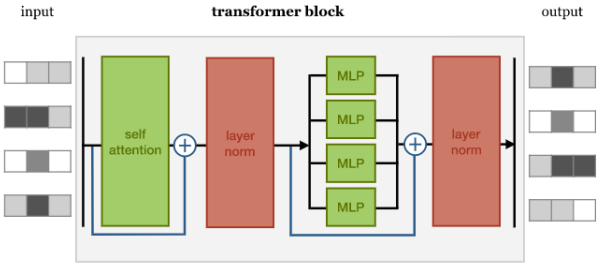


Figure 1: Transformer Block (Peter Bloem)

4.3 Experiments/Results/Discussion

Currently, during training the parameters of our feedforward model are optimized using mini-batch gradient descent with batch sizes of 100 over 500 epochs. Our training takes <2 minutes due to our relative lack of data. We chose to use smoothed L1 loss, which is more robust to potential outliers in the data (i.e players scoring in spurts of 50+ points a game) than L2 loss. This seems to work rather well, as our model's loss decreases significantly and then stabilizes. The learning rate was 0.0001, chosen to be relatively slow because it would smooth our gradient updates, at the cost of longer training time (which was not an issue for us).

The transformer model was trained in a similar way. It was optimized using AdamOptimizer using smooth L1 loss (Huber) over 50 epochs with a batch size of 32. We chose a small batch size to potentially break out of local minima, at the cost of a longer training time. Because our training time is relatively small overall, this was acceptable. Additionally, powers of two are a common choice for batch sizes. The learning rate is 0.0001, also relatively small since our training time is not an issue. The number of epochs required for the training error to plateau was much smaller in this case. Experimentation over different numbers of heads and depths found that a depth of 2 was sufficient (no further improvements with more heads) and any head number between 1 and 16 performed similarly, with very slight improvement at 8 heads. In this project we analyzed two transformers with 8 and 16 heads, with a depth of 2. One study from CMU, "Are Sixteen Heads Really Better Than One?" [5]

found that most heads are irrelevant at test time, even though the training error decreases in their task. We found similar results in our hyperparameter search over the number of heads. Nevertheless, we compare the 8 and 16 head models to see if there is any real difference. The L1 loss achieved on the train, validation, and test sets using each model are listed below.

Average Losses by Model

Model	Training Loss (L1)	Validation Loss (L1)	Test Loss (L1)
Fully Connected Log 2 Layers NN	7.78	11.24	11.62
Multi-Head Transformer, 16 heads, 2 blocks	7.97	9.44	9.22
Multi-Head Transformer, 8 heads, 2 blocks	8.01	9.58	9.45
Mean Loss	7.92	10.09	10.10

The second metric we can examine is the estimated total points scored in a match-up against the over/under betting line. The better our per-team prediction is, the more accurate our over/under line will be. This metric allows for a bit more flexibility. If we underestimated the points scored by team A by a basket, but also overestimated the points scored by team B by a basket then our over/under will still be accurate. Finally, we can compare our estimated spread to the betting spread and the actual spread.

Win Rate Against Bookies

	Model		
Bet Type/Timing	LogTwo	Transformer, 16 Heads	Transformer, 8 Heads
Over-Under/Close	46.4%	49.5%	49.1%
Over-Under/Open	47.5%	51.4%	48.3%
Spread/Close	47.5%	48.7%	48.0%
Spread/Open	48.0%	50.1%	50.3%

Our results above show that the model does a fairly poor job predicting the over/under and the spread at the close, which isn't particularly surprising given these objectives are not what the model is explicitly optimized for. We do see, however, that the transformer models do much better predicting the spread.

In both scenarios, our predictive power is relatively worse at the close then the open. This makes sense as there is more uncertainty when odds are first released (player's may get injured leading up to the game, etc.). However, since our model depends on knowing which players will start, we should judge the model's performance with respect to the closing odds. Since betting odds are inherently biased in the bookies' favor (you almost never find an "even money" bet), one must achieve a win rate of about 52% to be profitable [3].

Most of our initial winrates compared to the bookies are below 50%. To combat this, we introduce a buffer, which is a threshold amount of edge we need in our prediction against the bookies in order to trade against them. Below are the buffers required to achieve above a 50% winrate for each model, along with the winrates at push rates of about 70%, 80%, and 90% (meaning when betting only 30% of the time because of our confidence rules, what was the winrate?). Note that winrates at push rates are approximate—the exact values can be found in the Jupyter notebooks accompanying this project.

Buffer(Confidence) Required to Achieve >50% Win Rate

	Model		
Bet Type/Timing	LogTwo	Transformer, 16 Heads	Transformer, 8 Heads
Over-Under/Close	22	3	10
Over-Under/Open	18	0	4
Spread/Close	14	6	4
Spread/Open	6	0	0

Win Rates at Different Push Rates

	Model		
	Win Rates @ (70%/80%/90%) Push Rate		
Bet Type/Timing	LogTwo	Transformer, 16 Heads	Transformer, 8 Heads
Over-Under/Close	51%/53%/54%	52%/52%/54%	52%/54%/55%
Over-Under/Open	52%/55%/56%	53%/56%/52%	53%/55%/55%
Spread/Close	48%/46%/49%	50%/53%/55%	51%/51%/50%
Spread/Open	49%/50%/57%	51%/50%/58%	51%/51%/60%

To compare the three models, a 50% or above winrate at a lower buffer reflects increased confidence in the accuracy of the model, since we have to be less sure about the model relative to the bookies in

order to trade. Comparing the two Transformer models, we see that the model with more heads does a bit better by minimum buffer to achieve 50% win rate, but about the same by win rates at different push rates.

4.4 Aside: Self-Attention in the Transformer

One benefit of self-attention layers in the transformer model is that we can visualize the relationships between player positions that are learned by the attention layers. For the 16 head model, averaging over all the heads over recent 32 instances where a forward pass is made through the transformer yields this heatmap, where 'HOME' is the feature representing whether a team was at home or not:

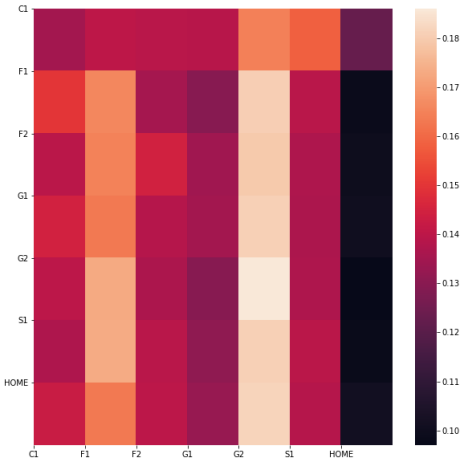


Figure 2: Average Weights, 16 Head Transformer

This plot shows the associations between different player positions. The cell a_{ij} contains the weight the vector j has in relation to player position i. Most notably, we see that the G2 position is the most important to everyone else, followed by the F1 position. Whether a team is HOME or not is the least associated with the player positions, which makes sense. For a single head as well (appendix), we spot variances in this heatmap, in that a single head may learn that the G2 is more important, for example.

5 Conclusion/Future Work

Altogether, we used player statistics from NBA.com as well as FiveThirtyEight's RAPTOR player data in order to predict points scored in NBA games. We trained three models, a fully connected neural network consisting of $\log_2 k$ layers, where k is the dimension of player statistics, and two transformers model using multi-head self-attention layers (8 and 16 heads each). We achieved an average L1 loss on the test set of the 2020 season of 11.62 for the first model. For the transformer model, the L1 loss was 9.45 and 9.22 for 8 and 16 heads. Against the bookies, our base models won an average of 49.1% and 48.3% of the time against the over/under at the open and close respectively. Our models also won an average of 49.5% and 48% of the time against the spread at the open and close, respectively. Using a buffer to measure how much confidence our model must have in order to trade, our first model saw winrates above 50% with a buffer of 22 and 22 against the over/under and spread at the close, while the 16 head transformer saw winrates above 50% with a buffer of 3 and 6. The models have relatively similar performance against the bookies at the same push rates, but the 16 head transformer model was most confident in its predictions at the same buffer rates.

Given more time and computational resources, we would investigate further forms of data that might have predictive value for these games, as it seems that more complex models are not necessarily better on our current dataset. For both models, our loss reached a plateau quickly with minimal training time. This suggests that we could have benefited from more data, and a wider range of types of data.

6 Contributions

Github Repository: `https://github.com/matteosantama/bookmaker`

Matteo: Created entire data pipeline and processing, built and evaluated log two layer model, created evaluation statistics methodology against bookmakers.

Bodhi: Incorporated RAPTOR data, converted data to vectorized format by player for attention, built and evaluated Transformer models

Milestone, Report, and Presentation were worked on equally.

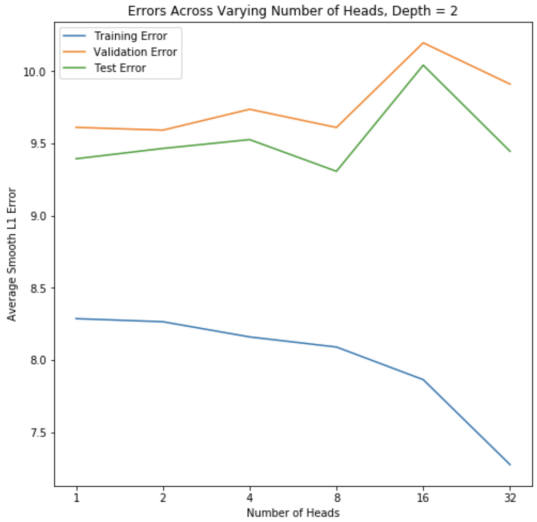
7 Appendix

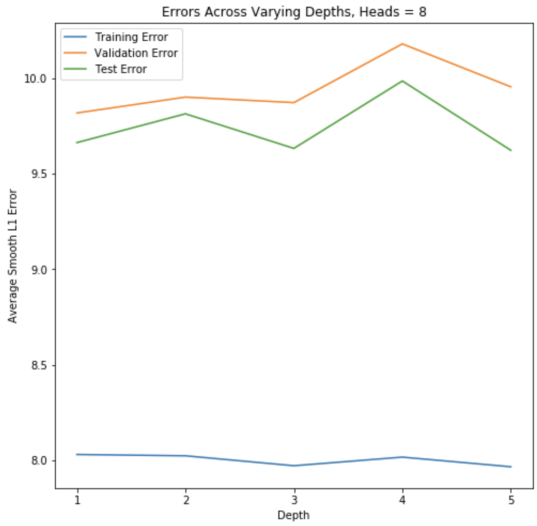
7.1 Transformer Model: Some details

In particular, for our self-attention layers we use multi-head, scaled-dot product attention. This is done as follows: with $q_i = W_q x_i$, $k_i = W_k x_i$, and $v_i = W_v x_i$ as our query, key, and value vector components, we calculate $w'_{ij} = q_i^T k_j$, then take the softmax operation over the w'_{ij} to obtain w_{ij}. Then, the output vector component $y_i = \sum_j w_{ij} v_j$. We also scale the initial weights by the dimension to prevent the vanishing gradient issue, as is typically done. Additionally, we use multiple heads, which combines several attention mechanisms for each position in our vector. This self-attention layer is normally used in a "Transformer" architecture, which translates sequences. As shown in the diagram, we used a architecture for our transformer block like Peter Bloem's: a self attention layer, followed by a layer norm, a feed forward network, and another layer norm, combined with residual connections.

7.1.1 HyperParameter Tuning

Two experiments were done: different numbers of heads at a depth of 2 transformer blocks, and different depths with 8 heads. The results are shown below in graphs.





7.1.2 Extra Attention Visualizations

The two following heatmaps are the average attention representations learned by the 8 head model, and the attention representation learned by a single head of the 8 head model.

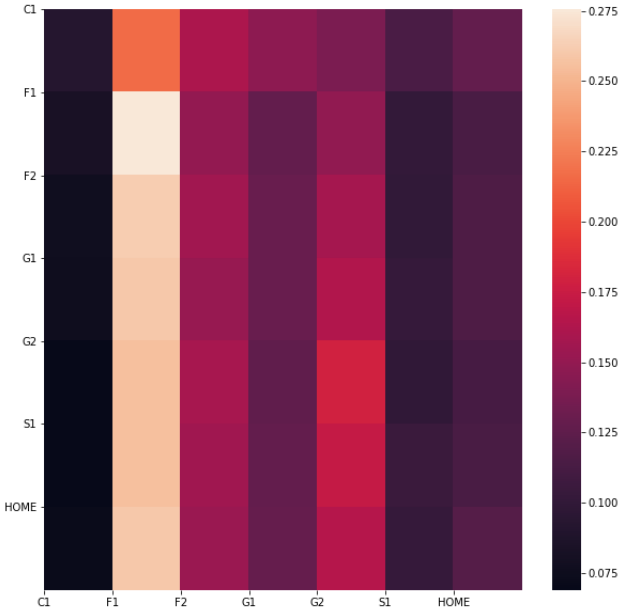


Figure 3: Average Attention Representation, 8 Head Model

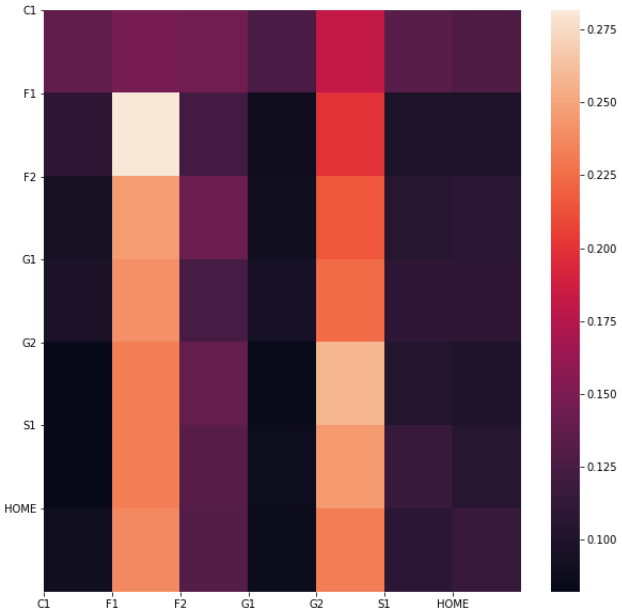


Figure 4: Single Head Learned Attention Representation

References

[1] Loeffelholz Bernard, Bednar Earl, and Bauer Kenneth W. "Predicting NBA Games Using Neural Networks". en. In: *Journal of Quantitative Analysis in Sports* 5.1 (2009), pp. 1–17. URL: https://ideas.repec.org/a/bpj/jqsprt/v5y2009i1n7.html (visited on 11/17/2020).

[2] Peter Bloem. *Transformers from scratch | peterbloem.nl*. URL: http://peterbloem.nl/blog/transformers (visited on 11/17/2020).

[3] A. Bucquet and V. Sarukkai. *The Bank is Open: AI in Sports Gambling*. 2018. URL: http://cs229.stanford.edu/proj2018/report/3.pdf.

[4] A. McCabe and J. Trevathan. "Artificial Intelligence in Sports Prediction". In: *Fifth International Conference on Information Technology: New Generations (itng 2008)*. Apr. 2008, pp. 1194–1197. DOI: 10.1109/ITNG.2008.203.

[5] Paul Michel, Omer Levy, and Graham Neubig. "Are Sixteen Heads Really Better Than One?" In: Nov. 2019. URL: https://arxiv.org/pdf/1905.10650.pdf.

[6] *NBA SCORES AND ODDS ARCHIVES*. sportsbookreviewonline.com. URL: https://www.sportsbookreviewsonline.com/scoresoddsarchives/nba/nbaoddsarchives.htm.

[7] *nba-raptor*. FiveThirtyEight. URL: https://github.com/fivethirtyeight/data/tree/master/nba-raptor.

[8] *NBA.com*. National Basketball Association. URL: https://www.nba.com/stats/.

[9] Associated Press. *Sports betting market expected to reach $8 billion by 2025*. 2019. URL: https://www.marketwatch.com/story/firms-say-sports-betting-market-to-reach-8-billion-by-2025-2019-11-04.

[10] M. C. Purucker. "Neural Network Quarterbacking". In: *IEEE Potentials* 15.3 (Aug. 1996), pp. 9–15. ISSN: 1558-1772. DOI: 10.1109/45.535226.

[11] Nate Silver. *Introducing RAPTOR, Our New Metric For The Modern NBA*. 2019. URL: https://fivethirtyeight.com/features/introducing-raptor-our-new-metric-for-the-modern-nba/.

[12] Ashish Vaswani et al. "Attention Is All You Need". In: *arXiv:1706.03762 [cs]* (Dec. 2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762 (visited on 11/17/2020).