CS230

Neural Networks for Market Predictions in Fantasy Basketball

Sammy Mohammed
Department of Computer Science
Stanford University
sammym@stanford.edu

Jacob Tie-Shue
Department of Computer Science
Stanford University
jtieshue@stanford.edu

1 Introduction

In Daily Fantasy Basketball, participants draft a fantasy team of 8 players while staying under a given salary for the sum of their players [1]. The objective is simple: draft a team that will score the most fantasy points on a given night. Teams must contain certain positional players while remaining under a fixed salary cap [2]. Our project focuses on creating a DraftKings NBA fantasy score predictor. In essence, given a list of players, stats, and pricing, we want to create a model that determines the optimal DraftKings team for a given day. DraftKings is a platform for fantasy sports and sports betting; we want to apply our score predictor to 50-50 style NBA fantasy betting (meaning that if 100 participants enter, the top 50 participants get 180% of their buy-in back). [2] In order to achieve this, our project involves using a deep neural network in order to predict fantasy points for NBA players on a given night. We reference historical NBA performance per player, with an input data set featuring information such as salary, points per game, and offensive rating.

2 Related work

Most related work in prediction does not center around fantasy basketball. Given the plethora of data and many statistics, we believe this is an under-explored area. Nonetheless, contemporaries have done similar work to help predict sports games. In "Sports data mining technology used in basketball outcome prediction" [3], a comparison study was built using neural networks, a logistic classifier, SVM, and Naive Bayes. They found that the simple logistic classifier worked best for predicting game victories.

Another paper that we plan to draw inspiration from is "Deep Artificial Intelligence for Fantasy Football Language Understanding", written by three engineers from IBM. Their use case is similar to ours, as are their machine learning methods for structured statistical data. To deal with unstructured data, they developed a novel language model using Doc2vec, Watson Knowledge Studio annotation, and webhose.io. Their model was able to convert articles, podcast, and other sports media into features that could be used to perform deep learning. We hope that we can apply their learnings to our project,

We also refer to our work in CS229, and our following paper "ML Kings: Using Machine Learning to Beat DraftKings" [4], which involved building a five-layer neural network trained over our dataset in order to predict player performance relative to DraftKings estimates. We received promising results from our work in this subject area in CS229, which served as a primary motivator in selecting our project focus for CS230.

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

3 Novelty

While our initial work comes from our CS229 paper, we focused on adding novelty by upgrading the architecture we initially used in CS229. This came in the form of incorporating unique non-linearities and experiment with up-sampling on our dataset. Our implementation of a small convolutional network also pulls closely from our learnings in CS230. Our future work with unstructured data explores adding features to our model, and offers opportunity to explore work with text and larger datasets.

4 Dataset and Features

We are using an open-sourced (MIT license) project on Github that contained scrapers for both NBA data and historical DraftKings data. With this data, we believe we will be able to retrieve performance stats on all competing NBA players from 2015 to late 2021. This dataset has 51 columns per player per game, and features over 80,000 attributes. Our 51 features include both general statistics, such as points, rebounds, assists, blocks, as well as advanced statistics, such as defensive rating and true shooting percentage. These statistics are used to predict our fantasy points for a given player.

We choose such a large dataset in order to try and see if individual players have performance variation temporally, and not just purely in their individual stats. For example, a player averaging 30 points over their last five games may be not be as great of value as a player averaging 25 points in their last five games, depending on the rosters each player had to face.

In addition to our numerical data, we also incorporated unstructured data into our model. We did so by utilizing Social Power NBA, an open-sourced dataset found on Kaggle [5]. This dataset contains combined on-court performance data for NBA players in the 2016-2017 season, alongside salary, Twitter engagement, and Wikipedia traffic data. By scraping Twitter and Wikipedia, it allows us to leverage the fact that better players are generally more popular. Specifically, it allowed us to consider non-game related data, to see if discussion and popularity of a player is strongly correlated with performance on a given night.

5 Methods

5.1 Simple Neural Network

The first method that we utilized was a simple neural network. Each neuron is comprised of a linear layer followed by a ReLU activation:

$$h_\theta(x) = ReLU(w^T x + b)$$

Our optimizer of choice was Adam optimization, which is modeled by the following update rules:

$$\theta_{j+1} := \theta_j - \alpha m_j$$

$$m_j := \beta m_{j-1} + (1-\beta)\frac{\partial}{\partial \theta_j} J(\theta).$$

Utilizing Adam provided good opportunities for hyperparameter tuning. We utilized L1 loss as our objective:

$$J(\theta) = |h_\theta(x^{(i)}) - y^{(i)}|$$

Since NBA game data is prone to outliers, i.e. a given player underperforming/overperforming across any number of statistical categories on a given night, we opted for L1 over L2 when choosing our objective function.

Our baseline model was a seven-layer network from the project milestone. We then experimented with hyperparameter tuning for further neural networks.

We continued iterating by testing less complex models with fewer layers; we used a three-layer and a five-layer model, respectively.

5.2 Regularized Gradient Boosting

We then experimented with ensemble algorithms [6]. We chose to use boosting/XGBoost, a highly scalable end-to-end tree boosting system, with support for regularized learning objectives. This algorithm uses K additive functions to predict the output for a given data set with n examples and m features [7]. Explicitly, it solves for $\hat{y}$:

$$\hat{y}_i = \phi(\mathbf{x}_i) = \sum_{k=1}^{K} f_k(\mathbf{x}_i), f_k \in \mathcal{F},$$

where $\mathcal{F}$ is the space of regression trees [7]. We then minimize the regularized objective:

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k),$$

where

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2$$

where l is the convex loss function between $\hat{y}_i$ and y_i. After incorporating an additive manner [7], our final minimizing objective became:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(\mathbf{x}_i)) + \Omega(f_t)$$

We improved our methods using hyperparameter tuning as well as incorporating the aforementioned unstructured data as features for our model. Per our meeting with our TA mentor, we focused our efforts into improving our existing methods rather than incorporating vastly new machine learning techniques for our task.

6 Results and Evaluation

Our results are summarized in the following table:

	Training L1 loss	Testing L1 loss
Baseline Neural Network, 7-layer, dropout = .2, learning rate = .01	11.16	11.25
7-layer Neural Network + Unstructured Data, dropout = .2	11.15	11.21
3-layer Network + Unstructured Data, dropout = .15, learning rate = .02	7.45	7.55
3-layer Network + Unstructured Data, dropout = .1, learning rate = .02	7.28	7.47
5-layer Network + Unstructured Data, dropout = .2, learning rate = .01	7.2	7.2
5-layer Network + Unstructured Data, dropout=.1, learning rate=.02	7.03	7.3
XGBoost, max depth = 5, learning rate = .01	5.1	11.4

Drawing on our work from our CS229 paper [4] as well as our initial experiments from our CS230 Milestone, we utilized a 7-layer neural network as a baseline model, trained on just numerical data. It had a pretty standard neural network setup, with dropout set at .2 and ReLU activations at each layer. As shown in the table, the 7-layer baseline achieved a train/test loss of 11.16/11.25. Adding our unstructured data augmentation to the 7-layer network did not improve performance by much, with train/test loss improving to 11.15/11.21. As flagged by our TA mentor, we hypothesize that 7-layers was too many for our use case, which caused our model to not learn and predict effectively.

To reduce complexity, our next experiment was a 3-layer neural network, using our unstructured data agumentation. Without changing learning rate or dropout hyperparameters, we observed a marked improvement, with train/test lost dropping to 7.45/7.55. This validated our hypothesis that simplifying a model could lead to better results, even if the task seems complex. Loering the dropout rate and increasing the learning rate led to another slight improvement, with train and test loss of 7.28 and 7.47, respectively.

After our 3-layer experiments, we figured that there should be a "sweet spot" amount of layers; 7 was too big, but perhaps the optimal amount lay in between 3 and 7? Our answer was 5-layers. For a

5-layer network with original dropout and learning rate, and using unstructured data, we reported a train/test loss of 7.2/7.2, our best numbers yet. We also ambitiously tried to lower the dropout rate to achieve even better results. While this did lower our train loss to 7.03 as expected, our model then exhibited overfitting, with test loss increasing to 7.3.

The final experiment we conducted was with XGBoost. As stated in the preceding sections, we used XGBoost just in case ensemble algorithms were better suited to our prediction task. Our XGBoost seemed initially promising, as we reduced training loss all the way down to 5.1. However, these experiments displayed a considerable amount of overfitting, with a test loss of 11.4.

6.1 Application of results

Using our best-performing model from our experiments (5-layer neural network), we decided to attempt to create optimal DraftKings lineups. We accomplished this using a Simplex algorithm, which simplified our NP-hard challenge by finding the optimal lineup (maximizing total predicted points scored), while satisfying all of the constraints of a valid lineup (positions and salary, etc) [8] [4]. This infrastructure allowed us to quickly generate optimal lineups for any given date, and back-test our performance. For example, when we run the Simplex algorithm on April 25, 2018, the optimal line-up given our predictions is:

Player Name	Position	Salary	Points
Lebron James	PF	12000	53.1
Russell Westbrook	PG	10600	75.75
Victor Oladipo	SG	8100	36
Domantas Sabonis	C	4200	32.75
Jae Crowder	SF	4100	46
OG Anunoby	SF	3400	10.25
PJ Tucker	PF	3400	23.75
Cory Joseph	PG	3300	26.75
TOTALS	-	49100	323.25

Our actual generated score of 323.25 for that given night handily surpasses the average winning DraftKings score of 280 points [9]. This gives us confidence that our model does a reasonably good job at winning on a night. Furthermore, our Simplex testing uncovered an average scores of about 290, which is promising in yielding expected winnings over an entire season.

7 Conclusion and Future Work

In short, we determined that using neural networks trained on NBA stats as well as player intangibles like influence, is a viable method of beating the average DraftKings score on a given night and make money. We also believe the work done here is applicable to many other market domains, like trading and resource allocation. This work is broadly applicable and can offer advantages over human players in domains like prediction markets.

Further improvements remain by incorporating greater amounts of unstructured data, like player sentiment scraped from the web, and fan opinions on given players. This data could be fed into a transformer model and fed as an input to the neural network, but time and computation constraints restrict us from adding these features now. Another potential future improvement can lie in our data processing; by specifically weighting features based on their predictive impact. For example, it's currently unknown how influential a player's salary relates to their fantasy points – being able to weigh specific fields could lead to improvements.

8 Contributions

Jacob worked on the simple neural network experiments and led the written report work. Sammy worked primarily on setting up the XGBoost approach and building the less complex neural network model alongside Jacob. He also focused on gathering and incorporating the unstructured data for the model.

9 Code

Our code is located at github.com/SammyMohammed/cs230-fantasy-basketball.

References

[1] Ryan J. Martin, Sarah Nelson. "Fantasy sports, real money: Exploration of the relationship between fantasy sports participation and gambling-related problems". In: *Addictive Behaviors*. Vol. 39. 2014.

[2] DraftKings. *Rules and Scoring*. URL: `https://www.draftkings.com/help/rules/overview`.

[3] Chenjie Cao. "Sports data mining technology used in basketball outcome prediction". Masters Dissertation. Technological University Dublin, 2012.

[4] Fede Zalcberg Jacob Tie-Shue Sammy Mohammed. *ML Kings: Using Machine Learning to Beat DraftKings*. Dec. 2021.

[5] Noah Gift. *Social Power NBA*. URL: `https://www.kaggle.com/datasets/noahgift/social-power-nba?resource=download`.

[6] Robert E. Schapire. "The strength of weak learnability". In: *Machine Learning*. Vol. 5. 1990.

[7] Tianqi Chen, Carlos Guestrin. *XGBoost: A Scalable Tree Boosting System*. 2016. URL: `https://arxiv.org/pdf/1603.02754.pdf`.

[8] Jean-Michel Réveillac. "Linear Programming". In: *Optimization Tools for Logistics*. 2015.

[9] Brian Hourigan. *What it Really Takes to Win an NBA GPP*. URL: `https://rotogrinders.com/articles/what-it-really-takes-to-win-an-nba-gpp-1210935`.