McGill University

FINAL REPORT - GROUP 23

ECSE 457 - DESIGN PROJECT

Picking Winners for Fantasy Basketball with Machine Learning

Group Members
Ege AYDEDE - 260576559
Stephen POOLE - 260508650
Florence REGOL - 260639261
Asher WRIGHT - 260559393

Supervisor
Prof. Ioannis PSAROMILIGKOS

April 16, 2017

Abstract

This project was motivated by the work of David Hunter et al. [5] where the objective was to predict winning teams in fantasy hockey competitions using integer linear programming. In this project, we consider the problem of picking winning lineups in daily fantasy basketball competitions using machine learning. The objective of this project is to learn the principles of machine learning, and be able to apply those on a large collected dataset from the NBA website in order to predict winning lineups for FanDuel daily NBA competitions. In order to achieve this, the design process was broken into three steps: Research, data collection, and implementation. Most of the research and data collection was performed in the first semester of the design project. During the second semester, the focus was on building a finalized version that could be competitive against real people in these competitions. This system was ultimately successful, producing competitive and profitable lineups.

Acknowledgements

We would like to acknowledge and thank everyone who helped us with this project, including our supervisor, Professor Psaromiligkos, and two helpful talkers, Islay Wright and Ben Potter.

Contents

List of Figures

Abbreviations

Most of the terms used in this report are not abbreviated. However, some abbreviated terms that may not be clear to all readers are:

- **ML**: Machine learning

- **NBA**: National Basketball Association

- **NN**: Neural network

- **PF**: Power forward

- **PG**: Point guard

- **SF**: Small forward

- **SG**: Shooting guard

- **SSH**: Secure Shell

1 Introduction

The primary objective of this project is to build a machine learning system to predict winning line-ups of basketball players for daily fantasy basketball competitions. A winning line-up is any line-up that, upon being entered into the competition, returns more money than the entry fee (gains a profit). We hypothesized this could be a successful machine learning project since we believe that there are unknown relationships between players in a match, both on the same team and on opposing teams. The project was separated into milestones:

1. Learn the necessary material regarding topics in machine learning

2. Design a system architecture

3. Collect a sufficient amount of data

4. Find relevant tools and technology to implement the designed architecture

5. Test the system, and improve

The importance of this project has two components, one personal and one related to the sports industry. The personal importance of this project has to do with the team working on it. This project has been, and will hopefully continue to be, a valuable learning opportunity. It has allowed the creators to learn more about data collection, machine learning, software projects, best practices, and engineering design. On the other hand, with respect to the sports industry, given a system that can accurately and consistently predict winning line-ups for fantasy basketball, the way in which fantasy sports are played may change. Rather than rely on intuition, or optimization techniques alone, there may be an incentive to use a machine learning approach if the lineups from such a system are higher scoring. This could not only change how fantasy competitions are played, but could be adapted to select which players should actually be played, in real life. Theoretically, a coach could use the same technique, although with a different success metric (winning a game instead of maximizing fantasy points) to figure out who to play. In both cases, machine learning could help find patterns that we did not know exist.

This report is structured as follows. First it will provide background on the fantasy basketball competitions that we targeted. Then, it will briefly summarize the requirements of the system. Following that, it will provide a detailed overview of the design of the system, and the results. Finally, it will discuss future plans, the impact on society and the environment, and how the work was divided amongst the authors.

2 Background

Note that most of the content in this section is directly taken from our report for the previous semester, since the background is nearly identical.

2.1 Inspiration

The inspiration for this project came directly from the paper, "Picking Winners" written by David Hunter, et al. The paper targets "top heavy" daily fantasy hockey competitions and uses an integer programming (IP) optimization technique to try to succeed at them. The optimization technique involved the researchers coming up with ideas about what *could* be beneficial for a line-up, building a model around the idea, and then optimizing and testing it. For example, one of their ideas

was related to choosing a defensive player paired with a forward on the same team. They justified this idea by arguing that a single goal would be worth more, as points would be awarded to the defenceman who assisted the forward, and to the forward who scored. Upon reading this paper, we felt that the same goal could be achieved without the necessary prior knowledge of the sport. Instead, we felt that a NN could determine these relationships independently, removing the sports acumen required by the human designer.

2.2 Neural Networks

As we began exploring machine learning techniques, we decided, through the help of our research as well as our supervisor, that a neural network would allow for an effective implementation of our system. Neural networks are a set of interconnected nodes, wherein the inputs are converted to outputs by passing them through a series of functions [2]. Figure 1 gives an illustrative view of a neural network. The architecture of neural networks is inspired by the structure of a human brain, where the nodes represent neurons, and the links that connect the nodes represent synapses [2].
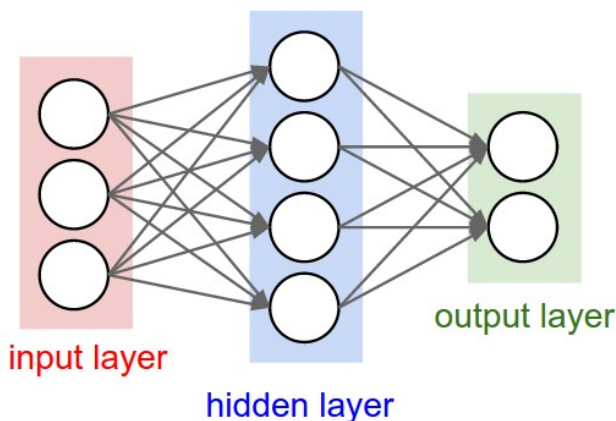

Figure 1: Illustration of neural network [2]

As shown in Figure 1, a neural network consists of an input layer, an output layer, and one or more hidden layers. Not surprisingly, the input layer takes in the inputs to the system, and the output layer yields the outputs of the system. For example, the input to a particular neural network might be two basketball teams, and the output might be the team we expect to win the match. The question remains, however, how the neural network determines which output to generate based on a given input. This is done through the help of the activation functions in the nodes of the network, as well as the weighted links that connect all the nodes [4]. The combined effort of these two components are what allow learning to take place in a neural network.

In doing our research, we had to ask ourselves whether machine learning, and neural networks in particular, would be appropriate as a means of solving our particular problem. As we found while watching a lecture series on machine learning by Professor Yaser Abu-Mostafa, there are three conditions that should be met in order for machine learning to be considered a viable option [3]. First, there needs to exist a real (albeit potentially unknown) mathematical relationship between the inputs and the outputs of the system. For our problem, we postulate that this condition is met, since there exists a relationship between the players playing in a match, and the ideal fantasy lineup. Second, machine learning is especially appropriate when this relationship is not

known or well defined. We felt this was also true, as there are many variables at play in any given basketball game, without a clear relationship as to how they relate to fantasy points. Third and finally,, there needs to be enough data to adequately train the NN. With access to recorded NBA data dating back to 1970, we felt that this condition was satisfied.

Seeing as all three conditions described above were satisfied, we deemed a NN to be an appropriate solution for the matter. In theory, sufficient training of the network should allow learning to take place such that it be able to solve our problem of accurately predicting high-performing fantasy basketball lineups.

3 Requirements

3.1 Design Problem

The problem we are trying to solve is to beat daily fantasy basketball competitions using machine learning with neural networks. In order to fully understand this problem, it is important to detail the specifics of these daily fantasy basketball competitions.

3.1.1 Competition Choice

Although fantasy competitions are often thought of as long-term league competitions with drafting and brackets, these were not the types of competitions that we targeted. There are many different types of competitions with varying rules. Instead of trying to create a system that can compete in all of these competitions, we decided to target a subset, which met a set of criteria:

- Daily fantasy competitions

- Low entry fee (< $10)

- Allow for multiple entries

- High prize pool and participant count

Daily refers to the fact that the competitions are on a given day where real basketball games are being played and involve only the games being played that day, with no residual effects on future competitions. Fantasy refers to the fact that the competitions use an arbitrary points system. Allowing multiple entries means that multiple lineups can be submitted to the same competition. A header for one of these competitions from FanDuel.com can be seen in figure 2. This header shows the low entry fee of $7, the high prize pool of $100,000, and that multiple entries are allowed. It also shows a list of the real games that are occurring that night.

Figure 2: Daily NBA fantasy competition header

One of the criteria of the Picking Winners paper was that the prize pool be "top-heavy". With top-heavy competitions, "contests give a disproportionately high fraction of the entry fees to the

top scoring lineups." [5]. The entry fees for these competitions are usually less than $10, and the top placing competitors usually win thousands of dollars. The payout structure of a top-heavy competition with a $4.44 entry fee and $400,000 prize-pool can be seen in Figure 3.

1st:	$100,000	13th - 15th:	$300	446th - 620th:	$18
2nd:	$40,000	16th - 19th:	$250	621st - 870th:	$16
3rd:	$20,000	20th - 25th:	$200	871st - 1270th:	$15
4th:	$10,000	26th - 35th:	$150	1271st - 1770th:	$14
5th:	$5,000	36th - 50th:	$100	1771st - 2370th:	$13
6th:	$3,000	51st - 70th:	$75	2371st - 3170th:	$12
7th:	$2,000	71st - 110th:	$50	3171st - 4170th:	$11
8th:	$1,000	111th - 160th:	$40	4171st - 6165th:	$10
9th:	$750	161st - 230th:	$30	6166th - 9165th:	$9
10th:	$500	231st - 320th:	$25	9166th - 13164th:	$8
11th - 12th:	$400	321st - 445th:	$20	13165th - 23308th:	$7

Figure 3: FanDuel daily fantasy competition payout

One can see that the monetary difference between first and tenth place is 200 times, whereas the monetary difference between tenth and one-hundredth is only 10 times. The idea behind participating in a top-heavy competition is that, even if many of your lineups do not win, one or two may win a lot, offsetting the losses of the other entries. Hunter et al. [5] felt that some of their lineups could come in the top 10 of these competitions, and that they thus had a higher expected value participating in these top-heavy competitions. One reason we felt we also had a chance at a high ranking entry is that we believed that our system could return uncommon selections of players for our lineups that other methods (intuition or optimization) would not return. The reason that this could be beneficial is because, usually, less-picked players are cheaper, and thus a good but less-picked player would generate more points per dollar spent, allowing for a higher total score. However, we were not strict on this requirement, and participating in other types of contests, including 50-50 contests, where the top 50% of competitors are given near double their entry fee.

3.1.2 Entries

After selecting the competition to enter, the participant must create an entry, or lineup. This lineup must comprise players that are playing in the real games on that day. Additionally, the lineup must have a certain amount of specific basketball positions. A common requirement is that the lineup must have two point guards (PG), two shooting guards (SG), two power forwards (PF), two small forwards (SF), and one center (C). Note that these positional requirements are not necessarily representative of a real basketball team. Figure 4 shows the lineup selection interface for FanDuel.com. On the right side, one can see the required positions for a given competition, along with the lineup's budget, in the top right corner. This budget is the amount of money each contender is allowed to spend on their lineup. On the left side, one can see the price of each player, along with their position, and their average fantasy performance for the season, on which the price is dependent.

Figure 4: Fantasy basketball competition lineup

3.1.3 Scoring

The participant's goal is to have the lineup with the highest score. The score is based on how well their selected players perform in real life, and is based on the amount of free throws, three pointers, field goals, blocks, steals, assists, and turnovers each player gets. The scoring system for competitions on FanDuel can be seen in Figure 5.

Figure 5: Fantasy basketball competition scoring

3.2 Requirements and Constraints

In order to solve the described problem, we had to comply with the rules of the competitions mentioned above. However, we also had to abide by other requirements and constraints, detailed in this section.

3.2.1 Functional Requirements

- The system should be able to use all previous NBA data

- The system should be able to output lineups for daily fantasy competitions based on which teams are playing, how much each player costs, and which positions are needed

10

- The system should output a projected score for each lineup

3.2.2 Usability Requirements

- The system should use a MySQL database

- The database should be secure and have personalized credentials for all users

- The system source code should be version controlled

3.2.3 Technical Requirements

- The system should be able to train to convergence on the order of minutes, so as to allow for cross-validation on the order of hours (or at most, a couple of days)

- The system should be modular, allowing for easy architectural alteration (e.g. changing a scoring algorithm)

- The system should also allow the NN hyperparameters, such as the learning rate, amount of data used for training, and architecture, to be altered easily

3.2.4 Constraints

- The system should not require any expensive services

- The system should be trainable on our personal computers (but not necessarily be able to be cross-validated on our computers)

- The system must be completed within eight months (two university terms)

4 Design and Results

There are two major sections into which the design solution can be divided. These two sections are data and implementation. The first section, data, takes heavily from last semester's report, as there were only minor changes.

4.1 Data

The data section of our design solution includes anything relevant to data, including collecting, storing, and accessing the data.

4.1.1 Collecting Data

Initially we had two candidates for the primary source of obtaining all NBA player and team related statistics. These two sources were:

- stats.nba.com

- basketball-reference.com

Both sources contained similar amounts of information on player matches, player season averages, teams, and matches. Basketball Reference was slightly more detailed in some aspects, such as player season averages and player matches. However, we decided to use stats.nba.com as our primary source, since we felt the official statistics platform of the NBA would have more reliable data compared to that of an unaffiliated site.

The collection of data from stats.nba.com was done by finding a webpage with pertinent data, scraping that data using Python, and storing it in a MySQL database. We automated the process of scraping the data by writing python scripts that iterate through all web pages of a specified type and searching through HTML tags to retrieve the desired data. This usually entails locating an HTML table, iterating through its rows and columns, and retrieving stats for points, assists, rebounds, etc. Two python libraries were used to build the scraper:

- BeautifulSoup4

- Selenium

BeautifulSoup allows for objectification of HTML tags on a web page such that can be accessed through python. Selenium automates the usage of a chosen browser. While, in most cases, BeautifulSoup alone would be enough to access the HTML elements of a given web page, stats.nba.com generates all of the data that we need through JavaScript. This leads to a problem, as BeautifulSoup cannot tell when a website using JavaScript has fully loaded. This is solved by using Selenium to automate browser control, as we can specify that a page is not fully loaded until all of its JavaScript components have fully loaded.

4.1.2 Storing Data

With respect to storing data, we considered two potential methods, each with their own advantages and drawbacks. The methods of storing data were:

- local database

- remote database

The method of using a local database was enticing due to the lack of potential bottleneck associated with the connection bandwidth of a remote database. However, while examining the data on stats.nba.com, we saw that there could be the potential for needing database schema changes, as the structure was sometimes changing. In the case of a local database, this would have created an inconsistency in the local copies of each group member.

A remote database solves this issue, and also has other benefits. For one, it offers the ability to update a persistent table in parallel. In other words, multiple users can use multiple tables simultaneously to read, insert, or delete entries. Ultimately, this convenience was more beneficial than the cost of overhead in access time to a remote database.

4.1.3 Accessing Data

Each group member was given credentials with which he/she could access the remote database through SSH. This was possible from any location with internet access.

4.2 Implementation

We began designing the software architecture before finishing the aforementioned database design. Our initial design was a single NN that took in, as input, all of the stats of every active NBA player. Additionally, it had an extra input for every player which was set to 1 if the player was playing tonight, and 0 if the player was not. These were called the "who's playing indicators". The output of this NN was the best lineup for the competition. A block diagram of this system can be seen in Figure 6.
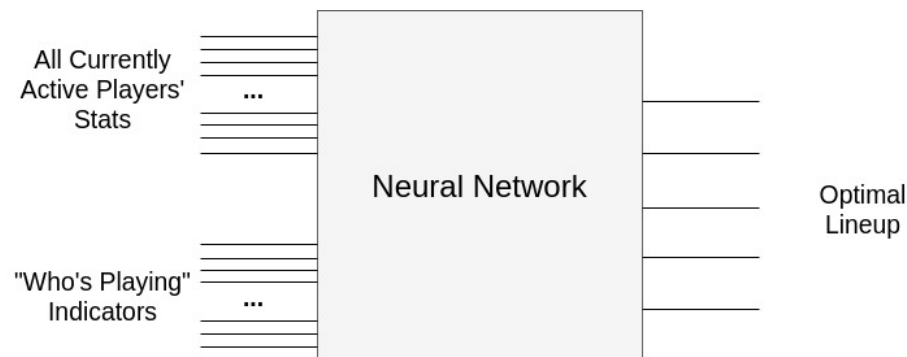

Figure 6: First design iteration

After discussing this design within our group and with our supervisor, we realized it had major limitations, listed below. First, this system would only use data from active players. This was not taking advantage of the many decades of basketball data to which we had access. Second, it would also require a massive amount of data to be able to learn properly. This is because, generally, the amount of data required to learn adequately is correlated with both the complexity, and the number of, inputs (as discussed in COMP 551). Third, there would be many unused inputs each time the system is used. Since there are usually around six NBA games each night, there are 120 active players on a single night. There are about 500 total active players, which means that there would be about 380 unused inputs. Although this is not explicitly a problem, it makes it seem like there would be a better way to structure the design. The final limitation of the initial NN design was that it would be prone to discover imaginary correlations. Since the NN only knows which players are playing, and not on which team, it would be bound to find correlations between players that are not playing against each other. To explain, if Team A is playing against Team B in one game, and Team X is playing against Team Y in another, the NN may find a relationship between players on Team A and on Team B. This is despite the fact that they are not playing against each other, and that these games are independent.

Taking these limitations into account, we designed an improved architecture that was more flexible and required less data to train. We decided on a three step system, of which only one part would involve machine learning. The first system was used to calculate players' scores, the second system used these as inputs to predict player game scores, and the third system selected optimal lineups. An overview of this software architecture can be seen in Figure 7.

These three systems will now be looked at in detail.

System 1: Calculating Player Scores

The first step of the design solution was to calculate a score for each player who has played in the NBA. This score is a representation of how good the player is at scoring fantasy points "in a vacuum". That is to say, how well a player will perform with respect to the fantasy system, if we
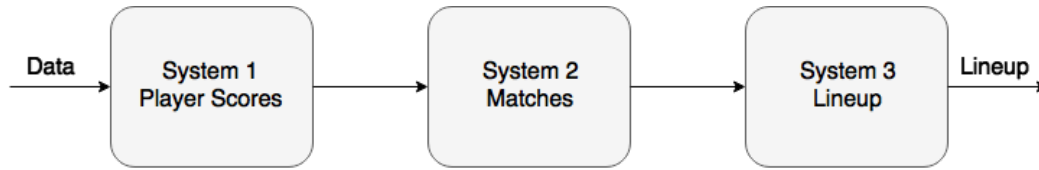
13

Figure 7: Redesigned System Architecture

ignore the match-specific variables, namely who is on his team and who he is playing against. Calculating the most accurate score to represent this is not an exact science. The most straightforward solution was to use the same points system as FanDuel for calculating fantasy points. Since we wanted to spend more time focusing on the second system (NN), we opted for this simple equation, the details of which can be seen in Figure 4. Due to the modularity of the code, this score can be easily changed and validated on, and ways to do that will be discussed in future plans. With this formula, we calculated two different types of this score for each player: a short term score, and a long term score. The short term score used the player's average statistics of the last n (chosen as 5) games. The long term score used the player's average statistics of their previous season. In the case where it was a player's first season, it took his current season's statistics so far as input.
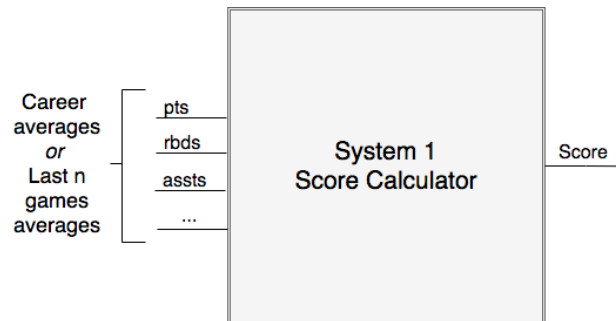

Figure 8: Player scores system

System 2: Matches Neural Network
The scores for every player, obtained in the first step, are then used as inputs for the matches NN. The matches neural net puts two teams of players, each represented by two scores, "against each other". That is to say, the first fourteen inputs are used for seven of Team A's players (short term and long term score each), and the next fourteen are used for seven of Team B's players. We selected seven as a compromise between being too restrictive (not all teams have many active players in each position), and not having enough inputs. The players on each team are ordered by their basketball position. The basketball positions given by our source are guard (G), forward (F), and center (C). As not each team plays the same amount of players in each position, we had to determine what a representative combination of the above positions would look like, which will be discussed in the Data Results section. Ultimately, we decided that a representative set for which we had data would be one center, three guards, and three forwards.
The output of the NN is a "fantasy game score" for each player. The benefit of this NN is that, since all basketball matches follow the same rules, with the most impactful variable being the skill level of each player on either team, we can train this net on every match that has happened in the

14

past 50 years, and still use it to predict scores for matches today. A point guard in 1980 who had incoming scores of [10, 10] will be treated the same as a point guard today with those scores. This NN has a relatively low amount of inputs, none of which are unused. This means it needs less data to train successfully. This design can be seen in Figure 9.
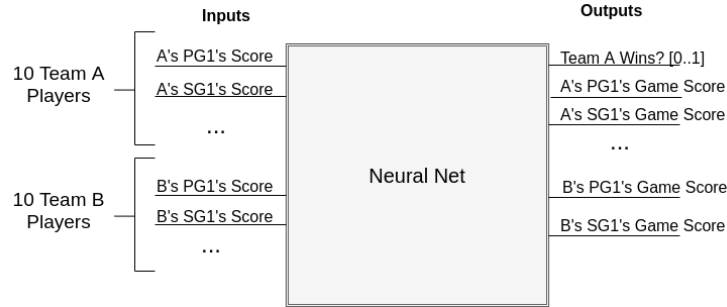

Figure 9: Desired complete matches neural network

Initially, this network was set to only predict the team that won or lost, and not the game scores of the players. This first prototype was thus not applicable for fantasy basketball, but was used to become familiar with TensorFlow and best practices. We experimented with different input features to see how they impacted the accuracy of the network. In the end, with these ordered positions, the network reached a 63% prediction accuracy. We did not spend much time attempting to increase this value, as it was an aside from our main goal, predicting lineups.

System 3: Choosing Lineups
The choosing lineups system takes as input the player game scores, salaries, positions of each player in tonight's games, as well as the total budget we are working with, and outputs lineups with maximized summed player scores. This problem was similar to a backpack problem, but with more constraints. We selected the python library PuLP as the tool to use to solve this constraint problem, and found it to be quite effective. As well, we were able to use this system to generate multiple lineups by applying Gaussian noise to the players' scores before passing these to the system.

The objective function we sought to maximize had the following structure:

$$z = p_1 * x_1 + p_2 * x_2 + ... + p_n * x_n$$

In this, z is the objective function, n is the number of players we have on a given night, each p variable represents one of the n players, and each x variable is a binary decision variable that will indicate whether or not the corresponding player has been chosen or not.

Maximizing the objective function was subject to six constraints. The first five constraints specified the number of players of each position type we were allowed to have in our lineup. Specifically, the constraints of our fantasy league require our lineup to have exactly two point guards, two shooting guards, two small forwards, two power forwards, and one center. The constraint specifying the number of allowed point guards, for example, looked like this:

$$x_1 + ... + x_j = 2$$

where x_1 to x_j are the binary decision variables of the point guards playing on a given night. The remaining four positional constraints are structured in the same way.

15

The final constraint ensures that we respect the total budget we are working with when purchasing players. It simply says that the the sum of the costs of the players selected cannot exceed the total budget we have to work with. This is shown in the equation below:

$$p_1 * x_1 + p_2 * x_2 + ... + p_n * x_n <= TotalBudget$$

As was the case in the objective function, n is the number of players we have on a given night, each p variable represents one of the n players, and each x variable is the binary decision variable for the corresponding player.

Although there exist several ways of solving integer linear programming programs, the python PuLP library handles this decision for us. Inputting the objective function and constraints mentioned above is sufficient for PuLP to find an optimal lineup solution.

Cross Validation & Testing

In order to properly test this system, we needed to define a success metric, or a way of measuring how well the system was performing. Ideally, this metric would be profitability, but this was not possible without the historic competition data. Instead, we used a heuristic which is given in Equation 1

$$Performance = \frac{MaximumTheoreticalLineupScore}{ActualScoreofPredictedLineup} \tag{1}$$

This formula yielded around 60% with random choices between the top seven players on each team. In order to determine what percentage needed to be reached to be profitable, we examined current competitions, and found an effective rule. A system was profitable in most of these competitions when this performance metric reached 80%, i.e. when the score reached 80% of the maximum theoretical score. Figure 10 shows this relationship, where the last winning score is roughly 79% of the maximum. In order to increase our performance, we began to train the network. In

Figure 10: Scores for best lineup and barely profitable lineup

doing so, we realized that one of the limitations of training and attempting to cross-validate the hyperparameters was the computational time. To remedy this, we rented a remote computer with an NVIDIA P4000 GPU, and changed our machine learning code to be GPU optimized by switching from TensorFlow to the GPU enabled version of TensorFlow. The service we used was called Paperspace, and it was chosen for its low hourly rates with no commitment requirement. We ran cross-validation on Paperspace, and were able to increase our performance metric to 82.9% - slightly above the profitability line. Thus, we began testing in actual competitions; the results of doing so will be discussed in the implementation results section.

4.3 Results and Discussion

There are two types of results that will be given in this report. The first are those related to how much data was scraped from our source, or generated, and the second is related to the success rate of the neural network.

4.3.1 Data Results

We managed to scrape, collect, and otherwise compute a significant amount of data to be stored in our database. We were able to gather all data for every basketball game since 1979. We have data for 42000 matches, 10000 player-seasons (unique players for a specific season), 2000 players, and 36 teams, including some that are no longer active. For each match, there are about 15 players with statistics for that match. This is entered in our "player matches" table, and we currently have data in this table for about 25% of matches. This corresponds to 250,000 entries, and yields about 13000 inputs for our neural net. However, the different architectures which have been presented all impose different constraints on the inputs themselves. For example, our neural network with ordered player scores required the match input to have both teams with at least the minimum number of players (7 in this instance). The neural net with players ordered by position had even more constraints. Not only did it require a minimum number of players per team, but it also needed a minimum number of players per *position*. When performing an analysis on the different player positions per lineup, we found significant variation between teams. This was complicated even further as some players played multiple positions, denoted by two positions separated by a hyphen. For example, the position "F-G" meant that the player could either play a forward or a guard. We found 21 possible combinations of these positions that teams could play. As an example, playing "C, G, G, F, F, C-F" could be a single combination, whereas "F-G, C, C-F, F, F, G, G" could be another. We found that none of the possible 21 combinations had enough data to train our neural network on. Even our initially designed input format of requiring two guards, one center, and two forwards per team (meaning only five inputs per team) caused roughly 21% of inputs to be discarded, simply because these inputs did not meet the criteria of having these specific numbers of players in each positions.

To avoid losing significant amounts of input data due to constraints, we simplified our input scheme by combining the multi-position players with the single-position players. Our methods of combining these players is shown in Figure 11. Note that the first combination has zero centers (hence no blue bar). In using this scheme, we were able to reduce the amount of lost data to a mere 11%.
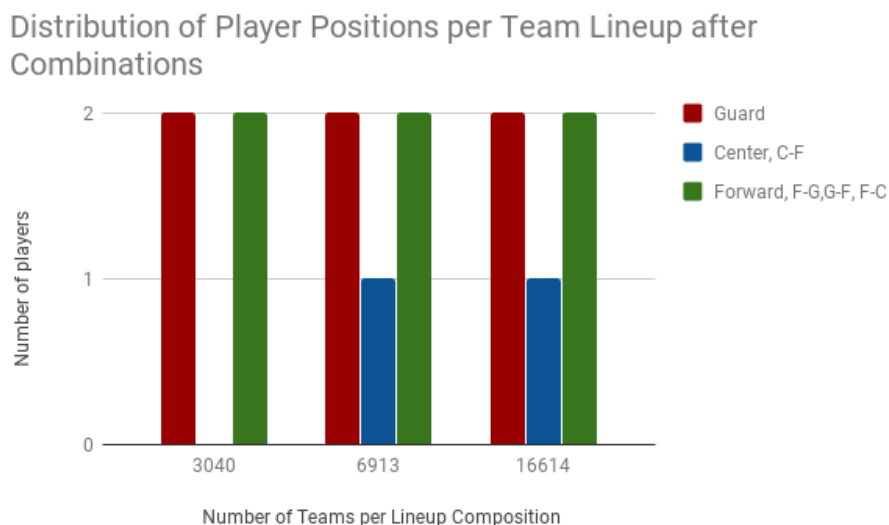

Figure 11: Team Compositions by Player Positions After Combinations

Table 1: Results of Cross-validation for Top 10 and Bottom 10

Batch Size	Hidden Layers	Learning Rate	Dropout	Train Performance	Test Performance
5000	64-64	0.01	0.25	82.15	78.34
10000	64-64	0.01	0.25	81.99	78.31
5000	128-64-128	0.001	0.25	80.13	78.23
5000	64-64	0.001	0.25	79.74	78.05
10000	64-64-64	0.01	0.25	81.96	77.80
5000	64-64-64	0.01	0.25	82.10	77.80
10000	64-64	0.001	0.25	78.84	77.79
10000	64-64	0.01	0.5	79.55	77.76
5000	128-64-128	0.01	0.5	80.48	77.73
...	...	...	...	...	...
10000	32-32-32	0.001	0.5	58.00	58.27
10000	64-64-64	0.1	0.5	56.68	56.73
5000	128-64-128	0.1	0.25	54.29	54.10
5000	64-64-64	0.1	0.25	54.29	54.10
10000	128-64-128	0.1	0.25	54.29	54.10
10000	128-64-128	0.1	0.5	54.29	54.10
10000	32-32-32	0.1	0.25	54.29	54.10
10000	64-64-64	0.1	0.25	54.29	54.10
5000	64-64	0.1	0.25	54.06	53.97
5000	128-64-128	0.1	0.5	54.08	53.79

4.3.2 Implementation Results

As mentioned in the description of the NN in the implementation section, we tested many different system architectures. The final three-system architecture will be the only one discussed in this section, as it is the only one used to enter real competitions. For the results of the earlier architectures, please see our report from the previous term.

Note that the provided results results are on the cross-validated version of the neural network. The results of cross-validation, i.e. how the hyperparameters were selected, can be seen in Table 1. We validated on the learning rate (0.001, 0.01, 0.1), the number of hidden layers (2 to 3), the nodes per hidden layer (32, 64, 128), and the dropout (0.50, 0.25). The first ten rows of the table show the best ten combinations of hyperparameters, whereas the bottom ten show the worst performing. One can see the importance of selecting hyperparameters by the significant difference in performance between the best set and worst set of hyperparameters.

Our final network used some of the parameters of the best performing network from crossvalidation, but, through manually adjusting some of these best performing results further, was slightly different. The final configuration had two hidden layers, with 128 and 64 hidden nodes in the first and second respectively, a learning rate of 0.01, and a dropout of 0.25 for the first layer, and 0.20 for the second layer. This configuration yielded a training and testing performance of 83.0% and 82.9%, respectively. Furthermore, all accuracy results were obtained by using 70% of the input data to train the network, and 30% to test.

After selecting these hyperparameters, we further trained the system, and increased the amount of data we used (since more was available at this time). During training, we carefully measured the training and validation performances. One can see how the mean average error (per player score) change with each epoch in Figure 12, for one example network (not the optimal hyperparameters).

Reported in this figure are also the corresponding performance percentages at certain positions, including the start, the last meaningful point before overfitting, and the end. The reason both the training and validation accuracies are reported is to detect overfitting. In the case of overfitting, the validation error will increase while the training error will decrease. This can be seen to occur around epoch 2000. Thus, this system would not be trained past 2000 epochs. The ability of the system to overfit indicates that the model is not the limitation in increasing the accuracy of the system.

Figure 12: Mean error versus epoch for sample (non-optimal) network

After training the system, and reaching the maximum testing performance of 82.9% (above the profitability line), we entered the system's first lineups into competitions. Upon doing this, we realized we had overlooked some potential issues. First, the database was not updated, meaning that the short-term scores of each player were actually scores from months ago. Second, we found that we were accidentally ignoring one or two games per night, due to a bug with the positional constraints in our code. Thus, our output was only predicting a lineup on a subset of the games each night. Third and finally, we found that many of our selected players did not end up playing each night, yielding zero points. This third issue proved the most challenging to tackle. Although we checked for injuries, we did not perform a more thorough check than this. Initially, we hoped that we could solve this with more checks in our code, but we quickly learned that this was not feasible. Ultimately, before submitting each lineup, it is an important step for the user to read recent news about the NBA player. Sometimes players do not play for personal issues, or because they were only temporarily playing in the spot of another player. This is both difficult and unnecessary to try to predict. Instead, we added an option for our system to ignore certain players, entered manually.

After solving these issues, we continued entering competitions. To our surprise, the system was successful. Figure 13 shows a brief summary of each competition we participated in where all of our players played. One can see that we are overall profitable, but not reliably so, and on a small sample size. When examining the lineups in detail, one can make some interesting observations. For instance, sometimes the network was able to make good predictions that were missed by other competitors. For each player selected, at the end of the competition, a value is given that shows what percentage of other competitors also selected that player. In our best lineup, our players were picked on average 7.5% by others. By contrast, the nearest competitor's players averaged a 27% pickrate. Our lineup containing these picks can be seen in Figure 14. Note

Date	Contest	Score	Opponent	Entry Fee	Winnings
04/03	NBA Beat the Score: 260 (Beginners Only, $50 Guaranteed)	303.8 (1 of 52)	Tournament	$1	$1.22
04/03	NBA Beat the Score: 275 ($500 Guaranteed)	294.4 (1 of 595)	Tournament	$1	$1.04
04/01	$2K Sun NBA Fadeaway ($0.25 to Enter)	283.3 (3916 of 9580)	Tournament	$0.25	$0
04/01	$2K Sun NBA Fadeaway ($0.25 to Enter)	289.6 (3162 of 9580)	Tournament	$0.25	$0
03/30	$5K Fri NBA Rabies Shot ($5K Guaranteed)	324.4 (151 of 1340)	Tournament	$4.44	$10
03/28	$15K Wed NBA Block (Single Entry)	319.8 (830 of 8929)	Tournament	$2	$4
03/28	$15K Wed NBA Dribbler (Single Entry)	287.5 (5459 of 17857)	Tournament	$1	$0

Figure 13: Competition record for submitted lineups

that a typical score value for a player with a $5000 salary is about 15.

Figure 14: Good rare picks of our system versus those of a near competitor

This supports our hypothesis that our system would be able to make good predictions that were missed by others, thus giving it the potential to perform quite well in top-heavy competitions.

It is also worth noting that, even in the competitions that the system does not succeed, it is still performing above average, always in the top 50th percentile. This is especially remarkable since almost all of the other competitors are marked as experienced. Figure 15 shows a sample of the winning and losing players for a competition, including their experience emblems. Competitors with a white star and blue circle have been in over 1000 competitions and have won at least $1000 across four contests whereas competitors with a blue star and white circle have been in at least 500 competitions or have won at least $1000 across four contests. We measured the percentage of experienced players for a single competition, and found it to be 85%. Our system was able to perform quite well against these experienced players, turning a profit in lineups that had all players active. However, the system is still far from being reliably profitable, as its outputs are

Figure 15: Experience level of competitors

high in variance. There are still improvements to be made.

5   Future Plans

We plan to continue to improve on the design and implementation of this project. We think that there are many areas that can be improved, and would like to divide these into two sections: general improvements, and machine learning improvements.

5.1   General Improvements

There are many ways that the entire system can be improved in a general sense. For one, it would be possible to add additional features (inputs) that could increase the performance of the NN. Next, we could improve the scoring calculation function of system one, either by removing it and instead passing all of the players' stats as input to the NN, or by coming up with a more representative formula that we validate on. Next, we could factor in the standard deviation of each player in order to have a risky value associated with each generated lineup. Lineups with high-variance players are more risky, whereas with more consistent players are less risky. Finally, currently, in order to run the system on a day, it requires running three unoptimized scripts, which take in total around 15 minutes. This could easily be consolidated into running a single script that takes less time. This would make it more convenient to run.

5.2   Machine Learning Improvements

This subsection will detail the different machine learning improvements that we feel could be made on our system. In particular, this involves the second stage of the system, the neural network. Currently, although we performed cross-validation on the network, we did not perform as thorough a cross-validation as would be ideal. As discussed, the system was validated on the number of hidden layers (2 and 3), the number of nodes per hidden layer (32, 64, 128), the learning rate (0.001, 0.01, 0.1), and the amount of dropout (0.25, 0.5). Not only would we like to increase the amount of values for nodes per hidden layer, dropout, and learning rate that we validate on, we would also like to add other hyperparameters, including the amount of days taken for a short term score, the amount of data used (how many years), and the weight on the error of recent games (to make more recent games worth more). Second, we would like to incorporate a machine learning technique to reduce the variance of our predictions. Through some research, we have

found that a technique called Bootstrap Aggregation, or Bagging, would be suitable here. A bootstrapped dataset is a dataset created out of the original dataset. With an original dataset of size n, a bootstrapped dataset would be created by randomly taking n values from this original dataset with replacement. With replacement means that it is possible to take the same value multiple times. Bagging is the process of creating "l" bootstrapped datasets, training the network on all "l" of these, and then averaging the results for each prediction. This ideally will reduce the overall variance.

6   Impact on Society and Environment

This section was taken from the previous report, as the system is unchanged in these aspects from last term.

6.1   Use of Non-Renewable Resources

Seeing as our entire project is made up of software components, the environmental impact, in particular the use of non-renewable resources, is fairly minimal. Having said this, the training aspect of the neural network certainly requires processing power, which consumes energy. The amount of energy consumed by a CPU or GPU varies greatly, but, for high end hardware, the total power usage can reach 500 W [6]. As the amount of data the net is trained on increases, and the amount of times we retrain the net increases, so will the amount of time the computer runs for, and thus so will the total energy consumed. The amount of energy used by our system, and thus the total environmental impact, will be directly proportional to the amount of times the network is trained by all users. Theoretically, if this system was to be released to the public, then the accumulation of everyone training their respective neural networks could eventually become an environmental concern. With one thousand users training a net for an average of ten hours, we would see an energy consumption of about 18 GJ, which is equivalent to charging one million iPhones [7]. Although this may seem insignificant, with one million users (note that there are 60 million people in North America who participate in fantasy sports competitions), [8] the energy used would be equivalent to charging one billion iPhones. Currently, since we are in Quebec, the energy being consumed is almost entirely made up of that generated with renewable resources. However, most of the planet's energy is produced via non-renewable resources[1]. Thus, if the project ends up growing to this scale, it would be important to continue to consider and evaluate the use of non-renewable resources.

6.2   Environmental Benefits

If our technology was developed to the point at which it could be used with relatively low power requirements (e.g. no need for each user to train the neural network), then widespread use and adoption of our software could potentially reduce the amount of energy consumed by the fantasy sports community. Curretntly, fantasy sports participants spend significant amounts of time online trying to figure out how to improve their fantasy teams; some spend several hours a day doing such research [9]. If our system was adopted by a large enough portion of the people playing fantasy sports, and could be run by their computers easily, it could remove all the research and decision making from the hands of these participants. This could then in turn mean they will spend less time on their computers managing their fantasy teams, thus using less energy. This is a somewhat special case, as we would have to develop a version of our product that comes pretrained, runs with low power, and is distributed in such a way that the user does not use more

energy downloading and setting up the software than he/she saves with it. Currently, this is out of the scope of the project.

6.3 Safety and Risk

The most significant risk associated with our project will actually be posed to the fantasy sports industry itself. If we are able to successfully use machine learning to predict the optimal line-ups for fantasy sports, then the industry may switch from a human-human competition to an entirely ML one. After releasing the software, there could also be a high amount of people who were not previously playing fantasy sports who may now see it as profitable. These people may flood in and take over the game. After a while, however, it would no longer be profitable, since everyone playing would have the best, and equivalent, equipment. As well, more traditional fans may argue that it removes the fun or skill of fantasy sports, and may stop playing. If the project turns out to be exceptionally successful, and is released to the public, it could completely change the way that fantasy sports is played, and we see this as a risk to that industry. However, we also feel that it is unlikely that the system is able to outperform all humans, and is certainly unlikely that it would be adopted in such a way as to ruin the industry.

6.4 Benefits to Society

There are a few benefits to society that could result from this project, if it is successful. The first benefit of our project would be to hopefully help people become interested in neural networks and machine learning in general. We argue that this is important since more and more devices use machine learning techniques, and this would lead to a better understanding of the current technology. The second benefit is that it could help further the movement of data-backed sports hiring decisions, which is to use statistics for selecting players, rather than intuition. Michael Lewis writes about this method, and its success, in his book Moneyball [10]. The reason this is a benefit to society is because it could make the hiring process for sport players blinder - more friendly towards minorities. The third benefit is related to expanding the list of domains to which machine learning techniques can be applied. If we are able to demonstrate a successful system for predicting optimal fantasy lineups, then others may try to apply a similar technique to a different problem. The problem could be something more meaningful, such as a pharmaceutical company determining how to select an optimal drug, which is an example mentioned in Picking Winners, by David Hunter et al [5].

7 Teamwork Report

7.1 How We Worked as a Team

In order to facilitate and improve the efficiency of teamwork within our group, we made use of several tools and strategies throughout the semester. For communication within the group, we used the Slack messaging app. For collaboration on code, we made use of git for version control, and github for code reviews. Weekly in-person meetings were had amongst all team members in order to come with fresh project ideas as a group, and to ensure that all of us were on the same page. Weekly in-person meetings with our supervisor were also had for similar reasons, and to ensure that we stayed on track throughout the year. Without the use of these tools and strategies, the development process for this project would have been much more difficult.

7.2 Individual Contributions

Below we show the individual contributions of each person. Note, however, that much work was done as a group, so there are many areas that were worked on collectively by everyone.

7.2.1 Ege

- Responsible for all database and remote computing aspects of the project

- Built the majority of the scrapers used to collect the data

- Designed the database schema for housing the data

- Set up the remote computing environment with Paperspace

7.2.2 Stephen

- Coded the system for computing player scores and updating database

- Created constraints for, and wrote the code for, the linear programming constraint solver (system 3)

- Added ability for system 3 to output multiple lineups

7.2.3 Florence

- Built first prototype of neural network to predicts winning team

- Built second prototype of neural network for predicting player game scores

- Handled the creation of inputs from raw data

- Performed cross-validation on the network

- Built function to connect systems and output a lineup with Asher

7.2.4 Asher

- Worked on scripts to compute player scores, and scripts to connect tables in the database

- Worked on data retrieval code from database to obtain data in a format appropriate for the neural network.

- Helped to implement scrapers for some player statistics

- Built function to process daily competition rules

- Built function to connect systems and output a lineup with Florence

7.3 Evaluation of Teamwork and Difficulties

The four of us worked very well together as a team and did not run into any major issues. We plan to continue working together on this project through the summer, and have a superior product for the next NBA season.

8   Conclusion

Our design project has been a success in many ways, although there are still improvements to be made. We were able to design an architecture for our system that allowed us to train a neural network on historic basketball games and use this neural network to predict optimal lineups for future fantasy basketball competitions. These lineups ended up profitable, although not reliably so, and only for a small sample, so far. This system, consisting of three components, including a neural network to estimate player fantasy points on a given day, was tested and cross-validated in order to maximize performance. However, performing a more thorough cross-validation would be the first step in improving its performance. Further to this, we plan to improve this system in other ways. This includes making it more reliable through the application of other machine learning techniques and also the improvement of the data used and the scoring system used.

References

[1] "How much of U.S. energy consumption and electricity generation comes from renewable energy sources?", U.S. Energy Information Administration, 2017. [Online]. Available: https://www.eia.gov/tools/faqs/faq.php?id=92.

[2] Karpathy, "Convolutional Neural Networks for Visual Recognition", Stanford. [Online]. Available: http://cs231n.github.io/neural-networks-1/.

[3] Abu-Mostafa, Yaser. "Learning From Data", Caltech, 2012. [Online]. Available: https://work.caltech.edu/telecourse.

[4] "A Basic Introduction To Neural Networks", University of Wisconsin-Madison. [Online]. Available: http://pages.cs.wisc.edu/ bolo/shipyard/neural/local.html.

[5] Hunter, David Scott, et al. "Picking Winners Using Integer Programming." *ArXiv:1604.01455 [Stat.OT]*, 6 July 2016.

[6] Kaminski, J. (2009). "How much power does your video card use?." CNET. Available: https://www.cnet.com/news/how-much-power-does-your-video-card-use/.

[7] Mayo, B. (2015). iPhone 6s Plus battery rated 2750 mAH, 5% smaller capacity than iPhone 6 Plus. 9to5Mac. Available: https://9to5mac.com/2015/09/21/phone-6s-plus-smaller-battery/.

[8] Kounang, N. (2016, March 18). Inside the brains of bros on fantasy sports. Retrieved April 17, 2018, from https://www.cnn.com/2016/03/18/health/fantasy-sports-psychology/index.html

[9] YouGov. "How Much Time Do You Spend on Managing Your Fantasy Football Team?*." Statista - The Statistics Portal, Statista, www.statista.com/statistics/284718/amount-of-time-spent-on-managing-fantasy-football-team/, Accessed 30 Nov 2017

[10] Lewis, Michael. "Moneyball: The Art of Winning an Unfair Game". New York: W.W. Norton, 2003.