# Optimizing Predictions of March Madness via Neural Networks

15.053 Team Project

Abraham Shalom, Aidan Einloth, Mercedes Riley, Emily Weinschreider

# Introduction

The NCAA Division I Men's Basketball Tournament (known informally as March Madness) is a single-elimination tournament that has been played each spring in the United States since 1939. It features 68 college basketball teams that have either won their division conference or have been selected by a NCAA selection committee. It has become one of the most famous annual sporting events in the United States, and the American Gaming Association estimates that over 70 million tournament brackets were created in 2017 alone. Even though millions of brackets have been made, there has never been a “perfect bracket,” or one that accurately predicts the winner of every game. Because the odds of picking a “perfect bracket” are so slim, ESPN has created a scoring function that delegates points for the number of winners the bracket accurately predicts, with increasing point values for later rounds.

Our goal in creating this optimization model is to maximize the number of winning teams that we are able to accurately predict, which will in turn lead to a high score. We aim to train on regular season statistical data from 2003-2013 that has been compiled by the teaching faculty and use that to predict which teams are going to win. We will predict every possible matchup between the 68 teams, resulting in 2,278 predictions for each year, and we will use these definitive predictions to generate a bracket.

# Data Source

Before explaining the model and how to solve the problem, we will introduce the data and the form that it was given.  All data used was given by the 15.053 staff in the form of csv files.

*teams.csv*: This file contains 4-digit IDs assigned to every team so teams can be identified by this number within other datasets.

*RegularSeasonDetailedResults.csv*: This file contains regular season statistics for every team from 2003-2013 which will be used to create feature vectors for each team.

The statistics utilized in our model included:

- Field goals made (fgm)
- Field goals attempted (fga)
- Three pointers made (fgm3)
- Three pointers attempted (fga3)
- Free throws made (ftm)
- Free throws attempted (fta)
- Offensive rebounds (or)
- Defensive rebounds (dr)
- Assists (ast)
- Turnovers (to)
- Steals (stl)
- Blocks (blk)
- Personal fouls (pf

*TourneyDetailedResults.csv*: This file contains the tournament results from 2003-2013. The data includes the two teams ID numbers, who won, the score, and the statistics from the game. This information will help us learn what regular season statistics perform best in the tournament over the years.

*TourneySeeds.csv*: This file identifies the seeds for all teams in the NCAA tournament from 2003-2013. The data includes the year of the season, the ID number of the team, and the seed which is represented by a letter, (W,X,Y, or Z), for the region followed by a two digit number from 01-16 which is the seed within the region. 01 is the best and 16 is the worst.

*TourneySlots.csv*: This file identifies how teams are paired within the tournament. This file makes the structure of the bracket which means it tells which seeds play each other and in what round. The data included in this file is the year of the season, the slot which is a way to uniquely identify every game, the strongseed, which is the expected better seed, and the weakseed, which is the expected worse seed. The slot is four characters and tells us what round we are in, (R1,R2,R3,R4,R5,R6) along with the expected winning seed, with the seed corresponding to the seed defined in TourneySeeds.csv. An example is R1W1, round 1 with expected winner of W1.

# Model Overview

To solve this problem, we created a quantitative representation of each team. We then compared the teams to determine who should win each game. This process led to a completed bracket based on the way we valued the teams. Ultimately, our optimization model is to minimize the error in prediction, which in theory maximizes the score of the bracket based on ESPN's score function. Below we will outline the steps to the model in more depth.

1. Build team vectors:
    - We construct a vector to represent each team during each season.
    - It is composed of the team's averages on the statistics provided in the file, all of which are included except personal fouls, along with its seed for the tournament.
    - This allows us to have a quantitative representation of each team, which we will use to build our predictive model.
    - After narrowing down our team database for each season to only those 64 (or 68) included in the tournament, we normalize our data so that each value is between 0 and 1.
2. Define matchup vectors:
    - We define a matchup as the difference between the first team's and second team's representative vector.

- Doing this will allow us to have a representation of each matchup that shows the difference in quality between the two teams that faced off.

**Example of Pre-Normalized Teams and Matchup Vector**

| **Team 1** | | **Team 2** | | **Matchup: Team 1 and 2** | |
|---|---|---|---|---|---|
| score | 69.37500 | score | 82.454545 | score | -13.079545 |
| fgm | 24.75000 | fgm | 29.757576 | fgm | -5.007576 |
| fga | 54.62500 | fga | 61.575758 | fga | -6.950758 |
| fgm3 | 6.40625 | fgm3 | 6.878788 | fgm3 | -0.472538 |
| fga3 | 17.84375 | fga3 | 19.303030 | fga3 | -1.459280 |
| ftm | 13.46875 | ftm | 16.060606 | ftm | -2.591856 |
| fta | 18.75000 | fta | 23.212121 | fta | -4.462121 |
| or | 11.06250 | or | 11.787879 | or | -0.725379 |
| dr | 24.96875 | dr | 25.939394 | dr | -0.970644 |
| ast | 11.90625 | ast | 14.696970 | ast | -2.790720 |
| stl | 6.37500 | stl | 7.363636 | stl | -0.988636 |
| blk | 3.87500 | blk | 5.090909 | blk | -1.215909 |
| seed | 8.00000 | seed | 3.000000 | seed | 5.000000 |
| **Name: 1274 (Miami)** | | **Name: 1199 (FSU)** | | **class** | **1.0 (Miami wins)** |

**Example of Post-Normalized Teams and Matchup Vector**

| **Team 1** | | **Team 2** | | **Matchup: Team 1 and 2** | |
|---|---|---|---|---|---|
| score | 0.767731 | score | 0.912475 | score | -0.144743 |
| fgm | 0.733827 | fgm | 0.882300 | fgm | -0.148473 |
| fga | 0.828793 | fga | 0.934253 | fga | -0.105460 |
| fgm3 | 0.611058 | fgm3 | 0.656131 | fgm3 | -0.045073 |
| fga3 | 0.675740 | fga3 | 0.731002 | fga3 | -0.055263 |
| ftm | 0.712189 | ftm | 0.849239 | ftm | -0.137050 |
| fta | 0.709121 | fta | 0.877878 | fta | -0.168757 |
| or | 0.702043 | or | 0.748077 | or | -0.046034 |
| dr | 0.807021 | dr | 0.838394 | dr | -0.031372 |
| ast | 0.554170 | ast | 0.684062 | ast | -0.129892 |
| stl | 0.614023 | stl | 0.709245 | stl | -0.095223 |
| blk | 0.570871 | blk | 0.750000 | blk | -0.179129 |
| seed | 0.500000 | seed | 0.187500 | seed | 0.312500 |
| **Name: 1274 (Miami)** | | **Name: 1199 (FSU)** | | **class** | **1.0 (Miami Wins)** |

3. Generate training data:
    - We use these matchup vectors to generate a training set:
        i. For classification models, we label each matchup vectors with a 1 if team one wins, 0 if team two wins.
        ii. For the regression models, we label each matchup vector with the difference between the final scores of team one and team two.
4. Classification:
    - Using this training data, we build a model that will predict:
        i. 1 or 0 if doing binary classification
        ii. The margin of victory if running a regression

# Method of Solution

After defining and building our matchup vectors, along with their respective classifications, we faced the challenge of actually implementing the binary classification. Since March Madness is known for being hard-to-predict, we believed that the data would be highly nonlinear. Due to this, we decided that implementation of support vector machines or linear regression would not be effective at accurately predicting matchup outcomes. Instead, we turned towards building neural networks. Although these have the disadvantage of being difficult to understand and calibrate, they allow us to approximate almost any function, including our highly nonlinear dataset.

First, a brief introduction to neural networks. A neural network is composed of several hidden layers, where each hidden layer contains units. Each unit has an associated activation function and offset term. The layers in each unit are connected by edges, where each edge has an associated weight. This weight multiplies the input to the unit before adding the offset and applying the activation function. The output of this function is then passed to the next layer as input, undergoing all the aforementioned transformations again. The final layer contains a special activation function, which varies depending on the expected output of the network.

As a modeler, one has many decisions to make when designing a neural network that balances the ability to fit any function without overfitting to the data. The modeler must choose the amount of hidden layers, the amount of units in each layer, and the activation functions for each layer. These have a major impact on the predictive power of the model, and there is no prevailing methodology to build an effective network. Instead, it is up to the modeler to try different options for each parameter.

We built three different neural networks in Python, using three different approaches in an attempt to improve predictive power, and later compared the results.

**Link to code:**

**https://github.com/abishalom/MarchMadness/blob/master/053%20Project.ipynb**

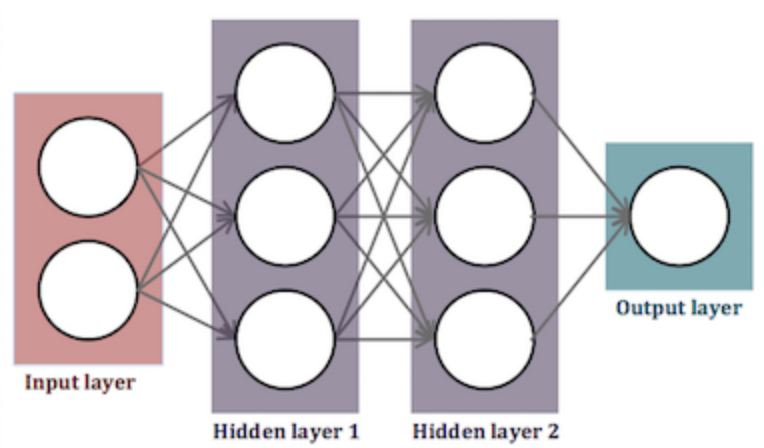


**Figure 1. Neural Network: An Abstract Representation**

Our first neural network is a binary classifier, which takes matchup vectors as inputs and outputs a 1 if team one is predicted to win, or a 0 if team two is the predicted winner. In this neural network, we used a sigmoid activation function, which allows us to predict which classification (1 or 0) is more likely for the given game vector. The loss (or error) is calculated based on these predictions on each timestep, and we aim to minimize this loss when training our network. In this case, we used a binary cross-entropy loss function, given as

$$-\sum_i (y_i \log p_i + (1 - y_i) \log(1 - p_i))$$

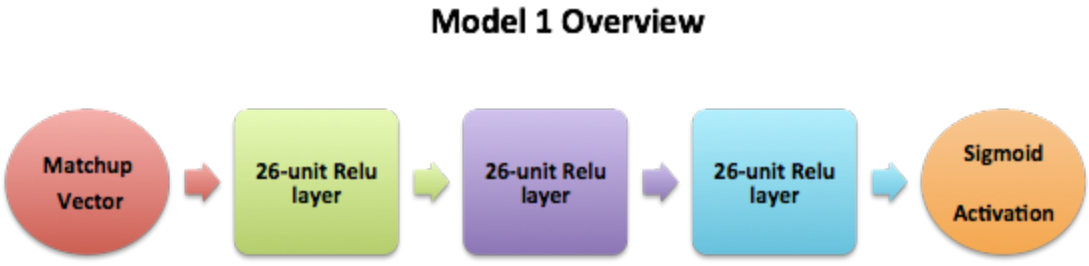


**Figure 2. Neural Network of Model 1**

**Note: "Relu" of* $x = max\{x, 0\}$

The second idea we had was building a regression network, which again takes matchup vectors as inputs and outputs the expected difference between team one's score and team two's. We believed that having a regression network would allow the model to learn to distinguish between games that are more lopsided and those that we called "toss ups", and thus

perform better at predicting the winner. To convert score predictions into binary classification, we simply predicted a 1 if the difference was greater than 0, and a 0 otherwise. In this network, we used a different loss function, mean absolute error, which is given as:

$$\frac{1}{n}\sum_{i}^{n}|y_i - p_i|$$

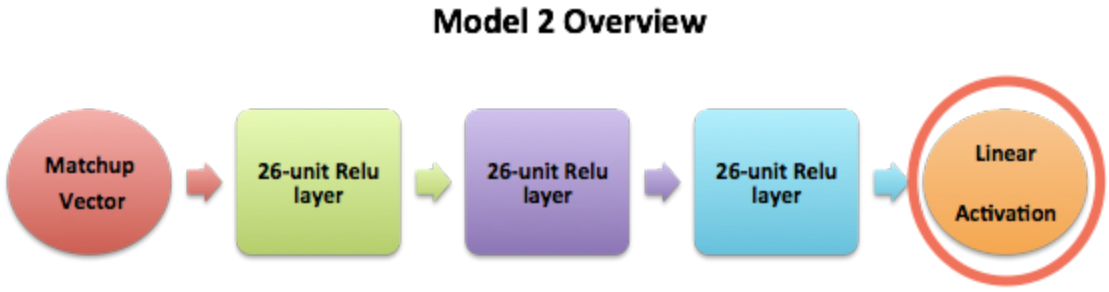


**Figure 3. Neural Network of Model 2**

Our third model was a bit more complex, and was born out of our attempts to tinker with the definition of our matchup vectors. Instead of taking matchup vectors as input, this model simply takes the two team vectors as inputs, and thus learns how to define a matchup vector based on the training data. The output of this model, as in model #2, is the predicted difference between scores, making this a regression model. We used the same loss function, mean absolute error, since this is also a regression model. We used the same criteria as in model #2 in order to predict winners based on the predicted scoring margin.

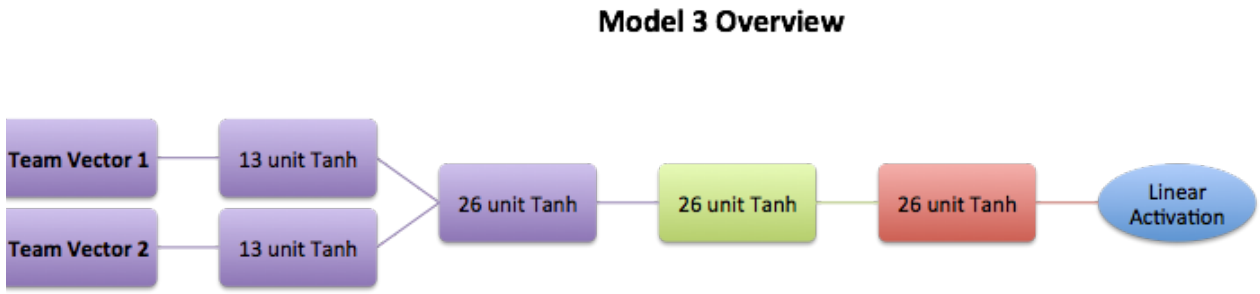


**Figure 3. Neural Network of Model 3**

**Note: "Tanh" is the hyperbolic tangent function.*

At their core, our neural networks aim to minimize their respective errors by tuning the weights at each layer. It finds this minimum using gradient descent, which adjusts the weights of each layer based on the error on each step. In order to calculate the gradient, we used a technique called back-propagation, which does a backwards pass through the network and calculates what percentage of the error to attribute to each node. Then, the nodes are adjusted using a modified version of gradient descent, called Adam, which has an adaptive learning rate.

One common problem with neural networks is the danger of overfitting. Since neural networks can model any function, the models are susceptible to overfit to the training data, leading to high performance when training but low predictive power. In order to combat this, we added dropout layers to the neural network. On every step, each neuron could be excluded with a given probability, and the network must adapt to make sure that its predictions are still accurate even without this neuron. In doing this, the network is able to reduce dependence on each individual neuron and improve the generalization of the model. Another important factor we added was a regularization term to the loss function. As shown in class, this helps prevent the model from overfitting complex data by limiting the magnitude of its weights. When implemented correctly, both of these techniques help reduce the risk of overfitting and improve the predictive power of the model.

# Results

We used a cross validation approach as a proxy for the performance of our models, training on randomized partitions of 9/10 of the data and testing on the other 1/10. Using this methodology, we were surprised to discover that all three of our models performed similarly, resulting in testing accuracies of approximately 70%, but never exceeding this percentage.

At first, we believed that we could improve the accuracy by implementing our second model, which took into account margin of victory. The idea was that the model would be able to better distinguish between games that were more lopsided and those that were “toss ups”, leading to better accuracy overall. Unfortunately, after running the cross validation on this new model we discovered that it did not notably improve performance over our basic classification model.

After this, we attempted to fix our limited definition of the “matchup vector”, which was a simple subtraction between two team vectors. This led to the implementation of our third model, which simply takes two team vectors as inputs, and thus the definition of a matchup is determined by the model. We paired this idea with our previous idea of regression, rather than classification, with the hopes of improving predictive performance. However, after implementing this approach we were surprised to discover no notable improvements in performance compared to the first two models.

We believe the stable outcome of performance was due to several reasons. The first reason is the human element of basketball. It is what makes March Madness unique and fun to watch and keeps it unpredictable. It often happens that a team with clearly better statistics simply has a bad day, while the underdog plays the game of its life, creating an upset. This unpredictability makes it difficult for the models to create an accurate decision boundary, since there are so many outlier games whose results surprise everyone. Another factor is that not all statistics are created the same. It is possible for a team to amass huge stats against bad teams while being merely average against stronger competition; in this case, our model weighs both stats equally, and thus may have prediction issues based on this.

# Conclusion

In summary, we have implemented three neural networks to predict March Madness game results with high accuracy. Our method of approach began by compiling ten years of statistical data on game measurements into a workable format and standardizing the team and matchup vectors to be input into each neural network. We built three different models using both regression and classification to find the best method for prediction accuracy; interestingly, all three models yielded an accuracy rate of nearly 70%. The accuracy is high enough to prove our neural network approach was effective, while the remaining 30% of incorrect outcomes shows there is an inherent randomness due to various human elements of the sport that cannot be modeled, leaving fans of March Madness with a bit of unpredictable excitement.

# Possible Improvements

Due to time and limited experience with machine learning techniques, there were a couple things we did not consider in our model that we think would help.

The first possible improvement is with the idea of risk aversion. The way that ESPN's scoring function is set up, games in later rounds have a value that is significantly higher than in earlier rounds. Simply, this means choosing games correctly in the later rounds gives you a better chance of winning. A person can get every single game correct in the first two rounds, but if they do not have a single team in the Final Four, they will not have enough points to win. Therefore, we propose that a risk factor be added to the model. This risk factor, measured as risk aversion, would increase round to round so that the first round, the model is the least risk averse and the final round is the most risk averse. We would try different values of risk at each round to see what works best. The reason we did not implement this idea is because of the complexity of creating a numerical value for risk. We could not come up with a logical way of

creating this value so we thought it would be best to avoid. Also, our model is meant to output predictions for all the possible matchups in the tournament, without concern for what rounds each game is played in. Building in this risk aversion would have meant reworking the entire model to only predict the outcomes of scheduled games.

The second possible improvement is the idea of making the model probabilistic instead of deterministic. This means we would create win probabilities for each match-up so that when we run our model there is some chance of different results, which better models the variability of real-life march madness. This will help account for upsets in the earlier rounds, but will still have the better team winning most of the time. In practice, this functionality could be used in a bracket group allowing multiple entries in order to maximize the chances of winning. The reason we did not implement this in our model is because the problem was asked in a deterministic way and there is not enough data for us to make a valid win probability. Based on win probability calculators used by companies like FiveThirtyEight, there are many factors, like additional stats and travel time, considered that we do not have access to. For example, we could research and find RPI and SOS for the years that were not provided in the csv file given to us by the 053 staff. With this lack of data, our probabilities would be inaccurate and not very useful to the model. With more time to collect data and study how to create these probabilities, this attribute would be extremely useful to the model to account for randomness.