Can Deep Learning Be Used to Detect Bias in NBA Referees?

Sebastian Algharaballi-Yanow

Cognitive Science Department, University of California, San Diego, 9500 Gilman Dr, La Jolla, 92093, California, USA.

The source code for this project can be found on Github: https://github.com/sebastianalgharaballi/NBARefereeBias.

1 Abstract

This project focuses on using various deep learning techniques to detect potential bias in NBA referees. After importing and cleaning 47 separate data sets that housed valuable statistics regarding currently active referees and the 30 NBA teams they have officiated over the years, I developed a total of 4 deep learning models that have the ability to predict referees that are potentially biased towards or against the teams they officiated. Of those models, which included three feed-forward neural networks and one 1D convolutional neural network (CNN), the CNN generated remarkable test metrics (accuracy, precision, recall, and F1-score) compared to the rest of the models. My experiment proved that CNNs have the capabilities to process tabular data and be deployed to successfully solve binary prediction tasks.

2 Introduction

The National Basketball Association (NBA) is one of the most prominent professional sports leagues in the world. Of course, being in a constant national spotlight comes with many benefits, but also many disadvantages. Of those drawbacks, the NBA has notably developed an officiating problem that, at times, has taken away the joy from both fans and players who love the natural outcomes that the game itself produces. The NBA houses one of the most infamous referee scandals of all time. In a span of games from 2005 to 2007, former referee Tim Donaghy placed tens of thousands of dollars in bets on games he officiated and had been approached by mob associates to work on the gambling scheme with him [1]. He was eventually sentenced to 15 months in prison and banned from officiating for life. Most recently, teams have made several public complaints regarding officiating, with multiple parties claiming that referees are working against certain teams and players to influence the outcome of games in an effort to affirm their personal biases and favoritism [2].

The current NBA referee assignment process is very vague to the public, with one source stating that assignments of refereeing crews for regular season games are made to simply balance the experience of referees across games, a “completely arbitrary” process that includes no thought of the advanced statistics regarding that particular crew/teams playing [3]. Other sources mention that playoff referees, not regular season, are assigned based on their performance in terms of “missed calls” and mid-season input from teams [4], a process that seems to, yet again, not include advanced statistics regarding each referee and each team that qualified for the playoffs that year.

The NBA needs to determine a better way in handling referee assignments, as it is clear that their current methodology is very problematic and controversial, leading to potential continual bias inflicted on the individual players and teams. As deep learning continues to make revolutionary advances, I believe that a practical solution for the NBA to implement is none other than a neural network-driven system that can detect potential referee bias through the NBA’s publicly available advanced statistics. For many years, neural networks have been used to solve complex prediction tasks like an advanced- metric-based bias detector. With the implementation of feed-forward neural networks and even a 1D convolutional neural network, this experiment can help the NBA solve this substantial issue.

Feed-forward neural networks are considered the ”default” deep learning algorithm for binary prediction tasks [5]. The reason for this is rooted in their ability to learn non-linear relationships between input features and target labels, which is a crucial requirement for effective classification. A feed-forward neural network consists of multiple layers of interconnected nodes, with each node performing a simple mathematical operation on its inputs and passing the result to the next layer. The first layer of the network takes the input features, and the last layer produces the final output. In binary prediction tasks, the output layer typically contains a single node that produces a binary output, indicating the predicted class label. One of the key strengths of feed-forward neural networks is their ability to automatically learn features from raw input data through backpropagation. The network iteratively adjusts its weights to minimize the difference between its predicted outputs and the true labels. This process is how feed-forward neural networks can capture complex, non-linear relationships between input features and target labels, which is particularly important for my binary prediction tasks. The decision boundary between the two classes indicating potential bias is not a simple linear function of the input features.

While feed-forward networks are strong for binary prediction tasks, the modern deep learning era continues to focus on more advanced networks with even stronger abilities and more applications. One of these revolutionary networks are Convolutional Neural Networks. CNNs are particularly well-suited for tasks that involve images or other types of data that have a grid-like structure, and have shown very high accuracy for projects in image classification. Using CNNs for tabular data like mine is still a relatively new area of research, but it has shown promising results. One of the main challenges with using CNNs for tabular data is that tabular data is not naturally grid-like, which is what CNNs are designed to work with. To overcome this challenge, researchers have proposed various methods to reshape the tabular data into a grid-like structure that can be fed into a CNN [6]. One approach is to reshape the tabular data into an image-like format, where each row of the table is treated as a pixel row in an image, and each column is treated as a channel. The resulting ”image” can then be fed into a CNN, which applies convolutional and pooling operations to learn important features and patterns in the data. I will attempt to implement this recently discovered approach in my study.

3 Methods

3.1 Data Collection

Before going into the neural network methodology I used to carry out this experiment, we must speak about the data gathering, cleaning, and implementation process. In order to build the most successful bias detector system, high quality data was needed to ensure our solution can be used in the real-world. Luckily, NBA referee data is publicly available information. Basketball Reference, a company that specializes in gathering all data basketball-related, collects separate data sets for every referee that has ever officiated an NBA game [7]. Per referee, Basket Reference has statistics for every game that referee has officiated per 30 NBA teams. This dataset provides information such as the referees’ total games officiated for each team, win/loss percentages, average points, free throw attempts, personal, offensive, and shooting fouls, and statistics for the opponents that each team faced. By comparing the differential in these statistics between the respective team and its opponent, I was able to gain insight into potential biases that referees may have towards certain teams. However, I needed to develop a clear ”Bias” metric that the neural network could use as a target variable in the prediction task. This will be further discussed in the next section.

3.2 Data Processing

I used the Python programming language for all of my necessary data processing steps. I chose to incorporate data from 47 current NBA referees for my project. I only selected referees that are officiating games to this day and have at least 10 years of experience. This criteria ensured that I would have a large mass of data to train my networks on. Since there are 30 NBA teams and 47 referees that have officiated many games for all teams, my dataset was going to have 1410 rows. Therefore, preprocessing this large dataset was crucial in order to ensure that my later results with my neural networks would be accurate.

In the first data processing step, the necessary data files were read and processed using Pandas library. First, a CSV file named ”Ref Names.csv” was read and stored in a Pandas DataFrame. This file contained the 47 referees, the date of their first ever game officiated, and most recent game date. Null values were removed using the ”dropna()” method, and the column names were changed using the ”rename()”

method for better readability. This initial step was important to conduct in order to ensure that the referees in my study have at least 10 years of experience.

Next, I needed to process the individual referee data. As mentioned before, Basketball Reference manages the game-related data for each active NBA referee. However, this data is separated from the Referee Names file, meaning that each official has their own .csv file that corresponds to the statistics regarding each of the 30 NBA teams. Therefore, the first thing I needed to do was read in every one of these data frames. While this took 47 lines of code, the steps to initially reading in the .csv files were very repetitive. Then, in order to change the variable names to a tidy format in a quick and efficient manner, I looped over all of the data sets and changed the variable names. This process was done by creating a dictionary that housed the .csv files I had just read in, generating the for loop for my desired column names, and dropping all missing values to mitigate later processing problems with my neural networks. Then, I saved my results in 47 new, cleaned data frames, and was ready to concatenate the referee data into one, large table.

I started the concatenation process by creating a list of the 47 cleaned data frames, in addition to a list of the referee names in alphabetical order. Then, I concatenated all cleaned data frames into one and reset the table index to ensure clarity and tidiness once more. Lastly, I wanted to format the referee names in an orderly manner. Since my table contained data corresponding to the 30 NBA teams per referee, I knew that it would be wise to group each set of 30 teams by referee in the data frame. In other words, I wanted the first 30 rows to be paired with referee Aaron Smith, the second 30 rows to be paired with Ben Taylor, and so on, until the end of my 47 name list. Therefore, I looped through each group of 30 rows and assigned the respective referee's name to their data that corresponded with each of the 30 teams. I then defined a function to format the referee's name as ”First Name, Last Name”, then applied that function to each row in the data frame. I concluded my data processing by moving the ”Referee Name” column to the first position in the data frame, and was finally ready to begin developing my bias metric. At this point, my data frame was 1410 rows x 21 columns, with each column containing crucial statistics I would use to create my bias metric.

In order for my neural networks to have a clearly-defined target variable and additional, concrete statistic that defines a biased referee, I needed to create a numerical equation that took in the game data that is often viewed as a clear sign of referee bias if major discrepancies in the numbers are found. Those statistics are win/loss percentage for the corresponding team, and average point differential, free throw differential, personal foul differential, shooting foul differential, and offensive foul differential, all between the respective team and their opponents when being officiated by that referee. If any of the above metrics are skewed one way or the other, it may be a sign that the referee is favoring towards or against the team of subject. With that in mind, I created a weighted formula that clearly defines a referee's ”Bias Score” towards each of the 30 teams they have officiated:

$$\begin{aligned}\text{bias\_score} = {} & (0.2 \times \text{win\_loss}) + (0.2 \times \text{point\_diff}) + (0.15 \times \text{ft\_diff}) \\ & + (0.15 \times \text{pf\_diff}) + (0.15 \times \text{sf\_diff}) + (0.15 \times \text{of\_diff})\end{aligned}$$

The formula assigns different weights to each statistical factor based on their relative importance in determining bias. For instance, win/loss records are given a weight of 0.2, while point differentials are also given a weight of 0.2. Other factors, such as shooting foul differentials, are given a lower weight of 0.15. By plugging in the relevant data for each factor, the equation produces a numerical bias score that can be used to assess the degree of potential bias exhibited by a given NBA referee. In an effort to binarize the scores into a definition of "Bias" and "No Bias", I utilized MatPlotLib to plot my bias scores against each of the 6 statistical categories. I was able to precisely view the spread of my data and the portion of outliers that immediately stood out to me. After careful analysis, I defined my "golden zone", the bias score that indicates an NBA referee is clear from one-sided officiating, as -0.7 to 0.7:

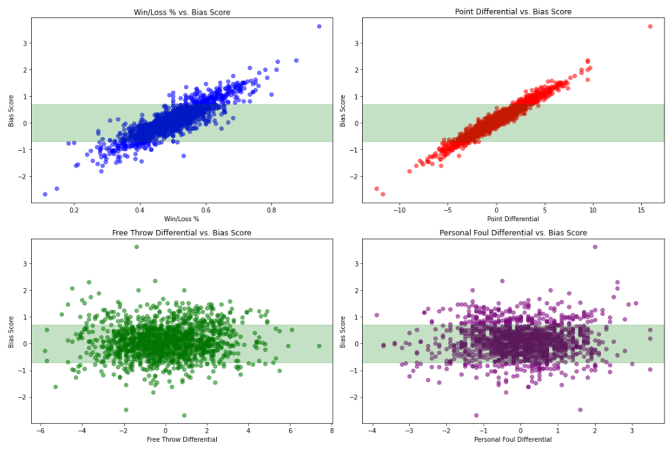

Fig. 1 First four bias score plots. Green indicates "golden zone".

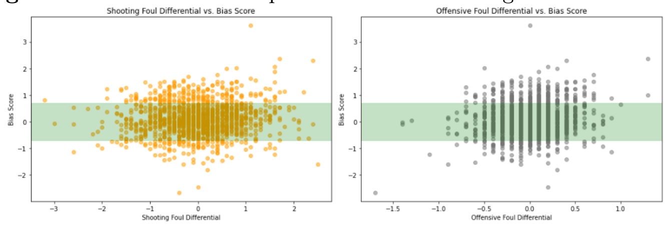

Fig. 2 Last two bias score plots. Green indicates "golden zone".

As shown above, I aligned my "No Bias" category in an effort to separate the noticeable outliers of the data from the typical differentials between teams that were plotted on each graph. If a referee obtained a bias score that fell between -0.7 and 0.7, they would be categorized as "No Bias". Otherwise, I would categorize them as "Bias". This is how the binary target variable for my neural networks were fully developed. At the end of this final processing step, I added these two new columns to my data frame under the names "Bias Score" and "Bias". Therefore, my final data frame was

1410 rows x 23 columns, an excellent amount of data to successfully train my neural networks.

All of the coding steps I outlined above are located in the Jupyter Notebook that is presently in my GitHub repository (linked on first page).

3.3 Neural Network Implementation (with PyTorch)

In order to develop the best neural network system for my task, I decided to implement three feed-forward neural networks and one convolutional neural network. Feed-forward networks can be fine-tuned in so many different ways to achieve peak performance, which is why I wanted to explore three iterations of that network. The general structure I outlined below was not predefined in any manner. Meaning, as one model finished training and evaluation, I analyzed ways I could further improve the model and implemented those differences until I had three, trained models. At the end of my training and tuning process, here is what the network structure of my three models consisted of:

1. The first FNN consisted of 3 fully connected layers with 128, 64, and 1 neurons, respectively. The ReLU activation function was used in all hidden layers, and the sigmoid function was used in the output layer to produce binary classification results. The input size was determined by the number of input features, which excluded the bias feature. The neural network was trained for 500 epochs with a batch size of 24 using the binary cross-entropy loss function and Adam optimizer with a learning rate of 0.0015 and L2 regularization of 0.001.

2. The second FNN consisted of 5 fully connected layers with 256, 128, 64, 32, and 1 neurons, respectively. The ReLU activation function was used in all hidden layers, and the sigmoid function was used in the output layer. The neural network was trained for 500 epochs with a batch size of 64 using the binary cross-entropy loss function, Adam optimizer with a learning rate of 0.0015, and L2 and L1 regularization of 0.0001.

3. The third FNN consisted of 5 fully connected layers with 1024, 512, 256, 128, and 1 neurons, respectively. The ReLU activation function was used in all hidden layers, and the sigmoid function was used in the output layer. The neural network was trained for 500 epochs with a batch size of 64 using the binary cross-entropy loss function, Adam optimizer with a learning rate of 0.001, and L2 regularization of 0.001.

The differences between my three FNNs are primarily in the number of layers and the number of neurons in each layer, as well as the regularization techniques and hyperparameters used during training. The first FNN has three fully connected layers (with 128, 64, and 1 neurons), compared to the other two that have five fully connected layers and many more neurons, with the most neurons being in the third model. The batch size was also increased from 24 to 64 in the two models after my initial run. I experimented with both L2 and L1 regularization in my second trial in an effort to see if different regularization had an effect on test performance. I also experimented with a larger learning rate in my final iteration.

My convolutional neural network was much more difficult to implement due to the data manipulation process, regarding mapping tabular data to image-like data, I outlined in the introduction. Therefore, I chose to build only one CNN for my prediction task to see how it compared to my best fully-connected network. My CNN architecture consisted of 3 convolutional layers followed by 2 fully connected layers. Each convolutional layer had 32 filters, a kernel size of 22, and a padding of 1. The first layer took in the input with 1 channel (1 feature), and the subsequent layers took in 32 channels. Each convolutional layer was followed by a max pooling layer with a kernel size of 2 and a stride of 2, which halved the length of the feature maps output by the convolutional layers. The output of the final max pooling layer was flattened and passed through two fully connected layers. The first fully connected layer had 32 neurons and used the ReLU activation function. The second fully connected layer had a single neuron and used the sigmoid activation function, outputting a value between 0 and 1 that represented the predicted probability of the input being in class 1 (Biased). During training, the binary cross entropy loss function was used, and the Adam optimizer was used for optimization.

In order to carry out the experiment with all of my networks, I first needed to label encode the categorical variables in my dataset, and further binarize my "Bias" column by representing an instance of bias as "1" and no bias as "0". This would allow my networks to train smoothly and maintain the present size of the dataset. This is why I chose label encoding over one-hot encoding, since one-hot encoding would create an abundance of new columns for every instance that corresponded to a given categorical variable, making it very hard for the network to find patterns in data and ultimately train. Then, I was able to split the dataset into training and testing sets, which is where I selected my target variable as the "Bias" column and generated a testing set that was 20% of my original data. After converting my X and y training and testing variables into PyTorch tensors, I was ready to train the model and begin experimentation for that instance.

4 Experiment

In order to evaluate each of my models on the respective test set, I used accuracy, precision, recall, and F1-score as my predefined metrics, as well as the overall testing loss. Since my dataset had a class imbalance, with 901 negative (no bias) instances and 509 positive (bias) instances, using accuracy alone could be misleading. For instance, if the model always predicts "no bias", it would still achieve an accuracy of about 64%, which is not a good performance. Therefore, I used precision, recall, and F1-score as well to have a more comprehensive understanding of the model's performance. Precision measures the fraction of true positives among the total predicted positives. It answers the question, "how many of the predicted positives are actually true positives?" A high precision indicates that the model rarely predicts a positive instance when it's actually negative, which is desirable for my use case. Recall measures the fraction of true positives among the total actual positives. It answers the question, "how many of the actual positives were correctly identified?" A high recall indicates that the model can correctly identify most of the positive instances, which is also desirable for my use case. F1-score is the harmonic mean of precision and recall, and it provides a balance between the two metrics. A high F1-score indicates that the model has both high precision and high recall.

My first, most simple feed-forward network performed the second best out of my three FNN models. The model achieved a 55.91% accuracy, 33.33% precision, 32.27% recall, and 32.79% F1-score. The final test loss was recorded as 0.1094, and the training loss showed a very jagged convergence to the minimum. This indicates unstable performance and shows that some tuning was needed in order to improve the model as a whole. Correctly classifying about 1/3 of my test instances was definitely not the performance I was looking for, which is why I dramatically tuned this first model in favor of the second feed-forward network I had described earlier.

As mentioned before, I increased the number of layers and the number of neurons in each layer, as well as the regularization techniques (using both L1 and L2) and batch size used during training for my second fully-connected model. This model improved slightly in performance compared to my first trial, but was still not generated the optimal performance I was looking for. It achieved a 55.32% accuracy, 33.33% precision, 34.04% recall, and 33.68% F1-score, and a lower test loss of 0.0985. However, the training loss still showcased a far from smooth convergence to the minimum, yet another indication that this second FNN was still not performing very well.

Since I saw small increases to overall performance after developing a much more complex network and hyperparameter structure, I decided to make an even more compounded network for my third FNN iteration, mainly by increasing the amount of neurons while reverting back to using only L2 regularization. I also slightly increased the learning rate to see if it could improve model convergence. What turned out to be my final implementation of a feed-forward network in my study achieved a 56.03% accuracy, 33.33% precision, 31.91% recall, and 32.61% F1-score. The test loss for this iteration was recorded as 0.1466, and my training loss, while converging in a smoother manner than the other two models, was still jagged enough to confidently say that the model was still not performing well at all.

After three iterations that involved careful tuning and improvements to the overall structure and methods used in each model, my three FNN models all performed around the same, poor levels. I was slightly surprised by this, especially after my prior research and coming to a clear understanding that feed-forward neural networks are made to succeed in binary prediction tasks like mine. Despite the underwhelming performance, I did not give up hope and decided to begin developing the convolutional neural network I outlined earlier.

It was initially very difficult to develop the convolutional network due to the feature engineering and computational knowledge needed to map my tabular dataset to an image-like state. However, I was able to successfully achieve this feat and deploy the model onto my dataset. The results were eye-opening. The CNN achieved a test accuracy of 91.84%, precision of 87%, recall of 88%, and F1-score of 88%. For good measure, I also recorded the evaluation metrics for when "Not Bias" (0) was considered the positive class, and the results were even better, with a 94% precision, recall, and F1-score across the board. To make things even better, the test loss converged on the smoothest path yet, and finished with a value of 0.1922.

After running and evaluating each of my four models, it is without a doubt that the convolutional neural network completely defeated the other models by a wide margin. Shockingly enough, I only needed to train the CNN on 20 epochs for it to achieve this optimal performance, compared to the 500 epochs the feed-forward networks required. Mind you, my CNN model was deployed without any hyperparameter or network tuning, so evaluation metrics close to 90% on the original version of the model is remarkable, in an understatement. I hypothesize that the superior performance of the CNN is due to its ability to effectively extract and capture spatial features from the input data, which is crucial in tabular data with complex dependencies and interactions among the variables. In contrast, the FNNs may have struggled to learn such dependencies due to their linear network structure and limited capacity to model interactions among multiple variables. Regardless the reason, my experiment proved that convolutional neural networks can successfully be used to process and predict complex, binary tasks like NBA referee bias.

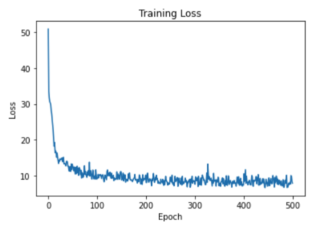

Fig. 3 FNN Training Loss (1)

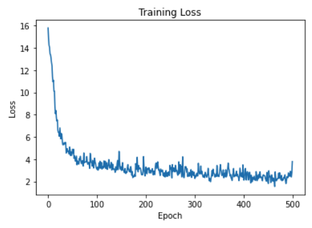

Fig. 4 FNN Training Loss (2)

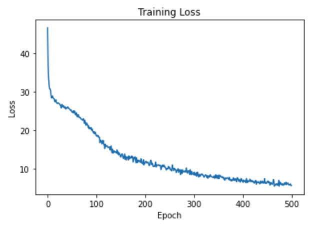

Fig. 5 FNN Training Loss (3)

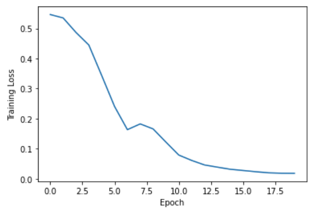

Fig. 6 CNN Training Loss

5 Conclusion

To conclude, this research project was developed in an effort to produce a practical deep learning solution the NBA can implement in order to solve their well-known referee bias problem. The sheer importance of proper data collection and cleaning, network development and architecture, careful tuning, and open-mindedness in regards to implementing non-traditional methods to solve pre-defined problems were on full display throughout the study. At the end of the project, it was determined that convolutional neural networks, an entity of deep learning that is normally used for image recognition tasks, could successfully be used in binary, tabular, bias-detection tasks and perform at levels that three optimized FNNs could not achieve. While the results were eye-opening and showcases the true power CNNs hold in the deep learning world, more extensions are always in the realm of possibilities with projects like these. I plan on further studying this topic in the near future and gather even more referee data that could be used to train the model. I also plan on optimizing my convolutional network in an effort to achieve even greater performance. I could even extend the capabilities of the network and develop a system that predicts the 3 best, bias-free referees to officiate a game between two input teams. The possibilities are endless, which, once again, shows just how powerful convolutional neural networks are.

I am proud to say I discovered a new development for the NBA, an organization that has yet to use deep learning as a solution to their referee issues. I hope that, one day, this project acts as a stepping stone to a full bias-detector algorithm that I could design for the NBA. For now, I will continue researching how I could make my convolutional neural network even better than what it is, collect more data, and fine-tune my model to increase its accuracy. In addition, I plan to share my findings and results with other researchers and experts in the field, to receive feedback and collaborate on how to further improve this technology. My ultimate goal is to help create a fairer and more transparent system within the NBA, which can only be achieved by identifying and eliminating any biases that may exist in the officiating of games. By using the power of deep learning and data analysis, I am confident that this is a goal that can be accomplished.

References

[1] Eden, S.: How former ref Tim Donaghy conspired to fix NBA games. ESPN. https://www.espn.com/nba/story/_/id/25980368/how-former-ref-tim-donaghy-conspired-fix-nba-games (2019)

[2] Golliver, B.: As NBA playoffs heat up, officating complaints overshadhow the action. Washington Post. https://www.washingtonpost.com/sports/2022/05/09/nba-playoffs-referees-fouls-complaints/ (2022)

[3] Price, J., Wolfers, J.: Racial Discrimination Among NBA Referees. TIME. https://time.com/wp-content/uploads/2014/12/price_wolfers_paper.pdf (2014)

[4] Zillgitt, J.: NBA's first-round playoff referees include mix of experience, first-timers. USA Today. https://www.usatoday.com/story/sports/nba/2019/04/12/how-nba-selects-its-playoff-refs/3435412002/ (2019)

[5] Thawakar, O., Gajjewar, P.: Training optimization of feedforward neural network for binary classification. In: 2019 International Conference on Computer Communication and Informatics (ICCCI), pp. 1–5 (2019). https://doi.org/10.1109/ICCCI.2019.8822184

[6] Lin, X., Zhao, C., Pan, W.: Towards Accurate Binary Convolutional Neural Network. 31st Conference on Neural Information Processing Systems (NIPS 2017). https://proceedings.neurips.cc/paper_files/paper/2017/file/b1a59b315fc9a3002ce38bbe070ec3f5-Paper.pdf (2017)

[7] Basketball Reference. NBA Referee Datasets. https://www.basketball-reference.com/referees/?__hstc=213859787.87fd76a5ac0fe8bf4bb6e0f05e58343b.1676504540392.1676504540392.1676504540392.1&__hssc=213859787.1.1676504540392&__hsfp=3600436941 (accessed on 12th March 2023)