NBA Sports Betting Model Using Deep Learning

Sunan Tajwar[*], **Jason Huang**[*]

st4147@nyu.edu, yh3595@nyu.edu

Abstract

This project explores the application of deep learning techniques, including RNN, LSTM and Transformer network paradigm in the domain of NBA sports betting. Our aim is to develop a predictive model that leverages historical game data, player performance statistics, and other relevant variables to forecast game outcomes with an accuracy high enough to gain profit. By integrating different deep learning model architectures, our project seeks to demonstrate how deep learning can provide insights into sports analytics and betting strategies.

Introduction

The intersection of sports and data analytics has been gaining interest over the recent years. Similar to the stock market, lots of researches has been exploring the application of machine learning and deep learning in sports betting. In this research, we mainly focus on sports betting in the NBA field, which involves numerous variables and factors, making accurate predictions possible but also challenging. Traditional betting models has been using Machine Learning techniques including Native Bayes, Random Forest and etc. , which may not capture the complexities of the NBA games.

Recent developments in deep learning have opened new avenues for predictive analysis in sports. Deep learning utilizes neural networks with multiple layers to learn from large datasets. These models can identify intricate patterns and relationships in data, which are often imperceptible to human analysis and decisions.

In the context of NBA sports betting, deep learning has the advantage of analyzing vast amounts of data, including historical team statistics, player performance, win or loss trends, or even social media sentiment if accessible. All these features contributes to a more informed predictions from deep learning models on game outcomes. The ability to process and learn from diverse data sources gives deep learning models a significant edge over traditional data.

With that said, this project aims to leverage the capabilities of deep learning models to enhance the accuracy of NBA sports betting predictions, by employing different model architectures on a binary classification task that essentially predicts if the home team will win in NBA games. Our work can serve as a tool helpful for sports analysts and sports enthusiasts by offering a more data-driven approach to understanding and predicting NBA games.

[*]These authors contributed equally.

Literature Review

Previous Work on NBA Sports Betting

Early Statistical Models Initial approaches in sports betting were grounded in basic statistical methods, such as regression analysis to predict game outcomes (Askew 2023). These models often consider factors like team and player performance statistics, historical match outcomes, and simple ranking systems. While providing a fundamental understanding, these models lacked the depth and complexity needed to accurately capture the dynamic nature of NBA games.

Machine Learning and Deep Learning Models With the rise of machine learning, focuses has been shifted towards more advanced predictions using machine learning techniques. These models includes Native Bayes, decision trees, random forests, and support vector machines (Jordan 2021) (Garfjohnson 2023). Among these models, neural networks, especially deep learning models, have gained prominence due to their capability of processing large datasets with multi-dimensional complex, non-linear features (Loeffelholz, Bednar, and Bauer 2009; J. and Rastegari 2013; Ivanković et al. 2010) . Deep learning models in NBA sports betting analyze not only traditional statistics but also unstructured data, as well as in-game variables. Therefore, as our attempt to make informed predictions on NBA data, we feature a [insert data analysis steps] and compared our results against three different deep learning models: RNN, LSTM and Transformer based binary classifiers.

Recurrent Neural Network (RNN) RNNs (Sherstinsky 2018) are a class of neural networks particularly adept at processing sequences of data, making them ideal for tasks like time series analysis, natural language processing, and, in our case, sports analytics. Unlike traditional neural networks, RNNs have a unique feature: they possess internal memory. This memory allows them to remember and utilize the information from previous inputs in the network, providing a way to capture temporal dependencies in data. In the context of classifiers, RNNs analyze input data sequentially, updating their hidden state at each step based on both the current input and the previously received inputs. This sequential approach enables RNNs to model time-dependent data effectively. However, RNNs can struggle with long-range dependencies due to issues like vanishing or exploding gradients.

Long Short-Term Memory (LSTM) LSTM (Sherstinsky 2018) networks are an advanced type of RNN designed to overcome the limitations of traditional RNNs, particularly in handling long-range dependencies. LSTMs maintain a more complex internal state that can capture long-term dependencies in sequential data, making them highly effective for tasks requiring the understanding of such relationships. This capability is achieved through a unique architecture comprising different gates (input, forget, and output gates) that regulate the flow of information. These gates determine what information should be kept or discarded at each step, allowing the LSTM to preserve important information over extended sequences. In classifiers, LSTMs analyze input sequences while maintaining a memory of relevant past information, leading to improved performance on tasks where context over long sequences is crucial.

Transformer The Transformer architecture (Vaswani et al. 2023) represents a significant departure from the RNN and LSTM-based approaches. Transformers do not process data sequentially; instead, they use a mechanism called 'attention' to draw global dependencies between input and output, enabling the model to process entire sequences of data simultaneously. This parallel processing ability allows Transformers to achieve remarkable efficiency and effectiveness in handling large and complex datasets. The architecture consists of two main components: an encoder that processes the input data and a decoder that generates the output. In classification tasks, the Transformer can rapidly process and analyze the entire data sequence, capturing intricate patterns and relationships with a level of sophistication unattainable by traditional RNNs or LSTMs. Its ability to handle long-range dependencies and its scalability make it particularly well-suited for complex classification tasks in domains like language translation, text summarization, and sophisticated data analysis in sports betting models.

Data Collection & Pre-processing

NBA Game Data In the process of finding comprehensive and reliable NBA statistics spanning multiple seasons, we used web scraping techniques on *basketball-reference*, which contains large data stores on in-depth game-to-game analysis spanning over a decade. These web scraping techniques followed the instructions and guidance from VikParuchuri (Dataquest 2023) and his repo *nba_games*. We employed a Python-based approach, leveraging the synergistic capabilities of BeautifulSoup and the PlayWright API to extract data from the Basketball-Reference website systematically. The primary objective was to collect a diverse array of statistics, including team performance metrics, team standings, game results, home and away differences, and other pertinent information, crucial for in-depth analysis and insights into the NBA's in-game dynamics between the 2014-2022 seasons.

BeautifulSoup, a Python library, facilitated the parsing of the intricate HTML structure of Basketball-Reference. Its functionality allowed for precise identification and extraction of relevant data elements, navigating through the website's markup to isolate and retrieve essential statistics. Moreover, the integration of the PlayWright API elevated the scraping process by enabling browser automation. This API empowered the script to simulate user actions, interact with dynamically rendered content, and navigate through multiple pages seamlessly. PlayWright's capabilities in handling JavaScript-intensive elements significantly enhanced the data retrieval process, ensuring a more comprehensive collection of NBA statistics for analysis.

This approach easily facilitated the aggregation of statistical information and laid the groundwork for deeper analytical insights into team trends and strategies, and overarching patterns that shaped the outcome of NBA games and landscape over the course of a decade. The resulting dataset became a valuable resource for researchers and analysts seeking to explore and understand the nuances of the sport across varied seasons, fostering a deeper understanding of the evolving dynamics within the NBA, and how it affected the outcome of individual games.

As we moved onto the pre-processing of the decades worth of NBA game data, we began by deleting irrelevant information that stays consistent across all games such as minutes played, and indices that basketball-reference used to identify teams internally. We then sorted the data by the game dates and then proceeded to create our target variable. Our target variable was essentially an indicator of whether or not the team in question will win their next game. In doing so, we are essentially setting up a binary classification problem. We then removed all columns that contained multiple null values because it would be impossible for us to accurately source or impute the data given the size and volatility of the dataset. However, even after this, we were able to retain the majority of the in-depth statistical features and game/team-identifying features.

We then applied a *Min-Max Scaler* on all of the statistical features, to normalize the feature values while still preserving the original shape and distribution of the data. Our next goal was to model the concept of momentum and team form over the course of a season into our dataset, and later analyze how a team's recent winning/losing streak affects their chances to win their next game. We did so by implementing a "lookback" window for a period of 10 games for all the statistical features for each team per game. We added the mean of each of these statistical categories over that rolling window of 10 games for each game as an additional feature in our dataset.

Lastly, we wanted to manipulate our dataset to store the data for how the teams matchup per game. We merged game data for both teams involved in each game using the game data and opponent keys, in order to create a singular matchup dataset. The matchup data frame was the final iteration of our NBA game dataset. We then used the target variable that we had created as our target for our RNN, LSTM, and Transformer variables. However, during training, we only used the statistical features from the matchup dataset to predict the outcome of the games via the target variable.

NBA Odds Data The second half of our data collection process included sourcing the NBA odds data. It was difficult to source betting data on recent NBA seasons due to the incompleteness of many of the datasets and the increased difficulty of web scraping it due to inconsistencies across different websites and betting platforms. We were fortunate enough to find a set of odds data that had been used before in a GitHub repo by *garhjohnson*. We decided to use a dataset from their repository on the odds data for each game from the 2019-2020 NBA season. In using this we decided that after the training and testing of our model, we would use this odds data to backtest the performance of our prediction and betting model on games in the 2019-2020 season between October and February. To due so, we also separated the data we simulated our bets on from the 2019-2020 season from the training dataset. When modeling the bets based on the outcomes predicted by our betting model, we first altered the odds data to follow the same nomenclature for writing the dates and team names as our games dataset. Then after using our trained model to predict the outcomes of the games we have betting data on, we merged the predicted data and odds data into a singular dataframe using the team and game date keys. We then used this dataset to calculate our profits and losses from simulated bets on our odds and outcome data.

Model Building

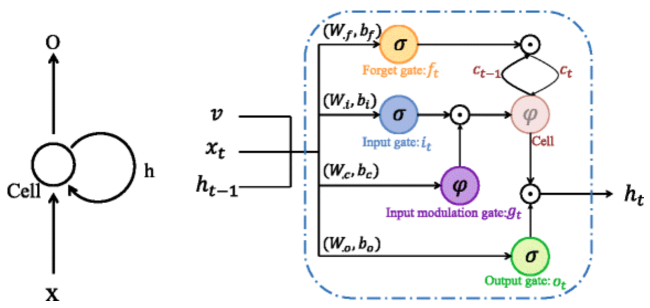

Figure 1: RNN Unit Architecture(Guo et al. 2018)

Base RNN The first model we trained was a Recurrent Neural Network (RNN), whose architecture is designed as a Sequential model using Keras, configured for sequence prediction tasks. The model begins with an Embedding layer, essential for converting input sequences into dense vectors, followed by multiple GRU (Gated Recurrent Unit) layers, each comprising 64 units and set to return sequences. The inclusion of multiple GRU layers, stacked one after another, contributes to capturing intricate temporal patterns within sequential data, potentially enabling the model to learn hierarchical representations of input sequences. To add complexity and enhance feature extraction, a TimeDistributed layer with a Dense sublayer of 32 neurons and ReLU activation is applied to each time step of the sequence. Finally, the model concludes with a GRU layer of 32 units followed by a Dense layer with a sigmoid activation function to output predictions. The model is compiled with the binary cross-entropy loss function and the Adam optimizer, while accuracy is monitored as the evaluation metric during training. This architecture is tailored for sequence-based classification tasks, leveraging the GRU layers' ability to capture sequential dependencies in the data.

RNN with Pruning The next model we trained was a pruned RNN model, which was constructed using Keras with TensorFlow's model optimization toolkit. The pruned RNN model architecture initially resembles the previous architecture but introduces a pruning technique to reduce model complexity and potentially enhance generalization. It begins with an Embedding layer, followed by multiple GRU layers, each containing 64 units and set to return sequences, aimed at capturing temporal dependencies within sequential data. However, distinct from the original design, this pruned model incorporates only one TimeDistributed layer with a Dense sublayer of 32 neurons and ReLU activation. Following this, a GRU layer with 32 units is applied before the final Dense layer with a sigmoid activation for classification. The key addition to this model lies in the implementation of model pruning using TensorFlow Model Optimization's pruning API. The pruning is applied to the model's weights through a sparsity configuration, utilizing a polynomial decay schedule from 50% to 90% sparsity over 1000 steps. By pruning model weights, this technique aims to compress the model size and potentially improve computational efficiency without compromising predictive performance. The model is compiled using binary cross-entropy loss, the Adam optimizer, and accuracy as the evaluation metric during training, offering a pruned alternative to the original architecture for sequence-based classification tasks.

LSTM

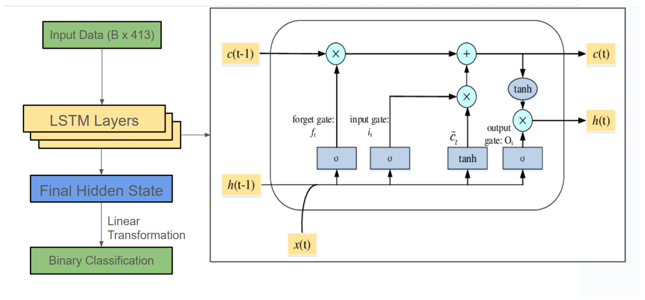

Figure 2: LSTM Architecture - LSTM unit from (Tomar et al. 2020)

The LSTM Classifier model, built using PyTorch, presents an architecture finely tuned for sequence processing tasks. It kicks off with an input dimension of 413, accommodating a wide range of features typical in complex datasets like those found in natural language processing or time-series analysis. Central to this model are the LSTM layers, with a hidden dimension set at 100 units, offering a balanced blend of model complexity and computational efficiency. This dimensionality plays a pivotal role in capturing intricate patterns within the sequential data.

Distinguishing itself with a layered approach, the model incorporates 2 LSTM layers, optimizing its capacity to learn and interpret sequential dependencies. Each layer operates in tandem, enhancing the model's ability to discern long-range patterns that simpler architectures might miss. Following the LSTM modules, the model employs a fully connected (Linear) layer, mapping the LSTM's intricate understanding of the sequence to a binary classification output, given its output dimension of 2.

We also initializes hidden (h0) and cell (c0) states to zeros, ensuring a clean slate for each new sequence processing. The model then adeptly navigates through the LSTM layers, culminating in a linear transformation to produce a precise binary output. This output is derived from the final time step's hidden state, a technique characteristic of many sequence-based classification tasks, aiming to capture the most recent and relevant temporal features.

The model's LSTM architecture aims to harness sequential information for predictive analytics, particularly in binary classification scenarios.

Transformer

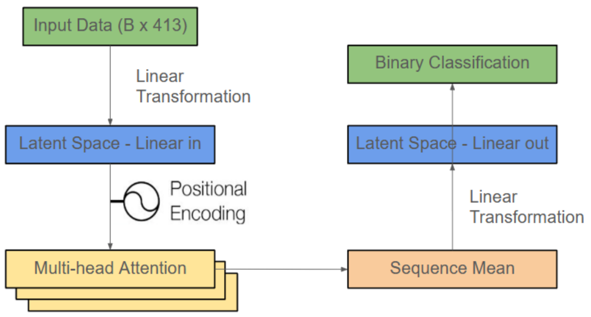

Figure 3: Transfomer Architecture

The Transformer Classifier is designed to employ a transformer-based architecture renowned for its efficiency and effectiveness in processing sequential data.

The initial step in the model's pipeline involves a linear transformation layer, mapping the input features to a higher-dimensional space, specifically a `dim_model` of 128. This transformation is crucial for preparing the data to be effectively processed by the transformer architecture.

Central to this model is the positional encoding component, a mechanism that injects information about the position of each element in the sequence. This feature is vital in a transformer model, as it compensates for the architecture's lack of inherent sequence awareness, unlike RNNs or LSTMs.

Following positional encoding, the data flows through the core of the model - the Transformer encoder. The encoder is constructed from layers, each featuring a multi-head attention mechanism with 8 heads and dropout set to 0.2, enhancing the model's ability to focus on different parts of the sequence and providing regularization to prevent overfitting. The model stacks 4 such layers, creating a deep network capable of capturing complex relationships in the data.

Post transformation, the model aggregates the output using a mean operation over the sequence dimension, ensuring that the subsequent classification is informed by the entire sequence. The final step in the model's journey is another linear layer, which maps the transformer's rich, high-dimensional representation down to the number of classes, in this case, 2, suitable for binary classification tasks.

Model Evaluation

Base RNN The base RNN model, trained over ten epochs, displayed a consistent training and validation accuracy around 50%, indicating struggles in surpassing random chance in predictions. The model's inability to discern meaningful patterns is evident from the low yet consistent accuracy. The validation accuracy hovering around 49-50% suggests limited generalization, possibly due to overfitting or insufficient complexity. The overall accuracy post-training was slightly better at 51.8%, but the precision, recall, and F1-score for the positive class were notably low, indicating challenges in correctly identifying and capturing positive instances.

RNN with Pruning The RNN model with pruning, over ten epochs, showed similar performance to the base model, with stagnant accuracy around 50%. This suggests that pruning did not significantly improve the model's ability to predict outcomes. The validation accuracy remained approximately 49-50%, and post-training metrics showed marginal improvement. However, the low precision, recall, and F1-score for the positive class persisted, indicating that pruning did not effectively address the model's limitations in differentiating between classes.

LSTM The LSTM model, trained over 10 epochs, showed initial performance in the first epoch with an accuracy of 49.58%, precision of 49.69%, recall of 53.65%, and an F1 score of 51.59%. The accuracy improved slightly in the second epoch, accompanied by significant improvements in recall and F1 score, but subsequent epochs revealed fluctuations in performance. The highest accuracy was 50.32% in the seventh epoch, with variability in precision and recall. The model's testing accuracy remained slightly above 50%, with consistent loss values suggesting challenges in learning, possibly due to dataset complexity or model limitations.

Transformer The transformer model's performance over 10 epochs showed a consistent but modest level of effectiveness. The first epoch began with an accuracy of 49.68%, and the highest recorded accuracy was 50.55% in the fourth epoch. There were small fluctuations in precision, recall, and F1 scores, but the overall performance did not show significant improvement, hovering around the 50% mark. The transformer model struggled to learn and distinguish patterns effectively, indicating a need for further optimization or alternative approaches.

Betting Evaluation

Using the 2019-2020 NBA season, we did a comprehensive analysis using various recurrent neural network (RNN) architectures, including Long Short-Term Memory (LSTM) and transformer models, to forecast NBA game outcomes. Leveraging historical game data spanning October to February, we trained these models to predict the probability of each team winning individual matchups. Alongside this, we utilized an odds dataset containing money lines for each game, enabling us to simulate betting scenarios based on the model predictions. After the initial training and testing of each model, we used each model to predict the outcomes of the NBA games in the 2019-2020 season which we we also have the odds data.

The predictive power of the RNN, LSTM, and transformer models allowed us to generate probabilities for each team's victory in upcoming games. Merging these probabilities with the corresponding money lines from the odds dataset, we devised a betting strategy that allocated wagers proportional to the discrepancy between the model's predictions and the odds offered by bookmakers. Subsequently, we meticulously tracked the performance of these simulated bets over the October-February period in the 2019-2020 NBA season. Profits and losses were calculated by comparing the actual game outcomes with the betting decisions made using the model-generated probabilities and the odds dataset. Positive returns indicated instances where the model's predictions outperformed the odds, resulting in profitable bets, while negative returns highlighted scenarios where the predictions fell short, leading to losses.

The assessment of profitability and losses not only gauged the models' accuracy in forecasting game results but also shed light on the viability of utilizing deep learning in making informed betting decisions within the dynamic landscape of professional basketball. Interestingly enough, the deep learning models we used across the RNN, LSTM, and transformer models, did not outperform a random guessing model, as most of them only had accuracy scores slightly above 50%. The Transformer model did not even score an accuracy over 50%. The performance metrics reflected the lack of accuracy with precision scores also around 0.5 and an f1 score between 0.6 and 0.7. However, an accuracy of only slightly over 50% was enough to generate a significant amount of profit.

We created a function, "calculate profit", which served as the cornerstone in determining the profit accrued per game based on predictive outcomes, actual winners, bet amounts, and associated money lines. By taking into account the model's prediction of the game winner ("prediction"), the actual winner ("winner"), the bet amount placed ("bet"), and the corresponding money line ("moneyline"), this function systematically computed the profit or loss incurred. If the model's prediction aligned with the actual winner and the money line was positive, signifying an underdog status, the function calculated the profit by multiplying the bet amount by the ratio of the money line to 100. Conversely, if the model's prediction matched the winner for a negative money line, indicative of a favorite, the profit was calculated by dividing the bet by the absolute value of the money line and then multiplying by 100. In cases where the prediction did not match the actual winner, the function computed a loss equivalent to the bet amount. This function streamlined the process of evaluating game-specific profits or losses. We used the function to calculate the profits or losses for every game in the October to February period in the 2019-2020 season, assuming a $100 wager per bet, and recorded our earnings in Table 1 below.

	Prediction Accuracy	**Profits/Losses**
RNN	0.508	$42431.53
RNN Pruned	0.508	$42431.53
LSTM	0.502	$42431.53
Transformer	0.498	-$4151.74

Table 1: Model Performance and Profits

In the realm of sports betting, even a predictive accuracy slightly above 50% can be sufficient to yield a profit, particularly in the context of moneyline bets on NBA games. A predictive accuracy slightly over 50% implies that the model is consistently making correct predictions more often than chance. In the context of moneyline betting, where odds are assigned to each team based on their perceived likelihood of winning, even a marginally better predictive performance opens avenues for exploiting discrepancies between the model's predictions and the odds offered by bookmakers.

A predictive accuracy of slightly over 50% means that the model can identify mispriced odds or instances where bookmakers undervalue certain teams. By capitalizing on these opportunities and placing bets accordingly, even with a relatively narrow edge, bettors can accumulate profits over time. The key lies not only in the accuracy of predictions but also in strategic betting, optimizing wager sizes based on the discrepancies between the model's predictions and the implied probabilities from the moneyline odds. Over a significant number of games, this slight edge can compound into a profitable strategy, demonstrating that consistent, albeit modest, predictive accuracy can translate into a successful betting endeavor in the dynamic and competitive landscape of NBA games.

Conclusion and Future Work

In this project, we attempted to leverage the tools of deep neural networks in the form of RNNs, LSTMs, and Transformers, in an attempt to unravel dynamic patterns and trends within the vast statistical datasets of NBA data to gain an advantage over traditional bookmakers in the betting market. These models, each with different strengths, aimed to decode the multitude of complexities in NBA game dynamics. However, the path to refining these predictive models opened our eyes to multiple challenges and shortcomings when it came to generalizing predictive trends in the outcome of NBA games.

With our three primary deep-learning models of RNNs, LSTMs, and Transformers, we tried to model individual NBA games and matchups using sequence-based prediction tasks. RNNs as in many cases struggled with NBA data in terms of keeping track of long-term dependencies for teams across multiple games and were unable to generalize concepts such as a team's recent form and strengths/advantages over the teams developed over time. In response to the lack of improved accuracy over random guessing produced by RNNs, we attempted to use LSTM models to try and capture intricate temporal relationships that may be better at capturing long-term sequences and memory in the context of teams and the games they play over some time. They too however struggled to generalize the mass amounts of NBA data and complex statistical patterns and predicted the same result for the majority of their predictions, facing many of the same challenges as RNNs. The improvement between LSTMs and RNNs was ultimately negligible. Our last attempt at trying to model and generalize our dataset of NBA game statistics dataset, we tried to use a Transformer model. The Transformer model, however, performed worse than the RNN or LSTM models, once again unable to capture the complexities of our vast dataset. The overall performance of the transformer was worse than a random guess. The failure of these models would suggest that although we took pre-processing steps to model the NBA games and data as such, our dataset ultimately did not contain intricate sequential patterns that are best suited for RNNs, LSTMs, and Transformers to handle.

However, in the evaluation of our models on the betting simulations we ran using games and odds from the 2019-2020 NBA season, it becomes evident that while the pursuit of improving the performance of NBA sports betting models and predictions through deep learning continues to face many challenges, even a slight edge in accuracy can pave the way for profitable betting strategies. Even with prediction accuracies only slightly above 50% (better than a random guess), we can generate a relatively good profit from our bets, assuming we bet a standard wager on every game. This success would suggest that it is worthwhile and possible to further generalize NBA statistics and betting odds to take advantage of under and/or over-valuations of teams and matchups by traditional bookmakers.

This takes us to possible avenues of future work in the field of predictive analytics for sports and corresponding betting strategies, in particular when applied to the NBA. We believe that the first step in future research should be enhanced feature engineering regarding our statistical data on NBA games. We would look to enhance the feature selection process, and the possible addition of more advanced and derived statistics that may help better generalize the data and/or be more indicative of wins and losses. This may include more statistical measures such as player-specific data, player and team metrics that evolve throughout a game or season and external features such as injury reports and social media context around games through Natural Language Processing techniques. The next step in future work would be to explore different modeling techniques such as ensemble techniques where we could combine predictions from multiple models to generate more accurate and robust predictions. Lastly, we could try different deep learning techniques that better capture statistical relationships over time, such as using Reinforcement Learning, which may be suitable for this problem because of its sequential decision-making process and adaptive ability to dynamic models.

GitHub Repository

NBA Sports Betting GitHub
Raw Link: https://github.com/JasonHysy/Nba-Sports-Betting-Model

References

Askew, J. 2023. Betting Strategy – Best Sports Betting Strategies. December 17, 2023.

Dataquest. 2023. NBA Games Project. https://github.com/dataquestio/project-walkthroughs/tree/master/nba_games.

Garfjohnson. 2023. NBA-sports-betting-model: This model predicts and bets on every NBA game in a season. https://github.com/garfjohnson/NBA-Sports-Betting-Model/tree/master. Accessed: [insert access date here].

Guo, Y.; Liu, Y.; Bakker, E. M.; Guo, Y.; and Lew, M. S. 2018. CNN-RNN: a large-scale hierarchical image classification framework. *Multimedia Tools and Applications*, 77: 10251–10271.

Ivanković, Z.; Racković, M.; Markoski, B.; Radosav, D.; and Ivković, M. 2010. Analysis of basketball games using neural networks. In *2010 11th International Symposium on Computational Intelligence and Informatics (CINTI)*, 251–256.

J., A.; and Rastegari, H. 2013. A Review of Data Mining Techniques for Result Prediction in Sports. 2.

Jordan. 2021. Using machine learning to predict NBA winners against the spread.

Loeffelholz, B. J.; Bednar, E. M.; and Bauer, K. W. 2009. Predicting NBA Games Using Neural Networks. *Journal of Quantitative Analysis in Sports*, 5.

Sherstinsky, A. 2018. Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network. *CoRR*, abs/1808.03314.

Tomar, A.; Ghosh, B.; Manjunath, C.; Addapalli, V.; and Krishna. 2020. Employing Deep Learning In Intraday Stock Trading.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2023. Attention Is All You Need. arXiv:1706.03762.