CSCI 567 Final Project

Gumas Dean [*1] **Daniels Zachary** [*1] **Martínez Esteban** [*1] **Sidhu Agam** [*1] **Cobb Spencer** [*1]

Abstract

Predicting player performance in fantasy sports, particularly in the context of the English Premier League (EPL), involves modeling complex, sequential data influenced by numerous factors. This paper explores the use of various machine learning architectures to predict player performance, with a particular focus on the recurrent neural network (RNN) architecture and its ability to capture temporal dependencies in player data. We investigate and implement several model architectures, including a deep 1-dimensional CNN, 2-dimensional CNN, and an RNN with an attention mechanism. Our results demonstrate that the RNN model outperforms other architectures, particularly for goalkeepers and midfielders, by effectively incorporating the temporal context of player performance. The addition of an attention layer further enhances the RNN's ability to assign meaningful weights to sequential data, improving its predictive accuracy. The model's performance was evaluated using metrics such as mean squared error (MSE) and mean absolute error (MAE) across various player positions, revealing significant improvements in error reduction for most positions, with the exception of forwards. Additionally, hyperparameter optimization, including the window size and feature selection, was important for model accuracy, with position-specific tuning leading to the best results. The findings of this study underline the importance of modeling temporal dependencies and leveraging complex, multi-dimensional player features for improved performance prediction in fantasy sports applications.

Code repository located here: https://github.com/DeanGumas/ml-premier-predictor

[*]Equal contribution [1]Department of Computer Science, University of Southern California. Correspondence to: Dean Gumas <gumas@usc.edu>, Zachary Daniels <zsdaniel@usc.edu>, Esteban Martínez <em36854@usc.edu>, Agam Sidhu <agamsidh@usc.edu>, Spencer Cobb <srcobb@usc.edu>.

1. Introduction and Background

The Premier League, commonly known as the English Premier League (EPL), is the highest level of football in England. Each season, 20 teams compete in a schedule that usually runs from August to May. Teams play a total of 38 matches, facing each of the other 19 teams both at home and away.

In this project, the main objective is to predict the performance of EPL players, specifically as measured by their Fantasy Premier League (FPL) points. FPL points provide a standardized metric for evaluating a player's contributions during a match, accounting for key actions such as goals, assists, clean sheets, saves, minutes played, while penalizing events like missed penalties or yellow cards. Using FPL points as the target metric establishes a clear baseline to evaluate the accuracy of predictions and measure how effectively models capture player performance.

Forecasting the performance of football players, particularly in the English Premier League (EPL), is an inherently complex and multifaceted challenge. Football is a low scoring sport, characterized by high variability and unpredictable outcomes. A player's performance in any given match can be influenced by a multitude of factors, including team dynamics, opposition strength, tactical adjustments, injuries, and even external conditions like weather. These dynamic variables introduce substantial noise into the data, making reliable predictions a daunting task.

Adding to the difficulty are underlying factors that remain poorly understood or difficult to quantify, such as psychological states, team morale, or subtle tactical nuances, which further complicate the problem. Accurately predicting a player's future performance requires not only an understanding of broader team trends but also the ability to model individual attributes such as playtime, positioning, and recent form.

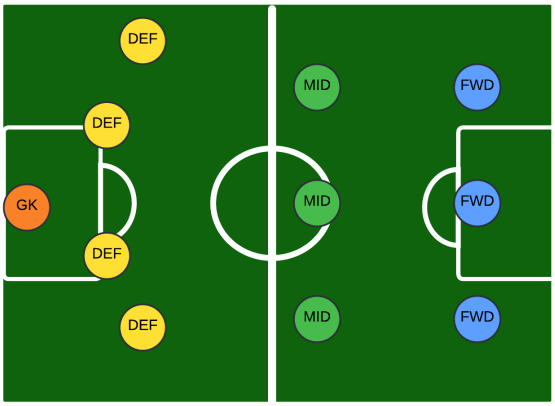

Figure 1. Depiction of players on the pitch, showing typical positions for goalkeeper (GK), defenders (DEF), midfielders (MID), and forwards (FWD).

This project builds upon the work presented in the paper "Deep Learning and Transfer Learning Architectures for English Premier League Player Performance Forecasting" (Frees et al., 2024). The paper explores various machine learning methodologies, such as convolutional neural networks (CNNs), for forecasting player performance in the English Premier League. By utilizing this work as a foundation, the aim is to refine and expand the proposed strategies to improve prediction accuracy and address the challenges inherent in this task. Since the CNN model demonstrated the best performance among the approaches tested in the paper, this project focuses exclusively on comparing against this model as the benchmark.

A variety of architectures were tested, including a deep 1-dimensional CNN, a 2-dimensional CNN, and a recurrent neural network (RNN) with a temporal attention layer. All models featured a dense feedforward network that processes the output of the CNN or RNN before a linear output layer, allowing each model to integrate the learned feature representations with external scalar metrics (i.e. match difficulty). Of all evaluated architectures implemented, the deep 1D CNN and RNN/Attention based approaches proved the most effective, providing modest gains against the baseline 1D CNN approach. Due to the computational overhead of training the deep CNN model, the RNN/Attention model was optimized through grid search across several parameters to determine those that would be optimal for the application.

2. Method to Implement

2.1. 1D CNN

The baseline methodology, as described in the reference study, utilized a one-dimensional convolutional neural network (1D CNN) to process raw player performance data. The convolutional output was subsequently flattened and concatenated with a scalar input representing the difficulty of the upcoming match. This combined input was then passed through a two-layer dense feedforward neural network, which produced a final scalar prediction of the player's fantasy performance score. The convolutional kernel size was adapted based on the players' position: a kernel size of 1 was optimal for goalkeepers, defenders, and forwards, while a size of 2 was more effective for midfielders. Additionally, the input window size—the number of weeks of historical player data used for prediction—varied by position. Defenders and forwards utilized a window size of 9 weeks, goalkeepers 6 weeks, and midfielders 3 weeks.

Week 1			Week 2		
Pts	**Minutes**	**Goals**	**Pts**	**Minutes**	**Goals**
8	72	1	5	23	0

Table 1. 2-week 1D sample input window for the player Trent Alexander Arnold

As mentioned, the baseline achieves the best results with a 1x1 convolution. This baseline convolutional network can essentially be viewed as performing a linear mapping before the fully connected layers, due to the kernel size being one. In this configuration, each spatial position is processed independently, with features transformed according to the number of filters applied. Consequently, the network can be interpreted as a fully connected neural network preceded by a channel-wise linear transformation implemented using 1x1 convolutions.

2.2. Deep 1D CNN

The first proposed improvement was a deep CNN architecture, which extended the baseline model by incorporating two sequential convolutional layers with average pooling layers in between, prior to flattening and passing the output to the two-layer dense feedforward network. The underlying hypothesis was that with additional convolutional layers, the model could be enabled to extract hierarchical feature representations which in turn can capture more complex patterns within the input data and improving performance. However, despite the theoretical promise of this approach, experimental results showed comparable performance to the baseline model, coupled with greatly increased computational overhead.

2.3. 2D CNN

The next model tested was a two-dimensional convolutional neural network (2D CNN), which applied a single convolutional layer with a 2D kernel designed to pro-

cess player data spanning multiple weeks. Unlike the 1D CNN, which operates across a single axis, the 2D CNN applies convolutions along both the temporal and feature axes. The temporal axis represents the sequence of weeks, capturing the progression of a player's performance over time, enabling the model to track important trends. The feature axis, on the other hand, represents the metrics recorded for each week, such as goals, assists, and match difficulty. The hypothesis behind this approach was that 2D convolution could better contextualize performance trends across weeks, potentially mitigating the impact of outlier performances (e.g., unusually strong or weak weeks).

Additionally, the two-dimensional convolution can theoretically recognize patterns among related metrics that occur close to each other. For example, a player's points from consecutive weeks are adjacent in the data. Thus, repeatedly applying the shared kernel enables the network to recognize patterns related to changes over time.

However, this method performed worse than the baseline during initial testing. This outcome is understandable, as the 2D convolution operated not only across the temporal axis (weeks) but also across feature dimensions, potentially diluting the integrity of individual data fields.

Week	Pts	Minutes	Goals	Clean Sheet
Week 1	8	72	1	Yes
Week 2	5	23	0	No
Week 3	1	90	0	Yes
Week 4	6	90	0	Yes

Table 2. 2-week 2D sample input window for the player Trent Alexander Arnold

2.4. Bidirectional RNN + Attention

Of all the architectures previously considered, both those we newly implemented as discussed previously, and those recreated from the baseline paper, the best results were achieved using the 1D CNN. This outcome highlights the value of modeling sequential data by focusing on localized temporal patterns. However, to better understand player performance trends, it is crucial to consider the inherent temporal dependencies in the data. Specifically, player performance tends to exhibit a strong correlation with recent history. For instance, a player who has been performing well in recent weeks is more likely to continue the trend, but a player experiencing a slump is less likely to deliver outstanding results immediately. This suggests that temporal patterns are key to accurately modeling and predicting performance.

Moreover, while the temporal aspect is significant, it is also evident that short-term dependencies carry more weight than long-term ones in this domain. Player performance is often influenced more by recent form, fitness, and external factors—not events from distant weeks. These observations make a case for adopting architectures specifically designed to capture such temporal dynamics.

This reasoning led us to consider the use of Recurrent Neural Networks (RNNs), which are well-suited for modeling sequential data and learning temporal dependencies. By structuring the input to represent a player's data from consecutive weeks, we enable the RNN to track short-term trends while effectively encoding the temporal relationships. This approach aligns with the natural progression of player performance and allows the network to focus on capturing meaningful patterns in the sequence. With this motivation, we developed our RNN architecture as a natural extension to address the temporal aspects of player performance prediction.

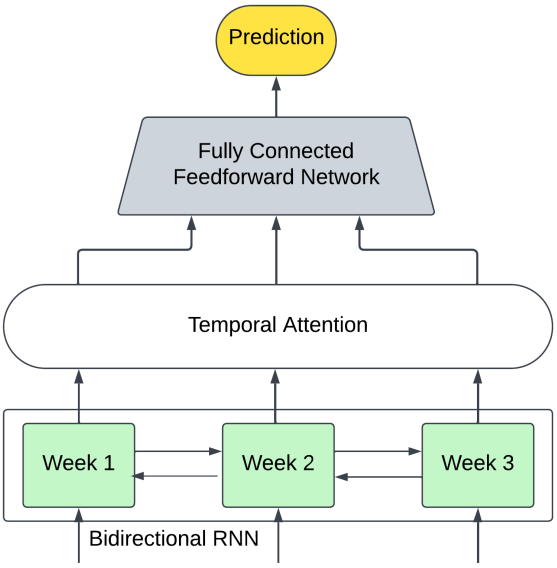

Figure 2. Diagram of RNN Attention Mechanism

The final model combined a recurrent neural network (RNN) with a single attention layer. The core intuition with this approach was that the bidirectional RNN would capture temporal dependencies in the player data, allowing context to be shared across weeks, thereby mitigating the influence of outlier weeks. The directional nature of the network allows the model to simultaneously process data in forward and backward movements, facilitating the integration of insights from previous performances with future trends. This flow enhances the model's ability to understand a player's progression by balancing short-term variations with long-term trends. Initial experiments with a standard RNN demonstrated marginal improvements over the baseline, motivating further exploration of this

architecture.

The first modification to the RNN structure involved stacking multiple RNN layers to boost the model's ability to learn temporal dependencies on the data. Unfortunately, this modification did not yield significant improvements due to computational overhead and possibly vanishing gradients that tend to be associated with deeper RNNs. However, introducing a bidirectional RNN led to notable gains, as it leveraged both the past and future context in the sequence. To further enhance the model's ability to distill meaningful patterns from the RNN outputs, a single attention layer was incorporated, taking each sequential RNN output vector as a separate input token. In theory, this layer would dynamically assign context-aware weights to the sequential outputs of the RNN, effectively creating a refined and well-contextualized representation for downstream processing. This addition proved effective, demonstrating improvements over both the bidirectional RNN and the baseline model during initial testing. Consequently, this architecture was selected for further evaluation.

This approach introduced new hyperparameters, including the dimensionality of the RNN hidden states and the attention layer. To streamline the implementation within the project's time constraints, the dimensions of the hidden states and attention matrices were kept identical.

In this work, a single weight matrix was trained to compute attention scores across the entire input sequence, rather than employing separate key and query matrices for each input token. However, the approach retains the value matrix, which is used to compute the weighted sum of the input features based on the learned attention scores. Although this method is less expressive than traditional attention mechanisms, it significantly reduced computational complexity while allowing the model to effectively learn the relative importance of each week of data. In initial testing, performance was quite comparable to a traditional attention implementation, justifying the simplified implementation for this project. The output of the attention layer was flattened, concatenated with the match difficulty scalar, and passed through the two-layer dense feedforward network as in previous models.

Future work could explore a full attention layer with separate query and key matrices as well as adding multiple sequential attention layers with dense networks in between. However, these extensions were not pursued in this study due to time and compute constraints.

3. Implementation

For help with our implementation, we leveraged the data collection work completed by Anand Vaastav, who compiled premier league statistics and player scoring data from 2020-21 and 2021-22 utilized in the reference paper (Vaastav, 2023). To ensure consistency, we used the same data for this study. We also used the data cleaning and preparation functions included in the code repository for the reference paper. These functions helped collect data by player names, correct minor errors such as misspellings, and remove player data for weeks when the player logged less than a set number of minutes. Failing to drop benched players artificially inflates model performance, as predicting these players to score zero points becomes a trivial task. These functions were also helpful in selecting the subset of features we would use as input to our models. Feature selection proved important to performance, as it affected the size of our network. This is discussed further in the Results section of this paper.

For our models, we used the PyTorch (Paszke et al., 2019) framework to help construct and train the various architectures tested. PyTorch has a dynamic computation graph, a key feature of its framework, which plays a vital role in building intricate architectures including CNNs and RNNs. The torch.nn class was used along with its various subclasses to create the building blocks of our networks. PyTorch also provides the various activation functions, softmax implementations, and tensor mathematics necessary to create and develop our networks without building the foundational requirements from scratch.

For the RNN model we investigated in particular, we used the torch.nn.RNN module to create the bidirectional RNN. This RNN architecture was developed with a number of layers equivalent to the number of weeks of player data being processed per prediction (the window size). The hidden state size was set as a hyperparameter, which was gridsearched across the values of 64, 128, and 256.

Then, each sequential output of this RNN was passed to a custom attention layer, which plays a prominent role in the effectiveness of the model. The attention layer used a trainable weight matrix, which was multiplied by the RNN output, and the product was passed through a hyperbolic tangent (tanh) activation function to add non-linearity while also normalizing the values. This output was then multiplied by a trainable value matrix and converted into attention scores with a softmax function. Finally, the output was calculated as a weighted sum by multiplying the original input from the RNN by its computed attention weights. This formed the output of the attention layer, which would then be processed by our dense feedforward network, permitting the network to make predictions based on the learned context and temporal dependencies.

The dense feedforward network served as the final processing stage in the model. The network converted the attention output into a single scalar, representing the ex-

pected player performance in the upcoming match. The output received from the attention layer was flattened to be a one-dimensional vector to ensure it was compatible with the architecture of the feedforward network. The vector was then concatenated with a scalar indicating the difficulty of the upcoming match. This formed our feedforward input layer, with dimension:

$$(window\_size * rnn\_hidden\_state\_dimension) + 1$$

Here the plus one is due to the scalar addition of the match difficulty to the input layer. This input layer would then connect to a hidden layer using a ReLU activation function. This hidden layer was designed to have a tunable hyperparameter number of neurons to add flexibility for tuning the model. However, to avoid excessive iterations and reduce the computational burden that may arise in grid search, the number of neurons for this was set equivalent to the hidden state dimension of the RNN. This hidden layer would then connect to a single output neuron, representing the expected player fantasy performance score in the upcoming match.

The design for other networks tested was similar, instead using the torch.nn.Conv1d and torch.nn.Conv2d modules for the initial layers. These implementations allowed convolutional operations to be customized to one-dimensional and two-dimensional data. All networks described in the previous section, methodology, were implemented with the PyTorch (Paszke et al., 2019) architecture for training and optimizing neural networks. The approach used for training the model was through using standard backpropagation (handled by PyTorch), with mean squared error (MSE) being employed as a loss function to compare the network output with the player's actual fantasy performance in the predicted match. This choice of loss function was identical to that used in the reference paper to ensure consistency.

An early stopping mechanism was also implemented to mitigate overtraining of the models. This mechanism was designed with two important parameters: tolerance and patience. The tolerance parameter is set to be the minimum reduction in validation loss needed to continue training. In other words, the tolerance parameter would be set to a value the validation loss must decrease by. The patience parameter is the maximum number of consecutive integers that the training model can do without meeting this threshold.

Mean Squared Error (MSE) and Mean Absolute Error (MAE) were used to evaluate the performance of each network. Identical seeds for PyTorch, NumPy, and the random library were set for all comparisons to ensure no network had an unfair advantage when shuffling data. Matplotlib plots confirmed that both training and validation errors decreased over time. (Hunter, 2007).

4. Results

Once it was established that the RNN model delivered the best results, we proceeded to fine-tune its performance by implementing a grid search to optimize the model's hyperparameters.

Since the RNN was a completely different model from the baseline, we could not assume that hyperparameters such as the learning rate or early stopping criteria would remain optimal. The RNN's unique training dynamics required re-evaluating these settings to ensure stable training and avoid overfitting, rather than relying on assumptions from the baseline model.

Experiments were conducted using learning curves to determine the optimal training hyperparameters. The final settings included 2000 epochs, a learning rate of 0.00001, a batch size of 32, and early stopping with a tolerance of 1×10^{-4} and a patience of 20 iterations. A custom grid search implementation was developed to identify the best configurations for various hyperparameters, including the window size (number of weeks), the number of dense neurons, the number of features to include, and the stratification strategy tailored for GK, DEF, MID, and FWD models.

Fine-tuning these parameters posed a significant computational challenge. During the initial fine-tuning rounds, where multiple parameters were adjusted simultaneously, we trained up to 800 different models, representing the cross-product of all parameter settings. The training was conducted on an M3 MacBook Pro and required up to 24 hours for only 200 epochs, highlighting a substantial time constraint. For evaluation and additional training involving up to 2000 epochs, a PC with a Nvidia RTX 3060 GPU was used.

It makes sense to tune and test different hyperparameters for each player position, particularly because some positions are more sensitive to specific numerical features. For example, a goalkeeper's performance evaluation is highly correlated with whether the team keeps a clean sheet, making this feature especially important for models predicting goalkeeper performance. Customizing hyperparameters for each position allows the model to capture these unique dependencies better, improving overall predictive accuracy.

4.1. Player Numerical features

It is interesting to note that the baseline model achieved its best performance when the input features were restricted solely to the points the player accumulated in previous weeks, excluding additional metrics such as goals, assists, or whether the team achieved a clean sheet. At first glance, this result may seem counterintuitive, as it

is natural to assume that incorporating more detailed information would lead to improved predictions by providing the model with a richer understanding of player performance.

However, this can be explained by the fact that points act as a summary statistic, effectively encapsulating the most relevant aspects of a player's performance. In doing so, points inherently reduce the complexity of the input data, allowing the model to focus on the most meaningful patterns without being distracted by less relevant details. Including additional metrics such as goals or assists could introduce noise or redundancy into the model, particularly if these features are strongly correlated with points or do not independently contribute valuable new information.

4.2. Comparison against baseline

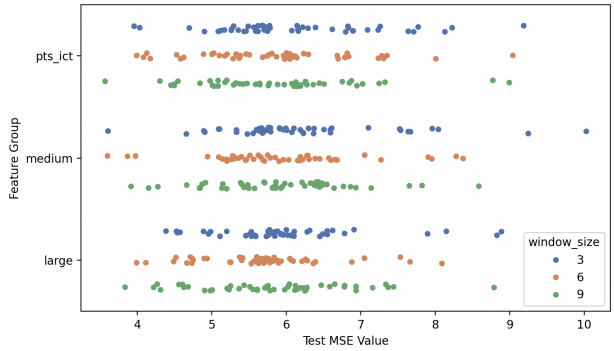

Figure 3. Distribution of test MSE values by feature group (pts_ict, medium, large) under varying window sizes (3, 6, 9).

In contrast, for our RNN models, including these additional player features leads to improved predictions. This result highlights the capability of our new architecture to extract meaningful patterns from more complex data. Unlike the baseline model, the RNN effectively processes the additional features and appears to weight them appropriately, balancing their contributions without letting them introduce unnecessary noise. This ability to integrate diverse input features underscores the strength of our approach in handling richer, multidimensional datasets while still maintaining robust prediction accuracy.

4.3. Window Size

As mentioned previously, the window size hyperparameter determines how many previous weeks of data are used to predict a player's performance. In the baseline paper, the authors found that the optimal window size varied by position: a window of size 9 worked best for defenders and attackers, a size of 3 for midfielders, and a size of 6 for goalkeepers. However, in our experiments, we found that different window sizes yielded the best results for each position: 9 weeks for goalkeepers, 3 weeks for midfielders and defenders, and 15 weeks for forwards. This highlights the varying importance of temporal context across player roles.

The RNN's ability focus on recent data allowed us to effectively use large windows for goalkeepers and attackers without negatively impacting predictions. For goalkeepers, this enabled the model to capture trends over extended periods, such as consistent clean sheets or defensive strength. Similarly, attackers benefited from longer windows that accounted for performance streaks or patterns over multiple weeks. In contrast, for defenders and midfielders, we observed no significant benefit from using more than 3 weeks of data, as their performance patterns appear to rely more on shorter-term trends.

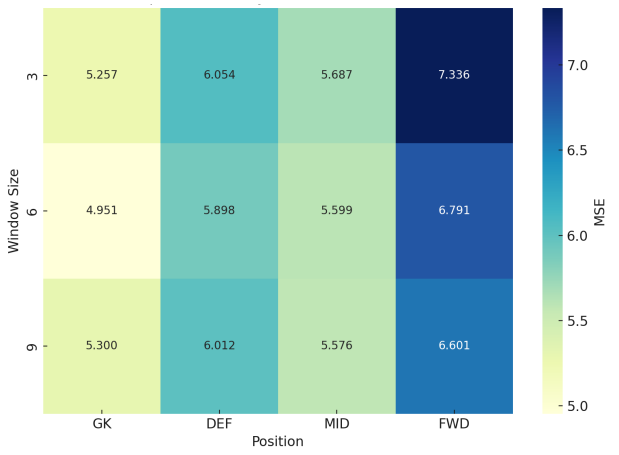

Figure 4. Heatmap of test MSE values across player positions and varying window size hyperparameters.

It is also important to note that, unlike the baseline, our weekly input data incorporates more numerical parameters, providing the model with a richer feature set for each time step. This additional information likely contributed to the RNN's ability to perform well even with varying window sizes, further enhancing its predictive accuracy across player positions.

4.4. Overfitting

Having a large number of fully connected neural nodes initially provided good performance for certain player positions during the experimentation phase (200 epochs). However, when training the model for a longer duration, around 2000 epochs, a clear tendency toward overfitting emerged. In some cases, the training MSE was signifi-

cantly lower than the test MSE, with the test error exceeding the training error by over 300%, particularly for forwards and midfielders. This tendency to overfit became more prominent with larger model sizes as well, leading to us selecting 64 for the RNN hidden states, attention weight, and value matrices sizes in our final model.

4.5. Optimal Parameters

Interestingly, through our grid search and evaluation, the smallest window size of 3 proved optimal for the defender and midfield positions, while the largest proved most optimal for goalkeepers and forwards. This is possibly due to more performance consistency across the defender and midfield positions, while the goalkeepers and forwards are more prone to wild swings due to goals against or goals which can greatly affect their fantasy points.

Pos	Weeks	Tolerance	L. Rate	# Features
GK	9	1×10^{-4}	1×10^{-5}	5
DEF	3	1×10^{-4}	1×10^{-5}	10
MID	3	1×10^{-4}	1×10^{-5}	10
FWD	15	1×10^{-4}	1×10^{-5}	10

Table 3. Optimal Parameters for RNN Models by Position

Numerical features of the goalkeeper:

- Total Points
- Minutes
- Clean Sheets
- Red cards

Numerical features of the other positions:

- Total Points
- Minutes
- Clean Sheets
- Red Cards
- Penalties Saved
- Yellow cards
- Goals Scored
- Assists
- BPS
- Own Goals

Our initial grid search using window sizes of 3, 6, and 9 outperformed the baseline model for goalkeepers, defenders, and midfielders. However, forwards showed a high error rate, exceeding the baseline. Observing that performance improved with larger window sizes, we ran a second grid search using 9, 12, and 15 for forwards, with 15 performing the best. The additional data appeared to help compensate for the volatility in scoring patterns, suggesting larger windows could be even more beneficial in the future.

4.6. Comparison against baseline

Figure 5 presents a comparison of the test mean squared errors (MSE) for all models evaluated during the project: CNN 1D (baseline), CNN Deep, CNN 2D, and RNN + Attention. This graph allows us to visualize the relative performance of each architecture for different player positions (GK, DEF, MID, FWD) as well as the average MSE across all positions. Including this graph demonstrates the comprehensive exploration of model architectures, highlighting how the RNN model consistently achieved superior results compared to other configurations, particularly for goalkeepers (GK) and midfielders (MID).

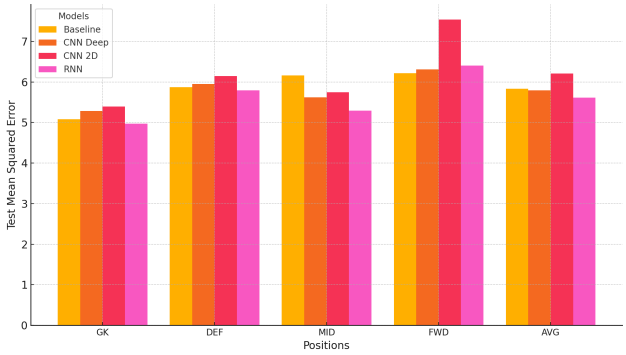

Figure 5. Performance comparison between all models, stratified by player position.

Figure 6 focuses on a direct comparison between the best-performing model (RNN) and the baseline model (1D CNN). This graph emphasizes the improvements achieved by the RNN architecture in capturing temporal dependencies and handling the complexity of player performance data. The comparison underscores the RNN's lower MSE for all positions, with the exception of forwards (FWD), where the baseline slightly outperformed the RNN.

4.7. Comparison against baseline

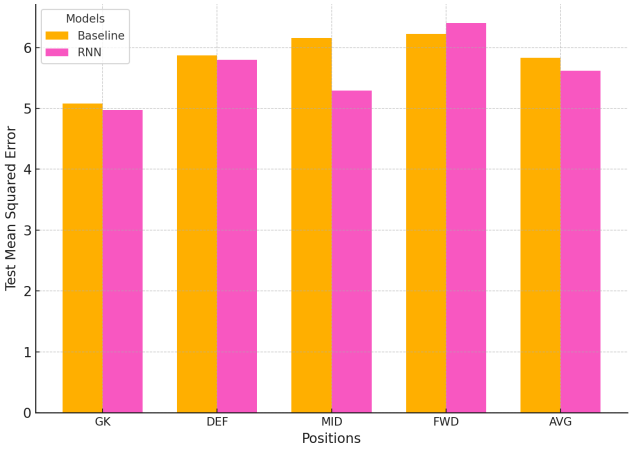

Figure 6. Performance comparison between best experimental model (RNN) and the baseline model (1D CNN), stratified by player position.

The following table provides a detailed numerical breakdown of the test MSE for both the baseline and RNN models for all positions. The table includes the percentage change in performance for each position, calculated as the relative difference between the baseline and RNN errors. This data complements the visual insights of the graphs by quantifying the magnitude of improvement (or decline) for each position. For example, the table highlights a substantial improvement for midfielders (a 14.12% reduction in error), while forwards showed a slight increase in error (2.89%).

Position	Baseline	RNN	% Change
GK	5.08	4.97	-2.17%
DEF	5.87	5.80	-1.19%
MID	6.16	5.29	-14.12%
FWD	6.22	6.40	+2.89%
AVG	5.83	5.62	-3.58%

Table 4. Comparison of Test Mean Squared Errors Between Baseline and RNN Models

5. Conclusion

In this study, we explored various machine learning architectures to predict player performance in fantasy football, with a focus on capturing temporal dependencies in player data. By comparing different models, including CNN based architectures and a bidirectional RNN with attention, we found that the RNN model with attention consistently outperformed the baseline 1-dimensional CNN model for the majority of positions, particularly for goalkeepers and midfielders, but struggled predicting for forwards. The incorporation of an attention mechanism allowed our RNN model to better capture relevant context from past performance, leading to more accurate predictions.

Our experiments highlighted the importance of fine-tuning hyperparameters, including the window size, number of dense neurons, and feature selection, to tailor the model for different player positions. Notably, position-specific feature sets and window sizes enabled the RNN to exploit the unique characteristics of each position, improving prediction accuracy. While the RNN model demonstrated better results for most positions, we also observed some challenges, such as slight performance degradation for forwards, which warrants further investigation.

Despite the promising results, the study was limited by time and computational resources, and future work could explore additional enhancements, such as multi-layer attention mechanisms, more complex feature engineering, and additional model architectures. Further research could address issues of overfitting, particularly in longer training sessions, to improve model robustness and generalization.

Overall, this work contributes to the growing body of research in predictive modeling for fantasy sports and offers a foundation for future developments in performance prediction using deep learning. The ability to model temporal dependencies and integrate diverse player features is essential for improving prediction accuracy and provides valuable insights for both fantasy sports enthusiasts and professional analysts.

References

Frees, D., Ravella, P., and Zhang, C. Deep learning and transfer learning architectures for english premier league player performance forecasting, 2024. URL https://arxiv.org/abs/2405.02412.

Hunter, J. D. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. doi: 10.1109/MCSE.2007.55.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library, 2019. URL https://arxiv.org/abs/1912.01703.

Vaastav. Fantasy premier league repository. https://github.com/vaastav/Fantasy-Premier-League, 2023. Accessed: 2024.