Predicting Win Probability for NBA Games

Jiajie Chen Mohammad Suffian Hamzah Daniel Allen Lepkofker
Scott Thomas Merrill
College of Computing, Georgia Institute of Technology
{jiajiechen, mhamzah3, dlepkofker, smerril7}@gatech.edu

Abstract

With recent legislation allowing sports gambling in some US states, there is ample opportunity to bet and make money on sportsbooks' quoted odds and inefficiencies in their calculations. In this paper, we explore different time-dependent machine learning models and conduct experiments to determine their efficacy in predicting the outcome of NBA games. We trained models to evaluate the outcome of each game by examining the participating teams' previous 10 games. We further tune the hyper-parameters of these models, apply regularization to prevent overfitting, experiment with various loss functions and devise self-supervision tasks to augment their training. We finally show how our models can be ensembled to further improve performance and outperform popular modeling methods applied in the sports betting domain.

1. Introduction

The Professional and Amateur Sports Protection Act of 1992 (PASPA) imposed federal restrictions on sports betting. In 2018, the act was repealed leaving the regulation of sports betting to the states. Today, sports betting is legal in nearly half of the United States and has quickly become one of the country's fastest growing industries. Sportsbooks set betting lines and must balance risk and reward when pricing bets; offering payouts that are too low given the risk of a bet will discourage gamblers and limit profitability while payouts that are too large may expose the Sportsbook to significant losses. To optimize profitability and minimize risk, Sportsbooks must constantly adjust their prices based on what bettors are betting on. Thus, Sportsbooks are akin market makers; they quote some initial lines for a particular game and adjust their quotes based on supply and demand to ensure they profit regardless of the game's outcome. While price discovery through supply and demand works well in many markets, it can also lead to significant market inefficiencies; the Gamestop short squeeze of 2021, for example, demonstrates how prices may significantly deviate from their fair values. Given the ignorance of many bettors and behavioral biases that encourage people to bet in favor of their team, similar mispricings are feasible and even expected in betting markets. In this paper, we seek to use Deep Learning to model the win probabilities of NBA games to help bettors make more appropriate betting decisions and enable a more efficient betting market whereby prices more closely reflect their expected values.

2. Related Work

The problem of predicting a games outcome has monetary incentives and has existed since the beginning of sports betting. Technological innovations of recent years have increased both the amount and availability of data, encouraging the use of Machine Learning and big data techniques to model game outcomes. Lock et al. use Random Forests to predict the live, in-game win probability of NFL games [4]; their model performs well on easy examples – when one team is significantly better than the other – as well as in later stages of the games. However, their model has difficulty predicting the win probability at the beginning of a game when teams are closely matched. Alameda-Basora use Bayesian Networks to predict the total points scored in NBA games at the end of each quarter. When backtested on the 2018-2019 NBA season, his model generated a profit of 10%, thus indicating potentially inefficient betting market prices. Finally, Torres used a Multilayer Perceptron (MLP) to predict the winner in NBA games but found that NBA "experts" could predict a games outcome with a higher degree of accuracy. Further, he found his model only slightly outperforms a simple linear regression model.

3. Dataset

Our dataset comes from nba.com and was scraped using a Python API developed by [6]. Our models were trained using all NBA regular season games from the 2008 – 2017 seasons with the 2017-2018 season used as the validation set. Our trained models were then used to predict the outcomes of games in the 2018-2019 NBA season. Our training set contains 10,646 games while the validation and test sets each include 1,230 games.

To structure the data to enable for the modeling of time dependencies significant pre-processing was required. Specifically, for each game G, we collect the results of the last 10 games played by the home team $H_1, \ldots, H_{10}$ and the results of the last 10 games played by the away team $A_1, \ldots, A_{10}$. This structure enables our models to learn representations that account for a teams overall quality and their current ”form”; that is, how well they’ve played in recent games. Each of H and A contains 94 features relating to the game including:

- Basic Box Score Statistics: points, rebounds, assists, steals, turnovers, fouls, etc.
- Advanced Box Score Statistics: team’s offensive and defensive efficiencies, pace, etc.
- Miscellaneous Features: home indicator, team travel indicator and days between games

Conveniently, the training, validation and testing sets are all complete with no missing features. While there are 94 features for each game, it’s important to note that many of these features are highly correlated such as 3 pointers made and 3 pointers attempted. Given this high degree of correlation, the feature space can likely be reduced significantly using dimensionality reduction techniques without sacrificing much loss in the explanatory power of the data. Further, for each game the target label was manually derived and set to indicate if the home team won game G. This definition of the target label resulted in our dataset being unbalanced as the home team was found to win roughly 60% of the games. The complete dataset used for our experiments can be found here.

4. Approach

As mentioned earlier, previous attempts to use Machine Learning to model sports games have relied on methods such as Random Forests, Bayesian Networks and MLPs. While these approaches have achieved varying levels of success for different sports prediction task, each method utilizes an approach that ignores time dependencies. That is, these model’s will make the same predictions regardless of the ordering of the training data. Failure to model time dependencies is will likely lead to erroneous predictions in the context of sports; good NBA teams for example often go through periods where they play poorly just as bad NBA teams may experience stretches where they play well. Such periods of good or poor play may be a result of easy stretches in a teams schedule, luck or more complicated structural changes in a team due to injuries, player acquisitions or managerial firings. Moreover, to better capture the time dependencies inherent in sports we propose modeling games using Long short-term memory neural networks (LSTM) and Gated Recurrent Unit Networks (GRU). These models are sequential in nature and have been used to achieve state of the art predictions in various time series tasks across many domains. We also experiment with the use of Convolutional Neural Networks (CNN). While CNNs are predominately used in image classification settings, Borovykh et. al [2] demonstrate their ability to capture conditional dependencies and effectively model financial time series data.

While LSTMs, GRUs and CNNs are accredited methods to model time series data, we anticipate several reasons these models may fail to apply in the context of sports betting. We discuss the potential deficiencies of these models and our strategies to alleviate them in the following sections.

4.1. Problem 1: Trivial Parameterizations

Blindly training any model on our target dataset may result in trivial parameterizations; due to the relative imbalance of our dataset, a classifier can easily achieve a 60% accuracy by simply always predicting the home team will win. Alternatively, our model may learn parameterization such that it always predicts the better team will win. To combat these issues, we trained our models using an α-balanced Focal Loss criterion [7] defined as follows:

$$FL(p_t) = -\alpha_t(1-p_t)^\gamma log(p_t) \tag{1}$$

where p_t corresponds probability the classifier assigns to the an example to the correct class and α and γ are tune-able parameters. α_t is a scaling parameter that assigns different weights to different classes and is often set such that incorrect prediction from minority classes produce more loss than erroneous predictions of the majority class. Moreover, with this weighting scheme, our models are encouraged to correctly classify instances in which the away team wins. γ controls the value assigned to well-classified examples. Larger values of γ serve to assign less weight to well-classified examples and incentivizes the model to accurately classify the more difficult examples. Note that when γ is 0 and alpha is the same for each class, Focal Loss reduces to Cross Entropy loss.

4.2. Problem 2: Lack of Data

With 10 NBA seasons and over 10,000 training examples lack of data isn’t the first issue that comes to mind. Relative to the immense size of our feature space, however, 10,000 training examples is likely insufficient. We devise two strategies to tackle this problem. First, we attempt to reduce the dimensionality of the feature space prior to training our models using principal component analysis (PCA), auto-encoders and MLPs. This may be a good idea since many features are highly correlated and many more have low signal to noise ratios. In addition to reducing the feature space, we also attempted to artificially increase the amount of training examples using two self-supervised surrogate tasks. The first surrogate task is a simple data augmentation approach whereby training samples are both weakly and strongly augmented. The weakly augmented training example is then one-hot encoded to create a ground of truth pseudo-label. This pseudo-label is then assigned to the strongly augmented example and our models are tasked with predicting this label. The second surrogate task involves shuffling the ordering of the previous 10 home and away games and forcing our models to predict the ordering of such labels. In summary, we attempt to combat the curse of dimensionality by both reducing the size of the feature space and artificially increasing the amount of training examples.

5. Experiments

Several experiments were performed to identify the optimal parameters for our models. In the following section we discuss the training and parameter selection of our models.

5.1. Model and Hyper-parameter Selection

The first step in the experimentation process involved to selecting a base recurrent network to build our prediction model around. While we briefly explored a basic implementation of a recurrent neural network, it became clear that performance would vastly improve with LSTM and GRU networks; both of these architectures allow for greater control of state information and maintain meaningful memory of previously processed game data. For each model, we further explored varying the number of layers and the number of hidden states in each layer. Figure 1) shows a visual of the impact of these parameters; the number of layers corresponds to the number of stacked LSTM cells while the number of hidden states corresponds to the number of cells in each layer.

To find the optimal number of layers and hidden states we first attempted select a paramaterization that overfit to the training set with the goal of later applying regularizations to reduce such overfitting. This strategy ensures our models have the capacity to represent the dataset. As seen in figure 2 We found that a single layer with 64 hidden states was sufficient to overfit. Increasing the number of layers and hidden states also served to increase the amount of observed overfitting. To limit the size of our architecture search, we limited our search to evaluate 4 model parameterizations – 64 hidden features and 1 layer; 64 hidden features and 2 layers; 128 hidden features and 1 layer; and 128 hidden features and 2 layers.

As for the CNN model, our architecture selection was largely constrained by the dimensionality of the input data. Moreover, the model utilized a single convolutional layer

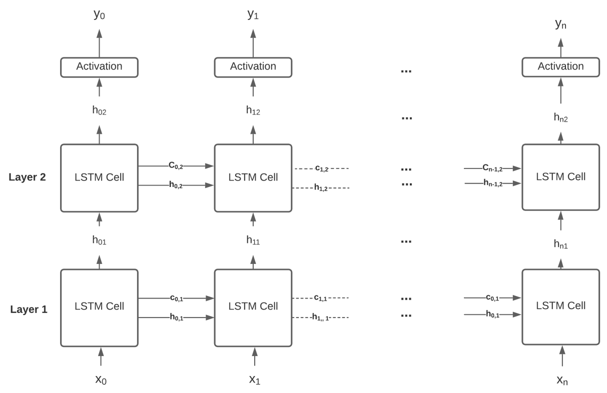


Figure 1. Visualization of stacked LSTM modules.

with a kernel size of 10 x 1, followed by two connected layers. ReLU activation and dropout layers were added in between the model layers. We varied two hyper-parameters for the CNN model: The number of output channels in the convolutional layer and the output size of the first connected layer. We found that increasing the output channels quickly overfitted the data as seen in figure 3, and modifying the output size of the connected layer did little to improve the model. Thus, we settled on a single CNN model with 4 output channels in the convolutional layer and an output size of 80 in the first fully connected layer.

5.2. Regularization Strategies

To reduce overfitting of we consider both the use of dropout and weight decay. With dropout, hidden layers are dropped randomly during training according to a tun-able probability and the model is optimized over the restricted network. Weight decay prevents overfitting by annealing model weights on each training pass thus preventing overly large weights from accumulating.

We found that dropout of stacked modules required very high hyper-parameter values to produce any recognizable regularization impact in the model; intuitively we believe this effect is due to the fact that a single LSTM or GRU module using hidden layers of 64 features was capable of overfitting and thus dropout applied between stacked layers was unable to prevent the final layer from overfitting. Conversely, we found that weight decay had a much greater regularization impact as shown in figure 4, where we achieved peak validation accuracy with a weight decay parameter value of 1×10^{-3}. We believe the weight decay parameter value provides the ”sweet spot” between incrementally learning while not completely overfitting to new training examples. We thus chose to proceed with our experiments using weight decay regularization, however in the future it may be beneficial to try other techniques like L1 regularization. With L1 regularization, the cumulative absolute value

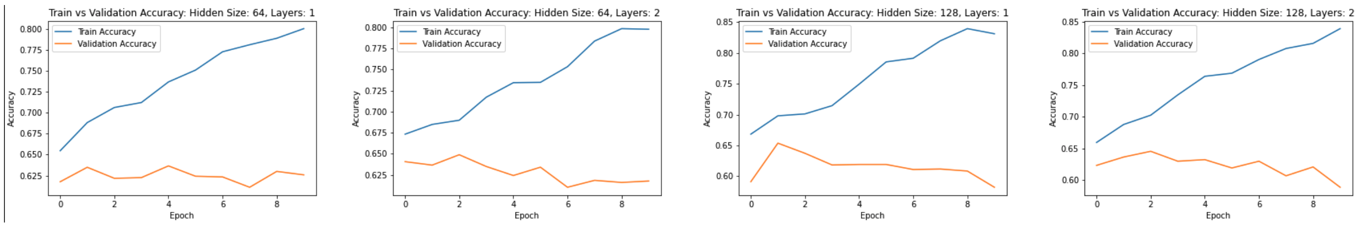


Figure 2. Learning curves for variations of hyper-parameters in LSTM/GRU.

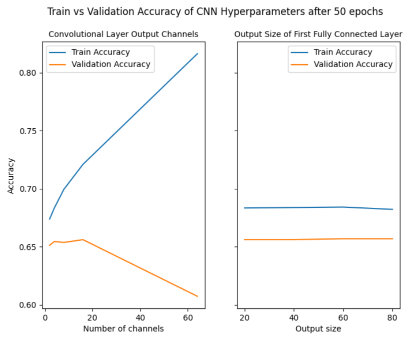


Figure 3. Training and validation accuraccies for variations of hyper-parameters in CNN.

of network weights is minimized which consequently has the effect of acting as a feature extractor. This may be a more appropriate form of regularization given our feature space which includes many irrelevant features. In contrast, weight decay (L2 regularization) has the bias of spreading out the information in the network; thus, a higher proportion of the network weights will likely be assigned to the many irrelevant features.

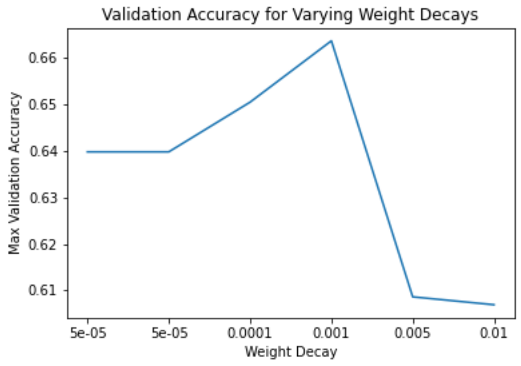


Figure 4. Learning curve for variation of weight decay hyper-parameter.

5.3. Optimizing Focal Loss Parameters

Optimizing the focal loss parameters first involved selecting an appropriate alpha-weighting scheme to handle the class imbalance of our dataset. With just two unique classes, setting $\alpha_t = 1 - \frac{\text{number of examples in class}}{\text{total number of examples}}$ appears most appropriate as this setting equates the maximum loss from misclassifying all positive examples to the maximum loss achieved from misclassifying all the negative examples. We then continued to optimize γ_t. Figure 4 shows the performance of our LSTM model with 2 layers and 128 hidden states for varying values of gamma. As can be seen, the value of γ_t that produced that maximum validation accuracy appears to lie between 1.5 and 2.5. Moreover, we find that hard-negative mining and encouraging the correct classification of the more difficult outlier examples forces the network to learn more complex and less trivial parameterizations which improves generalization ability.

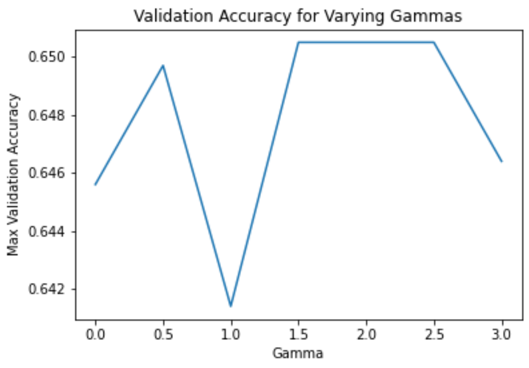
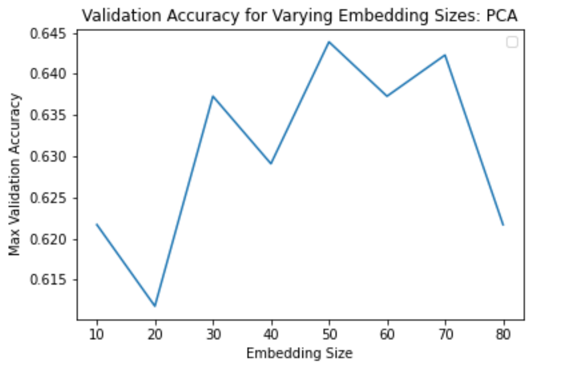


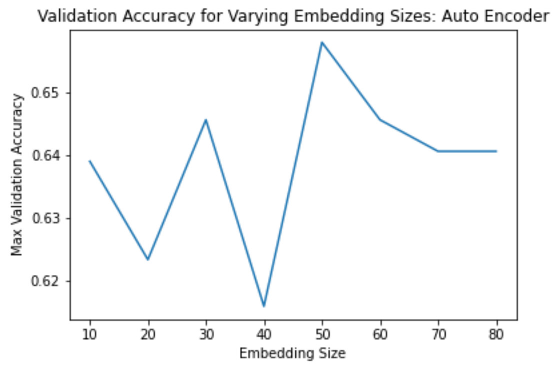
Figure 5. Learning curve for varying γ_t and $\alpha_t = 1$.

5.4. Optimal Embeddings

Given our dataset contains only 10,000 entries and 1,880 features for each entry (two teams last 10 games played) dimensionality reduction techniques were required to reduce overfitting and improve generalization. We explore three techniques to reduce the size of the feature space: PCA, where data is projected into different dimensional space using a linear combinations of features, auto-encoding, where the embedding model learns to compress the feature space while maintaining relevant information, and MLP encodings, where the features are passed through a multi-layer perceptron resulting in fewer dimensions of linearly combined features. The maximum validation accuracies achieved by training an LSTM network over 10 epochs with various embedding sizes is shown in figure 6.

Each embedding method tested improved generalization of our models, albeit different optimal embedding sizes were necessary for different embedding techniques. We observed, however, that each of these embedding methods showed decreasing accuracy as embedding size increased, supporting our belief that reducing the dimensionality of the data would allow for better generalization and less overfitting. These techniques provided similar benefit to regularization due to the nature of the dataset and were necessary for improved performance.





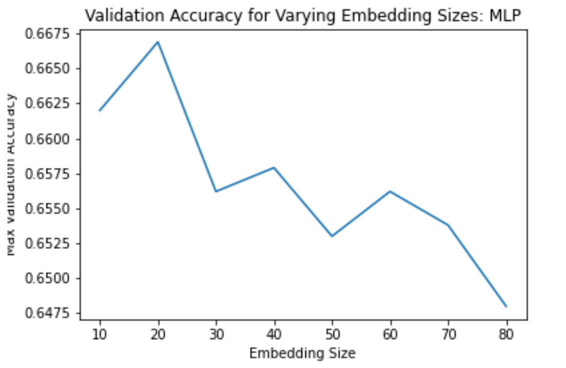


Figure 6. Learning curves for variable embedding sizes with PCA, Auto Encoder, and MLP.

5.5. Surrogate Tasks

To augment the training set with newly generated examples to address the curse of dimensionality we used of two surrogate tasks as mentioned in section 4.2. The first task took each training sample and created two augmentations, one weak and one strong; the weak augmentation provided a pseudo-label which was assigned to the strongly augmented example. The networks were then tasked with predicting the pseudo-label of the strongly augmented example. As can be seen in figure 7, we were unable to improve validation accuracy with the first surrogate task, showing decreasing validation accuracy as the magnitude increased for both weakly and strongly augmented sets. Given the fact that augmentation was performed across a very large feature space, we believe the efficacy was due to the fact that these small perturbations accumulated over many important features resulting in a training sample distant in feature space from the original example. In future experiments, augmentation would only be performed for a small subset of support features that don't significantly alter the outcome of each game; this method would effectively allow us to fill our dataset with more "correct" samples and generalize better to missing data.

The second surrogate task performed involved shuffling the orders of the last 10 home and away games and having our models predict the labels of the re-ordered samples. As can be seen in figure 8 this task was also unable to provide additional training that improved performance. This may indicate that the order of games does not aid in learning the prediction task at hand. While neither of these surrogate tasks served to improve the validation performance, both produced interestingly displayed regularization affects that helped reduce overfitting. Thus in future experiments it may be interesting to consider applying these techniques in place of other regularizers.

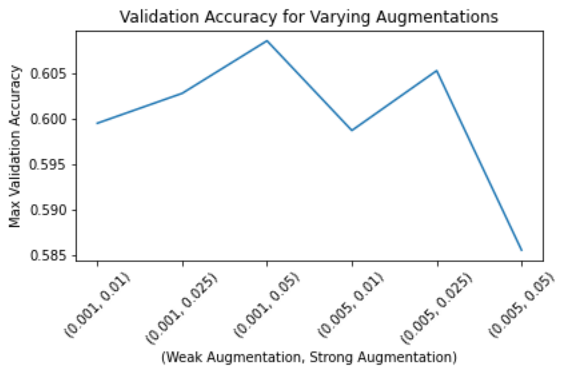


Figure 7. Learning curve for varying levels of data augmentation.

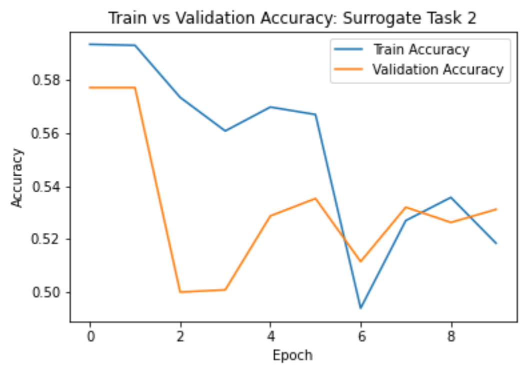


Figure 8. Learning curve for shuffling order of last 10 games as training epochs increase.

5.6. Ensembling Models

The final experiment we performed was ensembling models by training many models and using majority vote to select a label for each sample. Combining the results of many models has a tendency to reduce overfitting and variance as the biases of each model are averaged. Figure 9 demonstrates this feature of ensembling methods – as the number of models ensembled increased, the ensembled model tended to perform better in validation and generalize better to the data. This perhaps implies the models in the ensembled were not correlated and thus by combining uncorrelated predictions variance was reduced and generalization improved. In the future, we would test other methods such as boosting and stacking.

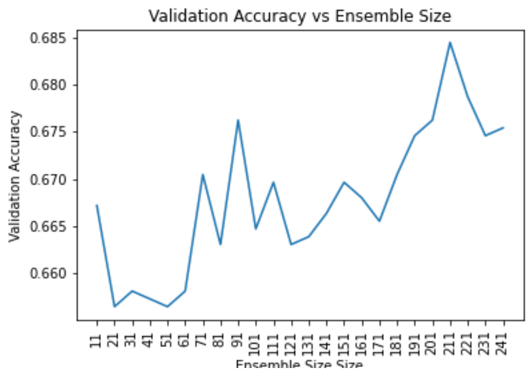


Figure 9. Learning curve for ensembling models as ensemble size increases.

6. Results

From our experiments, hyper-parameters proved to have a significant impact on model performance. The validation accuracy of the LSTM and GRU models were particularly sensitive to the hidden sizes, number of layers, embedding method, embedding size and gamma of the focal loss parameter. Moreover, for each model architecture and embedding method, grid search was run to find the hidden size, number of layers, embedding size and gamma that maximized validation accuracy. Table 1 shows the optimal validation accuracy's achieved for each model and embedding method. These are additionally compared against a baseline Linear Regression classifier, Random Forest Classifier and CNN model mentioned previously. As can be seen, LSTM models appear to slightly outperform the GRU models. This is consistent with the findings of others that LSTM architectures are more equipped to handle longer time-dependent sequences of data than GRU's. Finally, we note that the CNN performs quite well; given this fact and the effectiveness of Borovykh's CNNs models on financial data, such models appear to be well equipped to handle time series forecasting tasks. Overall, we find that each of these models produced modest out-performance against the baseline Random Forest and Linear Regression Classifiers suggesting some benefit to modeling time dependencies in NBA games.

Table 2 shows the 5 top performing models on the validation set, the ensembled model and their performance on the test set. As can be seen, a models performance on the validation set is not highly correlated with their performance on the test set. The ensembled model, however, appears more robust showing similar validation and testing accuracy. The voting mechanism performed by the model thus performs an averaging across many models which serves to reduce the classifiers overall variance. This reduction in variance indicates the individual classifiers in the ensemble may not be highly correlated as if they were, less robustness to the test set would likely be noted. Interestingly, the validation accuracy of the ensembled model is larger than any of the individual model. Moreover, reducing the variance through bagging appears to improve the predictive performance of the model. This might suggest many of these individual classifiers are relying on leverage points to make predictions; that is, they rely heavily on just a few features in making a prediction. When using just a few leverage points to make a prediction, we run the risk of outliers in these features. Averaging across many models thus prevents the model from relying heavily on any singleton feature. Such adds a regularization effect analogous to weight decay that improves generalization.

7. Conclusion

While we find that each of LSTM's, GRU's and CNN's provided improvements over the baseline time invariant models, such improvements were modest. Several complexities in the dataset greatly increased the difficulty of the prediction task. One such difficulty were NBA rules changes occurring over the training period which altered the underlying data generator process and prevented our mod-

Model	Hidden Size	Number of Layers	Embedding Size	Gamma	Validation Accuracy (%)
LSTM PCA Embedding	64	2	40	2.5	65.39
LSTM Auto Encoder Embedding	128	1	50	2.5	66.62
LSTM MLP Embedding	64	2	50	2	65.39
GRU PCA Embedding	128	2	50	2.5	65.97
GRU Auto Encoder Embedding	128	2	60	1.5	66.13
GRU MLP Embedding	64	2	50	1.5	65.06
Logistic Regression Classifier	-	-	-	-	62.85
Random Forest Classifier	-	-	-	-	64.55
CNN Model	-	-	-	-	64.96

Table 1. Model performance tested on data

Model	Hidden Size	Number of Layers	Embedding Size	Gamma	Validation Accuracy (%)	Test Accuracy
LSTM Auto Encoder Embedding	128	1	50	2.5	66.62	61.92
LSTM Auto Encoder Embedding	128	2	60	2	66.37	60.44
GRU Auto Encoder Embedding	128	2	60	1.5	66.13	59.46
GRU PCA Embedding	128	2	50	2.5	65.97	63.40
GRU PCA Embedding	128	2	50	1.5	65.79	62.58
Ensemble	-	-	-	-	68.45	65.05

Table 2. Model performance tested on data

els from generalizing well. One such rule change was the reset of the shot clock after an offensive rebound. Prior to 2018, the shot clock was reset to 24 seconds after an offensive rebound. In 2018, the reset of the shot clock after an offensive rebound was reduced to 14 seconds. This effectively reduces the value of possession after an offensive rebound since the expected points scored if you give a team 14 seconds to shoot is certainly less than the expected points if you give a team 24 seconds to shoot.

Another limiting factor is that our dataset does not include every aspect that could influence the game outcome. One notable missing feature is a teams starting line-up. A team with strong performance in the past 10 games can lose to a weaker team if key players are missing from their lineup. Furthermore our networks were tasked with modeling the outcomes of games without this pivotal information. In future work, we will explore mechanisms to augment our dataset to include lineup information and other features that may enable for more accurate models.

8. Work Division

See table 3 for division of work.

References

[1] Nba scores and odds archives. 8

[2] Cornelis W. Oosterlee Anastasia Borovykh, Sander Bohte. Conditional time series forecasting with convolutional neural networks. 2018. 2

[3] Houshang Darabi Samuel Harford Fazle Karim, Somshubra Majumdar. Multivariate lstm-fcns for time series classification. *CoRR*, abs/1801.04503, 2018. 8

[4] Dennis Lock and Dan Nettleton. Using random forests to estimate win probability before each play of an nfl game. *Journal of Quantitative Analysis in Sports*, 10(2):1–9, 2002. 1

[5] Filip Železný Ondřej Hubáček, Gustav Šourek. Exploiting sports-betting market using machine learning. *International Journal of Forecasting 35*, pages 783—-796, 2019. 8

[6] Thomas Kelley Swar Patel, Randy Forbes and Ethan Swan. An api client package to access the apis for nba.com. `https://github.com/swar/nba_api`, 2021. 1

[7] Ross Girshick Kaiming He Tsung-Yi Lin, Priya Goyal and Piotr Dollár. Focal loss for dense object detection. *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017. 2