COGS 181 Final Project Report
Raghav Kansal
A92121242

Predicting soccer results using sequential models II

Abstract

This project is an extension of my COGS 185 project from last year which tested the effectiveness of using sequential models (Hidden Markov Model and Recurrent Neural Networks) in predicting the results of soccer matches given a previous sequence of results for each team (this was approved by Prof. Tu in office hours). Building upon the results of the previous paper, which concluded that RNNs are indeed effective in predicting results with training and testing accuracies of 65% and 54% respectively compared to a non-sequential neural network baseline of 37% and 39%, this experiment sought to optimize the model by testing cell types, architectures and various hyperparameters. It has produced a significant improvement over the previous paper's results, with training and testing accuracies of 97% and 71% using a 1 layer, 300 cell, LSTM network with an input sequence length of 3. This demonstrates clearly sequential models can have extremely high accuracy in predicting soccer results with minimal data.

Introduction

A team's results are inherently sequential, with one game played after another. While intuitively it may seem likely that there is a correlation between previous results and future ones, there has been little statistical evidence in support of this [1]. Kansal [2] (my COGS 185 paper) showed that while a single result may not be a significant predictor of the next (as evidenced by the failure of a Hidden Markov Model), a greater number of previous results can produce reasonably accurate predictions via a recurrent neural network (RNN), which had a training and testing accuracy of 65% and 54% respectively for an input sequence length of eight games. A non-sequential vanilla neural network in comparison was unable to surpass accuracies of 37% and 39% respectively.

The advantage of an RNN over a regular neural network is that it can learn sequential relationships between data. It therefore stands to reason that the superior performance of the RNN over the neural network in this dataset indicates there is indeed a temporal or sequential correlation between a team's soccer results.

The experiment in this paper seeks to improve upon these results in Kansal [2] by comprehensively testing architectures and hyperparameters, and analyzing the loss curves. Instead of only Long-Short Term Gates (LSTM) networks as in the previous paper, RNNs were implemented using both LSTM cells and Gated Recurrent Unit (GRU) cells, as well as with a simple attention mechanism. Additionally both model hyperparameters (layer numbers and sizes) and training hyperparameters (epochs, batch size and learning rate) were optimized for.

Methodology

Dataset

There are 20 teams which play within a single season in most leagues. In every round, or game day, they each play a single game. A novelty in my approach is to define the input at a single time point as a 20 dimensional vector representing the results of every team on a single game day. This is referred to as a single observation. A sequence of such vectors, or observations, therefore corresponds to the results on consecutive game days. The idea behind the RNN models is to take a sequence of these observations and to predict what the next will be (i.e. predicting the results of the next game day).

Football.csv [3] was used for the raw data database, which was preprocessed to produce the 20D vectors. The training sets were defined to have the sources be sequences ranging from lengths of 1 to 15 observations, with the target in each case the observation following the sequence. For this experiment, additional data was scraped from the database to increase the total number of seasons from 64 to 88. Since there are 38 game days per season, this corresponds to a total 3,344 observations.

The results of each game day were encoded in the 20D vector as the goal difference of each team. For example, if team A beat team B 5-3, the entry for team A would be 2 and team B would be -2. This encoding produces better results than a simple ternary encoding, e.g. win = 1, draw = 0, loss = -1, as the goal difference provides more information about and can be a better measure of the team's performance.

Accuracy

This encoding allows the accuracy to be evaluated in two ways:

1) Firstly, the standard accuracy measure, where if the model predicts a win, draw or loss accurately, it is considered a correct prediction; the accuracy is therefore the percentage of correct predictions

2) Secondly, the euclidean distance between the prediction and target vectors, which is a measure of the average difference between the predicted goal differences and the true goal differences; this has not been previously used other models so it is not useful for making comparisons, however it is nevertheless an interesting measure of the model quality

Architectures

RNNs were tested both with and without an attention mechanism. The architecture without attention used only the hidden state of the last input observation to make a prediction. The attention mechanism is an extremely simplified and modified implementation of the global one used in neural machine translation, wherein all hidden states are used to make a prediction instead of simply the last input. Since in this model, the decoder has no meaningful hidden states of its own, the conventional alignment vector would have no meaning, so the context vector is simply a concatenation of the hidden states.

The hyperparameters tested were the number of layers, number of hidden cells, and type of cells (LSTM or GRU). Another hyperparameter of sorts which was tested was the input sequence length of the observations. As explained above, the length of the input sequence of observations was varied from 1 to 15 to test which amount of prior observation would provide the best prediction for the future. Theoretically, it may seem unlikely that increasing the sequence length and hence providing more information could *decrease* the quality of predictions, however, practically longer inputs may lead to greater difficulty in training as the number training samples available decreases.

All architectures were implemented using Keras.

Training

Training was analyzed carefully using the training loss curves to optimize for the following hyperparameters: learning rate, batch size and number of epochs.

Hyperparameter Optimization

The table below summarizes the hyperparameters optimized for and the value tested:

Table 1: Hyperparameters

Number of Cells	Number of Layers	Cell Type	Attention?	Number of Epochs	Learning rate	Batch size	Sequence Length
200	1	LSTM	Yes	60	0.01	16	1 - 15
300	2	GRU	No	100	0.001	32	
More if beneficial				140	0.0001	64	

Theoretically, the optimum value can be found only by testing every possible permutation of hyperparameters. However since the number of permutations grows exponentially with the number of parameters, this is practically unfeasible. Instead in this experiment the accuracies were optimized along a smaller number of hyperparameter dimensions at a time to approximate the optimal solutions.

Experiment

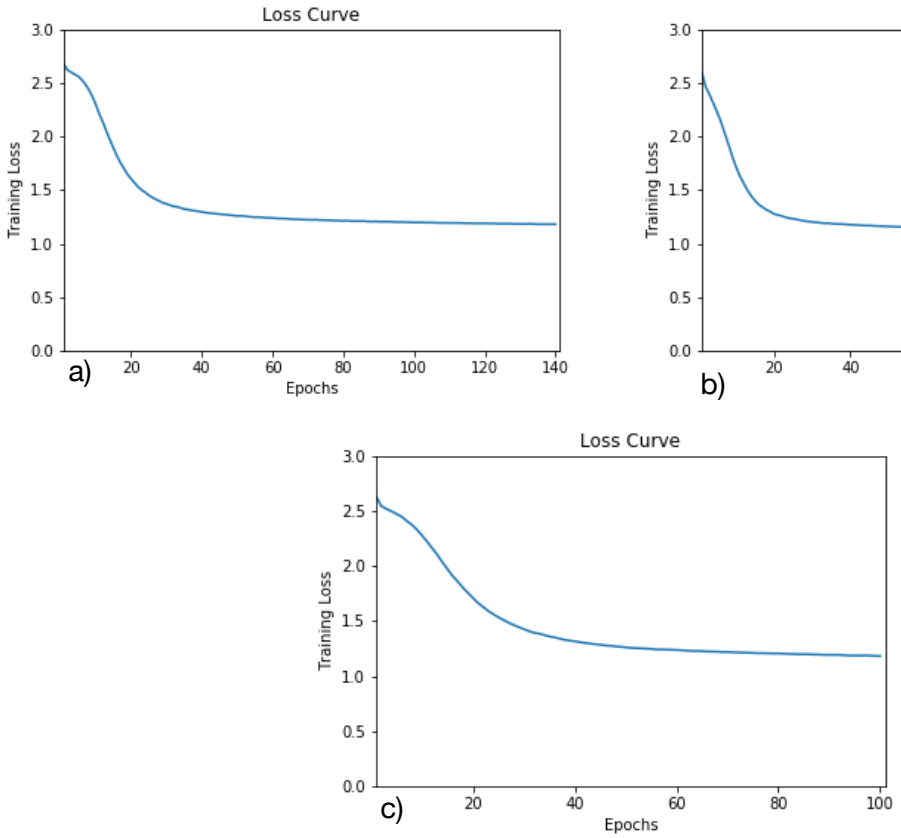
All the spreadsheets and plots are available on my Github **https://github.com/raghsthebest/SMSP2**, as well as my code which is in the "RNN_Non_Ternary.ipynb" notebook.

As explained above, the accuracy was optimized along specific hyperparameter dimensions instead of all simultaneously. Firstly, the training was optimized, by choosing the optimal batch size, learning rate and number of epochs. All 27 permutations above of the three were tested in this regard.

Number of Epochs

For a suitable learning rate and batch size (more on that below) increasing the number of epochs should improve the model's accuracy on the training data set as a more and more optimal solution is found. However, the two issues in continuously increasing the number of epochs are that: a) (trivially) the training time increases and b) the model may overfit on the training data. In the first case, we therefore need to analyze whether an increase in epochs is worth it in terms of the decrease in loss.

Figure 1: *Sample loss curves for 1 layer, batch size 32, 0.001 learning rate, LSTM a) 300 cells, no attention, input sequence length 4, b) 300 cells, attention, input sequence length 7, c) 200 cells, no attention, input sequence length 4*



As Figure 1 illustrates, for a variety of architectures, while the loss did continue to decrease as the number of iterations decreased, the decreases were extremely small after around 60 epochs. This means there is little to gain by increasing the number of epochs further and further. Generally, the conclusion was that 60 and 100 epochs, depending on the specific architecture, were good compromises in this regard, especially for testing purposes, although for the absolutely best results, increasing the epochs may be beneficial.

The second concern with too many epochs - overfitting - was tested by calculating the testing accuracies for 60, 100 and 140 epochs. The testing accuracy decreasing as the number of epochs increases is a sign of overfitting. However in every test with this model, the testing accuracy either increased or stayed level for increased iterations, so in overfitting was not an issue.

Batch Size and Learning Rate

It is necessary to consider these two hyperparameters together as they are closely related. Generally, small batch sizes and higher learning rates can lead to faster convergence but perhaps noisy and hence not the most optimal solutions. Vice versa, larger batch sizes and lower learning rates may lead to a better solution but slowly. Again, we desire the best compromise, with reasonably fast convergence and a solution of high accuracy.

Figure 2: *Sample loss curves for 300 LSTM cells, no attention, 1 layer, input sequence length 4; Left to Right - 0.0001, 0.001, 0.01 learning rate; Top to Bottom - 64, 32, 16 batch sizes.*

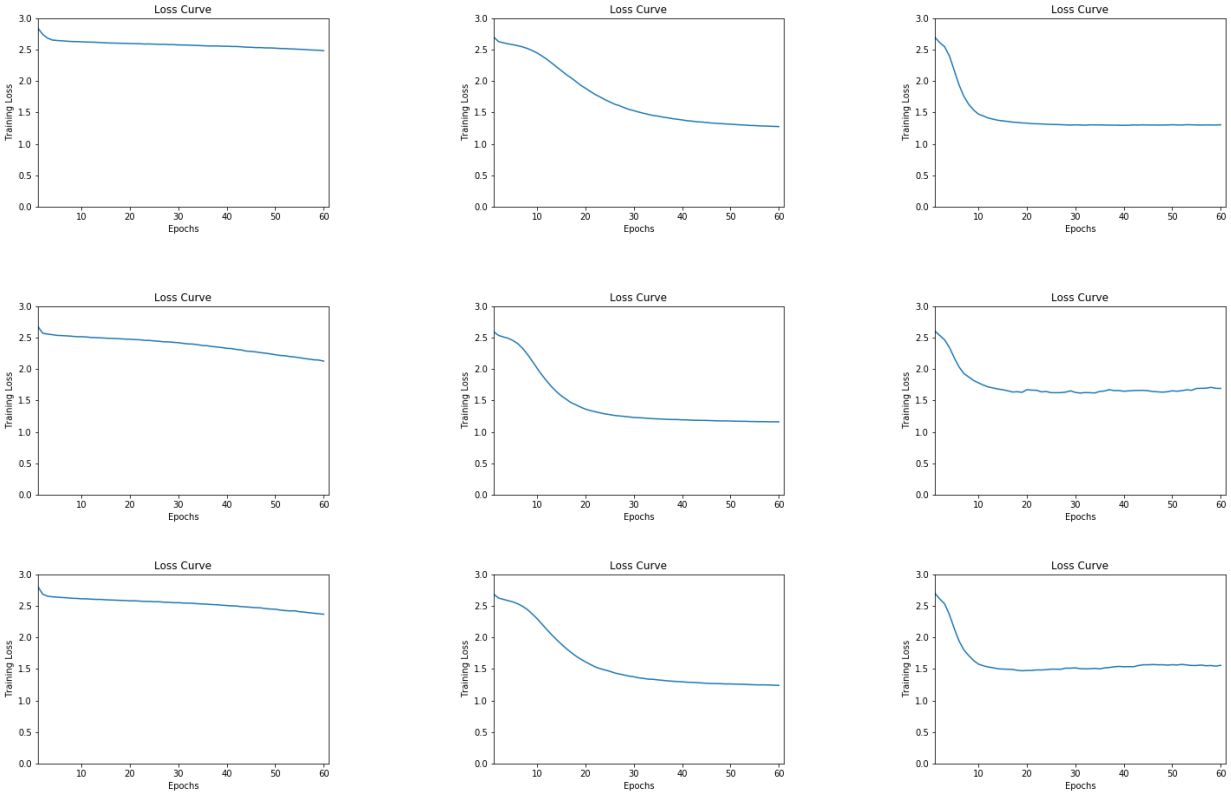


Figure 2 illustrates the effect of batch size and learning rate very clearly. As learning rate is increased, the model does indeed converge faster. However when the learning rate is too high, the solution can be worse (i.e. a higher loss). The loss can even increase as the iterations increase since a high learning rate can move the model too far to the other side of the minimum. The effect of the batch size is not as clear, but we can see that for the 0.0001 learning rate, the loss decreases faster as the batch size decreases, but conversely for the 0.01 learning rate, decreasing the batch size makes the solutions more noisy.

For the specific architecture in Figure 2, a batch size of 64 and learning rate of 0.01 seems to be the best compromise, with an extremely fast convergence and a stable, optimal solution. However, for some other architectures this combination led to unstable solutions so, to be safe (i.e. to avoid instability), the best combination was settled on to be a batch size of 32 and learning rate of 0.001

Next the architecture parameters were analyzed.

Sequence Length

Input sequences were tested with lengths ranging from 1 to 15.

Figure 3: *Accuracy vs sequence length, for 200 cells, no attention, 1 layer, 100 epochs*

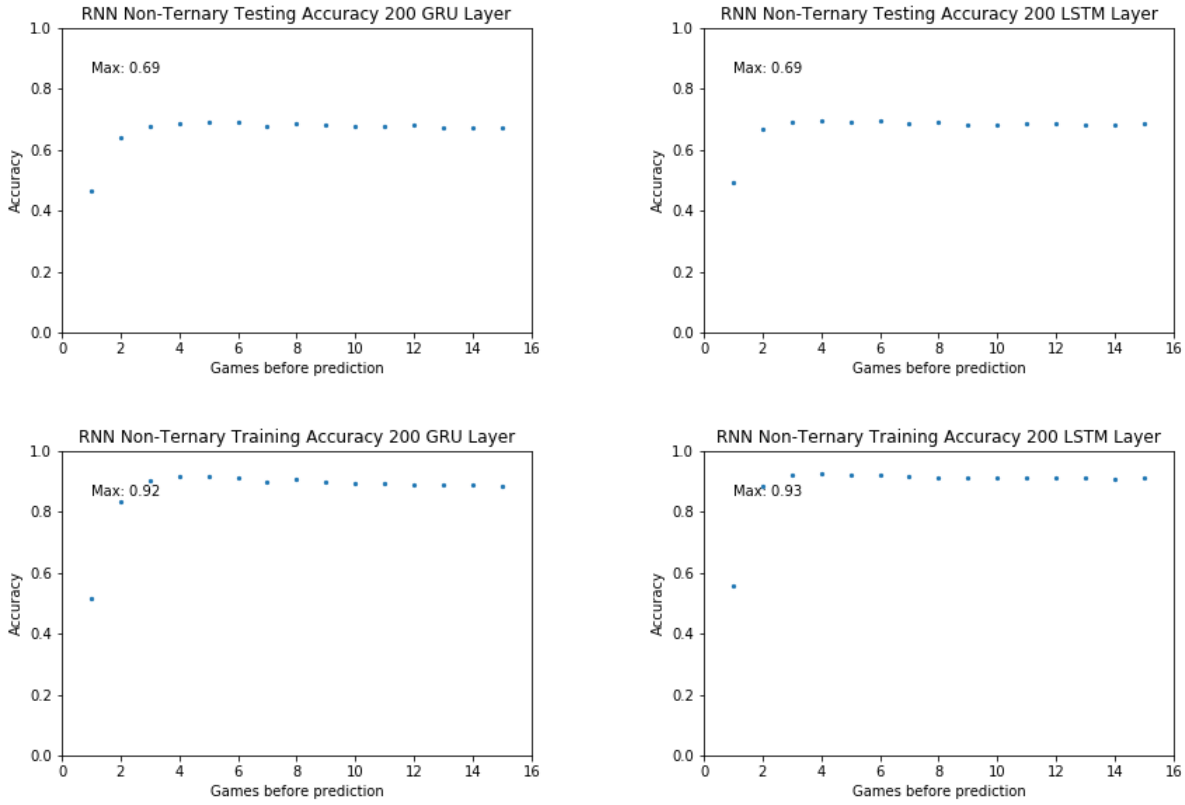


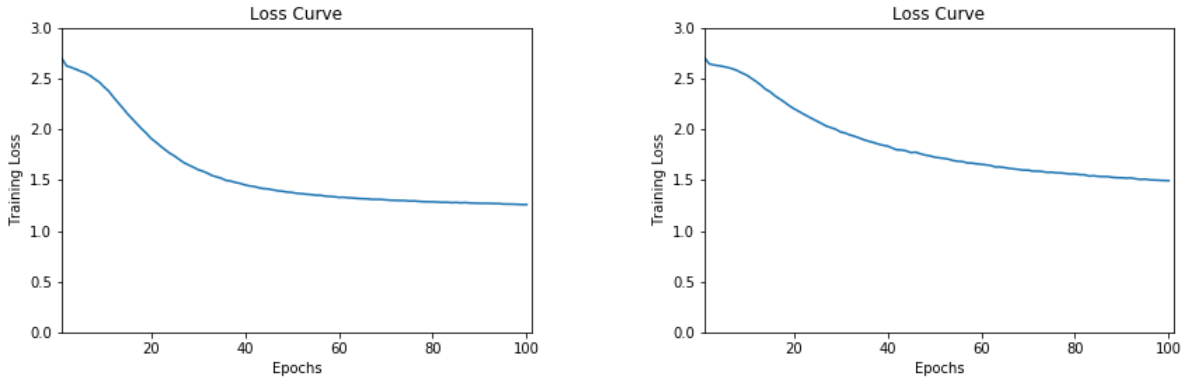
Figure 3 shows that the training and testing averages for two different architectures both start low but rapidly peak at around 3-5 games before the prediction before marginally decreasing as the sequence length increases. This shows that there clearly is a sequential correlation between teams' results since accuracy improves for longer sequences initially. However, the correlation is relatively short term, roughly of around four games prior, since using game days more than three behind does not improve the accuracy. Sequence lengths of 3, 4 and 5 were used henceforth as the optimal values.

Number of Layers

The total number of hidden cells was held constant here so that any differences in accuracy weren't simply the result of increasing the parameters. The number of layers were tested by using 1 layer, then seeing if there was a significant increase by using 2 instead, in which case more and more layers would then have been tested until the optimal was found. However, there was not a significant difference between the training or testing accuracies of a 1 or 2 layer network with the same parameters otherwise.

Moreover, as seen in Figure 4 (next page), the training loss converged much faster for a 1 layer network, therefore, one layer was taken to be the optimal amount.

Figure 4: *Training loss curves for 200 LSTM cells, no attention, batch size 32, learning rate 0.001, input sequence length of 4; Left: 1 layer, Right: 2 layers*



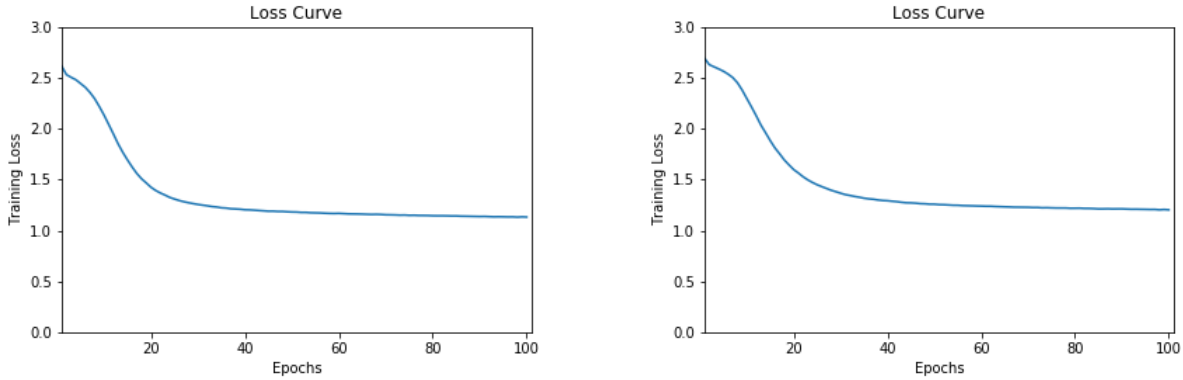
Number of Cells

Similarly as with the number of layers, 200 cells were tested with initially, then increased to 300. If a significance improvement had been observed, then the cell number would have increased further until the optimal was found. The difference between the best training and testing accuracies of a network with 200 and 300 cells (1 layer, no attention, 100 epochs) were around 3% and 1.5% respectively with both LSTM and GRU cells. This was a small enough difference to not motivate a further increase in cells.

Type of Cells

The highest training and testing accuracies for a 1 layer, 300 cell network without attention, trained for 100 epochs, were: using LSTM cells 95.5% and 70.5% respectively, and using GRU cells 93% and 70.5%. As seen in Figure 5, there is very little separating them in terms of training ease either. LSTM cells were chosen henceforth, because of the slightly higher training accuracy.

Figure 5: *Sample training loss curves for 300 cells, no attention, batch size 32, learning rate 0.001 Left: GRU, Right: LSTM*



Attention

Adding the simple attention mechanism gave maximum testing and training accuracies of 95% and 71% respectively. These were the same as that of a network without attention but same in every other regard. Consequently this particular attention mechanism does not appear to improve the model.

Conclusion

Using the optimal hyperparameters obtained from the above experiments, with a 1 layer, 300 cell, LSTM network, with an input sequence length of 3 games, trained for 140 epochs with a batch size of 32 and learning rate of 0.001, training and testing accuracies of 97% and 71% respectively were achieved. These are significant improvements over the 65% and 54% accuracies achieved in Kansal [2] and are competitive with state of the art commercial models [4] despite using much less and far simpler data. This accuracy can only be improved by using an even larger dataset and incorporating more sophisticated features in the input such as player performances, new player signings, injuries etc. which are used in commercial models.

The high accuracies also prove there exists a sequential correlation between a team's results in soccer. However the correlation is relatively short-term because, as the data shows, there is no benefit in inputting results from more than 3 game days prior.

References

[1] Tan, Brandon. “Form in Soccer: Not Always a Winning Formula.” *Princeton Sports Analytics*, Princeton Sports Analytics, 20 Sept. 2017, princetonsportsanalytics.com/2016/06/12/form-in-soccer-not-always-a-winning-formula/.

[2] Kansal, R. “Predicting soccer results using sequential models” June 2018, https://github.com/raghsthebest/SMSP/blob/master/Final%20Report.pdf

[3] https://github.com/footballcsv

[4] https://botprediction.com/en/index