Football Prediction

David Monschein, Marius Dörner

Final Report for Praktikum Neuronale Netze WS18/19
Karlsruhe Institute of Technology

Abstract

foobar

1. Introduction

In the last few years, neural networks became increasingly more popular and were applied to a wide range of problems in different fields, e.g. in computer vision, natural language processing or speech recognition. In this project we studied the application of neural networks to a more unusual task, namely to predict the outcome of football matches. More precisely, given the data of two football teams we had to predict the outcome of a match between them, i.e. either predict the winner or predict if the game ends in a draw.
For development we used the *European Soccer Database* [**?**] which contains more than 25.000 professional football matches and statistics for 299 teams and their associated players collected over multiple seasons from various European soccer leagues like the German Bundesliga, the English Premier League and the Spanish Primera División.
For this project, we developed two neural network models: A feed-forward model and a recurrent neural network (RNN) model. We compare their performance against bookmaker predictions on the same matches.
This report is organized as follows: Section 3 describes the structure of our dataset, the features we used as input for our networks and our data loading procedure. Section 4 introduces our models and their training procedures. Section 5 describes our experimental setup and presents our results. We conclude our work in Section **??**.

2. Related Work

3. Data

4. Method

We developed two models that are based on the *Siamese Network* approach [2] which is extensively used for tasks that require to find similarities between two inputs, a recent example for this is *Facenet* [3] a face recognition model. In our case we are given the data of two football teams and need to predict the outcome of a match between them, basically a comparison task. Teams are represented as a matrix $T \in \mathbb{R}^{11 \times m}$ where the row T_i contains the attributes of the i-th player of a given team. We denote the *away team* with T^a and the *home team* with T^h. Note that we only consider the eleven starting players for each team as we don't have data for substitute players. A description of the player attributes is given in Section 3.

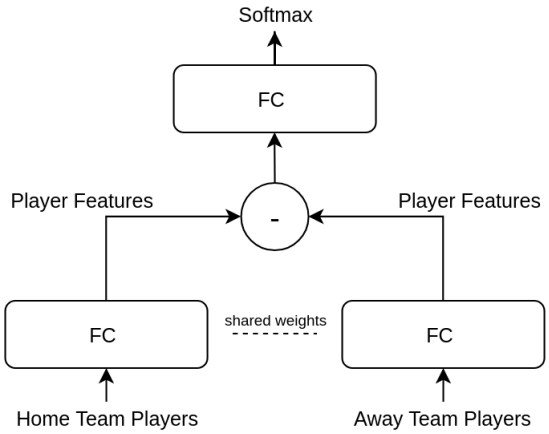

Figure 1: First model for match outcome prediction.

4.1. Fully Connected Model

Our first model consists of two fully connected neural networks as depicted in Figure 1. The first network learns an embedding $f(T) \in \mathbb{R}^d$ for the team data. We use this network for both teams during a forward pass, which means that the weights for the inputs are shared during the backwards pass. Given the features $f(T^h)$ and $f(T^a)$ for the home and away team we use the difference $f(T^h) - f(T^a)$ as the input for the second network, which predicts the probabilities p_h, p_d, p_a for home win, draw and away win using the Softmax activation function in the output layer. For the inner layers we use the ReLU (rectified linear unit) activation function. We also use *Dropout* [4] layers with probability 0.1 in both networks. The layers for the team embedding are depicted in Table 1 and the layers for the prediction network are depicted in Table 2.

4.2. Match Histories

Our first model only considered data from a single game but football teams play on a weekly basis during a season and performance over the last couple of games may be predic-

Layer	Input Size	Output Size	Activation
linear1	$11 \times 35 = 385$	128	ReLU
linear2	128	128	ReLU
dropout1	128	128	

Table 1: Feature layers for the FC model.

Layer	Input Size	Output Size	Activation
linear1	128	128	ReLU
linear2	128	128	ReLU
dropout1	128	128	
linear2	128	3	Softmax

Table 2: Prediction layers for the FC model.

tive of performance in future matches because of influences that are hard to measure numerically e.g. how competent a teams coach is in picking strategies or how good a team fits together on an interpersonal basis. For our second model we want to consider this additional information for both teams during the prediction but we first need to extract this from the dataset.

For the performance of a team we consider the goal differences of past matches from the teams point of view, e.g. if Team A wins 3 to 1 against Team B then the goal difference from Team As' point of view is 2 whereas the difference from Team Bs' point of view is -2. Thus the goal difference of a draw is 0 for both teams. Given a team and a date we call the series of goal differences of this teams past matches in this season up to the given date its *match history*. For example a team that played three matches with outcomes 3 to 1, 2 to 3 and 0 to 0 has the match history $(2, -1, 0)$. Note that we only consider games that were played in the same season, we don't go over seasons boundaries as professional football teams often change considerably during the off season (buying and selling players, new coaches etc.). We also want to emphasise here that match histories have different lengths depending on their end date, e.g. at the first matchday of a new season no team has a history yet while on the last matchday each team has a full season worth of games in its history (in most leagues around 30 games). Thus the length of the match history is variable during the seasons and we need to consider that while building our model.

We can construct match histories efficiently using only one pass over our dataset if all matches in a season are ordered by their match dates (ordering between seasons does not matter). We iterate over the matches by ascending date and create a buffer data structure, e.g. a Python deque, for each 'new' team we encounter, this buffer holds all the row indices of the past matches of its team in the dataset. For each match entry we add the current buffer content of both teams as a new column. Basically, each match now contains an list of dataset indices that reference the past matches (other rows in the dataset) of both teams. We implemented this using the PyTorch dataset class but this could also be implemented with other frameworks, for example using Pandas DataFrames.

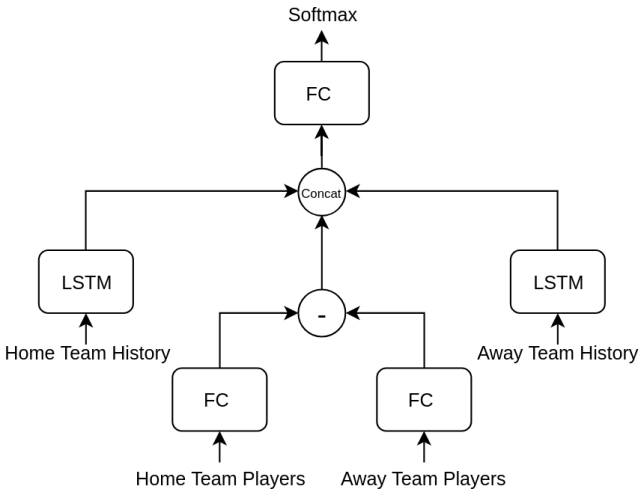

Figure 2: Second model for match outcome prediction.

4.3. RNN Model

For our second model we want to use the match histories as additional information. As we already described that match histories have different lengths at different match dates a recurrent neural networks seems fitting for use as a match history encoder. We again use the player data T^a and T^h in the same way as in the fully connected model, that is we use the network described in Table 1 to learn embeddings $f(T^a)$, $f(T^h)$ for the away and home team and input the difference $f(T^h) - f(T^a)$ into the prediction network. For the recurrent neural network we use a LSTM [5] with one hidden layer with 32 neurons. As a goal difference is a single number we use a Linear Layer with shape 1×32 and Tanh activation before the LSTM. We feed the series of goal differences completely into the LSTM and use its last hidden state as an additional input for our prediction network. Note that there are matches for which the match history is empty, i.e. at the start of a season, in these cases we don't use the LSTM at all and only use the predictions from the fully connected model. Similar to the fully connected network we use same LSTM (but reinitialize its hidden state after the first team) to encode the match histories of the teams which means that its weights are also shared for both inputs during training.

The input of the prediction network is the difference between the team embeddings $f(T^h) - f(T^a)$ concatenated with the last hidden states of the LSTMs for both match histories. The prediction network is the same as in the first model (Table 2) but with a larger input size (192 instead of 128) in its first layer to account for the additional inputs.

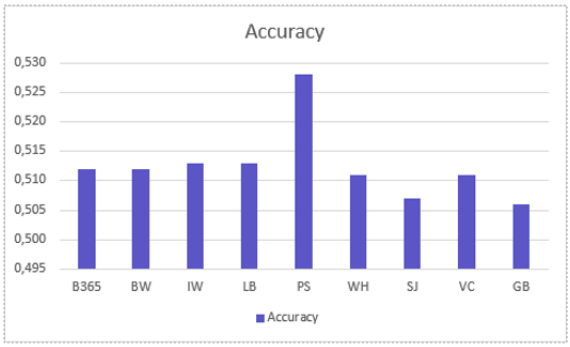

Figure 3: Prediction accuracy for different bookmakers in our dataset as evaluated on the test set. Average accuracy for bookmakers is 0.513.

5. Experiments

5.1. Setup

We implemented our models using the PyTorch framework [6] and trained them on Googles *Colab*[1] service which provides free access to a Nvidia K80 GPU. Our training data was found on Kaggle as described in Section 3.

5.2. Bookmaker Predictions as Baseline

For nearly every match in our dataset *odds* from various bookmakers, online services where people can bet on the outcome of football matches, were included. Bookmakers denote odds $o_h, o_a, o_d \in \mathbb{R}_{\geq 1}$ for each possible outcome, i.e. home win o_h, away win o_a and draw o_d which can basically be seen as multipliers for the returns on successful bets, e.g. betting 10 euro on an outcome with 1.5 odds returns 15 euros on a win. To make a profit or break even a bookmaker needs to set the odds in a way that reflects the outcome of a match, e.g. the most likely outcome should have the lowest odds and thus the lowest payout from the bookmakers point of view and so on. Therefore the odds actually represent the bookmakers predictions w.r.t. the match outcome and in theory there is even an inverse relationship between probabilities and odds, $p_x = 1/o_x$, but in practice bookmakers actually decrease the odds to add a profit margin for themselves, a process called *fixing the odds*. A more thorough review of bookmaker mathematics is out of the scope for this document, even though we can still assume that the odds in our database reflect which outcome the bookmakers believed to be most likely, as they would probably still assure that this one is the most "unattractive" to bet on.
For these reasons we use the predictions given by the odds as baseline to compare our models against. We evaluated their predictions on the matches in our test set using an ensemble of nine different bookmakers. Their results are shown in Figure 3, the mean prediction accuracy is 0.513.

[1] https://research.google.com/colaboratory

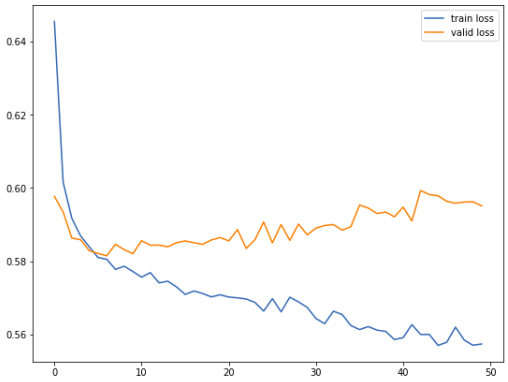

Figure 4: Training loss for the fully connected model.

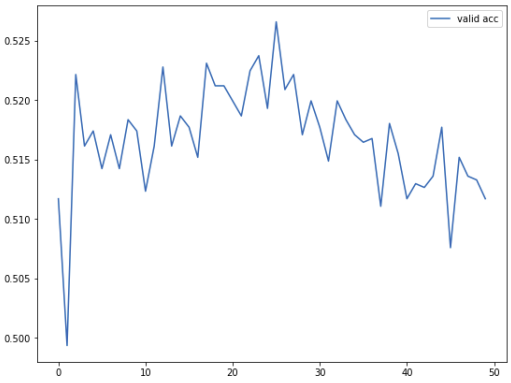

Figure 5: Validation accuracy for the fully connected model.

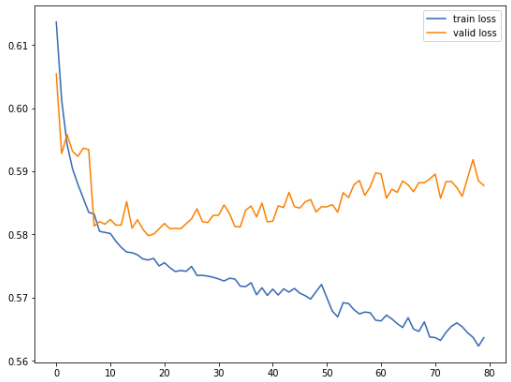

Figure 6: Training loss for the recurrent model.

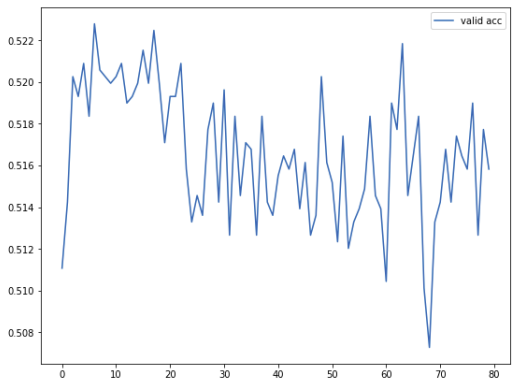

Figure 7: Validation accuracy for the recurrent model.

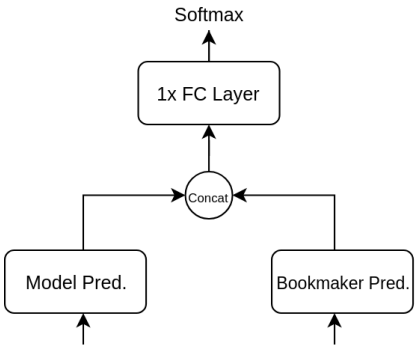

Figure 8: Refining bookmaker odds with our model predictions.

5.3. Training

We train both of our models with the cross entropy loss function. Our fully connected model was trained with the Adam [7] optimizer with a learning rate of 0.0025 and a batch size of 1024. The training loss and validation accuracies for this models are shown in Figure 4 and 5. We see that this models tends to overfit quickly even with small learning rates so we only trained it for five epochs for our evaluation.
For our recurrent model we only batch sizes of 32 matches as the variable length match histories we consider here have to either be padded or cut accordingly so that each match in the mini batch has histories of the same length. As padding is problematic here, e.g. padding with zeros would mean to add additional draws to the match history, we cut them on the same length for each match, so small batch sizes reduce the amount of cutting needed. The model was trained using the PyTorchs' SGD optimizer with a learning rate of 0.01 and 0.01 momentum. Loss curves and validation accuracies for the recurrent model are shown in Figures 6 and 7. This models also starts to overfit after some time, so for evaluation we stopped training after 30 epochs.

Model	**Accuracy**
Baseline (Bookmaker)	0.51
FC Net	0.49
Recurrent Net	0.51
Odds Refinement with FC Net	0.51
Odds Refinement with Recurrent Net	0.52

Table 3: Results for the different models as evaluated on the test set.

	pred home win	pred draw	pred away win
home win	0.33	0.00	0.12
draw	0.16	0.00	0.10
away win	0.13	0.00	0.16

Table 4: Confusion matrix for the fully connected model evaluated on the test set.

5.4. Evaluation

The results for our models on the test set can be found in Table 5.3, the fully connected network reaches 0.49 testing accuracy and the recurrent model reaches 0.51 accuracy. Also we tested two additional models that use bookmaker data as additional input where we fuse the predictions given by the bookmakers and the predictions given by our models to try to refine the bookmaker predictions (odds refinement in Table 5.3). We combine both predictions in a single layer with Softmax activations, as depicted in 8 and train it with the same hyperparameters as the original models.
One major problem we encountered was that the models basically never predict draws as shown in their confusion matrices in Table 4 and 5, we only saw draw predictions in heavily overfitted models but these perform significantly worse on the other labels. Note that draws were the least common result in our dataset and also bookmakers did not perform better on predicting them than our models, on the test set they also basically never predicted draws (on the whole data set they only predict draws a couple of times), Table 6 depicts the confusion matrix of the betting provider *bet365* as evaluated on our test set.

	pred home win	pred draw	pred away win
home win	0.38	0.00	0.06
draw	0.20	0.00	0.06
away win	0.18	0.00	0.13

Table 5: Confusion matrix for the recurrent model evaluated on the test set.

	pred home win	pred draw	pred away win
home win	0.39	0.00	0.06
draw	0.19	0.00	0.07
away win	0.16	0.00	0.13

Table 6: Confusion matrix for the bookmaker BW365 evaluated on the test set.

6. References

[1] Hugo Mathien, European Soccer Database (Version 10), Retrieved from https://www.kaggle.com/hugomathien/soccer

[2] Bromley, Jane and Guyon, Isabelle and LeCun, Yann and Säckinger, Eduard and Shah, Roopak, Signature verification using a "siamese" time delay neural network, Advances in neural information processing systems, 1994

[3] Schroff, Florian and Kalenichenko, Dmitry and Philbin, James, Facenet: A unified embedding for face recognition and clustering, Proceedings of the IEEE conference on computer vision and pattern recognition, 2015

[4] Nitish Srivastava and Geoffrey Hinton and Alex Krizhevsky and Ilya Sutskever and Ruslan Salakhutdinov, Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Journal of Machine Learning Research, 2014, Volume 15.

[5] Hochreiter, Sepp and Schmidhuber, Jürgen, Long Short-Term Memory, November 15, 1997, Neural Comput., Volume 9

[6] Paszke, Adam and Gross, Sam and Chintala, Soumith and Chanan, Gregory and Yang, Edward and DeVito, Zachary and Lin, Zeming and Desmaison, Alban and Antiga, Luca and Lerer, Adam, Automatic differentiation in PyTorch, 2017, NIPS-W

[7] Diederik P. Kingma and Jimmy Ba, Adam: A Method for Stochastic Optimization, 2014, CoRR abs/1412.6980, http://arxiv.org/abs/1412.6980, arXiv, 1412.6980