Football Prediction

David Monschein, Marius Dörner

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

Abstract

Predicting the outcome of football games is interesting to a lot of people, from football enthusiasts to people working in the betting industry. It is also an interesting research problem, due to its complexity and difficulty even humans can not precisely predict the outcome of soccer games. In the lab we used data about past matches, including teams and their players, to train different types of *Neural Networks*. We created a simple fully connected model and a more complex network which uses a *Long-Short-Term Memory* architecture. We evaluated their prediction accuracy and compared them to predictions from various betting services. We found that even relatively simple neural network models can be competitive with professional bookmakers.

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 and their players 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 a dataset that 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 fully connected (FC) model and a recurrent neural network (RNN) model and compared 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. Section 5 describes our experimental setup and presents our results. We conclude this paper in Section 6.

2. Related Work

There are a lot of existing machine learning approaches which deal with predicting football games.

The creator of the database used a Support Vector Machine (SVM) to predict the football matches. He stated that his approach is able to classify 53% of the matches correctly [1]. But it has to be said that there is no information about the evaluation process, so the accuracy of around 53% is not confirmed.

Andrew Carter presented an approach which uses a Deep Neural Network (DNN) based on Tensorflow [2]. He was able to achieve an accuracy of 51% and in addition he developed a betting strategy that can be used in combination with his DNN approach.

There are plenty of other machine learning projects which address the prediction of soccer games [3, 4]. The methods used within these projects vary greatly, they used a lot of different evaluation methods and metrics, so it is not possible to compare them directly.

3. Data

The data we used has been published on Kaggle and is called the "European Soccer Database"[1]. As the name suggests, it contains information about the outcome of past football games. The data is provided as an SQLite database with several tables. The next sections introduce the coarse structure of the database and explain the challenges encountered while dealing with the data.

3.1. Structure

The SQLite database holds data about 11 different leagues. Summed up, it consists of around 26000 matches of 299 teams. For each match it contains all players that started in this match, for both the home team and the away team. Furthermore there are attributes for all teams, which quantify the properties of a certain team. An example for such an attribute is the overall rating of a team, it is a value between 0 and 100 and approximates how good a team is. Similar to the team attributes the database contains information about 36 different attributes for each player (e.g. rating, shot power, ...). The team attributes and the player attributes are documented at different timestamps, so the database respects teams and/or

[1] https://www.kaggle.com/hugomathien/soccer/

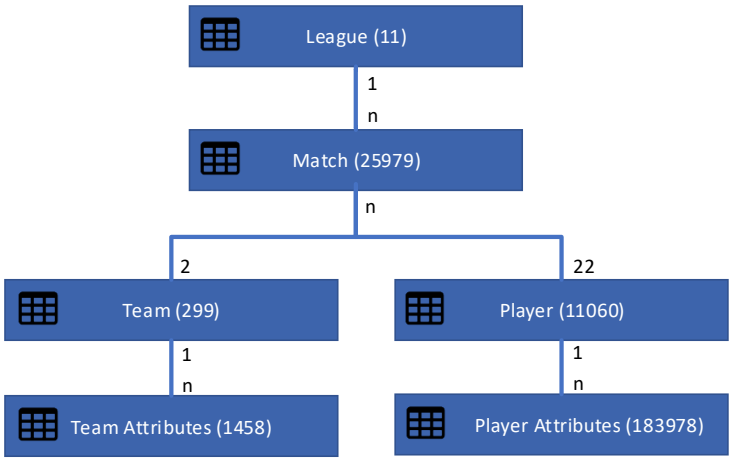

Figure 1: Tables and their relations in the used SQLite database

players which get better or worse over time. During the preprocessing we select the attribute sets with the smallest time difference to the considered match. It is important that only attributes are used which are known before the match begins. The attributes were extracted from the video game FIFA[2]. Figure 1 shows a simplified view of the tables in the SQLite database.

3.2. Challenges

We encountered two problems during the preprocess tasks. The first problem is related to a lot of missing attributes and obviously wrong values. The second problem is caused by unbalanced data. The home team statistically wins in 44% of all cases, therefore the database contains more matches that are labelled with a home win in comparison to draws and away wins.

3.2.1. Incomplete Data

The three major challenges, due to inaccurate and incomplete data, were:

1. Entirely missing player IDs → The only possible solution to deal with this is ignoring the matches because the given data is not sufficient to predict a football game reasonably.

2. A small subset of player IDs are missing → We applied different strategies. The ones that worked the best were inserting default values and using aggregated values for the player attributes (min, max, avg, deviation, ...).

3. Missing player attributes → The problem is similar to the missing player IDs for a match. The possible solutions are the same, aggregations and inserting default values worked the best.

[2]https://www.easports.com/de/fifa

We tried different approaches for dealing with this inaccuracies and selected the ones that worked the best. For missing player IDs we introduced a "default" player with attributes that are equal to the average of all other players in the team. With missing player attributes we dealt in the same way, we inferred default values.

3.2.2. Unbalanced Data

As already mentioned, the home team wins statistically about 44% of all matches. The away team wins 30% of the matches and the remaining 26% are draws. We do not want to introduce a bias so it was necessary to balance the data in the preprocessing phase.
We used **under sampling** to realize this. That means we removed around 14% of the home wins to balance the data. However, in the evaluation phase we recognized that the impact of balancing the data is negligible.

3.3. Preprocessing

Querying the needed the data from the SQLite database on every startup was very slow and also introduced the overhead of preprocessing prior to the training. Therefore we decided to query the needed data once, preprocess it and write it back to a CSV file. The main advantage of this approach is that we do not need to run queries against the database before training, which makes the start up significantly faster.
We split up the data in 80% Training and 20% Test. The Training data was seperated in 85% pure Training data and 15% Validation data. We implemented a dataset which conforms to the PyTorch Dataset structure that allows us to use existing features of the PyTorch framework (e.g. batching).

4. Method

We developed two models that are roughly based on the *Siamese Network* approach [5] which is extensively used for tasks that require to find similarities between two inputs, a re-

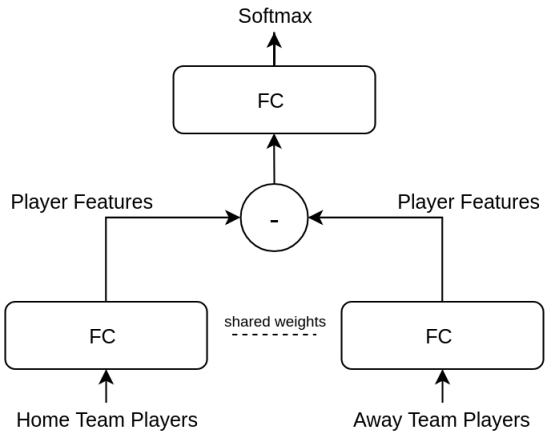

Figure 2: First model for match outcome prediction.

cent example for this is *Facenet* [6] a face recognition model. The defining characteristic of Siamese models is that a single network learns to embed two inputs consecutively into a feature space while optimizing a distance function, often minimizing the distance between similar and maximizing the distance between dissimilar inputs. In our case we are also given two inputs with varying degrees of similarity between them but instead of computing a distance we need make a classification decision. Therefore we developed models with "two stages", first we embed the input in a way similar to Siamese networks and then we use these embeddings to make a classification decision.

In the following we represent teams as matrices $T \in \mathbb{R}^{11\times m}$ where the row T_i contains the attributes of the i-th player of this team. Player attributes are as described in Section 3. In football we distinguish between the *home team*, the team at whose venue the match takes place and the *away team*. We denote the former with T^h and the latter with T^a. Note that we only consider the eleven starting players for each team as we don't have data for substitute players.

4.1. Fully Connected Model

Our first model consists of two fully connected neural networks as depicted in Figure 2. 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 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 its output layer. For the inner layers we use the ReLU (rectified linear unit) activation function. We also use *Dropout* [7] 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.

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.

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 predictive 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 from both points of view. 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 season 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 and we need to consider that while developing our model.

We can construct 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 Queue, 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

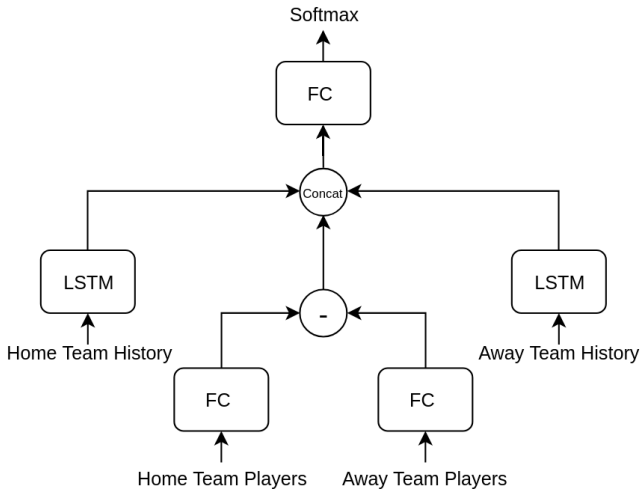

Figure 3: Second model for match outcome prediction.

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.

4.3. RNN Model

For our second model we want to use the match histories as additional information. As we already described that they have different lengths at different match dates a recurrent neural networks seems fitting here. We again use the player data T^a and T^h in the same way as in the first 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 [8] with one hidden layer with 32 neurons. As a goal difference is a single number we input in a Linear Layer with shape 1×32 and Tanh activation and forward this layers output into the LSTM. We feed the series of goal differences entirely 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 recurrent network at all and predict outcomes using only the team embeddings, same as in our first model.
Similar to the fully connected network we use same LSTM to encode the match histories of both teams (of course we reinitialize its hidden state for both 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 LSTM for both match histories (Fig. 3). The prediction network is the same as in the first model (Table 2) but with a larger input size (192 instead of 128) to account for the additional inputs.

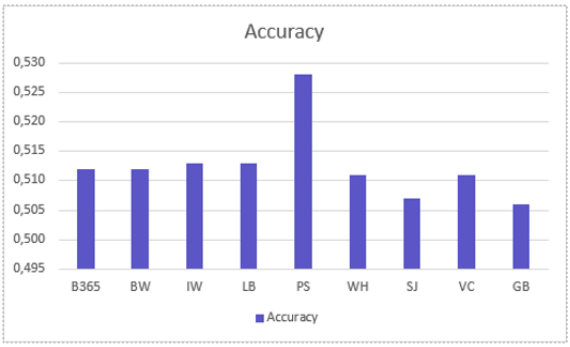

Figure 4: Prediction accuracy for different bookmakers in our dataset as evaluated on the test set. Average accuracy for bookmakers is 0.513.

5. Experiments

5.1. Setup

We implemented our models using the PyTorch framework [9] and trained them on Googles *Colab*[3] 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 report, even though we can still assume that the odds in our database reflect the outcomes the bookmakers believed to be most likely, as profit margins are often relatively low [10]. For these reasons we use the predictions given by the odds as baseline to compare our models against. We evaluated them on the matches in our test set for nine different bookmakers and measured their accuracies. Results are shown in Figure 4, the mean prediction accuracy is 0.513.

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

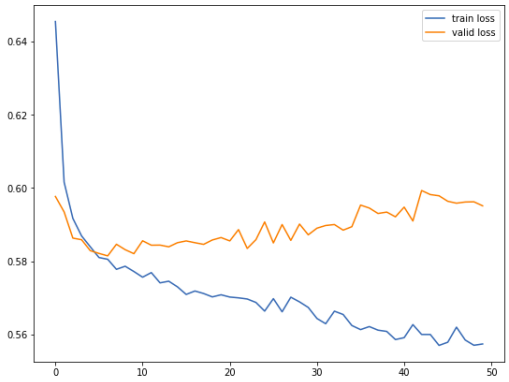

Figure 5: Training loss for the fully connected model.

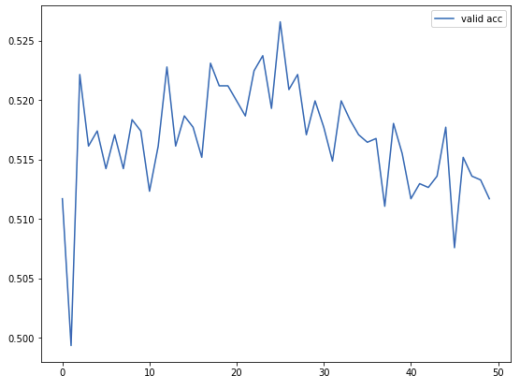

Figure 6: Validation accuracy for the fully connected model.

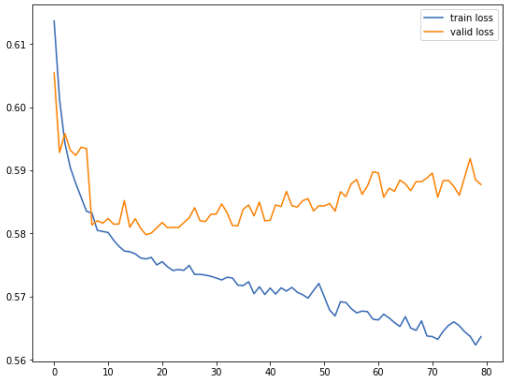

Figure 7: Training loss for the recurrent model.

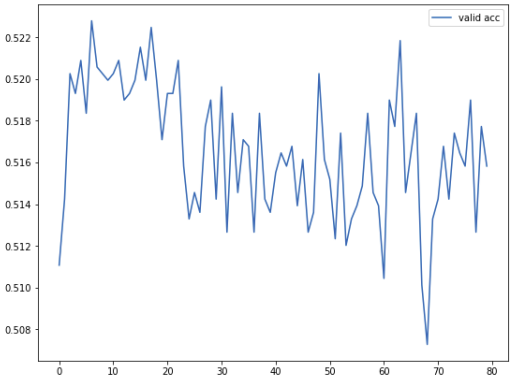

Figure 8: Validation accuracy for the recurrent model.

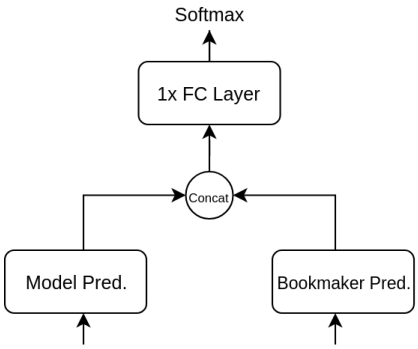

Figure 9: Refining bookmaker odds with our model predictions.

5.3. Training

We train both of our models with the cross entropy loss function. Our fully connected model was trained with the Adam [11] optimizer with a learning rate of 0.0025 and a batch size of 1024. The training loss and validation accuracies are shown in Figure 5 and 6. We see that this model tends to overfit quickly even with small learning rates so we only trained it for five epochs for 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 7 and 8. 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 9 and train it with the same hyper-parameters 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. Conclusion

Predicting football games is a very popular and challenging task. Different authors and practitioners already applied a broad range of different machine learning algorithms and techniques to accurately predict the outcome of upcoming football games. Even for humans it is very hard to predict whether the home team wins, the away team wins or the game ends with a draw. The accuracy of humans and machine learning approaches show that there are many more factors which have an impact on the outcome of a football game. Maybe in-depth data-collection and data-engineering can improve the prediction accuracy, sometimes it might even be *luck* if a team wins or loses.
It is indisputable that there will be a lot of contributions in the context of predicting sport results in the future. A lot of people are fascinated by sports and the betting industry always strives for higher profits. For that reason there will always be interest in this topic. It remains to be seen if new machine learning approaches are able to boost the performance of computer-based predictions further and if they are able to beat human experts consistently.

7. References

[1] H. Mathien. The ultimate Soccer database for data analysis and machine learning. Accessed on 26. February 2019. [Online]. Available: https://www.kaggle.com/hugomathien/soccer/home

[2] A. Carter. A Beginners Guide to Beating the Bookmakers with TensorFlow. Accessed on 26. February 2019. [Online]. Available: https://github.com/AndrewCarterUK/football-predictor

[3] R. Bunker and F. Thabtah, "A machine learning framework for sport result prediction," *Applied Computing and Informatics*, vol. 15, 09 2017.

[4] A. C. Constantinou, "Dolores: a model that predicts football match outcomes from all over the world," *Machine Learning*, vol. 108, no. 1, pp. 49–75, Jan 2019. [Online]. Available: https://doi.org/10.1007/s10994-018-5703-7

[5] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, "Signature verification using a "siamese" time delay neural network," in *Proceedings of the 6th International Conference on Neural Information Processing Systems*, ser. NIPS'93. San

Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993, pp. 737–744. [Online]. Available: http://dl.acm.org/citation.cfm?id=2987189.2987282

[6] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," *CoRR*, vol. abs/1503.03832, 2015. [Online]. Available: http://arxiv.org/abs/1503.03832

[7] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, Jan. 2014. [Online]. Available: http://dl.acm.org/citation.cfm?id=2627435.2670313

[8] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735

[9] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *NIPS-W*, 2017.

[10] P. Marek, "Bookmakers' efficiency in english football leagues," 09 2018.

[11] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980