CHALMERS
UNIVERSITY OF TECHNOLOGY

Football Match Prediction using Deep Learning

Recurrent Neural Network Applications

Master's Thesis in Computer Science – algorithms, languages and logic

DANIEL PETTERSSON
ROBERT NYQUIST

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2017
EX031/2017

Master's Thesis EX031/2017

Football Match Prediction using Deep Learning

Recurrent Neural Network Applications

DANIEL PETTERSSON
ROBERT NYQUIST

Supervisor and Examiner:
Professor Irene Yu-Hua Gu, Department of Electrical Engineering

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2017

Football Match Prediction using Deep Learning
Recurrent Neural Network Applications
DANIEL PETTERSSON
ROBERT NYQUIST

Supervisor and Examiner:
Professor Irene Yu-Hua Gu, Department of Electrical Engineering

Master's Thesis EX031/2017
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2017

Football Match Predicition using Deep Learning
Recurrent Neural Network Applications
DANIEL PETTERSSON
ROBERT NYQUIST
Department of Electrical Engineering
Chalmers University of Technology

Abstract

In this thesis, the deep learning method Recurrent Neural Networks (RNNs) has been investigated for predicting the outcomes of football matches. The dataset consists of previous recorded matches from multiple seasons of leagues and tournaments from 63 different countries and 3 tournaments that include multiple countries.

In the thesis work, we have studied several different ways of forming up input data sequences, as well as different LSTM architectures of RNNs that may lead to effective prediction, along with LSTM hyper-parameter tuning and testing. Extensive tests have been conducted through many case studies for the prediction and classification of football match winners.

Using the proposed LSTM architectures, we show that the classification accuracy of the football outcome is 98.63% for many-to-one strategy, and 88.68% for many-to-many strategy. The prediction accuracy starts from 33.35% for many-to-one and 43.96% for many-to-many, and is increasing when more information about a match from longer time duration of data sequence is fed to the network. Using the full time data sequence, the RNN accuracy reached 98.63% for many-to-one, and 88.68% for many-to-many strategy.

Our test results have shown that deep learning may be used for successfully predicting the outcomes of football matches. For further increasing the performance of the prediction, prior information about each team, player and match would be desirable.

Keywords: Football, deep learning, machine learning, predictions, recurrent neural network, RNN, LSTM

Acknowledgements

We would firstly like to thank our supervisor Irene Yu-Hua Gu at the Department of Electrical Engineering at Chalmers University of Technology, where this thesis has been conducted. We would like to thank her for the help she has been giving throughout this work.

We would also like to express our thanks to Forza Football for providing us with data, equipment, and workspace.

We have grown both academically and personally from this experience and are very grateful for having had the opportunity to conduct this study.

Daniel Pettersson, Robert Nyquist, Gothenburg, June 2017

Contents

List of Figures

List of Tables

1 Introduction

There have been several tries at predicting sport games using data from the past, but humans are still superior at predicting sport outcomes. There are multiple commercial services which have sports analysis and prediction as their main business. They use “sophisticated software and statistical algorithms” to aid their data tracking, but at the core they still have experts analysing the games manually[1].

This work aims at predicting the outcome of football matches using deep learning and recurrent neural networks, RNNs.

1.1 Background

Accurate football prediction is very valuable for the small Gothenburg based company Football Addicts and their smartphone application Forza Football. This thesis is a collaboration with the company Football Addicts and the project has used data available from the company.

1.2 Goals

This thesis aims at predicting football matches using deep learning and RNNs. The goal is to have a prediction accuracy that can compete with betting companies, by letting the network focus on players performance.

1.3 Constraints

There are many deep learning methods that can be used to predict football matches, depending on, for example, the data available. Due to the type of data available for this project, RNN is the method chosen and no other methods will be tested.

The thesis only uses data from a fixed set of leagues. Players transferred from a league outside of the leagues to one league in the set will not have the historic data.

[1]Examples: `http://www.stats.com/football/` and `http://instatfootball.com/`

All these players will be treated the same, even though in real life they have different history.

Our project does not consider any kind of ranking of different leagues used to train the model. This means that the model might predict a draw or even a win for a top team in a lower ranked league, even though it is highly improbable. Matches with two teams from different leagues with a large rank difference occur rarely and should therefore not be a problem.

1.4 Problem Formulation

This report aims at investigating whether deep learning can be used to predict football matches by analyzing the following:

- Can RNNs be used for the purpose of match predictions?
- Is history about players important when predicting the outcome of a match?
- How long history is relevant for a player?

1.5 Disposition

The disposition of this report generally follows that of a standard technical report. Chapter 2 covers the background theory necessary to understand the conducted study. The proposed methods can be found in Chapter 3. The results along with the setup used for the study can be found in Chapter 4, where the result of each experiment is followed by a discussion. Finally in Chapter 5 the conclusion from the study is presented.

2

Background Theory

This chapter aims at describing the theory needed to understand the work conducted, as well as present related work that is useful for this thesis. It starts off with an introduction to machine learning, neural networks, and deep learning.

2.1 Machine Learning

Machine learning is a subfield to artificial intelligence which has increased in popularity over the last few years (especially neural networks and deep learning), both in research and in industries. In contrast to traditional rule-based artificial intelligence, where an algorithm is more or less a list of predefined static rules, machine learning tries to use data to learn to make predictions or decisions. Many problems make it unfeasible to construct or design rules to help find a solution due to the sheer amount of constraints or complexity of the problem, which is where machine learning can leverage data to learn a solution.

2.1.1 Neural Networks

Artificial neural networks (ANNs) are a computational approach that is based on the way a biological brain solves problems. The human brain is composed of nerve cells called *neurons* that are connected with each other by *axons*. ANNs are composed of multiple nodes, which imitate the biological neurons of a human brain. The neurons are connected by links, which imitate the biological axons, and they interact with each other. Each node takes input data, performs a simple operation, and passes the result to other nodes. Like a biological brain, an ANN is self-learning and can therefore excel in areas where the solution is difficult to express by a traditional programming approach.

The core of neural networks, neurons, is just a simple activation function that has multiple inputs and one output. The neuron can be seen as a composition of several other weighted neurons and the network can be described by the network function

$$f(x) = K\left(\sum_i w_i g_i(x)\right) \tag{2.1}$$

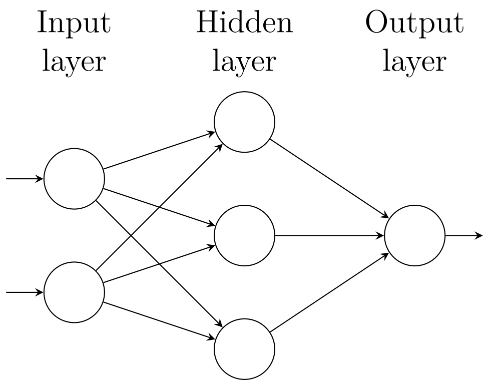


Figure 2.1: Example neural network, two input nodes, three hidden, one output.

where w_i are weights, g_i are other functions and K is the activation function e.g., the logistic function, hyperbolic tangent or rectifier ($K(x) = \max(0, x)$). A network consists of several layers of these neurons, where connections go from one layer to the other. The first layer is the input layer, the last layer is the output, and all the layers in between are hidden layers. A deep neural network can have several hundred hidden layers.

A neural network with at least one hidden layer with a finite number of neurons in that layer can approximate any continuous function. This is known as the universal approximation theorem, and is one reason to why one could believe that neural networks can be used for general artificial intelligence. It seems likely that being able to approximate functions is a very good property to possess when trying to learn how to behave.

2.1.2 Deep Learning

Deep learning is a technique that builds on deep neural networks (DNNs), a form of artificial neural network. The difference is that deep neural networks have multiple hidden layers. There have been big advancements in the field of deep learning during the last few years. Two of the most acclaimed papers are about DeepMind's network that plays Atari 2600 games [31] and their network that beats the world champion in the board game GO [32].

2.1.3 Recurrent Neural Networks

One key constraint or limitation of a normal neural network is that the input and output is of fixed length. If you input images they all need to have the same size, and the output, e.g., when doing classification is a probability of different fixed classes. The recurrent neural network, RNN, tries to address this issue of fixed size input/output. RNN introduces loops between events or steps allowing information to be used in a later stage, just like a memory. One can see a loop as a copy of the network with the same parameters that just sends the state to the next step, see

Figure 2.2. This enables a RNN to be used on sequences of input and output which can take previous information into account. It has been shown that this works very well for a number of situations like natural language processing, video classification, image classification, etc. [38, 3, 7, 43].

Given a sequence $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T)$, the RNN updates its recurrent hidden state $\mathbf{h}_t$ by

$$\mathbf{h}_t = \begin{cases} 0, & t = 0 \\ \Phi(\mathbf{h}_{t-1}, \mathbf{x}_t), & \text{otherwise} \end{cases} \tag{2.2}$$

where Φ is a nonlinear function. Traditionally the update for the recurrent hidden state, $\mathbf{h}_t$, is implemented as

$$\mathbf{h}_t = g(W\mathbf{x}_t + U\mathbf{h}_{t-1}), \tag{2.3}$$

where g is a smooth and bounded function such as a logistic sigmoid function, W and U are weight matrices.

One big problem with a straightforward RNN is the *vanishing* or *exploding* gradient. This can happen during training and the calculation of gradients for the backpropagation. The gradients are calculated using the chain rule which multiplies small numbers many times, making the error signal decrease with an exponential factor. The same thing can also happen when gradients are too large, which can result in the exploding gradient. This is just as bad because the network's early layers are unable to learn as the error cannot propagate properly [21]. Both problems can be avoided by using Long Short-Term Memory and Gated Recurrent Unit, discussed in the next 2 sections.

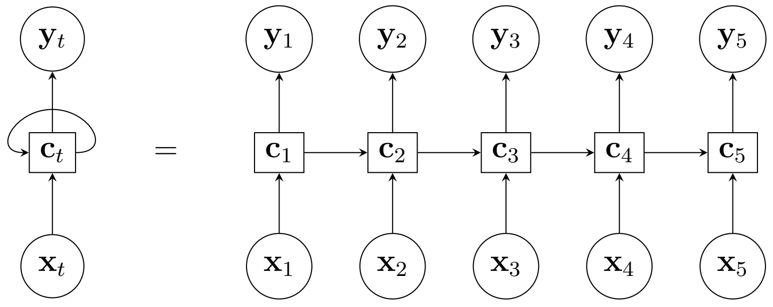

$\mathbf{y}_t$
$\mathbf{c}_t$
$\mathbf{x}_t$
=
$\mathbf{y}_1$ $\mathbf{y}_2$ $\mathbf{y}_3$ $\mathbf{y}_4$ $\mathbf{y}_5$
$\mathbf{c}_1$ $\mathbf{c}_2$ $\mathbf{c}_3$ $\mathbf{c}_4$ $\mathbf{c}_5$
$\mathbf{x}_1$ $\mathbf{x}_2$ $\mathbf{x}_3$ $\mathbf{x}_4$ $\mathbf{x}_5$

Figure 2.2: Example of a synced recurrent neural etwork, many-to-many. At each timestep, there is an output corresponding to the input at that time.

Figure 2.3 shows a RNN where only one output is used, classifying the entire sequence given instead of every timestep in a sequence.

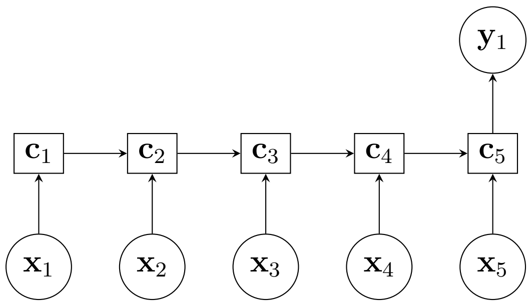


Figure 2.3: Many-to-one Recurrent Neural Network where a sequence input returns a fixed output.

2.1.3.1 Long Short-Term Memory

A Long Short-Term Memory (LSTM) unit is a recurrent network unit that is designed to remember values for either a long or a short duration of time [20] e.g., if the LSTM unit detects an important feature from an early input sequence, it carries this information over a long distance. This is significant for many applications, such as speech processing, music composition and time series prediction.

Since a LSTM does not use an activation function within its recurrent components the stored value is not iterativelty squashed over time. Each j-th LSTM unit maintains a memory c_t^j at time t [8]. The output h_t^j is computed by

$$h_t^j = o_t^j \tanh(c_t^j), \tag{2.4}$$

where o_t^j is an *output gate* that is computed by

$$o_t^j = \sigma(W_o \mathbf{x}_t + U_o \mathbf{h}_{t-1} + V_o \mathbf{c}_t)^j, \tag{2.5}$$

where σ is a logistic sigmoid function, V_o is a diagonal matrix, and W_o and U_o are weight matrices.

Memory cell c_t^j is updated by

$$c_t^j = f_t^j c_{t-1}^j + i_t^j \tilde{c}_t^j, \tag{2.6}$$

where the new memory content is

$$\tilde{c}_t^j = \tanh(W_c \mathbf{x}_t + U_c \mathbf{h}_{t-1})^j \tag{2.7}$$

and f_t^j is a *forget gate*. The forget gate modulates how much of the memory will be forgotten and an *input gate*, i_t^j decides the degree to which the new memory content is added to the memory cell. The gates are computed by

$$f_t^j = \sigma(W_f \mathbf{x}_t + U_f \mathbf{h}_{t-1} + V_f \mathbf{c}_{t-1})^j, \tag{2.8}$$

$$i_t^j = \sigma(W_i\mathbf{x}_t + U_i\mathbf{h}_{t-1} + V_i\mathbf{c}_{t-1})^j. \tag{2.9}$$

A LSTM unit is able to decide whether to keep the existing memory via the introduced gates. See Figure 2.4 for a graphical illustration of a LSTM unit and Figure 2.5 for an example of a multi-layer LSTM network.

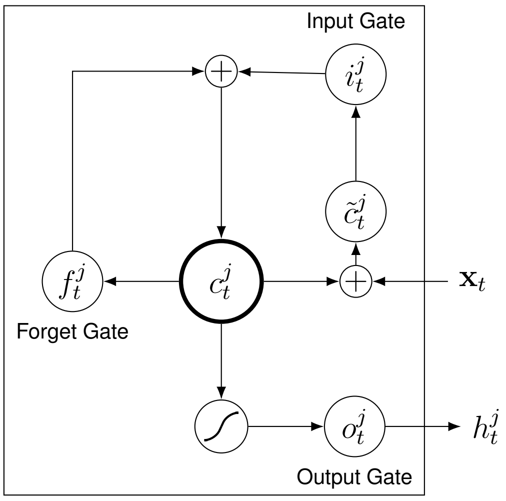


Figure 2.4: Illustration of a LSTM unit.

The following part will describe some different variants of LSTM.

- **LSTM with peepholes**

LSTM units with peepholes include the internal state when calculating the values of all gates [13]. The gates for the j-th LSTM with peephole at time step t are calculated by

$$o_t^j = \sigma(W_o\mathbf{x}_t + U_o\mathbf{h}_{t-1} + P_o\mathbf{c}_{t-1})^j \tag{2.10}$$

$$f_t^j = \sigma(W_f\mathbf{x}_t + U_f\mathbf{h}_{t-1} + P_f\mathbf{c}_{t-1})^j \tag{2.11}$$

$$i_t^j = \sigma(W_i\mathbf{x}_t + U_i\mathbf{h}_{t-1} + P_i\mathbf{c}_{t-1})^j \tag{2.12}$$

where P are matrices with weights that need to be learned by the network.

A LSTM with peephole has an improved ability to learn tasks that require precise timing and counting of the internal states [17].

- **Convolutional LSTM**

A drawback for LSTM is its usage of full connections in input-to-state and state-to-state transitions in which no spatial information is encoded when handling spatiotemporal data [42]. To overcome this problem, all the inputs $\mathbf{x}_t$, outputs $\mathbf{h}_t$,

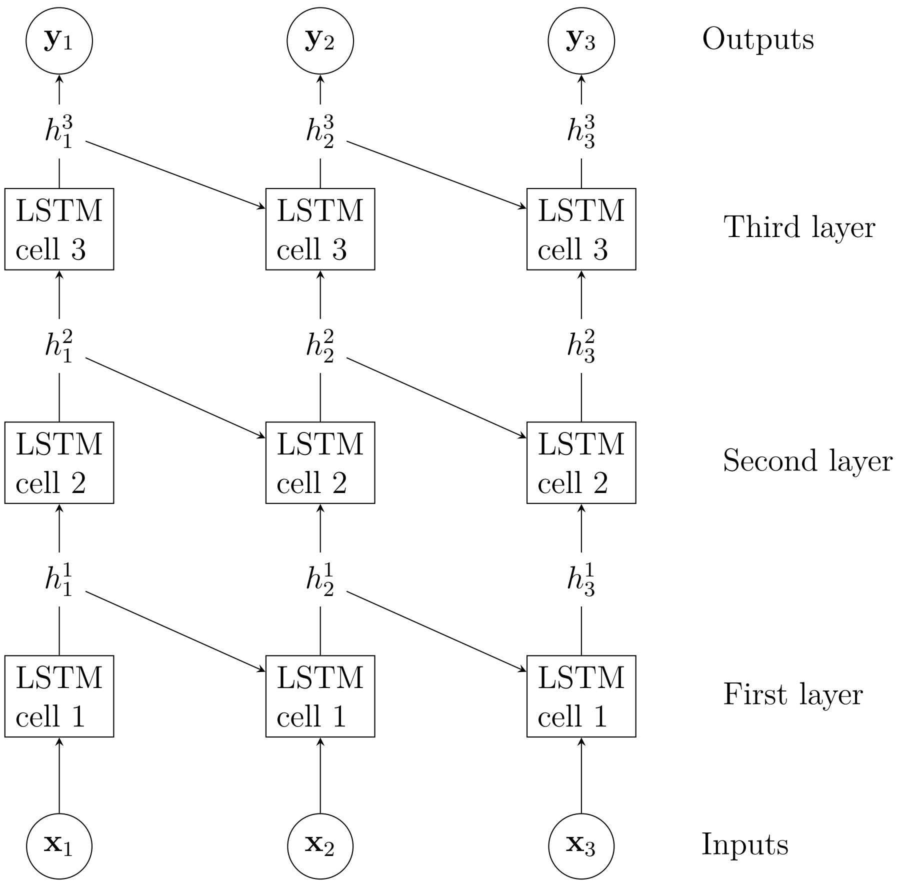


Figure 2.5: An example of a multi-layer LSTM network with 3 layers. $\mathbf{x}_t$ is the input at time t, h_t^j is the output of layer j at time t. The output from layer j at time t is fed to layer $j+1$ at time t. The output of layer j at time t is also passed on to the same layer j at time $t+1$.

memory cells $\mathbf{c}_t$ and gates $\mathbf{o_t}$, $\mathbf{f_t}$, $\mathbf{i_t}$ of the convolutional LSTM are 3D tensors whose last two dimensions are the spatial dimensions. The outputs, memory cells and gates for the j-th convolutional LSTM at time t are calculated by

$$h_t^j = o_t^j \circ \tanh(c_t^j) \tag{2.13}$$

$$c_t^j = f_t^j \circ c_{t-1}^j + i_t^j \circ \tanh(W_c * \mathbf{x}_t + U_c * h_{t-1}^j) \tag{2.14}$$

$$o_t^j = \sigma(W_o * \mathbf{x}_t + U_o * \mathbf{h}_{t-1} + V_o \circ \mathbf{c}_t)^j \tag{2.15}$$

$$f_t^j = \sigma(W_f * \mathbf{x}_t + U_f * \mathbf{h}_{t-1} + V_f \circ \mathbf{c}_t)^j \tag{2.16}$$

$$i_t^j = \sigma(W_i * \mathbf{x}_t + U_i * \mathbf{h}_{t-1} + V_i \circ \mathbf{c}_t)^j \tag{2.17}$$

where U, V and W are weight matrices, $*$ is the convolution operator [41] and $\circ$ is the entrywise product [22].

2.1.3.2 Gated Recurrent Unit

A gated recurrent unit, GRU, is designed to adaptivly reset or update its memory [8]. Unlike a LSTM, a GRU does not have a separate memory cell.

The activation h_t^j of GRU j at time t is a linear interpolation between the previous activation h_{t-1}^j and the candidate activation $\tilde{h}_t^j$ [8]. An *update gate* z_t^j decides how much the unit updates its activation:

$$h_t^j = (1 - z_t^j)h_{t-1}^j + z_t^j\tilde{h}_t^j \tag{2.18}$$

$$z_t^j = \sigma(W_z\mathbf{x}_t + U_z\mathbf{h_{t-1}})^j \tag{2.19}$$

The candidate activation $\tilde{h_t^j}$ is computed by

$$\tilde{h}_t^j = \tanh(W\mathbf{x}_t + U(\mathbf{r}_t \odot \mathbf{h}_{t-1}))^j, \tag{2.20}$$

where $\mathbf{r}_t$ is a set of *reset gates.* When r_t^j is off (r_t^j is close to 0), the reset gate makes the unit act as if it is reading the first symbol of an input sequence. This way it is allowed to forget the previously computed state.

Reset gate r_t^j is computed by

$$r_t^j = \sigma(W_r\mathbf{x}_t + Ur\mathbf{h}_{t-1})^j. \tag{2.21}$$

See Figure 2.6 for a graphical illustration.

As the forget and input gates are combined into a single update gate the GRU is simpler than a standard LSTM unit and therefore computationally more efficient.

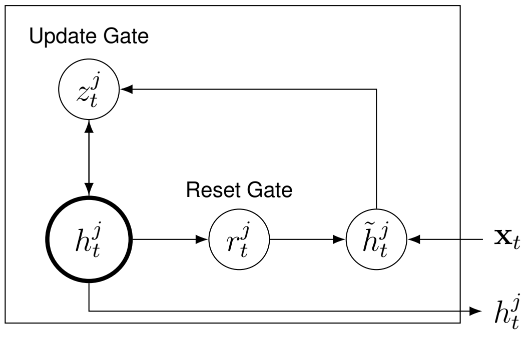
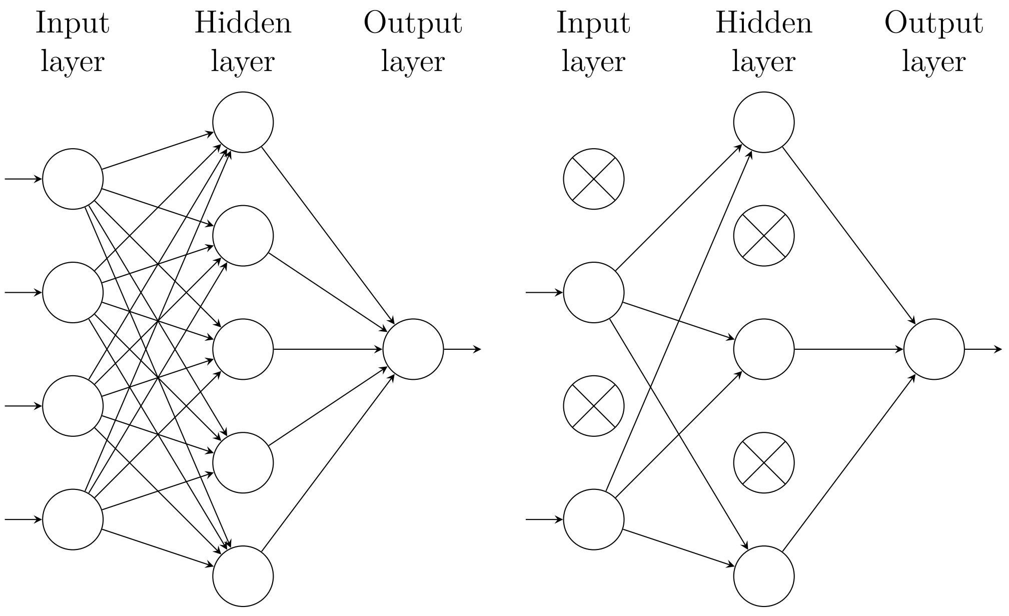


Figure 2.6: Illustration of a GRU.

2.1.4 Dropout

Neural networks with a large amount of parameters have a problem with overfitting [34]. A model that overfits on the training data will result in bad performance. By introducing dropout, the risk for overfitting decreases. The idea is to randomly drop units along with their connections during the training to prevent units from co-adapting too much. At training time each node has a probability p to be dropped out.



(a) Network without dropout applied. **(b)** Network with dropout applied.

Figure 2.7: Left: A standard neural net with 1 hidden layer. **Right:** An example of a thinned network produced by applying dropout to the network on the left.

Applying dropout to a neural networks can be seen as sampling a “thinned” network from the original. The thinned network consists of all nodes (with respective connection) that survived the dropout. Figure 2.7(b) show a sampled thinned network from the network in Figure 2.7(a). A neural network with n units can be seen as a collection of 2^n possible thinned neural networks. For each training stage a new thinned neural network is sampled and trained. So training a neural network with dropout can be seen as training a collection of 2^n thinned networks. Each network gets trained rarely or not at all.

2.1.5 Embeddings

Entities in the dataset can be modeled in a d-dimensional vector space, called the “embedding space” or distributed representation. Each entity i is assigned a vector $V_i \in R^d$ [5]. Within the embedding space there is a specific similarity measure that captures the relation between entities. One of the earliest use of distributed representation date back to 1988 due to Rumelhart, Hinton and Williams [40].

Embeddings can be used to fight the *curse of dimensionality* [4]. The curse of dimensionality is the phenomena that arises when working with data in high-dimensional space. Choosing attributes, and how many, that describes the entities is tricky. The solution is to let the network learn the distributed representation. The network will adjust the representations during training which enables it to capture information about the entities [30].

Representing entities in an embedding space has helped algorithms to achive better performance in various areas, such as statistical language modeling and various natural language processing (NLP) tasks [10, 15, 39, 33].

2.1.6 Softmax Classifier

The softmax function is a generalization of the logistic function that squashes a K-dimensional vector of arbitrary values to a K-dimensional vector of real values in range [0,1] that add up to 1 [6]. The output of the softmax function can then be used to represent a probability distribution over K different possible classes.

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \text{ for } i = 1 \dots K \tag{2.22}$$

The softmax function is used in various multiclass classification methods, such as multinomial logistic regression, multiclass linear discriminant analysis, naive Bayes classifiers, and artificial neural networks. In a neural network-based classifier the function in the final layer is often a softmax function.

2.1.7 Cross Entropy Error

The cross entropy can be used as an error measure for neural networks. It describes the entropy of a distribution $\mathbf{y}'$ with respect to another distribution $\mathbf{y}$ and measures how many bits you need to encode data on average [16].

$$H(\mathbf{y}', \mathbf{y}) = -\sum_i y'_i \log y_i \tag{2.23}$$

2.1.8 Adam Optimizer

Adam, derived from adaptive moment estimation, is an algorithm for first-order gradient-based optimization of stochastic objective functions that is well suited for problems that are large in terms of data and parameters [27]. Adam is also suited for problems with very noisy and sparse gradients. The method only requires first-order gradient.

Adam computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients. The magnitudes of parameter updates are invariant to rescaling the gradient, its stepsizes are approximately bounded by the stepsize hyperparameter, it does not require stationary objective, it works with a sparse gradient and naturally performs a form of step size annealing. It combines the advantages of *AdaGrad* [11], to deal with sparse gradients, and *RMSProp* [19], to deal with non-stationary objectives. It is proven to be well-suited for a wide range of non-convex optimization problems in the field machine learning.

2.1.9 Hardware

A deep learning algorithm can contain a large number of parameters. This may lead to training taking a long time and therefore the choice of hardware is important.

2.1.9.1 Central Processing Unit

Traditionally, mathematical computations have been done on the Central Processing Unit (CPU). This includes the computations done for training a neural network. Today's CPUs have multiple cores [12]. In order to optimize the run time, calculations that can be done separately should be done on separate cores.

2.1.9.2 Graphics Processing Unit

A Graphics Processing Unit (GPU) is a processor that is designed to execute computations in parallel, which is common in 3D graphics[35].

Modern CPUs often have an integrated GPU, which shares the system memory with the CPU. A GPU can also be a standalone chip, which it is often referred to as a dedicated GPU. A dedicated GPU has its own memory which is often faster than the system memory, helping to speed up computations. It is the dedicated GPUs that nowadays are common to use for machine learning. Operations required to train a DNN can be made in parallel. Therefore GPUs exceed CPUs in performance when it comes to training. Running a program for training a neural network on a GPU requires that the GPU manufacturer supports it. For high-end GPUs there are two major manufacturers today, AMD and NVIDIA. Both of them supports the possibility to train neural networks.

NVIDIA have developed their hardware-software architecture CUDA [28] that allows code to be run on their GPUs with a high computational power. An alternative to CUDA is OpenCL for parallel computing on the most common types of processor, including GPUs, from multiple manufactures [36]. OpenCL is maintained by the non-profit consortium The Khronos Group. Both alternatives have their pros and cons [25]. Which one should be used differs from project to project and what hardware is available.

2.1.10 Software Libraries

There exist several software libraries for simplifying the process of implementing machine learning algorithms. They provide tools for setting up a functional neural network.

For this project the software library Tensorflow [1] was used to build the neural network. Tensorflow was chosen because it is a well documented library that is popular and therefore a lot of functions available via forums and blogs.

2.2 Predicting the Outcome of Football Matches

Predicting the outcome of football matches is today done by both football experts and machine learning algorithms, with various outcome. This section discusses some challenges in predicting the outcome and some previous work.

2.2.1 Challenges of Predicting the Outcome of Football Matches

Both human and computer predictions have their challenges. Humans are emotional and can affect the analysis of teams playing each other and therefore the prediction. A computer does not have the knowledge of the current mental health of the team. Cracks between players and coaches might affect the outcome. For both humans and computer there is also always the problem of deciding which attributes are important.

Football, as many other sports, is highly stochastic. One lucky hit among hundreds of passes, shots, and dribbles can in the end change the whole outcome of the game. This makes it more complicated to predict the outcome of football matches, both for humans and computers.

2.2.2 Previous Work

Several attempts have been made to predict the outcome of football matches, and other sports [24, 18, 23, 2]. The accuracy varies depending on what sport, league and approach that has been used. Most attempts are done on one or a low number of leagues by building statistical models. Basic neural networks have been used in a few cases, but none have used deep learning to train a network to learn football matches and high dimensional representations of players and teams. These approaches are just predictions before the match and cannot be used for prediction for ongoing matches.

3

Proposed Methods for Football Match Prediction

Previous works on predicting outcomes of team sport matches using machine learning have mainly focused on team data [24, 18, 23, 2]. The focus also lies on a single tournament, league or team.

Players representing a football club during a match are constantly changing. Both because a specific player does not make the team or that a player is transferred to a different team. The approach for this thesis is to focus on the players. In such a way it takes into account if a player does not play for a team in a specific match, which will have an impact.

A match is played at a specific time, and events occur at a relative time in a game. The order of matches and events matter since they have an impact on the future. Therefore a long short-term memory network (LSTM) [14] is exploited for this project.

3.1 Data

The dataset used for this project includes the information for each player in the teams. The dataset includes matches for leagues and tournaments for the 54 countries in the Union of European Football Associations (UEFA)[1], USA, Brazil, Chile, Mexico, Argentina, Uruguay, Australia, Japan and China. A full list of leagues and tournaments can be seen in Appendix B.

Since players move between clubs and leagues, the data used is for multiple leagues over multiple years. In this way, information about a player is moved along with the player when a player changes club and therefore changes in teams are noticed. Information about players is gathered from events in which they are involved in during a game. The following events in each game have players connected to them:

Lineups: Each starting player and head coach for a team.

Position: Starting position for each starting player.

[1]`http://www.uefa.com/memberassociations/uefarankings/index.html`

Goal: Goalscorer and possible assisting player.

Card: Which player gets the card and which type of card.

Substitution: Player entering the field and player leaving the field.

Penalty: Which player takes the penalty, and what is the outcome. Whether the penalty is saved the by goalkeeper is also connected to the event.

Below is an example of how the raw data of a goal looks like and Appendix A contains the structure of the raw data for a whole match.

```
"goal": [
    {
        "assistPlayerId": "f54pd3cld0qkc8ol15bk9ye39",
        "assistPlayerName": "F. Fernández",
        "contestantId": "410jti6axb01yhvbc0axsp8li",
        "lastUpdated": "2017-02-12T16:36:59.362Z",
        "periodId": 1,
        "scorerId": "ex186m2z3yp666b1g219a8ted",
        "scorerName": "A. Mawson",
        "timeMin": 36,
        "type": "G"
    }
]
```

The dataset consists of 35234 games distributed between 45.50% home wins, 29.61% away wins and 24.89% draws. All events include either the absolute or relative time. There is no ranking on what events are more important than others. Various sites and newspapers provide this information about football games, in different structures. Therefore, the project can be remade without having the data provided from a specific source.

3.2 Network Input

There are several ways to encode the data before feeding it to the network. The dataset contains several event types, with different structures and different numbers of attributes. We somehow need to merge or fuse this into a generalized input which the network can work with. A normal feed forward neural network requires all input to be of a fixed size, meaning that the dataset needs to be preprocessed so that everything is the same size. By using a RNN we can let all examples be of different length, but we still need the input vector at each time step to be fixed in size. This means that even though we have different types of events, they must have the same shape when fed to the network. There are several possibilities and techniques that can be can utilized to solve this problem that are outlined below.

3.2.1 Deep Embeddings

Inspired by *word2vec* as it works well for natural language processing, where embedding is used [29]. It has different inputs for each event type in the embeddings, and the embedding is a learned representation for the network input. This network should be trained end-to-end, learning both the embeddings and the primary task at the same time [37, 26, 9]. One then feeds the appropriate event at each time step, and set the remaining to be zero or other default “empty” value. Figure 3.1 depicts an example of embedded input at time t.

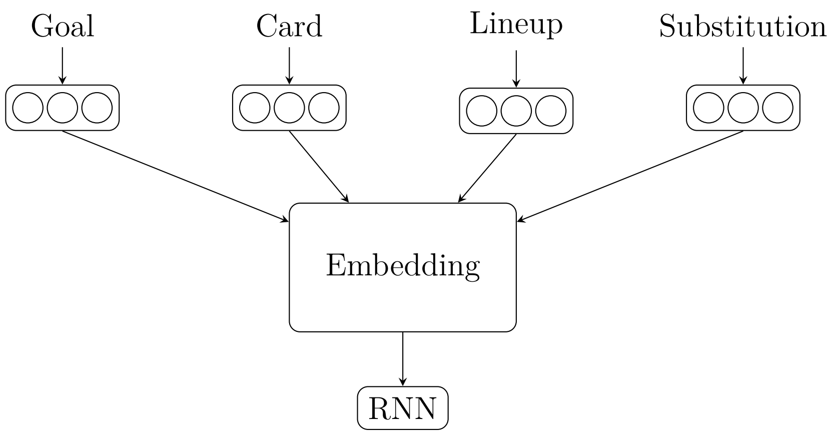


Figure 3.1: Architecture of deep embeddings.

3.2.2 One-Hot Vector with all Attributes

By using a one-hot vector which encodes all attributes for this event, the attribute vector contains columns for all the attributes of all events. This means that for different events several columns are unused, and are set to zero. All attributes except player and team information are one-hot encoded, the player and team are looked up in two different embedding spaces, as can be seen in Figure 3.2. This results in a very sparse input, but includes all different attributes for all events in the same vector.

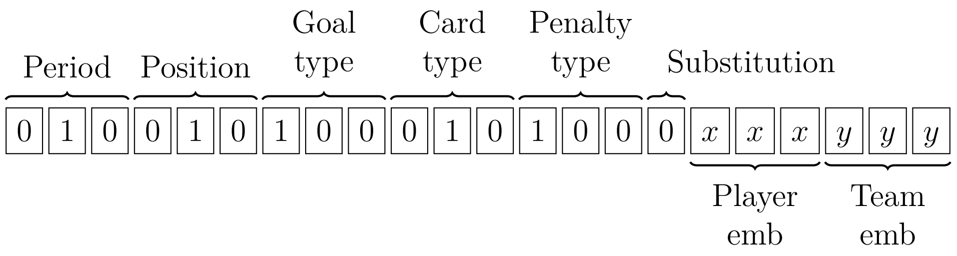


Figure 3.2: Example of one-hot encoded attributes concatenated with player and team embedding.

3.2.3 Concatenated Embedding Vectors for all Attributes

This is a slight variant of the one-hot vector above. Instead of using a sparse one-hot encoded vector we provide one embedding per event type, and a lookup for all the values in the attribute vector, and concatenate the resulting embeddings. These embeddings are also learned during end-to-end training of the network.

3.3 Naive Statistical Model

A pure statistical model is created to see what prediction accuracy is possible to reach with a simple model using the same data. The model does not care about players in each team. It only considers goals and cards. Below is the algorithm to predict the outcome of a match.

$$GoalScore_t = gh_t \cdot k_{gh} + ga_t \cdot k_{ga} - ch_t \cdot k_{ch} - ca_t \cdot k_{ca} \tag{3.1}$$

$$YellowCardScore_t = y_t \cdot k_y \tag{3.2}$$

$$RedCardScore_t = r_t \cdot r_y \tag{3.3}$$

Feature scaling is applied to each score to normalize the data.

$$x' = \frac{x - min(x)}{max(x) - min(x)} \tag{3.4}$$

$$TotalScore_t = GoalScore'_t - YellowCardScore'_t - RedCardScore'_t \tag{3.5}$$

$$Score_m = TotalScore_{t1} - TotalScore_{t2} \tag{3.6}$$

$$f(Score_m) = \begin{cases} \text{home win,} & \text{if } Score_m > \text{Home win threshold} \\ \text{away win,} & \text{if } Score_m < \text{Away win threshold} \\ \text{draw,} & \text{otherwise} \end{cases} \tag{3.7}$$

$GoalScore_t$	Goal score for team t
gh_t	Home goals for team t
k_{gh}	Home goal rate
ga_t	Away goals for team t
k_{ga}	Away goal rate
ch_t	Goals conceded home for team t
k_{ch}	Goal conceded home rate
ca_t	Goals conceded away for team t
k_{ca}	Goal conceded away rate
$YellowCardScore_t$	Yellow card score for team t
y_t	Yellow cards for team t
k_y	Yellow card rate
$RedCardScore_t$	Red card score for team t
r_t	Red cards for team t
k_r	Red card rate
$TotalScore_t$	Total score for team t
$Score_m$	Score for match m

Each team t in each match gets a score that depends on goals, made and conceded, yellow and red cards for the 2015, 2015/2016, 2016, 2016/017 seasons until 2016-11-05. The predicted outcome is then decided by the difference in score between TotalScore'$_{t1}$ and TotalScore'$_{t2}$. The k variables are used as a ratio to tune the model.

3.4 Deep Learning Architecture

A few different models have been used and tested in this project. The core of our RNN consists of LSTM or GRU cells and a softmax classifier. For input both one-hot, see Secion 3.2.2, and embedding vectors, see Section 3.2.3, are used and tested. The longest sequence in the dataset is calculated and that is used to pad all the other sequences with zero vectors so that they are the same length. Dynamic unrolling of the sequence takes place by also feeding a sequence length parameter. This makes the network only run for as many time steps that each sequence contains.

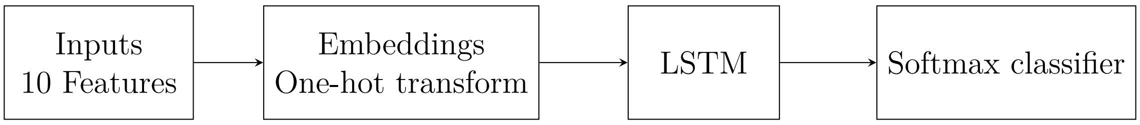


Figure 3.3: The simple high level architecture of the model.

The block diagram of the model can be seen in Figure 3.3 that consists of inputs, input processing, some number of LSTM layers and units, and lastly a softmax classifier. This architecture can also be seen in detail in Figure 3.4. The input vector consists of 10 integer features: period, home team, away team, main player, assisting player, position, goal type, card type, penalty type, and substitute. These

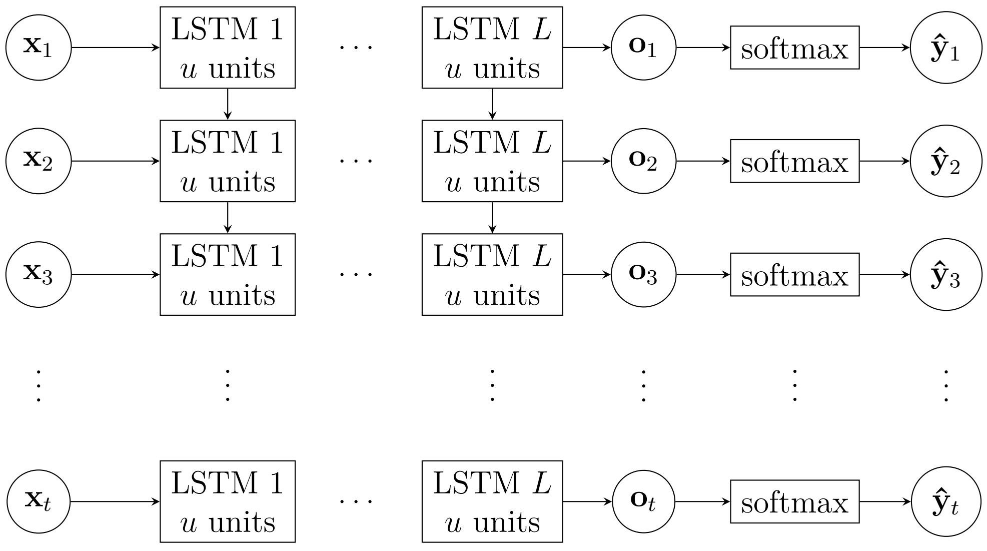


Figure 3.4: Detailed illustration of the architecture used. $\mathbf{x_t}$ is the input vector at time step t, $\mathbf{o_t}$ is a single layer feed forward neural network, $\hat{\mathbf{y}}_\mathbf{t}$ is the predicted class at time t, L represents the number of layers, and u number of LSTM units per layer.

features are then either transformed to a one-hot encoding per feature and then concatenated together, or an embedding lookup per feature and then concatenated to form a bigger vector with real valued numbers. An example of an input vector can be (penalty event by the away team in the second half):

$$\begin{bmatrix} 2 & 0 & 1309 & 929 & 28619 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

and the concatenated embedding vector like this (abbreviated for readability):

$$\begin{bmatrix} 0.8029604 & 0.18677819 & 0.58298826 & \dots & 0.3224628 & 0.41138625 & 0.05778933 \end{bmatrix}$$

The softmax classifier has an output of three different classes, being home win, draw, and away win. The loss function used to calculate the error is a standard cross entropy error, discussed in Section 2.1.7 The loss and accuracy are tested in two different ways, many-to-one and many-to-many, both introduced in Section 2.1.3 and further discussed in Sections 3.4.1 and 3.4.2.

3.4.1 Many-To-One

One being many-to-one, meaning that the loss was only calculated once, on the last time step and then being used to do backpropagation through time for the entire sequence. The calculations for accuracy of this technique is described as follows:

For each match m, there is a target (i.e., class label) y_m:

$$y_m = \begin{cases} [1,0,0] & \text{(Home victory)} \\ [0,1,0] & \text{(Draw)} \\ [0,0,1] & \text{(Away victory)} \end{cases}$$

and an output $\hat{y}_m$ from the last event in match m

$$\hat{y}_m \in \{[1,0,0],[0,1,0],[0,0,1]\}$$

The accuracy is then calculated by

$$\mathrm{r}_m = \begin{cases} 1 & \text{if} \quad \hat{y}_m = y_m \\ 0 & \text{otherwise} \end{cases} \tag{3.8}$$

$$\text{accuracy} = \frac{\sum\limits_{m=1}^{M} \mathrm{r}_m}{M} \tag{3.9}$$

where M is the number of matches.

3.4.2 Many-To-Many

The second approach is an average loss over the entire sequence, calculating the loss for each time step and then averaging this loss for the backpropagation. This technique is called sequence loss, since it calculates the total loss of a sequence of outputs. It is used to get a better prediction over the entire sequence. Since the aim is to predict the result at any time, not just in the end, this might give a better result. The calculations for the accuracy of this technique is described as follows:

For each match m, there is a target (i.e., class label) y_m:

$$y_m = \begin{cases} [1,0,0] & \text{(Home victory)} \\ [0,1,0] & \text{(Draw)} \\ [0,0,1] & \text{(Away victory)} \end{cases}$$

$$\hat{y}_m = [\hat{y}_m^1, \hat{y}_m^2, ..., \hat{y}_m^n], \quad \hat{y}_m^i \in \{[1,0,0],[0,1,0],[0,0,1]\} \quad \text{for} \quad i = 1...n,$$

where $\hat{y}_m^i$ is the predicted outcome for event i, and n is the number of events in the match m.

The number of correct predictions for each match is calculated by

$$\mathrm{r}_m^i = \begin{cases} 1 & \text{if} \quad \hat{y}_m^i = y_m \\ 0 & \text{otherwise} \end{cases} \quad \text{for } i = 1...n, \tag{3.10}$$

and then the accuracy over all matches is calculated by

$$\text{accuracy} = \frac{\sum\limits_{m=1}^{M}\sum\limits_{i=1}^{n} \mathrm{r}_m^i}{\sum\limits_{m=1}^{M} \#\text{elements in } \mathrm{r}_m} \tag{3.11}$$

where M is the number of matches.

4

Results and Evaluation

This chapter shows the result of using different methods and hyperparameters to classify and predict football matches. The figures in this chapter only include the case studies that we found to work best for ease of readability and visibility. Figure 4.1 shows a diagram that describes how the different parts of the system is connected.

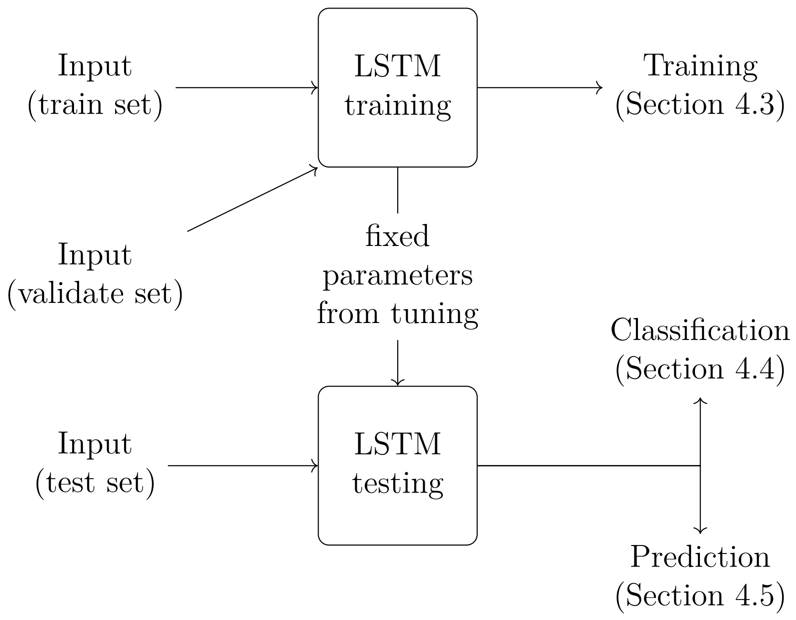


Figure 4.1: An illustrative map of the system describing different parts of this chapter.

4.1 Setup

The network was trained and evaluated on a desktop computer running Linux. The computer specifications can be found in Table 4.1 and more details about the graphics card used can be found in Table 4.2. Versions of the software libraries used can be found in Table 4.3.

CPU	Intel Core i7 6700K 4 GHz
Motherboard	ASUS Z170-A S-1151 ATX
RAM	Corsair Vengeance LPX 16GB 2666MHz DDR4
GPU	ASUS DC2 OC Strix NVIDIA GeForce GTX 960 4GB
SSD	Samsung 850 EVO

Table 4.1: Specifications of the computer specifications used for training.

Manufacturer	Asus
Clock frequency	1126MHz
Cuda cores	1024
Memory	4.0 GB GDDR5
Memory frequency	7010Mhz

Table 4.2: Specifications of the graphics card specifications used for training.

4.2 Dataset

The dataset used in the evaluation is described in detail in Section 3.1. It contains 35234 matches for two years, from 2015 until 2017. Table 4.4 lists the number of matches in each of the training, validation, and testing sets. The validation and testing dataset are normalized to contain roughly the same number of classes, so that the results on the testing set should not be biased. Unfortunately this does not make our results entirely comparable with others, since the data is always biased towards home winners. If we compare the test accuracies from Table 4.5 with the non-normalized test set we got numbers as high as 55% for case studies 1–6.

4.3 Training

Training was a big and time consuming part of the project. There are parameters to tune, architectures, input encodings, and losses to compare just to get the best inference when trying to predict outcome. Some techniques result in better classification but lower prediction than others, so in the next few sections we will describe how and what we did to improve the results.

Linux kernel	4.4.0-64-generic
Nvidia driver	375.26
cuDNN	5.1
CUDA	8.0
Python	3.6
Tensorflow	1.0

Table 4.3: Software used for training.

Train	Validate	Test	Discarded	Total
24410	3608	6660	557	35234
69.3%	10.2%	18.9%	1.6%	100%

Table 4.4: Dataset split into three parts used for training, validation, and testing.

4.3.1 Tuning Parameters

Several combinations of hidden layers, units per layer, batch size, learning rate, and embedding sizes were tested. This subsection includes the result for the parameters that were most successful, both in terms of accuracy and how feasible the performance requirements were. We were constrained by hardware and time which forced us to pick branches of parameters that seemed good to explore further, most of the results gathered during this project has been discarded since we only chose a few top ones. In the following section we describe the parameters of our LSTM architectures and how they were tuned.

Dropout The first networks we trained did not utilize dropout (see Section 2.1.4). After analyzing the first few networks performance it was very clear that overfitting was occuring, reducing the accuracy of the network. We understood that dropout must be used in this case to reduce overfitting and increase accuracy. Therefore all networks shown in this chapter have been trained with a 50% dropout rate, all other networks were discarded because of the low accuracy in comparison.

Learning rate We experimented with changing the learning rate of the network but found no surprise here. When increasing the learning rate too much the loss randomly jumped or improved just very little. When decreasing the learning rate too much the loss never improved at all. We found values around 10^{-4} to work very well, both in times of convergence and time used to train which is why we choose to fix it as 10^{-4}.

Batch size We tried batch sizes between 1 and 500 and noticed that sizes over 100 were too large for our data and setup. The accuracy of the network was very stable, but the loss could not make it improve on the training set at all. We decided to use values like 30 or 50 to make the training time faster for new tests, but decided to keep it at 10 when doing the final training.

Embedding dimensions When using the inputs to do embedding lookups before feeding to the LSTM one needs to decide on how big and how many dimensions the embeddings should contain. We first started with the team and player embeddings. We tried values between 1 and 100, it seemed to make no difference between having 10 or 100 dimensions for teams or players. We did notice a big difference when having as low as 1 or 2, then we would not see any convergence and the loss would just randomly jump around. We also tried different values for the rest of the embeddings, but saw the same thing as with the teams or players embedding dimension size. Since there are many different values for the team and players we decided to continue with a dimension size of 30, and for the rest of the values we used 10 to ease the size of the input for computational reasons.

4.4 Classification

Our tests are divided into two parts, classification and prediction. Classification uses all the data available and simply does a classification of which class a match belongs to. There are three different classes, home win, draw, and away win.

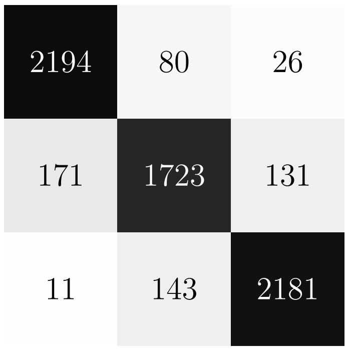
We decided to pick seven different case studies out of all that we have tried and show data and plots of all of these. They are numbered and the first six uses the many-to-many approach described in Section 3.4, where the loss and accuracy were calculated as the average loss/accuracy of all outputs for every time step, see Section 3.4.2.

The seventh case study uses the many-to-one approach, see Section 3.4.1, which calculated the loss and accuracy only on the last prediction. This has of course affected the value "accuracy" in tables, as the average accuracy is much lower than the last accuracy since it includes all of the accuracies at all time steps. This can be seen in Table 4.5 where all the case study parameters and accuracy are shown.

Case Study	Layers	LSTM Units	Train Average Accuracy	Test Average Accuracy	Train Accuracy	Test Accuracy
1	1	3	0.5979	0.4595	–	0.7777
2	1	256	0.6150	0.5003	–	**0.8868**
3	2	256	0.6179	0.4952	–	0.8459
4	2	512	**0.6254**	0.4909	–	0.8532
5	2	1024	0.6221	0.4873	–	0.8389
6	1	2048	0.6116	**0.5022**	–	0.8682
*7	2	256	–	–	**1.0000**	**0.9863**

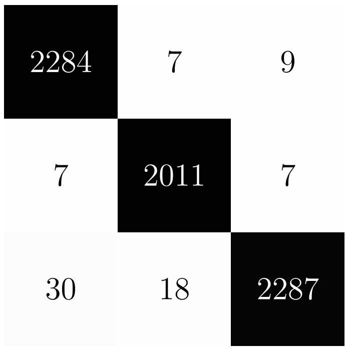
Table 4.5: Comparison between the different training case studies. Case Study 7 uses the many-to-one approach where the accuracy is only calculated at the end of the sequence, while Case Studies 1–6 uses many-to-many and calculates the accuracy for all events in the sequence and takes the average of them all. The "Train Accuracy" and "Test Accuracy" columns contains the last event classification for both approaches.

Observing the classification results there is one clear winner out of all case studies. Case study 7 has a training accuracy of 100% and a testing accuracy of 98% which is very high. Comparing this with the other six having a test accuracy at 45–50%, we can see that it is much easier to classify using only the latest information instead of doing a prediction at every time step and then averaging this. This is also very evident when looking at the accuracy in Figures 4.3, and 4.4, and loss in Figures 4.5, and 4.6. Case study 7 quickly drops the loss below 1, both for training and testing, before running training for the first epoch. Figure 4.2 shows the confusion



2194	80	26
171	1723	131
11	143	2181

(a) Case study 3, using many-to-many (Section 3.4.2).



2284	7	9
7	2011	7
30	18	2287

(b) Case study 7, using many-to-one (Section 3.4.1).

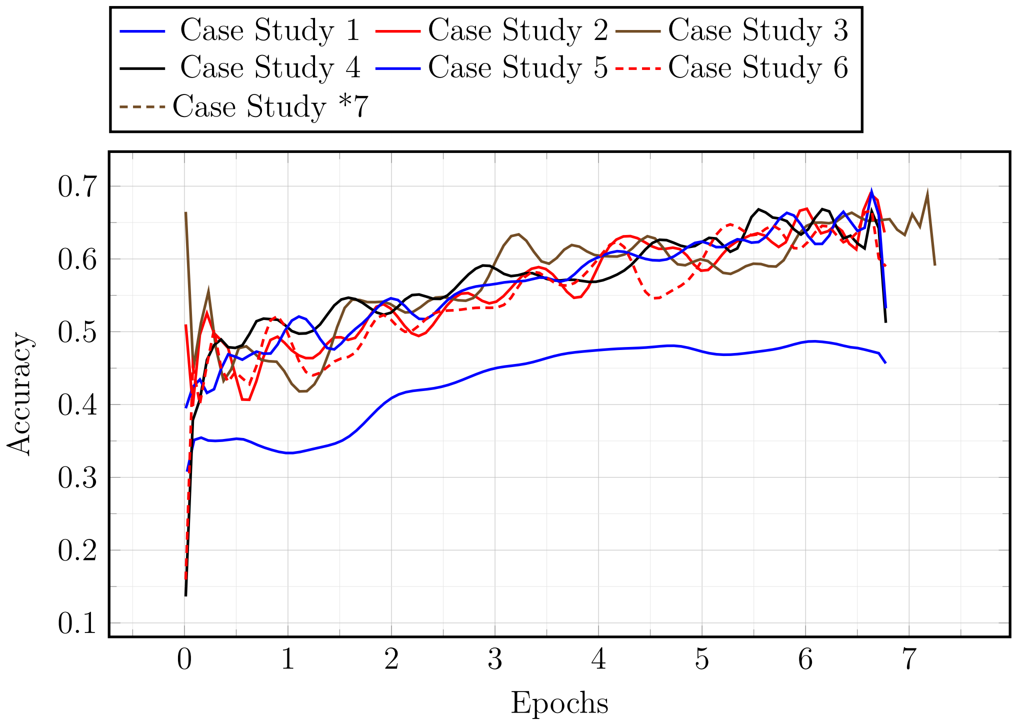
Figure 4.2: Confusion matrices for classification. Each row is actual truth, row 1 for home win, row 2 for draw, and row 3 for away win. Each column is the predicted value in the same order. The color of the background shows how big part of the distribution is on that cell, where black means 100%, white 0%, and gray shades for all the values in between. The values in the confusion matrices are calculated for only the last event.

matrices for case studies 3 and 7. These matrices visualizes the detailed performance of our predictions, we can see that case study 7 has very few errors and a clear black diagonal. Case study 3 not as good, some gray areas close to the diagonal. The accuracy of case Study 7 follows the loss and reaches above 90% before finishing the first epoch, and after that slighty increasing towards 98%. Ignoring this clear winner and just comparing case studies 1–6 we see that case study 1 never reaches the same accuracy as the others, the same thing with the loss, it never goes as low as for the other 2–5. This is not surprising since Case study 1 only has one layer of LSTM with three units, not much room to recognize patterns in the data. Comparing the rest of the Case studies 2–5, it is slighty disappointing, no clear winner by the plots in Figure 4.5, 4.6 or 4.3, 4.4, but at least Table 4.5 can show us the values in the end of training by zooming in. Some values are a bit surprising, case studies 3, 4, and 5 are all lower in test accuracy than case study 2 even though it has the least amount of parameters available, the only one better than case study 2 is case study 6 which has one huge layer with 2048 units inside. We also tried networks much larger than this, all the way up to 10 layers with 2048 units, but saw no convergence even after several days of training.

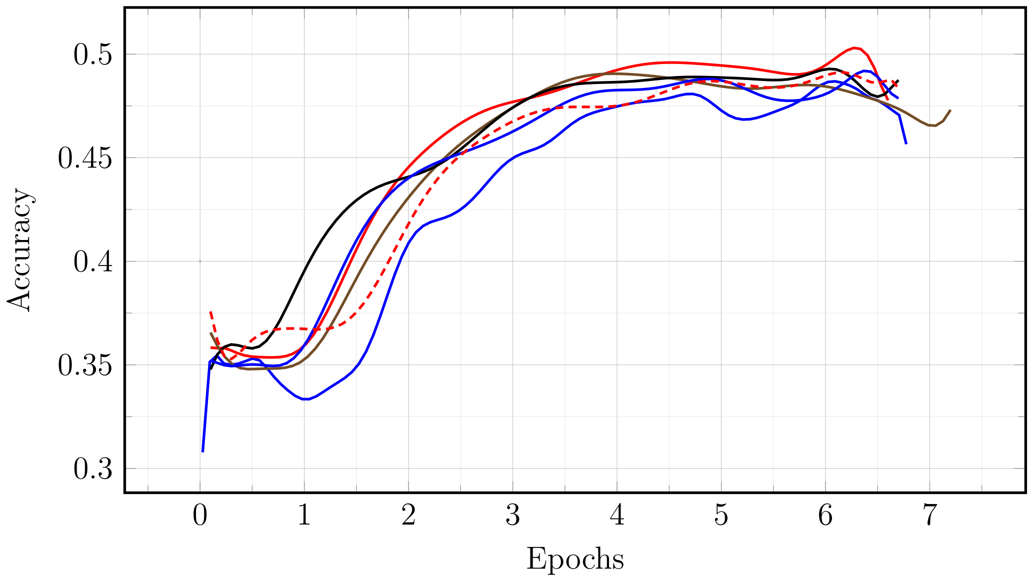
4.4.1 Discussion

We thought and hoped that there would be a bigger difference between the different case studies, but it seems that it only makes a small difference in classification results. We thought that bigger networks would result in better classification accuracy, but it seems to be only slighty better if even better at all.

One thing that makes a huge difference is how the accuracy is calculated. When

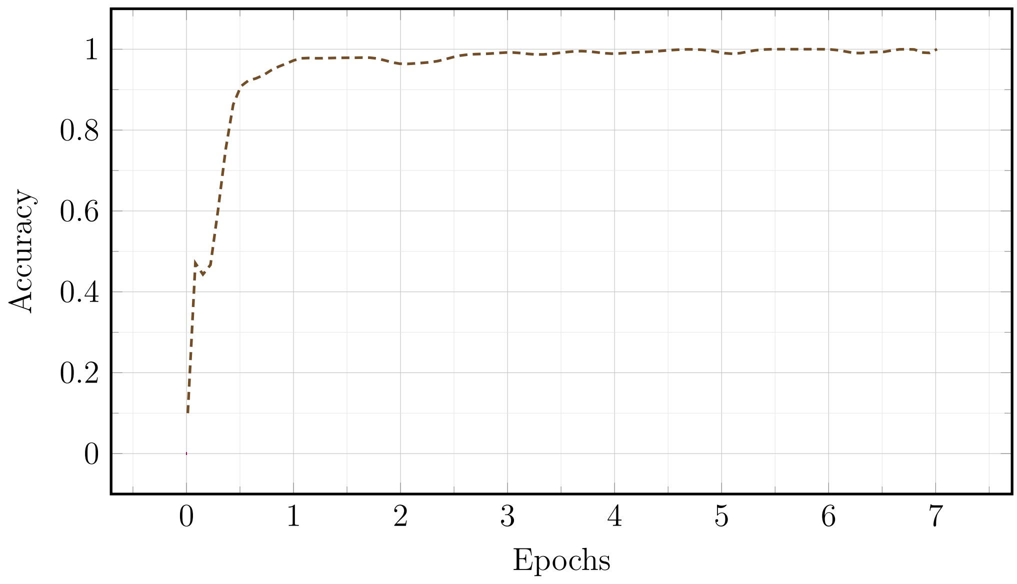


(a) Training accuracy.

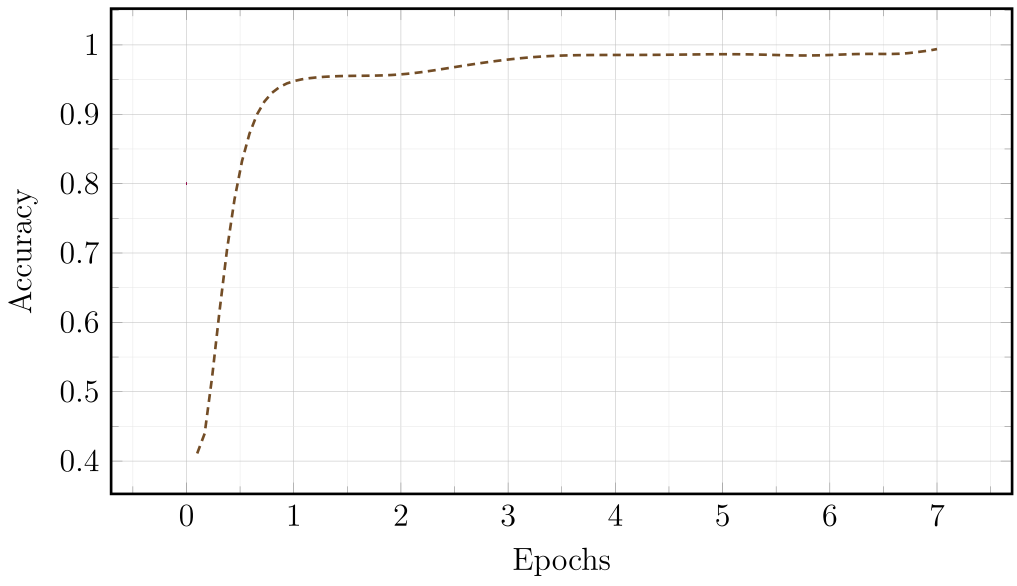


(b) Validation accuracy.

Figure 4.3: Accuracy for classification of Case studies 1–6. The color and shape of the legend is used throughout the rest of the following plots.

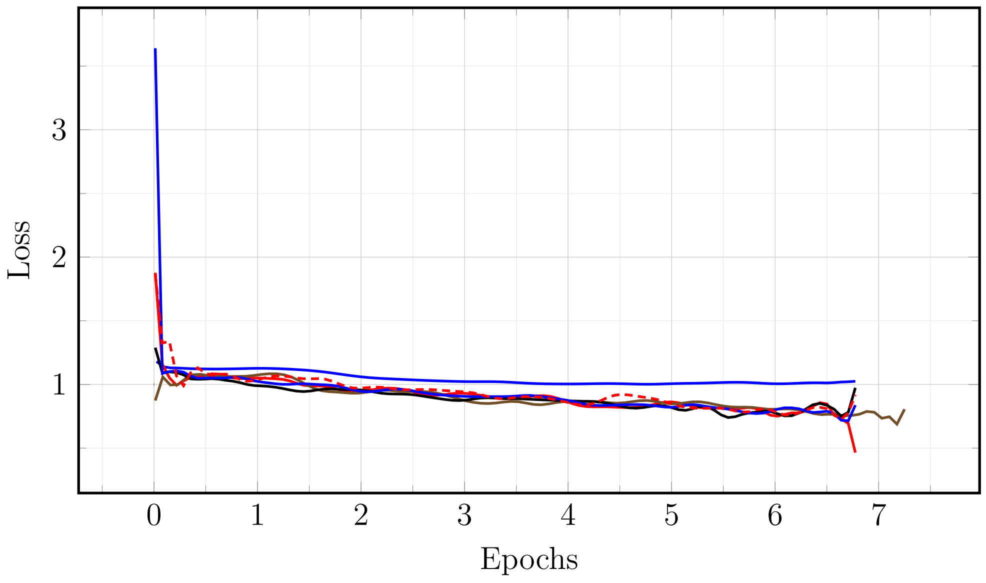


(a) Training accuracy.

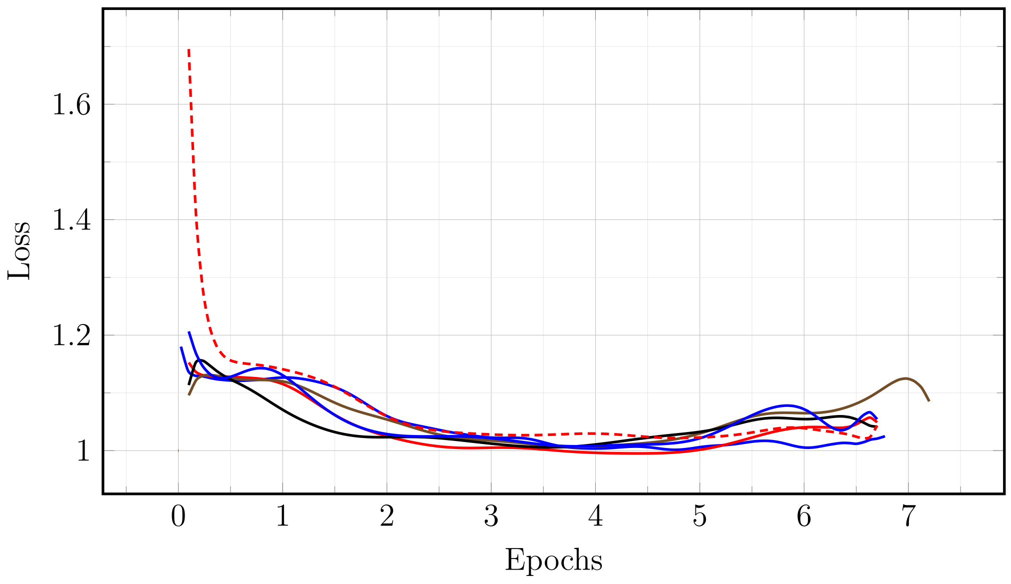


(b) Validation accuracy.

Figure 4.4: Accuracy for classification of case study 7.

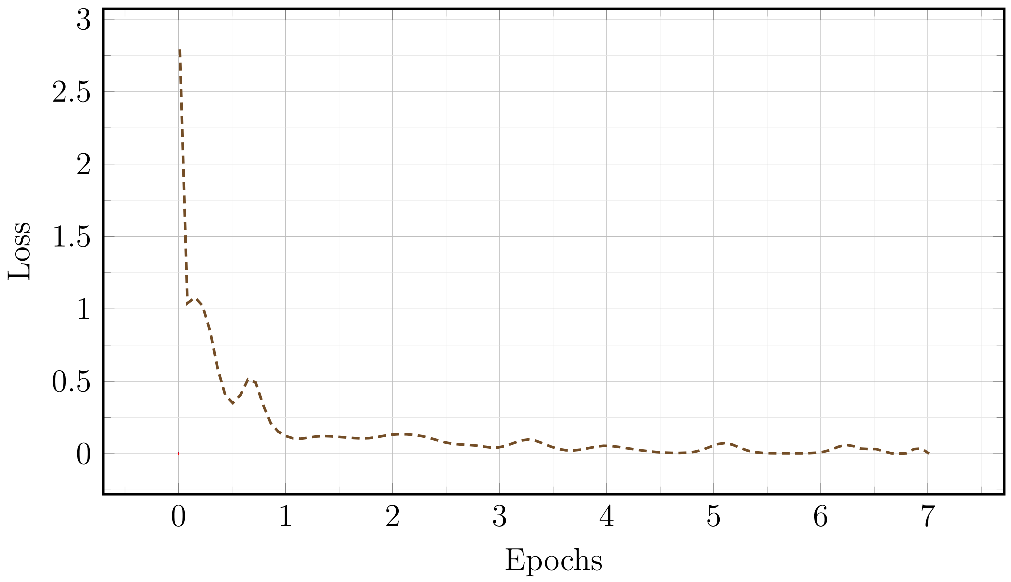


(a) Training loss.

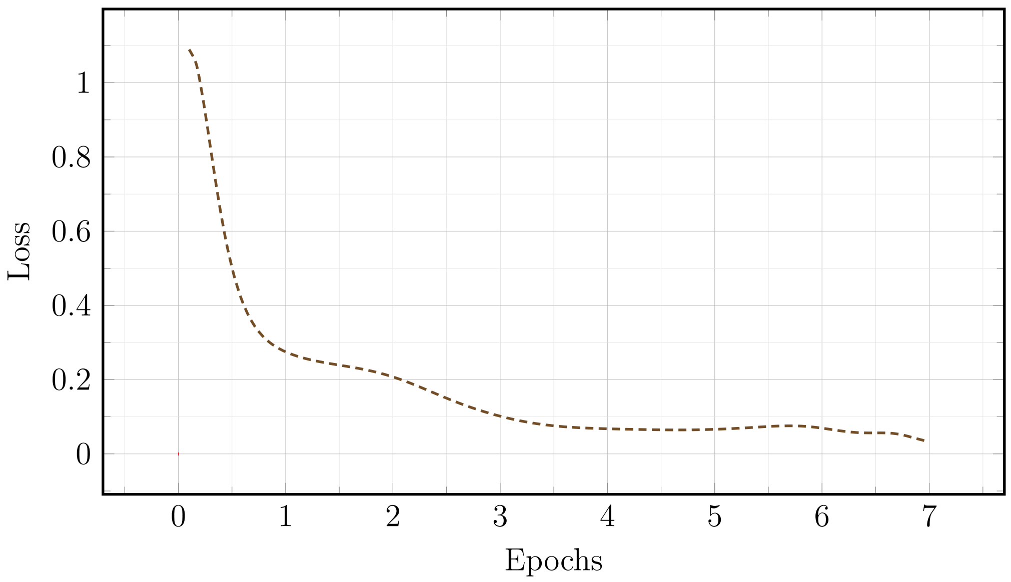


(b) Validation loss.

Figure 4.5: Loss for classification of case studies 1–6.



(a) Training loss.



(b) Validation loss.

Figure 4.6: Loss for classification of case study 7.

calculating the outcome on all events and then taking the average we get a lower classification accuracy than when we calculated the outcome on only the last event. This is no surprise, in the first case the calculated outcome on the first event has the same impact as the calculated outcome on the last event. This does not make sense since the calculated outcome on the last event contains more information about the match, it should have a bigger impact.

For case study 3 we can see that it does few away winner classifications when the outcome is home winner and the other way around, which is a positive thing. This is not the case with case study 7. It does more away win predictions than draw predictions when the outcome is home winner and more home winenr in predictions than draw predictions when the outcome is away winner. This is however so few cases that it should not be considered a problem.

The interested reader who wonders what accuracy case studies 1–6 have on the last prediction Table 4.6 shows this as we move on to the prediction section.

4.5 Prediction

This section describes the results from prediction. Prediction is performed by using only part of the previous data. The network predicts the future values which have not yet been seen. We decided to predict every 15th minute of all matches. This means that we have one prediction before the match starts, 15th minute, 30th minute, 45th minute (half time), 60th minute, 75th minute, 90th minute, and full time (since matches can have extended full time).

			Prediction Accuracy During Match							
Case Study	Layers	LSTM Units	0 m	15 m	30 m	45 m	60 m	75 m	90 m	Full Time
1	1	3	0.4284	0.4361	0.4631	0.5039	0.5747	0.6469	0.7305	0.7777
2	1	256	**0.4396**	**0.4479**	**0.4705**	**0.5151**	0.5831	0.6797	0.8048	0.8868
3	2	256	0.4226	0.4283	0.4537	0.4982	0.5661	0.6644	0.7784	0.8459
4	2	512	0.4355	0.4404	0.4621	0.5062	0.5741	0.6728	0.7813	0.8532
5	2	1024	0.4313	0.4393	0.4659	0.5051	0.5740	0.6687	0.7775	0.8389
6	1	2048	0.4324	0.4380	0.4638	0.5115	0.5811	0.6813	0.7954	0.8682
*7	2	256	0.3335	0.3539	0.4151	0.5048	**0.6280**	**0.7409**	**0.8825**	**0.9863**

Table 4.6: Prediction comparison between the different case studies.

Table 4.6 lists all the case studies with their parameters (same as before), and prediction accuracy before the match starts and every 15th minute until full time. Figures 4.7, 4.8, 4.9, 4.10, 4.11, 4.12, 4.13, and 4.14 shows the prediction accuracy at each 15th minute time interval and full time. Unfortunately due to smoothing, since the accuracy varies a lot, the values are a bit lower as compared to the best results recorded into Table 4.6. Out of case studies 1–6, case study 1 is also the worst one, same as in the classification, since it has such a small set of parameters to learn.

We can see that case study 7 still has the best full time accuracy, which is to be expected, but if we look at the “0th minute” column we can see that case study 7 actually is the worst. Even the tiny case study 1 has a better prediction than case study 7 with a 42% accuracy compared to 33%. This can also be seen in Figures 4.7, 4.8, and 4.9, where case study 7 is by far the worst predictor. Although this changes when we reach minute 45, then case study 7 is somewhat equal to the rest of the networks, and after this time step it improves and exceeds the other networks performance. It seems to sacrifice earlier accuracy for the later, where it has an excellent result.

Figures 4.15 and 4.16 show confusion matrices for case study 3 and case study 7. What can be seen in both case study 3 and case study 7 is that in the earlier predictions a lot of the guesses are on home winner. This is somewhat expected since the training dataset has a bias towards the home winners. As the time progresses to later predictions we can see that the distribution moves towards the correct state with black on the diagonal. Case study 3 is much better at finding away winners in the earlier predictions, in minute 0 it correctly predicts 999 away wins in comparison to 134 for case study 7. Towards the end of predictions we can see that very few mistakes happen where both networks predict an away winner when the target is a home winner and vice versa, most of the mistakes there are instead towards the “closest” class which must be seen as a positive result. It is a lot more likely to end in a draw than an away winner when the target is a home winner, it requires more for the actual result to go from one team winning to the other team winning. On the last prediction we can see just how good case study 7 is, predicting 173 and 25 matches wrong on home winner compared with 186 and 47 for case study 3. For draw matches case study 7 predicts 157 and 161 matches wrong, where case study 3 has 301 and 239. And last, when away winner is the true target, case study 7 has 28 and 233 wrong, and case study 3 has 36 and 309.

4.5.1 Discussion

The fact that all case studies gets better prediction the further match time goes is not a surprise, as more information should lead to a better prediction.

As seen in Figures 4.15 and 4.16 both case study 3 and 7 seem to have learned from the training data. It is more likely that the outcome is a home winner. Case study 3 also seems to have learned a better representation, than case study 7, of the outcome from the begining. Home winner is most likely, away winner the second and draw the last. The further a match goes, both networks make few away winner predictions when the outcome is home winner and the other way around which is a positive result.

The two different networks in case study 7 and case study 2 switch at beeing best at predictions around half-time. It could be due to the reason that the further a match goes on, the more data the networks need to handle. Therefore the smaller network might not be able to handle the amount of data in the end of a match, while the large network does not have any advantages over the small network in the begining

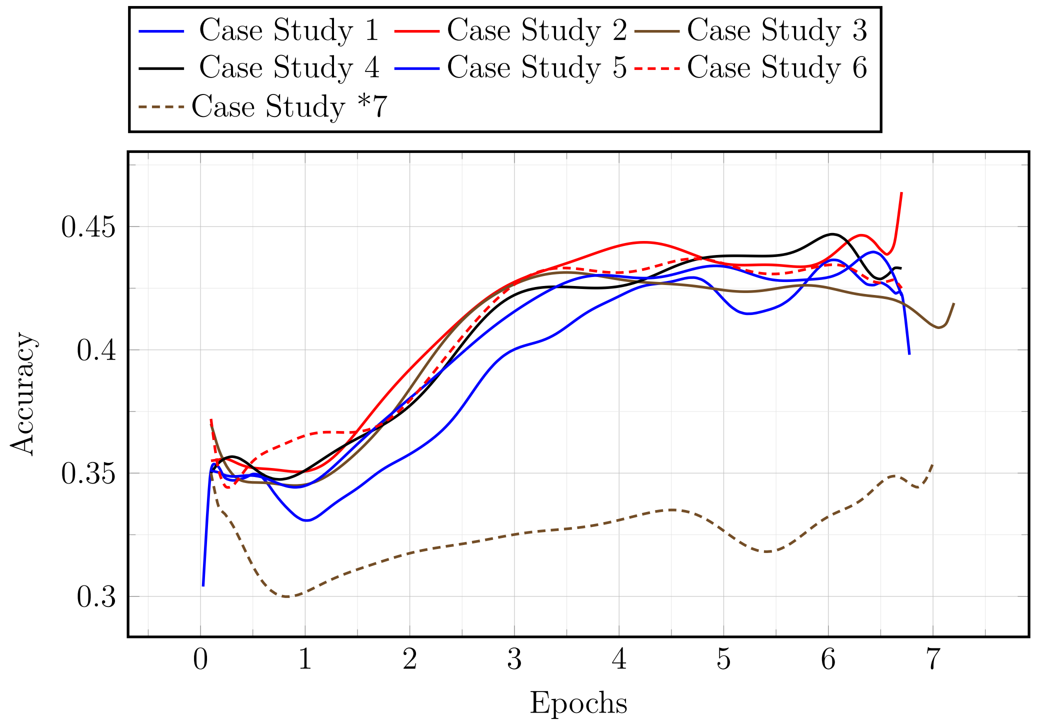


Figure 4.7: Prediction accuracy at match minute 0. The color and shape of the legend are used throughout the rest of the following plots. NOTE: Case Studies 1–6 use the many-to-many approach (Section 3.4.2), while Case Study *7 uses the many-to-one (Section 3.4.1).

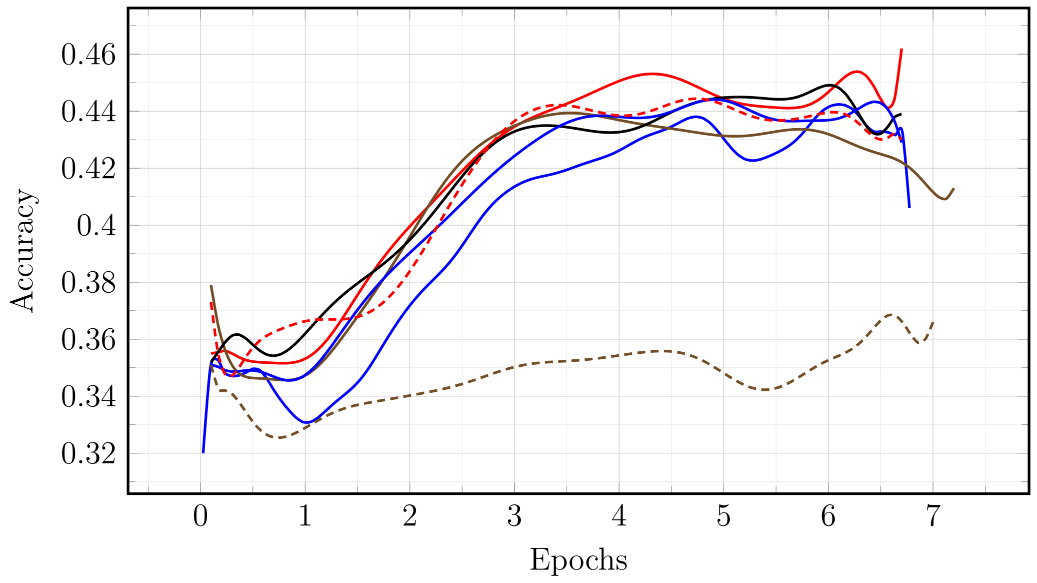


Figure 4.8: Prediction accuracy at match minute 15 for case studies 1–7.

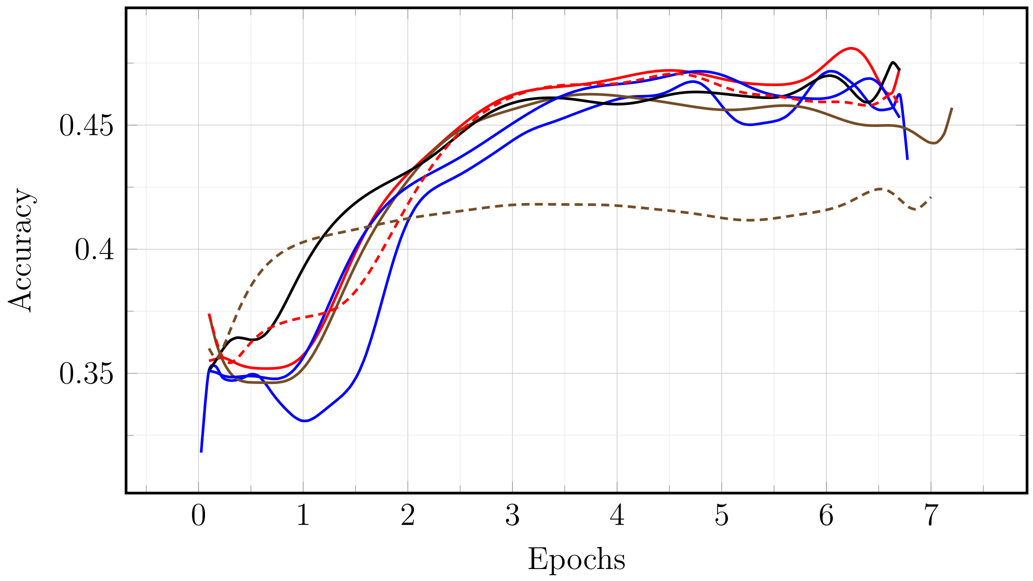


Figure 4.9: Prediction accuracy at match minute 30 for case studies 1–7.

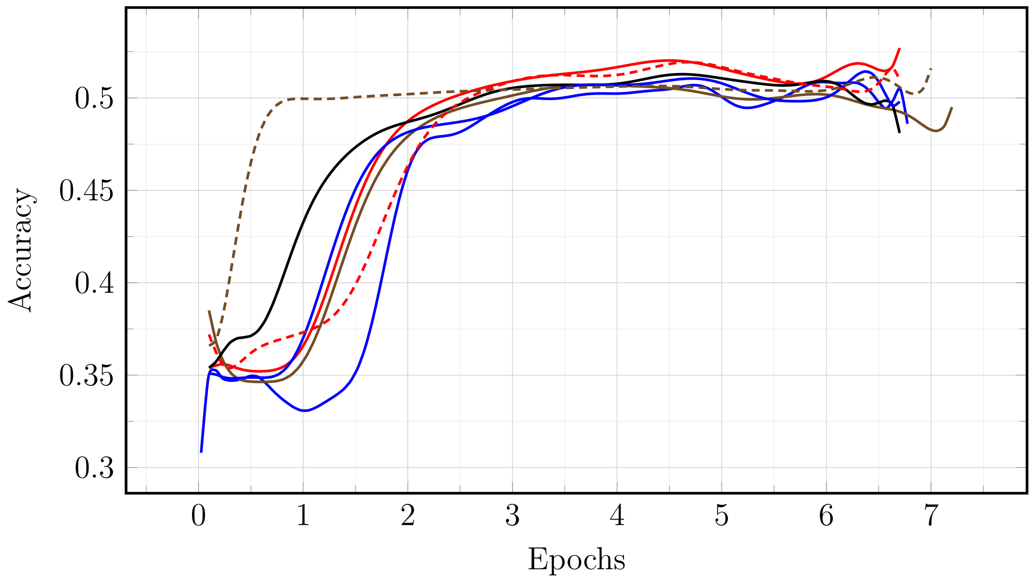


Figure 4.10: Prediction accuracy at match minute 45 for case studies 1–7.

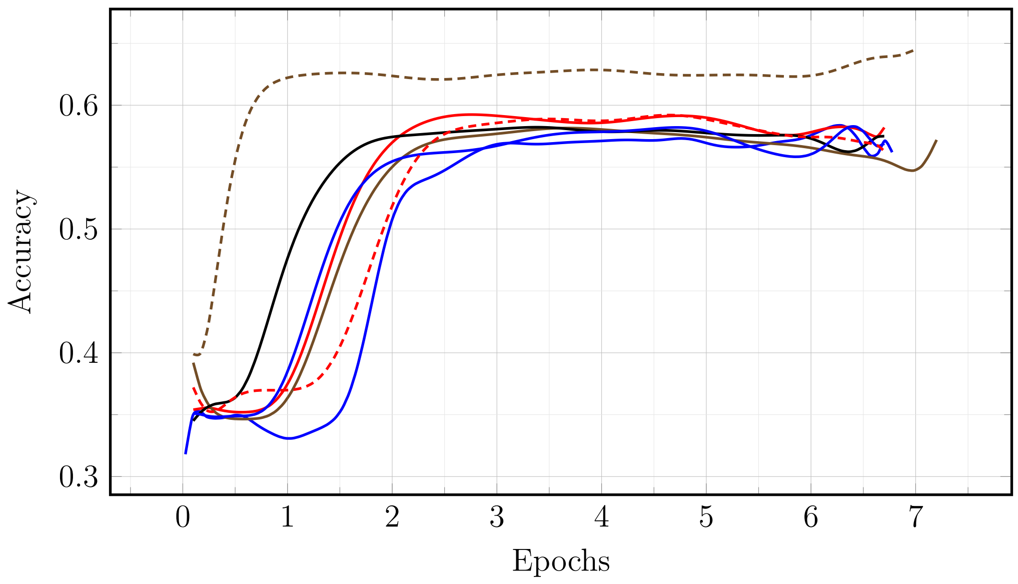


Figure 4.11: Prediction accuracy at match minute 60 for case studies 1–7.

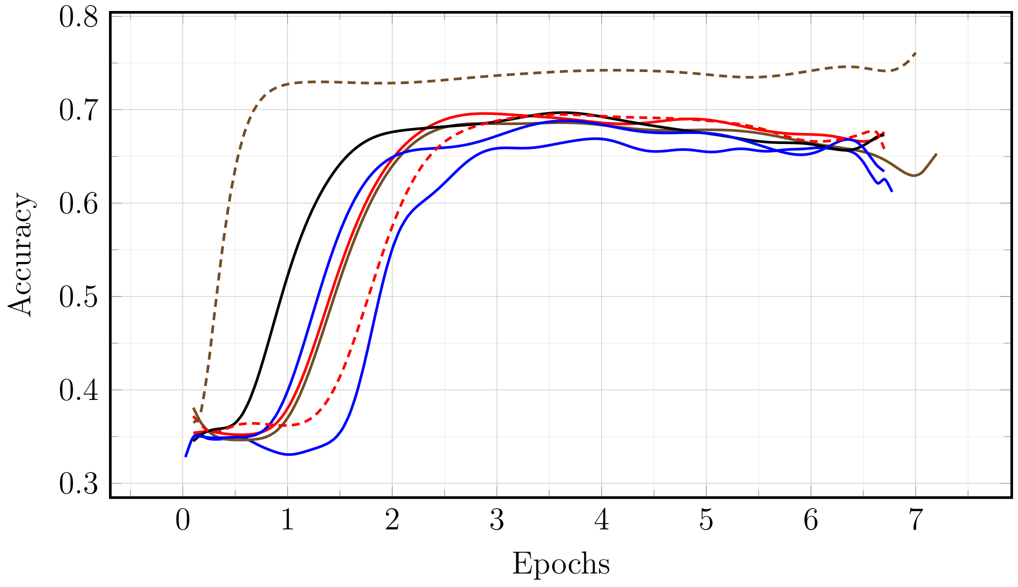


Figure 4.12: Prediction accuracy at match minute 75 for case studies 1–7.

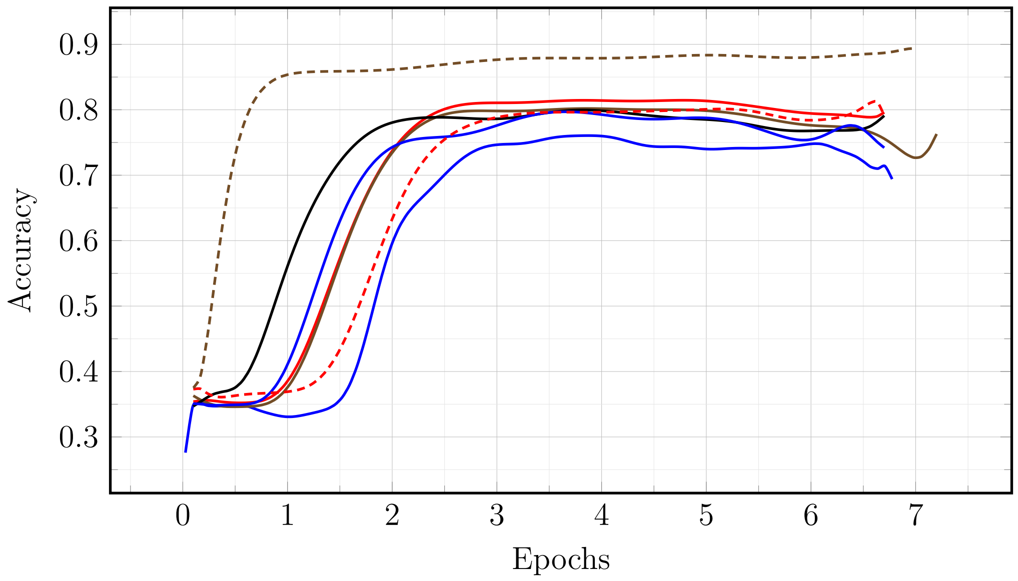


Figure 4.13: Prediction accuracy at match minute 90 for case studies 1–7.

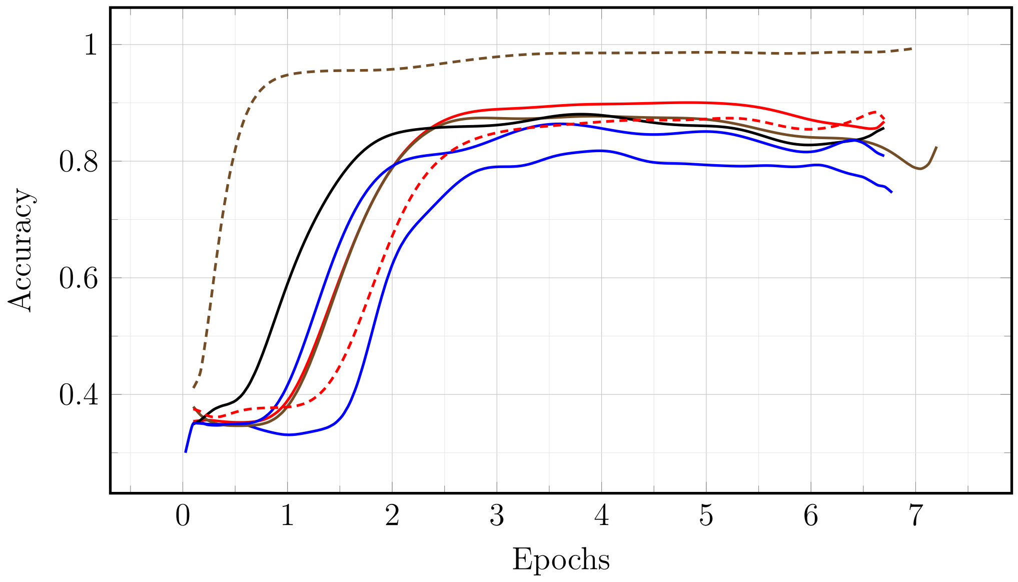
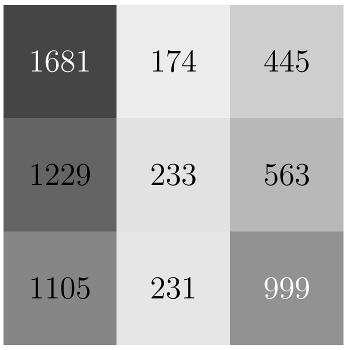
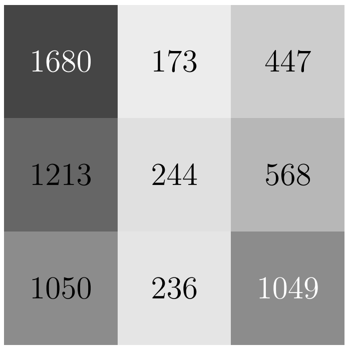


Figure 4.14: Prediction accuracy at full time for case studies 1–7.



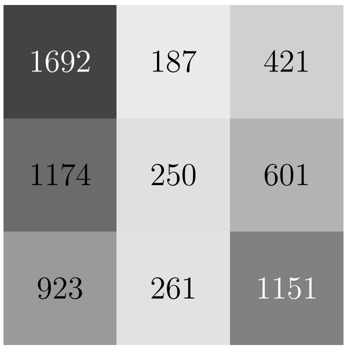
1681	174	445
1229	233	563
1105	231	999

(a) Match minute 0



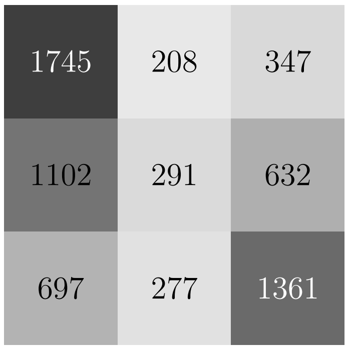
1680	173	447
1213	244	568
1050	236	1049

(b) Match minute 15



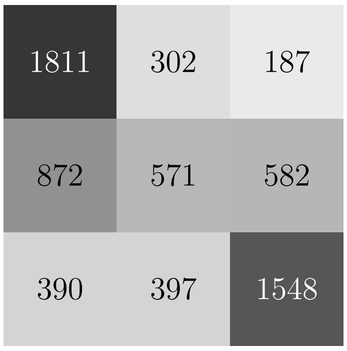
1692	187	421
1174	250	601
923	261	1151

(c) Match minute 30



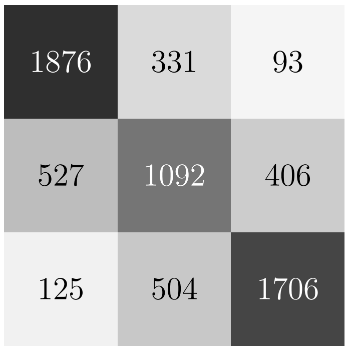
1745	208	347
1102	291	632
697	277	1361

(d) Match minute 45



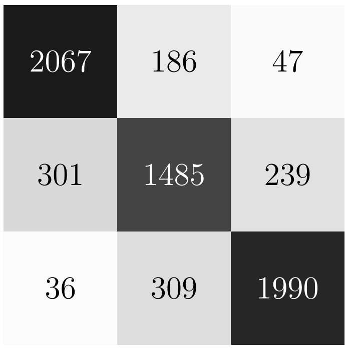
1811	302	187
872	571	582
390	397	1548

(e) Match minute 60



1876	331	93
527	1092	406
125	504	1706

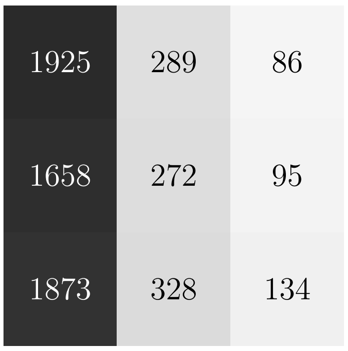
(f) Match minute 75



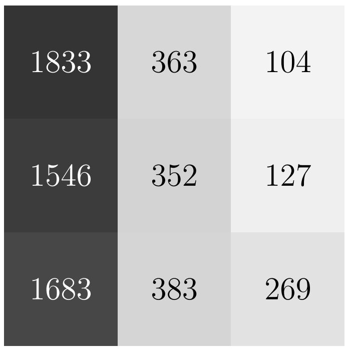
2067	186	47
301	1485	239
36	309	1990

(g) Match minute 90

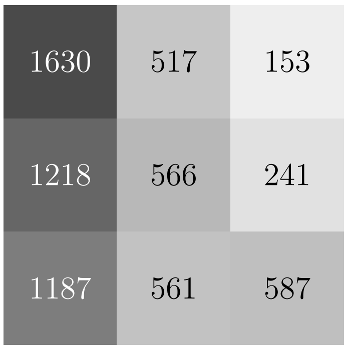
Figure 4.15: Confusion matrices for match minute 0, 15, 30, 45, 60, 75, and 90 for network case study 3 with two layers of 256 units.



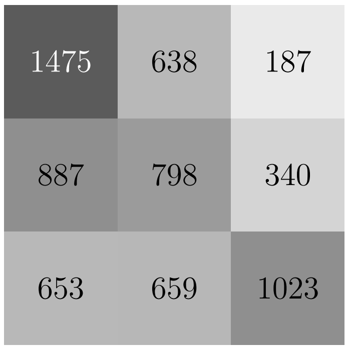
(a) Match minute 0



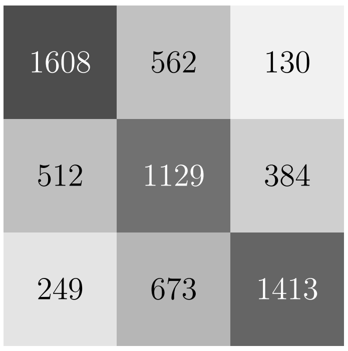
(b) Match minute 15



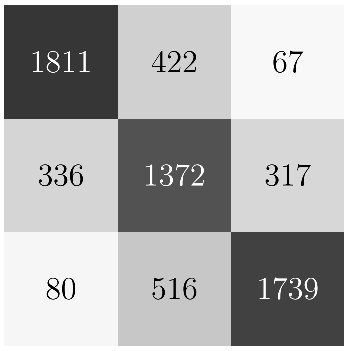
(c) Match minute 30



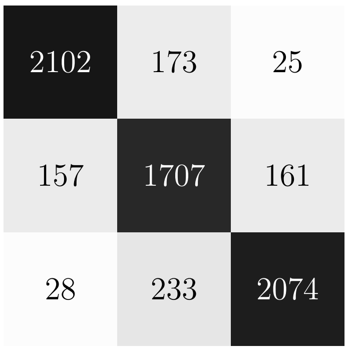
(d) Match minute 45



(e) Match minute 60



(f) Match minute 75



(g) Match minute 90

Figure 4.16: Confusion matrices for match minute 0, 15, 30, 45, 60, 75, and 90 for network case study 7 with two layers of 256 units.

since the amount of data is small. The results presented in Table 4.6 discard this guess since no other larger network than case study 7 outperformed the network in case study 2 at any time.

4.6 Comparison

In this section we present the result from the naive statistical model from Section 3.3, as well as human performance, to get a clear picture of what kind of performance we have achieved.

4.6.1 Naive Statistical Model

The model is tested on matches played after 2016-11-05, which is 20% of the matches in the dataset. After each match in the test set the *TotalScore* for both teams are updated. It was able to reach 46.45% in test accuracy.

4.6.2 Human Accuracy

Since there are no recorded human predictions in our data set, different prediction statistics have been used to get an estimation of how good humans are at predicting the outcome of football games. Only statistics where home winner, away winner or draw predictions have been made have been used for the estimation.

4.6.2.1 Betting Companies

Odds from 5 different betting companies were collected, *A, B, C, D* and *E*. We gathered odds for 5 leagues and 2 tournaments between 2017-04-18 and 2017-05-21, which once is shown in Table 4.7.

League/Tournament	Region
La Liga	Spain
Bundesliga	Germany
Premier League	England
Serie A	Italy
Ligue 1	France
Champions League	Europe
Europa League	Europe

Table 4.7: Leagues and tournaments that odds were gathered for.

For each game we assume that the team with the lowest odds on it is the predicted winner, without taking to account that betting companies can change their odds for economical benefits. All odds are gathered after the lineups for each game is released, which can affect the odds. Table 4.8 show the accuracy and prediction distribution of the betting companies. What matches are selected for each company differs since the data could only be collected between the release of the lineups and the start of the match. The data for some matches could therefore not be collected for all betting companies in time. Equal odds is when the betting company have the lowest odds on two or more outcomes.

Company	Predicted Games	Accuracy	Predictions
A	64	48.44%	Home win: 64.06% Draw: 0.0% Away win: 35.94% Equal odds: 0.0%
B	61	49.18%	Home win: 62.30% Draw: 0.0% Away win: 36.07% Equal odds: 1.64%
C	63	47.62%	Home win: 61.90% Draw: 0.0% Away win: 38.10 % Equal odds: 0.0%
D	62	46.77%	Home win: 61.29% Draw: 0.0% Away win: 38.71 % Equal odds: 0.0%
E	63	50.79%	Home win: 61.90% Draw: 0.0% Away win: 36.51% Equal odds: 1.59%

Table 4.8: Betting companies, their accuracy and prediction distribution.

4.6.2.2 10 Newspapers

Since the 1960s there have been something called 10 Newspapers bets in Sweden (10 tidningars tips)[1]. Sports experts from each newspaper/magazine, both national and local, predict the outcome of 13 games each week, a total of 676 games. The paper with most correct predictions at the end of the year wins. Table 4.9 show the accuracy of the 10 newspapers competing in 2015[2].

[1]`http://www.spelaspel.se/odds/tio-tidningars-tips`
[2]`https://om.svenskaspel.se/2016/01/18/vinnare-tio-tidningars-tips-2015/`

Newspaper	Accuracy
Sydsvenska dagbladet	46.75%
Expressen	46.60%
Aftonbladet	46.45%
Dala-Demokratin	46.45%
Borlänge Tidning	45.86%
Vi Tippa	45.86%
TT	45.71%
Dagens Spel	45.56%
Skånska Dagbladet	45.12%
Dagens Nyheter	43.05%

Table 4.9: Competitors in the 10 Newspapers challenge 2015 and their accuracy.

4.6.2.3 Forza Football Users

The Forza Football application has the possibility for users to predict the outcome of a game. By comparing which outcome for a match that got the highest amount of predictions with the actually outcome we were able to get an insight into the accuracy of football fans. We looked at matches between 2015-06-01 and 2017-04-01. Table 4.10 shows the accuracy depending on how many predictions the games have.

Predictions	Amount of Matches	Accuracy
All matches	83963	47.49%
>100 predictions	40916	50.92%
>500 predictions	17490	52.57%
>1000 predictions	10520	54.34%
>5000 predictions	2858	59.13%
>10000 predictions	1215	59.50%
>20000 predictions	317	55.21%

Table 4.10: Accuracy for Forza Football users predictions. The predictions can be made both before and after the lineups are known.

Since users just predict without any chance of winning money, gut feeling might play a big role in how they predict. They are more likely to predict wins for their favourite teams or against teams that they don't like when there is no reward involved.

4.6.3 Discussion

The human accuracy on predicting the outcome of football matches is difficult to measure. The accuracy differs depending on what matches are predicted. Predictions on different leagues and tournaments give different accuracies and humans predict the outcome on a much smaller set of leagues and tournaments than the network. This makes it hard compare the network against human accuracy.

Other statistical models are made for one or few leagues and can therefore be tuned to work for that specific league or leagues. Therefore it is not entirely fair to compare it with other models.

The human accuracy and the naive statistical model accuracy should just be viewed as a guideline of what peformance the network should have.

4.7 Remarks and Further Discussion

One theory of why the prediction accuracy is not higher is that the networks have problems with learning multi-dimensional representations of the teams and players. This could be solved with having deeper data about the matches. Since each match does not have more information than teams, lineups, goals, assists, penalties, card and substitutions, a lot of leagues have to be used. Since the data set only consists of the 2-3 last seasons for each league and tournament, a lot of different leagues, without any large amount of transfers between them, have to be used to get enough amount of data. If a smaller set of leagues were used, however with more seasons for each league and more data about each match, most of the teams and players would occur more often in the dataset. This would probably require a bigger network to be able to use all the data and to learn more about the players and the teams and thereby make better predictions. Also the only pure positive event for a team during a game in our dataset are goals and penalties. These events include forwards and midfielder a lot more than defenders and goalkeepers. Therefore the network might not realise the importance of which defenders and goalkeepers play in a game.

One important thing to take in consideration when predicting games is the physical and psychological health of the team and the players. These are things that are hard to both gather information about, to put a measure on and use for a computer.

Since there exist few documented predictions from one and the same person over a lot of matches it is hard to measure how good he performance of the network is compared to humans. For the competitions that exist, the competitors are able to concentrate on a few leagues while the network predicts the outcome for matches all over the world. This also increases the difficulty with compareing the network against a human.

5 Conclusion

A LSTM network performs wells on classification and has potential to perform well on predictions of the outcome of football games. Using the proposed LSTM architecture the final test classification accuracy of the outcome was 98.63% for the many-to-one approach and 88.68% for the many-to-many approach. When only teams and lineups are known the prediction accuracy is 34.30% for many-to-one and 45.90% for many-to-many. The more informaion the networks are fed about a match, i.e. the longer an ongoing match is played, the better the network performce on predicting the outcome. At full time many-to-one reached 98.63% and many-to-many 88.68%. This is no suprise since more information should lead to better predictions. However, the increased prediction accuracy over minutes played in a match indicates that the network is able to learn about football.

Future work can be performed on this subject; for example, different data could be used, different inputs and architectures could be tested and other things than winner could be predicted, such as the amout of goals or cards.

Bibliography

[1] Martín Abadi et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems". In: *arXiv preprint arXiv:1603.04467* (2016).

[2] Burak Galip Aslan and Mustafa Murat Inceoglu. "A comparative study on neural network based soccer result prediction". In: *Intelligent Systems Design and Applications, 2007. ISDA 2007. Seventh International Conference on.* IEEE. 2007, pp. 545–550.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *CoRR* abs/1409.0473 (2014). URL: `http://arxiv.org/abs/1409.0473`.

[4] Yoshua Bengio et al. "A neural probabilistic language model". In: *Journal of machine learning research* 3.Feb (2003), pp. 1137–1155.

[5] Antoine Bordes et al. "Learning structured embeddings of knowledge bases". In: *Conference on artificial intelligence.* EPFL-CONF-192344. 2011.

[6] John S Bridle. "Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition". In: *Neurocomputing.* Springer, 1990, pp. 227–236.

[7] Kyunghyun Cho et al. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *CoRR* abs/1406.1078 (2014). URL: `http://arxiv.org/abs/1406.1078`.

[8] Junyoung Chung et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling". In: *arXiv preprint arXiv:1412.3555* (2014).

[9] Vataya Chunwijitra, Ananlada Chotimongkol, and Chai Wutiwiwatchai. "A Hybrid Input-type Recurrent Neural Network for LVCSR Language Modeling". In: *EURASIP J. Audio Speech Music Process.* 2016.1 (Dec. 2016), 93:1–93:12. ISSN: 1687-4714. DOI: `10.1186/s13636-016-0093-x`. URL: `https://doi.org/10.1186/s13636-016-0093-x`.

[10] Ronan Collobert and Jason Weston. "A unified architecture for natural language processing: Deep neural networks with multitask learning". In: *Proceedings of the 25th international conference on Machine learning.* ACM. 2008, pp. 160–167.

[11] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization". In: *Journal of Machine Learning Research* 12.Jul (2011), pp. 2121–2159.

[12] David Geer. "Chip makers turn to multicore processors". In: *Computer* 38.5 (2005), pp. 11–13.

[13] Felix A Gers and Jürgen Schmidhuber. “Recurrent nets that time and count”. In: *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on.* Vol. 3. IEEE. 2000, pp. 189–194.

[14] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. “Learning to forget: Continual prediction with LSTM”. In: *Neural computation* 12.10 (2000), pp. 2451–2471.

[15] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. “Domain adaptation for large-scale sentiment classification: A deep learning approach”. In: *Proceedings of the 28th international conference on machine learning (ICML-11).* 2011, pp. 513–520.

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* `http://www.deeplearningbook.org`. MIT Press, 2016.

[17] Alex Graves. “Long short-term memory”. In: *Supervised Sequence Labelling with Recurrent Neural Networks* (2012), pp. 37–45.

[18] Jordan Gumm, Andrew Barrett, and Gongzhu Hu. “A machine learning strategy for predicting march madness winners”. In: *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2015 16th IEEE/ACIS International Conference on.* IEEE. 2015, pp. 1–6.

[19] Geoffrey Hinton, NiRsh Srivastava, and Kevin Swersky. “Neural Networks for Machine Learning Lecture 6a Overview of mini–batch gradient descent”. In: (2012).

[20] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[21] Sepp Hochreiter et al. *Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies.* 2001.

[22] Roger A Horn. “The hadamard product”. In: *Proc. Symp. Appl. Math.* Vol. 40. 1990, pp. 87–169.

[23] Josip Hucaljuk and Alen Rakipović. “Predicting football scores using machine learning techniques”. In: *MIPRO, 2011 Proceedings of the 34th International Convention.* IEEE. 2011, pp. 1623–1627.

[24] Anito Joseph, Norman E Fenton, and Martin Neil. “Predicting football results using Bayesian nets and other machine learning techniques”. In: *Knowledge-Based Systems* 19.7 (2006), pp. 544–553.

[25] Kamran Karimi, Neil G Dickson, and Firas Hamze. “A performance comparison of CUDA and OpenCL”. In: *arXiv preprint arXiv:1005.2581* (2010).

[26] Douwe Kiela and Léon Bottou. “Learning Image Embeddings using Convolutional Neural Networks for Improved Multi-Modal Semantics”. In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP-14).* Doha, Qatar, 2014.

[27] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980 (2014). URL: `http://arxiv.org/abs/1412.6980`.

[28] David Luebke. “CUDA: Scalable parallel programming for high-performance scientific computing”. In: *Biomedical Imaging: From Nano to Macro, 2008. ISBI 2008. 5th IEEE International Symposium on.* IEEE. 2008, pp. 836–838.

[29] Tomas Mikolov et al. “Distributed Representations of Words and Phrases and their Compositionality”. In: *CoRR* abs/1310.4546 (2013). URL: `http://arxiv.org/abs/1310.4546`.

[30] Tomas Mikolov et al. “Efficient estimation of word representations in vector space”. In: *arXiv preprint arXiv:1301.3781* (2013).

[31] Volodymyr Mnih et al. “Playing atari with deep reinforcement learning”. In: *arXiv preprint arXiv:1312.5602* (2013).

[32] David Silver et al. “Mastering the game of Go with deep neural networks and tree search”. In: *Nature* 529.7587 (2016), pp. 484–489.

[33] Richard Socher et al. “Parsing natural scenes and natural language with recursive neural networks”. In: *Proceedings of the 28th international conference on machine learning (ICML-11)*. 2011, pp. 129–136.

[34] Nitish Srivastava et al. “Dropout: a simple way to prevent neural networks from overfitting”. In: *Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[35] D Steinkraus, I Buck, and PY Simard. “Using GPUs for machine learning algorithms”. In: *Document Analysis and Recognition, 2005. Proceedings. Eighth International Conference on*. IEEE. 2005, pp. 1115–1120.

[36] John E Stone, David Gohara, and Guochun Shi. “OpenCL: A parallel programming standard for heterogeneous computing systems”. In: *Computing in science & engineering* 12.3 (2010), pp. 66–73.

[37] Jaeyong Sung, Ian Lenz, and Ashutosh Saxena. “Deep Multimodal Embedding: Manipulating Novel Objects with Point-clouds, Language and Trajectories”. In: *CoRR* abs/1509.07831 (2015). URL: `http://arxiv.org/abs/1509.07831`.

[38] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. “Sequence to Sequence Learning with Neural Networks”. In: *CoRR* abs/1409.3215 (2014). URL: `http://arxiv.org/abs/1409.3215`.

[39] Peter D Turney and Patrick Pantel. “From frequency to meaning: Vector space models of semantics”. In: *Journal of artificial intelligence research* 37 (2010), pp. 141–188.

[40] DRGHR Williams and Geoffrey Hinton. “Learning representations by back-propagating errors”. In: *Nature* 323.6088 (1986), pp. 533–538.

[41] David A Wilson. “Convolution and Hankel operator norms for linear systems”. In: *IEEE Transactions on Automatic Control* 34.1 (1989), pp. 94–97.

[42] SHI Xingjian et al. “Convolutional LSTM network: A machine learning approach for precipitation nowcasting”. In: *Advances in Neural Information Processing Systems*. 2015, pp. 802–810.

[43] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. “Recurrent Neural Network Regularization”. In: *CoRR* abs/1409.2329 (2014). URL: `http://arxiv.org/abs/1409.2329`.

Appendix A

Raw data

```
{
  "liveData": {
    "card": [
      {
        "contestantId": "a3nyxabgsqlnqfkeg41m6tnpp",
        "lastUpdated": "2017-02-13T20:10:56.553Z",
        "optaEventId": "1381274270",
        "periodId": 1,
        "playerId": "eczc8ttdfagqliov10cuqeuol",
        "playerName": "Y. Toure",
        "timeMin": 10,
        "type": "YC"
      }
    ],
    "goal": [
      {
        "contestantId": "a3nyxabgsqlnqfkeg41m6tnpp",
        "lastUpdated": "2017-02-13T20:29:43.523Z",
        "optaEventId": "1268066259",
        "periodId": 1,
        "scorerId": "e9l5r0txzrjbenli05rn3k4wl",
        "scorerName": "R. Sterling",
        "timeMin": 29,
        "type": "G"
      },
      {
        "contestantId": "1pse9ta7a45pi2w2grjim70ge",
        "lastUpdated": "2017-02-13T21:41:59.188Z",
        "optaEventId": "491505339",
        "periodId": 2,
        "scorerId": "2smmv9hdoiet7necaf00ol51x",
        "scorerName": "T. Mings",
        "timeMin": 69,
        "type": "OG"
    }
  ],
  "lineUp": [
    {
      "contestantId": "1pse9ta7a45pi2w2grjim70ge",
      "player": [
        {
          "firstName": "Artur",
          "lastName": "Boruc",
          "matchName": "A. Boruc",
          "playerId": "eaeh3ogbfsfp0obus2wk66r4l",
          "position": "Goalkeeper",
          "positionSide": "Centre",
          "shirtNumber": 1
        },
        {
          "firstName": "Dan",
          "lastName": "Gosling",
          "matchName": "D. Gosling",
          "playerId": "80y7k4ht4s4nkta9wnf30y40l",
          "position": "Substitute",
          "shirtNumber": 4
        }
      ],
      "teamOfficial": {
        "firstName": "Eddie",
        "id": "ezlkz2b39qic4wu066cgl1o0l",
        "lastName": "Howe",
        "type": "manager"
      }
    },
    {
      "contestantId": "a3nyxabgsqlnqfkeg41m6tnpp",
      "player": [
        {
          "firstName": "Wilfredo Daniel",
          "lastName": "Caballero",
          "matchName": "W. Caballero",
          "playerId": "6bhm9cfkwxv5ojh49rqv11g45",
          "position": "Goalkeeper",
          "positionSide": "Centre",
          "shirtNumber": 13
        },
        {
          "firstName": "Fabian",
          "lastName": "Delph",
            "matchName": "F. Delph",
            "playerId": "9q6c7nbjhgwp2yrelznmk1gb9",
            "position": "Substitute",
            "shirtNumber": 18
          }
        ],
        "teamOfficial": {
          "firstName": "Josep",
          "id": "3r7wi8r14k888ip3y2bzbacid",
          "lastName": "Guardiola i Sala",
          "type": "manager"
        }
      }
    ],
    "matchDetails": {
      "matchLengthMin": 98,
      "matchLengthSec": 29,
      "matchStatus": "Played",
      "period": [
        {
          "end": "2017-02-13T20:50:28Z",
          "id": 1,
          "lengthMin": 50,
          "lengthSec": 16,
          "start": "2017-02-13T20:00:12Z"
        },
        {
          "end": "2017-02-13T21:53:50Z",
          "id": 2,
          "lengthMin": 48,
          "lengthSec": 13,
          "start": "2017-02-13T21:05:37Z"
        }
      ],
      "periodId": 14,
      "scores": {
        "ft": {
          "away": 2,
          "home": 0
        },
        "ht": {
          "away": 1,
          "home": 0
        },
        "total": {
          "away": 2,
          "home": 0
        }
      },
      "winner": "away"
    },
    "missedPen": [],
    "substitute": [
      {
        "contestantId": "a3nyxabgsqlnqfkeg41m6tnpp",
        "lastUpdated": "2017-02-13T20:15:31.422Z",
        "periodId": 1,
        "playerOffId": "d66tmyj9pt69gub8pcjgo3bh1",
        "playerOffName": "Gabriel Jesus",
        "playerOnId": "5ch5dk1z9opa2iwd64kb4e5sl",
        "playerOnName": "S. Aguero",
        "timeMin": 15
      }
    ]
  },
  "matchInfo": {
    "competition": {
      "country": {
        "id": "1fk5l4hkqk12i7zske6mcqju6",
        "name": "England"
      },
      "id": "2kwbbcootiqqgmrzs6o5inle5",
      "name": "Premier League"
    },
    "contestant": [
      {
        "country": {
          "id": "1fk5l4hkqk12i7zske6mcqju6",
          "name": "England"
        },
        "id": "1pse9ta7a45pi2w2grjim70ge",
        "name": "AFC Bournemouth",
        "position": "home"
      },
      {
        "country": {
          "id": "1fk5l4hkqk12i7zske6mcqju6",
          "name": "England"
        },
        "id": "a3nyxabgsqlnqfkeg41m6tnpp",
        "name": "Manchester City",
        "position": "away"
      }
    ],
    "date": "2017-02-13Z",
    "description": "AFC Bournemouth vs Manchester City",
    "id": "9n1lozeiz5rv8atl9hs5cjkyx",
    "lastUpdated": "2017-02-13T22:16:15Z",
    "ruleset": {
      "id": "79plas4983031idr6vw83nuel",
      "name": "Men"
    },
    "sport": {
      "id": "289u5typ3vp4ifwh5thalohmq",
      "name": "Soccer"
    },
    "time": "20:00:00Z",
    "tournamentCalendar": {
      "endDate": "2017-05-21Z",
      "id": "2c1fh40r28amu4rgz0q66ago9",
      "name": "Premier League 2016/2017",
      "startDate": "2016-08-13Z"
    },
    "venue": {
      "id": "8vnu2g41orfaa25n2djgrv4jm",
      "longName": "Vitality Stadium",
      "shortName": "Vitality Stadium"
    },
    "week": "25"
  }
}
```

Appendix B

Leagues and tournaments

League	Region	Seasons
Premier League	England	2015/2016, 2016/2017
Championship	England	2015/2016, 2016/2017
FA Cup	England	2015/2016, 2016/2017
League Cup	England	2015/2016, 2016/2017
La Liga	Spain	2015/2016, 2016/2017
Segunda B	Spain	2015/2016, 2016/2017
Copa Del Rey	Spain	2015/2016, 2016/2017
Bundesliga	Germany	2015/2016, 2016/2017
2 Bundesliga	Germany	2015/2016, 2016/2017
DFB Pokal	Germany	2015/2016, 2016/2017
Ligue 1	France	2015/2016, 2016/2017
Ligue 2	France	2015/2016, 2016/2017
Coupe De France	France	2015/2016, 2016/2017
Serie A	Italy	2015/2016, 2016/2017
Serie B	Italy	2015/2016, 2016/2017
Copa Italia	Italy	2015/2016, 2016/2017
Primeira Liga	Portugal	2015/2016, 2016/2017
Segunda Liga	Portugal	2015/2016, 2016/2017
Taca Da Liga	Portugal	2015/2016, 2016/2017
Premier League	Russia	2015/2016, 2016/2017
2 Devision	Russia	2015/2016, 2016/2017
Cup	Russia	2015/2016, 2016/2017
Premier League	Ukraine	2015/2016, 2016/2017
Druha Liga	Ukraine	2015/2016, 2016/2017
Cup	Ukraine	2015/2016, 2016/2017
First Division A	Belgium	2015/2016, 2016/2017
First Division B	Belgium	2015/2016, 2016/2017
Cup	Belgium	2015/2016, 2016/2017
Super Lig	Turkey	2015/2016, 2016/2017
2 Lig	Turkey	2015/2016, 2016/2017
Cup	Turkey	2015/2016, 2016/2017

League	Region	Seasons
First League	Czech Republic	2015/2016, 2016/2017
Super League	Switzerland	2015/2016, 2016/2017
Eredivisie	Netherlands	2015/2016, 2016/2017
Super League	Greece	2015/2016, 2016/2017
Bundesliga	Austria	2015/2016, 2016/2017
1. HNL	Croatia	2015/2016, 2016/2017
Liga 1	Romania	2015/2016, 2016/2017
Superliga	Denmark	2015/2016, 2016/2017
Premier League	Belarus	2015/2016, 2016/2017
Ekstraklasa	Poland	2015/2016, 2016/2017
Allsvenskan	Sweden	2015, 2016, 2017
Major League Soccer	USA	2016, 2017
Serie A	Brazil	2016, 2017
Liga MX	Mexico	2015/2016, 2016/2017
Primera Division	Argentina	2016, 2016/2017
Liga Alef	Israel	2015/2016, 2016/2017
Championship	Scotland	2015/2016, 2016/2017
Primera Division	Uruguay	2015/2016, 2016/2017
A League	Australia	2015/2016, 2016/2017
Super League	China	2015, 2016, 2017
J1 League	Japan	2015, 2016, 2017
Super Liga	Slovakia	2015/2016, 2016/2017
Fl1 active cup	Liechtenstein	2015/2016, 2016/2017
NB1	Hungary	2015/2016, 2016/2017
Divizia Nationala	Moldovia	2015/2016, 2016/2017
1 Deild	Iceland	2015, 2016, 2017
Veikkausliiga	Finland	2015, 2016, 2017
Superliga	Albania	2015/2016, 2016/2017
Premier Division	Republic of Ireland	2015, 2016, 2017
Premier Liga	Bosnia and Herzegovina	2015/2016, 2016/2017
Erovnuli liga	Georgia	2015/2016, 2016, 2017
Virsliga	Latvia	2015, 2016, 2017
First league	Macedonia	2015/2016, 2016/2017
Esiliiga A	Estonia	2015, 2016, 2017
First League	Montenegro	2015/2016, 2016/2017
Premier League	Armenia	2015/2016, 2016/2017
National Division	Luxembourg	2015/2016, 2016/2017
A Lyga	Lithuania	2015, 2016, 2017
Premier League	Malta	2015/2016, 2016/2017
Premier League	Wales	2015/2016, 2016/2017
1 Deild	Faroe Islands	2015, 2016, 2017
Premier Division	Gibraltar	2015/2016, 2016/2017
1 Divisio	Andorra	2015/2016, 2016/2017
Campionato	San Marino	2015/2016, 2016/2017
Champions League	Europe	2015/2016, 2016/2017
Europa League	Europe	2015/2016, 2016/2017
Club World Cup	Worldwide	2015/2016, 2016/2017