INSTITUT FÜR INFORMATIK
Datenbanken und Informationssysteme

Universitätsstr. 1 D–40225 Düsseldorf

Heinrich Heine Universität Düsseldorf

Vorhersage von Fußballergebnissen mittels Deep Learning Ansätzen

Fabian Niklas Constant

Masterarbeit

Beginn der Arbeit:	29. Dezember 2020
Abgabe der Arbeit:	25. Juni 2021
Gutachter:	Prof. Dr. Stefan Conrad
	Prof. Dr. Gunnar W. Klau

Erklärung

Hiermit versichere ich, dass ich diese Masterarbeit selbstständig verfasst habe. Ich habe dazu keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Düsseldorf, den 25. Juni 2021

Fabian Niklas Constant

Zusammenfassung

Sport zählt zu den populärsten Interessensgebieten der Bevölkerung und ist ein zunehmender Wirtschaftsfaktor. Allein in Deutschland werden im Jahr ca 70. Milliarden Euro im Sportbereich ausgegeben. Davon werden bis zu vier Milliarden Euro in Sportwetten gesetzt, wovon 55 Prozent im Fußball getätigt werden.

Diese Arbeit beschäftigt sich mit der Vorhersage von Fußballergebnissen mit Hilfe von *Deep Learning* Ansätzen. Hierbei wird ein *Long Short-Term Memory*- und ein Transformer Modell genutzt. Der Transformer wurde zuvor in diesem Kontext noch nicht verwendet.
Deep Learning ist ein Teilbereich des *Machine Learnings*. Dabei werden künstliche neuronale Netze benutzt. Diese sind Algorithmen für Clustering-, Klassifizierungs- und Regressionsprobleme. In dieser Arbeit werden die Spielergebnisse in die Klassen: Sieg, Unentschieden und Niederlage vorhergesagt.
Das *Long Short-Term Memory* Modell gehört zum Bereich der rekurrenten neuronalen Netze. Beim Transformer handelt es sich um ein Modell, welches auf dem *Attention Mechanism* aufbaut und auf rekurrente Strukturen verzichtet.

Um Spielvorhersagen treffen zu können, wird zunächst ein Datensatz erstellt, der mittels unterschiedlicher Quellen und Informationen entsteht. Dabei wird darauf eingegangen, welche Sportinformationen wie gesammelt werden. Zudem wird die Zusammensetzung des Datensatzes erläutert.

Das *Long Short-Term Memory*- und das Transformer Modell werden einzeln in ihrer Architektur beschrieben. Außerdem wird erklärt, wie der Datensatz als Eingabe und Ausgabe fungiert und eingelesen wird. Es handelt sich bei der Eingabe nämlich um zwei unterschiedliche Informationszugaben. Zum einen werden Spielstatistiken als Zeitreihen erstellt und als Sequenz modelliert und zum anderen Zusatzinformationen zu den jeweiligen Mannschaften hinzugefügt. Abschließend werden bei beiden Modellen die Hyperparameter untersucht.

Um die Modelle zu trainieren, werden unterschiedliche Trainingssätze erstellt und anschließend mittels eines Testsatzes geprüft. Die Ergebnisse werden mit unterschiedlichen Evaluationsmethoden analysiert und verglichen.
Es stellte sich heraus, dass der Transformer in allen Punkten besser abschneidet und mit einer *Accuracy* von 69 Prozent Fußballspiele korrekt vorhersagt.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Sport gehört zu den beliebtesten Interessensgebieten der Bevölkerung und ist zudem als Wirtschaftsfaktor von zunehmender Bedeutung. Private Haushalte in Deutschland geben beispielsweise jährlich ca. 70 Milliarden Euro im Sportbereich aus. Davon sind 80 Prozent im aktiven Sektor und 20 Prozent im passiven Bereich.[1]
Zum passiven Bereich gehört die Sportwette, bei der die Deutschen bis zu vier Milliarden Euro jährlich setzen. (Meyrahn et al., 2014) Die populärste Sportart der Welt ist der Fußball mit etwa 3,5 Milliarden Fans [2], davon gibt es 48,25 Millionen in Deutschland [3]. 55 Prozent aller Sportwetten werden im Fußball getätigt (Meyrahn et al., 2014).
Die Entscheidung, welche Mannschaft ein Spiel gewinnen wird, ist für verschiedene Interessensgruppen bereits im Vorfeld wichtig. Bei Sportwetten etwa werden enorme Geldbeträge verwendet. Hierbei sind Buchmacher, Fans und potenzielle Bieter daran interessiert, wie die Quoten eines Spiels geschätzt werden (Ulmer et al., 2013). Durch die Spielvorhersage und die Quoten zu einem Spiel kann ein Buchmacher entscheiden, ob auf dieses Spiel gewettet werden sollte. Bei Sportvorhersagen wird in der Regel die Klasse, bestehend aus Sieg, Niederlage und Unentschieden, vorhergesagt (Prasetio et al., 2016).
Die Menge an sportbezogenen Daten, die online zur Verfügung stehen, nehmen zu. Dadurch kann eine große Anzahl von *Features* verwendet werden, um die Gewinn- oder Verlustchancen bevorstehender Spiele zu treffen.
Diese Arbeit beschäftigt sich mit Sportvorhersagen und verwendet dabei *Deep Learning* Algorithmen. *Deep Learning* ist ein Teilbereich des *Machine Learnings*, bei dem künstliche neuronale Netze verwendet werden. Die Verwendung der Techniken der neuronalen Netze ist für diesen Zweck ein neueres Studiengebiet. Diese Techniken haben sich bei Ableitungen hochgenauer Klassifikationsmodelle in anderen Bereichen als effektiv erwiesen (Mohammad et al., 2016).
Im *Machine Learning* ist die Klassifikation das am häufigsten benutzte Verfahren (Abdelhamid et al., 2012; Mohammad et al., 2015). Hierbei wird verfolgt, eine Zielvariable vorherzusagen, indem ein Klassifikationsmodell basierend auf einem Trainingssatz erstellt und zur Vorhersage des Wertes der Klasse von Testdaten benutzt wird (Witten et al., 2005). Dieses Vorgehen wird auch als *supervised Learning* bezeichnet (Norvig und Intelligence, 2002).

1.2 Zielsetzung

Um Spielergebnisse beim Fußball vorhersagen zu können, werden in dieser Arbeit Fußballdaten aus der 1. Bundesliga verwendet. Diese ist die höchste Spielklasse des deutschen Fußballs und besteht aus 18 Mannschaften. Es werden die Saisons der Jahre 2015/2016 bis 2019/2020 betrachtet. Hierbei werden Informationen von Fußballspielern,

[1] https://www.bmwi.de/Redaktion/DE/Textsammlungen/Branchenfokus/Wirtschaft/branchenfokus-sportwirtschaft.html
[2] https://www.topendsports.com/world/lists/popular-sport/fans.htm
[3] https://de.statista.com/statistik/daten/studie/727901/umfrage/fussball-fans-in-deutschland-nach-alter/

deren Mannschaften und Spielstatistiken herangezogen.
Diese Informationen werden in zwei verschiedene Eingaben aufgeteilt. Zum einen werden Spielstatistiken aus mehreren Spieltagen als Zeitreihen verwendet und als Sequenz modelliert. Zum anderen werden aktuelle Zusatzinformationen zusätzlich in die Modelle eingespeist. In der Regel verfügen diese Modelle lediglich über eine Eingabe.
Bei den Modellen handelt es um das *Long Short-Term Memory* - und um das Transformer Modell, im Folgenden auch als *Long Short-Term Memory* bzw. Transformer abgekürzt. Das *Long Short-Term Memory* ist ein Modell aus den rekurrenten neuronalen Netzen und hat die Fähigkeit eine Art Erinnerung an frühere Erfahrungen wie ein Kurzzeitgedächtnis zu bilden. Der Transformer ist ein Modell, welches mit Aufmerksamkeitsoperationen einen Kontext zwischen den Daten herstellt.
Das Ziel ist eine Vorhersagegenauigkeit der Spielergebnisse zu haben, die mit den von Wettunternehmen konkurrieren könnten.

1.2.1 Problemstellung

In dieser Arbeit soll untersucht werden, ob man *Deep Learning* zur Vorhersage von Fußballergebnissen verwendet werden kann. Dabei wird Folgendes analysiert:

- Können *Long Short-Term Memory* - bzw. Transformer Modelle für Spielvorhersagen verwendet werden?
- Wann sind die Fußballvorhersagen der Ergebnisse während der Saison am besten?
- Bei welcher Mannschaft werden die Spielergebnisse am häufigsten korrekt vorausgesagt?
- Welches Modell eignet sich besser zu Spielvorhersagen?

1.3 Gliederung

Nach der Einleitung in Kapitel 1 befasst sich das Kapitel 2 mit verwandten Arbeiten, die sich mit der Vorhersage von Fußballspielergebnissen mittels Techniken aus dem *Machine Learning* oder *Deep Learning* beschäftigen. Danach werden in Kapitel 3 die Modelle und die dazu gehörenden grundlegenden Begriffe bzw. Techniken vorgestellt. Anschließend wird die Erstellung des Datensatzes in Kapitel 4 beschrieben. Dabei werden die einzelnen Quellen und Extrahierungen der Informationen beschrieben. In Kapitel 5 wird das *Long Short-Term Memory* Modell in Bezug auf diese Arbeit erklärt. Dabei wird die Architektur und das Einbinden der Daten erläutert. Das Kapitel 6 beschreibt analog zu Kapitel 5 den Transformer. In Kapitel 7 folgt die Evaluation. Einleitend wird der Datensatz, die eingesetzten Evaluationsmaße und Ergebnisse der einzelnen Modelle vorgestellt. Das abschließende Kapitel 8 umfasst das Fazit, in dem die Arbeit rekapituliert, die Problemstellungen beantwortet und ein Ausblick auf mögliche Weiterentwicklungen gewährt wird.

2 Verwandte Arbeiten

Die verwandten Arbeiten beschäftigen sich mit Forschungen, die einen vergleichbaren Kontext zum Thema dieser Arbeit haben. Dabei werden Arbeiten betrachtet, die sich ebenfalls mit dem Vorhersagen von Fußballergebnissen in einer Liga mittels *Deep Learning* - bzw. *Machine Learning* Ansätzen beschäftigen.

2.1 Predicting The Dutch Football Competition Using Public Data: A Machine Learning Approach

Diese Forschungsarbeit (Tax und Joustra, o.D.) beschäftigt sich mit Fußballvorhersagen in der niederländischen Liga. Hierbei werden Naive Bayes, *Neural Networks*, *Random Forest* und *Genetic Programming* benutzt.
Der Datensatz besteht aus Mannschaftsinformationen, wie etwa die vorherigen Spielergebnisse, der Durchschnittsanzahl von Toren oder der Wettquoten. Diese Informationen werden kombiniert.
Die höchste *Accuracy* liegt hierbei bei 56 Prozent mittels LogitBoost. LogitBoost ist ein Algorithmus des maschinellen Lernens.

2.2 Football Result Prediction by Deep Learning Algorithms

In dieser Forschungsarbeit von Samba (Samba, 2019) werden Fußballvorhersagen mittels *Multilayer Perceptron* erstellt. Dies ist eine Klasse von *Feedforward Artificial Neural Network*.
Der Datensatz besteht aus zwölf Saisons von jeweils vier verschiedenen Fußballligen in den Jahren 2006 bis 2017. Hierbei handelt es sich um die erste und zweite Liga aus England, der Premier League und der EFL Championship, und der ersten und zweiten Liga aus Frankreich, der League 1 und League 2. Die *Features* bestehen für ein Spiel aus den verschiedenen Wettquoten, dem Marktwert der Mannschaften und verschiedenen Fußballstatistiken, wie zum Beispiel aus Toren, Punkten und Torschüssen. Diese Statistiken werden kumulativ für die letzten fünf Spiele gesammelt.
Das erfolgreichste Ergebnis war eine *Accuracy* von 60 Prozent in der Premier League. Zudem wurden noch die meisten richtig vorhergesagten Mannschaften ermittelt. Das waren mit 84 Prozent Manchester City in der Saison 2017/2018 und mit 79 Prozent FC Chelsea in der Saison 2016/2017.

2.3 Football Match Prediction using Deep Learning

In dieser Arbeit (Pettersson und Nyquist, 2017) werden Fußballvorhersagen anhand eines Long Short-Term Memory Modells erstellt. Es werden Vorhersagen eines Spiel nach verschiedenen Minuten im Spiel durchgeführt.
Die Informationen des Datensatzes basieren auf 54 unterschiedlichen Fußballligen verschiedener Länder. Die *Features* bestehen aus der Startaufstellungen jeder Mannschaft und der Position jedes Spielers. Zudem werden Informationen der Torschützen, Torvorlagengeber, Spieler, die eine gelbe oder rote Karte bekommen haben, Spieler, die eingewechselt werden, und die Elfmeterschützen gesammelt.

Ein Fußballergebnis wird alle 15 Minuten vorhergesagt, also sieben Mal während des Spiels und zusätzlich nach Spielende. Es stellte sich heraus, dass die Ergebnisse besser wurden, je länger ein Spiel andauerte. Die Ergebnisse liegen in einem Bereich von 43 bis 88 Prozent während des Spiels und bei 98 Prozent nach einem Spiel.

2.4 A Sure Bet: Predicting Outcomes of Football Matches

Die Forschungsarbeit (Sebastien Goddijn, 2018) aus Stanford hat Spielvorhersagen mit einem Long Short-Term Memory Modell für die Premier League gebildet.
Dafür wurde ein Datensatz aus Informationen von den Fußballspielern und Spielstatistiken erstellt. Die Spielerinformationen bestehen aus der Position und Statistiken jedes Spielers. Diese Statistiken beinhalten beispielsweise die Offensiv-, Defensiv- und Flankenwerte. Zudem wird noch die Summe der letzten Siege, Niederlagen und Unentschieden für jede Mannschaft gesammelt.
Hierbei ergab sich ein Wert von 47 Prozent im Test für die Vorhersagen der Ergebnisse.

2.5 Transformer

Eine Forschungsarbeit, in der ein Transformer zur Fußballvorhersage erstellt wurde, gibt es meines Wissens zur Zeit nicht.

3 Grundlagen

Dieses Kapitel handelt von den Grundlagen, die benötigt werden, um einen Transformer bzw. ein Long Short-Term Memory Modell verstehen zu können. Es werden verschiedene Techniken und ein Teilbereich der künstlichen Intelligenz vorgestellt.

3.1 Machine Learning

Maschine Learning verfolgt das Ziel, ein zu erstellendes Modell mithilfe von bekannten Daten zu lernen, um anschließend Vorhersagen für unbekannte Daten zu treffen. Diese Algorithmen lassen sich in zwei Verfahren einteilen, das *Supervised* - und *Unsupervised Learning*.

3.1.1 Supervised Learning

Supervised Learning kann in zwei Vorhersagemethoden angwendet werden: in die Klassifizierung und die Regression.
Zur Realisierung des *Supervised Learnings* (Norvig und Intelligence, 2002) wird ein Datensatz benötigt, welches aus der Eingabe und dessen Zielvariablen besteht. Die Eingabe besteht aus Merkmalen, auch *Features* genannt. Die Ausgabe beschreibt die Vorhersage, die auch *Labels* genannt wird. Mit diesem Datensatz können überwachte Lernalgorithmen trainiert und ein Modell erzeugt werden. Ein Algorithmus findet zwischen der Eingabe und dem *Label* eine Beziehung, die mit der richtigen Ausgabe korreliert. Auf dieser Grundlage können mit neuen Eingaben Ausgaben vorhersagt werden. Das Ziel des überwachten Lernens ist, das richtige *Label* für neue Eingabedaten vorherzusagen.

Der Algorithmus kann mit folgender Formel beschrieben werden:

$$y = f(x) \tag{1}$$

y ist die vorhergesagte Ausgabe, die durch die Abbildungsfunktion bestimmt wird und den Eingabewert x einer Klasse zuordnet.

Für die **Klassifizierung** werden Datenpunkte und festgelegte Klassen benötigt. Bei den festgelegten Klassen handelt es sich um diskrete Werte. Die Aufgabe des Algorithmus ist es, den Eingabewert in eine Klasse auf Grundlage des Trainings zuzuweisen.

Die **Regression** hat das Ziel einen Zusammenhang zwischen Datenpunkten zu erstellen. Dabei ist die lineare Regression die am häufigsten verwendete Form. Hierbei wird eine Gerade zwischen unabhängigen Variablen gezogen und somit versucht, abhängige Variablen zu erklären.

3.1.2 Unsupervised Learning

Der Unterschied zum *Supervised Learning* liegt in den gegebenen Daten für das Training und dem Ziel des Algorithmus. Die Daten bestehen anstatt aus der Eingabe und dessen

korrektem *Label*, nur aus der Eingabe. Ziel des Algorithmus ist das Erkennen von Korrelationen innerhalb der Daten und auf dieser Basis Datenpunkte zu gruppieren.
Mit *Unsupervised Learning* (Hinton, Sejnowski et al., 1999) werden Muster bzw. Untergruppen innerhalb des Datensatzes gefunden.
Ein Anwendungsfall ist hierfür das Clustering. Beim Clustering werden Daten in Clustern gruppiert. In der Regel werden zwischen den Daten Distanzmaße, wie etwa die euklidische Distanz, benutzt und jeweils dem Cluster zugeordnet, bei dem der Datenfall die maximale Ähnlichkeit hat.

3.2 Künstliches neuronales Netz

Durch künstliche neuronale Netzwerke besteht die Möglichkeit, mit Funktionsabläufen, Strukturen in Abhängigkeit eines Funktionsziels zu erlernen, die dem menschlichen Gehirn ähneln.
Künstliche neuronale Netzwerke sind Algorithmen für Clustering-, Klassifizierungs- und Regressionsprobleme. Diese werden zum Erlernen nichtlinearer komplexer Hypothesen genutzt, wenn der Datensatz beispielsweise zu groß ist und zu viele Merkmale vorhanden sind. (Wilson, 1998-2012)

3.2.1 Aufbau

Künstliche neuronale Netzwerke bestehen aus mehreren Schichten, die aus mehreren Neuronen bestehen. Als Schichten gibt es *input-* und *output layer* und einen oder mehrere *hidden layers*.

Künstliche Neuronen
Künstliche Neuronen können durch folgende Formel (Cevik und Guzelbey, 2008) dargestellt werden:

$$y = \phi \sum_{i=1}^{n} w_i * x_i \tag{2}$$

Dabei steht x_i für die **Eingaben** und w_i für das **Gewicht** der einzelnen Eingaben. Die Eingaben und Gewichte werden paarweise multipliziert. Die Produkte werden addiert und bilden die **Übertragungsfunktion** $\sum_{i=1}^{n}$, die die Netzeingabe des Neurons berechnet. Anschließend wird die **Aktivierungsfunktion** ϕ mit der Netzeingabe aufgerufen, die den Schwellenwert darstellt.
Ein einzelnes künstliches Neuron hat, wie beim Axon eines biologischen Neurons, mehrere Eingänge, *inputs*, aber nur einen Ausgang, *output*. Der Ausgang kann sowohl der Eingang des nächsten Neurons, als auch der letzte Ausgang des Netzwerkes sein.
Die einfachste Form eines künstlichen neuronalen Netzwerkes ist das ***Feedforward neural network***. In diesem Modell werden die Informationen in einer Richtung verarbeitet und durchlaufen dabei mehrere verborgene Knoten. Jeder Knoten beschreibt ein künstliches Neuron. Die Informationen bewegen sich dabei niemals rückwärts. (Haykin, 1999)

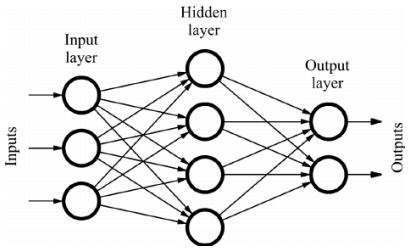


Abbildung 1: Beispiel für ein Feedforward Neural Network

Die Aktivierungsfunktion bestimmt, ob ein Knoten aktiviert wird oder nicht. Diese Funktion kann die Leistung des künstlichen Neurons in Abhängigkeit von der Eingabe erhöhen oder verringern. Nur wenn die Eingabe einen Schwellenwert überschreitet, wird ein höherer Wert ausgegeben, sonst wird sie deaktiviert. Die Hauptaufgabe der Aktivierungsfunktion ist, das Signal im Knoten in ein Ausgangssignal umzuwandeln.
Ein Beispiel für eine Aktivierungsfunktion ist *Rectifier*, auch **ReLU** (Nair und Hinton, 2010) genannt.

$$y = \begin{cases} 0 & x < 0 \\ x & \geq 0 \end{cases} \tag{3}$$

Hierbei wird der maximale Wert zwischen Null und dem Eingabewert x genommen. Der Eingabewert x stellt sich selber dar, wenn dieser größer gleich Null ist. Ansonsten ist der Eingabewert negativ und wird auf Null gesetzt. Diese Aktivierungsfunktion tritt in Kraft, wenn der Eingabewert positiv ist.

Ein weiteres Beispiel ist die **Softmax-Funktion** (Goodfellow et al., 2016).
Die Softmax-Funktion wird meist bei der letzten Schicht verwendet, wenn es sich um Wahrscheinlichkeiten, also um eine multinomiale logistische Regression handelt. In dieser Aktivierungsfunktion wird eine Softmax-Funktion verwendet.

$$P(y = j|x) = \frac{e^{x^T * w_j}}{\sum_{k=1}^{K} e^{x^T * w_k}} \tag{4}$$

Bei der Eingabe handelt es sich um einen Vektor und bei der Ausgabe um Wahrscheinlichkeiten zu jeder möglichen Ausgabe. Die Summe der Ausgaben ergibt immer Eins.

Die Eingabeschichten erhalten die Informationen von außen, verarbeiten diese und geben sie gewichtet weiter.
In den *hidden layers* werden die Daten erneut gewichtet und zum nächsten Neuron bis zur Ausgabeschicht weitergeleitet.
Als Aktivierungsfunktion für den *input* - und den *hidden layer* wird zum Beispiel die ReLU-Funktion verwendet. Für den *output layer* wird bei vielen Klassen die Softmax-Aktivierungsfunktion verwendet.
Die Gewichtung findet in jeder Ebene statt. Der Prozess der Informationsveränderung ist

nicht sichtbar, daher wird diese Schicht auch *hidden layer* genannt. Im *output layer* befinden sich die resultierenden Entscheidungen. (Grossi und Buscema, 2007)

3.3 Rekurrentes neuronales Netz

Rekurrente neuronale Netze (Sherstinsky, 2020), im Folgenden auch RNN genannt, sind Untergruppen der künstlich neuronalen Netzen. In diesen Netzwerken sind Zyklen zugelassen, mit denen Verbindungen zwischen Neuronen in einer Schicht oder Neuronen einer Schicht mit ihrem Vorgänger ermöglicht werden. Dadurch können Rückkopplungen aus den vorherigen Ergebnissen gebildet und als "Gedächtnis"gedeutet werden.
Die Vorhersagen werden nicht nur von den Eingaben, die während des Algorithmus gewichtet werden, beeinflusst, sondern auch von *hidden states*. Diese stellen den Kontext, basierend auf früheren Ein- bzw. Ausgaben, dar. Bei einem RNN handelt es sich um ein Netz, welches Schleifen benutzt, um die Zeitschritte einer Sequenz zu durchlaufen, was ermöglicht, Informationen aufrecht zuhalten.

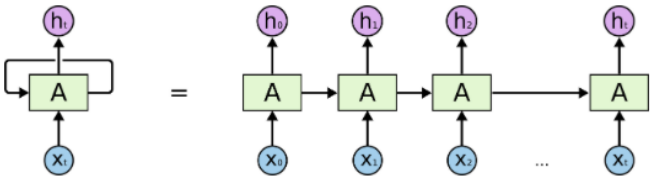


Abbildung 2: Struktur eines RNNs

Dieses Netz besteht aus einer Wiederholung von Schichten, die immer gleich aufgebaut sind. x_t ist die Eingabe für den Zeitschritt t. A ist der *hidden state*, der das "Gedächtnis" des Netzwerkes ist. Dieser wird auf Grundlage der Eingabe und des vorherigen *hidden states* berechnet. Am Anfang wird der vorherige *hidden layer* mit Nullen initialisiert, da dieser noch nicht berechnet wurde. h_t ist die Ausgabe für den Zeitschritt t.
Der Nachteil dieses "Gedächtnisses" ist, dass nur auf aktuellere Informationen zurückgegriffen werden kann. Dadurch können Informationen, die vor einer großen Anzahl vorheriger Zeitschritte entdeckt wurden, nicht mehr verbunden werden.
Ein weiterer Nachteil bei RNNs wird mit den *Vanishing Gradients* (Hochreiter, 1998) beschrieben. Das Modell lernt beim Training durch Veränderungen der Parameterwerte. Sind diese Änderungen jedoch sehr klein, kann das Modell nicht effektiv lernen.

3.4 Long Short-Term Memory Modell

Long Short-Term Memory Modelle (Hochreiter und Schmidhuber, 1997), im Folgenden auch LSTM genannt, sind eine spezielle Kategorie von RNNs, die das oben stehende Problem lösen. Diese sind in der Lage langfristige Abhängigkeiten zu lernen. Das Erinnern von Informationen über längere Zeiträume hinweg ist das Standardverhalten. LSTMs haben ebenfalls eine kettenartige Struktur, die sich aber in den Wiederholungseinheiten unterscheiden. Eine Einheit besitzt vier Netzwerkschichten.

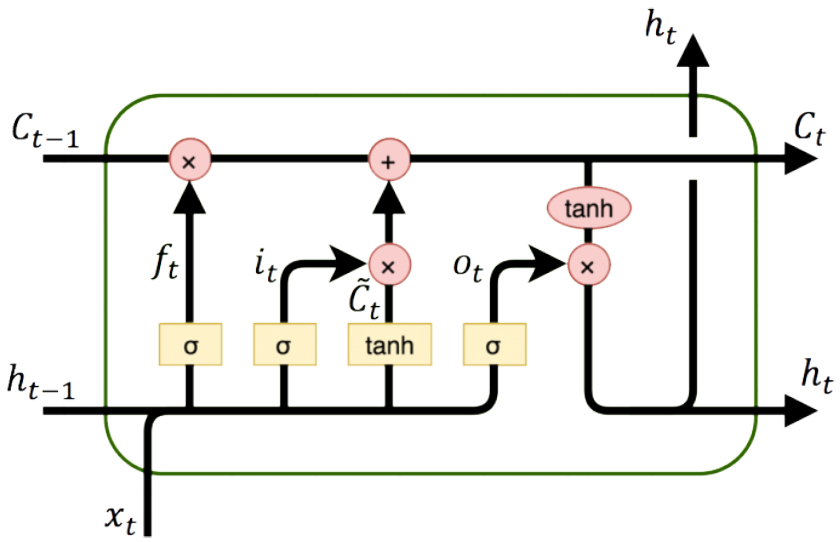


Abbildung 3: Struktur einer LSTM Einheit

Das Wichtigste des LSTMs ist der Zellzustand, der durch die obere Linie von C_{t-1} bis C_t in der Abbildung 3 dargestellt wird. Der Zellzustand arbeitet gradlinig den Prozess ab und hat nur wenige lineare Interaktionen.
Das LSTM hat die Fähigkeit Informationen aus dem Zellzustand, C_{t-1}, mit Hilfe des vorherigen *hidden state* h_{t-1} und der Eingabe x_t, zu entfernen bzw. hinzuzufügen. Dies wird im unterem Teil der Abbildung dargestellt. Die Entscheidung, ob Informationen verworfen oder hinzugefügt werden, regeln die *Gates* σ. Diese haben die Möglichkeiten, optionale Informationen durchzulassen. Die *Gates* bestehen jeweils aus einer sigmoidalen neuronalen Netzschicht und punktweisen Multiplikationsoperationen. Eine Sigmoid-Schicht gibt Zahlen zwischen Null und Eins aus, die festlegen, wie viel von jeder Komponente durchgelassen werden sollen.

Ein LSTM hat drei dieser *Gates*, um den Zellzustand zu regulieren.

3.4.1 Der Ablauf des LSTMs

Den Ablauf des LSTMs kann man in vier Schritte aufteilen:

1. Als Erstes wird untersucht, welche Informationen verworfen werden können. Das wird mithilfe des ersten *Gates* durchgeführt, auch *„forget gate layer“* genannt. Hierfür werden die vorherige Zellvorhersage h_{t-1} und die Eingabe x_t für den Zeitschritt t verwendet. Es wird eine Zahl zwischen Null und Eins für jeden Zellzustand in C_{t-1}, dem vorherigen Zellzustand, berechnet. Dabei wird bei einer Eins die Information vollständig behalten und bei einer Null vollständig verworfen. Wenn es noch keinen vorherigen Zellzustand gibt, werden statt dessen Nullen benutzt.

$$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f) \tag{5}$$

2. Im nächsten Schritt wird die Entscheidung getroffen, welche Informationen gespeichert werden sollen. Diesen Prozess kann man in zwei Teile zerlegen:

   Das folgende *Gate*, dieses Mal „*input gate layer*“ genannt, entscheidet, welche Werte aktualisiert werden sollen.

   $$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i) \tag{6}$$

   Die tanh-Schicht erstellt einen Vektor mit neuen Kandidatenwerten $\tilde{C}_t$, die dem Zellzustand hinzugefügt werden sollen.

   $$\tilde{C}_t = tanh(W_c * [h_{t-1}, x_t] + b_C \tag{7}$$

3. Der dritte Schritt beinhaltet die Aktualisierung des alten in den neuen Zellzustand. Zuvor wurden alle Entscheidungen getroffen, welche Informationen verworfen oder gespeichert werden sollen. Dies wird nun in die Tat umgesetzt.
   Der vorherige Zellzustand wird mithilfe der Funktion f_t multipliziert, die angibt, welche Informationen vergessen werden sollen.
   Danach wird das Produkt aus Schritt zwei, $i_t * \tilde{C}_t$, das die Kandidatenwerte darstellt, skaliert mit dem Wert, den man für die Aktualisierung der einzelnen Zustandswerte bestimmt hat, dazu addiert.

   $$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{8}$$

4. Zum Abschluss wird die Ausgabe basierend auf dem Zellstatus berechnet, der ein weiteres Mal gefiltert wird.
   Zunächst wird das letzte *Gate* aufgerufen, das entscheidet, welche Informationen des Zellstatus ausgegeben werden sollen. Danach werden die Zellzustandswerte durch die Funktion tanh zwischen -1 und 1 normalisiert und mit dem Wert des letzten *Gates* multipliziert.
   Somit bestehen lediglich die Daten weiter, die ausgegeben werden sollen.

   $$o_t = \sigma(W_o * [h_{t-1}, x_t] + b_o) \tag{9}$$

   $$h_t = o_t * tanh(C_t) \tag{10}$$

(Kamath et al., 2019)

3.5 Encoder-Decoder Modelle

Beim Encoder-Decoder Modell (Goodfellow et al., 2016) handelt es sich um ein künstliches neuronales Netz. Dieses hat die Aufgabe Eingabedaten zu komprimieren und zu kodieren. Mit diesen reduzierten Daten wird eine Ausgabe konstruiert, die den Eingabedaten entspricht. Der Encoder dient zur Reduzierung der Daten. Das Modell lernt hierbei, das Rauschen zu ignorieren, um die Eingabedimension zu reduzieren und die Eingabedaten in eine kodierte Form zu komprimieren. Der Decoder wird zur Rekonstruktion genutzt. Hierbei lernt das Modell, wie es mit der kodierten Darstellung die Daten wieder

rekonstruiert.
Encoder-Decoder Modelle (Cho et al., 2014) werden häufig zur Vorhersage neuronaler maschineller Übersetzungen oder zur Vorhersage von Sequenz zu Sequenz verwendet, wie etwa dem Übersetzen von Sätzen. Das Modell besteht beispielsweise aus zwei rekurrenten neuronalen Netzen, um die Eingabe in einer Vektordarstellung zu kodieren und diese in eine andere Darstellung zu dekodieren. Dabei haben die Eingabe und die Ausgabe eine feste Größe, die aber nicht gleich groß sein müssen. Das bekannteste Modell ist das *Sequence to Sequence Model*.

Ein Encoder-Decoder Modell besteht aus einem Encoder und einem Decoder.

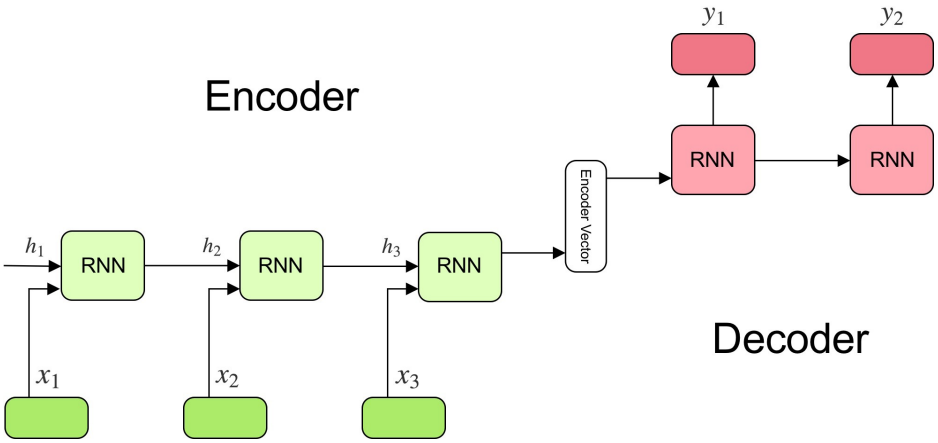


Abbildung 4: Struktur eines Encoder-Decoder Modells

3.5.1 Encoder

Der Encoder wandelt die Werte der Eingabesequenz in einen Vektor fester Länge um. In jedem Zeitschritt wird die Eingabe x und der vorherige *hidden state* h_{t-1} empfangen und einzeln gewichtet. Dabei relevante Information werden gesammelt und ein neuer *hidden state* erzeugt.

$$h_t = f(W^{(hh)} * h_{t-1} + W^{(hx)} * x_t) \tag{11}$$

Ein Encoder kann aus einem Stapel von RNNs bestehen. Dadurch kann das Modell den Kontext und zeitliche Abhängigkeiten der Sequenzen verstehen. Die Ausgabe ist der letzte *hidden state* des Encoders, der Encoder Vektor. Darin sind die nützlichsten Eingangsinformationen enthalten, die auch als Eingabe für den Decoder fungieren, um die besten Ergebnisse zu erzielen.

3.5.2 Decoder

Ein Decoder erhält den Encoder Vektor und hat die Aufgabe, eine neue Sequenz mithilfe des Vektors zu erstellen. Der Aufbau ist dem des Encoders sehr ähnlich. Der Decoder kann ebenfalls aus RNN-Schichten bestehen. Dabei nimmt jede rekurrente Einheit einen *hidden state* von der vorherigen Einheit und produziert einen eigenen.

$$h_t = f(W^{(hh)} * h_{t-1}) \tag{12}$$

Die Ausgabe des aktuellen Zeitschrittes wird mit einer Softmax-Funktion ermittelt, um die Wahrscheinlichkeit für jede mögliche Ausgabe zu erhalten. Hierbei wird der aktuelle *hidden state* betrachtet.

3.6 Attention Mechanism

Der *Attention Mechanism* (Luong et al., 2015) ist ein Verfahren, der *Sequence to Sequence Models* verbessert. Ein Encoder komprimiert die ganze Eingabe in einem einzigen Vektor. Dabei kann es schwierig werden alle Informationen zu sammeln, da die Eingabe beliebig groß sein kann. Dieser Vektor ist die einzige Darstellung der Daten, die der Decoder sieht. Wenn nun der Decoder versucht die Eingabe zu rekonstruieren, kann es sein, dass für den Decoder in jedem Generierungsschritt einige Teile der Quelle wichtiger sind als andere. Jedoch besitzt der Decoder nur die eine feste Darstellung um alle relevanten Informationen zu extrahieren. Das Problem soll der *Attention Mechanism* lösen.
Beim *Sequence to Sequence Model* werden die Zwischenzustände des Encoders verworfen, was kein Nachteil ist, wenn es sich um kleine Sequenzen handelt.
Die Idee ist nun, diese Zwischenzustände ebenfalls zu benutzen, um **Kontextvektoren** zu konstruieren, die der Decoder benötigt, um bessere Ausgangssequenzen zu erzeugen. Somit kann dieser in jedem Decoderschritt auf einen bestimmten Zustand des Encoders zugreifen und selektiv bestimmte Elemente aus der Sequenz verwenden, um das *Label* zu erzeugen. Bei diesem Mechanismus gibt es zwei Punkte: den *Alignment Vektor* und den Kontextvektor. Diese werden nun genauer betrachtet.

3.6.1 Alignment Vektor

Der *Alignment Vektor* (Bahdanau et al., 2014) hat die Aufgabe, die Werte zu gewichten, auf die sich der Decoder bei jedem Zeitschritt konzentrieren soll. Dieser Vektor hat die gleiche Länge wie die Eingabe und wird in jedem Zeitschritt des Decoders berechnet. Jeder Wert zeigt die Gewichtung eines Wertes innerhalb der Eingabe an. Es wird eine Matrix mit Hilfe dieses Vektors und der Eingabesequenz erstellt, die *Align*.
Es gibt drei Wege die *Alignment* Werte zu berechnen. Diese Berechnungen verwenden die Encoder *Labels* und den *hidden state* des Decoders.

$$score(h_t, \bar{h}_s) = \begin{cases} h_t^T \bar{h}_s & dot \\ h_t^T W_a \bar{h}_s & general \\ v_\alpha^T tanh(W_\alpha [h_t; \bar{h}_s]) & concat \end{cases} \tag{13}$$

Dot
: *Dot* ist die Multiplikation des *hidden state* des Encoders mit dem *hidden state* des Decoders.

General
: *General* berechnet auf ähnliche Weise wie *Dot* die *Alignment* Werte, es wird jedoch eine Gewichtsmatrix einbezogen.

Concat
: Der *hidden state* des Encoders und des Decoders werden addiert. Danach wird die Summe durch eine lineare Schicht mit einer tanh-Aktivierungsfunktion durchlaufen und am Ende mit einer Gewichtsmatrix multipliziert.

Die *Alignment* Werte werden durch die Softmax-Funktion in einen Wertebereich zwischen Null und Eins umgewandelt.

3.6.2 Kontextvektor

Der Decoder benötigt den Kontextvektor (Bahdanau et al., 2014), um den nächsten Wert in der Sequenz vorherzusagen. Hierfür werden die zuvor berechneten *Attention Scores* mit den *Labels* des Encoders multipliziert. Durch die Zunahme der *Attention Scores* werden nun die Zustände gewichtet und es wird angezeigt, wie viele relevante Informationen sich im jeweiligen Zustand befinden.

$$c_t = \sum_s a_{ts} * \bar{h}_s \tag{14}$$

Im Decoder werden die *Alignment* Werte und der Kontextvektor berechnet, der mit den *hidden state* verkettet wird.

3.7 Transformer

Encoder-Decoder Modelle haben große Einschränkungen bei der Arbeit mit langen Sequenzen, weil die Informationen der ersten Elemente verloren gehen. Dieses Problem kann man mit dem *Attention Mechanism* lösen.
Im Decoder werden in jedem Schritt alle Zustände des Encoders betrachtet. Der Mechanismus extrahiert die Informationen aus der gesamten Sequenz und bildet eine gewichtete Summe aller vergangenen Encoder Zustände. Somit kann man für jedes Element der Ausgabe ein bestimmtes Element der Eingabe dessen Wichtigkeit zuordnen.
Der Decoder lernt in jedem Schritt sich auf die wichtigen bzw. richtigen Elemente der Eingabe zu konzentrieren, um die nächste Vorhersage zu treffen.
Bei einem großen Korpus ist dieser Mechanismus sehr zeitaufwendig und rechnerisch ineffizient. Dies soll der Transformer verbessern.
Der Transformer (Vaswani et al., 2017) ist ein Modell, welches Merkmale für jeden Wert mit *Self-Attention mechanism* extrahiert. Dabei wird untersucht, wie wichtig andere Werte in der Eingabe in Bezug auf diesen Wert sind.
Beim Transformer werden keine rekurrenten Einheiten verwendet, sondern nur gewichtete Summen und Aktivierungen. So kann das Modell parallel und effizient arbeiten. Dieses Verfahren nennt man *autoregressive* (Gregor et al., 2014). Hierbei werden nicht die vorherigen Erkenntnisse im *hidden state* bereitgestellt, sondern als weitere Eingabe in das Modell eingegeben.

3.7.1 Hauptbestandteile des Transformers

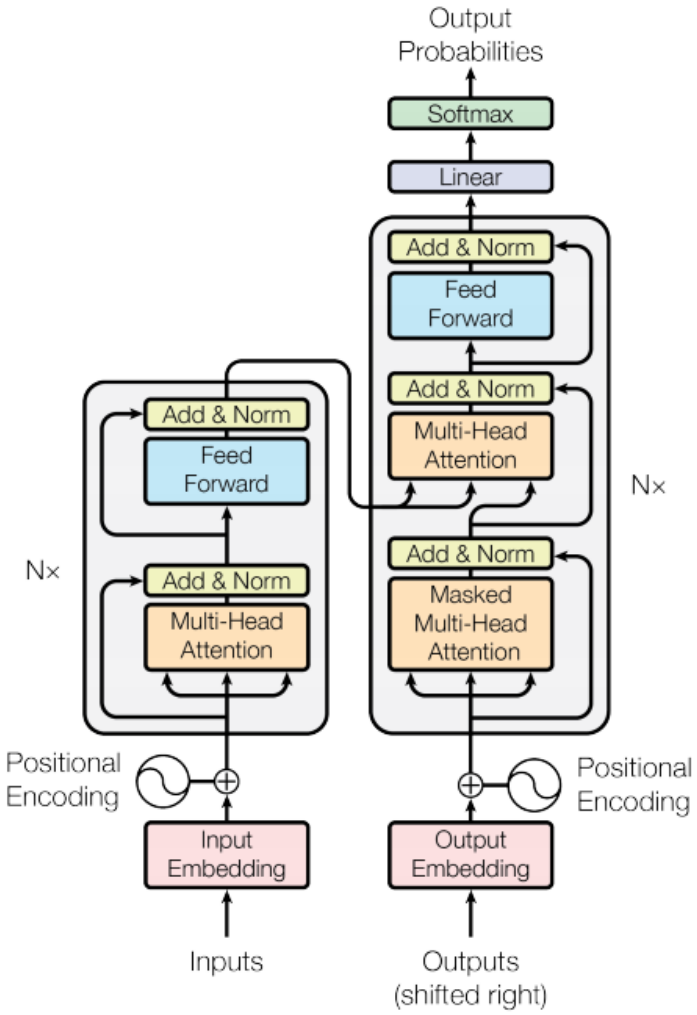


Abbildung 5: Beispiel für einen Transformer

Auf der linken Seite der Abbildung 5 ist ein Encoder-Netzwerk zu sehen und auf der rechten Seite ein Decoder-Netzwerk. Beide haben einen Kernblock aus einem *Attention-* und *Feedforward network*. Der Ablauf der Blöcke kann sich wiederholen.
Bevor der Aufbau weiter erläutert wird, werden die Kernblöcke detailliert beschrieben.

Self-Attention Mechanism
Der *Self-Attention Mechanism* ist ein *Attention Mechanism*, der in einer einzelnen Sequenz verschiedene Positionen in Beziehung setzt, um eine Darstellung derselben Sequenz zu berechnen.
Dafür wird ein gewichteter Durchschnitt über die Sequenz berechnet. Für die Berechnung des Durchschnitts werden drei Matrizen eingeführt: *Query*, *Values* und *Key*, auch

abgekürzt mit Q, V und K. Die *Query*, *Values*, *Key* und das *Label* sind Matrizen. Das *Label* ist eine gewichtete Summe der *Values*, wobei das Gewicht für jeden *Value* durch eine Kompatibilitätsfunktion der *Query* und dem entsprechenden *Key* berechnet wird.
Für diese neu eingeführten Begriffe benötigt man drei Gewichtsmatrizen, K, Q und V, der Dimension k x k und drei lineare Transformationen für jedes x_i:

$$Q_i = W_q * x_i, K_i = W_k * x_i, V_i = W_v * x_i \tag{15}$$

Die drei Matrizen stammen von der gleichen Eingabe. Dadurch kann der *Attention Mechanism* des Eingabevektors auf sich selbst angewendet werden, der *Self-Attention Mechanism*.

Scaled Dot-Product Attention
Als Nächstes werden die *Attention Values* berechnet. Damit wird berechnet, wie stark der Fokus auf andere Stellen der Eingabesequenz in Bezug auf einen Wert an einer bestimmten Position gelegt wird.
Dies wird mit Hilfe der Matrizen Q, K und V und der Dimension d_k durchgeführt. Es wird das Punktprodukt der *Query* mit allen *Keys* des jeweiligen Wortes bewertet. Im nächsten Schritt wird dieses Produkt durch die Quadratwurzel von d_k dividiert und darauf eine Softmax-Funktion angewendet, um die Gewichtung der Werte zu erhalten. Abschließend multipliziert man die Matrix V hinzu, um die Werte der Wörter zu behalten, auf die man sich konzentrieren soll. Zudem werden die Werte der irrelevanten Informationen minimiert bzw. entfernt.

$$Attention(Q, K, V) = softmax(\frac{Q * K^T}{\sqrt{d_k}}) * V \tag{16}$$

Multi-Head Attention
Die *Attention Values* werden im *Self-Attention Mechanism* auf den gesamten Satz fokussiert. Allerdings würde hierbei das gleiche Ergebnis herauskommen, wenn die Reihenfolge der Werte unterschiedlich ist. Es soll aber die Aufmerksamkeit auf verschiedene Segmente der Wörter gerichtet werden.
Der *Multi-Head Attention Mechanism* erzeugt verschiedene Ausgabematrizen von Werten. Es werden mehrere *Heads* benutzt, weil es dadurch möglich ist, Informationen aus verschiedenen Darstellungsunterräumen an verschiedenen Positionen gemeinsam zu bearbeiten. Der *Self-Attention Head* kann dies durch die Mittelwertbildung nicht. Die Ausgabematrizen werden in einer Matrix zusammengesetzt, da die *Feedforward* Schicht nur eine Matrix erwartet. Es wird eine Verkettung mit dem Produkt aus den Ausgabematrizen des *Self-Attention Mechanism*, Q, K und V mit deren Gewichtsmatrizen angewendet. Diese Matrix enthält Informationen aus allen Aufmerksamkeitsköpfen.

$$\begin{aligned} MultiHead(Q, K, V) &= Concat(head_1, ..., head_n)W^O, \\ where\ head_i &= Attention(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{17}$$

Positional Encoding
Das *Positional Enconding* dient zur Repräsentation der Position der Werte in der Sequenz und fügt diese Kodierung zur Einbettung, also der Eingabe, hinzu. Dies wird mit einer Sinus und Cosinus Funktion realisiert.

$$\begin{aligned} PE_{(pos,2i)} &= sin(pos/10000^{2i/d_{model}}) \\ PE_{(pos,2i+1)} &= cos(pos/10000^{2i/d_{model}}) \end{aligned} \tag{18}$$

In diesen Formeln beschreibt *pos* die Position des Wertes und i die Dimension. d_{model} ist die Dimensionsgröße des Modells. Es wird eine Sinusfunktion verwendet, da man so auf Sequenzlängen extrapolieren kann, die länger sind, als jede Sequenz im Training.

3.7.2 Der Aufbau

Der Encoder
Die Eingabe wird in eine *Input Embedding* eingefügt. Anschließend wird ein *Positional Encoding* hinzugefügt. Als nächstes folgt der Encoder, der aus einer Schicht mit zwei Unterschichten besteht und mehrmals ausgeführt werden kann.
Der erste Teil dieser Unterschichten ist der *Multi-Head Attention Mechanism*, gefolgt von einer *Feedforward* Schicht.
Beide Unterschichten besitzen eine Weiterleitung, die *residual connection*, in der Abbildung 5 "Add & Norm ", die den Ausgang der Schicht mit dem Eingang summiert und normalisiert. Durch die Weiterleitung werden die positionsbezogenen Informationen beibehalten, die vorher im *input Embedding* hinzugefügt wurden. Vor jeder Weiterleitung wird eine Regularisierung angewendet.
Bevor die Eingabe dazu addiert und die Werte normalisiert werden, wendet man einen *Dropout* an. Diese Methode schaltet eine Anzahl von Neuronen aus. In diesem Fall sind es zehn Prozent, die im nächsten Berechnungsschritt nicht beachtet werden.

Der Decoder
Der Decoder ist ähnlich wie der Encoder aufgebaut. Allerdings unterscheiden sich die Eingaben von Encoder und Decoder, da beim Decoder die jeweiligen *Labels* der Eingaben des Encoders die jetzigen Eingaben sind. Der Hauptteil hat keine zwei Unterschichten, sondern drei. Die erste Schicht ist eine *Masked Multi-Head Attention*. Diese *Multi-Head Attention* ist maskiert, damit verhindert wird, dass die Positionen nicht von nachfolgenden Positionen abhängen. Dies wird realisiert, indem die Werte, die im *hidden state*, also innerhalb des *Scaled Dot-Product Attention* wären, auf $-\infty$ gesetzt wird.
In der mittleren Schicht, namens "*Encoder-Decoder Attention*", wird die *Multi-Head Attention* über das *Label* des Decoders durchgeführt. Der *Key* und der *Value* stammen aus dem *Label* des Encoders, die *Query* kommt aus der vorherigen Schicht.
Diese Schicht erlaubt jeder Position im Decoder, alle Positionen in der Eingabesequenz zu beachten. Die letzte Schicht ist das *Feedforward*.
Am Ende des Decoders transformiert eine lineare Schicht die gestapelten Decoder zu einem vollständigen Netzwerk. Dieser Vektor ist größer und dessen Werte werden *Logits*

genannt. Die Softmax Schicht wandelt die *Logits* in Wahrscheinlichkeiten um, und die Zelle mit der höchsten Wahrscheinlichkeit wird ausgewählt.
(Vaswani et al., 2017)

4 Datensatz

Der Datensatz wird benötigt, um die Eingaben und *Labels* für die Modelle zu erzeugen. Um ihn für diese Arbeit zu erstellen, werden verschiedene Fußballinformationen aus der 1. Bundesliga gesammelt. Die 1. Bundesliga besteht aus den 18 besten Mannschaften Deutschlands. Diese spielen jeweils in 34 Spieltagen zweimal gegeneinander: einmal heim und einmal auswärts. Für diese Spieltage werden Statistiken für jede Mannschaft erhoben. Die Statistiken können in Spielstatistiken, Mannschaftsinformationen und Tabelleninformationen unterteilt werden. Sie dienen zur Erstellung des Datensatzes. Die Statistiken befinden sich auf verschiedenen Webseiten.

Die Wahl der Webseiten wird in Abhängigkeit von Struktur und Vertrauenswürdigkeit getroffen. Einerseits muss man die Webseite parsen und crawlen können und anderseits sollte der Inhalt korrekt geschildert werden. Deshalb fällt die Wahl auf die Webseiten www.kicker.de, www.fifaindex.com, www.oddsportal.com und www. football-data.co.uk.

4.1 Grundlagen des Datensatzes

Kicker
Der Kicker ist ein Sportmagazin, das seit 1920 zweimal wöchentlich Reportagen, Hintergrundberichte, Interviews und Spielanalysen zu Sportereignissen veröffentlicht. [4] Der Schwerpunkt liegt dabei aber auf dem Fußball. Seit 1997 erscheint zusätzlich zum Magazin die Webseite www.kicker.de, in der Sportneuigkeiten aktuell online gestellt werden.[5] Magazin und Website erreichen wöchentlich fünf Millionen Leser und Nutzer.

Fifaindex
Fifa ist eine Videospielserie von EA Canada, die es seit über 20 Jahren gibt und eine Fußballsimulation ist [6] [7]. Das erste Spiel aus der Videospielserie wurde 1993 mit dem Namen "FIFA INTERNATIONAL SOCCER" veröffentlicht[8]. Diese Videospielreihe ist das größte Sport-Video Spiel-Franchise [9].
Anhand realer Spieldaten erstellt EA für das Videospiel Fifa Bewertungen zu den Mannschaften und deren Spielern und aktualisiert diese fortlaufend[10]. Außerdem wird der Spielstil aller tatsächlichen Mannschaften beschrieben, ob zum Beispiel Fortuna Düsseldorf auf Konter oder Ballbesitz spielt.
Die Webseite FIFA Index sammelt Informationen, wie Spielerwerte, Mannschaftswerte, Transferbudget, Spielerrollen und Spielstil von jeder Mannschaft der Videospielserie Fifa.
Die Spielerwerte bestehen aus dem Namen, dem Alter, der Position, dem aktuellen Ge-

[4] https://www.olympia-verlag.de/marken/kicker/
[5] Kontaktaufnahme mit Kicker Redaktion Digital, Tobias Zuber, Community-Manager, am 25.05.2018
[6] https://www.ea.com/about
[7] https://www.ea.com/de-de/games/fifa
[8] https://ogdb.eu/index.php?section=title&titleid=1102
[9] https://www.ea.com/de-de/games/fifa
[10] https://www.ea.com/de-de/games/fifa/fifa-20/ratings/fifa-20-player-ratings-top-100

samtwert, dem Potential des Spielers und weiteren Details über ihn. Dabei liegt der Gesamtwert und das Potential zwischen einem Wert von 1 und 99, wobei 99 der beste Wert ist.
Die Mannschaftswerte bestehen aus einem Defensiv-, Mittelfeld- und Offensivwert. Diese Werte werden seit 2012 während der Saison aktualisiert, wobei sich die Anzahl der Aktualisierungen in den letzten Jahren auf bis zu 48 erhöht hat.

Oddsportal
Seit ihrer Einführung im Oktober 2008 [11] enthält die Webseite Oddsportal Quotenvergleiche von 19 verschiedenen Sportarten, darunter auch Fußball. Die Quoten werden von über 60 globalen und über 20 lokalen Buchmachern gesammelt, einzeln angezeigt und ein Durchschnitt von ihnen berechnet. Die Spielseiten und Quoten werden alle 15 Sekunden aktualisiert. Die Webseite hat zudem noch ein Archiv, welches die Quoten der vorherigen Saisons sammelt.[12]

Football-data
Die Webseite Football-data sammelt, wie die Webseite Oddsportal, Wettquoten von Buchmachern und speichert diese zudem in CSV Dateien ab. Die Veröffentlichung war im Jahre 2001, und die Webseite erhebt ihre Daten bei den Wettanbietern Betbrain, Betbase und Oddsportal. Allerdings wird die Seite Betbase.info heute nicht mehr benutzt.[13]
Die Wettquoten gibt es für die Bundesliga seit der Saison 2000/2001 bis zur aktuellen Saison 2020/2021.

Aus diesen Webseiten werden verschiedene Daten für jede Fußballmannschaft und jedes Spiel gesammelt. Von der Webseite Kicker werden die Aufstellungen, die Statistiken und die Tabellenwerte erhoben. Die Aufstellungen bestehen aus elf Spielernamen, den Statistiken aus 13 Spielwerten und die Tabellenwerte aus fünf Werten. Beispielsweise bestehen die Spielwerte aus Torschüssen oder dem Ballbesitz einer Mannschaft und die Tabellenwerte aus dem Tabellenplatz oder der Tordifferenz.
Von der Webseite FIFA Index werden die Mannschafts- und Spielerwerte gesammelt.
Aus den Webseiten Oddsportal und Football-Data werden die Wettquoten zu jedem Spiel genommen.
Die Wahl dieser Daten wurde anhand der verwandten Arbeiten, siehe Kapitel 2, getroffen und selbst erweitert.

4.2 Erstellung der einzelnen Features

Für das Parsen und Crawlen der einzelnen Webseiten benötigt man außer den Standardbibliotheken die Bibliotheken Beautiful Soup und Pickle. Mit Hilfe von Beautiful Soup

[11] Kontaktaufnahme mit Oddsportal Support, Jan Bartonicek, am 31.03.2021
[12] https://www.oddsportal.com/faq/
[13] Kontaktaufnahme mit Football-data, Joseph Buchdahl, am 31.03.2021

werden die HTML-Dokumente geparst und mit pickle die Daten gespeichert. Im Folgenden wird beschrieben, welche Daten wie und auf welche Webseite gesammelt werden.

Tabellenwerte
Die Tabellenwerte werden auf der Webseite Kicker gesammelt. Dafür wird jeder Spieltag einzeln aufgerufen, also 34 pro Saison. Auf diesen Seiten werden die Tabellen in einer Grafik dargestellt und für alle 18 Mannschaften die aktuellen Tabellenwerte gespeichert. Die Tabellenwerte bestehen aus dem Tabellenplatz, der Anzahl der Siege, der Anzahl der Unentschieden, der Anzahl der Niederlagen und der Differenz der geschossenen Tore zu den Gegentoren.

Mannschafts- und Spielerwerte
Die Mannschafts- und Spielerwerte werden mit Hilfe von FIFA Index geparst. Hierfür werden Links für die jeweilige Saison gesammelt, sowie die für alle Mannschaften dieser Saison. Jede Mannschaft enthält eine Vielzahl von Links, die nach Kalendertagen sortiert werden. Für jedes Datum werden die Mannschafts- und Spielerwerte gesammelt und separat auf der Seite angezeigt. Die Mannschaftswerte sind in einer Grafik namens Team-Information gelistet, in der die Rivalenmannschaft, das Transferbudget und die Defensiv-, Mittelfeld- und Offensivwerte beschrieben sind. Die Spielerwerte sind in einer weiteren Grafik gespeichert, in der die Position, der Name, die Gesamtwertung, das Potential, die Lieblingsposition und das Alter aller Spieler aus der Stammelf, sowie der Reserve- und Ersatzspieler aufgeführt sind.

Für die Vorhersagen werden aber nur der Spielername und sein Gesamtwert benötigt. Das jeweilige Datum wird separat in einer extra Liste mitgespeichert und zusätzlich in das Format, TT:MM, umgewandelt. Anschließend werden die Daten vor und nach der Saison entfernt, da diese außerhalb der Saison liegen und deswegen nicht relevant sind.
Im nächsten Schritt werden die richtigen Kalenderdaten separiert und dafür von jedem Spieltag das Anfangsdatum über Kicker geparst. Nun werden die Kalendertage von FIFA Index mit den Kickerdaten verglichen und das Datum und dessen Index unmittelbar vor oder am gleichen Tag separiert.

Aufstellungen
Die Aufstellungen der Mannschaften werden mit den vorhandenen Spielerwerten von FIFA Index und der Webseite Kicker erstellt. Hierfür werden alle Links für jeden Spieltag pro Saison von Kicker genommen. Für jedes Spiel werden für beide Mannschaften die jeweiligen Aufstellungen gesammelt und von Torwart zu Angreifer schematisch nach der Position sortiert.
Man hat pro Saison zwei Dictionaries mit den Aufstellungen von Kicker und den Spielerwerten von FIFA Index. Der Kicker nennt nur die Spieler der Startaufstellung, also elf Werte. Fifa Index besitzt jedoch alle Spieler und dessen Werte, also mehr als elf Werte. Somit werden die Dictionaries von Kicker und Fifa Index gegenüber gestellt und für jeden Spieler aus den Kickerwerten die passenden Werte aus FIFA Index gesucht. Diese Werte sind durch die sortierte Auflistung im Kicker von Torwart zu Angreifer für

beide Mannschaften sortiert.

Statistiken
Die Statistiken werden erneut auf der Webseite von Kicker geparst. Auch hier werden alle Spieltage pro Saison gesammelt und jedes Spiel aufgerufen. Die Statistikwerte sind von der Heim- und Gastmannschaft getrennt aufgeführt. Allerdings sind ab dem Jahr 2020 die Statistikwerte in einer anderen Form im Seitenquelltext gelistet, sodass ab diesem Jahr etwas anders vorgegangen werden muss. Die Statistiken beider Mannschaften sind weiterhin getrennt aufgelistet, aber dieses Mal sind sie in einem extra *Tag* gruppiert. Als Statistikwerte gibt es die Tore, die Torschüsse, die Laufleistung, die gespielten Pässe, die angekommenen Pässe, die Fehlpässe, die Passquote, den Ballbesitz, die Zweikampfquote, die Fouls, die Anzahl der gefoulten Spieler und die Anzahl von Abseits und Ecken.

Wettquoten
Die Wettquoten werden mit Hilfe von Oddsportal und Football-Data gesammelt. Allerdings wird für die Website noch ein WebDriver benötigt, um auf die Seite von Oddsportal mehrmals in kurzer Zeit zugreifen zu können.
Die Spiele mit den Quoten einer Saison werden auf mehreren Seiten dargestellt. Da es für jedes Spiel Quoten von mehreren Buchmachern gibt, gibt es bei Oddsportal auch Durchschnittsquoten für jedes Spiel. Diese werden gesammelt und haben die Form: [Wettquote für einen Sieg der Heimmannschaft, Wettquote für ein Unentschieden und Wettquote für einen Sieg der Gastmannschaft]
Allerdings werden die Spiele nicht nach Spieltagen sortiert, sondern nach Datum. Wenn es in einer Saison Nachholspiele gibt, müssen diese extra umpositioniert werden.
Wenn auf Oddsportal keine Wettquoten für ein bestimmtes Spiel gegeben sind, können sie auch über Football-Data in einer CSV-Datei gezogen werden.
Die Form, in der die Wettquoten gesammelt werden, lautet: [Spieltagsnummer, Heimmannschaft, Gastmannschaft und die Wettquoten von diesem Spiel].

IDs der Mannschaften
Die IDs der Mannschaften werden benötigt, um beim Erstellen der Trainings- und Testsets die Daten zu den richtigen Mannschaften zuordnen zu können. Dies wird mit Hilfe der Tabellenwerte erstellt.

4.3 Aufbau und Idee des Datensatzes

Diese Daten erstellen alle Informationen für einen Spieltag einer Mannschaft. Insgesamt sind es ohne die Wettquoten 32 Werte. Eine Saison enthält 34 Spieltage mit 18 Mannschaften, also besteht ein Spieltag aus neun Spielen, jeweils mit zwei Mannschaften. Ein Spiel besteht demnach aus 67 Werten, den Werten von der ersten und zweiten Mannschaft und den Wettquoten aus diesem Spiel. Zusätzlich wird noch das Ergebnis, mit drei Werten und IDs der zwei Mannschaften gefüllt. Somit hat ein Spiel insgesamt 72 *Features*.

Die ersten 67 Werte bilden den *Input*, die nächsten drei Werte den *Output* und die letzten zwei Werte, die IDs, dienen der Zuordnung der Spiele der jeweiligen Mannschaften.

Der *Input* wird zudem in zwei Teile geteilt. Ein Teil enthält die Daten für die Zeitschritte und der andere Teil die Zusatzdaten. Die Zeitreihen sind die Daten, die in die LSTM Schicht bzw. in den Encoder des Transformers eingelesen werden. Dabei werden die Zeitreihen als Sequenzen verwendet. Die Zusatzinformationen werden erst nach den Berechnungen hinzugefügt. Dies soll die Vorhersage der Ergebnisse verbessern.

Nummer	Features	Quelle
1	Tabellenplatz	Kicker
2	Anzahl der Siege	Kicker
3	Anzahl der Niederlagen	Kicker
4	Anzahl der Unentschieden	Kicker
5	Tordifferenz	Kicker
6	Gesamtwertung des Angriffs	Fifaindex
7	Gesamtwertung des Mittelfeldes	Fifaindex
8	Gesamtwertung der Verteidigung	Fifaindex
9	Wertung von Spieler 1	Fifaindex
10	Wertung von Spieler 2	Fifaindex
11	Wertung von Spieler 3	Fifaindex
12	Wertung von Spieler 4	Fifaindex
13	Wertung von Spieler 5	Fifaindex
14	Wertung von Spieler 6	Fifaindex
15	Wertung von Spieler 7	Fifaindex
16	Wertung von Spieler 8	Fifaindex
17	Wertung von Spieler 9	Fifaindex
18	Wertung von Spieler 10	Fifaindex
19	Wertung von Spieler 11	Fifaindex
20	Tore	Kicker
21	Torschüsse	Kicker
22	Laufleistung	Kicker
23	gespielte Pässe	Kicker
24	angekommene Pässe	Kicker
25	Fehlpässe	Kicker
26	Passquote	Kicker
27	Ballbesitz	Kicker
28	Zweikampfquote	Kicker
29	Foul	Kicker
30	Gefoult worden	Kicker
31	Abseits	Kicker
32	Ecken	Kicker

Tabelle 1: Daten einer Mannschaft eines Spiels

Nummer	Features	Quelle
1	Heimsieg	Kicker
2	Unentschieden	Kicker
3	Auswärtssieg	Kicker
4	Wettquote für die Heimmannschaft	Oddsportal/Football-data
5	Wettquote für ein Unentschieden	Oddsprtal/Football-data
6	Wettquote für die Auswärtsmannschaft	Oddsprtal/Football-data
7	ID der Heimmannschaft	-
8	ID der Auswärtsmannschaft	-

Tabelle 2: Daten eines Spiels für beide Mannschaften

4.4 Vektoraufbau

Nachdem alle *Features* gesammelt wurden, können sie nun pro Saison zusammengeführt werden. Dafür wird eine dreidimensionale Matrix mit den Dimensionen 18, 34 und 72 erstellt. Die 18 steht für die Mannschaften, die 34 für die Spieltage und die 72 für die *Features* und die IDs der Mannschaft.
Der Vektor der *Features* ist eine Zusammenstellung der einzelnen Daten aus den beiden spielenden Mannschaften.
Bei der Befüllung der Matrix wird mannschaftsweise vorgegangen. Für jeden Spieltag werden die Daten der Mannschaft und dessen Gegnern genutzt und der Vektor mit den Daten gefüllt. Dabei muss bei jedem Spiel beachtet werden, ob die aktuelle Mannschaft die Heimmannschaft oder die Gastmannschaft ist. Außerdem werden die Tabellenwerte vom vorherigen Spieltag genommen, da diese erst nach dem Spieltag erstellt werden. Somit gibt es die Tabellenwerte, bevor der Spieltag beginnt. Beim ersten Spieltag werden die Werte durch Nullen ersetzt, weil sich alle Mannschaften auf demselben Platz befinden, da es noch keinen Spieltag gab. Somit kann mit den Tabellenwerten noch nicht gearbeitet werden und die Matrix wird mit Nullen gefüllt, damit diese nicht berücksichtigt werden.
Da die Mannschaftsnamen in mehreren Variationen geschrieben werden können, wird eine Vergleichsmethode hinzugefügt. Dies geschieht zum Beispiel bei der Mannschaft Bayer 04 Leverkusen, die auch nur Leverkusen oder Bayer 04 genannt werden kann. Die Vergleichsmethode überprüft die Schreibweise mit der Liste aller Mannschaften und gibt das Team mit dem höchsten Ähnlichkeitswert zurück. Somit kann auch mit verschiedenen Schreibweisen, die gleiche Mannschaft angesprochen werden. Diese Methode wird benötigt, da verschiedene Quellen benutzt werden.
Nachdem die Werte der 34 Spieltage von der ersten Mannschaft zusammengestellt sind, werden die Daten für die anderen 17 Mannschaften nacheinander erhoben und die Vektoren gefüllt.

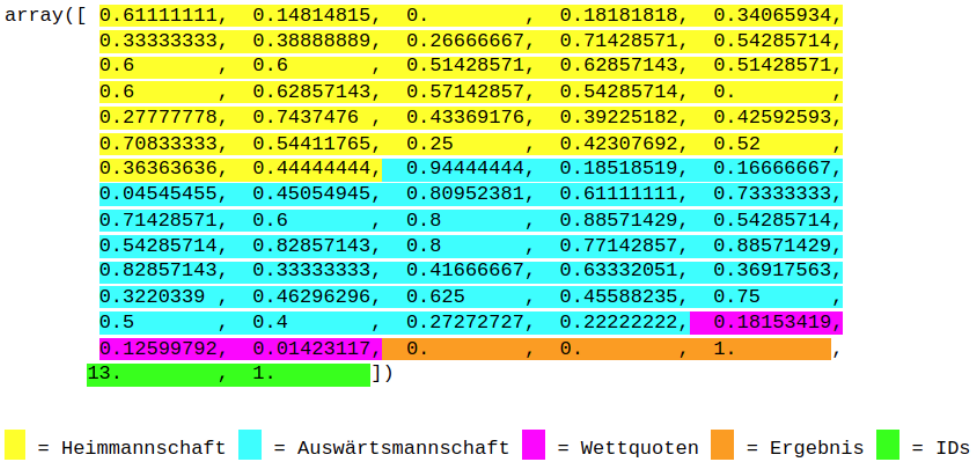


Abbildung 6: Vektor des Spiels 1. FSV Mainz 05 vs Borussia Dortmund aus der Saison 2015/2016

Normalisierung der Daten

Die Normalisierung wird verwendet, um die Werte der einzelnen *Features* auf eine gemeinsame Skala zu bringen, ohne die Unterschiede in den einzelnen Wertebereichen zu verzerren. Wenn die *Features* unterschiedliche Wertebereiche haben, wird eine Normalisierung benutzt. In diesem Datensatz haben wir beispielsweise Tabellen- und Spielerwerte, die einen anderen Wertebereich aufweisen. Die Tabellenwerte haben einen Wertebereich von 1 bis 18 und die Spielerwerte einen Wertebereich von 57 bis 90 (in der Saison 2019/2020). Da die Spielerwerte eine höhere Zahl haben, würden diese das Ergebnis stärker beeinflussen als die Tabellenwerte, obwohl nicht bekannt ist, ob die Spielerwerte relevanter für die Vorhersage sind oder nicht. Also werden alle Daten normalisiert, um alle *Features* auf den gleichen Wertebereich zu bringen.
Die Werte der einzelnen *Features* werden jeweils in einer Liste hinzugefügt und normalisiert. Die Spielerwerte aller Mannschaften werden in einer Liste zusammengefügt, da dies der gleiche Bereich ist. Analog wird das auch mit den Wettquoten gemacht. Die Normalisierung wird mit dem MinMaxScaler durchgeführt. Dies ist eine Methode, die die Werte auf einen Bereich zwischen Null und Eins setzt. Folgende Formel realisiert dies:

$$x_{std} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{19}$$
$$x_{scaled} = x_{std} * (max - min) + min$$

Hierbei sind *min* und *max*, das Minimum bzw. Maximum des jeweiligen *Feature*. Der

MinMaxScaler wird auf 23 unterschiedlichen *Features* angewendet. Der Spielstand und die Mannschafts-IDs werden nicht normalisiert, da dies keine *Features* sind.
Diese normalisierte Matrix wird wieder mit Hilfe von pickle gespeichert. Insgesamt ist die Matrix mit 44.064 Werten gefüllt, die jeweils eine Saison repräsentieren. In dieser Arbeit werden insgesamt fünf Saisons betrachtet.

5 Long Short-Term Memory

In diesem Kapitel wird die Architektur des LSTMs beschrieben und wie die Daten eingelesen werden.

5.1 Architektur

Das Modell besteht aus fünf Schichten, die die Zeitreihen und Zusatzinformationen zusammenführen und die Wahrscheinlichkeiten für einen Heimsieg, ein Unentschieden und einen Auswärtssieg ausgeben. Dies wird in Abbildung 7 dargestellt.

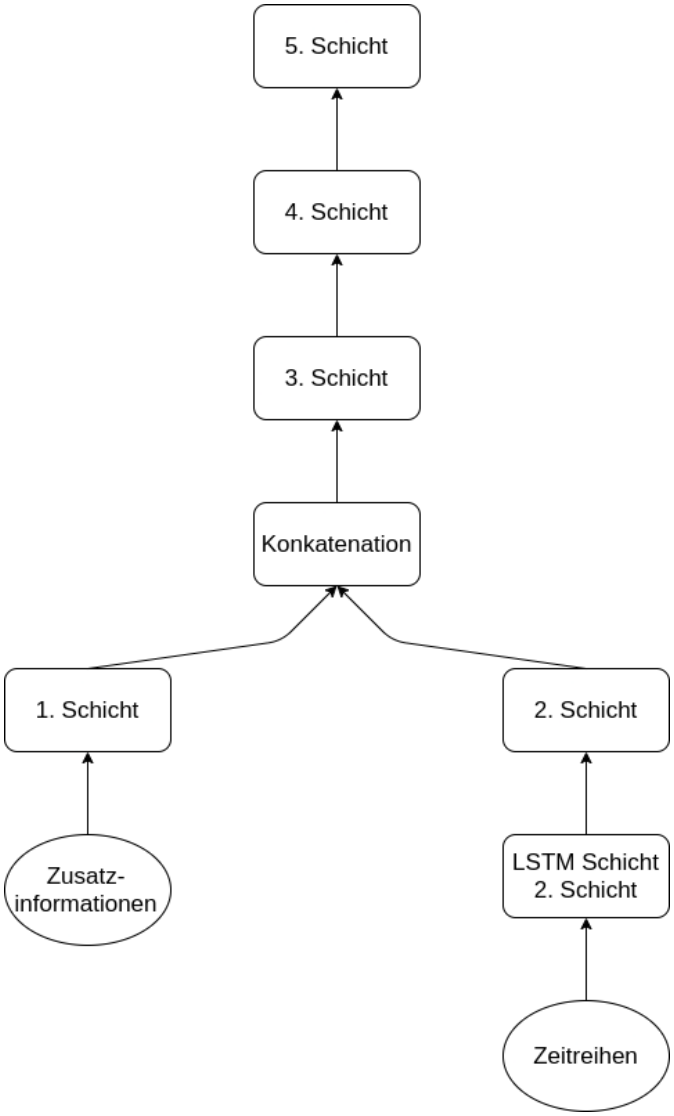


Abbildung 7: Architektur des LSTMs

Die erste Schicht liest die Zusatzinformationen ein. Um diese in eine Schicht einfügen zu können, werden sie in einen Tensor umgewandelt.
Bei der zweiten Schicht handelt es sich um eine LSTM Schicht. Diese erhält als Informationen die Zeitreihen, die ebenfalls in einen Tensor umgewandelt werden und arbeitet wie die LSTM Einheit. Die dritte Schicht dient für die Konkatenation der beiden Eingaben. Zunächst wird noch ein Dropout auf die LSTM Werte angewendet. Danach werden die Werte in die dritte Schicht eingefügt.
Daraufhin werden die erste und dritte Schicht konkateniert und in die vierte Schicht eingesetzt. Alle vier Schichten haben die gleiche Anzahl an *Units*, Einheiten, die aus einer positiven Zahl bestehen. Das ist die Dimensionalität des Ausgaberaums. Als Aktivierungsfunktion wird für die erste, dritte und vierte Schicht ReLu verwendet. Die LSTM Schicht besitzt eine tanh Aktivierung.
Die fünfte und letzte Schicht ist die Ausgabeschicht, bestehend aus drei Einheiten und der Aktivierungsfunktion Softmax. Die drei Einheiten bilden die Wahrscheinlichkeiten für den Heimsieg, das Unentschieden und den Auswärtssieg.

Um das Modell trainieren zu können, wird eine Verlustfunktion und ein Optimierer benötigt. Die Verlustfunktion wird verwendet, um den Fehler zu berechnen, den das Modell während des Trainings zu minimieren versucht. Sie ist in diesem LSTM die Sparse Categorical Crossentropy und vergleicht das vorhergesagte *Label* mit der wahren Ausgabe. Der Verlust wird mit folgender Formel berechnet und gibt an, ob sich die Parameteraktualisierung in die richtige oder falsche Richtung bewegt:

$$L(\theta) = -\sum_{i=1}^{k} y_i * log(\hat{y}_i) \tag{20}$$

Der Optimierer verbessert und verändert während des Trainings die Parameter, um die Verlustfunktion zu minimieren und somit die Vorhersage zu verbessern. Dabei werden die Modellparameter mit der Verlustfunktion verbunden, indem auf das *Label* reagiert und die Verlustfunktion aktualisiert wird. Dies wird in diesem Modell mit dem Adam Optimierer realisiert. Der Adam Optimierer ist ein stochastisches Gradientenabstiegsverfahren, das auf der adaptiven Schätzung von Momenten erster und zweiter Ordnung basiert. Die Methode ist rechnerisch effizient und hat wenig Speicherbedarf. Für den Optimierer wird der Hyperparameter *Learning Rate* (Kingma und Ba, 2014) benötigt. Dieser Parameter kontrolliert, wie schnell sich das Modell an das Problem anpasst. Wenn die *Learning Rate* zu niedrig ausgewählt wird, nimmt die Optimierung viel Zeit in Anspruch. Wenn die *Learning Rate* zu hoch ist, kann es passieren, dass das Training nicht konvergiert oder divergiert. Dann ist es möglich, dass sich der Verlust verschlimmert und der Optimierer das Minimum unterschreitet. (Smith, 2017)
Somit kann das Problem konfiguriert werden, das Modell wird trainiert und Vorhersagen werden durchgeführt. Dafür wird ein Trainings- und Testsatz benötigt. Diese zwei Datensätze werden gebildet, indem die Saison 2019/2020 als Testsatz und die vorherigen Saisons als Trainingssatz genommen werden. Damit können die Eingaben für das Modell erstellt werden, die aus Zeitreihen und Zusatzinformationen bestehen.

5.2 Zeitreihen

Die Zeitreihen bestehen aus Daten in einer zeitlichen Reihenfolge. Die Daten sind die Statistikwerte der jeweiligen Mannschaft und dessen Gegnern. Ein einzelner Zeitpunkt besteht aus Statistiken einer Anzahl gespielter Spiele. Die Anzahl der Zeitschritte ist abhängig von der Größe des Trainingssatz, also der Spieltage, die betrachtet werden können. Im besten Fall werden fünf vorherige Spiele für einen Zeitschritt berücksichtigt, dies wurde in der Projektarbeit ermittelt. Es werden fünf Spiele ausgewählt, wenn die Differenz der Spieltagsnummer x minus sechs größer als fünf ist. Dies ist der Fall, wenn es der zwölfte Spieltag ist. Wenn der Spieltag kleiner als zwölf ist, wird die Anzahl und Größe der Zeitreihen verkleinert, sodass genügend Zeitschritte betrachtet und eine vermutlich bessere Vorhersage getroffen werden kann.
Wenn beispielsweise der Spieltag 15 für Borussia Dortmund vorhergesagt werden soll, werden die Zeitreihen auf die Spiele 1 bis 14 einbezogen. Der Spieltag 14 hat von Dortmund und dessen Gegner von diesem Spieltag eine Zeitreihe der Spiele [13, 12, 11, 10, 9]. Der 13. Spieltag besitzt eine Zeitreihe der Spiele [12, 11, 10, 9, 8], der Spieltag 12 eine Zeitreihe der Spiele [11, 10, 9, 8, 7], usw., bis zum Spieltag sechs mit den Zeitreihen [5, 4, 3, 2, 1].

5.3 Zusatzinformationen

Die Zusatzinformationen werden genutzt, um die Zeitreihen besser einordnen zu können und so bessere Ergebnisse zu erhalten. Die Zusatzinformationen bestehen aus den Tabellenwerten der jeweiligen Mannschaft und deren Gegner. Dabei werden für jeden Zeitschritt die Tabellenwerte und Wettquoten genutzt. Für das vorherige Beispiel, den 15. Spieltag, werden die Tabellenwerte für die Spieltage 14 bis 6 für Borussia Dortmund und dessen Gegner, am jeweiligen Spieltag, genommen.
Durch die neue Einordnung können die Zeitreihen erneut interpretiert werden. Ähnliche Zeitreihen können für den ersten und letzten Platz andere Auswirkungen haben. Dies wird mit den Zusatzinformationen beachtet. (Hoang et al., 2016)

5.4 Daten einlesen

Nun besteht ein Datensatz, der in einen Trainings- und Testsatz unterteilt werden kann. Diese Sätze bestehen jeweils aus den Zeitreihen und den Zusatzinformationen. Der Trainings- und Testsatz haben eine ganze Liste von Zeitreihen und Zusatzinformationen, wobei die Liste beim Trainingssatz drei bis viermal größer ist. Die Daten entstehen, indem der Spieltag und die Mannschaft angegeben werden, für die eine Vorhersage ermittelt werden soll.
Um das Modell trainieren zu können, werden die Ergebnisse der Spiele benötigt, die im Trainingssatz betrachtet werden sollen. Diese Ergebnisse werden gefiltert, wenn die Zusatzinformationen für den Trainingssatz gesammelt werden. Somit stimmt die Anzahl der Tabellenwerte und die Anzahl der Ergebnisse überein.
Nun kann das Modell trainiert werden. Dafür wird das beschriebene Modell aufgebaut, indem die zwei Eingaben und die Lernrate als Parameter hinzugefügt werden, um die

richtigen Dimensionen für die Tensoren zu bilden. Um mit dem Modell trainieren zu können, werden die Zeitreihen und Zusatzinformationen als Eingabe, die Ergebnisse als Ausgabe und die Anzahl der Epochen benötigt. Eine Epoche besteht aus einem vollen Zyklus durch die Trainingsdaten. Dabei wird der Verlust und die Genauigkeit des Modells berechnet. Die Anzahl der Epochen wirkt sich auf das Ergebnis des Trainingsschrittes aus. Generell gilt, dass sich mit mehreren Trainingsepochen das Modell zu einem gewissen Punkt verbessert, da sich die Konvergenz erhöht und so eine höhere Genauigkeit erreicht wird. Allerdings kann mit einer zu hohen Epochenanzahl das Modell auch überangepasst werden.
Nach dem Training kann ein Spiel vorhergesagt werden. Hierfür wird das Modell zum Vorhersagen mit den Eingabedaten des Testsatzes aufgerufen. Das *Label* ist ein Array mit drei Werten, die die Wahrscheinlichkeiten für einen Heimsieg, für ein Unentschieden und für einen Gastsieg ausgibt. Die Summe der Wahrscheinlichkeiten liegt durch die Softmax-Funktion bei eins.

5.5 Experimente

Im LSTM Modell gibt es drei Hyperparameter, die manuell eingegeben werden. Es handelt sich um die *Units*, *Learning rate* und die Anzahl der Epochen.
Die *Units* (Keras, 2021) beschreiben die Dimensionalität des Ausgaberaums einer Schicht. Beim Modifizieren dieser Parameter verändern sich der Fehler, *Loss*, und die Genauigkeit, *Accuracy*, des Modells beim Training. Die *Units* werden mit Hilfe von Kerastuner (Keras Team, 2021) ermittelt. Hierbei wird ein Minium, ein Maximum und eine Schrittgröße als Parameter verwendet. Anschließend wird der *Loss* und die *Accuracy* verglichen und eine geeignete Anzahl gewählt. Es werden die *Units* zwischen 32 und 160, mit einer Schrittgröße von 32, gesetzt.
Die *Learning rate* kann im Allgemeinen nicht a priori berechnet werden (Reed und MarksII, 1999). Es wird ein Wert zwischen 1 und 10^{-6} empfohlen (Bengio, 2012). In diesem Bereich wird die *Learning rate* ausprobiert und die erfolgreichste ausgewählt. Dabei kommt es ebenfalls auf den *Loss* und die *Accuracy* an. Zudem gibt es einen Optimierer, der die *Learning Rate* während des Trainings anpasst.
Für die Anzahl der Epochen wird auch auf den *Loss* und die *Accuracy* geachtet.
Beim Auswerten dieser Experimente kann die Schlussfolgerung aufgestellt werden, dass die Anzahl der *Units* 32 ist, die *Learning rate* bei 5e-4 und die Anzahl der Epochen bei 25 liegt.

6 Transformer

Das zweite Modell, dass zur Vorhersage von Fußballspielen eingesetzt wird, ist der Transformer. Dieses Modell verarbeitet die Daten nicht wie das LSTM, sondern mit *Attention Mechanismen*. Aus diesem Grund können andere, wenn nicht sogar bessere, Ergebnisse erzielt werden. Bei *automatic speech recognition* (Zeyer et al., 2019) und *BERT* (Devlin et al., 2018) wurden bessere Ergebnisse mit einem Transformer erzielt als mit einem LSTM.
Der Transformer bekommt wie zuvor zwei unterschiedliche Eingabedaten und einen Satz Ausgabedaten. Dabei handelt es sich ebenfalls um die Zeitreihen, Zusatzinformationen und Ergebnisse der Spiele. In diesem Fall werden die Zeitreihen anderes verwaltet und das Modell hat nicht die typische Architektur eines Transformers, denn es wird lediglich der Encoder benutzt, da kein Decoder benötigt wird. Der Decoder würde eine Sequenz aus dem Encoder Vektor wieder erstellen, was in diesem Fall nicht benötigt wird.

6.1 Architektur

Die Architektur hat durch das Entfallen des Decoders eine ähnliche Struktur wie bei BERT (Devlin et al., 2018). Das Modell besteht aus mehreren Schichten und wird von unten nach oben, also von der Eingabe bis zur Ausgabe, beschrieben. Die Zeitreihen werden als Eingabedaten für den Transformer Encoder benutzt. Allerdings werden die Zeitreihen zusätzlich mit einem Time2Vec Vektor gebildet.

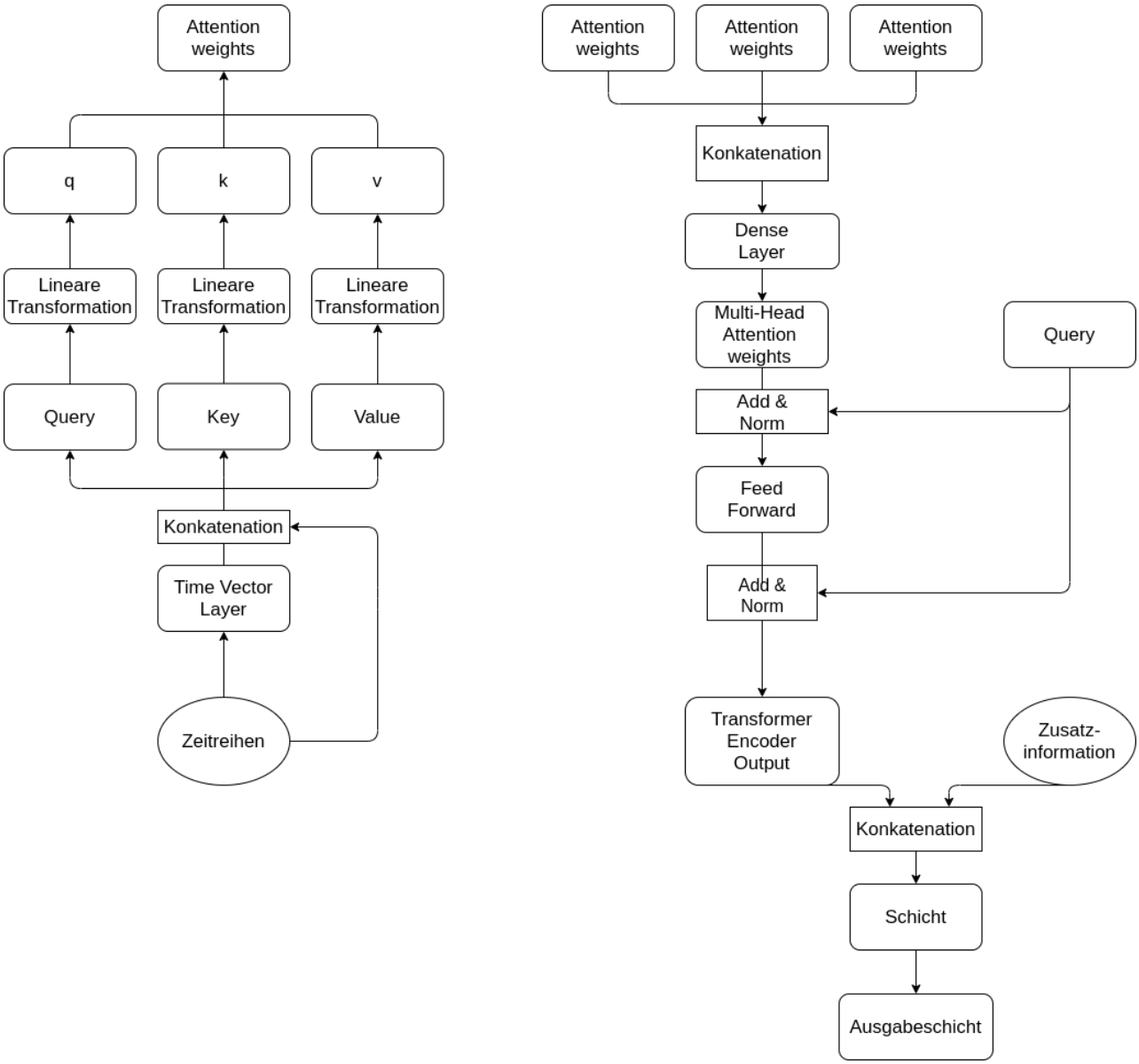


Abbildung 8: Architektur des Transformers

Time2Vec Vektor
Der Time2Vec Vektor (Kazemi et al., 2019) wird in periodischen und nicht-periodischen Mustern dargestellt. Ein periodisches Muster wäre beispielsweise das Wetter, das sich in den verschiedenen Jahreszeiten verändert. Ein nicht-periodisches Muster wäre eine Krankheit, die mit hoher Wahrscheinlichkeit auftritt, je älter die Person ist. Zudem soll die Präsentation der Zeit eine Invarianz gegenüber der Zeitskalierung vorweisen. Das bedeutet, dass die Präsentation der Zeit nicht durch unterschiedliche Zeitinkremente und lange Zeithorizonte beeinträchtigt wird. Diese beiden Punkte können in folgender Formel dargestellt werden:

$$t2v(\tau)[i] = \begin{cases} \omega_i\tau + \phi_i, & wenn\ i = 0. \\ F(\omega_i\tau + \phi_i), & wenn\ 1 <= i <= k. \end{cases} \tag{21}$$

Wenn i = 0 ist, wird das nicht-periodische Feature präsentiert. ω_i ist eine Matrix, die die Steigung der Zeitreihen definiert und ϕ_i eine Matrix, die angibt, an welchem Punkt sich die Zeitreihen mit der y-Achse schneiden. Es handelt sich bei dieser Formel um eine lineare Funktion.
Die zweite Formel zeigt das periodische Feature des Vektors. Die lineare Funktion wird in eine Sinusfunktion gesetzt und erzielt die besten und stabilsten Ergebnisse. ω_i beschreibt die Wellenlänge der Sinusfunktion, und ϕ_i verschiebt die Sinusfunktion entlang der x-Achse. Der Time2Vec Vektor ersetzt das *Postional Encoding* und bildet zusammen mit den Zeitreihen die Matrizen *Query*, *Key* und *Value*. Diese Matrizen erhalten jeweils eine lineare Transformation, indem sie einzelne Schichten durchlaufen. Danach werden die *Attention Scores* berechnet, die mit dem *Single-Head Attention* Mechanismus realisiert werden.
Diese beschreiben den Grad des Fokus, der auf einzelne Zeitschritte und deren Werte angewendet werden soll. Dies wird mit dem *Scaled Dot-Product* berechnet. Die *Attention*-gewichte werden im *Multi-Head Attention* verkettet, in einer Schicht zusammengefasst und bilden die *Multi-Head Attention* Gewichte. Diese ermöglichen die Kodierung mehrerer unabhängiger Transformationen der *Single-Head* Schichten im Modell.
Nun kann der Encoder aufgebaut werden. Dieser besitzt die *Attention* Schicht und die *Feedforward* Schicht und kann gestapelt, also mehrmals ausgeführt werden. Ein Decoder wird in diesem Transformer nicht verwendet, da dieser erneut eine Sequenz aus den Encoder *Labels* bilden würde. Dies wird jedoch nicht benötigt, da keine Sequenz erstellt werden soll, sondern eine Klasse aus Sieg, Niederlage und Unentschieden.
Nach der Berechnung des Transformer Encoder *Outputs* werden die Zusatzinformation hinzugefügt, mit dem Encoder *Label* konkateniert und in eine Schicht eingefügt. Die letzte Schicht ist die Ausgabeschicht, die identisch mit der Ausgabeschicht des LSTMs ist und aus drei Einheiten und der Softmax-Aktivierung besteht. Als Verlustfunktion und Optimierer wird die Sparse Categorical Crossentropy und der Adam Optimizer verwendet.

6.2 Zeitreihen

Die Zeitreihen werden ähnlich wie beim LSTM Modell gebildet. Es werden die Statistikwerte der eingegebenen Mannschaft und deren Gegner pro Spiel genutzt und dabei die Größe einer Zeitreihe und die Anzahl der Zeitreihen berechnet und auf die Mannschaften angewendet. Dies wird so lange wiederholt, bis die Anzahl der Zeitreihen erreicht ist.

Mit dieser Liste von Zeitreihen wird nun der Time2Vec Vektor erstellt. Dafür werden für die periodischen und nicht-periodischen *Features* jeweils zwei Matrizen erstellt, die für ω und ϕ stehen. Danach werden sie mit der obigen Formel berechnet.

6.3 Zusatzinformationen

Die Zusatzinformationen werden wie beim LSTM gebildet. Es werden pro Spiel die Tabellen-, Mannschafts- und Spielerwerte der beiden spielenden Mannschaften und der dazugehörigen Wettquoten gesammelt.

6.4 Daten einlesen

Der Datensatz wird nun in einen Trainings- und Testsatz geteilt. Dabei handelt es sich beim Trainingssatz um mehrere Saisons und beim Testsatz um eine Saison. Mit diesem Trainingssatz kann das Modell nun konfiguriert und trainiert werden. Für das Konfigurieren werden sechs Hyperparameter und die zwei Eingaben, die Zeitreihen und die Zusatzinformationen, benötigt. Die Hyperparameter bestehen aus den Dimensionen des *Keys*, des *Values* und der *Feedforward* Schicht und der Anzahl der *Attention Heads*. Diese müssen experimentell ausgesucht werden. Das Trainieren des Modells wird mittels der zwei Eingaben und der *Labels*, der Ergebnisse, sowie der Anzahl der Epochen durchgeführt. Die Epochen müssen ebenfalls experimentell ermittelt werden. Nach dem Trainieren kann das Spiel vorhergesagt werden, indem die Daten aus dem Testsatz angewendet werden. Das *Label* ist wieder ein Array aus drei Werten, die für die Wahrscheinlichkeit eines Heimsieges, eines Unentschiedens und eines Gastsieges stehen und mit einer Softmax-Funktion realisiert wird.

6.5 Experimente

Der Transformer besitzt, wie das LSTM, Hyperparameter. Allerdings besitzt dieses Modell sieben: Die Dimension für den *Key* und den *Value*, die Anzahl der *Attention Heads*, die Größe der *Hidden Layers* im *Feedforward* Netzwerk, die *Learning rate*, die *Units* und die Anzahl der Epochen.
Die *Learning rate*, die *Units* und die Anzahl der Epochen werden wie beim LSTM ermittelt.
Die anderen Parameter werden durch Ausprobieren in Anlehnung des *Papers* (Vaswani et al., 2017) ermittelt. Die Dimensionen des *Keys* und *Values* sind gleich groß. Die Schrittgröße ist bei den neuen Hyperparametern 128, außer bei der Anzahl der *Attention Heads*, hier liegt diese bei zwölf.
Abschließend wurden folgende Parametergrößen festgelegt:

Nummer	Features
Dimension Key	128
Dimension Value	128
Dimension Feedforward	128
Attention Heads	12
Units	128
Learning rate	5e-6
Epochen	25

Tabelle 3: Daten eines Spiel für beide Mannschaften

7 Evaluation

In diesem Kapitel werden der Transformer und das LSTM getestet und die Ergebnisse der Vorhersagen ausgewertet. Dazu werden die Testdaten vorher beschrieben und die Evaluationsmaße eingeführt.

7.1 Erhebung des Datensatzes

Der Datensatz besteht aus fünf Fußballsaisons der 1. Bundesliga. Dabei handelt es sich um die Saisons 2015/2016, 2016/2017, 2017/2018, 2018/2019 und 2019/2020. Jede Saison besteht aus 18 Mannschaften und 34 Spieltagen mit jeweils neun Spielen. Nach jeder Saison steigen mindestens zwei oder maximal drei Mannschaften in die 2. Bundesliga ab und mindestens zwei oder maximal drei Mannschaften von der 2. Bundesliga auf. Dies wird durch die Relegationsspiele entschieden. Dazu spielen der 16. Platz der 1. Bundesliga gegen den dritten Platz der 2. Bundesliga. Der Gewinner spielt in der kommenden Saison in der 1. Bundesliga. Die ersten beiden Mannschaften der zweiten Bundesliga steigen automatisch auf.
Der Datensatz wird nun in Trainings-, Validierungs- und Testsatz (1) oder in Trainings- und Testsatz (2) aufgeteilt. Somit liegen zwei Aufteilungen des Datensatzes vor, um zu untersuchen, ob sich ein zusätzlicher Validierungs- oder ein größerer Trainingssatz positiv auf den Test auswirkt. Die Aufteilung des Datensatzes beruht zu 60 Prozent auf dem Training, zu 20 Prozent auf der Validierung und zu 20 Prozent (1) auf dem Test, bzw. zu 80 Prozent auf dem Training und zu 20 Prozent (2) auf dem Test.

7.1.1 Trainingssatz

Der Trainingssatz ist der Teil der Daten mit dem das Modell trainiert. Hierbei wird das Modell durch die Trainingsmenge aktualisiert (Ripley, 1996). Der Trainingssatz besteht aus drei bzw. vier Saisons der Jahre 2015/2016, 2016/2017, 2017/2018 bzw. einschließlich 2018/2019. Die ersten drei Saisons werden konkateniert. Dadurch werden größere Zeitreihen gebildet und die Modelle können intensiver trainieren. Bei der Konkatenation gibt es diverse Probleme, die gelöst werden müssen:
Nach einer Saison steigen zwei bis drei Mannschaften in die 2. Bundesliga ab und zwei bis drei Mannschaften in die 1. Bundesliga auf. Somit ändern sich die Mannschaften in den drei Saisons. Dadurch vergrößert sich die Matrix, die alle *Features* der Mannschaften sammelt. Die erste und die zweite Dimension, also die Anzahl der Mannschaften und die Anzahl der Spiele vergrößern sich. Die Saisons werden nacheinander zusammengefügt und so vergrößert sie die Matrix in den Dimensionen von [18, 34, 72] auf [20, 68, 72].
Um dies zu realisieren, müssen sich die IDs der Mannschaften aus der zweiten Saison an die erste Saison anpassen, damit die richtigen Spiele beim Erstellen der Zeitreihen gesammelt werden. Außerdem werden neue IDs für die Aufsteiger hinzugefügt.
Beim Zusammenfügen der Daten bestehen für die Auf- bzw. Absteiger nur Daten für 34 Spieltage. Die restlichen Daten der Spiele, die diese Mannschaften nicht besitzen, werden mit -1 gefüllt.
Die Matrix für zwei Saisons ist nun gebildet und wird analog für die Zusammenführung

der dritten Saison, sowie der ersten beiden Saison durchgeführt. Insgesamt liegt eine Matrix der Größe [20, 102, 72] vor. Die drei Saisons haben insgesamt 20 Mannschaften, da sich die Aufsteiger aus der Saison 2017/2018 bereits in der Mannschaftsliste befinden, dazu 102 Spieltage und 72 *Features* inklusive deren IDs.
Beim Erstellen des Trainingssatzes muss beim Bilden der Zeitreihen und Zusatzinformationen auf die Auf- und Absteiger geachtet werden. Wenn es sich bei einer der zwei Mannschaften um einen Auf- bzw. Absteiger handelt, wird der Spieltag gegebenenfalls reduziert.
Soll beispielsweise der *Input* für den 50. Spieltag erstellt werden und es handelt sich um einen Aufsteiger, liegen für diese Mannschaft keine Daten der vorherige Saisons vor. Somit können anstatt 50 lediglich 16 Spieltage betrachtet werden und die Größe der Zeitreihen und Zusatzinformationen verringert sich. Bei einem Absteiger darf der Spieltag nicht berücksichtigt werden, da sich dieser nicht in der aktuellen Saison befindet.
Die vierte Saison, 2018/2019, wird einfach zu der Konkatenation hinzugefügt, sodass Zeitreihen und Zusatzinformationen aus der Konkatenation bzw. Zeitreihen und Zusatzinformationen aus einer Saison bestehen.

7.1.2 Validierungssatz

Der Validierungssatz ist der Teil des Datensatzes, mit dem die Modellanpassung des Trainingsdatensatzes bewertet werden kann. Dieser wird für die Feinabstimmung der Hyperparameter verwendet (Ripley, 1996).
Der Validierungssatz besteht aus der Saison 2018/2019. In diesem werden 34 Spieltage von 18 Mannschaften betrachtet. Für jedes Spiel wird die größtmögliche Zeitreihe und die dazugehörigen Zusatzinformationen gesammelt.

7.1.3 Testsatz

Der Testsatz ist der Teil des Datensatzes, der lediglich zur Beurteilung der Modelle genutzt wird (Ripley, 1996). Dieser wird erst verwendet, wenn das Modell fertig trainiert hat und keine Anpassung an den Hyperparametern vorgenommen werden muss.
Der Testsatz besteht aus der Saison 2019/2020. Dieser hat wie der Validierungssatz ebenfalls 34 Spieltage von 18 Mannschaften.

In der folgenden Abbildung 9 ist die Aufteilung erneut dargestellt. Es werden insgesamt zwei Modelle für das LSTM und zwei Modelle für den Transformer erstellt. Sie unterscheiden sich in der Aufteilung der Trainingsmengen. Jedes Modell hat einmal als Trainingsmenge drei Saisons und eine Saison als Validierung und das andere Mal vier Saisons als Trainingsmenge.

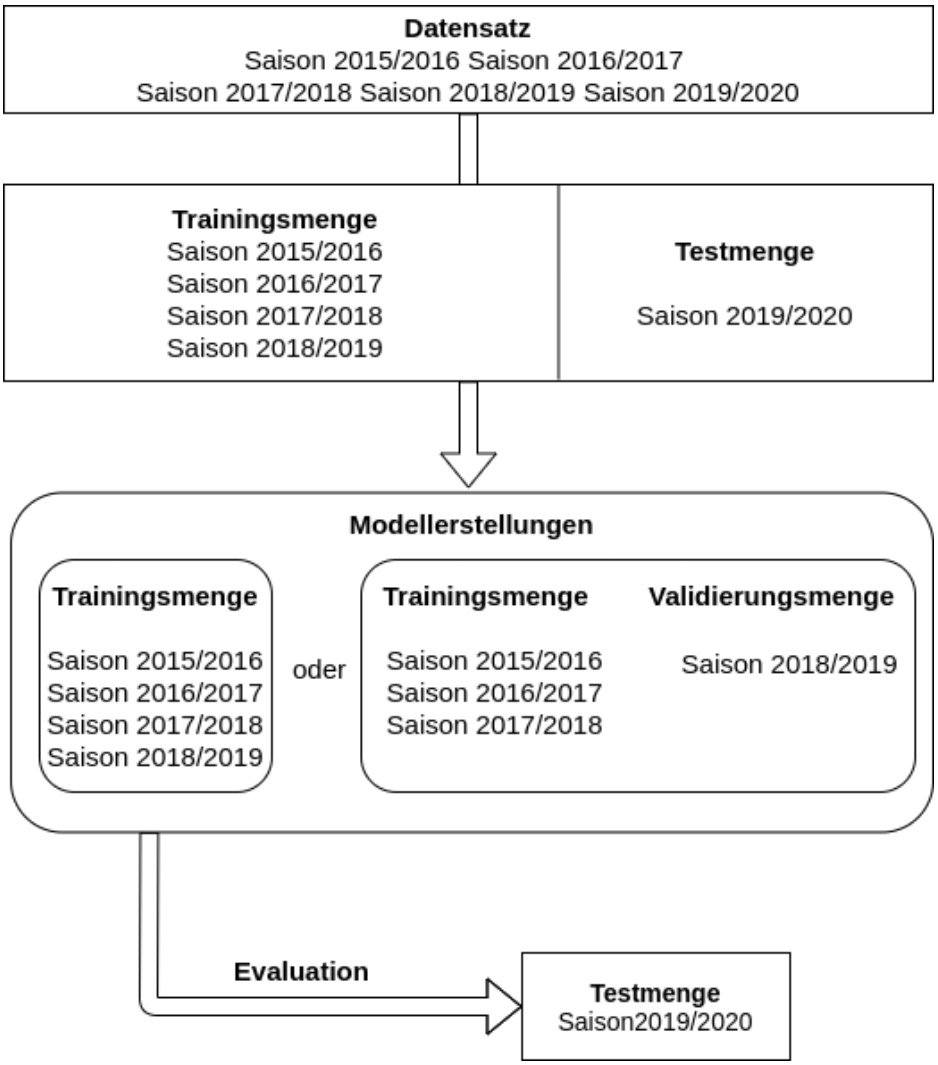


Abbildung 9: Ablauf der Modellerstellung

7.2 Evaluationsmaß

Die Beurteilung, wie exakt die Modelle die Ergebnisse im Testset vorhersagen, wird mit den Evaluationsmaßen *Accuracy*, *Precision* und *Recall* sowie *F-Measure* beschrieben. Diese werden im Folgenden eingeführt und in Bezug zu dieser Arbeit gebracht.
Hierfür müssen zunächst vier Werte eingeführt werden: *True Positives (TP)*, *True Negatives (TN)*, *False Positives (FP)* und *False Negatives (FN)*

True Positives
Die Werte *True Positives* sind die korrekt vorhergesagten positiven Werte, d.h. dass der Wert der tatsächlichen Klasse "Ja" und der Wert der vorhergesagten Klasse ebenfalls "Ja" ist. Die tatsächliche Klasse und die vorhergesagte Klasse gehen beide von einem Sieg aus.

True Negatives
Die Werte *True Negatives* sind die korrekt vorhergesagten negativen Werte, d.h. dass der Wert der tatsächlichen Klasse "Nein" und der Wert der vorhergesagten Klasse ebenfalls "Nein" ist. Die tatsächliche Klasse und die vorhergesagte Klasse gehen beide von keinem Sieg aus.

False Positives
Die Werte *False Positives* sind die falsch vorhergesagten positiven Werte, d.h. dass der Wert der tatsächlichen Klasse "Nein" und der Wert der vorhergesagten Klasse "Ja" ist. Die tatsächliche Klasse geht davon aus, dass die Mannschaft nicht gewinnt und die vorhergesagte Klasse, dass die Mannschaft gewinnt.

False Negatives
Die Werte *False Negatives* sind die falsch vorhergesagten negativen Werte, d.h. dass der Wert der tatsächlichen Klasse "Ja" und der Wert der vorhergesagten Klasse "Nein" ist. Die tatsächliche Klasse geht davon aus, dass die Mannschaft gewinnt und die vorhergesagte Klasse, dass die Mannschaft nicht gewinnt.

7.2.1 Accuracy

Die *Accuracy*, die Genauigkeit, ist der Anteil der richtigen Vorhersagen. Dies lässt sich mit der Formel

$$Accuracy = \frac{Korrekte\,Vorhersage}{Alle\,Vorhersagen} \tag{22}$$

oder mit

$$Accuracy = \frac{TP\ +\ TN}{TP\ +\ TN + FP\ +\ FN} \tag{23}$$

ausdrücken. Es ist die intuitivste Metrik, jedoch lediglich eine Aussage zur Gesamtbeobachtung.

7.2.2 Precision

Der *Precision* Wert beschreibt das Verhältnis von den korrekten positiven Beobachtungen zu den vorhergesagten positiven Beobachtungen. Als Beispiel können die Spiele dienen, die als Sieg vorhergesagt werden. Es werden alle richtig vorhergesagten Siege im Verhältnis zu allen Siegen, die das Modell als Sieg vorhergesagt hat, gesetzt.

$$Precision = \frac{TP}{TP\ +\ FP} \tag{24}$$

7.2.3 Recall

Der *Recall* Wert beschreibt das Verhältnis von den korrekt vorhergesagt positiven Beobachtungen zu den tatsächlich positiven Beobachtungen. Als Beispiel werden wieder die Siege betrachtet. Es werden alle richtig vorhergesagten Siege im Verhältnis zu allen tatsächlichen Siegen, gesetzt.

$$Recall = \frac{TP}{TP\ +\ FN} \tag{25}$$

7.2.4 F-Measure

Der *F-Measure* Wert beschreibt den gewichteten Durchschnitt von *Recall* und *Precision*, und somit wird auf die Bewertungen von *False Positive* als auch von *False Negative* geachtet.

$$F\text{-}Measure = \frac{2\ *\ (Precision\ *\ Recall)}{(Precision\ /\ Recall)} \tag{26}$$

(Powers, 2020)

Evaluation der Modelle
Die Evaluation der Modelle wird in Siege, Unentschieden und Niederlagen geteilt, um *Precision*, *Recall* und *F-Measure* anzuwenden. Außerdem wird untersucht, in welchem Abschnitt der Saison die meisten Spiele korrekt vorhergesagt wurden bzw. genau an welchem Spieltag.

7.3 Bewertung der Vorhersage

In dieser Arbeit werden neben der Analyse der Siege, Niederlagen und den Unentschieden noch weitere Punkte analysiert. Zudem wird untersucht, in welchen Phasen der Fußballsaison die meisten Spiele richtig vorhergesagt wurden, sowie die erfolgreichsten und am schlechtesten vorhergesagten Mannschaften. Außerdem wird das Modell zusätzlich durch *Accuracy* bewertet, an welchem Spieltag die meisten bzw. wenigsten Spiele richtig vorhergesagt wurden.
Die Bewertung der Vorhersagen wird in Prozente unterteilt. Ein Spiel einer Mannschaft wird als Niederlage bewertet, wenn sich die Prozente zwischen 0 und 40 befinden, als Unentschieden, wenn sich die Prozente zwischen 40 und 60 befinden und als Sieg, wenn sich die Prozente zwischen 60 und 100 befinden.
Insgesamt wurden 612 Spiele getestet. Hierbei sind 238 Siege, 238 Niederlagen und 136 Unentschieden zu verzeichnen. Das sind zu 39 Prozent Siege und Niederlagen und zu 22 Prozent Unentschieden.
Der Trainingssatz besteht einmal aus 1.836 Spielen, mit 690 Siegen, 690 Niederlagen und 456 Unentschieden, und zum anderen aus 2.448 Spielen, mit 923 Siegen, 923 Niederlagen und 602 Unentschieden. Das sind im kleineren Trainingssatz jeweils 38 Prozent Siege und Niederlagen und 24 Prozent Unentschieden und beim größeren Trainingssatz jeweils 37 Prozent Siege und Niederlagen und 26 Prozent Unentschieden.

7.3.1 LSTM

Es gibt zwei Modelle für das LSTM und diese unterscheiden sich in den Trainings- und Validierungssätzen. Das erste Modell besteht aus drei Fußballsaisons für den Trainingssatz und eine Saison für das Validierungssatz. Das zweite Modell besitzt keinen Validierungssatz, sondern hat vier Fußballsaisons für den Trainingssatz.

Allgemeine Bewertung
Das erste Modell hat eine *Accuracy* von 58,3 Prozent. Dieser Wert kommt zustande, indem alle Mannschaften und alle Spieltage betrachtet und deswegen alle Vorhersagen einbezogen wurden. Es wurden also 356,8 Spiele von 612 Spielen in der Saison richtig vorhergesagt, also 10,5 Spiele pro Spieltag.
Das zweite Modell hat eine *Accuracy* von 60,5 Prozent. Dies waren 370,2 Spiele von 612 Spielen, also 10,9 richtig vorausgesagte Spiele an einem Spieltag.
Bei der allgemeinen Bewertung ist zu sehen, dass das Modell ohne Validierungssatz und einem größeren Trainingssatz besser ist, als ein Modell mit Validierungssatz und einem kleinerem Trainingssatz.

Precision
Beim *Precision* Wert werden die richtigen Vorhersagen im Verhältnis zu allen vorausgesagten Siegen, Niederlagen bzw. Unentschieden dargestellt.

Modell	Sieg	Unentschieden	Niederlage
Erstes Modell	66,5%	40,6%	66,5%
Zweites Modell	67,4%	42%	67,4%

Tabelle 4: Precision Werte für die LSTM Modelle

In der Tabelle 4 wird deutlich, dass das zweite Modell etwas besser im Test abschneidet als das erste Modell. Die *Precision* Werte für das Unentschieden sind am niedrigsten. Dies bedeutet, dass das Modell öfter ein Unentschieden voraussagt, auch wenn das Spiel nicht mit einem Remis endete.

Recall
Der *Recall* Wert gibt das Verhältnis zwischen den richtigen Vorhersagen und den tatsächlichen Ergebnissen der Siege, Niederlagen und den Unentschieden an.

Modell	Sieg	Unentschieden	Niederlage
Erstes Modell	60%	54,4%	60%
Zweites Modell	62,7%	54,4%	62,7%

Tabelle 5: Recall Werte für die LSTM Modelle

Die Tabelle 5 zeigt, dass sich die Werte der Siege und Niederlagen im Gegensatz zu den *Precision* Werten verschlechtert und sich die Unentschieden-Werte gesteigert haben. Es wurden also weniger Spiele als tatsächliche Siege bzw. Niederlagen richtig vorhergesagt und dafür mehr Spiele als Unentschieden.

F-Measure
Der *F-Measure* Wert bildet sich aus den *Precision* und *Recall* Werten. Es ist der Mittelwert dieser beiden Metriken.

Modell	Sieg	Unentschieden	Niederlage
Erstes Modell	63,1%	46,5%	63,1%
Zweites Modell	65%	47,4%	65%

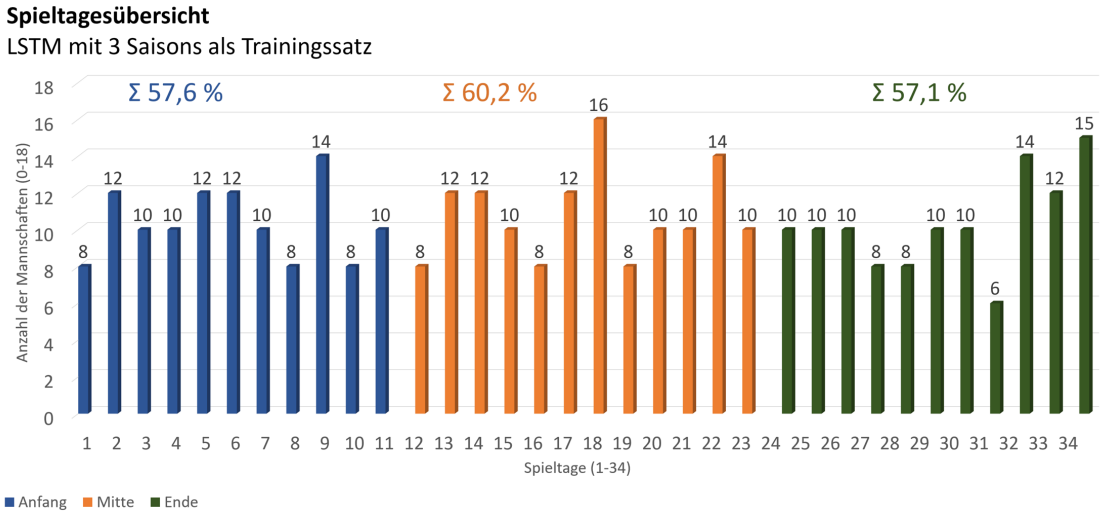
Tabelle 6: F-Measure Werte für die LSTM Modelle

Die Tabelle 6 zeigt, dass die Vorhersage der Siege und Niederlagen bei 63,1 Prozent im ersten und bei 65 Prozent im zweiten Modell lagen. Die Unentschieden-Werte befanden sich bei den Modellen bei 46,5 und 47,4 Prozent.

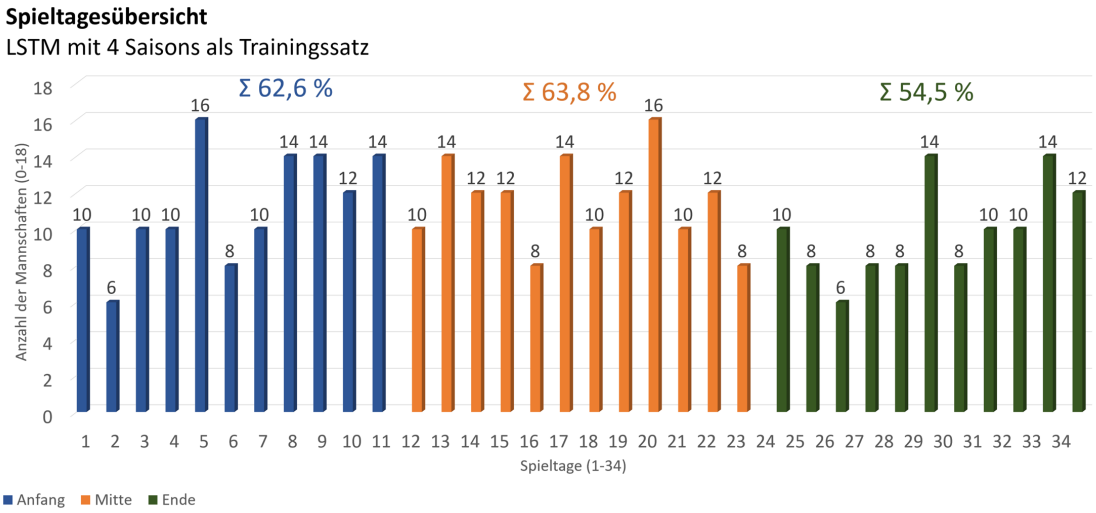
Abschließend lässt sich somit sagen, dass Siege und Niederlagen besser vorausgesagt wurden, als Unentschieden. Zudem lässt sich feststellen, dass ein Modell mit einem größeren Trainingssatz bessere Vorhersagen trifft, als ein Modell mit einem kleineren Trainings- und Validierungssatz.
Ein möglicher Grund dafür, dass das zweite Modell öfter Unentschieden voraussagte als das erste, ist, dass dieses im Trainingssatz einen etwas höheren Anteil an unentschieden Spielen besitzt.

Spieltagsanalyse

Dieser Abschnitt untersucht die Spieltage in der Saison, indem analysiert wird, welcher Bereich in der Saison am erfolgreichsten vorhergesagt wurde.



(a) Spieltagsübersicht für den Trainingssatz aus drei Saisons



(b) Spieltagsübersicht für den Trainingssatz aus vier Saisons

Abbildung 10: Spieltagsübersicht des LSTMs

Die Saison wird in drei Bereiche geteilt: den Anfang der Saison, also den 1. bis 11. Spieltag, die Mitte der Saison, also den 12. bis 23. Spieltag und das Ende der Saison, den 24. bis 34. Spieltag.
In der Abbildung 10 werden die Ergebnisse der Spieltage der beiden Modelle dargestellt. Es werden pro Spieltag die Anzahl der Mannschaften angezeigt, die korrekt vorausgesagt wurden. Dabei kann ein Spieltag einen Bereich von 0 bis 18 haben.
Im ersten Modell schneidet der mittlere Bereich mit 60,2 Prozent am erfolgreichsten ab. Das sind im Schnitt 5,4 von 9 Spielen pro Spieltag die korrekt vorausgesagt wurden. Der zweitbeste Bereich ist mit 57,6 Prozent der Anfang der Saison und der drittbeste Bereich mit 57,1 Prozent das Ende der Saison.
Der 18. Spieltag ist der mit den meisten richtig vorhergesagten Spielen, bei dem 16 Mannschaften also acht Spiele korrekt vorausgesagt wurden und der Spieltag mit den wenigsten richtig vorhergesagten Spielen ist der 31. Spieltag mit sechs Mannschaften, also drei Spielen.
Das zweite Modell hat mit 63,8 Prozent den besten Bereich ebenfalls in der Mitte der Saison. Das entspricht 5,7 Spielen pro Spieltag. Danach folgt erneut der Anfang der Saison mit diesmal 62,6 Prozent, was auch 5,6 Spielen pro Spieltag gleichkommt. Der niedrigste Bereich ist am Ende der Saison mit 54,5 Prozent zu finden.
Der erfolgreichste Spieltag ist der 5. und 20. Spieltag mit 16 Mannschaften also acht Spielen. Der schwächste Spieltag ist der zweite und 26. Spieltag mit sechs Mannschaften also drei Spielen.

Abschließend lässt sich sagen, dass die Vorhersagen am Ende der Saison weniger gut und in Mitte der Saison am besten sind. Dies kann daran liegen, dass sich alle Mannschaften in der Mitte der Saison am besten eingespielt haben und am konstantesten spielen. Dass das Ende der Saison der schwächste Bereich ist, könnte daran liegen, dass in diesem Abschnitt die letzten Punkte geholt werden können. Somit spielen die schlechteren Mannschaften dann anders als in den vorherigen Spielen, damit sie nicht absteigen. Außerdem könnten Mannschaften, die sich im Mittelfeld der Tabelle befinden und nicht mehr absteigen oder sich daher nicht mehr für die europäischen Plätze qualifizieren können, weniger ambitioniert spielen. Damit könnten sich die Ergebnisse in diesem Bereich verschlechtern.

Mannschaftsanalyse
In diesem Abschnitt wird analysiert, welche Mannschaften in der *Accuracy* am erfolgreichsten bzw. am schwächsten vorausgesagt wurden.
Im ersten Modell sind der FC Bayern München mit 76,5 Prozent und Fortuna Düsseldorf mit 67,6 Prozent am häufigsten richtig vorausgesagt worden. Diese Mannschaften befinden sich am Ende der Saison auf dem ersten Platz mit dem FC Bayern München und mit Fortuna Düsseldorf auf den 17. Platz. Die am seltensten richtig vorhergesagten Mannschaften sind der VFL Wolfsburg und der 1. FC Köln mit 38,2 Prozent bzw. 50 Prozent. Diese Mannschaften haben die Saison auf dem siebten bzw. 14. Platz beendet.
Im zweiten Modell schnitten der FC Bayern München mit 70,6 Prozent und Borussia Dortmund mit 67,6 Prozent am erfolgreichsten ab. Borussia Dortmund hat die Saison auf dem zweiten Platz beendet. Die Mannschaften mit der geringsten *Accuracy* sind der SC Freiburg mit 50 Prozent und Hertha BSC Berlin mit 52,9 Prozent. Diese Mannschaften schlossen die Saison auf dem achten bzw. zehnten Platz ab.

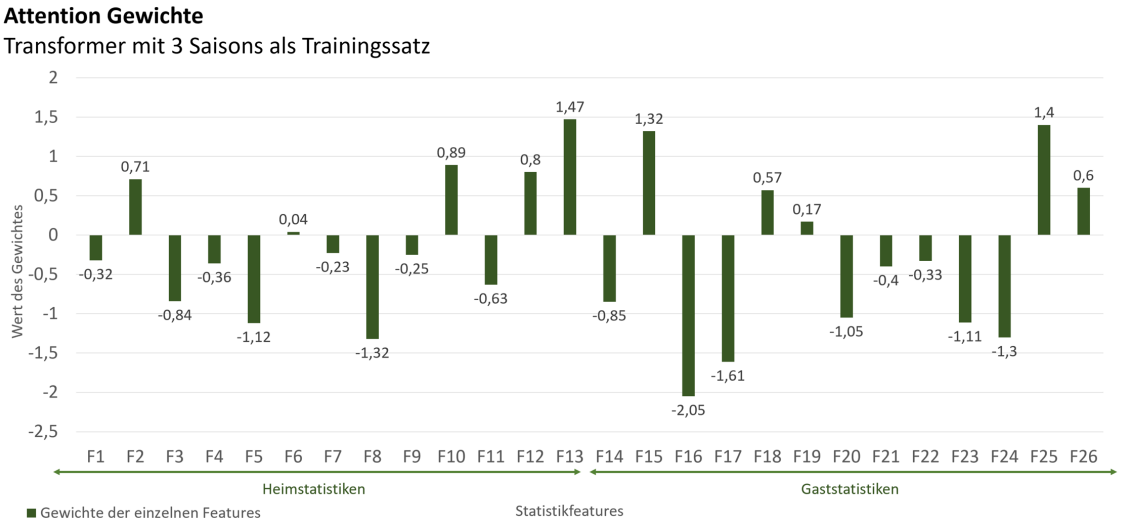
Abschließend lässt sich sagen, dass die Mannschaften, die ganz oben bzw. unten in der Tabelle stehen am erfolgreichsten vorhergesagt wurden. Die am schwächsten vorhergesagten Mannschaften befinden sich im Mittelfeld der Tabelle.

In der Analyse der beiden LSTM Modelle hat sich herausgestellt, dass das Modell mit einem größeren Trainingssatz besser darin ist Fußballspielergebnisse korrekt vorauszusagen. Außerdem ist der mittlere Bereich der Saison am erfolgreichsten richtig vorausgesagt worden, sowie die Mannschaften, die sich oben oder unten in der Tabelle befinden.

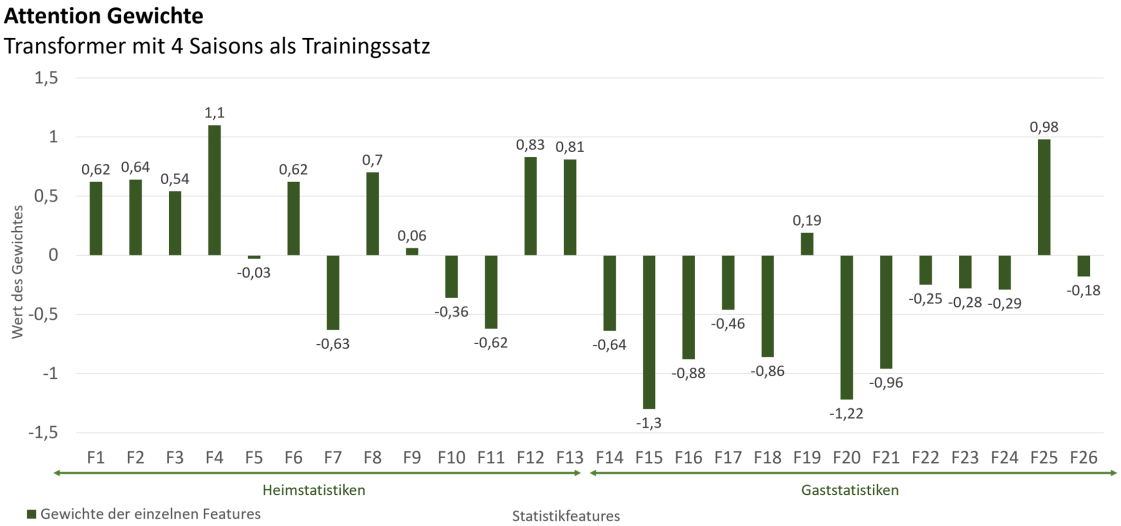
7.3.2 Transformer

Die zwei Modelle des Transformers haben die gleichen Trainingssätze und den gleichen Validierungssatz wie für das LSTM. Ein Modell besteht aus einer Saison für den Validierungssatz und drei Saisons für den Trainingssatz und das andere Modell aus vier Saisons für den Trainingssatz.

Erkenntnisse aus dem Training
Nach dem Training des LSTMs und des Transformers, wurden die Gewichtungen aus der *Attention* Schicht extrahiert.



(a) Spielstatistiken nach dem Training



(b) Spielstatistiken nach dem Training

Abbildung 11: Featurebewertungen der Spielstatistiken aus beiden Transformer Modelle

Dadurch wird deutlich, welche *Features* stärker bei der Vorhersage von Spielen gewichtet werden. In der Abbildung 11 ist dargestellt, dass die Spielstatistiken der Heimmannschaft stärker gewichtet werden, als die der Gastmannschaft. Die Heimstatistiken werden von F1 bis F13 dargestellt, und die restlichen Werte veranschaulichen die Gaststatistiken. Diese *Features* werden in der Abbildung 1 mit den Nummern 20 bis 32 beschrieben.
Den stärksten Einfluss aus der Heimmannschaft haben die Tore, Torschüsse und die gespielten Pässe und aus der Gastmannschaft die Fehlpässe und wie oft diese Mannschaft im Abseits stand.

Allgemeine Bewertung
Die allgemeine Bewertung wird mit der *Accuracy* dargestellt. Es wurden ebenfalls alle Mannschaften und alle Spiele betrachtet. Das erste Modell hat eine *Accuracy* von 65,3 Prozent, was 399,6 von 612 Spielen entspricht. Das zweite Modell hat eine *Accuracy* von 69 Prozent. Dies entspricht 422,3 von 612 Spielen.
Das zweite Modell mit dem größeren Trainingssatz ohne Validierungssatz schneidet im Test besser ab als das erste Modell.

Precision
Der *Precision* Wert besteht aus den richtig vorhergesagten Siegen, Niederlagen oder Unentschieden im Verhältnis zu allen vorhergesagten Siegen, Niederlagen oder Unentschieden.

Modell	Sieg	Unentschieden	Niederlage
Erstes Modell	69,1%	50,7%	69,1%
Zweites Modell	71,4%	53,8%	71,4%

Tabelle 7: Precision Werte für die Transformer Modelle

Das zweite Modell hat die erfolgreicheren Prognosen in allen drei Bereichen. Bei beiden Modellen wird deutlich, dass das Unentschieden am seltensten richtig vorhergesagt wurde. Das heißt, dass beide Modelle häufiger ein Unentschieden voraussagen, als es tatsächlich in der Saison auftritt. Ein möglicher Grund dafür kann sein, dass der Anteil der richtigen Prognosen von Siegen und Niederlagen in den Trainingssätzen mit elf bzw. 14 Prozent den Unentschieden gegenüber überwiegt. Somit werden die Spiele nicht klar als Siege, Niederlagen oder Unentschieden vorausgesagt, sodass sich die Spiele oft im Bereich zwischen 40 und 60 Prozent in der Vorhersage befinden.

Recall
Der *Recall* Wert besteht aus den richtig vorhergesagten Siegen, Niederlagen oder Unentschieden im Verhältnis zu allen tatsächlichen Siegen, Niederlagen oder Unentschieden.

Das erste Modell ist lediglich bei den unentschiedenen Vorhersagen besser. Bei den Siegen und Niederlagen ist das zweite Modell besser. Es werden in beiden Modellen tatsächlich weniger Spiele als Sieg bzw. Niederlage getippt und mehr Spiele als Unentschieden.

Modell	Sieg	Unentschieden	Unentschieden
Erstes Modell	63%	66,2%	63%
Zweites Modell	68,1%	62,5%	68,1%

Tabelle 8: Recall Werte für die Transformer Modelle

F-Measure
Beim *F-Measure* wird wieder der harmonische Mittelwert von *Precision* und *Recall* ermittelt.

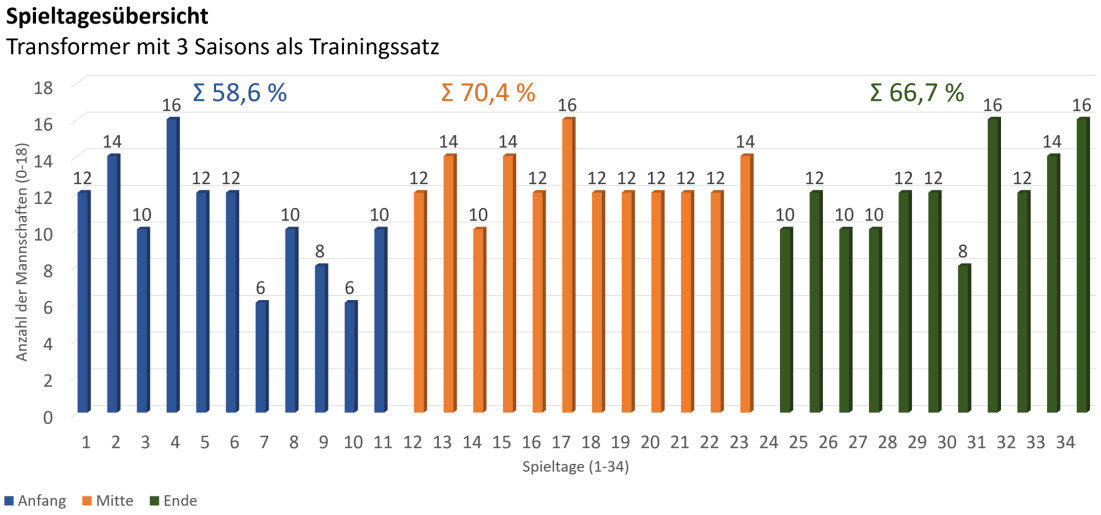
Modell	Sieg	Unentschieden	Niederlage
Erstes Modell	65,9%	57,5%	65,9%
Zweites Modell	69,9%	58,2%	69,9%

Tabelle 9: F-Measure Werte für die Transformer Modelle

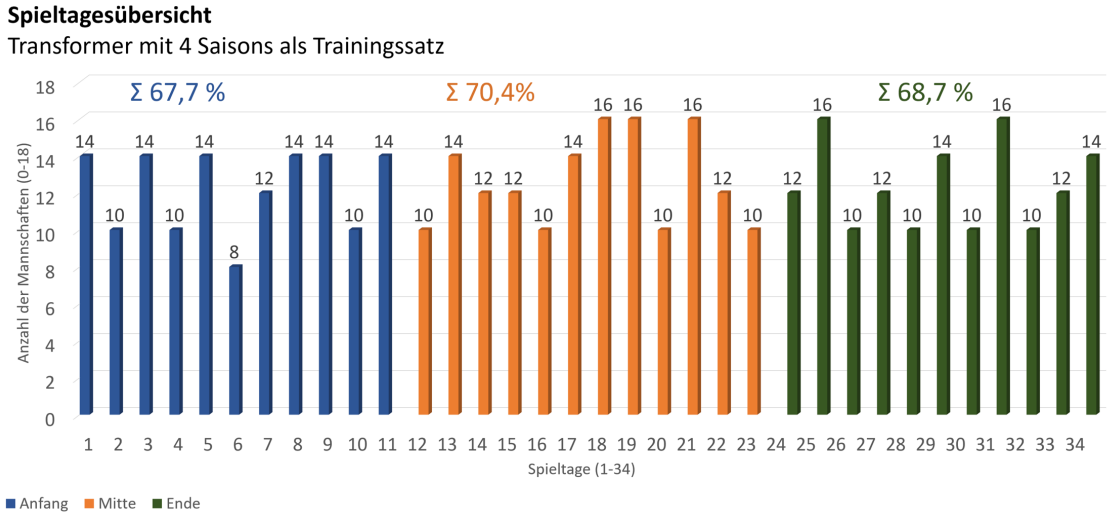
Die Tabelle 9 zeigt, dass alle Vorhersagen des zweiten Modells besser sind als beim ersten Modell. Dabei überwiegen die Siege und Niederlagen mit einem Abstand von vier Prozent. Beim Unentschieden hat man lediglich eine Differenz von 0,7 Prozent.

Abschließend lässt sich sagen, dass das zweite Modell erfolgreicher richtige Vorhersagen erstellt, als das erste Modell. Also ist auch hier ein größerer Trainingssatz besser als ein kleinerer mit zusätzlichem Validierungssatz. Siege, Niederlagen und Unentschieden werden im zweiten Modell besser vorausgesagt.

Spieltagsanalyse
Dieser Abschnitt untersucht erneut, wie sich die Vorhersagen über die Saison entwickelt haben. Es werden die Bereiche der Saison verglichen und analysiert, sowie an welchem Spieltag die meisten bzw. wenigsten Spiele richtig vorhersagt wurden.



(a) Spieltagsübersicht für den Trainingssatz aus drei Saisons



(b) Spieltagsübersicht für den Trainingssatz aus vier Saisons

Abbildung 12: Spieltagsübersicht des Transformers

Das erste Modell hat den erfolgreichsten Bereich in der Mitte der Saison mit 70,4 Prozent. Dies entspricht 6,3 Spielen pro Spieltag, die richtig vorhergesagt wurden. Der zweitbeste Bereich mit 66,7 Prozent ist das Ende der Saison, was einem Wert von sechs Spielen pro Spieltag gleichkommt. Den geringsten Schnitt hat der Anfang der Saison mit 58,6 Prozent, also 5,3 Spiele pro Spieltag.
Die Spieltage mit den meisten korrekten Vorhersagen waren der Spieltag 18 und 34 mit 16 Mannschaften, welches acht Spielen entspricht. Die schwächste Vorhersage eines Spiel-

tages war am siebten und zehnten Spieltag. Hier wurden nur drei Spiele korrekt vorausgesagt.
Das zweite Modell hatte den erfolgreichsten Bereich ebenfalls in der Mitte der Saison mit 70,4 Prozent, also 6,3 Spielen pro Spieltag. Es folgt der dritte Bereich der Saison mit den zweitbesten richtig vorausgesagten Spielen, nämlich mit 68,7 Prozent, also 6,2 Spielen pro Spieltag. Die Vorhersagen der Spiele am Anfang der Saison haben einen Schnitt von 67,7 Prozent, also 6,1 Spiele pro Spieltag. Also gibt das zweite Modell insgesamt ca. sechs von neun Spielen korrekt an. Die meisten Spiele werden an den Spieltagen 18, 19, 21, 25 und 31 mit acht von neun Spielen korrekt vorausgesagt. Am sechsten Spieltag werden mit vier die wenigsten Spiele richtig vorhergesagt.
Abschließend lässt sich sagen, dass das zweite Modell die Voraussagen besser und konstanter trifft, als das erste Modell. Es wurden in jedem Fall mehr Spiele korrekt vorhergesagt.

Mannschaftsanalyse
Dieser Abschnitt analysiert, welche Mannschaften am erfolgreichsten bzw. am weniger erfolgreichsten vorhergesagt wurden.
Beim ersten Modell sind der FC Bayern München und Borussia Dortmund mit 82,4 Prozent und 76,5 Prozent die am besten richtig vorausgesagten Mannschaften. Für den FC Bayern München entspricht dies 26,1 Spiele von 34 Spieltagen und für Borussia Dortmund 28 von 34 Spieltagen. Beide Mannschaften stehen am Ende der Saison oben an der Tabelle, auf dem ersten Platz für den FC Bayern und auf dem zweiten Platz für Dortmund. Die Mannschaften die am schlechtesten richtig vorhergesagt wurden, sind der SC Freiburg, VFL Wolfsburg und der 1. FC Köln mit jeweils 58,8 Prozent. Dies sind umgerechnet 20 von 34 Spieltagen. Diese drei Mannschaften stehen am Ende der Saison im Mittelfeld der Tabelle.
Beim zweiten Modell sind die am erfolgreichsten vorausgesagten Mannschaften wieder der FC Bayern München und Borussia Dortmund. Diesmal sind es aber 85,3 und 82,4 Prozent, also 29 bzw. 28 Spiele von 34 Spieltagen. Die Mannschaften, die die geringsten Durchschnitte haben, sind der SV Werder Bremen und FC Augsburg mit 58,8 Prozent bzw. 61,7 Prozent. Dies sind umgerechnet 20 bzw. 21 Spielen von 34. Die Mannschaften befinden sich in der unteren Hälfte der Schlusstabelle.

Abschließend kann festgestellt werden, dass das zweite Modell des Transformers erfolgreicher abschneidet als das erste Modell. Es werden Siege, Niederlagen und Unentschieden erfolgreicher vorhergesagt. Allerdings sagen beide Modelle öfter voraus, dass es ein Spiel unentschieden endet, als tatsächlich geschah.

7.3.3 Vergleich vom LSTM und Transformer

Dieser Abschnitt vergleicht den besten Transformer mit dem bestem LSTM bzw. alle Modelle. Es werden die *F-Measure* Werte, die Spieltage und Mannschaftswerte verglichen.

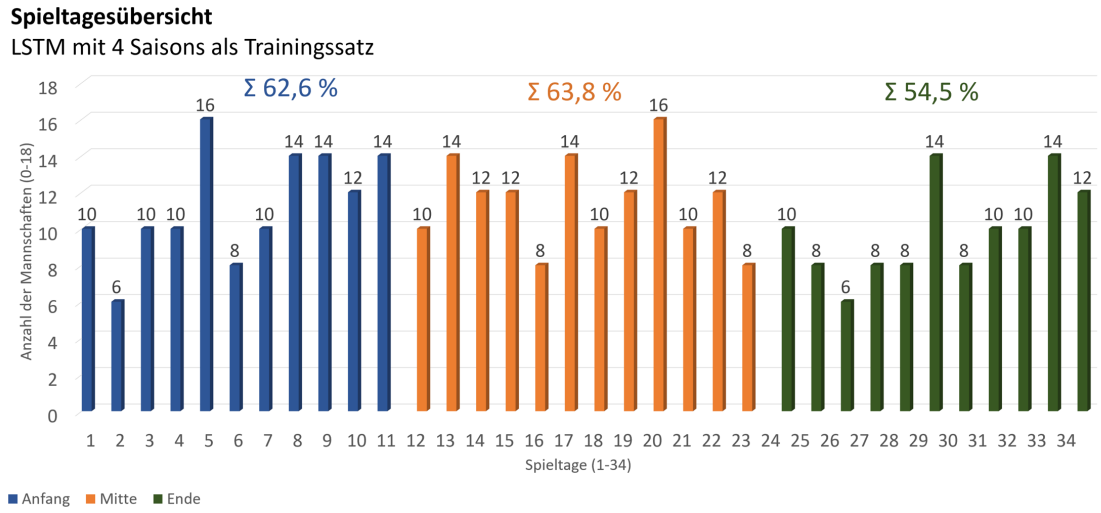
F-Measure

Modell	F-Measure Sieg	F-Measure Unentschieden	F-Measure Niederlage
LSTM	65%	47,4%	65%
Transformer	69,9%	58,2%	69,9%

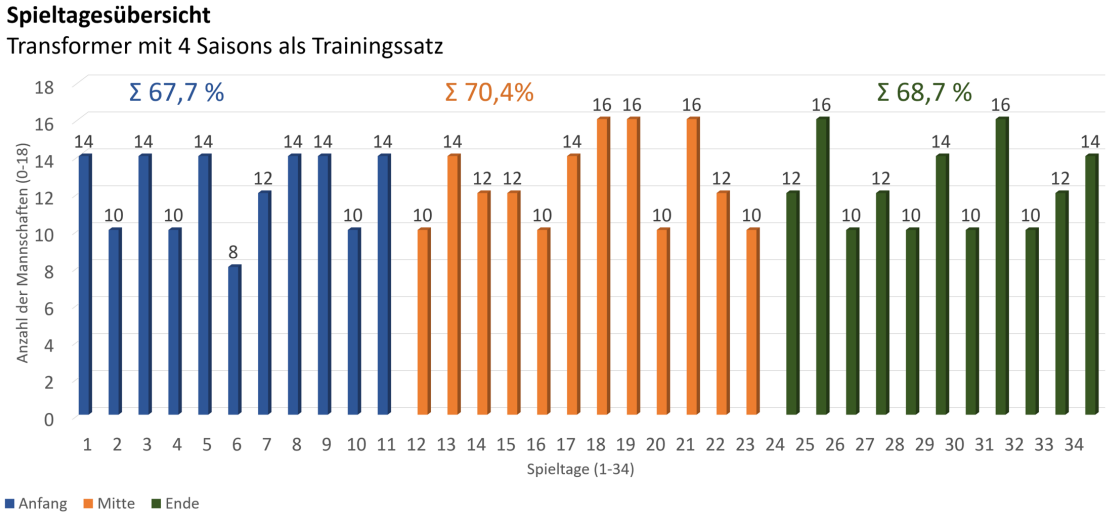
Tabelle 10: F-Measure Werte für das LSTM und Transformer Modell

Die Werte in der Tabelle 10 zeigen, dass der Transformer in den Bereichen Sieg, Niederlage und Unentschieden überwiegt. Beide Modelle schneiden im Bereich Unentschieden am schlechtesten ab und in den Bereichen Sieg und Niederlage am stärksten. Der Transformer liegt hierbei im Schnitt um 6,9 Prozent besser.

Spieltagsübersicht
Der Transformer ist in jedem Bereich der Saison erfolgreicher und hat ein höheres Minimum als das LSTM, siehe Abbildung 13.



(a) Spieltagsübersicht für das LSTM



(b) Spieltagsübersicht für den Transformer

Abbildung 13: Spieltagsübersicht des Transformers und LSTMs

Der Transformer erreicht eine *Accuracy* von 69 Prozent und das LSTM nur von 60,5 Prozent. Der erfolgreichste Bereich ist in beiden Modellen die Mitte der Saison. Das LSTM ist im letzten Bereich der Saison am schlechtesten und der Transformer am Anfang der Saison, wobei sich der Wert nur minimal vom letzten Bereich der Saison des Transformers unterscheidet.

Mannschaftsübersicht
Die erfolgreichsten Vorhersagen der Mannschaften sind bei den beiden Modellen die zwei stärksten Mannschaften der Liga, also der erste und zweite Platz. Allerdings sind die Werte des Transformers höher als beim LSTM. Die Transformerwerte befinden sich bei 82,4 und 85,3 Prozent und die LSTM Werte nur bei 70,6 Prozent und 67,6 Prozent.
Grundlegend lässt sich sagen, dass die am erfolgreichsten vorhergesagten Mannschaften die ersten beiden Plätze der Modelle belegen, bzw. die zwei schlechtesten Plätze.
Die schwächsten Vorhersagen für die Mannschaften befinden sich beim Transformer bei 58,8 und 61,7 Prozent. Die Werte beim LSTM sind wieder niedriger und liegen bei 50 und 52,9 Prozent. Bei den Mannschaften handelt es sich um Mannschaften im unterem bzw. im oberen Tabellenfeld. Es kann somit festgestellt werden, dass Mannschaften, die sich im Mittelfeld befinden, am schwierigsten vorherzusagen sind. Die Mannschaften, die sich im unterem Tabellenfeld halten, erschweren es dem Modell die richtigen Prognosen zu erstellen, da ihre Leistungen schwanken.

Modell	Erfolgreichste Mannschaft	Erfolgloseste Mannschaft
LSTM	Bayern München 70,6% Dortmund 67,6%	RB Leipzig 50% Hertha 52,9%
Transformer	Dortmund 85,3% Bayern München 82,4%	Bremen 58,8% Augsburg 61,7%

Tabelle 11: Mannschaftsübersicht für das LSTM und den Transformer

Abschließend ist festzuhalten, dass der Transformer höhere Quoten bei korrekten Vorhersagen in jedem Bereich hat und somit effektiver ist als das LSTM Modell. Zusätzlich sind Modelle mit einem größeren Trainingssatz ohne Validierungssatz erfolgreicher als Modelle mit kleinerem Trainingssatz und zusätzlichem Validierungssatz. Es wurden Mannschaften, die auf dem Weg waren abzusteigen bzw. die Meisterschaft zu gewinnen, erfolgreicher korrekt vorhergesagt, als Mannschaften die sich im Mittelfeld befanden. Zudem ist die Mitte der Saison, der erfolgreichste Bereich, um richtige Prognosen zu erstellen.

8 Fazit

In diesem Kapitel wird die Arbeit rekapituliert und die Inhalte nochmals zusammengefasst. Zudem werden die Problemstellungen betrachtet und beantwortet. Abschließend wird ein Ausblick gegeben.

8.1 Zusammenfassung und Fazit

In dieser Arbeit wurden zwei *Deep Learning* Ansätze zur Spielvorhersage von Fußballergebnissen erstellt, das LSTM- und das Transformer Modell. Dafür wurde zunächst ein Datensatz erstellt, der aus Spielstatistiken jeder Mannschaft aus früheren Spielen besteht. Zudem wurden Mannschafts- und Spielerinformationen hinzugezogen. Mannschaftsinformationen beinhalten die aktuellen Tabelleninformationen, also die für jeden Spieltag, wie beispielsweise die Platzierung oder die Anzahl an Siegen. Die Spielerinformationen schließen die aktuellen Bewertungen für jeden Spieler aus der Startelf pro Spieltag ein. Alle Informationen wurden für jedes Spiel und jede Mannschaft herangezogen und in einem Vektor gesammelt. Somit hatte jedes Spiel einen Vektor mit den Spielstatistiken, sowie Mannschafts- und Spielerinformationen beider spielenden Mannschaften.
Der Datensatz wurde in zwei Eingaben unterteilt. In der ersten Eingabe befinden sich die Zeitreihen, die aus den Spielstatistiken der letzten Spiele bestehen. Diese Eingabe wurde in die LSTM-Schicht und den Encoder des Transformers eingespeist. Die zweite Eingabe besteht aus den aktuellen Mannschafts- und Spielinformationen beider Mannschaften und wurde nach dem Auswerten der Zeitreihen in das Modell hinzugefügt. Als Ausgabe wurden bei beiden Modellen die Chancen für einen Heim- und Auswärtssieg und Unentschieden ermittelt. Diese Chancen wurden dann in Heim- oder Auswärtssieg oder Unentschieden aufgeteilt.
Anschließend wurden die Modelle mit zwei verschiedenen Trainingssätzen trainiert. Der eine Trainingssatz besteht aus drei Saisons und einer Saison als Validierungssatz, und der andere besteht aus vier Saisons und besitzt keinen Validierungssatz. Somit wurden vier verschiedene Modelle erstellt und mit einer weiteren Saison getestet.

8.1.1 Können *Long Short-Term Memory* - bzw. Transformer Modelle für Spielvorhersagen verwendet werden?

Beide *Deep Learning* Ansätze können für das Vorhersagen von Spielergebnissen verwendet werden. Diese sagen im Schnitt 65 bzw. 69,9 Prozent korrekt den Sieger bzw. Verlierer des Spiels voraus. Allerdings werden nur 47,4 bzw. 58,2 Prozent der Spiele als Unentschieden korrekt vorhergesagt.

8.1.2 Wann sind die Fußballvorhersagen der Ergebnisse während der Saison am besten?

Die Saison wurde in drei Bereiche geteilt: in den Anfang der Saison, die Mitte und in das Ende. Beim Test wurde ermittelt, dass die Mitte der Saison am erfolgreichsten richtig

vorhergesagt wurde. Der Transformer schnitt besser ab, und es wurden 70,4 Prozent der Spiele korrekt vorausgesagt, was 6,3 Spiele pro Spieltag entspricht.

8.1.3 Bei welcher Mannschaft werden die Spielergebnisse am häufigsten korrekt vorausgesagt?

Mannschaften, die sich ganz oben in der Tabelle bzw. ganz unten in der Tabelle befinden, werden am häufigsten korrekt vorausgesagt. Dabei wird eine *Accuracy* von bis zu maximal 85,3 Prozent vom Transformer erreicht und vom LSTM lediglich maximal 70,6 Prozent.

8.1.4 Welches Modell eignet sich besser zu Spielvorhersagen?

Das erfolgreichere Modell ist der Transformer. Dieser schneidet in jedem Bereich besser ab als das LSTM. Zudem ist zu erwähnen, dass sich ein größerer Trainingssatz besser eignet, als ein kleinerer Trainingssatz mit Validierungssatz.

Modell	F-Measure Sieg	F-Measure Unentschieden	F-Measure Niederlage
LSTM	65%	47,4%	65%
Transformer	69,9%	58,2%	69,9%

Tabelle 12: F-Measure Werte für das LSTM und Transformer Modell

Nicht nur in dieser Arbeit schneidet der Transformer bei der Vorhersage von Spielergebnissen besser ab, sondern er ist mit seinen Ergebnissen auch zutreffender als die Techniken, die in den verwandten Arbeiten beschrieben wurden und dabei maximal 60 Prozent korrekte Vorhersagen erreichten.

8.2 Ausblick

In dieser Arbeit wird deutlich, dass der Transformer bei Vorhersagen von Fußballergebnissen erfolgreicher abschneidet als das LSTM. Somit ist der *Attention* Mechanismus besser als die LSTM Technik.
Der Transformer kann noch weiter entwickelt werden, indem andere Informationen hinzugefügt werden. Dies könnten Spiele weiterer Saisons der 1. Bundesliga sein, aber auch aus anderen internationalen und nationalen Ligen. Hierbei könnte beispielsweise die Premier Legaue aus England in Betracht gezogen werden. Zudem könnten noch andere Informationen zur Mannschaft, zum Spiel oder zu den Fußballspielern eingefügt werden. Der Datensatz könnte erweitert werden, indem die Trainerinformation detaillierter berücksichtigt werden. Nach einem Trainerwechsel mitten in der Saison könnte eine Mannschaft anders spielen und somit andere Ergebnisse erzielen. Auch könnten Interviews von den Spielern oder Trainern vor dem Spiel hinzugezogen werden. Mit Text Mining könnte heraus gefunden werden, wie die Mannschaft zum jetzigen Spiel steht, etwa ob sie sich als Favorit sieht oder nicht.
Der Transformer kann zudem in anderen Sportbereichen Anwendung finden. Jede

Mannschaftssportart kann sogar auf die gleiche Weise, wie der Fußball, bearbeitet werden. Basketball hat zum Beispiel ebenfalls Spieler mit Wertungen, Spielstatistiken, Tabelleninformationen und Wettquoten zu jedem Spiel.
Außerdem wären noch andere Anwendungsbereiche möglich. So könnten Wettervorhersagen für verschiedene Regionen ermittelt werden. Dafür würden Informationen zu den vorherigen Wettervorhersagen gesammelt. Dies könnten etwa die Temperatur, Windstärke, Luftfeuchtigkeit, Niederschlagsmenge, Regenwahrscheinlichkeit etc. sein und würden zusammen als Zeitreihen fungieren. Diese Zeitreihen bestünden dann aus einer Menge von Tagen mit deren Wetterberichten. Als Zusatzinformationen müsste dann das Gebiet der Wettervorhersage festgelegt werden, wie zum Beispiel New York oder Buenos Aires, und hätte somit den regionalen Aspekt. Dadurch würden die Zeitreihen nochmals anders interpretiert. Zusätzlich könnte noch die Jahreszeit hinzugefügt werden, um die Wettervorhersage genauer einzuordnen.

Danksagung

Abschließend möchte ich all denjenigen meinen Dank aussprechen, welche mich während meiner Masterarbeit unterstützt haben.

Herrn Prof. Dr. Stefan Conrad und Herrn Prof. Dr. Gunnar W. Klau bedanke ich mich für Thema dieser Arbeit.

Besonderer Dank gebührt auch meinem Betreuer, Philipp Grawe, für seinen Rat und seine zuverlässige und hilfreiche Unterstützung.

Im Speziellen möchte ich mich bei Sibylle Constant und Sophie Heggemann für das Korrekturlesen meiner Masterarbeit bedanken.

Abschließend gilt der Dank meiner Familie, welche mich stets unterstützt und ermutigt haben.

Literatur

Neda Abdelhamid, Aladdin Ayesh, Fadi Thabtah, Samad Ahmadi und Wael Hadi (2012). „MAC: A multiclass associative classification algorithm“. In: *Journal of Information & Knowledge Management* 11.02, S. 1250011.

Dzmitry Bahdanau, Kyunghyun Cho und Yoshua Bengio (2014). „Neural machine translation by jointly learning to align and translate“. In: *arXiv preprint arXiv:1409.0473*.

Yoshua Bengio (2012). „Practical recommendations for gradient-based training of deep architectures“. In: *Neural networks: Tricks of the trade*. Springer, S. 437–478.

Abdulkadir Cevik und Ibrahim H Guzelbey (2008). „Neural network modeling of strength enhancement for CFRP confined concrete cylinders“. In: *Building and Environment* 43.5, S. 751–763.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk und Yoshua Bengio (2014). „Learning phrase representations using RNN encoder-decoder for statistical machine translation“. In: *arXiv preprint arXiv:1406.1078*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee und Kristina Toutanova (2018). „Bert: Pre-training of deep bidirectional transformers for language understanding“. In: *arXiv preprint arXiv:1810.04805*.

Ian Goodfellow, Yoshua Bengio und Aaron Courville (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press.

Karol Gregor, Ivo Danihelka, Andriy Mnih, Charles Blundell und Daan Wierstra (2014). „Deep autoregressive networks“. In: *International Conference on Machine Learning*. PMLR, S. 1242–1250.

Enzo Grossi und Massimo Buscema (2007). „Introduction to artificial neural networks“. In: *European journal of gastroenterology & hepatology* 19.12, S. 1046–1054.

S Haykin (1999). „Neural Networks: A Comprehensive Foundation 2nd edition, Prentice Hall“. In:

Geoffrey E Hinton, Terrence Joseph Sejnowski et al. (1999). *Unsupervised learning: foundations of neural computation*. MIT press.

Cong Duy Vu Hoang, Trevor Cohn und Gholamreza Haffari (2016). „Incorporating side information into recurrent neural network language models“. In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, S. 1250–1255.

Sepp Hochreiter (1998). „The vanishing gradient problem during learning recurrent neural nets and problem solutions“. In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.02, S. 107–116.

Sepp Hochreiter und Jürgen Schmidhuber (1997). „Long short-term memory“. In: *Neural computation* 9.8, S. 1735–1780.

Uday Kamath, John Liu und James Whitaker (2019). *Deep learning for NLP and speech recognition*. Bd. 84. Springer.

Seyed Mehran Kazemi, Rishab Goel, Sepehr Eghbali, Janahan Ramanan, Jaspreet Sahota, Sanjay Thakur, Stella Wu, Cathal Smyth, Pascal Poupart und Marcus Brubaker (2019). „Time2vec: Learning a vector representation of time“. In: *arXiv preprint arXiv:1907.05321*.

Keras (2021). *Dense layer*. URL: https://keras.io/api/layers/core_layers/dense/ (besucht am 17. 05. 2021).

Keras Team (2021). *Keras Tuner*. URL: https://keras.io/api/layers/core_layers/dense/ (besucht am 17. 05. 2021).

Diederik P Kingma und Jimmy Ba (2014). „Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980*.

Minh-Thang Luong, Hieu Pham und Christopher D Manning (2015). „Effective approaches to attention-based neural machine translation". In: *arXiv preprint arXiv:1508.04025*.

Frank Meyrahn, Iris an der Heiden, Gerd Ahlert und Holger Preuß (2014). „Wirtschaftsfaktor Sportwetten – Sportfaktor Lotterien". In: *BMWi*.

Rami M Mohammad, Fadi Thabtah und Lee McCluskey (2015). „Tutorial and critical analysis of phishing websites methods". In: *Computer Science Review* 17, S. 1–24.

Rami M Mohammad, Fadi Thabtah und Lee McCluskey (2016). „An improved self-structuring neural network". In: *Pacific-Asia conference on knowledge discovery and data mining*. Springer, S. 35–47.

Vinod Nair und Geoffrey E Hinton (2010). „Rectified linear units improve restricted boltzmann machines". In: *Icml*.

P Russel Norvig und S Artificial Intelligence (2002). *A modern approach*. Prentice Hall Upper Saddle River, NJ, USA:

Daniel Pettersson und Robert Nyquist (2017). „Football match prediction using deep learning". In: *Psychol. Sport Exerc.* 15.5, S. 538–547.

David MW Powers (2020). „Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation". In: *arXiv preprint arXiv:2010.16061*.

Darwin Prasetio et al. (2016). „Predicting football match results with logistic regression". In: *2016 International Conference On Advanced Informatics: Concepts, Theory And Application (ICAICTA)*. IEEE, S. 1–5.

Russell Reed und Robert J MarksII (1999). *Neural smithing: supervised learning in feedforward artificial neural networks*. Mit Press.

Brian D Ripley (1996). *Pattern recognition and neural networks*. Cambridge university press.

Stefan Samba (2019). „Football Result Prediction by Deep Learning Algorithms". Diss. Tilburg University.

Rohan Challa Sebastien Goddijn Evgeny Moshkovich (2018). „A Sure Bet: Predicting Outcomes of Football Matches". In:

Alex Sherstinsky (2020). „Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network". In: *Physica D: Nonlinear Phenomena* 404, S. 132306.

Leslie N Smith (2017). „Cyclical learning rates for training neural networks". In: *2017 IEEE winter conference on applications of computer vision (WACV)*. IEEE, S. 464–472.

N Tax und Y Joustra (o.D.). „Predicting The Dutch Football Competition Using Public Data: A Machine Learning Approach. 10, 2015". In: *Transactions of knowledge and data engineering* 10 ().

Ben Ulmer, Matthew Fernandez und Michael Peterson (2013). „Predicting soccer match results in the english premier league". Diss. Doctoral dissertation, Ph. D. dissertation, Stanford.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser und Illia Polosukhin (2017). „Attention is all you need“. In: *arXiv preprint arXiv:1706.03762*.

Bill Wilson (1998-2012). *The Machine Learning Dictionary*. URL: https://www.cse.unsw.edu.au/~billw/mldict.html (besucht am 12. 05. 2021).

Ian H Witten, Eibe Frank, Mark A Hall und Christopher J Pal (2005). „Practical machine learning tools and techniques“. In: *Morgan Kaufmann*, S. 578.

Albert Zeyer, Parnia Bahar, Kazuki Irie, Ralf Schlüter und Hermann Ney (2019). „A comparison of transformer and lstm encoder decoder models for asr“. In: *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE, S. 8–15.

Abbildungsverzeichnis

Tabellenverzeichnis