Story Teller

Time series forecasting

Bachelorarbeit

ausgearbeitet von

Kamil Uğur Sarp

zur Erlangung des akademischen Grades
Bachelor of Science (B.Sc.)

vorgelegt an der

Türkisch-Deutschen Universität
Fakultät für Ingenieurwissenschaften

im Studiengang
Informatik

Betreuer/in: Dr. Canan YILDIZ
Türkisch-Deutsche Universität

Istanbul, im August 2022

Inhaltsverzeichnis

Kurzfassung

Deep-Learning-Algorithmen wurden zu den erfolgreichsten Methoden für Aufgaben der natürlichen Sprachverarbeitung (NLP). Modelle, die erfolgreich mit sequenziellen Daten arbeiten, wie RNN und LSTM, bieten eine einfache und verständliche Lösung für solche NLP-Probleme. Ein rekurrentes neuronales Netz (RNN) ist eine Art künstliches neuronales Netz, das für die Arbeit mit sequentiellen Daten entwickelt wurde, die für Klassifizierungs- und Regressionsprobleme verwendet werden. In dieser Arbeit werden verschiedene Arten von rekurrenten neuronalen Netzen verwendet und beschrieben. Diese Algorithmen werden als Basismodell für unsere Zeitreihenprognose verwendet. Sie haben einen erheblichen Einfluss auf den Lernprozess. Ziel der Arbeit ist es, einen gegebenen Satz mit Hilfe von RNN und LSTM in einer Sprache wie Türkisch, die nicht so reich an Wörtern ist wie Englisch und Deutsch, und die von den anderen Sprachen in Bezug auf den Sprachbaum getrennt ist, erfolgreich mit dem richtigen nächsten Wort zu vervollständigen.

Schlüsselwörter: Maschinelles Lernen, rekurrente neuronale Netze,
Langzeit-Kurzzeitgedächtnis, Feed Forward Neuronales Netz, Wortvorhersage

Abstract

Deep learning algorithms became the most successful methods for Natural Language Processing(NLP) tasks. Models that work successfully with sequential data, such as RNN and LSTM, offer an easy and understandable solution to such NLP problems. A recurrent neural network(RNN) is a type of artificial neural network designed to work with sequential data used for classification and regression problems. Different types of Recurrent Neural Networks are used and described in the thesis. These algorithms are used as a baseline model for our time series forecasting. They have a substantial effect on the learning process. The purpose of the thesis is to successfully complete a given sentence with the right next word with the help of RNN and LSTM in a language such as Turkish, which is not so rich in terms of the number of words such as English and German, and which is separated from the other languages in terms of language tree.

Keywords: Machine Learning, Recurrent Neural Networks, Long-Short Term Memory, Feed Forward Neural Network, Word Prediction

1 Einleitung

In den letzten Jahren hat die Popularität des maschinellen Lernens und seine Anwendung im täglichen Leben erheblich zugenommen. Die Vorhersage des nächsten Wortes ist eines der Probleme, die bei der Bearbeitung von NLP-Problemen auftreten und die von der Textsequenz und der Grammatik der Sprache abhängen, mit der wir arbeiten. Der Hauptzweck dieser Arbeit besteht darin, korrekte Annahmen über die Zeitreihenvorhersage mit Hilfe eines neuronalen Netzes zu treffen und die genaue Wortvorhersage entsprechend dieser Annahmen zu erstellen. Die Arbeit besteht aus zwei Hauptteilen: dem theoretischen und dem praktischen Teil. Jeder Teil besteht aus mehreren Abschnitten.
Der erste Abschnitt enthält Neuronale Netze. In diesem Abschnitt wird untersucht, was neuronale Netze sind, wie sie funktionieren und woraus sie bestehen. Parameter und Hyperparameter werden in diesem Abschnitt erklärt.
Im folgenden Teil wird Deep Learning vorgestellt. In diesem Abschnitt geht es um die Strukturen, die unserem Algorithmus zu Grunde liegen. In diesem Abschnitt werden die Geschichte der Algorithmen für rekurrente neuronale Netze und ihre mathematischen Grundlagen erläutert. In diesem Abschnitt werden kurz die Unterschiede zwischen anderen überwachten maschinellen Lernanwendungen und RNN beschrieben. Auf diese Weise werden wir schnell die Unterschiede wie Vor- und Nachteile erkennen. Probleme, die bei der Arbeit mit diesem Algorithmus auftreten können, werden später in diesem Abschnitt beschrieben. Es wird auch erklärt, wie man diese Probleme von vornherein vermeiden kann und wie man sie lösen kann, wenn man auf diese Probleme stößt. Außerdem wurde eine kurze Erläuterung der Netzwerke des Langzeitgedächtnisses gegeben. Außerdem wird untersucht, wie diese Algorithmen funktionieren und wie sie Vorhersagen treffen.
Der vierte Abschnitt enthält verwandte Arbeiten. In diesem Abschnitt werden die Beispiele für die Texterstellung und die Artikel zur Vorhersage des nächsten Wortes untersucht, die mit dem Algorithmus des rekurrenten neuronalen Netzes oder einem anderen Algorithmus, der für diesen Zweck verwendet werden kann, erstellt wurden. Die Schlussfolgerungen, die ich daraus ziehe, werden sich darauf beziehen, was bei diesen Projekten hätte besser gemacht werden können und was ich aus diesen Beispielen in meine Arbeit einbringen kann.

Im fünften Kapitel werden diese Algorithmen in unserem Programm implementiert. Am Ende der Arbeit werden die Ergebnisse ausgewertet und Schlussfolgerungen gezogen.

2 Deep Learning

Bevor wir RNN eingehend erklären, müssen wir wissen, was Deep Learning ist, wo es eingesetzt wird und welche Art von Problemen damit gelöst werden. Deep Learning ist ein Teil des maschinellen Lernens, der künstliche neuronale Netze verwendet. Der Lernprozess kann in drei Teile unterteilt werden: überwachtes, halbüberwachtes und unüberwachtes Lernen.
Deep Learning ist ein wesentlicher Bereich der Datenwissenschaft. Es ermöglicht eine schnellere und einfachere Verarbeitung der Daten bei der Erfassung und Analyse großer Datenmengen. Deep Learning wird häufig bei der maschinellen Übersetzung, der Verarbeitung natürlicher Sprache, der Bildverarbeitung usw. eingesetzt. Beim maschinellen Lernen erfolgt dieser Prozess durch die Kennzeichnung der Daten und die Mitteilung an das Modell, welche dieser Daten die gewünschte Ausgabe enthalten und welche nicht. Die Merkmalsextraktion erfolgt beim Deep Learning automatisch und ist daher schneller und genauer.
Um all dies zu tun, müssen wir unser Modell mit den richtigen Daten füttern. Auf diese Weise kann das Programm eine genaue Bewertung vornehmen und seinem Zweck entsprechend arbeiten. Der Prozess der Dateneingabe lässt sich wie folgt zusammenfassen. Die Daten sollten in zwei Teile aufgeteilt werden, und zwar in Trainings- und Testdaten. So erhalten wir einen Trainingsdatensatz zum Lernen des gewählten Algorithmus und zur Anwendung der erforderlichen Parameter sowie einen Testdatensatz zur späteren Auswertung. Mit dem Testdatensatz wird nicht gelernt. Mit diesem Datensatz soll festgestellt werden, wie genau unser Programm arbeitet, und es soll sichergestellt werden, dass die Hyperparameter richtig gewählt sind.
Der Prozess, der beim überwachten Lernen durchgeführt wird, ist der Klassifizierungsprozess. Wir haben also den Ausgabewert im Datensatz, das Ziel und die Eingabewerte. Während des Trainings wird versucht zu lernen, wie diese Zielwerte erreicht werden können.
Beim unüberwachten Lernen gibt es keinen Zielwert im Datensatz. Hier versucht der Algorithmus, die Beziehung zwischen den einzelnen Eingaben zu verstehen. Anschließend gruppiert er die Daten, die einander nahe stehen, entsprechend dem Algorithmus.
Das Verstärkungslernen unterscheidet sich von den beiden anderen Lernmethoden. Es gibt ein Belohnungs- und Bestrafungssystem. Das System wird abhängig von der Ent-

scheidung belohnt oder bestraft. Daher muss es lernen, wie es selbst das beste Ergebnis erzielen kann.
In den meisten Quellen wird auch das halb-überwachte Lernen erwähnt. Bei dieser Methode enthält nicht jede Eingabe ihren Ausgabewert im Datensatz. Einige Algorithmen können mit diesen Daten arbeiten. Dies wird als halb-überwachtes Lernen bezeichnet. In dieser Arbeit werden RNN und LSTM verwendet. Um diese beiden Algorithmen zu verstehen, müssen wir die überwachten und unüberwachten Methoden kennen.

2.1 Neural Networks

Das Neural Network ist ein maschinelles Lernverfahren, das versucht, das menschliche Gehirn zu simulieren. Die Zellen des menschlichen Gehirns werden in ANNs als Neuronen bezeichnet. Diese Neuronen sind wie in unserem Gehirn miteinander verbunden. Die Verbindungen in unseren Neuronen werden in künstlichen neuronalen Netzen als Gewichte zwischen den Knoten modelliert. Ein künstliches neuronales Netz besteht strukturell aus einer Eingabeschicht, einer Ausgabeschicht und einer oder mehreren verborgenen Schichten. Die Neuronen in einer Schicht sind mit den Neuronen in der folgenden darauf Schicht verbunden.
Der Lernprozess ergibt sich aus den mathematischen Operationen zwischen den einzelnen verbundenen Knoten. In diesem Zusammenhang werden der Eingabewert und das Gewicht zwischen zwei Neuronen multipliziert. Liegt das Ergebnis dieser Berechnung über einem bestimmten Schwellenwert, werden die Daten an das nächste Neuron weitergegeben. Dieser Berechnungsprozess wird so lange wiederholt, bis der Ausgang erreicht ist. Ein künstliches neuronales Netz kann sich schnell an die Veränderungen der Eingaben anpassen. Es ermöglicht es, Arbeiten, die bei manueller Ausführung Stunden dauern würden, in wenigen Minuten zu erledigen. Sie können in einer Vielzahl von Anwendungen im wirklichen Leben eingesetzt werden. Die häufigsten Anwendungen von neuronalen Netzen sind:

- Vorhersage von Börsenkursen
- Erkennung von Gesichtern
- Erkennung von Betrug
- Spracherkennung
- Verarbeitung natürlicher Sprache

2.1.1 Hyper Parameter

Hyperparameter sind Parameter, die den Lernprozess steuert. Hyperparameter werden festgelegt, bevor der Lernprozess beginnt. Sie liegen außerhalb des Modells, und Hyperparameter können während des Trainings nicht geändert werden. Das Training wird mit diesen Hyperparametern durchgeführt, aber wir können nicht wissen, welche Parameter verwendet werden oder wie sie unser Modell beeinflussen. Wir wissen nur, dass unser Modell mit diesen Parametern gelernt hat.
Als Beispiele für Hyperparameter können folgende angeführt werden:

- Training/Test-Verhältnis
- Lernrate
- Optimierungsalgorithmus
- Aktivierungsfunktion
- Anzahl der versteckten Schichten
- Epoche
- Batchgröße

Was jeder dieser Parameter ist, wird im Detail in den jeweiligen Kapiteln erklärt.

Layers

In einem Modell für tiefes Lernen wird die Struktur, die die Informationen aus dem vorherigen Zustand aufnimmt und dann an den nächsten Zustand weiterleitet, als Schicht bezeichnet. Eine Schicht ist eine Struktur, die gewichtete Eingaben erhält, diese Gewichte entsprechend ihrer Aktivierungsfunktion verarbeitet und diese Werte dann an die nächste Schicht weitergibt. Dieser Prozess wiederholt sich, bis die Ausgabeschicht erreicht ist. Eine Schicht kann nur eine Aktivierungsfunktion enthalten. Die erste Schicht, die die Eingabewerte aufnimmt, wird als Eingabeschicht bezeichnet, die Schicht, die den endgültigen Wert ausgibt, als Ausgabeschicht, und alles, was dazwischen liegt, wird als versteckte Schicht bezeichnet.
Jede Verbindung zwischen zwei Neuronen hat ein Gewicht. Jedes Gewicht stellt die Korrelation zwischen diesen beiden Knoten dar. Wenn die Eingabe in einem Knoten der Eingabeschicht ankommt, wird diese Eingabe an den nächsten Knoten mit der Verbindung zwischen diesen beiden Knoten weitergeleitet. Während dieser Wert an den anderen Knoten weitergeleitet wird, wird dieser Wert mit dem Gewicht der Verbindung multipliziert. Dieser Vorgang wird so lange wiederholt, bis alle Werte an die nächste

Schicht weitergegeben wurden. Diese gewichteten Werte werden summiert und an eine Aktivierungsfunktion für jeden Knoten weitergeleitet. Die Aktivierungsfunktion transformiert diese hinzugefügten Gewichte abhängig von der Aktivierungsfunktion.

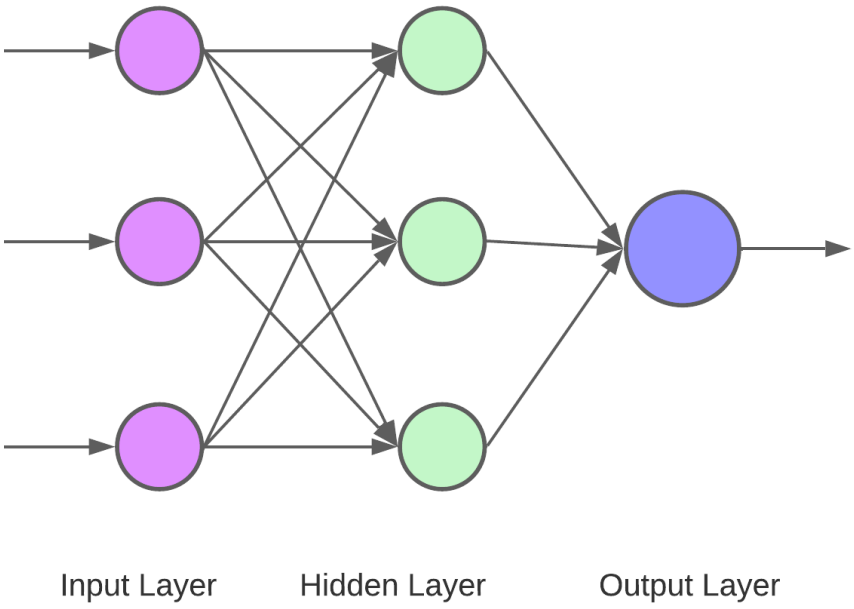

Abbildung 2.1: Eine einfache Schichtstruktur mit einer verborgenen Schicht (source: K Uğur Sarp)

Embedding Layer

Wenn wir mit Textdaten arbeiten, müssen wir diese Wortwerte in Zahlen umwandeln. Zu diesem Zweck können wir die One-Hot-Codierung verwenden. Dies kann jedoch sehr platzraubend sein, denn wenn wir 100.000 Daten haben, müssen wir jedes dieser Wörter mit 0en und 1en füllen. Für 100.000 Daten benötigen wir eine 100.000 x 100.000 Matrix. Mit der Einbettungsschicht können wir jedes Wort in einen Vektor definierter Länge umwandeln. Auf diese Weise haben alle Wörter tatsächliche Werte anstelle von 0en und 1en.

Dropout Layer

Die Dropout-Schicht dient hauptsächlich dazu, eine Überanpassung unseres Modells an die Trainingsdaten zu verhindern. Die Dropout-Schicht setzt die Kanten der versteckten Einheiten mit einer bestimmten Wahrscheinlichkeit auf Null.

Dense Layer

Die dichte Schicht ist vollständig mit den Neuronen der vorhergehenden Schicht verbunden, was bedeutet, dass die dichte Schicht alle Ausgabewerte der letzten Schicht erhält. Die dichte Schicht führt im Hintergrund eine Matrixmultiplikation durch.

Activation Function

Das Activation Function hilft dem neuronalen Netz, komplexe Muster zu lernen. Die Aktivierungsfunktion setzt die Werte aus der vorherigen Schicht gemäß der angegebenen Funktion in einen Bereich und überträgt die neuen Werte an die nächste Schicht. Sie nimmt den Ausgabewert und wandelt ihn in einen Wert um, der als Eingabe für das nächste Schicht verwendet werden kann. Sie hilft auch dabei, die Werte der Ausgänge auf eine Grenze zu beschränken, je nachdem, welche Aktivierungsfunktion verwendet wird. Da das Gewicht der Zelle und der Eingabewert multipliziert werden, wird der Bias dazu addiert. Wenn es keine Beschränkung gibt, könnte dieser Wert exponentiell ansteigen. Dies kann beim Training des Netzes zu Problemen führen.

Relu (Rectified Linear Unit)

Relu kann als f(x) = max(0,x) dargestellt werden. Das bedeutet, dass jeder Wert, der kleiner als Null ist, als Null übergeben wird, und dass jeder Wert, der größer als Null ist, als sein Wert eingegeben wird. Da die Ableitung einer positiven Zahl 1 ist, arbeitet relu rechnerisch besser als die meisten Aktivierungsfunktionen. Aus diesem Grund wird sie bei der Arbeit mit tiefen neuronalen Netzen häufig ausgewählt.

Softmax

Softmax wird bei Mehrklassen-Klassifizierungsproblemen verwendet. Es wandelt die Werte im Vektor in einen Wahrscheinlichkeitsvektor zwischen 0 und 1 um. Der Wert mit der höchsten Wahrscheinlichkeit wird auf 1 gesetzt und die anderen auf 0.

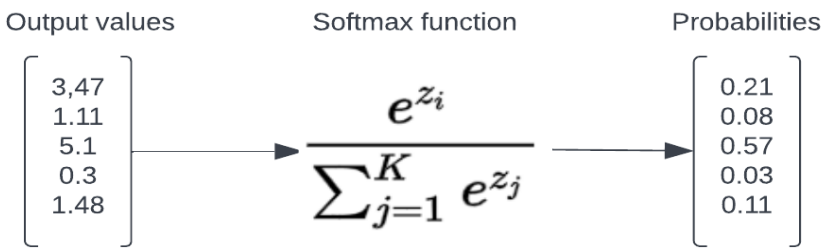

Abbildung 2.2: Konvertieren von Ausgabewerten in Wahrscheinlichkeiten (source: K Uğur Sarp)

Learning Rate

Das Learning Rate ist ein Feinabstimmungsparameter, der von Optimierungsalgorithmen beim maschinellen Lernen verwendet wird. Die Lernrate kann als der Schritt bei jeder Iteration in Richtung minimaler Verlust erklärt werden. Sie ist ein Parameter, der angibt, wie schnell das Modell lernt. Bei der Lernrate gibt es einen Kompromiss. Wenn wir eine sehr hohe Lernrate wählen, kann es sein, dass das Modell über das Minimum springt, wodurch es nie zur optimalen Lösung konvergiert. Wählt man eine niedrige Lernrate, da jeder Schritt am Ende jeder Iteration zu klein ist, dauert es zu lange, bis ein optimales Minimum erreicht wird. Die Lernrate ist ein wichtiger Parameter und muss sorgfältig gewählt werden, um eine optimale Lösung zu erreichen.

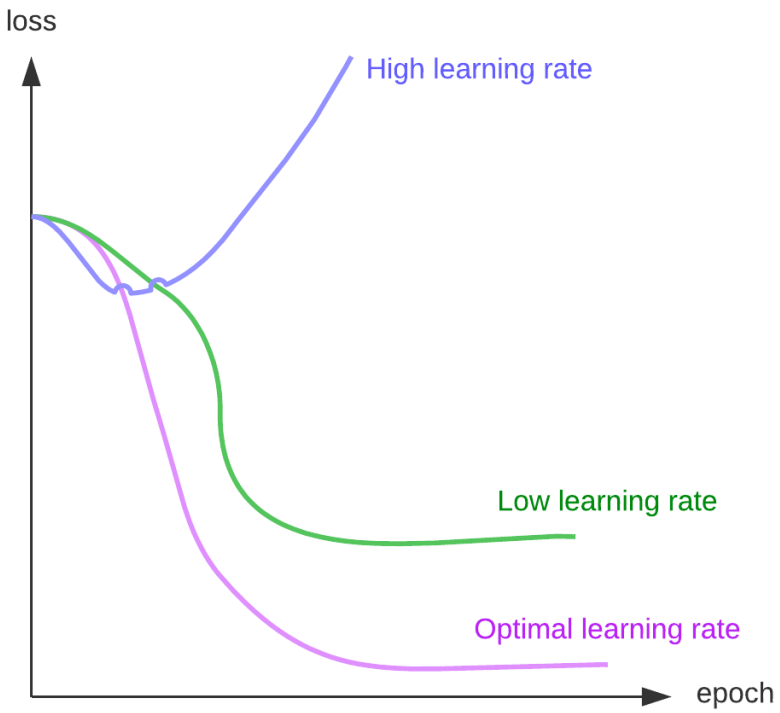

Abbildung 2.3: Auswirkungen auf die Lernrate (source: K Uğur Sarp)

Wie in Abbildung 2.3 zu sehen ist, benötigt eine niedrige Lernrate zu viele Epochen, um den minimalen Verlust zu erreichen; andererseits erreicht eine hohe Lernrate nie das Optimum.

Loss Function

Bei der Optimierung versuchen wir, bestimmte Funktionen zu maximieren oder zu minimieren. Bei der Arbeit mit einem neuronalen Netz versuchen wir, den Fehler zu minimieren. Diese Art von Funktion wird oft als Kosten- oder Verlustfunktion bezeichnet. Der Wert, der durch die Verlustfunktion berechnet wird, wird als Verlust bezeichnet.

Categorical Cross-Entropy

Es handelt sich um eine Verlustfunktion für Mehrklassen-Klassifizierungsaufgaben. Das Modell muss nur einen Wert unter allen Zielwerten vorhersagen.

$$Loss = -\sum_{i=1}^{n}(y_i * log\hat{y}_i)$$

Der Verlust wird berechnet:

- Jeder Zielwert wird mit dem Logarithmus des vorhergesagten Wertes multipliziert
- Jeder der berechneten Werte wird summiert
- Indem das Negative des summierten Werts genommen wird, wird ein Verlust berechnet

Sparse Categorical Cross-Entropy

Das Zielwertformat ist der einzige Unterschied zwischen der spärlichen kategorialen Kreuzentropie und der kategorialen Kreuzentropie. Bei der spärlichen kategorialen Kreuzentropie können die Zielwerte ganzzahlig und nicht 0 oder 1 sein. Die Verlustberechnung ist dieselbe.

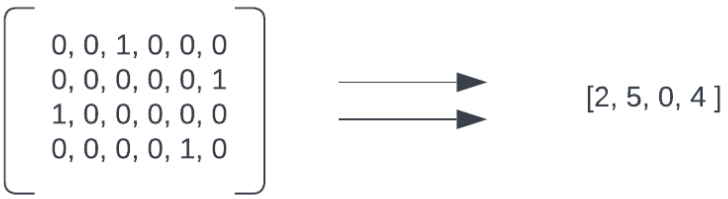

Abbildung 2.4: Zielwerte geschrieben als ganze Zahlen (source: K Uğur Sarp)

Optimiser

Ein Optimierer ist ein Algorithmus, der dazu dient, den Fehler zu minimieren oder die Genauigkeit zu maximieren. Es handelt sich um mathematische Funktionen, die von Parametern wie Gewichten, Verzerrungen usw. abhängen. Der Optimierer hilft bei der Bestimmung, wie die Gewichte geändert werden müssen oder wie sich die Lernrate verhalten soll.

RMSprop

Es handelt sich um eine gradientenbasierte Technik. Gradienten neigen dazu, zu verschwinden oder zu explodieren. Rmsprop wurde als stochastisches Verfahren entwickelt. Es befasst sich mit dem Problem, den Durchschnitt eines quadratischen Gradienten zu verschieben, um die Gradienten zu normalisieren. Diese Normalisierung verhindert, dass Gradienten verschwinden oder explodieren.

Adam

Adam verwendet eine gradientenbasierte Technik. Es verwendet einen exponentiell gewichteten Durchschnitt der Gradienten, um das Optimum zu erreichen. Durch die Verwendung dieser Durchschnittswerte erreicht Adam diesen optimalen Wert schneller.

Batch Size

Die Stapelgröße legt fest, wie viele Proben in einer Iteration durch das Netz geleitet werden. Der Stapel kann eine oder mehrere Proben enthalten. Nach jeder Iteration wird mit den Beispielen dieser Charge eine Vorhersage gemacht, mit den tatsächlichen Werten verglichen und dann der Fehler berechnet. Je nach Stapelgröße gibt es drei verschiedene Stapel.

- Batch-Gradientenabstieg: Batchgröße = Größe der Trainingsmenge
- Stochastischer Gradientenabstieg: Stapelgröße = 1
- Mini-Batch Gradient Descent: 1 < Batchgröße < Größe des Trainingssets

Wenn die Stapelgröße klein ist, kann das Modell bessere Vorhersagen machen, aber der Nachteil ist, dass es mehr Zeit zum Trainieren braucht und die Trainingsdaten möglicherweise übererfüllt. Ist die Stapelgröße groß, ist das Modell weniger genau, benötigt aber weniger Zeit und verhindert eine Überanpassung des Modells.

Epoch

Die Epoche ist die Zahl, die angibt, wie oft das Modell auf dem gesamten Trainingsdatensatz ausgeführt wird. Durch die Stichproben im Trainingsdatensatz werden die Parameter in einer Epoche aktualisiert. Eine Epoche kann einen oder mehrere Stapel enthalten. Wenn die Anzahl der Epochen erhöht wird, steigt die Genauigkeit, aber es ist nicht immer garantiert, dass das Modell besser wird, und es wird höchstwahrscheinlich zu einer Überanpassung kommen.

2.2 Feedforward Propagation

Wie der Name schon sagt, ist ein künstliches neuronales Netz dem menschlichen Gehirn nachempfunden und zielt darauf ab, es zu stimulieren und wie dieses zu funktionieren. Neuronen bilden die Grundlage des Netzes, das auf diese Struktur abzielt. Neuronen sind durch ihr Gewicht mit anderen Neuronen verbunden. Bei Feedforwards ist die Richtung nur vorwärts; die Strukturen, die eine Rückkehr zur vorherigen Schicht ermöglichen, werden rekurrente neuronale Netze genannt. Die Richtung beginnt am Eingang und bewegt sich zum Ausgang. Die Gewichte, die in die jeweils nächste Schicht wandern, werden mit dem Gewicht in dieser Schicht multipliziert. Dann wird die Fehlerspanne zu diesem Ergebnis addiert und das Gewicht in dieser Schicht aktualisiert. Auf der Eingabeschicht wird keine Berechnung durchgeführt. Im Allgemeinen ist die Struktur schichtweise aufgebaut. Je nach der Anzahl der verwendeten Schichten wird sie als ein- oder mehrschichtig bezeichnet.

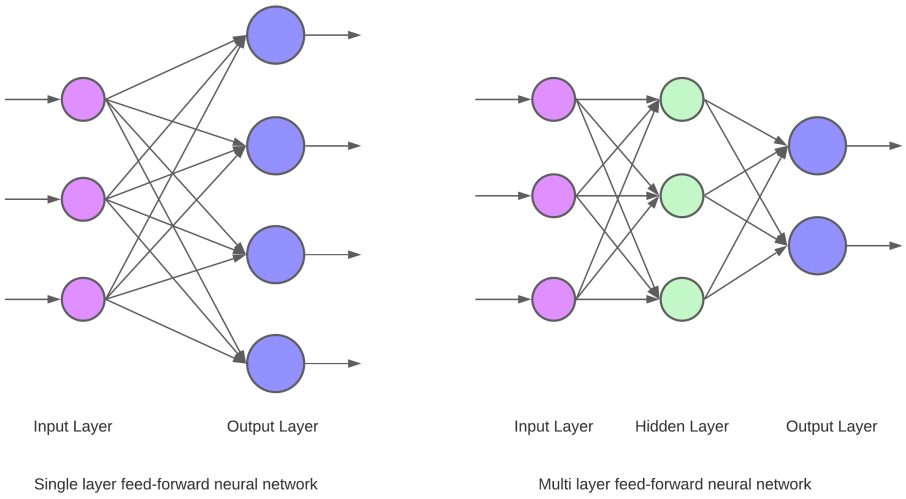

Abbildung 2.5: Single- und Multi-Layer Feed-Forward Netzwerk [5]

Wie im obigen Beispiel zu sehen ist, handelt es sich auf der linken Seite um ein einschichtiges neuronales Netz mit Vorwärtssteuerung. Auf der rechten Seite sehen wir die mehrschichtige Struktur mit einer versteckten Schicht. Der Unterschied zwischen mehrschichtigen neuronalen Netzen mit Vorwärtskopplung und einschichtigen Netzen besteht darin, dass mehrschichtige Netze mindestens eine versteckte Schicht enthalten. Mit zunehmender Anzahl versteckter Schichten nimmt auch die Verarbeitungsleistung dieser Netzstruktur zu, was jedoch zu einer Verlängerung der Verarbeitungszeit führen kann. Die beiden obigen Beispiele zeigen eine vollständig verbundene Struktur. Wir könnten diese Netzstruktur als teilweise verbunden bezeichnen, wenn die Knoten nicht verbunden wären.
Die Feedforward-Vermehrung ist ein robustes Modell, das bei der Arbeit mit linearen Daten verwendet wird. Die Operationen können variieren, je nachdem, ob es sich um ein- oder mehrschichtige Modelle handelt. Einschichtige Modelle werden in der Regel für linear trennbare Daten verwendet. Für dieses Beispiel wird ein bestimmter Schwellenwert gewählt. Wenn der Wert der Aktivierungsfunktion größer als dieser Schwellenwert ist, wird er zu 1, wenn er kleiner ist, zu 0. Häufig verwendete Aktivierungsfunktionen sind ReLU, Sigmoid und tanh.
Dennoch hat sie einen Nachteil. Da die abgeleiteten Werte dieser Funktion außerhalb eines kleinen Bereichs sehr klein sind, entsteht das Problem des verschwindenden Gradienten. In mehrschichtigen Netzen werden verschiedene Lerntechniken verwendet. Die gebräuchlichste ist die Backpropagation. Die Back-Propagation wird in einem eigenen Abschnitt erläutert.

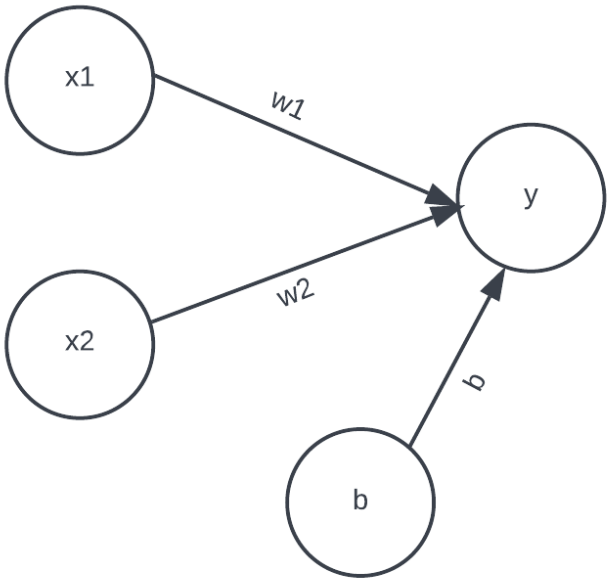

Abbildung 2.6: Einfaches Feed-Forward Netzwerk (source: K Uğur Sarp)

$$y = (x1 * w1 + x2 * w2 + b)$$

Eine mathematische Erklärung der Berechnungen bei der Arbeit mit Feedforward-Netzen. X-Werte sind unsere Eingabewerte, b ist die Vorspannung, und y ist die Ausgabe des Netzes. Die obige Gleichung wird berechnet, um den y-Wert zu ermitteln und eine Vorhersage zu treffen. Beim Training eines Feedforward-Netzes bleibt der Prozess unabhängig von der Anzahl der Neuronen, der Netzgröße usw. gleich.

2.3 Back Propagation

Der Back-Propagation-Algorithmus wird zum Trainieren von Feed-Forward-Netzen verwendet. Bei der Feed-Forward-Propagation wird das gesamte Netz nur in Vorwärtsrichtung berechnet. Es liefert eine Ausgabe, aber dieser Ausgabewert ist möglicherweise nicht optimal. Die Werte der Neuronen und die ihnen zugeordneten Gewichte können größer

oder kleiner erscheinen, als sie eigentlich sein sollten. In einer solchen Situation können wir nicht die gewünschte Genauigkeit erreichen. Um das gesamte Netzwerk stabiler zu machen, müssen wir die den Neuronen zugeordneten Gewichtungswerte in jeder Schicht aktualisieren. Hier kommt die Back-Propagation zum Einsatz. Bei der Feed-Forward-Methode wird nur vorwärts gerechnet, bis der Ausgang erreicht ist. Back-Propagation ordnet die Gewichte im Netzwerk neu an, indem es rückwärts geht. Die Arbeitslogik der Back-Propagation ist wie folgt:

- Der gesamte Satz wird mit Mini-Batches trainiert, bis die Ausgabe erreicht ist.
- Es wird zur nächsten Schicht weitergeschaltet, indem auf jeder Schicht Berechnungen durchgeführt werden. Die Backpropagation behält jedoch die Ergebnisse der einzelnen Schichten im Speicher, da sie diese Werte möglicherweise aktualisieren muss.
- Anschließend wird die Fehlerspanne ermittelt, indem die gewünschten und die tatsächlichen Ergebnisse verglichen werden.
- Sie berechnet die Fehlerspanne in jeder Schicht anhand der Kettenregel.
- Es misst die Fehler an den Knoten in jeder Schicht, bis es zur Eingabeschicht zurückkehrt.
- Mit Hilfe des Gradientenabstiegs werden diese Fehler korrigiert

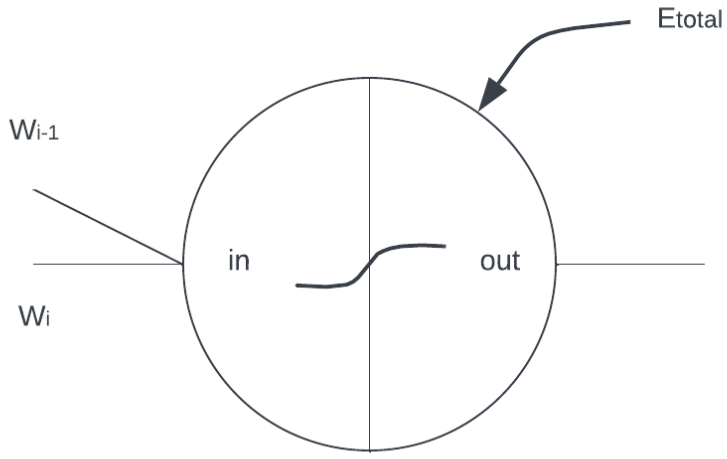

Abbildung 2.7: Basic Backpropagation-Struktur [6]

Nehmen wir an, wir haben die Gewichtung durch Feedforward-Propagation berechnet und müssen nun die Gewichte regularisieren. Wenn wir die Gewichtung für Wi aktualisieren wollen, müssen wir die Ableitungsformel wie folgt schreiben. (Der Einfachheit halber haben wir die Aktivierungsfunktion als Sigmoid und die Kostenfunktion als mittleren quadratischen Fehler gewählt).

$$\frac{\partial E_total}{\partial w_i} = \frac{\partial E_total}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial h_i} * \frac{\partial h_i}{(\partial w_i}$$

Um das Gewicht von wi zu aktualisieren, müssen wir die Kettenregel anwenden, bis ihre Ableitungen erreicht sind. Der Einfachheit halber aktualisieren wir das Gewicht kurz vor dem Ausgangswert, aber diese Kettenregel kann auf jedes Gewicht im Netz angewendet werden.

$$\sum_{i=1}^{n}(y - \hat{y})^2$$

Wie bereits erwähnt, haben wir für unsere Kostenfunktion den mittleren quadratischen Fehler gewählt.

$$\frac{\partial E_total}{\partial w_i} = 2 * (y - \hat{y})^2 * -1$$

Die Ableitung von Total nach ŷ wird berechnet. Wenn wir von der obigen Abbildung ausgehen, können wir uns vorstellen, dass wir vom Abschnitt Etotal zum Abschnitt Out gewechselt haben.

$$\hat{y} = \frac{1}{1 + e^x}$$

Formel für die Sigmoidfunktion

$$\frac{\partial \hat{y}}{\partial h_i} = \hat{y}(1 - \hat{y})$$

Ableitung von ŷ nach hi. Die gleiche Logik gilt auch hier. Von außen nach innen gehen.

$$h_i = x_i - 1 * w_i - 1 + x_i * w_i$$

Der hi-Wert (Ausgangswert) ist gleich dem eingehenden Gewicht, multipliziert mit den Werten des Neurons und dann summiert.

$$\frac{\partial h_i}{\partial w_i} = x_i$$

Von innen nach außen zum Gewicht i gehen.

$$\frac{\partial E_total}{\partial w_i} = (2 * (y - \hat{y}) * -1) * (\hat{y}(1 - \hat{y})) * (x_i)$$

Wir haben jede Ableitung berechnet, um das Gewicht von wi zu aktualisieren.

$$neww_i = w_i - r * \frac{\partial E_total}{\partial w_i}$$

Um schließlich die aktualisierte Gewichtung zu ermitteln, müssen wir die Lernrate, multipliziert mit dem Ergebnis, für das wir die Kettenregel angewendet haben, von der aktuellen Gewichtung abziehen.

2.4 Recurrent Neural Network

In diesem Abschnitt wurden die rekurrenten neuronalen Netze und ihre Funktionsweise kurz erläutert. Ein rekurrentes neuronales Netz ist die Deep Learning-Methode, die im Allgemeinen zur Vorhersage des nächsten Schritts verwendet wird. Im Gegensatz zu Feedforward-Netzwerken haben RNNs in jeder Schicht des Netzwerks die gleichen Parameter. Diese gemeinsamen Gewichte werden mit Hilfe von Backpropagation und Gradientenabstieg angepasst. Das wichtigste Merkmal, das es von anderen Methoden unterscheidet, ist, dass es sich den vorherigen Zustand merken kann. Da es sich die vorherige Eingabe merken kann, funktioniert es viel besser als andere Algorithmen bei NLP-Operationen. Ein weiterer Unterschied besteht darin, dass bei anderen neuronalen Netzen die einzelnen Eingaben unabhängig voneinander sind, während sie bei RNN miteinander verbunden sind. RNNs verwenden eine schleifenartige Struktur, die sich in sich selbst dreht, um ihre etablierten Beziehungen dauerhaft zu machen.
Da RNNs leistungsstarke Deep-Learning-Algorithmen sind, können sie in vielen Bereichen eingesetzt werden. Zu den Anwendungsbereichen von RNN gehören: traditionelle neuronale Netze, Musikproduktion, Gefühlsklassifizierung, Entitätserkennung und maschinelle Übersetzung, aber aufgrund des begrenzten Speichers und der Schwierigkeiten beim Training sind sie für komplexe Aufgaben nicht geeignet.
Obwohl RNNs leistungsstarke Algorithmen für NLP sind, ist es schwierig, sie zu trainieren. Das häufigste Problem, das bei der Arbeit mit RNNs auftritt, ist der verschwindende/explodierende Gradient. In Abschnitt 2.4.2 wird dieses Problem erörtert und wie es überwunden werden kann. Wir können dieses Beispiel als das einfachste Beispiel für ein RNN zeigen. Es nimmt den Wert von x(t) und den Wert von h(t-1) in jedem Zeitintervall t und erzeugt einen Wert von h(t).

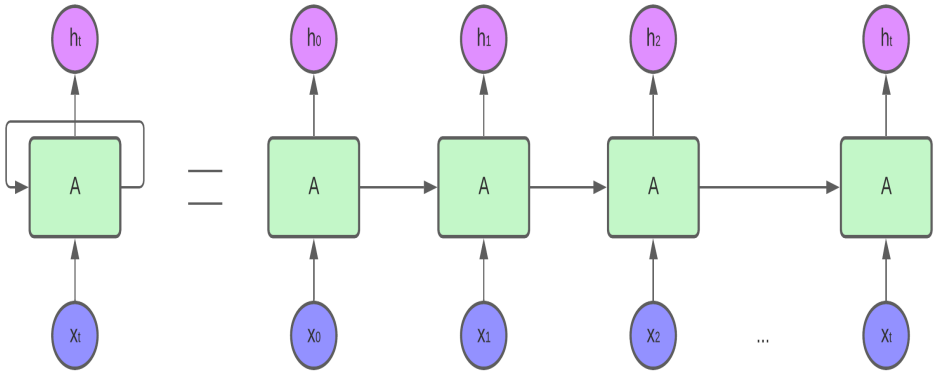

Abbildung 2.8: Recurrent Neural Network entrollt durch die Zeit [7]

Wie in der Abbildung zu sehen ist, hängt die folgende Eingabe vom vorherigen h(t)-Wert ab. Das Ergebnis des letzten Wertes wird so angeordnet, dass es für die Eingabe des nächsten Wertes verwendet werden kann.

$$h_t = \sum_{t=0}^{T} X_t W_t + U h_[t-1] + b$$

Wenn dieses Netz in Betrieb genommen wird, werden die Gewichte des ersten Wertes mit 0 initialisiert, da es keine Ausgangswerte für den vorherigen Zustand gibt. Um den Wert h(t) zum Zeitpunkt t zu berechnen, müssen wir die oben angegebene Formel befolgen. Da rekurrente Netze von der vorherigen Ausgabe abhängen, müssen h(t-1) und sein Gewicht ebenfalls in das Netz eingespeist werden.

- Aktueller Input (x(t)) und Gewicht dieses Inputs (W(t)) multipliziert
- Vorheriger Ausgangswert (h(t-1)) und sein Gewicht (U) multipliziert
- Sie alle addieren sich mit einer Tendenz von b.

Dieser einfache Prozess läuft so lange, bis der Ausgang erreicht ist.

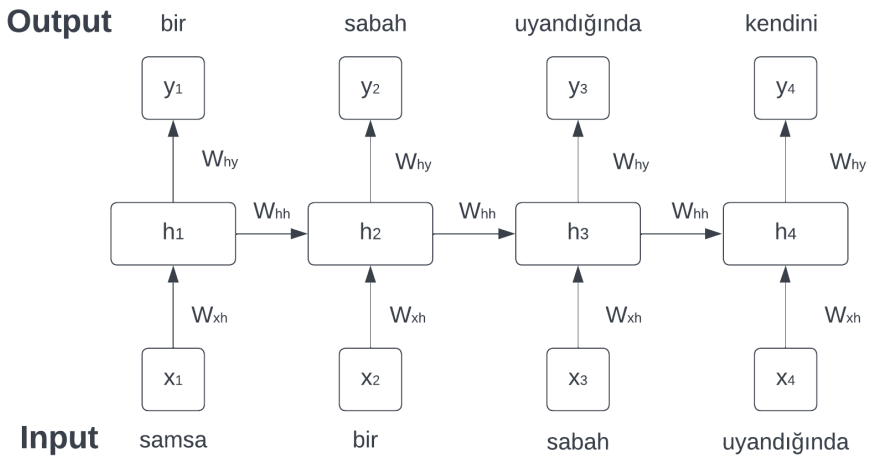

Abbildung 2.9: Einfaches RNN-Beispiel für Sequenzdaten. [8]

2.4.1 Memory Cells

Neuronale Netze eignen sich hervorragend für den Umgang mit implizitem Wissen, aber sie tun sich schwer mit Fakten. Daher können sie sich Informationen nicht so merken, wie wir es uns wünschen. Der stochastische Gradientenabstieg kann diese Informationen speichern, aber auch dieser Ansatz hat seine Schattenseiten: Rechenleistung, geringer Erfolg usw. Es fehlte ein System, das Informationen wie ein Mensch speichern und verarbeiten konnte. Um dieses Problem zu überwinden, wurde eine neue Methode namens Speicherzellen vorgeschlagen, ein System, das Informationen korrekt speichern und verarbeiten kann. Später wurden diese Speicherzellen auch in LSTM- und GRU-Strukturen verwendet.
Da die vorherige Ausgabe mit der Eingabelogik im aktuellen Schritt zusammenarbeitet, kann man sagen, dass es in den RNN-Einheiten einen gewissen Speichermechanismus gibt. Die Speichereinheiten in RNNs sind begrenzt. RNNs fallen exponentiell ab, wenn wir mit langen sequentiellen Daten arbeiten. Beispiel: Ausgabe h(t) zum Zeitpunkt t, die Eingabewerte sind x(t) und h(t-1). h(t-1) ist die Ausgabe des vorherigen Zustands. Man kann sehen, dass es hier einen Speichermechanismus gibt.

2.4.2 Vanishing/Exploding Gradient

Die Arbeitslogik des Backpropagation-Algorithmus bewegte sich wie erläutert vom Ausgang zum Eingang und regulierte so die Gewichte. Nachdem der Algorithmus den Gradienten der Kostenfunktion berechnet hat, aktualisiert er die Parameter anhand dieses Gradienten. Während der Algorithmus die Schichten durchläuft, wird dieser Gradientenwert immer kleiner. Er wird so klein, dass er gegen Null geht. In einem solchen Fall wird der Wert zu klein, um sein Gewicht zu ändern. Dieser Zustand wird als "verschwindender Gradient"bezeichnet. Wenn die gleiche Situation auftritt, d. h. wenn der Gradientenwert im Verlauf des Netzes schnell ansteigt und die Auswirkungen auf die Gewichte zu stark zunehmen, spricht man von einem explodierenden Gradienten. In beiden Fällen hat dies unerwünschte Folgen. Wenn er zu groß wird, führt er dazu, dass die Gewichte überlaufen, weil er zu viel macht; wenn er zu klein ist, verliert er mit der Zeit an Bedeutung.
Bis in die 2000er Jahre wurden Deep-Learning-Algorithmen wegen des verschwindenden/explodierenden Gradienten nicht verwendet. Während der Backpropagation, die die Gewichte über die Schichten reguliert, wurden diese Gewichte deutlich zu klein oder zu groß. Nachdem die Ursache für dieses Problem gefunden war, wurden mehrere Lösungen vorgeschlagen. LSTM kann das Problem des verschwindenden Gradienten lösen und wird daher als optimale Lösung für dieses Problem angesehen.
Das Problem kann auf verschiedene Weise gelöst werden.

- Richtige Gewichtsinitialisierung
- Verwendung nicht sättigender Aktivierungsfunktionen
- Batch-Normalisierung
- LSTMs
- Gradient Clipping

2.5 Long Short-Term Memory

Wie im Abschnitt über RNN erwähnt, leiden diese Einheiten unter dem verschwindenden und explodierenden Gradienten. Der Hauptgrund für dieses Problem war die sukzessive Multiplikation. Dieses Problem tritt bei der Arbeit mit dem Kurzzeitgedächtnis nicht auf, aber die Arbeit mit dem Langzeitgedächtnis war etwas problematisch. Um dieses rekurrente Problem zu lösen, wird die Gleichung für den versteckten Vektor im LSTM geändert. LSTM ist erfolgreich bei der Arbeit mit langen Datensequenzen wie maschineller Übersetzung, Texterzeugung, Bildbeschriftung, usw. LSTM verwendet eine RNN-Architektur. Die Einheiten, die sich erinnern können, werden im LSTM als Zellen bezeichnet. Diese Zelle verwendet eine Kombination aus der vorherigen Ausgabe und der aktuellen Eingabe. Diese Zelle entscheidet, was im Speicher bleibt und was gelöscht wird. Ein Beispiel: Sie sehen sich online eine Restaurantkritik an. Wenn wir einen Kommentar lesen, in dem es heißt: Ïch war sehr zufrieden mit dem Service, das Essen war wirklich köstlich, ich würde gerne wieder hingehen, wenn ich die Gelegenheit habe", achtet unser Gehirn automatisch auf die entscheidenden Wörter und konzentriert sich auf diese (ßufrieden, köstlich") und ignoriert den Rest. Das LSTM arbeitet so, dass es die notwendigen Eingaben im Speicher behält und den Rest ignoriert. LSTMs helfen bei der Lösung von Problemen wie verschwindenden/explodierenden Gradienten, da sie sich den vorherigen und den aktuellen Zustand merken können.

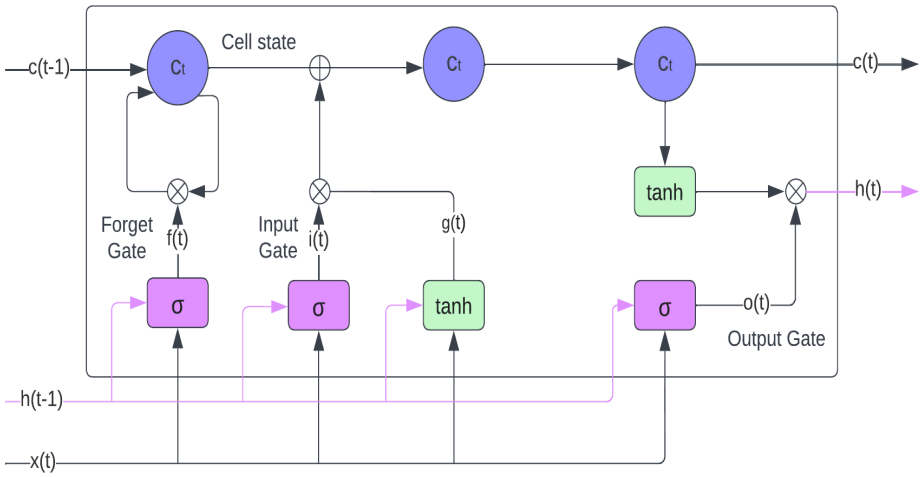

Abbildung 2.10: LSTM-Zellarchitektur [9]

Der wichtigste Punkt beim LSTM ist die Frage, was im Langzeitgedächtnis gespeichert und was gelöscht werden soll. Das Bild zeigt, dass der vorherige Langzeitzustand c(t-1) in den folgenden Zustand übergeht. Zunächst durchläuft es das Vergessenstor und lässt hier einige unerwünschte Informationen fallen. Dann fügt es neue Erinnerungen hinzu. c(t) wird unverändert in die andere Einheit geschickt. Zu jedem Zeitpunkt t wird ein Teil der Informationen vergessen und ein anderer Teil hinzugefügt. Nachdem die Information im Langzeitgedächtnis entschieden wurde, durchläuft sie die tanh-Funktion und wird mit der Ausgabe verglichen, wodurch die Ausgabe des Kurzzeitgedächtnisses entsteht.

- g(t) ist die Hauptschicht. Sie analysiert die aktuelle Eingabe x(t) und die vorherige kurzfristige Ausgabe h(t-1). Die wesentlichen Informationen werden im Langzeitgedächtnis gespeichert, der Rest wird verworfen.
- Vergiss, Input- und Output-Gates sind Gate-Controller. Sie verwenden die Sigmoid-Funktion als Aktivierungsfunktion, die sie Ausgangswerte zwischen 0 und 1 annehmen können. Wenn der Wert nahe bei 0 liegt, wird das Tor geschlossen, wenn er nahe bei 1 liegt, wird es geöffnet.
  - Wenn vergessene Gate-Werte nahe 1 sind, werden diese Werte aus dem Langzeitgedächtnis entfernt.
  - Wenn die Werte des Eingangsgatters nahe bei 1 liegen, werden diese Werte dem Langzeitspeicher hinzugefügt.
  - Wenn die Werte des Ausgangsgatters nahe bei 1 liegen, bestimmt es, welche Teile als Ausgang fungieren.
  - Das Gegenteil ist der Fall, wenn die Werte nahe bei 0 liegen.

Der Arbeitsmechanismus des LSTM lässt ihn entscheiden, welche Informationen langfristig gespeichert werden und wie lange diese Informationen benötigt werden. Aus diesem Grund funktioniert das LSTM besser mit langen Datenfolgen.

$$i_t = \sigma(W_x i^T x_(t) + W_h i^T h_(t-1) + b_i)$$

$$f_t = \sigma(W_x f^T x_(t) + W_h f^T h_(t-1) + b_f)$$

$$o_t = \sigma(W_x o^T x_(t) + W_h o^T h_(t-1) + b_o)$$

$$g_t = tanh(W_x g^T X_(t) + W_h g^T h_(t-1) + b_g$$

$$c_t = f_t \bigotimes c_(t-1) + i_(t) \bigotimes g_(t)$$

$$y_t = h_(t) = o_(t) \bigotimes tanh(c_(t))$$

- Wxi, Wxf, Wxo und Wxg sind die Gewichtsmatrizen, die mit dem Eingang x(t) verbunden sind
- Whi, Whf, Who und Whg sind die Gewichtungsmatrizen, die mit dem vorherigen Kurzzeitzustand h(t-1) verbunden sind
- b-Werte sind die Verzerrungsterme.

3 Stand der Technik

In diesem Abschnitt werden die Studien untersucht, die im Abstand von fünf Jahren durchgeführt wurden. Der Hauptgrund für die Wahl eines solchen Zeitraums ist, dass die Verarbeitung natürlicher Sprache ein neues Thema ist und sich die Algorithmen täglich weiterentwickeln und verändern. Punkte, die bei der Untersuchung berücksichtigt werden sollten: Der Datensatz, welche Algorithmen verwendet wurden, die Unterschiede zu anderen Studien und die erreichte Genauigkeit.
Die Artikel werden im Bereich der Texterstellung und der Verarbeitung natürlicher Sprache analysiert, wobei die Vorhersage des nächsten Wortes als Leitfaden dient. Die Grundlagen von Algorithmen und verschiedenen Ansätzen werden in diesen Artikeln untersucht.
Es ist immer noch ein Problem, dass maschinell erstellte Texte nicht so erfolgreich sind wie von Menschen erstellte Texte. Wenn dieses Problem gelöst ist, werden Bereiche wie Textzusammenfassung, maschinelle Übersetzung, Texterzeugung usw. ebenso erfolgreiche Ergebnisse liefern wie Menschen. 2019 schlugen Forscher[1] der Universität Peking ein neues LSTM vor, das bei der Lösung der oben genannten Probleme besser abschneidet. Bei diesem Modell werden die vorherigen und nachfolgenden Wörter im Speicher abgelegt und anschließend verwendet, um den Kontext des gegebenen Satzes zu verstehen. Als Datensatz verwenden sie die Fragen, die in der Hochschulprüfung in China gestellt werden. Dieser Datensatz enthält 120.000 Wörter und 5.000 Buchstaben. Die vorgeschlagene Methode erreichte bei den menschlichen Bewertungen eine Punktzahl von 4/5.
Jingyun Yang[2] und seine Kollegen schlugen das MCNN-ReMGU-Modell vor. Die Grundlagen des Modells sind die Mehrfensterfaltung und minimal gated unit (MGU). Zunächst extrahieren diese Faltungen die Informationen innerhalb der Wortfolgen. Anschließend wird diese Information mit MGU als Residuum verbunden. Auf diese Weise werden die Probleme des verschwindenden und explodierenden Gradienten gelöst, und die Wahrscheinlichkeit der Vorhersage des nächsten Wortes erhöht sich erheblich. Gleichzeitig wird die Merkmalsextraktion mit CNN verknüpft; dank dieser Verknüpfung werden diese Wortfolgen aussagekräftiger. WikiText-2 wurde als Datensatz verwendet. Zur Bewertung wird die Perplexität herangezogen. Je niedriger dieser Wert ist, desto besser arbeitet der Algorithmus. Es ist das erfolgreichste Modell unter den

verglichenen Algorithmen mit einem Perplexitätswert von 69,3, etwa 30 Punkte niedriger als LSTM.
Die Vorhersage des nächsten Wortes reduziert Tippfehler und beschleunigt den Schreibprozess. Forscher[3] der Nationalen Polytechnischen Universität Lviv fanden keine derartigen Werkzeuge, die dasselbe für die ukrainische Sprache leisten. Sie verwendeten LSTM und Markov-Ketten und kombinierten sie in ihren drei verschiedenen Modellen. Als Datensatz wählten sie Gedichte (da diese einem bestimmten Muster folgen, wie Menschen, die über ihr Telefon simsen). LSTM erreicht eine Genauigkeit von 75% bei der Vorhersage des nächsten Wortes. Bei den von den drei Modellen erstellten Texten wurde festgestellt, dass das Hybridmodell bei den menschlichen Bewertungen das erfolgreichste Ergebnis erzielte.
Andrew Hard[4] und seine Kollegen verwendeten ein föderiertes Lernsystem zur Vorhersage des nächsten Wortes auf Smartphones. Im Gegensatz zu den anderen derzeit verwendeten Methoden ist der Algorithmus des föderierten Lernens in Bezug auf die Sicherheit erfolgreich, da er keine Benutzerinformationen an die Server-Seite überträgt. Als Modellarchitektur wurde Coupled Input and Forget Gate (CIFG), eine spezielle Art von LSTM, verwendet. Der Vorteil von CIFG ist, dass es weniger Rechenleistung benötigt. Als Datensatz werden Wörter und Sätze verwendet, die zuvor vom Telefonbenutzer geschrieben wurden. In den gesammelten Daten, die auf der Nutzung durch den Benutzer basieren, beträgt die Klickrate für drei dem Benutzer vorgegebene Wörter 27%, womit der föderierte CIFG dem Server-CIFG, dem anderen verglichenen Algorithmus, überlegen ist.

4 Methodik

4.1 Datensatz und Vorverarbeitung

Da mein Ziel darin besteht, das nächste Wort auf Türkisch erfolgreich vorherzusagen, habe ich sowohl ein englisches als auch ein türkisches Buch ausgewählt, um die Ergebnisse der beiden zu vergleichen. Als türkischen Datensatz habe ich Dudaktan Kalbe gewählt. Für den englischen Datensatz wurde Sherlock Holmes ausgewählt. Der Grund für die Wahl dieser beiden Bücher ist, dass sie in Umfang und Länge ähnlich sind. Alle Experimente werden in diesen beiden Büchern durchgeführt. Der Datensatz wurde in Wortsequenzen von dreißig Längen unterteilt. Ich werde das Ergebnis mit zwei verschiedenen Ansätzen zur Vorhersage des nächsten Wortes testen. Dieselben Schritte wurden sowohl für die Zeichenweise[11] als auch für die Wort-für-Wort[12] Vorhersage angewendet. Das Vorverarbeitungsverfahren ist wie folgt.

- Alle Interpunktionszeichen werden aus dem Text entfernt.
- Buchstaben mit Zirkumflex werden in ihre Entsprechung umgewandelt.
  â → a, ê → e, î → i
- Alle Buchstaben wurden in Kleinbuchstaben umgewandelt.
- Zahlen wurden in ihre Wortentsprechungen umgewandelt.
  18 Ağustos → On sekiz Ağustos
- Es wurden Sequenzen von 30 Wörtern Länge erstellt.
- Jedes Wort wird in eine eindeutige Zahl umgewandelt.
- Dreißig Wörter lange Sequenzen werden mit diesen eindeutigen Nummern dargestellt.
  [..., "kendisini", bulduğu, "bu", "durumda "] → [..., 1265, 32, 853, 11]

Mit der Datenbereinigung und -vorverarbeitung wollen wir die Datenqualität verbessern. Aus diesem Grund ist die Vorverarbeitung von entscheidender Bedeutung, und wir müssen diese Schritte durchführen, bevor wir versuchen, unser Modell zu trainieren.

4.2 Training

Wie bereits erwähnt, habe ich zwei verschiedene Ansätze gewählt, um das nächste Wort vorherzusagen. Der erste Ansatz ist Wort für Wort. Der türkische Datensatz besteht aus:

- Gesamtzahl der Wörter: 65187
- Einzigartige Wörter: 15965
- Anzahl der Sequenzen mit 30 Wörter: 65157

Der englische Datensatz besteht aus:

- Gesamtzahl der Wörter: 98072
- Einzigartige Wörter: 7841
- Anzahl der Sequenzen mit 30 Wörter: 98042

Beim Erstellen von 30 Wörter langen Satzphrasen wurden neue Sätze gebildet, indem ein Wort bis zum Ende des Datensatzes verschoben wurde.
Der zweite Ansatz ist die Vorhersage des nächsten Wortes Zeichen für Zeichen. Außerdem wurde er nur auf dem türkischen Datensatz trainiert. Dieser Ansatz wurde nur getestet, um zu sehen, welcher Optimierer(Adam, RMSprop) bei der Vorhersage des folgenden char-basierten Wortes besser funktioniert.

- 480120 Zeichen lang
- Neununddreißig einzigartige Zeichen, einschließlich Interpunktionen.

Jeder Datensatz wird in 90% zum Trainieren, 5% zum Testen und 5% zum Validieren aufgeteilt.
Obwohl es zahlreiche Studien zur Sprachmodellierung mit LSTMs gibt, haben die Forscher die idealen Hyperparameterwerte noch nicht identifiziert. Mein Ziel war es also, die besten Hyperparameter für mein Problem zu finden. Die verwendeten Hyperparameter:

- Optimierer: [Adam, RMSprop]
- Batchgröße : 64
- Epoche : 20

- LSTM-Speicherzellen: 128
- LSTM-Zahl: [1, 2]
- Dropout-Rate : 0.2

Das Arbeitsprinzip ist bei beiden Algorithmen das gleiche. In das Netz werden Sequenzen der Länge von dreißig Wörtern eingespeist. Wie bereits erwähnt, handelt es sich bei dieser Sequenz um Zahlenwerte mit eindeutigen Zahlen für jedes Wort. In der Einbettungsschicht wird jedes Wort mit numerischen Vektoren dargestellt. Diese Werte gehen dann an die LSTM-Schicht. Nach diesem Schritt werden einige Werte mit einer Wahrscheinlichkeit von 20% verworfen. Die erste dichte Schicht reguliert die anstehenden Werte mit der ReLU-Funktion. Die Ausgabeschicht ist gleich der Anzahl der einzelnen Wörter. Jedes Wort erhält eine Wahrscheinlichkeit aus der Softmax-Aktivierung. Das Wort mit der höchsten Wahrscheinlichkeit wird als nächstes Wort ausgewählt.

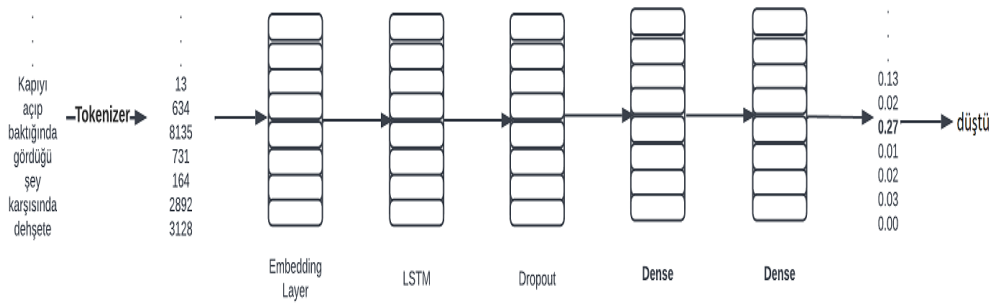

Abbildung 4.1: Arbeitsmechanismus des Word Modells (source: K Uğur Sarp)

Vorgefertigte Einbettungsschichten[13] können verwendet werden, um bessere Ergebnisse zu erzielen. Vorgefertigte Schichten werden mit großen Datensätzen trainiert, so dass optimale Gewichte für die Wörter in diesem Satz erreicht werden. Wenn wir mit dieser Ebene arbeiten, müssen wir diese Ebene nicht trainieren. So können wir mit weniger Arbeit eine höhere Genauigkeit erreichen. Wenn das kommende Wort in dieser Schicht vorhanden ist, werden die Gewichte auf die vorher trainierten Gewichte gesetzt, und wenn es nicht gefunden wird, wird der Vektor auf Null gesetzt.
Wenn zum Beispiel das kommende Wort ïzahïst und diese vortrainierte Schicht das

Wort enthält, werden die Gewichte automatisch gesetzt, ohne dass ein Training erforderlich ist, um eine optimale Genauigkeit zu erreichen
izah : [... 0.297573, 0.736949, 0.426299, -0.502678, 0.309295, -0.215791, 0.174483, ...]
Wenn das kommende Wort nicht in diesem Vokabular enthalten ist, werden seine Gewichte auf 0 gesetzt
jack : [... 0, 0, 0, 0, 0, 0, ...]
Ein frühes Anhalten kann eine Überanpassung verhindern, aber mein Ziel ist es, die höchste Genauigkeit in 20 Epochen zu erreichen, daher werde ich keinen Anhaltemechanismus verwenden. Nach 20 Epochen wird das Ergebnis mit der höchsten Wahrscheinlichkeit gewählt

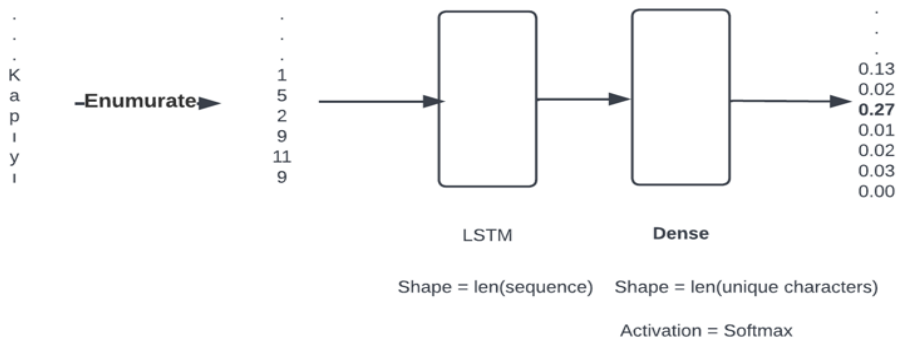

Abbildung 4.2: Arbeitsmechanismus des Zeichenweise Modells (source: K Uğur Sarp)

Beim zeichenweisen Modell werden die gleichen Schritte wie in der Vorverarbeitungsphase wortweise abgearbeitet. Der Arbeitsmechanismus besteht hier darin, anstelle des nächsten Zeichens die Buchstabenfolge zu erraten. Während die Länge unserer Sequenzen in unserem Wort-für-Wort-Modell 30 beträgt, verwenden wir hier Sequenzen mit einer Länge von 40. Beim Erstellen jeder neuen Sequenz werden neue Sequenzen erstellt, indem 3 Teile nach rechts geschoben werden. Als Stapelgröße wird 128 gewählt. Die aufgezählte Buchstabenfolge wird der LSTM-Schicht zugeführt, und das Ergebnis wird dann von der Dense-Schicht in einen Wahrscheinlichkeitsvektor umgewandelt, der alle Buchstaben enthält. Hier wird das nächste Zeichen mit der höchsten Wahrscheinlichkeit ausgewählt und der Ratevorgang abgeschlossen.

4.3 Modellarchitektur

LSTM ist ein leistungsstarker Algorithmus für die Arbeit mit sequentiellen Daten. Im Gegensatz zu RNN leidet LSTM nicht unter Problemen wie verschwindenden oder explodierenden Gradienten. Mit seinem Langzeitgedächtnis kann sich das LSTM wichtige Muster merken, die für die Erstellung eines erfolgreichen Textes unerlässlich sind. Die Architektur besteht aus drei Hauptschichten:

- Einbettungsschicht: Verstehen der Wörter und der Beziehungen zwischen diesen Wörtern.
- LSTM-Schicht: Verstehen der Beziehung zwischen sequentiellen Daten.
- Dichte Schicht: Ausgabe der Wortwahrscheinlichkeiten.

5 Experiment

In diesem Abschnitt werden die von uns durchgeführten Experimente und ihre Ergebnisse erläutert. Wie bereits erwähnt, werden die beiden Algorithmen, die Zeichen für Zeichen und Wort für Wort erzeugen, und die Ergebnisse der verschiedenen Parameter, die in diesen beiden Algorithmen ausprobiert wurden, verglichen. Wir haben die hier verwendeten Validierungsdaten nicht verwendet, um die Hyperparameter des Modells in irgendeiner Weise abzustimmen. Am Ende von 20 Epochen haben wir die Ergebnisse mit höchster Genauigkeit erhalten und miteinander verglichen. Aus diesem Grund haben wir nur die Validierungsgenauigkeit verwendet und uns nicht mit einer zusätzlichen Testgenauigkeit befasst. Alle anzugebenden Beispiele durchlaufen diese Validierungsgenauigkeit.

5.1 Experimental Setup

Das Experiment wurde auf einem Windows 10 64-Bit-Betriebssystem mit Intel(R) Core(TM) i7-6700 HQ CPU @2,60 GHz durchgeführt. Das System verfügt über 16 GB RAM.

5.2 Leistungsanalyse

Wenn wir die Ergebnisse mit den im Abschnitt Training angegebenen Parametern und Algorithmen vergleichen, kommen wir zu dem Schluss, dass der Algorithmus, der das nächste Wort wortweise vorhersagt, nicht erfolgreich genug ist. Wenn wir die Optimierer RMSprop und Adam vergleichen, sehen wir, dass Adam in beiden LSTM-Schichten erfolgreicher ist, mit einem Unterschied von etwa 1%. Selbst bei einem Wert von nur 6,82% erzielt Adam die besten Ergebnisse mit einer einzigen LSTM-Schicht. Der Adam-Optimierer erzielt das beste Ergebnis mit einer einzelnen LSTM-Schicht im englischen Datensatz. Der Algorithmus erreicht eine Genauigkeit(validation accuracy) von 13,25% im englischen Datensatz. Das ist fast doppelt so erfolgreich wie die Genauigkeit des türkischen Datensatzes.

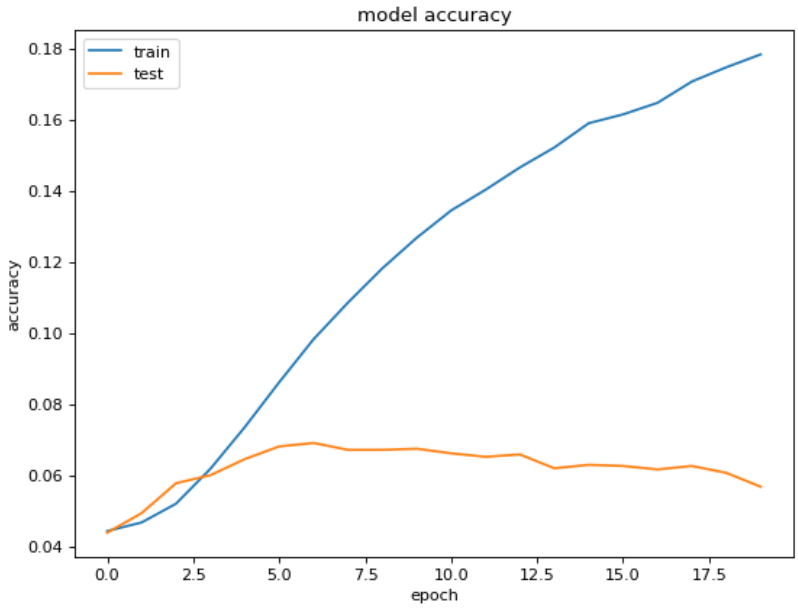

Abbildung 5.1: Eine LSTM-Schicht, Adam-Optimierer-Genauigkeitsergebnisse für den türkischen Datensatz.

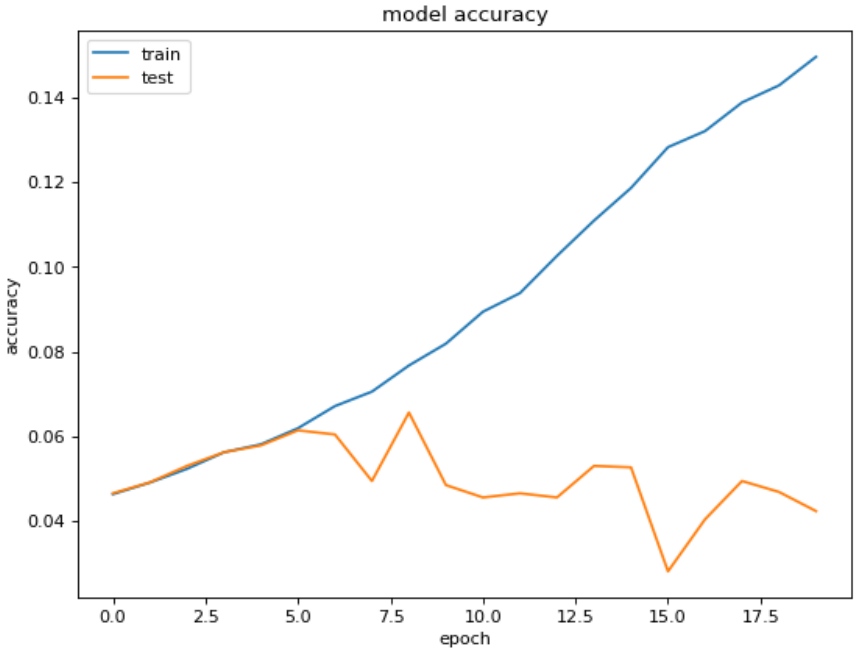

Abbildung 5.2: Eine LSTM-Schicht, RMSprop-Optimierungsergebnisse für den türkischen Datensatz.

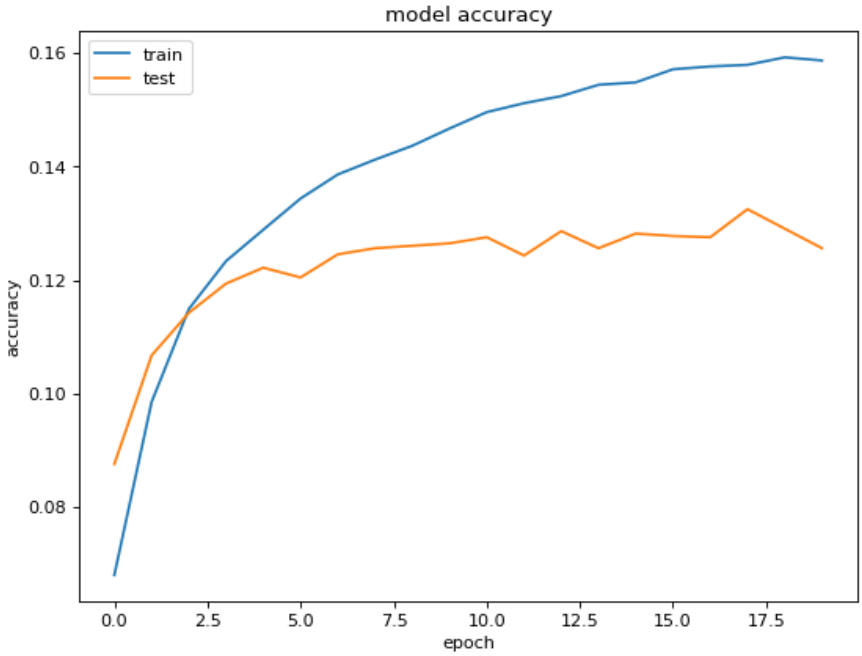

Abbildung 5.3: Zwei LSTM-Schichten, Adam-Optimierer-Genauigkeitsergebnisse für den englischen Datensatz.

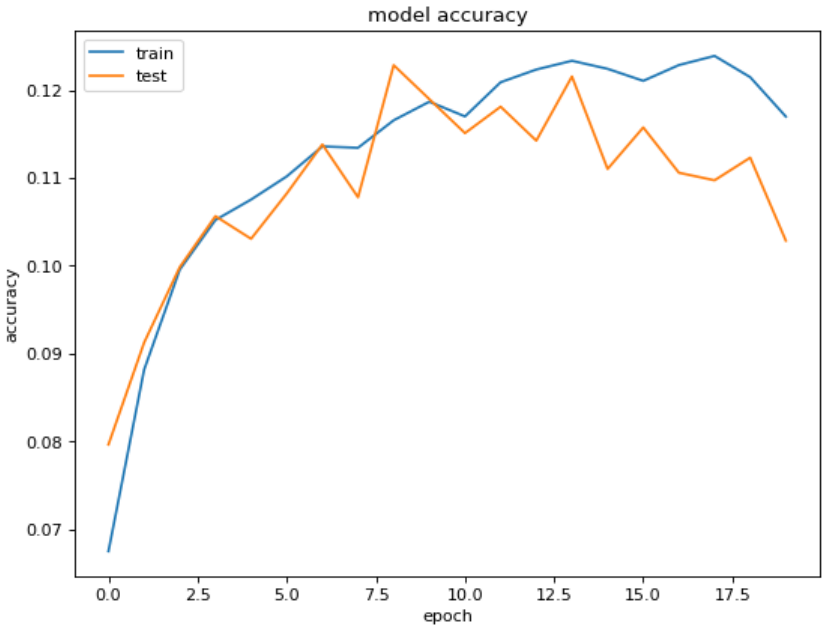

Abbildung 5.4: Zwei LSTM-Schichten, RMSprop-Optimierungsergebnisse für den englischen Datensatz.

LSTM-Layer-Anzahl	Adam (türkischer Datensatz)	RMSprop (türkischer Datensatz)	Adam (englischer Datensatz)	RMSprop (englischer Datensatz)
1	6.82%	6.21%	13.09%	12.02%
2	6.33%	6.13%	13.25%	12.28%

Tabelle 5.1: Die Auswirkung der Anzahl der LSTM-Schichten auf die Genauigkeit

Wie in den Bildern für den türkischen Datensatz zu sehen ist, beginnt die Genauigkeit nach etwa 7-10 Epochen zu sinken, da der Algorithmus die Muster nicht lernt, sondern sich die Daten einprägt.
Um zu testen, ob die vortrainierte Einbettungsschicht die Genauigkeit für den türkischen Test verbessert, arbeiten wir mit den erfolgreichsten Parametern, d.h. einer Schicht Adam-Optimierer. Wie erwartet, erreicht er mit eine Genauigkeit von 8,17%.

	Vortrainierte Einbettungsschicht	Normalerweise trainierte Einbettungsschicht
Accuracy	8.17%	6.82%

Tabelle 5.2: Vortrainierte vs. normale Einbettungsschicht

Andrew Hard und seine Teamkollegen vergleichen ein Modell[4], das Benutzer mit ihren Daten auf ihren lokalen Telefonen trainieren, anstatt dass Daten auf dem Server trainiert werden. Der Zweck dieses Vergleichs ist sowohl die Sicherheit als auch der Komfort in der Zugphase. Sie verwenden Coupled Input and Forget Gate (CIFG), eine Art von lstm. Es enthält 25% weniger Parameter im Vergleich zu einem normalen LSTM, daher ist es ein schnellerer Zugprozess. Etwa 600 Millionen Sätze werden trainiert. Die Zugphase dauert 4-5 Tage. Dieses vorgeschlagene Verfahren kann das richtige Wort mit einer Wahrscheinlichkeit von 16% vorhersagen. Die Wahrscheinlichkeit, dass eines der drei erratenen Wörter richtig ist, liegt bei 27%.

Story Teller (türkischer Datensatz)	Top-1-Rückruf Andrew Hard	Top-3-Rückruf Andrew Hard
8.17%	16.4%	27%

Tabelle 5.3: Story Teller vs Mobile Keyboard Prediction

Wenn wir die Ergebnisse des Algorithmus zur Vorhersage von Zeichen für Zeichen vergleichen, erreichen wir mit RMSprop eine höhere Genauigkeit. Während wir mit Adam 48,73% Erfolg haben, erreichen wir mit RMSprop eine Genauigkeit von 51,23%. Vergleicht man jedoch die Verlustraten der Modelle, so zeigt sich, dass der Adam-Algorithmus erfolgreicher arbeitet, während RMSprop nach der elften Epoche beginnt, den Datensatz zu überarbeiten, anstatt ihn zu lernen.

	Adam	RMSprop
Accuracy	48.73%	51.23%

Tabelle 5.4: Vorhersage von Zeichenweise

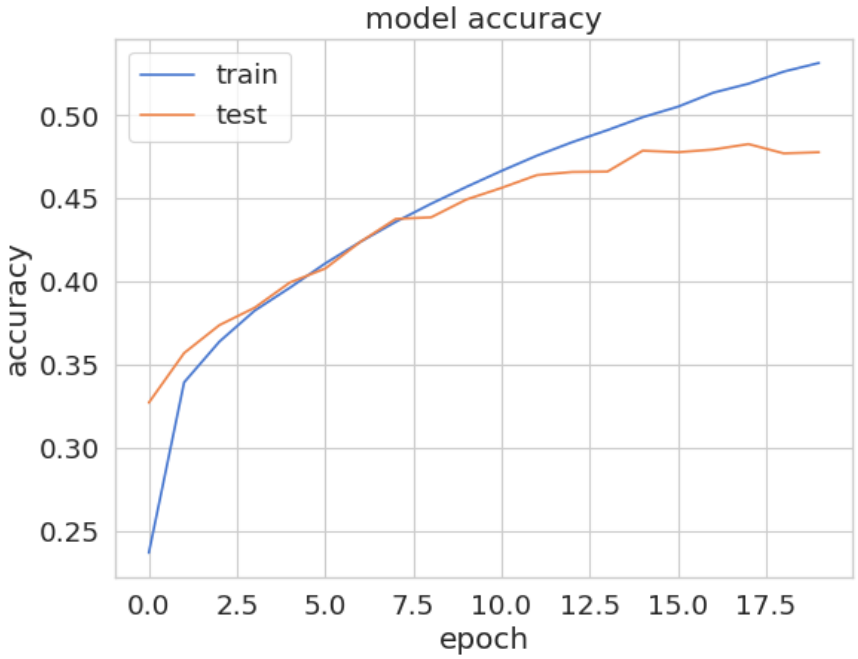

Abbildung 5.5: Zeichenweise Adam Optimierer Genauigkeit für türkischen Datensatz.

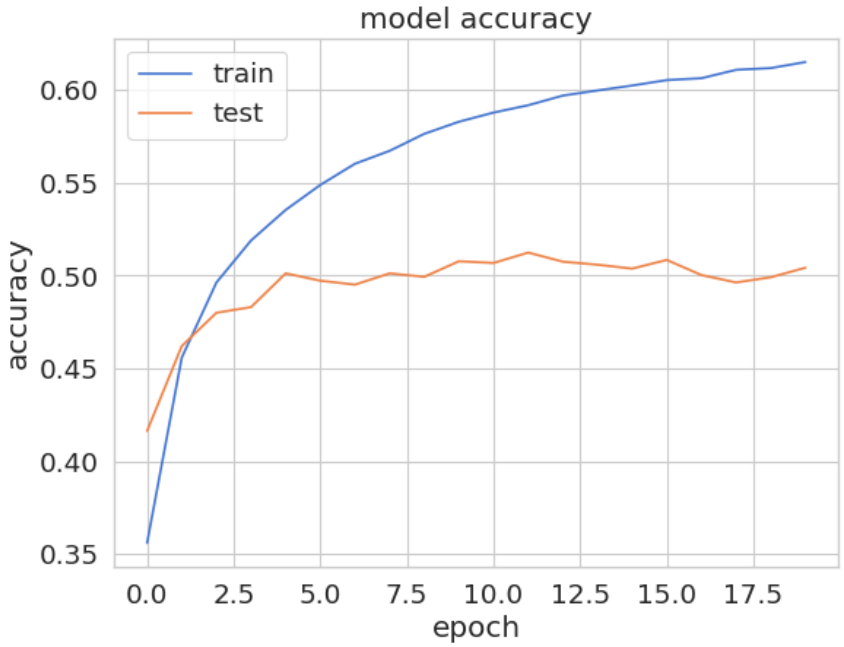

Abbildung 5.6: Zeichenweise RMSprop Optimierer Genauigkeit für türkischen Datensatz.

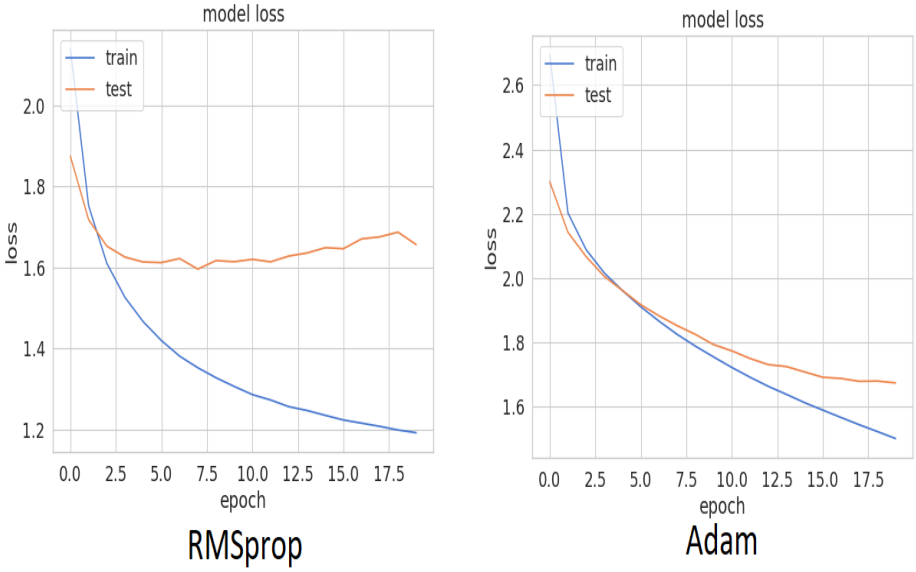

Abbildung 5.7: Verluste der beiden Modelle.

In diesem Artikel[10], der auf der IEEE-Konferenz 2019 über die Vorhersage des nächsten Wortes Zeichenweise veröffentlicht wurde. Bei Zeichenweise Strukturen wird eine baumförmige Struktur verwendet. In dieser Struktur besteht das Dokument aus verschiedenen Abschnitten wie Überschriften und Absätzen. Auf diese Weise ist es möglich, durch diese Blätter Vorhersagen über den Text zu treffen. Die verwendete Struktur ist LSTM. Arbeitet an 20 der beliebtesten Bücher im Projekt Gutenberg. Als Ergebnis der Arbeit an Nietzsches Buch werden 46% Erfolg erzielt.

	Story Teller	Aejaz Farooq Ganai
Accuracy	51.23%	46%

Tabelle 5.5: Story teller Zeichenweise accuracy

6 Fazit

Ziel dieser Arbeit ist die erfolgreiche Vorhersage des nächsten Wortes in der türkischen Sprache mit Hilfe des LSTM-Modells. Obwohl es möglich ist, das nächste Wort mit Methoden wie Zeichen für Zeichen, Wort für Wort und Satz für Satz vorherzusagen, besteht das Hauptziel darin, Vorhersagen Wort für Wort und Zeichen für Zeichen zu machen. Bei der Verwendung eines oder mehrerer geschichteter LSTM-Modelle wurde auch die Frage beantwortet, welche Art von Ansatz ein erfolgreicheres Ergebnis erzielen kann. Die Schritte, die in der Vorverarbeitungsphase durchgeführt werden, sind zwingend erforderlich, damit unser Modell erfolgreicher arbeiten kann.
Es kommen zwei Algorithmen zur Vorhersage des nächsten Wortes zum Einsatz: Zeichen für Zeichen und Wort für Wort. Die grundlegende Arbeitslogik dieser Algorithmen ist bei beiden gleich. Der einzige Unterschied besteht darin, dass das nächste Wort oder die nächste Buchstabenfolge vorhergesagt wird. Dieser Algorithmus wurde sowohl für Englisch als auch für Türkisch untersucht, um den Erfolg des Wort-für-Wort-Algorithmus für die türkische Sprache zu ermitteln. Der andere Algorithmus wird mit dem von Aejaz Farooq Ganai und seinen Kollegen auf der IEEE-Konferenz 2019 veröffentlichten Papier verglichen. Beim Vergleich der beiden Methoden war der in dieser Arbeit verwendete Algorithmus mit einer Rate von etwa 5% erfolgreich. In der von Andrew Hard und seinen Kollegen veröffentlichten Arbeit erreicht die Rate der Einbeziehung des richtigen Wortes in die drei vorhergesagten Wörter 27%, während die Erfolgsrate der Algorithmen bei etwa 13% liegt, wenn die Anzahl der vorhergesagten Wörter eins beträgt. In Anbetracht der Größe des in dieser Studie verwendeten Datensatzes kann die von unserem Algorithmus erreichte Konsistenzrate von 8,17% als ausreichend für diese Studie angesehen werden.
Beim Vergleich der beiden Methoden zeigt sich, dass der Algorithmus, der eine wortweise Vorhersage macht, erfolgloser ist als derjenige, der zeichenweise arbeitet. Da Türkisch von der Sprachstruktur her eine agglutinierende Sprache ist und sich von anderen Weltsprachen in Bezug auf die grammatische Struktur unterscheidet, ist der Erfolg der Vorhersagen gering. Die Gründe für den geringeren Erfolg im Vergleich zum Englischen werden im Folgenden erläutert.
Im Gegensatz zum Englischen wird in Fällen wie dem Ansteuern eines Ortes, dem Verlassen eines Ortes oder dem Spezifizieren von etwas durch das Hinzufügen der not-

wendigen Suffixe am Ende der Wörter angegeben. In den Fällen, in denen diese Suffixe hinzukommen, wird jedes Wort, obwohl der Wortstamm derselbe ist, als ein einzigartiger Wert angesehen, was zu einer Erhöhung der Komplexität führt. (Zum Beispiel haben ev, ev'e, evde, evden und ev'i den gleichen Wortstamm 'ev', aber nach der Vorverarbeitungsphase werden die Satzzeichen gelöscht und die Suffixe mit dem Wort kombiniert. Obwohl sie also denselben Wortstamm haben, betrachtet der Algorithmus sie als eigenständige Wörter).
Aus dem oben genannten Grund kann die Komplexität des Algorithmus steigen, wenn zu viele eindeutige Wörter vorhanden sind, und die nächste Wortsuche kann schwierig werden. (Stellen Sie sich das Ganze als einen Baum mit vielen Ästen vor: Je höher die Anzahl der eindeutigen Wörter, desto höher die Anzahl der Äste und desto schwieriger ist es, das richtige Ergebnis zu finden.)
Da der Zugang zu Rohtextdaten für türkische Bücher begrenzter ist als für andere Sprachen, gab es in dem verwendeten Datensatz viele Tippfehler in Wörtern. Wenn fehlende oder falsch geschriebene Wörter in der vortrainierten Einbettungsschicht enthalten sind, haben diese Wörter keinen Einfluss auf den Lernprozess, wodurch es schwieriger wird, eine höhere Genauigkeit zu erreichen. Erfolgreiche Ergebnisse auf dem Gebiet des NLP in der türkischen Sprache können leicht erzielt werden, wenn die Probleme, die ich in diesem Abschnitt und im Abschnitt über zukünftige Arbeiten erwähnt habe, gelöst werden.

6.1 Zukünftige Arbeiten

Obwohl die Ergebnisse der Studie gering erscheinen, ist der Mangel an Studien auf dem Gebiet des NLP für Sprachen wie Türkisch wichtig, um künftige Studien auf diesem Gebiet zu beleuchten. Mit der Zunahme von Studien in diesem Bereich und der Verwendung verschiedener Algorithmen können die Schwierigkeiten, die in dieser Arbeit aufgetreten sind, leicht überwunden werden.
Da ein in altem Türkisch geschriebenes Buch als Datensatz verwendet wird, sind die Komplexität und die Verständlichkeit der Wörter gering. In diesem Zusammenhang werden neuere und einfachere Texte auch die Genauigkeit erhöhen.
Da der in dieser Arbeit verwendete Datensatz relativ klein ist, eignet er sich auch eher für eine Überanpassung während der Trainingsphase. Solche Probleme werden bei der Arbeit mit einem größeren Datensatz weniger häufig auftreten.
Es wird von großer Bedeutung sein, die Tippfehler, Interpunktionsfehler und strukturellen Fehler im Datensatz zu bereinigen.
Obwohl der Wortstamm mit den Ableitungs- und Flexionssuffixen identisch ist, erscheint er wie ein neues Wort, was dazu führt, dass das Modell schlechtere Ergebnisse

erzielt. Eine Methode, die in Zukunft ausprobiert werden kann, besteht darin, die Komplexität zu reduzieren, indem nur die Wurzeln der Wörter berücksichtigt werden. Dadurch wird die Anzahl der eindeutigen Wörter reduziert.
In dieser Arbeit wurde nur die Reihenfolge der Wörter untersucht. Abgesehen davon machen Strukturen wie Länge, Häufigkeit usw. das Wort aus. Eine Situation, in der diese ebenfalls berücksichtigt werden, kann ebenfalls in Betracht gezogen werden.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1]Song, Z., Liu, L., Song, W., Zhao, X., amp; Du, C. (2019). A neural network model for Chinese sentence generation with key word. 2019 IEEE 9th International Conference on Electronics Information and Emergency Communication (ICEIEC).
[2]Yang, J., Wang, H., amp; Guo, K. (2020). Natural language word prediction model based on Multi-window convolution and residual network. IEEE Access, 8, 188036–188043.
[3]Shakhovska, K., Dumyn, I., Kryvinska, N., amp; Kagita, M. K. (2021). An approach for a next-word prediction for Ukrainian language. Wireless Communications and Mobile Computing, 2021, 1–9.
[4]Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, Daniel Ramage (2019) Federated Learning for Mobile Keyboard Prediction, arXiv:1811.03604[cs].
[5]Murat H., S. (2006). A brief review of feed-forward neural networks. Communications Faculty Of Science University of Ankara, 50(1), 11–17.
[6]Aggarwal, C. C. (2019). Neural networks and deep learning: A textbook. Springer. Page 115
[7]Ge´ron Aure´lien. (2019). Hands-on machine learning with scikit-learn and tensorflow: Concepts, tools, and techniques to build Intelligent Systems. O'Reilly. Page 498
[8]Aggarwal, C. C. (2019). Neural networks and deep learning: A textbook. Springer. Page 278
[9]Sundermeyer, M., Schlüter, R., Ney, H. (2012). LSTM neural networks for language modeling. Interspeech 2012.
[10]Ganai, A. F., amp; Khursheed, F. (2019). Predicting next word using RNN and LSTM cells: Stastical language modeling. 2019 Fifth International Conference on Image Information Processing (ICIIP).
[11]https://blog.devgenius.io/next-word-prediction-using-long-short-term-memory-lstm-13ea21eb9ead
[12]https://machinelearningmastery.com/how-to-develop-a-word-level-neural-language-model-in-keras/
[13]https://github.com/inzva/Turkish-GloVe
Iqbal, T., Qureshi, S. (2020). The survey: Text generation models in deep learning. Journal of King Saud University - Computer and Information Sciences

Chakraborty, S., Banik, J., Addhya, S., amp; Chatterjee, D. (2020). Study of dependency on number of LSTM units for character based text generation models. 2020 International Conference on Computer Science, Engineering and Applications (ICCSEA).

Tsoi, A. C. (1998). Recurrent neural network architectures: An overview. Adaptive Processing of Sequences and Data Structures, 1–26.

Ralf C. Staudemeyer, Eric Rothstein Morris. (2019). Understanding LSTM a tutorial into Long Short-Term Memory Recurrent Neural Networks

Yu, Y., Si, X., Hu, C., amp; Zhang, J. (2019). A review of Recurrent Neural Networks: LSTM cells and network architectures. Neural Computation, 31(7), 1235–1270.

Martin Sundermeyer, Ralf Schluter, and Hermann Ney (2012). LSTM Neural Networks for Language Modeling

Svozil, D., Kvasnicka, V., amp; Pospichal Jirˆí. (1997). Introduction to multi-layer feed-forward neural networks. Chemometrics and Intelligent Laboratory Systems, 39(1), 43–62.

Atlinar, F., Ayar, T., Darrige, A., AlQays, S., Bagci, A., amp; Amasyali, M. F. (2020). Masked word prediction with statistical and neural language models. 2020 Innovations in Intelligent Systems and Applications Conference (ASYU).

Hakan Erten, Mehmet Serdar Güzel and Erkan Bostancı (2020).Generating Turkish lyrics with Long short term memory

Rojas, R. (1996). The backpropagation algorithm. Neural Networks, 149–182.

Du, K.-L., amp; Swamy, M. N. (2013). Multilayer perceptrons: Architecture and error backpropagation. Neural Networks and Statistical Learning, 83–126.

A.D.Dongare, R.R.Kharde, Amit D.Kachare (2012). Introduction to Artificial Neural Network

Mishra, M., amp; Srivastava, M. (2014). A view of artificial neural network. 2014 International Conference on Advances in Engineering amp; Technology Research (ICAETR - 2014).

Ge´ron Aure´lien. (2019). Hands-on machine learning with scikit-learn and tensorflow: Concepts, tools, and techniques to build Intelligent Systems. O'Reilly. Pages (279-294, 331-370, 497-522, 525-534)

Aggarwal, C. C. (2019). Neural networks and deep learning: A textbook. Springer. Pages (3-29i 107-124, 129-141, 271-309, 318-334)

Goodfellow, I., Bengio, Y., amp; Courville, A. (2018). Deep learning. MITP. Pages (197-204, 373-416)

Chollet, F. (2021). Deep learning with python. Manning. Pages (27-37, 58-60, 101-102, 104-109, 111-114, 180-206, 271-279)

Code link: https://github.com/sarp-u/Next-Word-Prediction

Eidesstattliche Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbständig verfasst zu haben.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben.

Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Istanbul, 18. Juli 2022

Kamil Uğur Sarp