technische universität
dortmund

Bachelorarbeit

Analyse von Autoritätsausübung in Online-Diskussionsforen mittels Recurrent Neural Networks

Berat Özdemir
06. Juli 2018

Gutachter:
Prof. Dr. Katharina Morik
Lukas Pfahler (M.Sc)

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl 08
http://www-ai.cs.uni-dortmund.de

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation und Hintergrund

Das Internet bietet eine große Vielfalt an Plattformen zum Austausch von Gedanken, Meinungen, Erfahrungen und Wissen wie beispielsweise Kommentar-Sektionen, Online-Chats, soziale Netzwerke und Diskussionsforen. Weltweit nutzen laut Angaben des „Global Digital Report 2018“ [4] vier Milliarden Menschen das Internet. Die Daten, welche von der Masse an Nutzern erzeugt werden, sind von besonderem Interesse für die Sozialwissenschaften, Marktforschung und Sicherheitsbehörden, da aus dieser Menge an anfallenden Daten viel Wissen gewonnen werden kann. Durch die Menge der Daten ist es jedoch unmöglich, diese „von Hand“ auszuwerten, weshalb hier die Informatik als interdisziplinäre Wissenschaft in den Vordergrund rückt.

1.2 Aufbau der Arbeit

In dieser Arbeit werden die Nutzer eines religiösen Online-Forums bezüglich ihrer Autoritätsausübung innerhalb des Forums analysiert. Dabei ist die Zielsetzung Nutzer mit hohem Einfluss auf Gespräche und somit einer hohen Autoritätsausübung zu finden und zu gruppieren. Im Gegensatz zum üblichen Ansatz über Graphanalysen wird in dieser Arbeit ein Modell untersucht, welches durch ein Recurrent Neural Network (im Folgenden RNN) erlernt wird. Dieser Ansatz liegt nahe, da RNNs ein beliebtes, häufig genutztes Werkzeug bei der Arbeit mit sequentiellen Daten sind und ein Online-Forum prinzipiell aus Sequenzen von Nutzer-Beiträgen besteht. Daher wird mittels zwei Ansätzen untersucht, wie sehr sich Autoritätsbeziehungen im erlernten Modell wieder spiegeln.
In Kapitel 2 wird zunächst die mathematische Notation festgelegt, es werden Begriffe definiert und erklärt, um Uneindeutigkeiten zu vermeiden. Es wird eine kurze Einführung in Autoritätsbeziehungen gegeben und diskutiert, welche Merkmale sich zur Untersuchung eignen. Darauf hin werden in Kapitel 3 RNN in der Grundform und in der Variante „Long

short-term memory“(im Folgenden LSTM) vorgestellt und diskutiert, da die Bachelorarbeit grundlegend auf Verwendung dieser basiert. Danach wird in Kapitel 4 auf die Aufbereitung des Datensatzes eingegangen, in der die Probleme und Lösungsansätze erklärt werden. Auf Grundlage des vorangegangen wird dann in Kapitel 5 die Methode der Versuchsdurchführung erläutert, bei der zum einen auf den Aufbau des verwendeten Modells und zum anderen auf die beiden Ansätze zur Bestimmung autoritärer Nutzer eingegangen wird. Beim ersten Ansatz werden Gradienten im Modell und beim zweiten die mittlere Änderung des Zustandsvektors des RNN untersucht. Anschließend wird erklärt, wie die Nutzer basierend auf den beiden Ansätzen eingeordnet werden.
Nachdem die Theorie diskutiert wurde, wird in Kapitel 6 die Versuchsdurchführung beschrieben und die Ergebnisse präsentiert. Dabei werden die verschiedenen Ansätze untereinander und abschließend mit Ergebnissen ähnlicher Arbeiten verglichen. Im letzten Kapitel wird die Bachelorarbeit zum Abschluss noch einmal Zusammengefasst.

1.3 Verwandte Arbeiten

Im folgenden Abschnitt werden verwandte Arbeiten kurz zusammengefasst, bei denen entweder Gradienten bezüglich der Eingabe der Modelle zwecks Interpretation verwendet oder Interaktionen zwischen Nutzern von Online-Plattformen maschinell untersucht wurden, um so zum einen die Anwendbarkeit der Gradienten-basierten Analyse zu zeigen und zum Anderen um andere Methoden zur Untersuchung von einflussreichen Nutzern zu zeigen.
Die erste Arbeit ist das Werk *Interpretation of Prediction Models Using the Input Gradient* von *Yotam Hechtlinger* [7], auf der auch die Untersuchung in dieser Arbeit basiert. Dort wird eine Methode zum besseren Verständnis von Vorhersagemodellen vorgestellt, die auf Gradienten bezüglich der Eingabevariablen basiert. Diese Methode wird genauer in Kapitel 5 erklärt. In dem Paper werden zwei Beispiele angegeben, um die Effektivität der Methode zu präsentieren. Eines der Beispiele verwendet ein *Convolutional Neural Network*, welche mit einer Eingabe im *Bag of Words* Modell in dem *IMDB* Datensatz[1] vorhersagen soll, ob eine Rezension positiv oder negativ ist. Nach erlernen des Modells wurden bezüglich der Eingaben die Gradienten berechnet. Das Ergebnis war, dass Wörter, wie zum Beispiel *excellent*, *great* und *amazing*, einen hohen Einfluss darauf hatten, ob die Rezension als positiv klassifiziert wurde. Wörter, wie zum Beispiel *worst*, *bad* und *boring*, hatten einen hohen Einfluss darauf, ob die Rezension als negativ klassifiziert wurden. Diese Ergebnisse sind naheliegend für Menschen, da diese Wörter eine positive beziehungsweise negative Konnotation haben. Daher ist dieses Ergebnis eine Bestätigung für die Anwendbarkeit der Untersuchung auf Basis von Gradienten. Interessanterweise ist dabei noch anzumerken, dass in der Arbeit, basierend auf den berechneten Gradienten, das Vorhersagemodell approximiert werden konnte, indem es auf das Skalarprodukt $\langle \widetilde{g}, x \rangle$, wobei $\widetilde{g}$ der Vektor der

[1] Dieser Datensatz beinhaltet 50,000 Film-Rezensionen, welche positiv oder negativ gelabled sind

mittleren partiellen Ableitungen ist und x der *Bag of Words* Eingabe-Vektor ist, reduziert wurde. Dabei stimmen die Vorhersagen der Approximation bei 99.6% der Beobachtungen der Testmenge mit denen des komplexeren *Convolutional Neural Network* Modells überein.

Eine andere verwandte Arbeit ist der Artikel *Determining Influential Users in Internet Social Networks*[15] von *Michael Trusov* bei der mittels Poisson Regression untersucht wird, welchen Einfluss Nutzer auf die Aktivität von anderen Nutzern auf Social-Media-Plattformen haben. Unter anderem gehört zu den Ergebnissen, dass anhand der Daten einer Social-Media-Plattform, die Aktivität eines Nutzer auf dieser Plattform im Durchschnitt von circa ein fünftel seiner Freunde auf der Plattform beeinflusst wird.

Kapitel 2

Begriffserklärungen und Definitionen

In diesem Kapitel wird zunächst die mathematische Notation festgelegt, danach auf die für die Arbeit wichtigen Mathematischen Grundlagen eingegangen. Dabei wird der Gradient erklärt und die Grundbegriffe des Machine Learning werden erläutert. Abschließend wird auf die zu untersuchende Autoritätsbeziehung eingegangen.

2.1 Mathematische Notation

Hier wird die in der Arbeit verwendete mathematische Notation festgelegt, welche sich stark an die des Buches „Deep Learning“ [6] orientiert.

Bezeichnung	Beispiel	Erklärung
Skalar	a	Skalare Größen werden in normaler Schrift mit einem Kleinbuchstaben dargestellt
Vektor	$\boldsymbol{a}$	Vektoren werden durch fette Schrift mit einem Kleinbuchstaben dargestellt
i-te Komponente eines Vektors	a_i	Normale Kleinbuchstaben mit einem Index i stellen die i-te Komponente in einem Vektor dar
Konkatenation	$\boldsymbol{a} \circ \boldsymbol{b}$	Der $\circ$-Operator stellt in dieser Arbeit die Konkatenation zweier Vektoren dar $\boldsymbol{a} \circ \boldsymbol{b} = (a_1, ..., a_n, b_1, ..., b_m)$
Hamarad-Produkt	$\boldsymbol{a} \odot \boldsymbol{b}$	Der $\odot$-Operator stellt die Komponentenweise Multiplikation zweier Vektoren gleicher Dimension dar: $\boldsymbol{a} \odot \boldsymbol{b} = (a_1 \cdot b_1, ..., a_n \cdot b_n)$
Matrix	$\boldsymbol{A}$	Matrizen werden durch Großbuchstaben in fetter Schrift dargestellt
Transponierte Matrix	$\boldsymbol{A}^\top$	-
Element i, j der Matrix A	$A_{i,j}$	Ein Eintrag in der i-ten Zeile und j-ten Spalte der Matrix wird durch Großbuchstaben mit zwei Indizes in normaler Schrift dargestellt
i-te Zeile der $Q \times P$ Matrix A	$A_{i,:}$	Die i-te Zeile der Matrix als Zeilenvektor $A_{i,:} = (A_{i,1}, ..., A_{i,P})$
j-te Spalte der $Q \times P$ Matrix A	$A_{:,j}$	Die j-te Spalte der Matrix als Spaltenvektor $A_{:,j} = (A_{1,j}, ..., A_{Q,j})^\top$

2.2 Gradienten

Ein wichtiger Bestandteil und Grundlage für diese Arbeit sind Gradienten. Zum einen, weil die Lernalgorithmen auf ihnen basieren und zum Anderen, weil sie die Grundlage für die Analyse, welche in Kapitel 5 beschrieben wird, darstellen. Dabei ist die Definition des Gradienten folgende[10]:

2.2.1 Definition. Sei $\langle\ ,\ \rangle$ das euklidsche Skalarprodukt auf $\mathbb{R}^n$.. Der Gradient einer Funktion $f : \mathbb{R}^n \to \mathbb{R}$ in a bezüglich des Standardskalarprodukts ist der durch die Forderung

$$df(a)h = \partial_h f(a) = \langle grad\ f(a), h\rangle$$

eindeutig bestimmte Vektor in $\mathbb{R}^n$.

In kartesischen Koordinaten ergibt sich für den Gradienten:

$$grad\ f = \nabla f = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix} \tag{2.1}$$

Wobei $\frac{\partial f}{\partial x_i}$ die partielle Ableitung der Funktion bezüglich der Variable x_k ist. Sie ist wie folgt definiert:

$$\frac{\partial f}{\partial x_k}(x) = \lim_{h\to 0} \frac{f(x_1, ..., x_k + h, ..., x_n) - f(x_1, ..., x_k, ..., x_n)}{h} \tag{2.2}$$

Somit gibt die partielle Ableitung bezüglich der Variable x_k einer Funktion, ein Maß für den Einfluss der Variable auf den Wert der Funktion an, da der Grenzwert angibt, wie das Verhältnis der Änderung h der Variable zu h selbst ist.

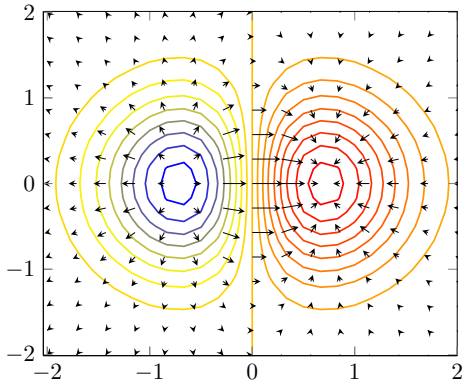

Abbildung 2.1: Gradienten von $f(x, y) = x \cdot e^{-x^2-y^2}$

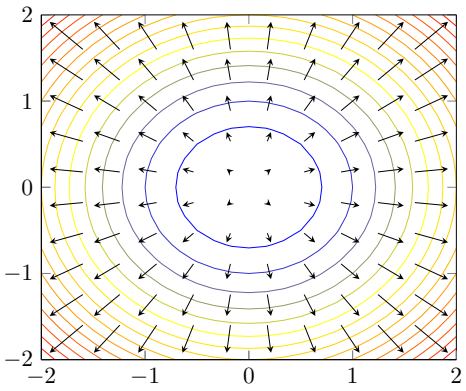

Abbildung 2.2: Gradienten von $f(x, y) = x^2 + y^2$

Die Intuition für die partielle Ableitung ist daher die der Steigung. Wird im Punkt a einer Funktion der Gradient berechnet, so ergibt sich ein Vektor der die Richtung des

steilsten Werteabstiegs zeigt. Die Länge des Gradienten kann dabei als Stärke der Steilheit angesehen werden. In Abbildung 2.1 und 2.2 ist dies gut zu erkennen. Es sind zwei Funktionen als Konturdiagramm abgebildet. Dabei haben die Funktionen an roten Stellen einen niedrigen und an blauen Stellen einen hohen Wert. Die Gradienten sind durch die Pfeile dargestellt und zeigen, wie zu sehen ist, in Richtung der niedrigen Werte.
Diese Eigenschaft wird bei Optimierungsaufgaben ausgenutzt, denn die Gradienten dienen als Wegweiser zu einem Minimum einer Funktion. Die Optimierung von RNNs basiert auf Gradienten und wird in Kapitel 3 vorgestellt.

2.3 Grundbegriffe des Machine Learning

Beim Machine Learning geht es um das generieren von Wissen aus einer Menge von Daten. Zu den Aufgabenstellungen gehören unter anderem Klassifikation, Regressionsanalysen und Clusteranalysen. Dabei wird zwischen *supervised learning* und *unsupervised learning* unterschieden. Beim *supervised learning* versucht der Algorithmus anhand vorgegebener Ein- und Ausgabepaare (x, y) aus dem Datensatz, welche *Samples* genannte werden, eine Funktion $F : X \to Y$ zu erlernen, die die vorliegenden Wertepaarungen bestmöglich bezüglich eines vorgegebenen Maßes erfüllt. Ein Beispiel hierfür wären maschinelle Übersetzer, welche die Übersetzung anhand von Daten lernen, indem sie beim Training Sätze in Sprache A als Eingabe und die Übersetzung in Sprache B als Ausgabe vorgegeben bekommen. Unter *Training* wird das Anpassen der Modellparameter verstanden. Dabei ist ein Modell eine Funktion $F : X \to Y$ mit $F(x; \Theta)$, dessen Verhalten durch eine Menge von Parametern Θ beeinflusst wird. Diese Parameter werden im Training so angepasst, dass die Funktion gewünschte Ausgaben berechnet. Ein Trainingsverfahren für *Recurrent Neural Networks* basierend auf Gradienten wird in Kapitel 3 erklärt.
Viele Verfahren basieren auf dem Minimieren von *Loss*-Funktionen, welche ein Maß für die mittleren Kosten der Modell-Vorhersagen sind. Diese Kosten werden von einer *Cost*-Funktion berechnet und können je nach Aufgabenstellung variieren. Eine einfache *Loss*-Funktion wäre die mittlere Quadratische Abweichung

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2,$$

wobei Y_i die Zielausgabe, $\hat{Y}_i$ die vom Modell berechnete Ausgabe ist und n die Anzahl aller Samples ist.

2.4 Autoritätsbeziehung

In diesem Kapitel wird auf die zu untersuchende Autoritätsbeziehung zwischen Nutzern eingegangen. Dazu werden die Eigenschaften einer derartigen Beziehung erklärt und darauf hin entschieden, welche Merkmale sich zur Erkennung von Autoritätspersonen eignen könnten. Laut Heinrich Popitz, findet eine Autoritätsbeziehung zwischen einer Autoritätsperson und einem Autoritätsabhängigen statt. Dabei schreibt er der Autoritätsperson eine „autoritative“ Macht zu, dessen Ausübung ohne Machtmittel wie Strafen, Belohnungen, Drohungen oder Versprechen stattfinden könne. Die Wirkung dieses Verhältnisses reiche für den Autoritätsabhängigen über eine bloße Verhaltensanpassung hinaus. Es führe zur Übernahme der Perspektiven und Kriterien der Autoritätsperson, damit zu einer nicht kontrollbedürftigen Konformität und zu einer psychischen Anpassung.[12, S. 79]
Daraus leiten sich folgende Eigenschaften einer Autoritätsbeziehung ab:

- Die Autoritätsperson kann das Verhalten des Autoritätsabhängigen, auch außerhalb seines Kontrollbereiches, beeinflussen.
- Das Beeinflussen des Verhaltens geschieht ohne Machtmittel.
- Der Autoritätsabhängige übernimmt die Perspektive und Kriterien der Autoritätsperson.
- Der Autoritätsabhängige erstrebt eine Konformität mit der Einstellung der Autoritätsperson.

Für diese Arbeit sind die letzten beiden Eigenschaften von Wichtigkeit. Auf Basis dieser wird versucht Autoritätspersonen zu erkennen. Die Grundidee dahinter ist, dass durch das Übernehmen der Perspektive und Kriterien sich der Wortgebrauch des Autoritätsabhängigen dem der Autoritätsperson anpasst. Bezogen auf das zu analysierende Forum wird erwartet, dass jene Nutzer Autoritätspersonen sind, die einen hohen Einfluss auf die gebrauchten Wörter in Beiträgen haben und somit Gespräche lenken.
Ein weiterer Ansatz ist es nach Nutzern zu suchen, welche generell einen hohen Einfluss auf die Dynamik, beziehungsweise die Vorhersagen des Modells, im Forum haben. Diese beiden Ansätze werden in Kapitel 5 näher beschrieben.

Kapitel 3

Recurrent Neural Networks

Recurrent Neural Networks gehören zur Familie der neuronalen Netze und sind auf das Verarbeiten von sequentiellen Daten spezialisiert. So sind sie anders als konventionelle Feedforward neuronale Netze nicht an eine Eingabe beziehungsweise Ausgabe mit fester Größe gebunden.
Sie finden in einem breiten Spektrum von Gebieten Anwendung. Beispielsweise nutzt Google-Translate ein Modell, welches aus mehreren Schichten von LSTM Einheiten, welche in Kapitel 3.2 näher beschrieben werden, besteht [16].

3.1 Standard Recurrent Neural Network

Wie bereits beschrieben, verarbeiten RNNs Sequenzen von Daten $\boldsymbol{i}^{(1)}, ..., \boldsymbol{i}^{(n)}$. Die Grundkonzepte dahinter sind, im Vergleich zu konventionellen Feedforward Neural Networks, eine rekurrente Verbindung zu nutzen, einen Zustandsvektor $\boldsymbol{h}_t$ einzuführen und gelernte Parameter unabhängig vom Zeitpunkt t zu verwenden. Hierbei bezieht sich der Term *Zeitpunkt* nicht zwangsläufig auf einen Abschnitt der Zeit in der realen Welt, sondern auf die Position der Eingabe in der Eingabesequenz. Beispielsweise könnte der Zeitpunkt t in einem Modell, welches natürlichsprachliche Texte als Eingabe hat, die Position eines Wortes in einem Satz sein. Der Zustandsvektor $\boldsymbol{h}_t$ stellt eine Zusammenfassung für das Modell relevanter Aspekte vorangegangener Eingaben dar, welcher prinzipiell verlustbehaftet ist, da eine beliebig lange Sequenz von Eingabedaten $\boldsymbol{i}^{(1)}, ..., \boldsymbol{i}^{(t)}$ auf den Zustandsvektor $\boldsymbol{h}_t$ mit fester Länge abgebildet wird. Die Stärke von RNNs zeichnet sich durch dieses *Gedächtnis* aus. Es wurde sogar bewiesen, dass das RNN in Abbildung 3.1 Turing-Vollständig ist, sodass mit geeigneten Parametern jedes Programm mit ihm simuliert werden kann[14].

3.1.1 Struktureller Aufbau

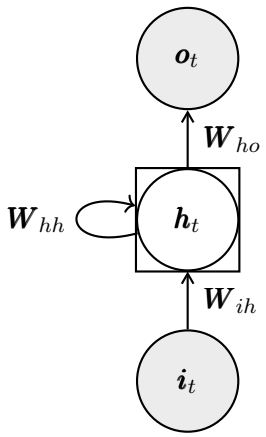

Abbildung 3.1:
Aufbau eines Standard RNN

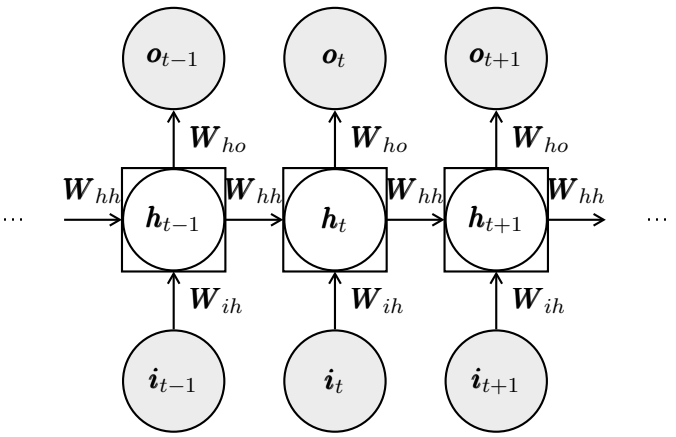

Abbildung 3.2:
Standard RNN aufgerollt

In Abbildung 3.1 ist der Aufbau eines Standard RNNs als Graph zu sehen, wie er häufig in der Literatur anzutreffen ist. Das abgebildete Netzwerk erzeugt pro Eingabevektor einen Ausgabevektor. Hierbei ist $\boldsymbol{i}_t$ der Eingabevektor, $\boldsymbol{h}_t$ der Zustandsvektor und $\boldsymbol{o}_t$ der Ausgabevektor. Die zu erlernenden Parameter sind die Gewichts-Matrizen $\boldsymbol{W}_{ih}$, $\boldsymbol{W}_{hh}$ und $\boldsymbol{W}_{ho}$. Diese Parameter werden über alle Zeitpunkte hinweg benutzt, wie in Abbildung 3.2 zu sehen ist, welche das RNN, statt mit rekurrenter Verbindung, als aufgerollten Graphen darstellt, in der alle Zeitschritte dargestellt werden.
Der Zustandsvektor $\boldsymbol{h}_t$ wird gemäß folgender Formel berechnet:

$$\boldsymbol{h}_t = f_{act}(\boldsymbol{b}_h + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{ih}\boldsymbol{i}_t) \tag{3.1}$$

Die Funktion f_{act} ist die Aktivierungsfunktion, welche je nach Anwendungsgebiet des Netzes variieren kann. Üblicherweise wird in RNNs die *sigmoid*-Funktion

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{3.2}$$

oder die *tanh*-Funktion

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3.3}$$

als Aktivierungsfunktion gewählt.

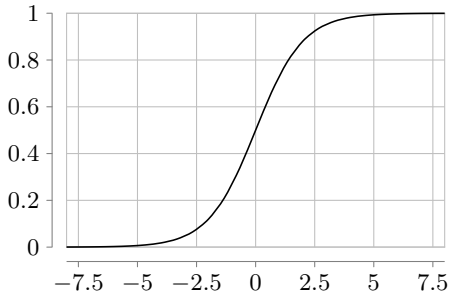

Abbildung 3.3: *sigmoid*-Funktion

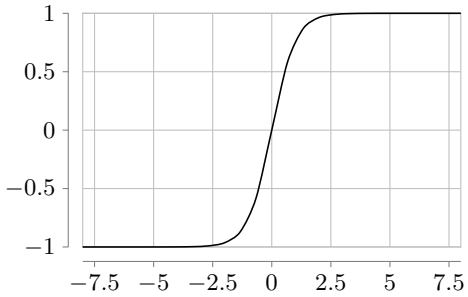

Abbildung 3.4: *tanh*-Funktion

Der Ausgabe-Vektor $\boldsymbol{o}_t$ errechnet sich wie folgt aus $\boldsymbol{h}_t$:

$$\boldsymbol{o}_t = \boldsymbol{W}_{ho}\boldsymbol{h}_t + \boldsymbol{b}_o \tag{3.4}$$

Die Vektoren $\boldsymbol{b}_h$ und $\boldsymbol{b}_o$ stellen die Bias-Vektoren dar. Weiterhin nehmen wir nun an, dass die Ausgabe des RNN Diskret ist, sodass $\boldsymbol{o}_t$ als nicht-normalisierte logarithmische Wahrscheinlichkeitsverteilung interpretiert werden kann[6, Seite 379]. Wird nun die *softmax*-Operation auf den Ausgabe-Vektor angewandt, ergibt sich der Vektor $\hat{\boldsymbol{y}}_t$, dessen Einträge sich zu einer Eins summieren, welcher eine normalisierte Wahrscheinlichkeitsverteilung über der Ausgabe darstellt.

$$\hat{\boldsymbol{y}}_t = softmax(\boldsymbol{o}_t) \tag{3.5}$$

Die *Softmax*-Funktion $\boldsymbol{\sigma}(\boldsymbol{v})$ ist eine Generalisierung der logistischen Funktion und bildet einen K-dimensionalen Vektor $\boldsymbol{v}$ zu einem K-dimensionalen Vektor $\boldsymbol{v}'$ ab, dessen Komponenten im Wertebereich $[0, 1]$ liegen und sich zu 1 aufsummieren lassen. Dies geschieht gemäß folgender Formel:

$$v'_j = \sigma(\boldsymbol{v})_j = \frac{e^{v_j}}{\sum_{k=1}^{K} e^{v_k}} \tag{3.6}$$

Es ist zu sehen, dass $\sum_{k=1}^{K} v'_j = 1$ gelten muss und zusätzlich alle Komponenten von v'_j positiv sind.
Um das Netzwerk trainieren zu können, muss eine *Loss-Funktion* ausgewählt und diese über das Anpassen der Gewichtsmatrizen minimiert werden. In der nächsten Sektion wird das Training durch *Backpropagation Through Time* und *Stochastic Gradient Descent* näher erläutert.

3.1.2 Training durch „Stochastic Gradient Descent“ und „Backpropagation Through Time“

Stochastic Gradient Descent

Wie in vielen Bereichen des Machine-Learning erfolgt das überwachte Training eines RNNs durch die Minimierung einer Kosten-Funktion. Diese Funktionen werden *Loss*-Funktionen $L(y, \hat{y})$ genannt und bemessen die *Kosten* für die getroffene Vorhersage $\hat{y}$ anhand der

vorgegebenen Ziele y durch das erlernte Modell. Eine Methode die Kostenfunktion zu Minimieren, erfolgt durch das *Gradient Descent*-Verfahren[3]. Die Kern-Idee dahinter ist, dass die Gradienten den *Weg* zu einem Minimum weisen und daher anhand dieser die zu erlernenden Parameter aktualisiert werden. Sei nun L die Loss-Funktion, f_w das erlernte Modell mit Parametern w und (x_i, y_i) das Sample i, wobei x_i die Eingabe ist und y_i die gewünschte Ausgabe des Modells. Beim *Gradient Descent* wird der mittlere Wert des Losses für n Samples minimiert. Dieser Wert wird auch *Empirical Risk* $E_n(f)$ genannt

$$E_n(f) = \frac{1}{n} \sum_{i=1}^{n} L(f_w(x_i), y_i) \tag{3.7}$$

Bei jeder Iteration werden die Gewichte w gemäß folgender Formel aktualisiert:

$$w_{t+1} = w_t - \gamma \frac{1}{n} \sum_{i=1}^{n} \nabla_w L(f_{w_t}(x_i), y_i) \tag{3.8}$$

Dabei stellt das γ die Gewichtung des Schrittes dar und muss sinnvoll ausgewählt werden. Wird die initiale Gewichtung w_0 hinreichend nah am Optimum und γ adäquat gewählt, so ist die Konvergenz linear[5]. Um diese Optimierung zu beschleunigen, wird das Berechnen der Gradienten approximiert. Eine beliebte Methode ist hierbei das Verfahren *Stochastic Gradient Descent*. Bei diesem Verfahren wird statt dem Gradienten von $E_n(f_w)$, der Gradient von $L(f_{w_t}(x_i), y_i)$ berechnet. Daraus resultiert die vereinfachte Gleichung

$$w_{t+1} = w_t - \gamma \nabla_w L(f_{w_t}(x_t), y_t) \tag{3.9}$$

Die Güte der Aktualisierungsformel hängt jedoch stark von der zufälligen Auswahl der Samples ab, die zur Berechnung der Gradienten genutzt werden.

Backpropagation Through Time

Da für *Stochastic Gradient Descent* die Gradienten berechnet werden müssen, wird im Folgenden eine Methode vorgestellt um in RNNs die Gradienten zu berechnen. Diese nennt sich *Backpropagation Through Time* und basiert im Grunde genommen auf der Kettenregel der Differentialgleichungen: Sei $f(y) = z$ und $g(x) = y$, dann gilt

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx} = f'(y) \cdot g'(x) = f'(g(x)) \cdot g'(x) \tag{3.10}$$

Sei nun ein RNN wie in Kapitel 3.1.1. definiert gegeben. Dann werden Ausgehend von der *Loss*-Funktion L die Gradienten berechnet. Dazu müssen die partiellen Ableitungen bezüglich der Gewichtsmatrizen berechnet werden.

$$\frac{\partial L}{\partial \boldsymbol{W}_{ho}}, \frac{\partial L}{\partial \boldsymbol{W}_{hh}}, \frac{\partial L}{\partial \boldsymbol{W}_{ih}} \tag{3.11}$$

Um dies zu bewerkstelligen, wird die Kettenregel mehrfach genutzt:

$$\frac{\partial L}{\partial \boldsymbol{W}_{ho}} = \frac{\partial L}{\partial \hat{\boldsymbol{y}}_t} \frac{\partial \hat{\boldsymbol{y}}_t}{\partial \boldsymbol{o}_t} \frac{\partial \boldsymbol{o}_t}{\partial \boldsymbol{W}_{ho}} \tag{3.12}$$

$$\frac{\partial L}{\partial \boldsymbol{W}_{hh}} = \frac{\partial L}{\partial \hat{\boldsymbol{y}}_t} \frac{\partial \hat{\boldsymbol{y}}_t}{\partial \boldsymbol{o}_t} \frac{\partial \boldsymbol{o}_t}{\partial \boldsymbol{h}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{W}_{hh}} \tag{3.13}$$

$$\frac{\partial L}{\partial \boldsymbol{W}_{hh}} = \frac{\partial L}{\partial \hat{\boldsymbol{y}}_t} \frac{\partial \hat{\boldsymbol{y}}_t}{\partial \boldsymbol{o}_t} \frac{\partial \boldsymbol{o}_t}{\partial \boldsymbol{h}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{W}_{ih}} \tag{3.14}$$

Je nach Loss-Funktion und Berechnung von $\hat{\boldsymbol{y}}_t$ können nun die partiellen Ableitungen abgearbeitet werden, da sowohl Loss-Funktion, als auch die Aktivierungsfunktionen differenzierbar sein müssen und somit die abgebildeten partiellen Ableitungen bekannt sind. Um $\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{W}_{ih}}$ abzuleiten muss gegebenfalls erneut die Kettenregel angewandt werden. Dies bringt auch das Problem des *Vanishing Gradient mit*, da durch wiederholte Multiplikation kleine Gradienten *erlöschen* und somit Gewichte nicht aktualisiert werden. Im nächsten Kapitel wird eine Variation von RNNs vorgestellt, die dieses Problem beheben.

3.2 LSTM

Das Konzept *Long Short-Term Memory*[8] wurde entwickelt um das Problem des *Vanishing/Exploding Gradient* zu lösen. Die Kernkonzepte dahinter sind die sogenannten *Gates*, welche den Einfluss der Eingabe und des letzten internen Zustandes auf den neuen internen Zustand regulieren. Im Folgenden wird ein Einblick auf die Architektur gegeben.

3.2.1 Struktureller Aufbau

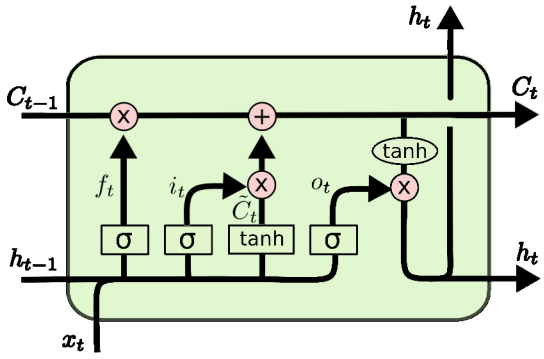

Abbildung 3.5: Aufbau einer LSTM-Zelle

Quelle: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

In Abbildung 3.5 ist der Aufbau einer LSTM-Zelle zu sehen. Die wichtigsten Bestandteile einer LSTM-Zelle sind die *Gates* f_t, i_t, o_t und der *Cell-State* C_t. Im Folgenden wird der Zusammenspiel dieser erklärt. Wie bereits oben erwähnt sollen die *Gates* die Stärke

der Einflüsse einzelner Komponenten regulieren. Dabei ist f_t, der *Forget-Gate* wie folgt definiert:

$$f_t = \sigma(\boldsymbol{W}_f \cdot (h_{t-1} \circ x_t) + \boldsymbol{b}_f) \tag{3.15}$$

Dieser Vektor soll anhand der aktuellen Eingabe und dem letzten Zustandsvektor h_{t-1} regulieren, *wie viel* Information vom alten *Cell-State* C_{t-1} in den neuen C_t gelangt. Wie alle anderen *Gates* werden die Werte des Vektors durch die *sigmoid*-Operation in den Wertebereich $[0, 1]$ abgebildet.(Siehe Abbild 3.3) Durch die Komponentenweise Multiplikation mit C_{t-1} und den Werten zwischen 0 und 1 wird diese Regulierung erreicht.
Der *Input-gate* i_t reguliert auf gleiche Weise, also die Komponentenweise Multiplikation, wie stark der Einfluss von neuen Informationen auf den *Cell State* C_t ist. Er ist wie folgt definiert:

$$i_t = \sigma(\boldsymbol{W}_i \cdot (h_{t-1} \circ x_t) + \boldsymbol{b}_i) \tag{3.16}$$

Der *Cell State* C_t setzt sich aus dem alten *Cell state* C_{t-1} und einem *Cell State Kandidat* $\widetilde{C}_t$ zusammen. Der Einfluss von C_{t-1}, also alter Information, wird durch f_t und der Einfluss von $\widetilde{C}_t$, also neuer Information, durch i_t gesteuert. Dabei wird C_t gemäß folgender Formel berechnet:

$$C_t = f_t \odot C_{t-1} + i_t \odot \widetilde{C}_t \tag{3.17}$$

Der *Cell State Kandidat* $\widetilde{C}_t$ beherbergt die neuen Informationen und wird wie folgt berechnet:

$$\widetilde{C}_t = tanh(\boldsymbol{W}_C \cdot (h_{t-1} \circ x_t) + \boldsymbol{b}_c) \tag{3.18}$$

Schließlich reguliert der *Output-Gate wie viel* vom *Cell State* C_t nach Außen, also in den Zustandsvektor h_t gelangen soll.

$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}_o \cdot (h_{t-1} \circ x_t) + \boldsymbol{b}_o) \tag{3.19}$$

Der Zustandsvektor h_t ist wie folgt definiert:

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot tanh(C_t) \tag{3.20}$$

Der Zustandsvektor ist letzlich die Schnittstelle nach Außen und anhand von ihm wird die Ausgabe des LSTMs berechnet.

Kapitel 4

Aufbereitung der vorliegenden Daten

4.1 Beschreibung der Rohdaten

Die zu verarbeitenden Daten sind ein Abbild der öffentlich zugänglichen Informationen des Online-Forums „Ahlu-Sunnah.com“ in Form einer Datei im JSON-Format. Das religiöse Online-Forum hatte laut Gründer „das Konzept eines öffentlichen, anonymen, von der Teilnehmerschaft her unbeschränkten islamischen Internetforums“ und wurde geschlossen, da dieses Konzept „gescheitert und zwangsweise dazu verdammt sei in einer geistigen Müllkippe und in einem Zankplatz verblendeter Gruppenfanatiker zu enden“.[11] Der Datensatz besteht aus einer Liste von Tupeln aus Key-Value-Pärchen, welche die Beiträge in dem Forum darstellen. Dabei besteht jeder Tupel aus mindestens folgenden Einträgen:

1. *author*: Autor des Beitrags
2. *id*: ID des Beitrags
3. *thread*: ID und Name des Threads, in welchem der Beitrag geschrieben wurde
4. *no*: Die Position des Beitrags in dem Thread
5. *html*: Roher HTML-Code des Beitrags
6. *text*: Der reine textliche Inhalt des Beitrags (ohne Schriftauszeichnungen etc.)
7. *gelehrte_references*: Erkannte Referenzen zu islamischen Gelehrten im Beitrag
8. *koran_references*: Erkannte Referenzen zu Koran im Beitrag
9. *language*: Die erkannte Sprache des Beitrags
10. *published*: Das Datum der Veröffentlichung des Beitrags im YYYY-MM-DD Format

Zusätzlich können in den Tupeln noch folgende Key-Value-Pärchen auftreten:

1. *named_ entity*: In dem Beitrag erkannte *Named Entities*
2. *bible_ references*: In dem Beitrag erkannte Bibel-Referenzen
3. *quoted_ users*: Namen der im Beitrag zitierten User
4. *media*: In dem Beitrag verlinkte Mediendateien (Smileys, Bilder, Videos)
5. *parent*: Die ID des Vorherigen Beitrags
6. *quoted_ posts*: IDs der im Beitrag zitierten Beiträge

Der Datensatz ist ca. 486 MB groß und umfasst 28282 Threads mit insgesamt 168591 Beiträgen, generiert von 3787 Nutzern. Für diese Arbeit sind vor allem die erkannten *Named Entities* von Relevanz, welche in 84288 Beiträgen erkannt wurden. Da dies nur knapp 50% aller Beiträge sind, könnte es das erlernen der Modellparameter negativ beeinflussen. Daher wird neben dem Modell, welches *Named Entities* nutzt, noch ein anderes verwendet, welche stattdessen auf einem festgelegten Vokabular basiert. Um dieses zu erzeugen, müssen die Inhalte der Beiträge vor verarbeitet werden, wie im folgenden Kapitel erklärt wird.

4.2 Theorie zur Datenvorverarbeitung bei natürlichsprachlichen Daten

Natürlichsprachliche Texte bieten eine Vielfalt an Eigenschaften, die es erschweren können mit ihnen zu arbeiten. Unter anderem gehören dazu die lexikalischen, linguistischen und morphologischen Eigenschaften. Diese liegen, vor allem in Online-Foren, verzerrt und uneinheitlich vor, da Nutzer dazu neigen ihren eigenen Schreibstil zu pflegen und wenig Wert auf Qualität zu legen. So ist es beispielsweise in Foren gängig Interpunktionen, Groß-/Klein- und allgemein Rechtschreibung zu missachten. Da diese Eigenheit der Nutzer variiert, ergibt sich dadurch eine hohe Vielfältigkeit an Variationen der lexikalischen und linguistischen Eigenschaften. Je nach Aufgabenfeld müssen die Daten spezifisch normalisiert werden. Beispielsweise könnten Rechtschreibfehler im Datensatz für bestimmte Aufgaben, wie die in dieser Bachelor Arbeit, störend sein, wo hingegen in einer Aufgabe, bei der es darum geht Autoren anhand von Texten zu erkennen, Rechtschreibfehler als Feature gewählt werden könnten. Des weiteren muss ein

4.2.1 Normalisieren der Wörter

Um die Daten zu vereinheitlichen und bessere Ergebnisse zu erzielen, werden die in den Beiträgen vorkommenden Texte normalisiert. Dies geschieht auf der lexikalischen und morphologischen Ebene. Zunächst werden alle Buchstaben in Kleinbuchstaben umgewandelt,

da somit Groß-/Kleinschreibungsfehler nicht mehr ins Gewicht fallen. Des weiteren müssen Schreibweisen vereinheitlicht werden: Beispielsweise gibt es Menschen, welche bevorzugt "ß" statt "ss" nutzen oder auch umgekehrt. Daher werden alle "ß" mit "ss" ersetzt. Aus dem selben Grund werden die Umlaute *ä, ö* und *ü*, durch *ae, oe* und *ue* ersetzt und diakritische Zeichen wie beispielsweise Akzente etc. entfernt. Für die Aufgabenstellung, Autoritätsbeziehungen über die Übernahme von genutzten Wörtern zu erkennen, ist es beispielsweise irrelevant, wie die Wörter konjugiert oder dekliniert sind. Daher werden, die in den Beiträgen vor-kommenden Wörter, auch morphologisch normalisiert, indem sie auf ihren Wortstamm abgebildet werden. Dies geschieht über das sogenannte *Stemming* oder *Lemmatization*, welche entweder fest regelbasiert sind oder auf statistischen Methoden beruhen. Bei Auf-gaben wie *Document Retrieval*, wo es darum geht relevante Dokumente in einer Menge von Dokumenten zu finden, ist die Morphologie der Wörter in den Dokumenten irrelevant bis störend, da hier nur die Bedeutung der Wörter wichtig ist und es konnte gezeigt werden, dass durch *Stemming* und *Lemmatization* bessere Ergebnisse erzielt werden können.[2] Ein weiterer Vorteil der Normalisierung liegt auch darin, dass das Vokabular kleiner wird und somit die zu erlernenden Modellparameter verringert werden.

4.2.2 Weiteres verfeinern der Daten

Bei der Arbeit mit natürlichsprachlichen Dokumenten gibt es weitere Methoden die Ergebnisse zu verbessern, in dem beispielsweise inhaltlich bedeutungslose Terme aus dem Vokabular des Modells verworfen werden. Dies wird damit begründet, dass Terme, die nahezu in jedem Dokument vorkommen, keinen Mehrwert bieten, um die Dokumente zu unterscheiden. Daher werden im Folgenden einige Maße zur Ermittlung der Relevanz von Termen vorgestellt.

- Das Zipfsche Gesetz:
  Das von *Georg Kingsley Zipf* entdeckte empirische Gesetz besagt[9], dass wenn eine Liste von Wörtern anhand ihrer Häufigkeit ihres Auftretens in einem Textkorpus absteigend sortiert wird, die Position eines Wortes umgekehrt proportional zu ihrer Häufigkeit im Textkorpus ist. Demnach ergibt sich eine hyperbelartige Verteilung der Wörter.

- Relevanz durch Häufigkeit:
  Terme, die im Dokument oder Korpus nur einmal Vorkommen, tragen wenig bis gar nicht zum Verständnis beziehungsweise Charakterisieren eines Dokumentes bei. Das gleiche gilt für Terme, welche in jedem Dokument vorkommen, wie beispielsweise Artikel oder Präpositionen. Daher ist es sinnvoll, eine obere und untere Grenze für die relative Häufigkeit eines Terms festzulegen, sodass nur Terme zwischen diesen Grenzen für die Analyse verwendet werden.

- Gewichtung der Relevanz mittels Inverser Dokumentenfrequenz (IDF)[1]
  Die Grundlage für das Maß IDF, ist die Annahme, dass ein Wort w_1, welche in wenigen Dokumenten vorkommt, eine höhere Relevanz und somit eine höhere Gewichtung als ein Wort w_2 haben muss, wenn w_2 in allen Dokumenten vorkommt, aber w_1 und w_2 gleich häufig im Korpus vorkommen. Daher ergibt sich die Gewichtung:

$$idf(w) = log_2(\frac{N}{\#(w)}) \tag{4.1}$$

  wobei N die Anzahl aller Dokumente im Korpus ist und $\#(w)$ die Anzahl der Dokumente im Korpus ist, in denen das Wort w vorkommt.

- Liste von sogenannten *stop words*
  Eine einfachere Form irrelevante Terme auszusortieren sind vorgefertigte *stop word*-Listen. Diese sind einfache Listen mit Füllwörtern wie Artikeln, Präpositionen, Pronomen etc., anhand derer das Vokabular für das Modell gefiltert werden kann.

4.3 Vorverarbeitung der Daten

Das RNN, welches trainiert werden soll und in Kapitel 5 näher vorgestellt wird, arbeitet mit Sequenzen von Vektoren und bekommt pro Zeitschritt t als Eingabe zwei Vektoren $\boldsymbol{u}_t$ und $\boldsymbol{n}_t$. Dabei werden die Beiträge in dem Forum mit diesen Vektoren kodiert:

- Der Nutzer-Vektor $\boldsymbol{u}_t$
  Dieser Vektor soll den Nutzer des Beitrags zum Zeitpunkt t kodieren. Dies erfolgt in der One-Hot Kodierung. Das heißt, es gilt $\boldsymbol{u}_t \in U \subseteq \{0,1\}^{|U'|}$, wobei $|U'|$ die Anzahl aller Nutzer im Datensatz und U' die Menge aller Nutzer ist. Des weiteren muss $\sum_{i=1}^{|U'|} u_{ti} = 1$ gelten, was bedeutet, dass genau ein Eintrag im Vektor $\boldsymbol{u}_t$ gleich Eins ist und der Rest gleich Null.

- Der Inhalts-Vektor $\boldsymbol{n}_t$
  Dieser Vektor stellt den Inhalt des Beitrags zum Zeitpunkt[1] t dar. Dabei werden zwei Ansätze verfolgt: Im ersten Ansatz werden erkannte Named Entities mit diesem Vektor kodiert. Im zweiten Ansatz Wörter aus einem festgelegten Vokabular. Die Kodierung bleibt jedoch die Selbe, weswegen hier nur die Kodierung der Named Entities beschrieben wird, da die Kodierung der Wörter mit einem Vokabular analog erfolgt. Der Vektor $\boldsymbol{n}_t \in NE \subseteq \{0,1\}^{|NE'|}$ ist ein Bitvektor. Sei $NE' = \{ne_1, ne_2, ..., ne_{|NE'|}\}$ die Menge aller Named Entities im Modell[2]. Dann gilt: ne_i wird im Beitrag zum Zeitpunkt t genutzt, genau dann wenn $n_{ti} = 1$.

[1] Hierbei ist mit Zeitpunkt die Position des Beitrags innerhalb der Sequenz aller Beiträge eines Threads gemeint

[2] Dabei werden *Named Entities* welche nur einmal im gesamten Datensatz vorkommen nicht betrachet

Ein Beitrag B_t wird also dargestellt durch das Tupel $(\boldsymbol{u}_t, \boldsymbol{n}_t)$. Weiterhin besteht ein Thread T im Forum aus einer Menge von Beiträgen $\{B_1, B_2, ...\}$ und das Forum F selbst aus einer Menge Von Threads $\{T_1, T_2, ...\}$.
Mit diesen Definitionen werden nun die Daten vor verarbeitet. Hierfür wurde in Python eine Klasse geschrieben, welche die Daten für das RNN-Modell vorbereitet. Dazu werden die Informationen aus den Werten der Keys *author*, *thread*, *no*, *text*, *published* und *named_ entity* der Tupel in der JSON-Datei verwendet. Die Klasse erzeugt intern eine Statistik, damit beispielsweise beim erzeugen der Trainingsdaten eingestellt werden kann, wie oft ein Named Entity beziehungsweise Wort vorkommen soll oder wie viele Beiträge ein Nutzer haben muss, damit das Tupel in die Trainingsdaten gelangt.
Beim Erzeugen der Trainingsdaten wird eine Liste von Threads (wie oben definiert) erzeugt, welche aufsteigend nach dem Erstellungsdatum beziehungsweise dem *published* Eintrag des ersten Beitrags der Threads sortiert sind. Die Beiträge in den Threads sind nach dem *no* Eintrag im Datensatz aufsteigend sortiert. Außerdem werden alle Threads, welche nur aus einem Beitrag bestehen rausgefiltert, da sie keine Informationen über das zu Untersuchende bieten.
Des Weiteren bietet die Klasse Funktionen zum Kodieren/Dekodieren der One-Hot- beziehungsweise Bitvektoren in Nutzernamen beziehungsweise Liste von Named Entities (oder Wörter) für die spätere Analyse der Ergebnisse.

4.3.1 Erstellung der $\boldsymbol{n}_t$ Vektoren basierend auf Wörtern aus einem Vokabular

In diesem Abschnitt wird erläutert wie die $\boldsymbol{n}_t$ Vektoren basierend auf benutzten Wörtern erzeugt werden und wie das Vokabular ausgewählt wird.
Zunächst werden als Wörter die Komponenten des Textes der Beiträge definiert, welche durch Leerzeichen getrennt sind. Daraus resultiert, dass Satzzeichen auch zu einem Wort gehören, Beispielsweise wäre „ ist,“ ein Wort. Die Anzahl der Wörter im Datensatz beläuft sich so auf 416,856 Wörter. Wie oben beschrieben müssen die Daten nun bereinigt werden. Dazu werden zunächst einmal alle Satz- und Sonderzeichen entfernt. Da Zahlen irrelevant für die Analyse sind, werden diese auch entfernt. Danach werden Stopwords anhand einer Liste[3] entfernt und alle restlichen Wörter in den Beiträgen normalisiert: Dazu werden über den Stemmer der NLTK-Library alle Wörter auf ihre Stammform reduziert und Groß- auf Kleinbuchstaben und *ä, ö, ü* auf *ae, oe, ue* abgebildet. Nach dieser Prozedur reduziert sich die Anzahl der Wörter auf 212,378. Dies ist noch immer eine beachtliche Menge, sodass die Anzahl noch weiter Reduziert werden muss. Um den Rechenaufwand zu vermindern und Speicher zu sparen, werden die 10000 relevantesten Wörter für das Vokabular verwendet und die n_t-Vektoren analog zu den *Named Entity*-Vektoren. Durch teilweise sehr kurzen

[3] https://github.com/stopwords-iso/stopwords-de/blob/master/stopwords-de.json

Beiträge, gewichtet das IDF-Maß sehr seltene Wörter zu stark, sodass es nicht verwendet werden kann. Daher werden nur Wörter ins Vokabular genommen, welche mindestens 50 mal und höchstens 1000 verwendet werden. Damit wird die Zahl der verwendeten Wörter auf 15477 reduziert.

Kapitel 5

Methode der Versuchsdurchführung

5.1 Aufbau des Modells

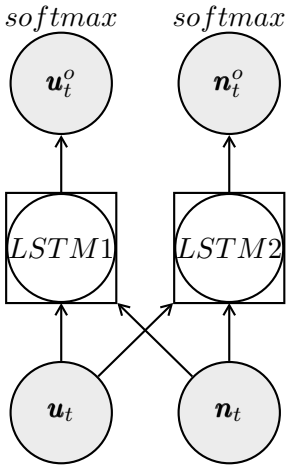

Abbildung 5.1:
Visualisierung des verwendeten Modells

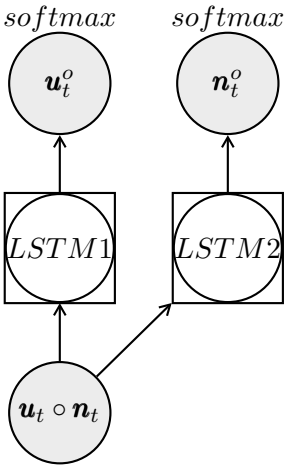

Abbildung 5.2:
Äquivalentes Modells, bei der Eingaben konkateniert werden

In der Versuchsdurchführung wird das in Abbildung 5.1 zu sehende Modell verwendet. Es besteht aus zwei LSTM-Zellen, welche jeweils als Eingabe den One-Hot kodierten Nutzervektor und den Inhaltsbitvektor erhalten. Dabei soll LSTM1 den auf Nutzer $\boldsymbol{u}_t$ mit Inhalt $\boldsymbol{n}_t$ antwortenden Nutzer $\boldsymbol{u}_t^o$ und LSTM2 analog dazu den Inhalt $\boldsymbol{n}_t^o$ der Antwort vorhersagen. Diese Vorhersage wird über einen *softmax*-Layer an der Ausgabe getroffen. Wie in Kapitel 4 beschrieben werden zwei Modelle trainiert. Die Modelle unterscheiden sich darin, dass die Inhaltsevektoren des einen Modells im Beitrag vorkommende *Named Entities* kodieren und das andere Modell Wörter aus einem ausgewählten Vokabular.
Dabei sei anzumerken, dass es nicht dabei geht, ein möglichst akkurates Modell zu erstellen. Dafür ist das Modell für eine derartig Komplexe Aufgabenstellung zu simpel gestaltet

und es wird keine hohe Genauigkeit erwartet. Stattdessen wird erwartet, dass sich trotz niedriger Genauigkeit durch das Training Zusammenhänge im Modell abzeichnen, welche dann durch die Gradienten analysiert werden können. Deswegen reicht es, dass das Modell im Training besser wird, was ein Indiz dafür ist, dass einige Zusammenhänge in den Daten erkannt wurden.
Das Modell wird in Python mit der Bibliothek *Keras*[1] implementiert. Da es in Keras nicht ohne weiteres möglich ist das Modell in Abbildung 5.1 nach zu bauen, weil mit der Bibliothek einer LSTM Zelle nur ein Vektor als Eingabe gegeben werden kann, wird das alternative Modell aus Abbildung 5.2 verwendet. Dabei werden die Eingabevektoren konkateniert, das heißt $\boldsymbol{u}_t \circ \boldsymbol{n}_t = (u_{t1}, ..., u_{t|U|}, n_{t1}, ..., n_{t|NE|})$. Dass diese Modelle äquivalent sind, wird nun im Folgenden gezeigt.
Das Modell wird mit *RMSProp*[2] optimiert. Als *Loss*-Funktion wird categorical_crossentropy"verwendet, da er für *softmax*-Ausgaben empfohlen wird.

Aquivalenz der abgebildeten Modelle

In dem Modell in Abbildung 5.1 fließen die Eingabevektoren über den Aktivierungsvektor des *Gates* wie folgt in das Modell ein:

$$f_t = \sigma(\boldsymbol{V}\boldsymbol{u}_t + \boldsymbol{W}\boldsymbol{n}_t + ...) \tag{5.1}$$

Dabei ist f_t ein e-dimensionaler Vektor. Es gilt $dim(\boldsymbol{u}_t) = |U|$ und $dim(\boldsymbol{n}_t) = |NE|$, das heißt V ist eine $e \times |U|$ und W eine $e \times |NE|$ Gewichtsmatrix. In dem Modell in Abbildung 5.2 wäre der Einfluss der Eingabe $(\boldsymbol{u}_t \circ \boldsymbol{n}_t)$ wie folgt:

$$f_t = \sigma(\boldsymbol{K}(\boldsymbol{u}_t \circ \boldsymbol{n}_t) + ...) \tag{5.2}$$

Die Gewichtsmatrix $\boldsymbol{K}$ muss die Dimension $e \times (|U| + |NE|)$ haben. Im Folgenden wird zwecks Vereinfachung der Darstellung der Index t ausgelassen, da er irrelevant für das zu zeigende ist. Die Modelle sind also gleich wenn gilt:

$$\boldsymbol{K}(\boldsymbol{u} \circ \boldsymbol{n}) = \boldsymbol{V}\boldsymbol{u} + \boldsymbol{W}\boldsymbol{n} \tag{5.3}$$

Nun wird gezeigt, dass diese Gleichheit gilt, wenn $\boldsymbol{K}$ wie folgt definiert ist:

$$K_{i,j} = \begin{cases} V_{i,j}, \text{ falls } j \leq |U| \\ W_{i,j-|U|}, \text{ falls } j \geq |U| + 1 \end{cases} \tag{5.4}$$

Dies kommt einem Konkatenieren entlang der Spalten der Matrizen $\boldsymbol{V}$ und $\boldsymbol{W}$ gleich. Wird nun $\boldsymbol{K}(\boldsymbol{u} \circ \boldsymbol{n})$ ausgerechnet, ergibt sich

$$\boldsymbol{K}(\boldsymbol{u} \circ \boldsymbol{n}) = (\sum_{j=1}^{|U|+|NE|} K_{1,j}(\boldsymbol{u} \circ \boldsymbol{n})_j \ , ... \ , \sum_{j=1}^{|U|+|NE|} K_{e,j}(\boldsymbol{u} \circ \boldsymbol{n})_j)^\top$$

[1] Siehe https://www.keras.io/
[2] Siehe https://keras.io/optimizers/#rmsprop

Betrachten wir nun die i-te Zeile des Ergebnisses und formen sie etwas um, dann ergibt sich mittels Definition von $\boldsymbol{K}$ und der Konkatenation der Vektoren

$$\sum_{j=1}^{|U|+|NE|} K_{i,j}(\boldsymbol{u} \circ \boldsymbol{n})_j = \sum_{j=1}^{|U|} K_{i,j}(\boldsymbol{u} \circ \boldsymbol{n})_j + \sum_{j=|U|+1}^{|U|+|NE|} K_{i,j}(\boldsymbol{u} \circ \boldsymbol{n})_j = \sum_{j=1}^{|U|} V_{i,j} u_j + \sum_{j=1}^{|NE|} W_{i,j} n_j$$

Daraus folgt unmittelbar die Gültigkeit von Gleichung 5.3 und somit sind beide Modelle äquivalent.

Training der Modelle

Beim Trainieren des Modells muss aufgepasst werden, dass das erlernte Modell angemessen generalisiert. Zur Validierung des Modells gibt es verschiedene Methoden. Eine beliebte Methode ist die k-Fold Kreuzvalidierung [13]. Dabei werden die Vorliegenden Daten in k gleich große Teile aufgeteilt und es werden k Trainings- und Testdatenpaare gebildet. In diesen Paaren wird eines der k Teile als Testdatensatz festgelegt und die Restlichen $k-1$ als Trainingsdaten verwendet. Damit wird die Unabhängigkeit der Performanz des Modells von den Trainings- beziehungsweise Testdaten überprüft. Der Nachteil hierbei ist, dass k Modelle Trainiert werden müssen. Da das Trainieren von LSTMs Zeitaufwändig ist, wird in diesem Versuch auf eine Kreuzvalidierung verzichtet. Stattdessen wird eine Einfache Teilung des Datensatzes durchgeführt, indem 80% der Daten als Trainingsdaten und 20% als Testdaten verwendet werden.
Die Eingabe in das Modell erfolgt über Sequenzen von Vier Beiträgen, da die durchschnittliche Länge bei ca. 6 liegt, dieser Wert jedoch von einigen Ausreißern nach oben hin beeinflusst wurde. Dabei werden Threads, welche weniger als vier Beiträge haben nicht mit Nullvektoren aufgefüllt, da ohnehin viele Nullvektoren, anhand der Tatsache, dass die Hälfte der Beiträge keine *Named Entities* besitzen, in den Trainingsdaten vorliegen. Stattdessen werden alle Threads, welche Zeitlich sortiert sind, konkateniert und diese Sequenz von Beiträgen in Teilsequenzen der Länge vier aufgeteilt.

5.2 Bestimmung autoritärer User

In diesem Abschnitt wird erläutert, wie das erlernte Modell genutzt wird, um autoritäre Nutzer zu erkennen.

5.2.1 Ansatz basierend auf Gradienten

Dieser Ansatz basiert auf der Arbeit [7] von Yotam Hechtlinger, welcher einen Ansatz zum Verständnis von Modellen, welche wie *blackbox*-Funktionen erscheinen, vorstellt. Im Folgenden wird kurz das Beispiel an der linearen Regression aus der Arbeit erklärt, dann das allgemeine Verfahren erläutert und am Ende auf das Modell in der Bachelorarbeit übertragen.

Grundkonzept anhand linearer Regression

Sei $\hat{f} : X \to Y$ ein Vorhersagemodell einer linearen Regression, wobei X die Eingabedomäne und Y der Ausgabebereich ist. Dann hat dieses Modell die Form $\hat{f}(\boldsymbol{x}) = \hat{\boldsymbol{\beta}}^T \boldsymbol{x}$, wobei $\hat{\boldsymbol{\beta}}$ die erlernten Modellparameter sind. Aufgrund des Skalarprodukts besteht eine 1:1 Beziehung zwischen den Komponenten beider Vektoren. Es ist leicht zu erkennen, dass wenn $\hat{\beta}_k$ einen Wert nahe Null annimmt, der Wert von x_k einen verschwindend geringen Einfluss auf die Ausgabe des Modells hat. Im Umkehrschluss hat der Wert von x_k einen großen Einfluss auf die Ausgabe, wenn $\hat{\beta}_k$ groß ist. Es kann also aufgrund der Modellparameter gesagt werden, wie groß der Einfluss der einzelnen Komponenten der Eingabe auf die Ausgabe ist. In komplexeren Modellen, wie beispielsweise bei neuronalen Netzen, kann anhand der Modellparameter eine solche Aussage nicht direkt getroffen werden, da mehrere Rechenschritte benötigt werden, um die Ausgabe zu berechnen. Trotzdem kann der Ansatz der Interpretation der Modellparameter der linearen Regression auf komplexere Modelle übertragen werden. Es gilt für das einfache lineare Modell

$$\hat{\beta}_k = \frac{\partial \hat{f}}{\partial x_k}.$$

Das heißt, die obigen Aussagen bezüglich Einflusses anhand von $\hat{\beta}_k$ gelten für die partiellen Ableitungen. Somit heißt es, wenn $\frac{\partial \hat{f}}{\partial x_k}$ einen Wert nahe Null annimmt, der Wert von x_k einen verschwindend geringen Einfluss auf die Ausgabe des Modells hat und im Umkehrschluss einen großen Einfluss, falls dessen Wert groß ist. Somit kann dieser Ansatz auf komplexere Modelle übertragen werden, indem statt Modellparameter, partielle Ableitungen und somit Gradienten betrachtet werden. Die Interpretation des Modells wird von der Analyse der Parameter auf die Analyse des Gradienten verschoben.

Allgemeines Konzept

Wie oben Beschrieben werden zum Verständnis des Modells Gradienten untersucht. Es soll herausgefunden werden, welche Komponenten der Eingabe, welche Komponenten der Ausgabe, wie stark beeinflusst. Sei $\hat{f} : X \to Y$ nun ein beliebiges Vorhersagemodell, wobei der Definitions- und Wertebereich mehrdimensional sein können. Wir nehmen im Folgenden an, dass beide mehrdimensional sind und $|X| = p$ und $|Y| = q$ gilt. Das heißt $\boldsymbol{x} \in X$ sind Tupel der Form $(x_1, ..., x_p)$ und $\boldsymbol{y} \in Y$ sind Tupel der Form $(y_1, ..., y_q)$. Des Weiteren sei $\hat{f}(x) = \boldsymbol{y} = (y_1, ..., y_q) = (\hat{f}_1(x), ..., \hat{f}_q(x))$. Um den Einfluss jeder Komponente der Eingabe $\boldsymbol{x}$ auf eine Komponente y_j der Ausgabe zu erfahren, muss also für jedes $\hat{f}_j$ der Gradient $\nabla \hat{f}_j(x)$ berechnet werden. Wird dies für jedes j getan, ergibt sich die Jacobi-Matrix

$$J_{\hat{f}}(x) = (\frac{\partial \hat{f}_j}{\partial x_k}(x))_{j=1,...,q;k=1,...,p} \tag{5.5}$$

dessen Zeile j den Gradienten $\nabla \hat{f}_j(x)$ (transponiert) darstellt. Nun wird für alle $x \in X_{Test}$ die Jacobi-Matrix $J_{\hat{f}}(x)$ berechnet und darauf basierend die mittlere Jacobi-Matrix $\widetilde{J}_{\hat{f}}$ gebildet:

$$\widetilde{J}_{\hat{f}} = \frac{\sum_{x \in X_{Test}} J_{\hat{f}}(x)}{|X_{Test}|} = (\frac{\sum_{x \in X_{Test}} \frac{\partial \hat{f}_j}{\partial x_k}(x)}{|X_{Test}|})_{j=1,...,q;k=1,...,p} \tag{5.6}$$

Dabei ist ein Eintrag dieser Matrix eine mittlere partielle Ableitung. Anhand der $q \times p$ Jacobi-Matrix $\widetilde{J}_{\hat{f}}$ können nun die Einflüsse der x_k auf ein y_j der Vorhersage interpretiert werden. Dazu muss nur der j,k-te Eintrag $(\widetilde{J}_{\hat{f}})_{j,k}$ der Matrix angeschaut werden: ist $(\widetilde{J}_{\hat{f}})_{j,k} \approx 0$, so hat x_k einen verschwindend geringen Einfluss auf y_j, ist $(\widetilde{J}_{\hat{f}})_{j,k} >> 0$, so ist der Einfluss von x_k auf die Vorhersage von y_j sehr groß.

Interpretation des Modells dieser Arbeit

Das oben beschriebene Konzept wird nun auf das Modell in Sektion 5.1 angewandt. Dazu wird das Vorhersagemodell in zwei Funktionen geteilt, da diese prinzipiell unabhängig voneinander sind.
Die erste Teilfunktion ist die Funktion $\hat{f}_u$, welche den Nutzer in der Antwort vorhersagen soll:

$$\hat{f}_u : U \times NE \to U \tag{5.7}$$

Die zweite Teilfunktion ist die Funktion $\hat{f_{ne}}$, welche die in der Antwort genutzten *Named Entities* vorhersagen soll:

$$\hat{f}_{ne} : U \times NE \to NE \tag{5.8}$$

Für beide Funktionen wird eine mittlere Jacobi-Matrix erstellt. Dabei erfolgt die Berechnung der partiellen Ableitungen numerisch für ein sehr kleines h über die Formel

$$\frac{\partial \hat{f}}{\partial x_k} \approx \frac{\hat{f}(x_1, ..., x_k + h, ..., x_p) - \hat{f}(x_1, ..., x_k, ..., x_p)}{h}. \tag{5.9}$$

Die Matrix $\widetilde{J}_{\hat{f}_u}$ ist dabei so zu interpretieren, dass seine Einträge aussagen, welche Nutzer beziehungsweise welche *Named Entities* einen Hohen Einfluss darauf haben, welcher Nutzer antworten wird. Analog dazu ist die Interpretation der Matrix $\widetilde{J}_{\hat{f}_{ne}}$, dass dessen Einträge Auskunft darüber geben, welche Nutzer beziehungsweise welche *Named Entities* einen Hohen Einfluss darauf haben, welche *Named Entities* in der Antwort benutzt werden. Da die Eingabevektoren, wie in Kapitel 5.1 beschrieben werden, konkateniert werden, ergibt sich für $\hat{f}_{ne}$:

$$\widetilde{J}_{\hat{f}_{ne}} = \begin{pmatrix} \frac{\partial \hat{f}_{ne1}}{\partial u_1} & \cdots & \frac{\partial \hat{f}_{ne1}}{\partial u_{|U|}} & \frac{\partial \hat{f}_{ne1}}{\partial ne_1} & \cdots & \frac{\partial \hat{f}_{ne1}}{\partial ne_{|NE|}} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \hat{f}_{ne|NE|}}{\partial u_1} & \cdots & \frac{\partial \hat{f}_{ne|NE|}}{\partial u_{|U|}} & \frac{\partial \hat{f}_{ne|NE|}}{\partial ne_1} & \cdots & \frac{\partial \hat{f}_{ne|NE|}}{\partial ne_{|NE|}} \end{pmatrix} \tag{5.10}$$

und für $\hat{f}_u$ ergibt sich:

$$\widetilde{J}_{\hat{f}_u} = \begin{pmatrix} \frac{\partial \hat{f}_{u1}}{\partial u_1} & \cdots & \frac{\partial \hat{f}_{u1}}{\partial u_{|U|}} & \frac{\partial \hat{f}_{u1}}{\partial ne_1} & \cdots & \frac{\partial \hat{f}_{u1}}{\partial ne_{|NE|}} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \hat{f}_{u|U|}}{\partial u_1} & \cdots & \frac{\partial \hat{f}_{u|U|}}{\partial u_{|U|}} & \frac{\partial \hat{f}_{u|U|}}{\partial ne_1} & \cdots & \frac{\partial \hat{f}_{u|U|}}{\partial ne_{|NE|}} \end{pmatrix} \tag{5.11}$$

Es ist anzumerken, dass der Inhalt der Matrizen 5.10 und 5.11 die mittleren partiellen Ableitungen über der Testmenge sind, wie in Gleichung 5.6 zu sehen. Aus Darstellungsgründen sind in den genannten Matrizen und bis zum Ende des Kapitels normale partielle Ableitungen abgebildet, gemeint sind jedoch die gemittelten.

Da die Komponenten der Vektoren $u \in U$ aufgrund ihrer Kodierung mit jeweils einem Nutzer und die Komponenten der Vekoren $n \in NE$ mit einem *Named Entity* korrespondieren, können die Einträge nun als Einflussfaktoren von Nutzern beziehungsweise *Named Entity* interpretiert werden.

In der Matrix $\widetilde{J}_{\hat{f}_{NE}}$ existiert für jeden Nutzer eine Spalte, dessen Einträge den Einfluss des Nutzers auf die *Named Entities* der Antworten wieder spiegeln. Dies sind die Spalten mit dem Index $i \leq |U|$. Das heißt $(\widetilde{J}_{\hat{f}_{NE}})_{:,i}$ ist ein Spaltenvektor, dessen Einträge den Einfluss des Nutzers, kodiert durch u_i, auf die in der Antwort genutzten *Named Entities* beinhaltet. Im Folgenden wird dieser Vektor User-Named-Entity-Einflussvektor $\boldsymbol{e}_i^{u-ne}$ genannt

$$\boldsymbol{e}_i^{u-ne} = (\widetilde{J}_{\hat{f}_{ne}})_{:,i} = (\frac{\partial \hat{f}_{ne1}}{\partial u_i}, ..., \frac{\partial \hat{f}_{ne|NE|}}{\partial u_i})^\top \tag{5.12}$$

Analog dazu werden aus den Spalten von $\widetilde{J}_{\hat{f}_u}$ der Nutzer-zu-Nutzer-Einflussvektor

$$\boldsymbol{e}_i^{u-u} = (\widetilde{J}_{\hat{f}_u})_{:,i} = (\frac{\partial \hat{f}_{u1}}{\partial u_i}, ..., \frac{\partial \hat{f}_{u|U|}}{\partial u_i})^\top \tag{5.13}$$

und Named-Entity-zu-Nutzer-Einflussvektor

$$\boldsymbol{e}_i^{ne-u} = (\widetilde{J}_{\hat{f}_u})_{:,i} = (\frac{\partial \hat{f}_{u1}}{\partial ne_i}, ..., \frac{\partial \hat{f}_{u|U|}}{\partial ne_i})^\top \tag{5.14}$$

definiert. Anhand der Annahme, dass autoritäre Nutzer einen großen Einfluss auf die in den Antworten genutzten Wörter haben, welche in Kapitel 2 getroffen wurde, können nun die Nutzer anhand der euklidschen Norm von $\boldsymbol{e}_i^{u-ne}$ eingeordnet werden. Denn Nutzer, dessen Einflussvektor $\boldsymbol{e}_i^{u-ne}$ eine hohe Euklidsche Norm besitzt, welches der Länge des Vektors entspricht, haben in hohem Maße große Einflussfaktoren und somit einen hohen Einfluss auf die Named-Entities in der Antwort. Das heißt je länger der Vektor, desto größer der Einfluss.
Darüber hinaus können Nutzer anhand der Ähnlichkeit ihrer Einflussvektoren mit Clusteralgorithmen gruppiert werden. So könnten Nutzer mit ähnlichem Einfluss gefunden werden.
Eine solche Einordnung und Gruppierung könnte auch mit $\boldsymbol{e}_i^{u-u}$ getan werden, sodass Nutzer, die Stark beeinflussen, wer als nächstes antworten wird, gefunden werden. Hier ist zu erwarten, dass Nutzer mit einer hohen Anzahl an Beiträgen hoch eingeordnet würden.
Interessant ist dabei auch, dass mit der Gruppierung durch $\boldsymbol{e}_i^{ne-u}$ herausgefunden werden kann, auf welche Themen jener Nutzer am häufigsten antwortet, da dieser Vektor in seiner Komponente j angibt, in wie weit Nutzer u_j durch dieses ne_i zu einer Antwort verleitet wird. So könnten die Interessen der Nutzer und ihre bevorzugten Themen gefunden werden.

5.2.2 Ansatz basierend auf mittlerer Änderung des Zustandsvektors

Ein weiterer Ansatz autoritäre Nutzer zu finden ist Nutzer anhand ihres Einflusses auf den Zustandsvektor einzuordnen. Dies wird gemacht, da angenommen wird, dass Nutzer, welche allgemein einen großen Einfluss auf die Modellvorhersage haben, auch eine gewisse Autorität besitzen müssen. Da der Zustandsvektor des RNN beziehungsweise LSTM von großer Bedeutung für die Modellvorhersage ist, kann gesagt werden, dass Nutzer eine Änderung des Zustandsvektors bewirken, einen großen Einfluss auf die Modellvorhersage haben. Der zu berechnende Wert ist $\widetilde{h}(u_t)$ und wird für jeden Nutzer berechnet

$$\widetilde{h}(u) = \frac{1}{n} \sum_{\forall(u_t,n_t)\in B; u=u_t} ||h_t - h_{t-1}||^2 \tag{5.15}$$

Das n in der Formel steht für die Anzahl der Beiträge des Nutzer, welche im Datensatz vorhanden sind und das B für die Menge aller Beiträge. Die Summe ergibt sich aus der Differenz des Zustandsvektors h_t zum Zeitpunkt t, an dem der User u den Beitrag veröffentlicht hat und dem Zustand h_{t-1}, welcher ein Beitrag zuvor, zu dem Zeitpunkt $t-1$, galt. Somit lässt sich ermitteln, welche User einen hohen Einfluss auf die Zustandsvektoren des RNN haben, welche für die Vorhersage des Modells relevant sind. Eine große Änderung des Zustandsvektors ist ein Indikator für einen großen Einfluss im Modell und folglich auch im Forum selbst. Nach dem Berechnen der mittleren Änderungen des Zustandsvektors für jeden User können diese danach eingeordnet werden.

Kapitel 6

Versuchsdurchführung

In diesem Kapitel wird der Versuch gemäß der Beschreibung im vorherigen Kapitel durchgeführt.

6.1 Versuch basierend auf Named-Entities

In dem Versuch basierend auf *Named Entities* wurde die Dimension der Zustandsvektoren der LSTMs auf 256 gesetzt. Der Nutzervektor u_t hat 3787 und der *Named Entities*-Vektor n_t hat 6051 Dimensionen.
Das Modell wurde zunächst mit einer Batchgröße[1] von 1 trainiert, jedoch verbesserte sich das Modell nicht, wie der Abbildung 6.1 zu entnehmen ist. Daher wurde die Batchgröße

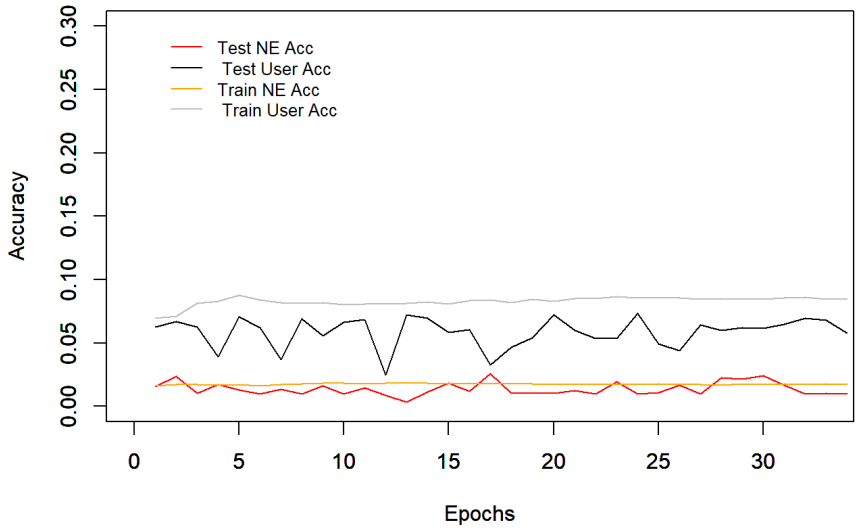

Abbildung 6.1: Die Genauigkeit des Modells während des Trainings

während des Trainings 32 erhöht, welches die Standardgröße von Keras ist. In Abbildung 6.2 ist die Entwicklung der Genauigkeit des trainierten Modells mit Batchgröße 32 pro

[1] Die Batchgröße gibt an, nach wie vielen Samples ein Gradientenupdate durchgeführt werden soll

Epoche[2] zu sehen. Zu jeder Epoche wurden die Modellparameter zwischengespeichert um so das genauste Modell heraus selektieren zu können.

Die graue Linie stellt die Genauigkeit der Vorhersagen über den antwortenden Nutzer über

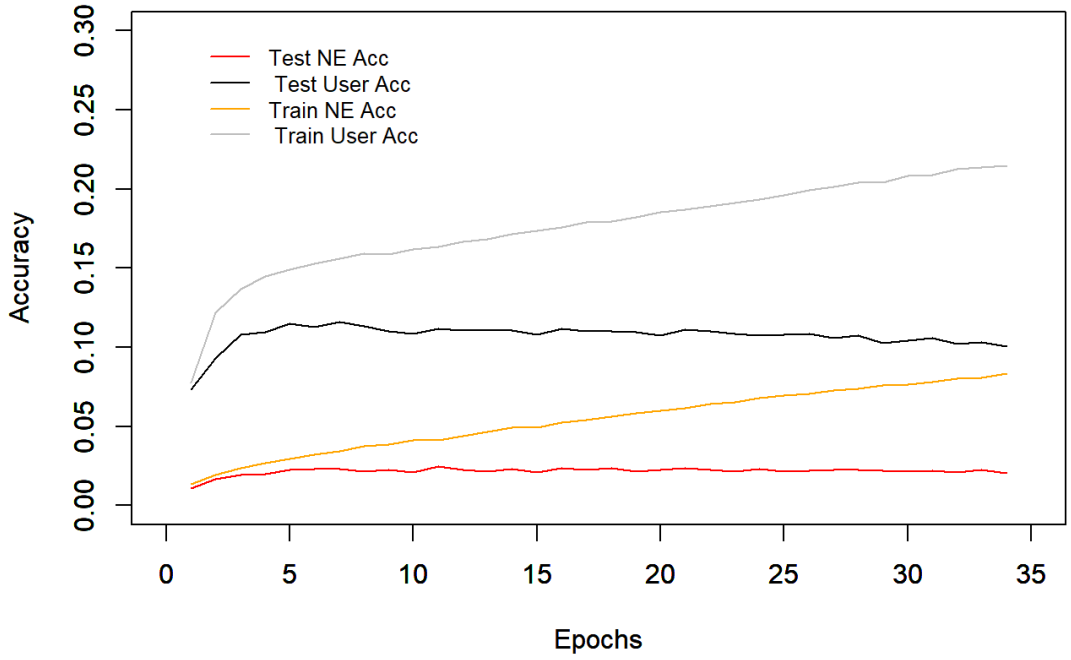

Abbildung 6.2: Die Genauigkeit des Modells während des Trainings

der Trainingsmenge dar, die schwarze Linie hingegen die Genauigkeit der Vorhersagen über den antwortenden Nutzer über der Testmenge. Die rote und orange Linie sind die Genauigkeiten über Test- und Trainingsmenge über die vorhergesagten *Named Entities*. Es ist zu sehen, dass ab circa der siebten Epoche die Genauigkeit über den Testdaten sinkt, die über den Trainingsdaten jedoch steigt. Dies ist ein Indikator für ein Overfitting, das heißt, das Modell lernt nur noch die Trainingsdaten auswendig und wird somit schlechter darin, akkurate vorhersagen über neue Beobachtungen zu treffen. Daher wurden zur Analyse die Parameter zu Epoche 7 Benutzt.

6.1.1 Ergebnisse und Auswertung: Gradienten

Zur Analyse wurde die euklidsche Norm der Einflussvektoren für alle Nutzer mit mehr als 9 Beiträgen berechnet und die Nutzer absteigend der Norm ihres Einflussvektors sortiert. Insgesamt wurden dabei 1385 Nutzer betrachtet.

[2]Mit Epoche ist eine komplette Iteration über den Testdaten gemeint

Nutzer Ranking anhand der Länge der Einflussvektoren e_i^{u-u}

Rang	Nutzername	$\|\|e_i^{u-u}\|\|_2$	Anzahl der Beiträge
1	not registered	11.005579	30
2	Mohammed Isa	0.7705352	8577
3	Denker	0.7453314	1310
4	BodenDerEhre2011	0.6535999	832
5	Az-Zubair	0.63272846	2757
6	al-shafino	0.57638544	237
7	AllahuAkbar	0.57502455	1068
8	jundu-l-islam87	0.5407472	2257
9	al-Hanbali	0.52628094	879
10	AnstrengungFiSebilillah	0.48395482	1175
11	Al-Balqani	0.48006767	1182
12	JihaduNafs	0.466549	214
13	Al-Faris	0.46609265	1733
14	Ibn Al-Jnoub	0.46064004	240
15	HarunRashid	0.45907095	600

Tabelle 6.1: Nutzer mit der größten euklidschen Norm des User-zu-User-Einflussvektors

In Tabelle 6.1 sind die 15 Nutzer mit der größten euklidschen Norm von e_i^{u-u} zu sehen. Es ist gut zu sehen, dass Nutzer mit einer relativ hohen Anzahl an Beiträgen weit oben in der Tabelle eingeordnet sind. Wird die gesamte Tabelle betrachtet, ist eine gewisse Korrelation zwischen der Anzahl der Beiträge und der Höhe des Einflusses[3] zu erkennen. (Siehe dazu Abbildung 6.3.) Um viel Einfluss in einem Forum zu haben, muss ein Nutzer präsent sein. Daher ist es plausibel, dass Autoritäre Nutzer zu jenen gehören, die viele Beiträge schreiben. Der hohe Wert des Einflusses könnte auch dadurch zu erklären sein, dass Nutzer mit vielen Beiträgen häufiger als jene mit wenigen Beiträgen in den Trainingsdaten repräsentiert sind und somit dadurch bedingt einen höheren Einfluss auf das Modell nehmen.

[3] mit Einfluss ist hier die Berechnete Norm $||e_i^{u-u}||_2$ gemeint

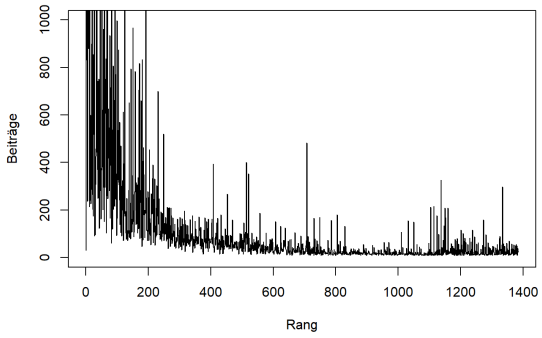

Abbildung 6.3: Anzahl der Beiträge der Nutzer sortiert nach ihrem $||e_i^{u-u}||_2$-Rang

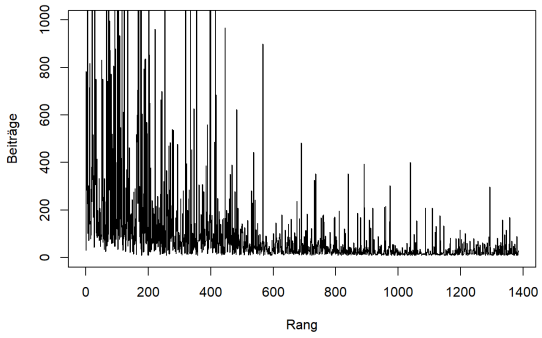

Abbildung 6.4: Anzahl der Beiträge der Nutzer sortiert nach ihrem $||e_i^{u-ne}||_2$-Rang

Nutzer Ranking anhand der Länge der Einflussvektoren e_i^{u-ne}

Rang	Nutzername	$\|\|e_i^{u-ne}\|\|_2$	Anzahl der Beiträge
1	not registered	9.4629	30
2	Qwer	0.33207536	784
3	Vitoria	0.3003301	284
4	Muslim96	0.29712105	325
5	Yasira	0.2819647	226
6	jundu-l-islam87	0.26533547	2257
7	tunismuslima17	0.26408496	102
8	ibrahimkhalil	0.25994548	204
9	ummyassine	0.25561762	303
10	prince	0.2553218	71
11	Abu Z Projekt	0.25345206	250
12	Bekim	0.25194472	716
13	Islamu_Dini	0.2517753	818
14	Abdullah Tarkan	0.23404108	141
15	Emina	0.23184797	153

Tabelle 6.2: Nutzer mit der größten euklidschen Norm des User-zu-NE-Einflussvektors

In Tabelle 6.2 sind die 15 Nutzer mit der größten euklidschen Norm von e_i^{u-ne} zu sehen. Auch hier ist eine Korrelation zwischen der Anzahl der Beiträge und des berechneten Einflusses des Nutzers zu sehen.

Rang	Nutzername	$\widetilde{h}^{user}(u)$	Anzahl der Beiträge
1	Al Qadi	32.92013234105603	141
2	Abu Allaal al-Sunni	32.006202697753906	2
3	Noori	31.88672637939453	2
4	Sirat	26.58215395609538	4
5	Abu Muhammad Amin	26.19055938720703	165
6	Uthman	25.03725395202637	400
7	zichan	23.72148388226827	44
8	Abu Ilias	22.48976162501744	471
9	hassibullah	21.895145416259766	2
10	Abu_Taymiyyah	21.515417203511277	935
11	qa3qa3 ibnAmr at Tamimi	20.97087860107422	3
12	Popito	20.80142593383789	2
13	jundu-l-islam87	20.133440888053933	2257
14	Abu Wazir	20.119913864135743	6
15	Rafidha Shredder	20.068714601652964	44

Tabelle 6.3: Nutzer mit der größten mittleren Änderung des Zustandsvektors des Nutzer-LSTMS

Rang	Nutzername	$\widetilde{h}^{NE}(u)$	Anzahl der Beiträge
1	al-Mardini	17.681190490722656	6
2	Al Qadi	17.300314656619367	141
3	Abu Allaal al-Sunni	16.865861892700195	2
4	Abu Juhayna	16.76406478881836	2
5	zichan	15.247817277908325	44
6	Isabella223	13.416586875915527	2
7	qa3qa3 ibnAmr at Tamimi	13.059540748596191	3
8	Devon100	12.942514419555664	2
9	jundu-l-islam87	12.838975772051745	2257
10	muslime	12.660674571990967	3
11	berliner	12.55994110107422	8
12	Abu Muhammad Amin	12.4214874903361	165
13	Popito	12.344392776489258	2
14	Khalid1	12.278617858886719	2
15	DieMitte	12.054265975952148	2

Tabelle 6.4: Nutzer mit der größten mittleren Änderung des Zustandsvektors des Named Entity-LSTMS

6.1.2 Ergebnisse und Auswertung: Zustandsvektor

Die mittleren Zustandsvektoren wurden für alle Nutzer berechnet, welche Beiträge in der Testmenge hatten. Insgesamt waren es 1025 Nutzer. In Tabelle 6.6 sind Nutzer zu sehen die im Schnitt die höchste Änderung des Zustandsvektors des LSTM verursachen, welche den nächsten Nutzer vorhersagt.
Es ist zu sehen, dass Nutzer mit sehr wenigen Beiträgen sehr hoch eingeordnet sind. Dies sollte jedoch nicht sein, da ein Nutzer, der Kaum Beiträge hat nur wenig Einfluss auf das Modell haben sollte. Interessant ist hier jedoch, dass es Nutzer wie *qa3qa3 ibnAmr at Tamimi* mit wenigen Beiträgen gibt, die Anhand von beiden mittleren Zustandsvektoren relativ hoch Eingeordnet wurden.

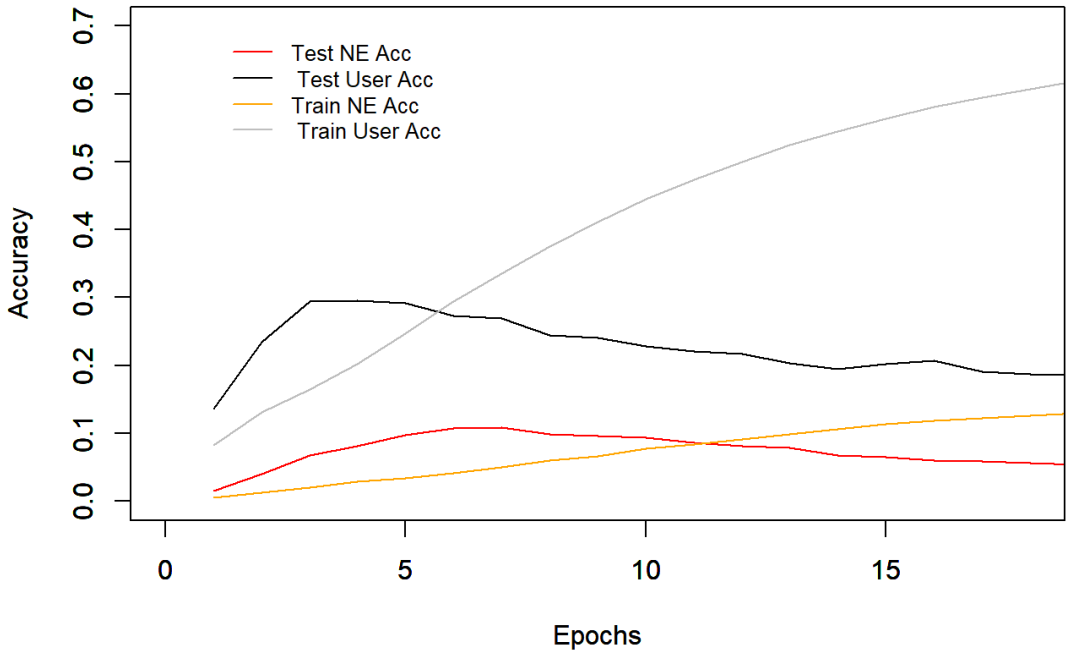

Abbildung 6.5: Die Genauigkeit des Vokabular basierten Modells während des Trainings

6.2 Versuch basierend auf Vokabular

In dem Versuch basierend auf einem Wörtern aus dem Text der Beiträge, wurde ein Vokabular erstellt, welches 15477 Wörter umfasst. Die Dimension der Zustandsvektoren der LSTMs wurde auf 512 gesetzt, die u_t Nutzervektoren waren wieder 3787-dimensional und die auf Wörtern basierenden n_t Vektoren 15477-dimensional. Die Batchgröße wurde wieder auf 32 gesetzt. Dieses Modell war mit einer Genauigkeit der Nutzervorhersage von 0.29 und knapp 0.1 der Wörtervorhersage auf der Testmenge mehr als doppelt so genau wie das auf Named Entities basierte Modell. Auch hier kann gesehen werden, dass sich das Modell ab der Vierten bis Fünften Epoche überanpasst und die Vorhersagegenauigkeit auf der Trainingsmenge bis zu 0.6 steigt, jedoch auf der Testmenge absinkt. Daher wurde zur Analyse das Modell zu Epoche 5 verwendet.

6.2.1 Ergebnisse und Auswertung: Gradienten

Diese Ergebnisse mussten aus zeitlichen Gründen ausgelassen werden. Das berechnen der Gradienten dauert zu Lange. Im Schnitt dauerte die Berechnung eines Einflussvektors 30s für einen Nutzer.

Rang	Nutzername	$\widetilde{h}^{user}(u)$	Anzahl der Beiträge
1	qa3qa3 ibnAmr at Tamimi	138.2243194580078	3
2	Takhliis	115.73455301920573	4
3	Al Qadi	112.09281355759194	141
4	Abu Allaal al-Sunni	111.2612533569336	2
5	Tayfun2333	110.03595733642578	2
6	hassibullah	104.69377136230469	2
7	Abu Juhayna	103.92947387695312	2
8	Abu Muhammad Amin	93.85609181722005	165
9	Abu Abdillah	89.69837951660156	2
10	Demeter	87.95371055603027	5
11	Abdulrezak	84.16537857055664	7
12	Sirat	83.02156575520833	4
13	AbuIman	81.1452865600586	5
14	Abu Noura ar-Rusi	80.6770248413086	5
15	zichan	79.65728721618652	44

Tabelle 6.5: Nutzer mit der größten mittleren Änderung des Zustandsvektors des Nutzer-LSTMS

Rang	Nutzername	$\widetilde{h}^{NE}(u)$	Anzahl der Beiträge
1	qa3qa3 ibnAmr at Tamimi	113.83101654052734	3
2	Takhliis	94.47765350341797	4
3	az2012	94.3515396118164	2
4	AbuIman	91.80194091796875	5
5	Tayfun2333	84.62037658691406	2
6	Al Qadi	80.02340290464204	141
7	Abu Abdillah	78.83657836914062	2
8	hassibullah	78.34059143066406	2
9	Abu Muhammad Amin	73.49404398600261	165
10	Abu Allaal al-Sunni	73.2818603515625	2
11	Isabella223	72.92855072021484	2
12	Shiri	72.86347198486328	3
13	Abu Juhayna	72.37963104248047	2
14	Demeter	71.63213729858398	5
15	berliner	70.98561935424804	8

Tabelle 6.6: Nutzer mit der größten mittleren Änderung des Zustandsvektors des Nutzer-LSTMS

6.2.2 Ergebnisse und Auswertung: Zustandsvektor

Die Ergebnisse sind ähnlich die des Modells, welches auf Named Entities basiert. Auch hier sind sehr viele Nutzer mit sehr wenigen Beiträgen hoch eingeordnet. Interessanterweise gibt es hier auch Nutzer, welche anhand beider mittleren Änderungen des Zustandsvektors hoch eingeordnet sind. Abgesehen davon gibt es Nutzer, die sowohl im Named-Entity-Modell basierend auf dem Zustandsvektor hoch eingeordnet sind, als auch im Vokabular-Modell.

6.3 Vergleich beider Versuche

Beim Training beider Vorhersagemodelle ist aufgefallen, dass das Modell, welches ein Vokabular benutzt Akkurater ist, als das NE-Basierte Modell.(Siehe Abb. 6.5 und 6.2) Im Vergleich ist er jedoch aufwendiger, da ein passendes Vokabular ausgewählt werden muss und die Wörter normalisiert werden müssen. Der Grund für die schlechte Genauigkeit des NE-Modells könnte darin liegen, dass knapp die Hälfte aller Inhalte der Beiträge im Forum aufgrund der Fehlenden NEs als Nullvektoren dargestellt wurden. Aufgrund des Fehlenden Ergebnisses der Gradienten können nur die Einordnungen anhand des Zustandsvektors verglichen werden. Hierbei ist interessant, dass über beide Versuche hinweg, gleich Namen hoch eingeordnet werden. So taucht der Nutzer *qa3qa3 ibnAmr at Tamimi* mit nur 3 Beiträgen in allen vier Zustandsvektorbasierten Tabellen auf und ist somit in allen Fällen in der Top 15 der Nutzer die den höchsten Einfluss auf den Internen Zustandsvektor haben. Ein Weiterer Nutzer, der in allen vier Tabellen auftaucht ist *Al Qadi*, dieser hat jedoch 141 Beiträge und somit ist es Wahrscheinlicher, dass er Autoritärer ist. Aufgrund des fehlenden Versuchs kann leider noch nicht gesagt werden, ob sich das Modell basierend auf dem Vokabuler mehr für die Analyse von autoritären Personen eignet.

6.4 Vergleich beider Einordnungsmethoden

Wird der Ansatz mit Gradienten mit dem der Zustandsvektoren verglichen, kann ausgehend von den Versuchen gesagt werden, dass der Ansatz mit Gradienten interpretierbarer ist. In den Tabellen der Einflussvektoren ist zu sehen, dass Nutzer mit einer hohen Anzahl an Beiträgen, höher eingeordnet sind. Bei den Tabellen die auf den Zustandsvektoren basieren, sind auch Nutzer mit sehr wenigen Beiträgen hoch eingeordnet, jedoch widerspricht dies den Voraussetzungen einer autoritären Person, die ihre Autorität nicht mit einer solch geringen Anzahl an Beiträgen erreichen kann. Jedoch dauert das Berechnen der Gradienten für einen Nutzer mit den erlernten Modellen circa 30 Sekunden, bei 3000 Nutzern dauert das etwas mehr als einen Tag. Die mittlere Änderung der Zustandsvektoren kann hingegen für alle Nutzer innerhalb von ein Paar Minuten berechnet werden.

Kapitel 7

Zusammenfassung und Ausblick

Ziel dieser Arbeit war es die Nutzer eines Online Forums mittels RNNs auf autoritäre Nutzer zu untersuchen. Dabei wurden zwei Modelle trainiert und zwei Ansätze ausprobiert. Dabei wurde auf die Problemstellung der Verarbeitung natürlichsprachlicher Daten eingegangen und ein Ansatz gesucht autorität zu definieren. Jedoch ist es schwierig die Ergebnisse zu bewerten, da es kein Objektives maß für Autorität gibt. Die Methode Gradienten zu Analysieren eignet sich jedoch Einflussreiche Nutzer zu finden, sowohl sie nicht unbedingt Autoritär sein müssen, da man Personen mit einer hohen Anzahl an Beiträgen schon einen gewissen Einfluss auf das Forum zusprechen kann. Die Gradientenmethode bietet außerdem noch weitere Möglichkeiten: die Nutzer könnten anhand ihrer Gradienten geclustert werden und sie könnten auf ihre bevorzugten Themen analysiert werden wie in Kapitel 5 angesprochen wurde.

Anhang A

Weitere Informationen

Abbildungsverzeichnis

Literaturverzeichnis

[1] Baeza-Yates, Ricardo und Berthier Ribeiro-Neto: *Modern Information Retrieval*. Addison Wesley, 1999.

[2] Balakrishnan, Vimala und Lloyd-Yemoh Ethel: *Stemming and Lemmatization: A Comparison of Retrieval Performances*. 2:262–267, 01 2014.

[3] Bottou, Léon: *Large-Scale Machine Learning with Stochastic Gradient Descent*. In: Lechevallier, Yves und Gilbert Saporta (Herausgeber): *Proceedings of COMPSTAT'2010*, Seiten 177–186, Heidelberg, 2010. Physica-Verlag HD.

[4] Bouwman, Vanessa: *Global Digital Report 2018*, 2018. `https://wearesocial.com/de/blog/2018/01/global-digital-report-2018`.

[5] Dennis, J. E. und Robert B. Schnabel: *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Society for Industrial and Applied Mathematics, jan 1996.

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

[7] Hechtlinger, Yotam: *Interpretation of Prediction Models Using the Input Gradient*, 2016.

[8] Hochreiter, Sepp und Jürgen Schmidhuber: *Long short-term memory*. Neural computation, 9(8):1735–1780, 1997.

[9] Kingsley, Zipf George: *The Psycho-Biology Of Language: AN INTRODUCTION TO DYNAMIC PHILOLOGY*. Routledge, 2014.

[10] Königsberger, Konrad: *Analysis 2*. Springer Berlin Heidelberg, 2004.

[11] Lahsan, Tariq Ibn: *Schließung des Forums*, 2014. `http://www.ahlu-sunnah.com/blog/?p=1`.

[12] Matthes, Joachim und Deutsche Gesellschaft für Soziologie: *Lebenswelt und soziale Probleme - Verhandlungen des 20. Deutschen Soziologentages zu Bremen 1980*. Ardent Media, 1981.

[13] Payam Refaeilzadeh, Lei Tang. Huan Liu: *Cross-Validation.* In: *Encyclopedia of Database Systems*, Seiten 532–538. Springer US, 2009.

[14] Siegelmann, Hava T. und Eduardo D. Sontag: *On the computational power of neural nets.* In: *Proceedings of the fifth annual workshop on Computational learning theory - COLT 92.* ACM Press, 1992.

[15] Trusov, Michael, Anand V Bodapati und Randolph E Bucklin: *Determining Influential Users in Internet Social Networks.* Journal of Marketing Research, 47(4):643–658, aug 2010.

[16] Wu, Yonghui, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes und Jeffrey Dean: *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*, 2016.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie Zitate kenntlich gemacht habe.

Dortmund, den 6. Juli 2018

Berat Özdemir

Eidesstattliche Versicherung
(Affidavit)

Name, Vorname
(Last name, first name)

Matrikelnr.
(Enrollment number)

Ich versichere hiermit an Eides statt, dass ich die vorliegende Bachelorarbeit/Masterarbeit* mit dem folgenden Titel selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

I declare in lieu of oath that I have completed the present Bachelor's/Master's* thesis with the following title independently and without any unauthorized assistance. I have not used any other sources or aids than the ones listed and have documented quotations and paraphrases as such. The thesis in its current or similar version has not been submitted to an auditing institution.

Titel der Bachelor-/Masterarbeit*:
(Title of the Bachelor's/ Master's* thesis):

*Nichtzutreffendes bitte streichen
(Please choose the appropriate)

Ort, Datum
(Place, date)

Unterschrift
(Signature)

Belehrung:
Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -).

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird gfls. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin") zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

Official notification:
Any person who intentionally breaches any regulation of university examination regulations relating to deception in examination performance is acting improperly. This offense can be punished with a fine of up to €50,000.00. The competent administrative authority for the pursuit and prosecution of offenses of this type is the chancellor of TU Dortmund University. In the case of multiple or other serious attempts at deception, the examinee can also be unenrolled, section 63, subsection 5 of the North Rhine-Westphalia Higher Education Act (*Hochschulgesetz*).

The submission of a false affidavit will be punished with a prison sentence of up to three years or a fine.

As may be necessary, TU Dortmund will make use of electronic plagiarism-prevention tools (e.g. the "turnitin" service) in order to monitor violations during the examination procedures.

I have taken note of the above official notification:**

Ort, Datum
(Place, date)

Unterschrift
(Signature)

**Please be aware that solely the German version of the affidavit ("Eidesstattliche Versicherung") for the Bachelor's/ Master's thesis is the official and legally binding version.