Institut für Maschinelle Sprachverarbeitung

Universität Stuttgart
Universitätsstraße 5B
D - 70569 Stuttgart

Bachelorarbeit

Sprechererkennung mit neuronalen Netzen

Felix Casel

Studiengang:	Informatik
Prüfer:	Prof. Dr. Sebastian Padó
Betreuer:	Dr. Christian Scheible
begonnen am:	10.10.2016
beendet am:	10.04.2017
CR-Klassifikation:	I.2.6

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben.
Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet.
Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens.
Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht.
Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Felix Casel

Suttgart, 05.04.2017

Declaration

I hereby declare that the work presented in this thesis is entirely my own.
I did not use any other sources and references that the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations.
Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before.
The electronic copy is consistent with all submitted copies.

Felix Casel

Stuttgart, 05.04.2017

Zusammenfassung

Diese Bachelorarbeit befasst sich mit dem Thema der automatischen Sprechererkennung in Prosatexten, welches mit der Verwendung künstlicher neuronaler Netze behandelt wird. Nach einer Schilderung der Problemstellung, wird ein kurzer Überblick über den aktuellen Stand der Forschung auf dem Gebiet der Sprechererkennung präsentiert. Danach werden zunächst einige Grundlagen des Maschinenlernens, sowie dazugehörige Algorithmen erläutert.
Anschließend wird eine Einführung in den Aufbau und die Funktionsweise künstlicher neuronaler Netze gegeben. Zudem wird auf weitere, im Zuge dieser Arbeit verwendete Methoden eingegangen.
Im Folgenden wird der eingesetzte Lösungsweg präsentiert; dabei wird zunächst ein naiver Ansatz als Vergleichsbasis geschaffen. Im Anschluss werden die in dieser Arbeit entwickelten Modelle vorgestellt.
Hiernach wird der Versuchsaufbau für durchgeführte Tests und die Implementierung der Modelle geschildert. Abschließend werden die Ergebnisse der mit den implementierten Modellen durchgeführten Tests untersucht und miteinander verglichen und analysiert.
Letztlich wird aus der Analyse der Ergebnisse ein Fazit gezogen.

Inhaltsverzeichnis

1 Einleitung **6**

2 Verwandte Arbeiten **7**

3 Grundlagen **9**

3.1 Maschinenlernen 9

3.1.1 Lineare Regression 12

3.1.2 Logistische Regression 13

3.1.3 Aktivierungsfunktionen 15

3.1.4 Kostenfunktionen 16

3.1.5 Gradientenabstieg 17

3.2 Deep Learning 20

3.2.1 Backpropagation 20

3.2.2 Klassifizierung 23

3.2.3 Konvolution 24

3.2.4 Zero-Shot-Learning 26

3.2.5 Optimierungsalgorithmen 26

3.3 Weitere verwendete Algorithmen 28

3.3.1 Maximum Entropie 28

3.3.2 Word2Vec 28

4 Methodik **30**

4.0.1 Kontextsuche 31

4.1 Naiver Ansatz 31

4.1.1 Maximum-Entropie 32
4.1.2 Logistische Regression 32
4.2 Konvolutionsmodelle 33
4.2.1 Konvolutionsmodell mit Softmax-Klassifikator 33
4.2.2 Konvolutionsmodell mit Regression 35

5 Ergebnisse **37**
5.1 Experimentaufbau 37
5.1.1 Implementierung einfaches neuronales Netz 38
5.1.2 Implementierung Konvolution mit Softmax-Klassifikator 38
5.1.3 Implementierung Konvolution mit Regression 39
5.2 Auswertung 39
5.2.1 Maximum-Entropie-Modell 39
5.2.2 Logistisches-Regressions-Modell 41
5.2.3 Konvolutionsmodell mit Softmax-Klassifikator 41
5.2.4 Konvolutionsmodell mit Regression 42
5.3 Analyse und Vergleich 43

6 Fazit und Ausblick **48**

7 Anhang **49**

1 Einleitung

Eine große Menge von Informationen in natürlicher Sprache befindet sich in wörtlicher oder indirekter Rede. Um an den vollen Informationsgehalt zu gelangen, ist es notwendig, den jeweiligen Sprecher zuordnen zu können. In Sachtexten, wie beispielsweise Nachrichtenartikeln, gibt die zugehörige Quelle Auskunft über Glaubwürdigkeit und Neutralität der Informationen. Da die manuelle Zuordnung der Sprecher jedoch für umfangreiche Texte ein langwieriger und aufwändiger Prozess ist, werden automatisierte Verfahren benötigt. Allerdings ist diese Zuordnung kein trivialer Vorgang. Oft gibt es mehrere potentielle Kandidaten oder der Sprecher wird nicht explizit benannt. Daher lässt sich eine korrekte Lösung mitunter nur aus dem Kontext erschließen, was die Entwicklung eines komplexen Verfahrens erfordert.

Ziel dieser Bachelorarbeit ist die Umsetzung eines solchen Verfahrens zur automatischen Attributierung von Sprechern in literarischen Texten mithilfe eines faltenden neuronalen Netzes. Dafür wird zunächst ein solches Verfahren implementiert und angewendet. Die Resultate des Verfahrens sollen auf ihre Qualität, sowie potentielle Schwachstellen analysiert werden. Auf Basis dieser Ergebnisse werden anschließend mögliche Verbesserungen für das Verfahren entwickelt und implementiert. Das daraus resultierende System wird erneut auf seine Qualität getestet und mit den Resultaten des ursprünglichen Entwurfes verglichen, um den Erfolg der Erweiterungen bewerten zu können.

2 Verwandte Arbeiten

Die Verarbeitung natürlicher Sprache ist ein sehr beliebtes und aktuelles Forschungsgebiet, in dem ständig neue Fortschritte gemacht werden. Auch im Bereich der Sprechererkennung gibt es bereits einige publizierte Ansätze, welche jedoch meist auf dem Entwurf fester Algorithmen basieren, anstatt dem Einsatz von Maschinenlernen.

So haben Zhang et al. [2003] bereits ein Paper veröffentlicht, in dem sie Methoden zur automatischen Identifikation und Zuordnung von Sprechern in Kindergeschichten zur späteren Sprachsynthese vorstellen. Dabei kommen regelbasierte Verfahren zur Suche von passenden, in vorangehenden Schritten des Modells bereits identifizierten Sprechern, in unmittelbarer Nähe zu der betreffenden Aussage zum Einsatz.

Einen etwas anderen Ansatz zur Identifikation von Sprechern in fiktionalen Texten verfolgen Glass and Bangay [2007], welche ihr Hauptaugenmerk auf die Identifikation von zur Rede zugehörigen Sprachverben und der jeweiligen Akteure derselben legen. Dieses Modell stützt sich hauptsächlich auf manuell ausgewählte syntaktische Analysen der betreffenden Textstellen, sowie ein dazugehöriges Regelwerk zur Auswahl des besten Kandidaten, und kann so bereits sehr überzeugende Resultate erzielen.

Auch Krestel et al. [2008] haben ein solches, auf syntaktischem Regelwerk basierendes System zur Analyse der Sprachverben, jedoch für die Quellen-Attributierung direkter und indirekter Rede aus Nachrichtenartikeln entwickelt.

Elson and McKeown [2010] setzen in ihrem entwickelten System zur Sprechererkennung in Geschichten zusätzlich zu syntaktischer Analyse ein Maschinenlern-Verfahren ein, welches Merkmale aus dem Text extrahiert und daraus eine Zuordnung zu Sprecher-Vektoren ermittelt.

Pareti und O'Keefe beschäftigen sich in [O'Keefe et al., 2012] mit Attributierung direkter und in [Pareti et al., 2013] mit Attributierung indirek-

ter Rede. Aufbauend auf zunächst regelbasierter Vorbearbeitung des Textes durch Klassifizierung der einzelnen Bestandteile in verschiedene Wortkategorien, sowie Festlegung bestimmter Merkmale wie beispielsweise Abstand oder Sequenz von Wörtern, setzen sie anschließend Maschinenlernverfahren in Form von logistischer Regression und „conditional random field“ ein, um eine automatische Attributierung der Rede zu erreichen.

Ein weiterer Ansatz unter Verwendung überwachten Maschinenlernens wurde von He et al. [2013] entwickelt. Auch bei diesem Ansatz liegt der Fokus auf der Analyse spezifischer, im Voraus definierter Sprachverben, auf deren Grundlage nach möglichen zugehörigen Sprechernamen gesucht wird.

In der Regel basiert also die Zuordnung der Sprecher fast ausschließlich auf der Analyse des Kontextes. Hoover [2016] entwickelt jedoch ein System, welches die Zuordnung des Sprechers allein anhand der getroffenen Aussagen ermittelt. Mittels Clusteranalyse gruppiert der Algorithmus Text anhand der Häufigkeit der verwendeten Wörter.

Das im Zuge dieser Bachelorarbeit entworfene System basiert im Wesentlichen auf dem von Kim [2014] entwickelten Verfahren, welches ein künstliches neuronales Netz mit Konvolutionsschicht einsetzt, um Aufgaben wie beispielsweise die Stimmungsanalyse von Texten zu lösen. Das Grundprinzip dieses Systems basiert auf der Verwendung von Wortvektoren und der Betrachtung von Sätzen als Bilder ohne Höhe. So lässt sich die üblicherweise für Bilddaten verwendete Fähigkeit der Konvolution, lokale Auffälligkeiten zu erkennen und als Merkmale zu extrahieren, für die Verarbeitung natürlicher Sprache zu Nutze machen.

3 Grundlagen

3.1 Maschinenlernen

„Ein Computerprogramm lernt von einer Erfahrung E hinsichtlich einer Klasse von Aufgaben A und Leistungsmaß L, wenn sich seine Leistung bei Aufgaben aus A, gemessen an L, durch die Erfahrung E verbessert.“[1] [Mitchell, 1997]

Ziel des Maschinenlernens ist es, den Lernvorgang so zu automatisieren, dass die Aufgabe bei möglichst geringem Aufwand für den Nutzer möglichst gut erfüllt wird. Dazu soll der Algorithmus nicht vom Programmierer fest vorgeschrieben sein, sondern selbst „lernen“, die Aufgabe A bestmöglich zu lösen.

Die Aufgabe A besteht in der Regel darin, zu einer gegebenen Eingabe eine bestimmte Ausgabe zu erzeugen. Hierbei ist die Eingabe durch eine Ansammlung bestimmter Merkmale gegeben (zum Beispiel einem Vektor oder einer Matrix aus mehreren Zahlen oder einem Satz aus mehreren Worten). Zwei der wichtigsten gängigen Aufgaben sind Regression und Klassifizierung:

Regression

Die Regression beschreibt die Aufgabe, zu einer bestimmten Eingabe eine Ausgabe in Form eines numerischen Wertes zu erzeugen. Es wird also eine Funktion der Form $f : \mathbb{R}^n \to \mathbb{R}$ berechnet. [Bengio et al., 2015]

Klassifizierung

Anders als bei der Regression, wird bei der Klassifizierung als Ausgabe des Algorithmus eine Zuteilung in eine von k Klassen, also eine Funktion der Form $f : \mathbb{R} \to \mathbb{N}$ berechnet. Das Ergebnis $y = f(x)$ der Funktion f steht dabei für die Kodierung einer solchen Klasse. [Bengio

[1] eigene Übersetzung

et al., 2015] Diese Kodierung kann beispielsweise in der Form einer einzelnen Zahl $n \in \{1...k\}$ oder durch eine „One-Hot-Kodierung“ in Form eines Vektors $y \in \mathbb{N}^k$ der Form

$$y_i = \begin{cases} 1, \text{falls x in Kategorie i} \\ 0 \text{ sonst.} \end{cases}$$

erzeugt werden. Beispiele für Klassifizierung sind u.a. Objekterkennung oder die Aufgabe dieser Arbeit, also der Zuordnung eines Eingabesatzes x zu einem von k möglichen Sprechern.

Die Qualität der Ausgaben des Algorithmus lässt sich durch das Leistungsmaß L quantifizieren. [Bengio et al., 2015] Es gibt viele verschiedene Möglichkeiten, das Leistungsmaß, je nach Aufgabe des Algorithmus passend zu definieren. Beispielsweise kann bei der Klassifizierung die Genauigkeit oder der F-Score als Leistungsmaß verwendet werden; bei der Regression hingegen, der relative oder absolute Fehler der Ausgabe gegenüber dem erwarteten Ergebnis. Das Leistungsmaß ist also eine Einschätzung davon, wie gut der Algorithmus die Aufgabe erfüllt.
Die Menge aller zur Verfügung stehenden Daten für einen Maschinenlernalgorithmus nennt man den Datensatz. Eine „Erfahrung“ E für den Algorithmus besteht aus einer einzelnen Eingabe aus dem Datensatz, einem sogenannten Datenpunkt. [Bengio et al., 2015]
Ein solcher Datensatz wird für gewöhnlich in drei Teile aufgeteilt: Trainingssatz, Testsatz und Entwicklungssatz, um eine möglichst genaue Einschätzung der Qualität des Algorithmus zu ermöglichen.

- Der Trainingssatz macht den Hauptanteil (für gewöhnlich etwa 80-90%) eines Datensatzes aus. Die Datenpunkte des Trainingssatzes dienen dem Algorithmus zum „Erlernen“ der Aufgabe.

- Der Testsatz (für gewöhnlich etwa 10% des Datensatzes) dient nach oder während dem Trainingsvorgang des Algorithmus dazu, dessen Leistungsmaß zu berechnen. Die Datenpunkte des Testsatzes werden nicht zum Training verwendet und dienen somit als ungesehene Beispiele für den Algorithmus.

- In der Entwicklungsphase werden die Datenpunkte des Entwicklungssatzes für die Auswertung des Leistungsmaßes verwendet; sie bieten also eine Richtlinie zur Optimierung des Algorithmus (etwa 10% des Datensatzes). Ein Entwicklungssatz ist lediglich eine Hilfestellung für den Entwickler des Algorithmus und somit nicht unbedingt notwendig. Da der Entwicklungssatz aus anderen Datenpunkten besteht als der Testsatz, lässt sich durch dessen Nutzung verhindern, dass der Algorithmus bereits bei der Entwicklung auf einen bestimmten Testsatz anstatt für alle möglichen, ungesehenen Datenpunkte optimiert wird. So kann es nicht zu einer unbeabsichtigten Verzerrung der erreichten Leistung kommen.

Allgemein lässt sich das Maschinenlernen in zwei Grundkategorien aufteilen: überwachtes und unüberwachtes Lernen.

Unüberwachtes Lernen

Algorithmen für unüberwachtes Lernen betrachten einen Datensatz und versuchen dabei, nützliche Eigenschaften des Datensatzes zu erlernen. Je nach Anwendung kann eine solche Eigenschaft beispielsweise eine Wahrscheinlichkeitsverteilung sein, die den Datenpunkten zugrunde liegt, oder andere Aufgaben, wie das Clustern der Datenpunkte anhand ähnlicher Merkmale [Bengio et al., 2015] Ein Beispiel für unüberwachtes Lernen ist der in dieser Arbeit verwendete Word2Vec-Algorithmus. [Mikolov et al., 2014]

Überwachtes Lernen

Im Gegensatz zum unüberwachten Lernen, bei dem das genaue Ergebnis des Algorithmus unbekannt ist, wird bei dem überwachten Lernen jedem Datenpunkt, also jeder Sammlung von Merkmalen, ein sogenanntes Label - ein gewünschtes oder erwartetes Ergebnis - zugeordnet. Anhand des verwendeten Leistungsmaßes lässt sich so zu jedem Zeitpunkt während der Laufzeit des Algorithmus die momentane Qualität der erzeugten Ausgabe relativ zur erwarteten berechnen. Dies erlaubt die Optimierung des Algorithmus für sehr spezifische Probleme, sowie die genaue Überwachung während des Trainingsvorgangs. Der im Zuge dieser Arbeit entworfene Algorithmus beispielsweise, fällt in die Kategorie des überwachten Lernens.

3.1.1 Lineare Regression

Künstliche neuronale Netze sind grob nach dem Vorbild der Biologie aufgebaut und bestehen aus einzelnen künstlichen Neuronen, die Schichtweise miteinander verbunden sind. Eine einfache Form eines solchen neuronalen Netzes erlaubt die Berechnung der „linearen Regression“. Sei $\hat{y}$ der von dem gesamten Netz berechnete Ausgabewert, so lautet die berechnete Funktion: $\hat{y} = \omega^T x$ aus einem Eingabevektor $x \in \mathbb{R}^n$ und einem Vektor von Parametern $\omega \in \mathbb{R}^n$. [Bengio et al., 2015]

Die Werte ω_i des Vektors ω werden auch „Gewichte“ genannt. Jedes künstliche Neuron hat ein Gewicht ω_i und berechnet den Ausgabewert $\hat{y}_i$ durch $\hat{y}_i = x_i \cdot \omega_i$. Diese Ausgabewerte werden, entsprechend dem biologischen Vorbild einer einzelnen vernetzten Nervenzelle auch „Aktivierungen“ genannt.

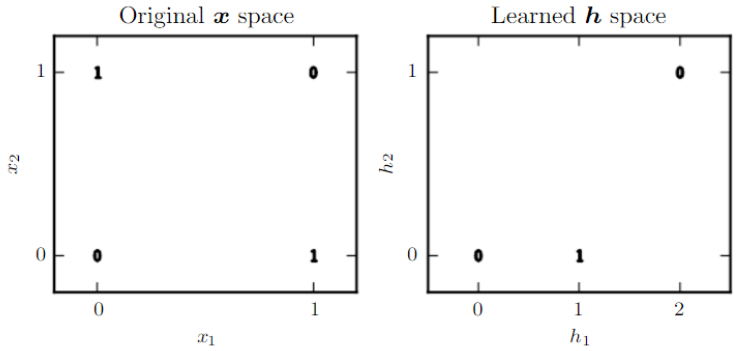

Abbildung 1: XOR-Funktion [Bengio et al., 2015]

Ein etwas komplexeres Modell der linearen Regression verwendet zusätzliche Parameter, sogenannte "Bias" b, sodass die Berechnung der Werte lautet:

$$\hat{y}_i = \omega_i \cdot x_i + b_i.$$

[Bengio et al., 2015]

Durch Änderung der Parameter anhand eines passend gewählten Leistungsmaßes lässt sich die Formel für $\hat{y}_i$ anpassen. Das Modell ist also in der Lage, durch wiederholtes Berechnen und Evaluieren der Ergebnisse, sowie damit verbundenem Anpassen der verwendeten Parameter, zu lernen.

3.1.2 Logistische Regression

Zwar kann ein einfaches Modell der linearen Regression bereits einige Funktionen erlernen, jedoch lassen sich viele Funktionen mithilfe nur einer „Schicht" solcher Einheiten nicht lösen, wie Bengio et al. [2015] beispielsweise für die „XOR"-Funktion zeigt:

In Abbildung 1 (links) sind die geforderten Ausgaben der XOR-Funktion abgebildet; die Ausgabe von $x_1 \cdot w_1$ muss von dem Wert von x_2 abhängen (und umgekehrt). Werden die Eingabewerte x_1 und x_2 jedoch mit festen w_1

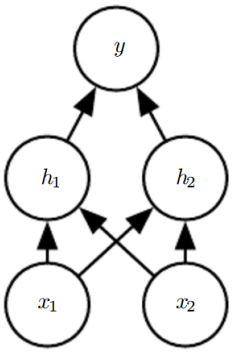

Abbildung 2: XOR-Netzwerk [Bengio et al., 2015]

bzw. w_2 multipliziert, so kann der Wert von x_1 den Parameter w_2 nicht beeinflussen. Ein solches Modell ist also nicht in der Lage, die XOR-Funktion zu lösen.
Damit ein lineares Modell das Problem lösen kann, müssen die ursprünglichen Eingaben zu Merkmalen transformiert werden, sodass alle Eingaben mit Ausgabe ‚1' auf einen einzelnen Punkt im Merkmalsraum abbilden (siehe Abbildung 1, rechts). Die Ausgabe nimmt nun mit steigendem h_1 zu und steigendem h_2 ab. [Bengio et al., 2015]

Das Modell wird also in zwei Schichten aufgeteilt (siehe Abbildung 2), wobei die erste mit f die Merkmale h erzeugt, welche dann von der zweiten Schicht linear zu der Ausgabe y zusammengesetzt werden. Schichten zwischen der Ein- und der Ausgabeschicht werden meist versteckte Schichten („hidden layers") genannt.

Da jede Komposition ausschließlich linearer Funktionen durch eine einzige lineare Funktion dargestellt werden kann, erzeugen offensichtlich auch beliebig viele Schichten mit linearer Ausgabe ein lineares Ergebnis. Daher muss der Funktion $f = \omega \cdot x + b$ für die Erzeugung solcher Merkmale eine nichtlineare Funktion g folgen, sodass $h = g(f(x)) = g(\omega \cdot x + b)$. Diese Funktionen werden typischerweise elementweise angewandt und als „Aktivierungsfunktionen" bezeichnet. [Bengio et al., 2015]

3.1.3 Aktivierungsfunktionen

Typische Beispiele für solche Aktivierungsfunktionen sind nichtlineare Funktionen wie etwa die Sigmoid-Funktion:

$$g(x) = \frac{1}{1+e^{-x}}$$

(siehe Abbildung 3, links) oder den Tangens Hyperbolicus

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

(siehe Abbildung 3, rechts), welche das Intervall $(-\infty, \infty)$ auf $(0, 1)$ bzw. $(-1, 1)$ abbilden,

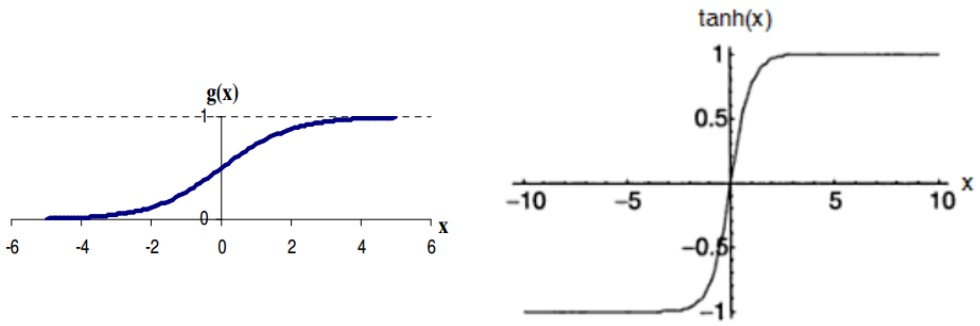

Abbildung 3: Sigmoid und TanH [Karlik and Olgac, 2010]

während für moderne Anwendungen häufig (wie beispielsweise von He et al. [2015]) die Verwendung sogenannter rektifizierter Lineareinheiten„rectified linear units, ReLU“, empfohlen wird. Traditionelle ReLU stellen die Funktion $f = max(0, x)$ dar (siehe Abbildung 4) und führen oft schneller zur Konvergenz bei besseren Ergebnissen als herkömmliche, Sigmoid-artige Aktivierungsfunktionen [He et al., 2015].

Mit Hilfe solcher Aktivierungsfunktionen und der Partitionierung des Modells in ein Netz aus mehreren Schichten ist es möglich, auch komplexe Funktionen berechnen zu können.

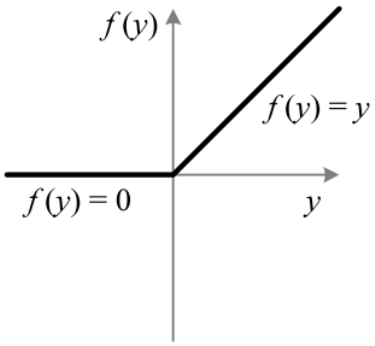

Abbildung 4: Rectified Linear Unit [He et al., 2015]

3.1.4 Kostenfunktionen

Allgemein lässt sich das Training eines künstlichen neuronalen Netzes als Optimierungsproblem betrachten, bei dem der von dem Modell erzeugte Fehler in der Voraussage minimiert werden soll. Wurde ein geeignetes Leistungsmaß gefunden, um die Qualität der Vorhersagen des Modells zu quantifizieren, so lässt sich für den erzeugten Fehler eine dazu passende „Kostenfunktion“ aufstellen.

Durch iterative Voraussage des Modells und Berechnung der Kostenfunktion lässt sich der Gradient berechnen, anhand welchem sich daraufhin eine geeignete Anpassung der Parameter des Modells vornehmen lässt. Mit Hilfe dieses Gradienten wird die Kostenfunktion durch iteratives Anpassen der Parameter zur Konvergenz gebracht, sodass der noch vom Modell erzeugte Fehler bei den Voraussagen möglichst gering ist.

Ein einfaches Beispiel für eine solche Kostenfunktion ist die quadratische Kostenfunktion: Seien $y = y_1, y_2, ...$ die gewünschten Ausgaben der Ausgabeschicht, $\hat{y} = \hat{y}_1, \hat{y}_2, ...$ die tatsächlich ausgegebenen Werte, sowie x die Eingabewerte, ω die Gewichte und b die Bias, so lautet die quadratische Kostenfunktion:

$$C(\omega, b) = \frac{1}{2n} \sum_x ||y(x) - \hat{y}||^2.$$

Nielsen [2015]

Hierbei gibt n die Gesamtanzahl der Eingabewerte x an.

Das verwendete Kostenmodell ist prinzipiell abhängig vom jeweiligen Problem. In der Praxis wird jedoch für einen Großteil der Probleme die „Maximum-Likelihood-Methode“ in Form der Kreuzentropie („cross-entropy“) -Kosten-funktion verwendet, welche in der Regel bessere Ergebnisse liefert als beispielsweise der MSE. Seien wieder y die gewünschten Ausgaben der Ausgabeschicht und $\hat{y}$ die tatsächlich ausgegebenen Werte, so lautet die cross-entropy-Kostenfunktion:

$$C = -\frac{1}{n} \sum_x \sum_j [y_j \; ln \; \hat{y}_j + (1 - y_j) \; ln \; (1 - \hat{y}_j)]$$

[Nielsen, 2015]

Die Festlegung eines Modells $p(y|x)$ liefert automatisch eine Kostenfunktion $log \; p(y|x)$, und hat damit den Vorteil, dass keine modellspezifische Kostenfunktion für jedes neue Problem entworfen werden muss [Bengio et al., 2015]. Je nach Problem lässt sich natürlich dennoch eine speziell darauf abgestimmte Kostenfunktion entwerfen, welche unter Umständen zu besseren oder schnelleren Resultaten führen kann.

3.1.5 Gradientenabstieg

Sei nun $C(v)$ eine beliebige Kostenfunktion, abhängig von nur zwei Variablen v_1 und v_2, so lässt die Funktion sich in einem Koordinatensystem darstellen (siehe Abbildung 5) Ziel des Trainings ist es, das Globale Minimum dieser Kostenfunktion zu finden. Da Kostenfunktionen in der Praxis meist von sehr vielen Variablen abhängen und ungleich komplizierter sein können, lässt sich dieses Minimum in aller Regel nicht analytisch ermitteln. Daher löst man

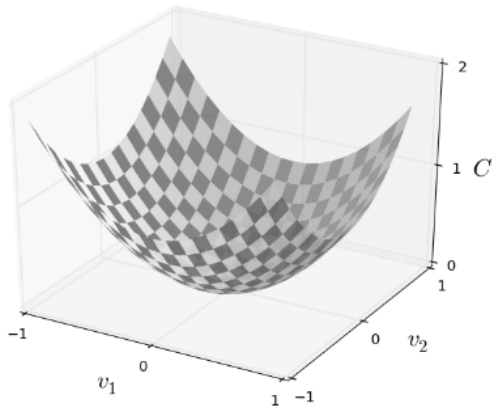

Abbildung 5: Kostenfunktion $C(v)$ [Nielsen, 2015]

dieses Problem durch **Gradientenabstieg** („**gradient descent**“). Zunächst wird eine zufälligen Position auf der „Oberfläche“ der Funktion ausgewählt, von der aus der Algorithmus dann schrittweise, entlang dem lokalen Gefälle, dem Minimum entgegen strebt (siehe Abbildung 6).

Die Richtung des steilsten Abstiegs wird dabei durch die Gegenrichtung des Gradienten ∇C bestimmt, welcher den Vektor der partiellen Ableitungen darstellt:

$$\nabla C = (\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2})^T. \qquad \text{[Nielsen, 2015]}$$

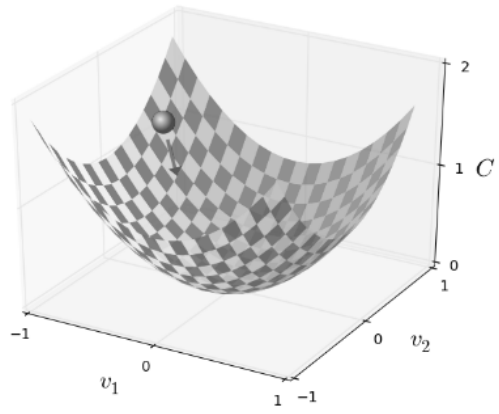

Abbildung 6: Gradientenabstieg $C(v)$ [Nielsen, 2015]

Die gewünschte Änderung Δv in Gegenrichtung des Gradienten in Form eines kleinen Schritts ergibt dann v':

$$v' = v - \Delta v = v - \eta \nabla C \qquad \text{[Nielsen, 2015]}$$

.

Hierbei beschreibt η die **Lernrate**, einen kleinen, positiven und wählbaren Parameter, welcher die Größe der Schritte Δv definiert. Für die Korrekte Funktion des Gradientenabstiegs, ist es wichtig, η gut zu wählen, da eine zu kleine Lernrate nur sehr langsam zur Konvergenz führt, während eine zu große Lernrate dazu führen kann, dass der Algorithmus kontinuierlich über das Minimum „springt“ und so nie zur Konvergenz kommt. In der Praxis ist die optimale Wahl der Lernrate stark vom jeweiligen Problem und der verwendeten Architektur abhängig und stellt ein Optimierungsproblem an sich dar.

Überträgt man nun die Formel für ∇C und Δv auf die Gewichte ω_k und Bias b_l eines künstlichen neuronalen Netzes, so ergeben sich folgende Update-Regeln für die Parameter:

$$\omega_k' = \omega_k - \eta \frac{\partial C}{\partial \omega_k}$$
$$b_l' = b_l - \eta \frac{\partial C}{\partial b_l}.$$

[Nielsen, 2015]

Greift man als Beispiel wieder die einfache Kostenfunktion MSE auf ($C = \frac{1}{n} \sum_x C_x$), so zeigt sich, dass der daraus resultierende Gradient ∇C durch die einzelnen Gradienten ∇C_x für jeden Eingabeparameter x berechnet wird:

$$\nabla C = \frac{1}{n} \sum_x \nabla C_x. \qquad \text{[Nielsen, 2015]}$$

Um beim Training durch iteratives Anpassen der Parameter nun C zu minimieren, muss daher für jeden Schritt die Berechnung aller ∇C_x separat durchgeführt werden, was bei einer großen Anzahl an Eingabewerten einen hohen Rechenaufwand bedeutet.

Dieses Problem lässt sich durch „**stochastischen Gradientenabstieg**“ (**„stochastic gradient descent“, SGD)** erheblich reduzieren: Anstatt jeden einzelnen der n ∇C_x zu berechnen, wählt der Algorithmus zufällig $m < n$ Eingabewerte $X_1, X_2, ..., X_j, ..., X_m$ aus, berechnet deren ∇C_{X_j} und darüber den Mittelwert:

$$\nabla C \approx \frac{1}{m} \sum_{j=1}^{m} \nabla C_{X_j} \qquad \text{[Nielsen, 2015]}$$

Alle Parameter werden anhand dieses approximierten ∇C angepasst; anschließend werden m neue X_j zum Training ausgewählt, so lange bis der gesamte Trainingsdatensatz ein Mal verwendet wurde, also ein gesamter Durchlauf („epoch“) abgeschlossen wurde. [Nielsen, 2015]

3.2 Deep Learning

Künstliche neuronale Netze lassen sich in beliebig vielen Schichten aufbauen. Dabei können die verschiedenen Schichten unterschiedliche Berechnungen durchführen. Wie das Beispiel von „XOR“ zeigt, sind mehrschichtige Netze in der Lage komplexere Funktionen zu berechnen als einschichtige; in der Praxis werden deshalb oft Netze aus vielen Schichten, sogenannte tiefe neuronale Netze (deep neural networks), eingesetzt - daher der Begriff „Deep Learning“.

3.2.1 Backpropagation

Zur Berechnung der partiellen Ableitungen $\frac{\partial C}{\partial \omega}$ und $\frac{\partial C}{\partial b}$ der Kostenfunktion C und der damit einhergehenden Veränderung der Gewichte und Bias eines „tiefen“ Netzwerks, dient der sogenannte „Backpropagation“-Algorithmus.

Zunächst sei die Notation wie folgt definiert: Jeder Schicht (layer) l sei je eine Matrix an Gewichten ω^l und Bias b^l zugeordnet. b^l_j sei dabei der Bias von Neuron j in Schicht l. Dementsprechend sei w^l_{jk} das Gewicht in Neuron j in Schicht l für die Aktivierung a^{l-1}_k des Neurons k in Schicht $l-1$ (siehe

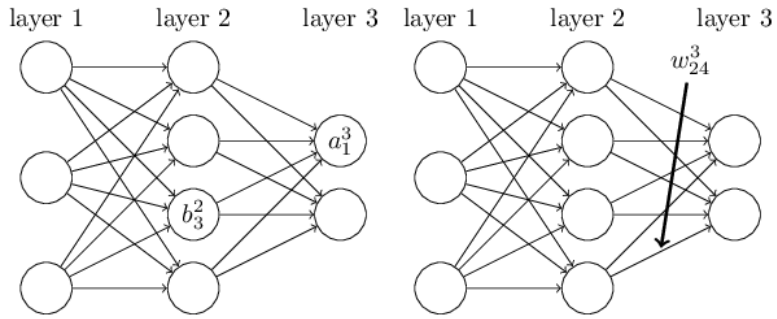

Abbildung 7: Gewichte, Aktivierungen und Bias [Nielsen, 2015]

Abbildung 7). Die Aktivierung a_j^l eines Neurons mit Aktivierungsfunktion σ berechnet sich also durch

$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right).$$

Vereinfacht man dies für eine gesamte Schicht ($\sigma(v)$ steht dabei für die elementweise Berechnung $\sigma(v)_j = \sigma(v_j)$), so ergibt sich für eine Schicht l:

$$a^l = \sigma(\omega^l a^{l-1} + b^l).$$

[Nielsen, 2015]

Solange die Gewichte und Bias der Neuronen nicht perfekt auf die Lösung des Problems angepasst sind, erzeugen sie bei der Berechnung „Fehler“ δ_j^l. Die Größe dieser Fehler bestimmt deren Einfluss auf die Kostenfunktion C. Sei z_j^l die „gewichtete Eingabe“ (weighted Input) eines Neurons j in Schicht l, also

$$z_j^l = \left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right),$$

dann berechnet sich der Betrag des Fehlers durch

$$\delta_j^l = \frac{\partial C}{\partial z_j^l}.$$

[Nielsen, 2015]

Ziel des Backpropagation-Algorithmus ist es, diese Fehler zu minimieren; zunächst wird die gesamte Berechnung des Netzwerks durchgeführt und mithilfe der Kostenfunktion der Fehler in der Berechnung relativ zum gewünschten Ergebnis ermittelt. Dann wird, beginnend bei der letzten Berechnung des Netzes, der Ausgabeschicht, dieser Fehler bis zur Eingabeschicht „zurückpropagiert“ und die einzelnen Gewichte und Bias der Neuronen werden entsprechend ihres Einfluss auf den Fehler angepasst. Die einzelnen Berechnungen lauten dabei wie folgt:
Der Fehler δ^L in der Ausgabeschicht L:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\, \sigma'(z_j^L).$$

Mit Hilfe des Hadamard-Produktes vereinfachte Schreibweise für die gesamte Schicht:

$$\delta^L = \nabla_a C \odot \sigma'(z^L).$$

Der Fehler jeder anderen Schicht l anhängig von der jeweils darauf folgenden Schicht $l+1$:

$$\delta^l = ((\omega^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l).$$

Die Änderungsrate der Kostenfunktion, abhängig von einem beliebigen Bias:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l.$$

Und schließlich die Änderungsrate der Kosten, abhängig von einem beliebigen Gewicht:

$$\frac{\partial C}{\partial \omega_{jk}^l} = a_k^{l-1} \delta_j^l.$$

[Nielsen, 2015]

Anhand obiger Formeln lassen sich für jedes Neuron die Gradienten der Gewichte und Bias durch Rückführung des Fehlers berechnen. Mithilfe eines Optimierungsalgorithmus wie beispielsweise (stochastischem) Gradientenabstieg, können nun schrittweise alle Parameter des neuronalen Netzes angepasst werden.

3.2.2 Klassifizierung

Erfordert das bearbeitete Problem nicht eine Lösung in Form einer kontinuierlichen Ausgabe, sondern eine Einteilung in bestimmte Labels oder Klassen, so müssen Klassifizierungsverfahren angewandt werden. Klassifikatoren werden anstelle gewöhnlicher Regressions-Schichten als Ausgabeschichten eingesetzt und verwenden spezielle Aktivierungsfunktionen.

Eine gebräuchliches solche Aktivierungsfunktion auf Basis der Kreuzentropie, welche auch in dieser Arbeit Anwendung findet, ist die Softmax-Funktion:

$$a_j^L = \frac{e^{z_j^L}}{\sum_k e^{z_k^L}},$$

[Nielsen, 2015]

wobei z_j^L wieder die gewichtete Eingabe der Ausgabeschicht bezeichnet. Die Summe $\sum_j a_j^L$ aller Aktivierungen a_j^L der Ausgabeschicht eines Softmax-Klassifikators beträgt genau 1; die Softmax-Funktion erzeugt also eine Wahrscheinlichkeitsverteilung der Ausgaben.

So wird eine direkte Einteilung in verschienede Klassen ermöglicht, indem jeder Klasse genau ein Ausgabeneuron zugeteilt wird. Die Klasse mit der höchsten Aktivierung ist die wahrscheinlichste Vorrausage des Netzwerks.

3.2.3 Konvolution

Eine spezielle Art von künstlichen neuronalen Netzen stellen "Konvolutions-"Netzwerke (convolutional neural networks, CNN) dar, also neuronale Netze, welche ein- oder mehrere Konvolutions-Schichten einsetzen. Diese verarbeiten Eingabedaten in Form von Bilddaten mit Höhe, Breite und Tiefe, wobei die Tiefe hier die Anzahl der gespeicherten Farbkanäle (üblicherweise rot, grün und blau) widerspiegelt.

Anders als bei gewöhnlichen Schichten eines neuronalen Netzwerks (nachfolgend „voll durch-verbundene Schichten", (fully connected layers)), erlaubt die auf Bildverarbeitung angepasste Architektur der Konvolutionsschichten, dass Neuronen nur mit kleinen Teilregionen von Neuronen vorangehender Schichten verbunden sind. So lässt sich die Menge der nötigen Gewichte und der damit verbundene Rechenaufwand erheblich reduzieren.
Zusätzlich zu den Konvolutionsschichten bestehen Konvolutionsnetzwerke aus voll durch-verbundenen Schichten für Aufgaben wie Klassifizierung oder Regression und sogenannten „Pooling"- oder „Subsampling"-Schichten, welche durch Downsampling den Datenumfang verringern. [Li et al., 2015]

Konvolutionsschichten

Die Funktionsweise der Konvolutionsschichten basiert auf der Verwendung sogenannter Filter, welche kleine Bereiche des Eingabebildes in Breite und Höhe, sowie der gesamten Tiefe abdecken. Diese Filter werden über das Bild bewegt und bilden so durch Berechnung von Skalarprodukten zwischen dem Filter und den Bildregionen ein zweidimensionales Array an Aktivierungen. Mehrere solcher Filter werden übereinandergelegt und erzeugen so wieder eine dreidimensionale Ausgabe (siehe Abbildung 8). [Li et al., 2015]

Pooling-Schicht

Pooling-Schichten reduzieren die Größe dieser so erzeugten Aktivierungsmatrizen. Für gewöhnlich wird „Max"-Pooling verwendet, aller-

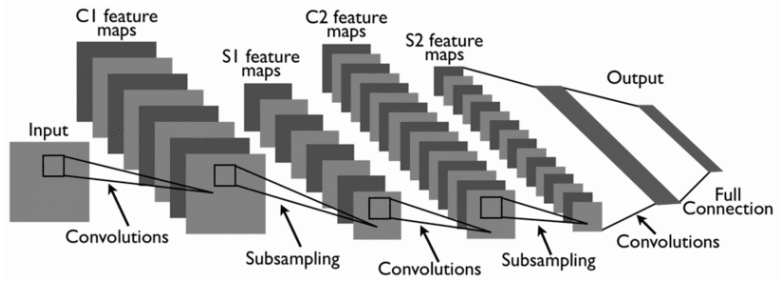

Abbildung 8: Konvolutions-Netzwerk [LeCun et al., 2010]

dings gibt es auch andere Varianten, wie beispielsweise das „Average"-Pooling.
Kleine Filter (oft 2x2 Filter) werden über die Aktivierungen bewegt und je ein einzelner Wert für die abgedeckte Region ermittelt (beim Max-Pooling der höchste Wert). Abhängig von der verwendeten Filtergröße und der Größe der Schritte, in denen der Filter über die Aktivierungen bewegt wird, werden die Höhe und Breite der Aktivierungsmatrix reduziert. Die Tiefe bleibt dabei unbeeinträchtigt, da das Pooling auf jeder Tiefenebene unabhängig arbeitet. [Li et al., 2015]

Ein Spezialfall des Poolings ist das in dieser Arbeit verwendete Global-Pooling (aufgrund der sequentiellen Natur von Sprachdaten auch „max-over-time pooling" genannt) [Collobert et al., 2011]. Dabei wird das Pooling nicht über kleinere Regionen, sondern je über eine gesamte Tiefenebene betrieben. Dies ist deshalb sinnvoll, weil die verwendete Datenstruktur eine ungewöhnlich hohe Tiefe, jedoch keine Höhe aufweist (siehe Unterunterabschnitt 4.2.1).

Das Ergebnis einer oder mehrerer solcher Konvolutions- und Pooling-Vorgänge wird dann in Form eines Vektors zur weiteren Verarbeitung an voll durch-verbundene Schichten weitergegeben.

3.2.4 Zero-Shot-Learning

Nicht in jedem Klassifizierungsproblem sind alle möglichen Klassen bereits bekannt. In solchen Fällen wird das sogenannte „Zero-Shot Learning“ eingesetzt, um einem Algorithmus zu ermöglichen, eine Einteilung auch in zuvor unbekannte Klassen zu treffen.

Eine Möglichkeit einer solchen Zuteilung in unbekannte Klassen, ist, Klassen nicht wie für herkömmliche Klassifikatoren als feste, binäre Einteilung zu definieren, sondern durch einen „Klassenvektor“ in einem Vektorraum. [Alameda-Pineda et al., 2016]

Dementsprechend verwendet die Ausgabeschicht solcher Netze beispielsweise Sigmoid-Aktivierungsfunktionen, anstelle von Klassifikatoren. Vom Algorithmus ermittelte Klassen werden dann anhand der Position des Ausgabevektors im Vektorraum zugeteilt. Zum Training der bereits bekannten Klassen werden ebenfalls Vektoren als Lösung benötigt.

Kostenfunktionen müssen ebenfalls entsprechend definiert werden; eine einfache Möglichkeit für eine solche Kostenfunktion C bietet beispielsweise der euklidische Abstand:

$$C(v) = \sqrt{\sum_i (y_i - \hat{y}_i)^2},$$

Wobei y den erwarteten Ausgabevektor eines Trainingsbeispiels und $\hat{y}$ den tatsächlich ausgegebenen Vektor bezeichnet.

3.2.5 Optimierungsalgorithmen

In der Praxis werden von künstlichen neuronalen Netzen oft enorme Datenmengen verarbeitet; dabei ist der erforderliche Rechenaufwand ein sehr wichtiger Aspekt.
Daher werden immer wieder neue Möglichkeiten entwickelt, diese Verfahren zu beschleunigen. Einen Ansatzpunkt dafür bieten auf (stochastischem) Gradientenabstieg aufbauende, verbesserte Optimierungsalgorithmen.

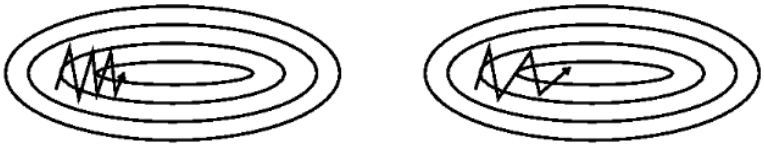

Abbildung 9: SGD mit (rechts) und ohne (links) Momentum [Ruder, 2016]

Momentum

Ein vergleichsweise einfaches Beispiel für einen solchen, beschleunigten Gradientenabstieg stellt der „Momentum“- (engl. „Impuls“) Algorithmus dar. Um den Abstieg zu beschleunigen, und Schwankungen zu verringern, wird den Parameterupdates v_t ein Bruchteil γ (üblicherweise etwa $\gamma = 0.9$) des vorherigen Updates hinzugefügt (siehe Abbildung 9). Seien θ die Parameter des Netzwerks, so lautet die Formel für den Momentum-Algorithmus:

$$v_t = \gamma v_{t-1} + \eta \nabla C(\theta)$$

$$\theta = \theta - v_t$$

[Ruder, 2016]

RMSprop

Ein weiteres Verfahren zur Beschleunigung des Gradientenabstiegs, welches auch im Implementationsteil dieser Arbeit Verwendung findet, ist sogenannte „RMSprop“-Algorithmus. Dieser speichert zu jedem Zeitschritt t einen abnehmenden Mittelwert vorangegangener Gradienten $E[g^2]_t$. Dieser Term hängt dabei nur von seinem jeweiligen Vorgänger $E[g^2]_{t-1}$ ab. Sei g_t der Gradient zum Zeitschritt t, so lautet die Formel für RMSprop:

$$E[g^2]_t = 0,9E[g^2]_{t-1} - +0,1g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

[Ruder, 2016]

3.3 Weitere verwendete Algorithmen

3.3.1 Maximum Entropie

Der Maximum-Entropie Klassifikator ist ein Maschinenlernverfahren, welches auf dem Prinzip der Maximalen Entropie beruht. Dieses besagt, dass derjenige Zustand ein System am besten repräsentiert, welcher die Entropie maximiert.
Dieser Klassifikator wird für verschiedene Probleme der Textklassifizierung verwendet.

Das Modell definiert die Wahrscheinlichkeit $P(c|d)$, dass das Dokument d der Klasse c zugehörig ist. Dabei lautet die Formel:

$$P(c|d) = \frac{1}{Z(d)} exp\left(\sum_i \lambda_{i,c} F_{i,c}(d,c)\right).$$

[Pang et al., 2002]

$Z(d)$ beschreibt eine Normalisierungsfunktion und $F_{i,c}$ ist genau dann 1, wenn ein Wort i vorkommt und das Dokument Klasse c zugeordnet wird, sonst 0.

Die $\lambda_{i,c}$ sind Gewichte, welchen den einzelnen Wörtern (Merkmalen) zugeordnet werden. Ein großes $\lambda_{i,c}$ besagt, dass das Wort i ein starker Indikator für Klasse c ist. [Pang et al., 2002]
Diese Gewichte werden mit Hilfe von Trainingsbeispielen ermittelt; sie erlauben die spätere Klassifizierung ungesehener Dokumente aufgrund ihrer enthaltenen Wörter.

3.3.2 Word2Vec

Mikolov et al. präsentieren in [Mikolov et al., 2013] ein effizientes neuronales Netzwerk, welches in der Lage ist, aus großen Mengen unstrukturierter Daten Vektorrepräsentationen für Wörter zu erlernen. Wörter werden dabei nach deren inhaltlicher Bedeutung gruppiert. Mit diesen erlernten Vektoren lassen

sich Berechnungen durchführen: Beispielsweise ist das Ergebnis der Berechnung vec(„Madrid“) - vec(„Spain“) + vec(„France“) näher an dem Vektor vec(„Paris“) als jeder andere Vektor. [Mikolov et al., 2013].

Implementierungen wie die von Kim [2014] zeigen, dass die Verwendung so trainierter Wortvektoren die Qualität mit natürlicher Sprache arbeitenden Maschinenlernverfahren verbessern kann. Ein auf dieser Architektur trainierter Datensatz mit mehr als 1,4 Millionen Vektoren wurde in Mikolov et al. [2014] veröffentlicht und wird im Implementierungsteil dieser Arbeit verwendet.

4 Methodik

Da die Daten beim gegebenen Problem, im Gegensatz zu vielen herkömmlichen Anwendungsbereichen, nicht in numerischer Form, sondern nur in natürlicher Sprache, welche nicht ohne weiteres quantifizierbar ist, vorliegen, lassen sich Deep-Learning Verfahren nicht darauf anwenden, ohne die Eingabedaten zuvor in eine passende Form zu bringen. Zunächst muss jedoch bestimmt werden, welche Informationen für die spätere Sprechererkennung verwendet werden sollen.

Zwar zeigt Hoover [2016] in seinem Paper, dass eine Zuordnung des Sprechers allein anhand des Gesagten möglich ist, jedoch steckt ein großer Teil der diesbezüglichen Informationen im jeweiligen Kontext. Zudem bedingt eine solche Zuordnung, dass sich die Aussagen von Sprechern im zugrundeliegenden Text entweder inhaltlich oder sprachlich stark genug unterscheiden, um eine eindeutige Unterscheidung treffen zu können. Die Qualität diesbezüglicher Informationen im jeweils vorliegenden Text wird offensichtlich allein durch dessen Autor bestimmt und lässt sich dementsprechend weder kontrollieren noch mit Bestimmtheit voraussagen. Daher scheint es sinnvoll, mehrere Varianten des Problems zu betrachten: sowohl die Zuordnung allein aufgrund des Zitates, als auch eine mit Miteinbeziehung (oder alleiniger Verwendung) des Kontextes.

Wird nun der Kontext miteinbezogen, so ist zu beachten, dass, je nach der Form eines Textes, der für eine Aussage relevante Kontext nicht unbedingt in unmittelbarer Nähe zum Zitat steht. So können manche Texte, insbesondere solche, die beispielsweise ursprünglich als Bühnenstück geschrieben wurden, oft überwiegend in längeren, rein in Dialogform geschriebenen Passagen vorliegen, was eine eindeutige Zuordnung aus dem unmittelbaren Kontext, selbst für einen Menschen, mitunter unmöglich machen kann.

4.0.1 Kontextsuche

Um dem Algorithmus eine Lösung des Problems zu ermöglichen, müssen zunächst die relevanten Textstellen extrahiert werden; eine korrekte Zuordnung der Sprecher zu allen Zitaten für die Trainingsdaten, sowie zur Überprüfung der Performanz muss offensichtlich von Hand erstellt werden und wird daher inklusive der damit einhergehenden Extraktion der markierten Zitate als gegeben vorausgesetzt.
Zunächst wird der gesamte Text eingelesen und mithilfe eines Tokenizers in Tokens eingeteilt, welche im folgenden weiterverarbeitet werden können. Das Modell sieht die Verwendung einer frei wählbaren Anzahl t und u von Tokens vor bzw. nach jedem vorkommen von wörtlicher Rede vor. Da die einfache Verwendung aller t und u unmittelbar vor- bzw. nachstehenden Tokens in vielen Fällen keinen für die Problemlösung relevanten Kontext beinhaltet, scannt der Algorithmus den Text wie folgt:
Vom Zitat ausgehend werden in beide Richtungen solange t bzw. u Tokens zum jeweiligen Kontext hinzugefügt, bis die gewünschten Anzahlen erreicht sind; dabei werden jedoch alle Vorkommen wörtlicher Rede durch jeweils ein Ersatztoken „$“ ersetzt, sodass der Kontext nur noch aus Erzählerbericht sowie den jeweiligen Verweisen auf wörtliche Rede besteht. Dadurch können auch in längere Dialogpassagen eingebettete Zitate durch umstehende Verweise auf Sprecher und „abzählen“ der Ersatztokens korrekt zugeordnet werden.

4.1 Naiver Ansatz

Als Baseline für das Modell dienen zunächst zwei naive Ansätze: Ein Maximum-Entropie-Modell, das den Originaltext in Form einzelner Tokens verarbeitet und ein einfaches künstliches neuronales Netz, das mittels logistischer Regression Sätze in Form eines Eingabevektors erhält.

4.1.1 Maximum-Entropie

Für das Maximum-Entropie-Modell werden die Sätze mit den zugeordneten Sprechern einfach in das für den Algorithmus erforderliche Format gebracht, um eine Unterscheidung zwischen Kontext- und Aussagedaten treffen zu können, werden an alle Tokens des davor- und danach stehenden Kontextes jeweils „Pre“ bzw. „Post“ angehängt. Dementsprechend erzeugt der Algorithmus unterschiedliche Gewichte für die selben Wörter, sollten sie in unterschiedlichen Kontextbereichen auftreten.

4.1.2 Logistische Regression

Ein erstes, simples Modell eines neuronalen Netzes besteht aus nur zwei Schichten: Einer voll durch-verbundenen Schicht (fully connected layer), für logistische Regression und einer Ausgabeschicht mit Softmax-Klassifikator (Siehe Abbildung 10). Für die Wortvektoren werden im voraus trainierte „Word2Vec“-Daten verwendet, welche in Form von Vektoren mit 300 Dimensionen vorliegen. [Mikolov et al., 2014]

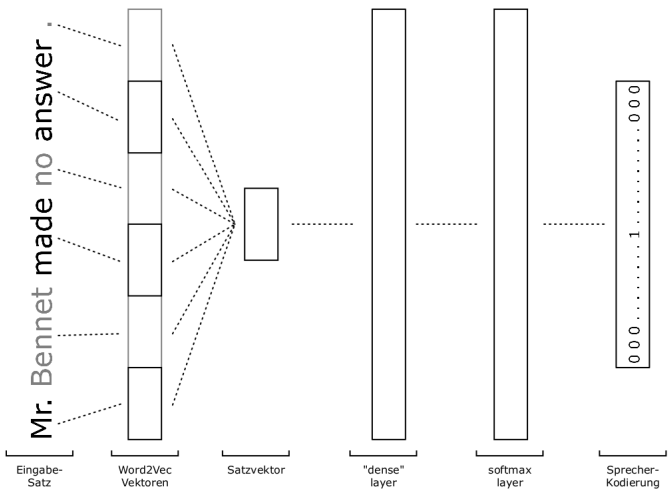

Abbildung 10: Einfaches NN-Modell

Da dieses Modell nur Daten in Form eines Vektors, nicht jedoch einer Matrix erlaubt, müssen die einzelnen Wortvektoren eines Satzes zu einem gemeinsamen Satzvektor zusammengefasst werden (beispielsweise durch Aufsummierung oder berechnen des arithmetischen Mittelwertes). Als Kostenfunktion des Modells wird „multi-class cross-entropy“ verwendet.

Offensichtlich gehen bei der Bildung von Satzvektoren Informationen über die genauen Einzelwörter verloren, zudem hat dieses simple Modell, wie schon das Maximum-Entropie-Modell keine Informationen über lokale Abhängigkeiten der einzelnen Wörter zur Verfügung, welche jedoch für eine präzise Vorhersage des Sprechers essentiell sind.

4.2 Konvolutionsmodelle

Aufgrund der Schwächen der einfacheren Modelle, werden im folgenden neuronale Netze verwendet, die sich die mathematische Konvolution zunutze machen („convolutional neural networks“ - CNN). Diese Netze erlauben die Erkennung und Verarbeitung postitionsabhängiger Informationen. Kim [2014] zeigt in seinem Paper, dass CNN unter Verwendung von Word2Vec-Wortvektoren in der Lage sind, komplexe Aufgaben, wie etwa Stimmungsanalyse einer Aussage, gut zu lösen.

4.2.1 Konvolutionsmodell mit Softmax-Klassifikator

Ein solches Modell (siehe Abbildung 11) wurde im Zuge dieser Arbeit entwickelt: Da CNN für gewöhnlich für die Verarbeitung von Bilddaten verwendet werden, bietet sich an, wie in der Arbeit von Kim [2014], Sätze als eindimensionale Bilder zu betrachten:
Die Breite des „Bildes“ ergibt sich dabei durch die Länge des Satzes, die Höhe des „Bildes“ beträgt immer genau Eins; die Tiefe (bei Bilddaten für gewöhnlich gegeben durch die Anzahl der gespeicherten Farbkanäle) beträgt in diesem Fall 300, also die Anzahl der Dimensionen der verwendeten

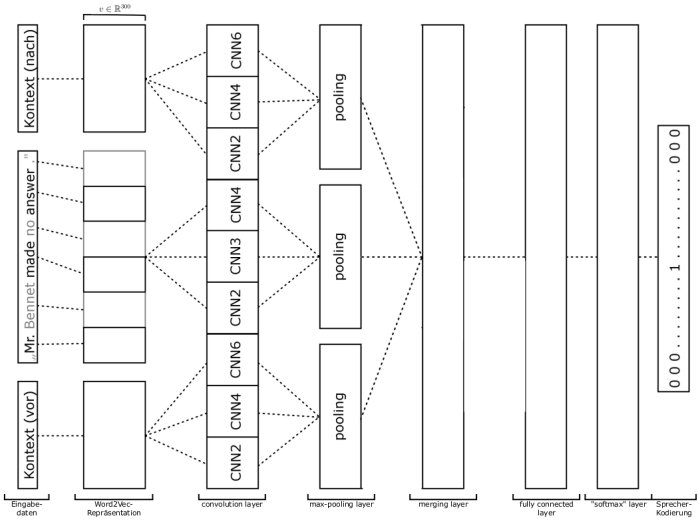

Abbildung 11: CNN-Modell mit Softmax Klassifikator

Word2Vec-Vektoren.

Die Eingabedaten werden zunächst von Konvolutionsschichten (convolution layers) mit verschiedenen Filterbreiten verarbeitet; wie bereits bei den vorangegangenen Modellen werden Kontextinformationen und eigentliches Zitat getrennt betrachtet.

Im zweiten Schritt findet ein globales „max-pooling“ statt; die Ergebnisse der verschiedenen Filter aller verwendeten Breiten je eines Eingabebereichs werden auf den jeweils maximalen Wert reduziert. Ergebnis dieses Poolings ist je ein einzelner Vektor mit Länge gleich der Anzahl der in der Konvolutionsschicht verwendeten Filter. Diese Vektoren werden anschließend von einer voll durch-verbundenen Schicht mit linearer Regression weiter verarbeitet und letztlich in eine Ausgabeschicht mit Softmax-Klassifikator geleitet. Auch bei diesem Modell wird multi-class cross-entropy als Kostenfunktion verwendet.

Dank der Konvolution ist dieses Modell in der Lage, positionsbedingte Abhängigkeiten zwischen einzelnen Wörtern der Eingabesätze miteinzubeziehen und so deutlich bessere Vorhersagen zu treffen. Allerdings kann dieses

Modell nur für bereits in den Trainingsdaten gesehene Sprecher (Klassen) Vorhersagen treffen, da das Ergebnis aus einer Sprecherkodierung besteht und somit kein direkter Zusammenhang zwischen den Eingabedaten oder Sprechernamen und dem erzeugten Ergebnis besteht.

4.2.2 Konvolutionsmodell mit Regression

Letztlich lässt sich das Problem auch mithilfe eines Zero-Shot Ansatzes lösen: Anstatt einer durch Softmax-Ausgabe erzeugten Sprecherkodierung wird als Ergebnis durch Regression ein Vektor $\hat{y} \in \mathbb{R}^{300}$ nach Vorbild eines Word2Vec-Vektors erzeugt, der den Namen des jeweiligen Sprechers repräsentiert. Um Ergebnisse für die Trainingsdaten in Form eines Einzelnen Vektors $y \in \mathbb{R}^{300}$ zu erzeugen, werden dementsprechend arithmetische Mittelwerte der Bestandteile der betreffenden Sprechernamen gebildet. Die Datenstruktur der Eingabedaten, sowie sowohl der Konvolutions- als auch der Pooling-Teil des neuronalen Netzes bleibt wie in 4.2.1. Anschließend werden die einzelnen Teile zunächst je in eine einzelne und dann eine gemeinsame voll durch-verbundene Schicht weitergeleitet, bevor sie schließlich an die Ausgabeschicht gelangen (siehe Abbildung 12).
Die von dieser Ausgabeschicht verwendete Kostenfunktion besteht aus einer Kombination von Kosinusähnlichkeit und euklidischer Distanz:

Kostenfunktion

Seien $\hat{y}$ die vom Modell erzeugten Ausgaben, so lautet die eingesetzte Kostenfunktion C:

$$C(\hat{y}) = f(\hat{y}) \cdot g(\hat{y}),$$

wobei $f(x)$ die Formel für den quadrierten euklidischen Abstand zu dem Vektor der erwarteten Ergebnisse y darstellt und $g(x)$ die Formel für die Kosinus-Ähnlichkeit zu y:

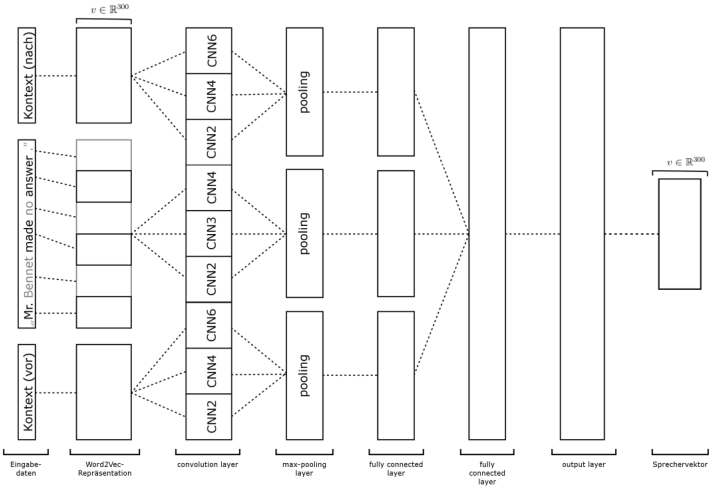

Abbildung 12: CNN-Modell mit Regression

$$
\begin{aligned}
f(\hat{y}) &= ||y - \hat{y}||_2^2 \\
&= \sum_{i=1}^{n} (y_i - \hat{y}_i)^2
\end{aligned}
$$

$$
g(\hat{y}) = 1 - \frac{y \cdot \hat{y}}{||y||_2 \, ||\hat{y}||_2}
$$

und somit

$$
C(\hat{y}) = \sum_{i=1}^{n} \left(y_i - \hat{y}_i)^2 (1 - \frac{y \cdot \hat{y}}{||y||_2 \, ||\hat{y}||_2} \right)
$$

Für den Gradienten ∇C ergibt sich folglich die Formel:

$$
\nabla C = 2(\hat{y} - y) \left(1 - \frac{\hat{y} \cdot y}{||\hat{y}||_2 \, ||y||_2} \right) - \frac{y \, ||\hat{y}||_2 - (\hat{y} \sum_{i=1}^{n} (y_i \, \hat{y}_i))}{||y||_2 \, \hat{y}||_2^3} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2.
$$

Dieses Modell soll es erlauben, auch in den Trainingsdaten ungesehene Klassen (Sprecher) anhand der Eingabedaten identifizieren zu können.

5 Ergebnisse

Dieser Teil der Arbeit befasst sich mit der praktischen Umsetzung der oben genannten Modelle; zunächst wird der genaue Versuchsaufbau geschildert, anschließend werden die erzeugten Ergebnisse vorgestellt und analysiert, sowie miteinander und mit den Ergebnissen anderer Arbeiten verglichen.

5.1 Experimentaufbau

Der folgende Abschnitt bezieht sich auf die Implementierung sowohl im Bezug auf verwendete Bibliotheken und Toolkits, als auch gewählte Funktionen und Parameter für die implementierten Algorithmen und den verwendeten Datensatz.

Als Datensatz für Training und Test der Algorithmen wurde eine von Hand annotierte Version von Jane Austens „Pride and Prejudice“ („Stolz und Vorurteil“) in englischer Sprache verwendet [He et al., 2013]. Nachdem Figuren ohne Äußerung, sowie, zur Vereinfachung des Problems solche, die nur eine einzige (und somit nicht durch Training voraussagbare) Aussage tätigen, sowie deren Aussagen entfernt wurden, umfasst dieser Datensatz 21 Sprecher mit insgesamt 1261 Aussagen, von denen 1022 den Trainingssatz, 112 den Entwicklungssatz und die übrigen 127 den Testsatz bilden.

Der für alle Modelle benötigte Tokenizer zum Parsen der Eingabetexte war bereits vor-implementiert und stammt aus dem Apache OpenNLP-Toolkit [Baldridge, 2005], ebenso wie das dafür eingesetzte, vor-trainierte Tokenizer-Modell zur Identifikation zusammengehöriger Zeichen, ebenfalls in englischer Sprache.

Das eingesetzte Maximum-Entropy-Modell verwendet eine Implementierung von „MALLET“ [Mallet, 2010], einem Maschinenlern-Tooklit für die Verarbeitung natürlicher Sprache. Alle weiteren Modelle wurden mithilfe von „Deeplearning4j“ [Dl4j], einer Open-Source Bibliothek für deep-learning im-

plementiert. Diese stellt umfangreiche Klassen und Methoden zur Implementierung neuronaler Netze bereit. Zudem werden alle verwendeten Word2Vec-Vektoren aus dem von Mikolov et al. [2014] auf Google-News vortrainierten Datensatz bereitgestellt.

5.1.1 Implementierung einfaches neuronales Netz

Für die Implementierung des einfachen neuronalen Netzes wurde stochastic gradient descent mit einer Mini-Batchgröße von 50 als Optimierungsalgorithmus verwendet. Die Anzahl der Gesamtdurchläufe beträgt 1500, da weitere Durchläufe das Ergebnis nicht weiter verbessern. Die Gewichte der einzelnen Neuronen werden nach dem „XAVIER“-Verfahren [Glorot and Bengio, 2010] initialisiert, also nach einer Verteilung (in diesem Fall der Gauß-Verteilung) mit Mittelwert 0 und Varianz

$$Var(W) = \frac{2}{n_{in}+n_{out}},$$

wobei n_{in} die Anzahl der eingehenden Neuronen angibt und n_{out} die Anzahl der Neuronen der nächsten Schicht. Als Aktivierungsfunktion für die versteckten Neuronen werden „ReLU“ verwendet.

5.1.2 Implementierung Konvolution mit Softmax-Klassifikator

Auch für die Implementierung des Konvolutionsmodells wurde stochastic gradient descent mit RMSprop als Optimierungsalgorithmus eingesetzt, jedoch mit einer Mini-Batchgröße von 10 verwendet. Auch hier werden die Gewichte per XAVIER initialisiert. Die Anzahl der Durchläufe beträgt 150.

Die Konvolutionsschicht verwendet 300 Filter der Breiten 2, 4 und 6 bzw. 3, 4 und 5 für Kontext und Zitat. Als Aktivierungsfunktion für die 500 Neuronen der voll durch-verbundenen Schicht wurde der Tangens Hyperbolicus gewählt. Gradienten werden bei dieser Implementierung normalisiert, indem die jeweiligen Beträge elementweise auf 3 beschränkt werden.

5.1.3 Implementierung Konvolution mit Regression

Die Parameter für die Implementierung des Konvolutionsmodells mit Regression wurden genauso gewählt wie die der obigen Konvolutionsimplementierung, mit Ausnahme der Aktivierungsfunktionen der zusätzlichen voll durchverbundenen Schichten, für welche ReLUs eingesetzt werden.

5.2 Auswertung

Die verschiedenen Modelle wurden jeweils in den Varianten „nur Aussage“, „Kontext und Aussage“ und „nur Kontext“ ausgewertet; dabei wurde die Qualität der Voraussagen in Form von Genauigkeit und Macro-F-Score bewertet. Aufgrund der relativ hohen Anzahl an Klassen für einen so kleinen Datensatz (selbst nach Reduzierung der Sprecher nur etwa 50 mal so viele Trainingsbeispiele wie Klassen) und der starken Verzerrung der Klassenverteilung (über 85% der Aussagen in den Testdaten werden von nur fünf der 21 Sprechern getätigt, während zehn der Sprecher in den Testdaten überhaupt keine Aussage treffen) ist insbesondere der F-Score der Ergebnisse sehr gering.
Weitere, nicht in den folgenden Abschnitten aufgeführte Versionen der Ergebnisse befinden sich im Anhang.

5.2.1 Maximum-Entropie-Modell

Interessanterweise erzeugt das Maximum-Entropie Modell die besten Ergebnisse mit nur dem Kontext als Eingabe (siehe Tabelle 1), während die Kombination aus Kontext und Aussage die schlechtesten Ergebnisse liefert.
Betrachtet man die von dem Modell erzeugten Gewichte anhand eines Auszugs der für die Klasse der Sprecherin „Elizabeth Bennet“ am wichtigsten eingestuften Tokens (siehe Tabelle 2), so zeigt sich, dass, vermutlich aufgrund der größeren Fülle der Informationen, abgesehen von dem intuitiv wichtigsten

Gewicht „ElizabethPost“, allgemein eine geringere Gewichtung aller Begriffe vorherrscht.

Sprecher	Präzision	Abruf	F-Score	wahrPos	falschPos	wahrNeg	falschNeg
Elizabeth_Bennet	0,083333	0,076923	0,08	3	33	64	36
Caroline_Bingley	0	0	0	0	4	120	3
Mr._Darcy	0,666666	1	0,8	2	1	124	0
Jane_Bennet	0,069767	0,125	0,089552	3	40	72	21
Lydia_Bennet	0	0	0	0	0	125	2
Maria_Lucas	0	0	0	0	0	127	0
Lady_Catherine	0	0	0	0	0	127	0
Mrs._Bennet	0,129032	0,173913	0,148148	4	27	82	19
Mr._Bingley	0	0	0	0	0	127	0
Mrs._Gardiner	0	0	0	0	5	117	5
Mr._Collins	0	0	0	0	1	116	10
Sir_William	0	0	0	0	0	127	0
Mr._Wickham	0	0	0	0	0	126	1
Mr._Gardiner	0	0	0	0	0	127	0
Charlotte	0	0	0	0	1	123	3
Kitty_Bennet	0	0	0	0	1	126	0
Colonel_Fitzwilliam	0	0	0	0	0	127	0
Mr._Bennet	0	0	0	0	1	111	15
Louisa_Hurst	0	0	0	0	0	127	0
Mary_Bennet	0	0	0	0	0	127	0
Mrs._Reynolds	0	0	0	0	1	126	0

Macro-Messungen:	
Genauigkeit:	0.0944
Präzision:	0.0451
Abruf:	0.0655
F-Score:	0.0532

Tabelle 1: Maxent-Modell, nur Kontext

Kontext und Aussage		nur Kontext		nur Aussage	
ElizabethPost	0.9083890880422812	ElizabethPost	0.8829520471252243	ever	1.1383509703481873
ever	0.7134982658450221	smilePost	0.8264720715697798	kind	0.937946887633032
Mr.	0.6962921828068273	conversationPost	0.8187029769092209	amiable	0.814845049808801
certainly	0.6131273947624993	answeredPre	0.7759527324353556	Oh	0.7931617076404686
Oh	0.5869110694366956	whichPre	0.660959872917659	certainly	0.7853764043386373
But	0.5825494469532116	couldPre	0.6583243547582327	able	0.7351245046029228
suppose	0.5692519904422605	thoughtPost	0.6578181686498148	never	0.7271077168214043
inPost	0.5539893709819498	himselfPre	0.6558726149179493	said	0.7207603786080017
ofPre	0.5294512771483175	inPost	0.6418088692460423	But	0.7031269212027417
must	0.5138418005907127	againPost	0.6241197776180251	Mr.	0.69890342662711

Tabelle 2: Gewichte MaxEnt-Modell

5.2.2 Logistisches-Regressions-Modell

Das Modell der logistischen Regression mit Satzvektoren zeigt nur geringe Verbesserung gegenüber dem Maximum-Entropie Ansatz, was Präzision und Abruf betrifft. Durch die Kombination aller Wortvektoren zu nur einem Satzvektor geht ein Großteil der Informationen, insbesondere solche, die die Positionen der Wörter betreffen, verloren, sodass eine korrekte Zuordnung nur in wenigen Fällen möglich ist.

Sprecher	**Präzision**	**Abruf**	**F-Score**	**wahrPos**	**falschPos**	**wahrNeg**	**falschNeg**
Elizabeth_Bennet	0,32653	0,421052	0,367816	16	33	56	22
Mrs._Bennet	0,40909	0,473684	0,439024	9	13	95	10
Mr._Darcy	0	0	0	0	9	118	0
Jane_Bennet	0,181818	0,095238	0,125	2	9	97	19
Mr._Bennet	0,2	0,2	0,2	3	12	100	12
Caroline_Bingley	0	0	0	0	3	121	3
Lady_Catherine	0	0	0	0	4	123	0
Mr._Wickham	0	0	0	0	0	126	1
Mr._Collins	0,5	0,384615	0,434782	5	5	109	8
Mr._Bingley	0	0	0	0	2	125	0
Mrs._Gardiner	0	0	0	0	0	118	9
Lydia_Bennet	0	0	0	0	0	125	2
Charlotte	0	0	0	0	1	120	6
Colonel_Fitzwilliam	0	0	0	0	0	127	0
Mrs._Reynolds	0	0	0	0	0	127	0
Mr._Gardiner	0	0	0	0	0	127	0
Sir_William	0	0	0	0	0	127	0
Kitty_Bennet	0	0	0	0	0	127	0
Mary_Bennet	0	0	0	0	1	126	0
Louisa_Hurst	0	0	0	0	0	127	0
Maria_Lucas	0	0	0	0	0	127	0

Macro-Messungen:	
Genauigkeit:	0.2756
Präzision:	0.0770
Abruf:	0.0750
F-Score:	0.0746

Tabelle 3: Logistische Regression

5.2.3 Konvolutionsmodell mit Softmax-Klassifikator

Das Konvolutionsmodell mit Softmax-Ausgabe erzeugt, wie erwartet, die besten Resultate bei vollem Informationsgehalt (siehe Tabelle 4), die schlechtesten bei Eingabe nur der Aussage (siehe Anhang, Tabelle 9). Allgemein ist die Qualität der Voraussagen leider relativ gering. Insbesondere der F-Score fällt sehr schlecht aus, da sich an den Ergebnissen zeigt, dass besonders viele der Sprecher mit nur wenigen Aussagen nie korrekt zugeordnet werden konn-

ten und dementsprechend deren Abruf und Präzision ‚0' beträgt, was sich wiederum stark auf die Makro-Erhebungen ausschlägt.

Sprecher	**Präzision**	**Abruf**	**F-Score**	**wahrPos**	**falschPos**	**wahrNeg**	**falschNeg**
Elizabeth_Bennet	0,410256	0,842105	0,551724	32	46	43	6
Mrs._Bennet	0,56	0,736842	0,636363	14	11	97	5
Mr._Darcy	0	0	0	0	0	127	0
Jane_Bennet	0,25	0,047619	0,08	1	3	103	20
Mr._Bennet	0,333333	0,2	0,25	3	6	106	12
Caroline_Bingley	0	0	0	0	1	123	3
Lady_Catherine	0	0	0	0	1	126	0
Mr._Wickham	0	0	0	0	0	126	1
Mr._Collins	0,666666	0,307692	0,421052	4	2	112	9
Mr._Bingley	0	0	0	0	0	127	0
Mrs._Gardiner	0	0	0	0	1	117	9
Lydia_Bennet	0	0	0	0	2	123	2
Charlotte	0	0	0	0	0	121	6
Colonel_Fitzwilliam	0	0	0	0	0	127	0
Mrs._Reynolds	0	0	0	0	0	127	0
Mr._Gardiner	0	0	0	0	0	127	0
Sir_William	0	0	0	0	0	127	0
Kitty_Bennet	0	0	0	0	0	127	0
Mary_Bennet	0	0	0	0	0	127	0
Louisa_Hurst	0	0	0	0	0	127	0
Maria_Lucas	0	0	0	0	0	127	0

Macro-Messungen:	
Genauigkeit:	0.4252
Präzision:	0.1057
Abruf:	0.1016
F-Score:	0.0923

Tabelle 4: CNN-Modell mit Softmax, Kontext und Aussage

5.2.4 Konvolutionsmodell mit Regression

Auch bei dem Konvolutionsmodell mit Regression werden die besten Resultate erwartungsgemäß von dem Modell „Kontext und Aussage" erzielt (siehe Tabelle 5), wobei hier besonders das Modell mit nur der Aussage als Eingabe besonders schlechte Ergebnisse erzeugt (siehe Anhang, Seite 51). So wird beispielsweise „Elizabeth Bennet", die häufigste Sprecherin, nie korrekt zugeordnet; dieses Verhalten lässt sich vermutlich auf fehlende Verweise auf die Namen der Sprecher in den Aussagen der Personen zurückführen, was wiederum die Zuordnung durch Regression auf den Vektor der Sprechernamen erschwert.

Sprecher	Präzision	Abruf	F-Score	wahrPos	falschPos	wahrNeg	falschNeg
Elizabeth_Bennet	0,433333	0,68421	0,530612	26	34	55	12
Mrs._Bennet	0,6	0,631578	0,615384	12	8	100	7
Mr._Darcy	0	0	0	0	6	121	0
Jane_Bennet	0	0	0	0	0	106	21
Mr._Bennet	0,333333	0,8	0,470588	12	24	88	3
Caroline_Bingley	0	0	0	0	1	123	3
Lady_Catherine	0	0	0	0	0	127	0
Mr._Wickham	0	0	0	0	1	125	1
Mr._Collins	1	0,076923	0,142857	1	0	114	12
Mr._Bingley	0	0	0	0	1	126	0
Mrs._Gardiner	1	0,111111	0,2	1	0	118	8
Lydia_Bennet	0	0	0	0	0	125	2
Charlotte	0	0	0	0	0	121	6
Colonel_Fitzwilliam	0	0	0	0	0	127	0
Mrs._Reynolds	0	0	0	0	0	127	0
Mr._Gardiner	0	0	0	0	0	127	0
Sir_William	0	0	0	0	0	127	0
Kitty_Bennet	0	0	0	0	0	127	0
Mary_Bennet	0	0	0	0	0	127	0
Louisa_Hurst	0	0	0	0	0	127	0
Maria_Lucas	0	0	0	0	0	127	0

Macro-Messungen:	
Genauigkeit:	0.4094
Präzision:	0.1603
Abruf:	0.1097
F-Score:	0.0933

Tabelle 5: CNN-Modell mit Regression, Kontext und Aussage

5.3 Analyse und Vergleich

Vergleicht man nun die Ergebnisse der einzelnen Modelle miteinander, so zeigt sich, dass die Verwendung von Konvolutionsschichten in allen Kategorien deutlich bessere Ergebnisse erzielt, als die beiden Baseline-Ansätze.
Zwischen den Ergebnissen der beiden Konvolutions-Modelle gibt es nur sehr geringe qualitative Unterschiede in Form leichter Abweichungen, je nach verwendetem Kontextmodell.

Allgemein zeigen allerdings auch beide Konvolutions-Modelle sehr schwache Resultate. Insbesondere im Vergleich zu anderen Arbeiten, wie etwa [He et al., 2013], deren Modell bei Analyse des selben Datensatzes Genauigkeiten von etwa 85% erreicht.
Bei genauerer Betrachtung der Modelle und ihrer Resultate lassen sich für die mangelnde Qualität der Ergebnisse folgende möglichen Ursachen feststellen:

Wort	Gewicht	Wort	Gewicht
of	0.3630453201371309	somePre	0.5041632599609812
herPre	0.26970634100757734	of	0.4495104841758033
LydiaPost	0.25457338377497085	Catherine	0.31729302422615363
should	0.23240400841464023	Rosings	0.3133525076874344
MaryPost	0.21721039639002446	with	0.3120022962148611
think	0.16644027237282136	Lady	0.3114003643039204
CatherinePost	0.16507022282500589	hadPre	0.27630089385363704
it	0.16419296062366825	that	0.2695917751999074
be	0.16341887472072752	Mr.Post	0.2544380892892173
is	0.1610238623897411	my	0.2509236245873043
anyPost	0.14578953946029002	Mrs.Pre	0.25060869930311025

Tabelle 6: Top-Gewichte Mary Bennet (links) und Mr. Collins (rechts)

Fehlende Syntaxinformationen

Modelle anderer Arbeiten, die Maschinenlernmethoden einsetzen, verwenden üblicherweise eine vorherige Bearbeitung des Textes, etwa durch „Part of Speech Tagging“, das Wörter in bestimmte syntaktische oder semantische Kategorien einteilt. So wird beispielsweise oft nach Sprachverben gesucht, sowie nach potentiell dazu gehörigen Nomen, die für mögliche Sprecher in Frage kommen. Auch lassen sich so unnötige Informationen wie Stoppwörter im Voraus entfernen. Selbst zwischen den Namen potentieller Sprecher und beliebigen anderen, möglicherweise inhaltslosen Wörtern, besteht zunächst kein für die Modelle erkennbarer Unterschied.

In der Praxis zeigt sich, dass dies zu schlechten Assoziierungen zwischen benutzten Wörtern und Sprechern führen kann. Zwar lassen sich die genauen Vorgänge innerhalb der neuronalen Netze und die so entstehenden Merkmale unglücklicherweise nicht nachvollziehen, jedoch lässt die Betrachtung der von dem Maximum-Entropie-Modell erzeugten Gewichte vermuten, dass bei selten sprechenden Personen (siehe Tabelle 6) die stärkste Gewichtung auf vermutlich bedeutungslosen Stoppwörtern wie „of“ liegt, oder den Namen gelegentlicher Konversationspartner innerhalb der Geschichte, was eine zuverlässige und korrekte Zuordnung unmöglich macht.

Kim zeigt zwar in [Kim, 2014], dass ein vergleichbares Modell ohne Syntaxanalyse gute Ergebnisse erzielen kann, jedoch unterscheiden sich die dort behandelten Probleme und Datensätze erheblich von denen dieser Arbeit: Ziel des von Kim entworfenen Systems ist es, die Grundstimmung (positiv/negativ oder subjektiv/objektiv) aus beispielsweise Kritiken oder Kundenbewertungen zu ermitteln; die Klassifizierung beschränkt sich also auf wenige Klassen mit leicht greifbaren Merkmalen.

Mangel an Trainingsdaten

Ein weiteres, zusätzlich zu oben genanntem Punkt beitragendes Problem ist der Mangel an den Modellen zur Verfügung stehenden Trainingsdaten. Generell ist ein Datensatz mit nur etwa 1000 Trainingsbeispielen für ein reines Maschinenlernverfahren bei 21 Klassen und einem vergleichsweise komplexen Problem sehr klein. Betrachtet man jedoch zusätzlich die Verteilung der einzelnen Klassen, so zeigt sich, dass diese extrem verzerrt ist (siehe Anhang, Tabelle 13). Rund 40% aller Aussagen stammen von nur einem Charakter, während ein Großteil der Charaktere nur in weniger als 5% der Fälle sprechen.

Ein Auszug aus den Ausgaben des CNN-Modells mit Softmax-Klassifikator (siehe Abbildung 13) zeigt, dass manche Sprecher selbst dann nicht zu den hoch bewerteten Klassen gehören, wenn ihr Name in unmittelbarer Nähe zum betreffenden Zitat steht. Den Modellen fehlt es also bei den meisten der Klassen an Training. Zwar erzielen He et al. in [He et al., 2013] auf dem selben Datensatz eine gute Genauigkeit, jedoch beruht das dort verwendete System auf der Suche nach im Voraus definierten Sprachverben und Syntaxregeln, was die Fülle an nötigen Trainingsbeispielen für individuelle Klassen reduziert.

Zu der Analyse allein anhand der Aussagen lässt sich zudem anmerken, dass diese Möglichkeit ein hohes Maß an inhaltlicher oder sprachlicher Konsistenz der Charaktere erfordert. Diese Voraussetzung ist nicht notwendigerweise bei Werken aller Autoren sowie aller vorkommenden Personen gegeben.

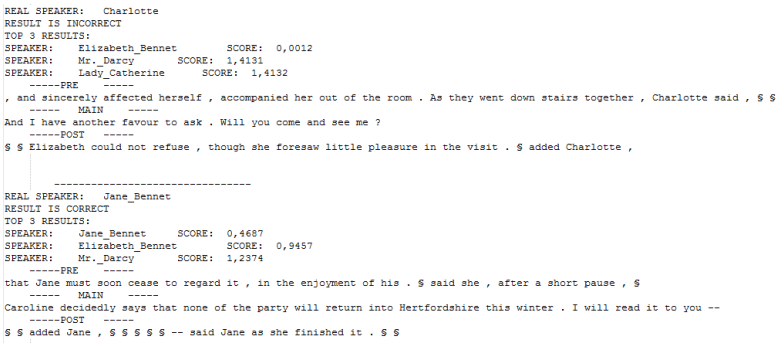

Abbildung 13: Ausgabe des Softmax-CNN-Modells (Auszug)

Ungeeignete Wort- und Sprechervektoren

Das Training der verwendeten Word2Vec-Vektoren beruht auf Google-News Artikeln, sodass die inhaltliche Bedeutung der Vektoren nicht ideal auf die Verwendung für Prosawerke abgestimmt ist. Insbesondere Vektoren für Namen sind sehr stark vom jeweiligen Kontext abhängig; abgesehen vom Geschlecht des Sprechers lässt sich durch Training auf fremden Texten keine inhaltliche Bedeutung auf die Sprechervektoren übertragen.

Zudem sind die verwendeten Wortvektoren auf einzelne Wörter beschränkt, in der Praxis setzen sich jedoch Namen häufig aus mehreren Wörtern zusammen.

Allgemein scheint das Problem der Sprechererkennung durch reines Maschinenlernen unerwartet komplex zu sein. Sollen alle sonst durch Syntaxanalyse bereitgestellten Informationen selbstständig und implizit durch ein neuronales Netzwerk gelernt werden, so sind vermutlich weitaus umfangreichere Datensätze vonnöten. Leider erfordert die manuelle Erstellung annotierter Prosatexte in solcher Menge überaus lange zusammenhängende Texte und ist zudem sehr aufwändig, was die Verbesserung der Ergebnisse durch Erweiterung der Datensätze erschwert.

Einen anderen Ansatz zur Verbesserung bietet möglicherweise die spezifische Anpassung der eingesetzten Wortvektoren, etwa durch Vortraining auf dem

verwendeten Datensatz. Vor allem bei den Sprechernamen könnte dies eine bessere Zuordnung durch stärkeren Zusammenhang zwischen den relevanten Vektoren mit sich bringen.
Andernfalls bleibt immer die Möglichkeit, das Modell um zusätzliche regelbasierte Vorbearbeitung des Textes mittels syntaktischer Methoden zu erweitern.

6 Fazit und Ausblick

Die Testergebnisse der einzelnen Modelle haben gezeigt, dass der Einsatz von neuronalen Netzen mit Konvolutionsschichten zwar bessere Resultate erzeugt als die Vergleichswerte der naiven Ansätze, jedoch fällt die Qualität der Ergebnisse im Allgemeinen verhältnismäßig schwach aus.

Dem Qualitätsvergleich zu Systemen anderer Arbeiten können die entwickelten Modelle nicht standhalten. Unter anderem lassen sich die Schwächen des Systems auf den Verzicht von regelbasierter Vormarkierung der für die Aufgabe relevanten Textbestandteile sowie den Mangel an zu Verfügung stehenden Trainingsdaten zurückführen.
Abschließend lässt sich feststellen, dass das System unter den gegebenen Voraussetzungen nicht ausreichend gut funktioniert und daher erheblicher Verbesserungen bedarf.

Mögliche Weiterentwicklungen der Modelle beinhalten ein besseres Training der Wortvektoren sowie eine Erweiterung der Modelle um Elemente der Syntaxanalyse. Auch ein umfangreicheres Training mithilfe größerer Datensätze könnte den Modellen erlauben, bessere Zuordnungen zu treffen.

7 Anhang

Sprecher	**Präzision**	**Abruf**	**F-Score**	**wahrPos**	**falschPos**	**wahrNeg**	**falschNeg**
Elizabeth_Bennet	0,070588	0,136363	0,093023	6	79	23	38
Caroline_Bingley	0,166666	0,25	0,2	1	5	119	3
Mr._Darcy	0	0	0	0	2	125	0
Jane_Bennet	0,047619	0,047619	0,047619	1	20	93	20
Lydia_Bennet	0	0	0	0	0	125	2
Maria_Lucas	0	0	0	0	0	127	0
Lady_Catherine	0	0	0	0	0	127	0
Mrs._Bennet	0	0	0	0	2	107	19
Mr._Bingley	0	0	0	0	8	119	0
Mrs._Gardiner	0	0	0	0	2	116	9
Mr._Collins	0	0	0	0	0	115	12
Sir_William	0	0	0	0	0	127	0
Mr._Wickham	0	0	0	0	0	126	1
Mr._Gardiner	0	0	0	0	0	127	0
Charlotte	0	0	0	0	0	127	0
Kitty_Bennet	0	0	0	0	1	126	0
Colonel_Fitzwilliam	0	0	0	0	0	127	0
Mr._Bennet	0	0	0	0	0	112	15
Louisa_Hurst	0	0	0	0	0	127	0
Mary_Bennet	0	0	0	0	0	127	0
Mrs._Reynolds	0	0	0	0	0	127	0

Macro-Messungen:	
Genauigkeit:	0.0629
Präzision:	0.0135
Abruf:	0.0206
F-Score:	0.0162

Tabelle 7: Maxent-Modell, Kontext und Aussage

Sprecher	Präzision	Abruf	F-Score	wahrPos	falschPos	wahrNeg	falschNeg
Elizabeth_Bennet	0,06	0,153846	0,08633	6	94	24	33
Caroline_Bingley	0	0	0	0	1	123	3
Mr._Darcy	0,333333	1	0,5	5	10	112	0
Jane_Bennet	0	0	0	0	1	105	21
Lydia_Bennet	0	0	0	0	0	125	2
Maria_Lucas	0	0	0	0	0	127	0
Lady_Catherine	0	0	0	0	0	127	0
Mrs._Bennet	0	0	0	0	0	108	19
Mr._Bingley	0	0	0	0	0	127	0
Mrs._Gardiner	0	0	0	0	3	116	9
Mr._Collins	0	0	0	0	0	114	13
Sir_William	0	0	0	0	0	127	0
Mr._Wickham	0	0	0	0	0	126	1
Mr._Gardiner	0	0	0	0	3	124	0
Charlotte	0	0	0	0	0	127	0
Kitty_Bennet	0	0	0	0	0	127	0
Colonel_Fitzwilliam	0	0	0	0	0	127	0
Mr._Bennet	0	0	0	0	3	110	15
Louisa_Hurst	0	0	0	0	0	127	0
Mary_Bennet	0	0	0	0	0	127	0
Mrs._Reynolds	0	0	0	0	1	126	0

Macro-Messungen:	
Genauigkeit:	0.0866
Präzision:	0.0187
Abruf:	0.0549
F-Score:	0.0279

Tabelle 8: Maxent-Modell, nur Aussage

Sprecher	Präzision	Abruf	F-Score	wahrPos	falschPos	wahrNeg	falschNeg
Elizabeth_Bennet	0,322033	0,5	0,391752	19	40	49	19
Mrs._Bennet	0,470588	0,421052	0,444444	8	9	99	11
Mr._Darcy	0	0	0	0	4	123	0
Jane_Bennet	0,5	0,047619	0,086956	1	1	105	20
Mr._Bennet	0,166666	0,066666	0,095238	1	5	107	14
Caroline_Bingley	0	0	0	0	0	124	3
Lady_Catherine	0	0	0	0	13	114	0
Mr._Wickham	1	1	1	1	0	126	0
Mr._Collins	0,4	0,307692	0,347826	4	6	108	9
Mr._Bingley	0	0	0	0	2	125	0
Mrs._Gardiner	0	0	0	0	1	117	9
Lydia_Bennet	0	0	0	0	0	125	2
Charlotte	0	0	0	0	0	121	6
Colonel_Fitzwilliam	0	0	0	0	0	127	0
Mrs._Reynolds	0	0	0	0	2	125	0
Mr._Gardiner	0	0	0	0	0	127	0
Sir_William	0	0	0	0	7	120	0
Kitty_Bennet	0	0	0	0	1	126	0
Mary_Bennet	0	0	0	0	1	126	0
Louisa_Hurst	0	0	0	0	0	127	0
Maria_Lucas	0	0	0	0	1	126	0

Macro-Messungen:	
Genauigkeit:	0.2677
Präzision:	0.1362
Abruf:	0.1116
F-Score:	0.1127

Tabelle 9: CNN-Modell mit Softmax, nur Aussage

Sprecher	Präzision	Abruf	F-Score	wahrPos	falschPos	wahrNeg	falschNeg
Elizabeth_Bennet	0,34	0,894736	0,492753	34	66	23	4
Mrs._Bennet	0,571428	0,210526	0,307692	4	3	105	15
Mr._Darcy	0	0	0	0	4	123	0
Jane_Bennet	0,333333	0,047619	0,083333	1	2	104	20
Mr._Bennet	0	0	0	0	0	112	15
Caroline_Bingley	0	0	0	0	0	124	3
Lady_Catherine	0	0	0	0	0	127	0
Mr._Wickham	1	1	1	1	0	126	0
Mr._Collins	0,333333	0,153846	0,210526	2	4	110	11
Mr._Bingley	0	0	0	0	0	127	0
Mrs._Gardiner	0	0	0	0	1	117	9
Lydia_Bennet	0	0	0	0	0	125	2
Charlotte	0	0	0	0	0	121	6
Colonel_Fitzwilliam	0	0	0	0	0	127	0
Mrs._Reynolds	0	0	0	0	4	123	0
Mr._Gardiner	0	0	0	0	1	126	0
Sir_William	0	0	0	0	0	127	0
Kitty_Bennet	0	0	0	0	0	127	0
Mary_Bennet	0	0	0	0	0	127	0
Louisa_Hurst	0	0	0	0	0	127	0
Maria_Lucas	0	0	0	0	0	127	0

Macro-Messungen:	
Genauigkeit:	0.3307
Präzision:	0.1228
Abruf:	0.1098
F-Score:	0.0997

Tabelle 10: CNN-Modell mit Softmax, nur Kontext

Sprecher	Präzision	Abruf	F-Score	wahrPos	falschPos	wahrNeg	falschNeg
Elizabeth_Bennet	0	0	0	0	1	88	38
Mrs._Bennet	0,324324	0,631578	0,428571	12	25	83	7
Mr._Darcy	0	0	0	0	1	126	0
Jane_Bennet	0	0	0	0	0	106	21
Mr._Bennet	0,15909	0,933333	0,271844	14	74	38	1
Caroline_Bingley	0	0	0	0	0	124	3
Lady_Catherine	0	0	0	0	0	127	0
Mr._Wickham	0	0	0	0	0	126	1
Mr._Collins	0	0	0	0	0	114	13
Mr._Bingley	0	0	0	0	0	127	0
Mrs._Gardiner	0	0	0	0	0	118	9
Lydia_Bennet	0	0	0	0	0	125	2
Charlotte	0	0	0	0	0	121	6
Colonel_Fitzwilliam	0	0	0	0	0	127	0
Mrs._Reynolds	0	0	0	0	0	127	0
Mr._Gardiner	0	0	0	0	0	127	0
Sir_William	0	0	0	0	0	127	0
Kitty_Bennet	0	0	0	0	0	127	0
Mary_Bennet	0	0	0	0	0	127	0
Louisa_Hurst	0	0	0	0	0	127	0
Maria_Lucas	0	0	0	0	0	127	0

Macro-Messungen:	
Genauigkeit:	0.2047
Präzision:	0.0230
Abruf:	0.0745
F-Score:	0.0334

Tabelle 11: CNN-Modell mit Regression, nur Aussage

Sprecher	Präzision	Abruf	F-Score	wahrPos	falschPos	wahrNeg	falschNeg
Elizabeth_Bennet	0,465517	0,710526	0,5625	27	31	58	11
Mrs._Bennet	0,382352	0,68421	0,490566	13	21	87	6
Mr._Darcy	0	0	0	0	0	127	0
Jane_Bennet	0	0	0	0	0	106	21
Mr._Bennet	0,272727	0,6	0,375	9	24	88	6
Caroline_Bingley	0	0	0	0	0	124	3
Lady_Catherine	0	0	0	0	0	127	0
Mr._Wickham	0	0	0	0	0	126	1
Mr._Collins	0	0	0	0	0	114	13
Mr._Bingley	0	0	0	0	2	125	0
Mrs._Gardiner	0	0	0	0	0	118	9
Lydia_Bennet	0	0	0	0	0	125	2
Charlotte	0	0	0	0	0	121	6
Colonel_Fitzwilliam	0	0	0	0	0	127	0
Mrs._Reynolds	0	0	0	0	0	127	0
Mr._Gardiner	0	0	0	0	0	127	0
Sir_William	0	0	0	0	0	127	0
Kitty_Bennet	0	0	0	0	0	127	0
Mary_Bennet	0	0	0	0	0	127	0
Louisa_Hurst	0	0	0	0	0	127	0
Maria_Lucas	0	0	0	0	0	127	0

Macro-Messungen:	
Genauigkeit:	0.3858
Präzision:	0.0534
Abruf:	0.0950
F-Score:	0.0680

Tabelle 12: CNN-Modell mit Regression, nur Kontext

Sprecher	Aussagen
Elizabeth_Bennet	403
Mrs._Bennet	142
Mr._Darcy	128
Jane_Bennet	100
Mr._Bennet	98
Caroline_Bingley	54
Lady_Catherine	52
Mr._Wickham	42
Mr._Collins	37
Mr._Bingley	35
Mrs._Gardiner	33
Lydia_Bennet	33
Charlotte	22
Colonel_Fitzwilliam	18
Mrs._Reynolds	14
Mr._Gardiner	13
Sir_William	11
Kitty_Bennet	9
Mary_Bennet	7
Louisa_Hurst	5
Maria_Lucas	5

Tabelle 13: Sprecherverteilung - „Pride and Prejudice“

Literatur

Xavier Alameda-Pineda, Timothy M. Hospedales, Elisa Ricci, Nicu Sebe, and Xiaogang Wang. Emerging Topics in Learning from Noisy and Missing Data. *Proceedings of the 2016 ACM on Multimedia Conference - MM '16*, pages 1469–1470, 2016. doi: 10.1145/2964284.2986910. URL `http://dl.acm.org/citation.cfm?doid=2964284.2986910`.

Jason Baldridge. The opennlp project. 2005. URL `http://opennlp.apache.org/index.html`.

Yoshua Bengio, Ian J Goodfellow, and Aaron Courville. Deep learning. *Nature*, 521:436–444, 2015. URL `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.672.7118&rep=rep1&type=pdf`.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural Language Processing (Almost) from Scratch. *Journal of Machine Learning Research*, 12:2493–2537, 2011. ISSN 0891-2017. doi: 10.1.1.231.4614. URL `http://www.jmlr.org/papers/v12/collobert11a.html`.

David K Elson and Kathleen R McKeown. Automatic Attribution of Quoted Speech in Literary Narrative. *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)*, (2009):1013–1019, 2010. URL `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.187.8113&rep=rep1&type=pdf`.

Kevin Glass and Shaun Bangay. A naive, salience-based method for speaker identification in fiction books. *Proceedings of the 18th Annual Symposium of the Pattern Recognition Association of South Africa (PRASA'07)*, pages 1–6, 2007.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the*

13th International Conference on Artificial Intelligence and Statistics (AISTATS), 9:249–256, 2010. ISSN 15324435. doi: 10.1.1.207.2059. URL http://www.jmlr.org/proceedings/papers/v9/glorot10a/glorot10a.pdf?hc_location=ufi.

Hua He, Denilson Barbosa, and Grzegorz Kondrak. Identification of speakers in novels. In *ACL (1)*, pages 1312–1320, 2013. URL https://pdfs.semanticscholar.org/9ae8/419e25a57178335debad63d233db7accc3a1.pdf.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *CoRR*, abs/1502.0, 2015. ISSN 15505499. doi: 10.1109/ICCV.2015.123. URL http://arxiv.org/abs/1502.01852.

D Hoover. Microanalyzing Parts of Texts. *Digital Humanities 2016: Conference Abstracts.*, pages 220–222, 2016. URL http://dh2016.adho.org/abstracts/44.

Bekir Karlik and Av Olgac. Performance analysis of various activation functions in generalized MLP architectures of neural networks. *International Journal of Artificial Intelligence and Expert Systems (IJAE)*, 1(4):111–122, 2010. ISSN 2180124X. URL https://www.researchgate.net/profile/Bekir_Karlik/publication/228813985_Performance_Analysis_of_Various_Activation_Functions_in_Generalized_MLP_Architectures_of_Neural_Networks/links/004635229e9d608b3a000000.pdf.

Yoon Kim. Convolutional Neural Networks for Sentence Classification. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, pages 1746–1751, 2014. ISSN 10709908. doi: 10.1109/LSP.2014.2325781. URL http://emnlp2014.org/papers/pdf/EMNLP2014181.pdf.

Ralf Krestel, Sabine Bergler, and R Witte. Minding the source: Automatic tagging of reported speech in newspaper articles. *Reporter*, pages 2–7, 2008. URL http://hnk.ffzg.hr/bibl/lrec2008/pdf/718_paper.pdf.

Yann LeCun, Koray Kavukcuoglu, and Clément Farabet. Convolutional networks and applications in vision. *ISCAS 2010 - 2010 IEEE International Symposium on Circuits and Systems: Nano-Bio Circuit Fabrics and Systems*, pages 253–256, 2010. ISSN 02714302. doi: 10.1109/ISCAS.2010.5537907. URL http://ieeexplore.ieee.org/abstract/document/5537907/.

Fei-Fei Li, Andrej Karpathy, and J Johnson. Cs231n: Convolutional neural networks for visual recognition. *Stanford University Lecture*, 2015. URL http://cs231n.stanford.edu/.

McCallum A Mallet. a machine learning for language toolkit. 2002, 2010. URL http://mallet.cs.umass.edu/.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013. URL https://arxiv.org/abs/1301.3781.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. word2vec, 2014. URL https://code.google.com/archive/p/word2vec/.

Tom M Mitchell. Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45 (37):870–877, 1997.

Michael A Nielsen. Neural networks and deep learning. 2015. URL http://neuralnetworksanddeeplearning.com/.

Tim O'Keefe, Silvia Pareti, James R. Curran, Irena Koprinska, and Matthew Honnibal. A Sequence Labelling Approach to Quote Attribution. *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, (July): 790–799, 2012. URL http://dl.acm.org/citation.cfm?id=2391033.

Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 79–86. Association for Computational Linguistics, 2002. URL http://dl.acm.org/citation.cfm?id=1118704.

Silvia Pareti, Tim O 'keefe, Ioannis Konstas, James R Curran, and Irena Koprinska. Automatically Detecting and Attributing Indirect Quotations. *Acl*, (October):989–999, 2013. URL https://pdfs.semanticscholar.org/4505/71e68d67d10098d60d5e4d4d40eb1f20b604.pdf.

Sebastian Ruder. An overview of gradient descent optimization algorithms. *Web Page*, pages 1–12, 2016. URL http://arxiv.org/abs/1609.04747.

Deeplearning4j Development Team. Deeplearning4j: Open-source distributed deep learning for the jvm. *Apache Software Foundation License 2*. URL https://deeplearning4j.org.

JY Zhang, AW Black, and Richard Sproat. Identifying speakers in children's stories for speech synthesis. *Interspeech*, pages 2041–2044, 2003. URL https://www.cs.cmu.edu/afs/cs.cmu.edu/Web/People/awb/papers/eurospeech2003/esper.pdf.