Technische Universität Berlin
Institut für Softwaretechnik und Theoretische Informatik
Fachgebiet Datenbanksysteme und Informationsmanagement DIMA

Analyse von Wort-Vektoren deutscher Textkorpora

Bachelorarbeit

Autor Andreas Müller
Studiengang Technische Informatik
Matrikel Nr. 333471

Abgabedatum 9. Juli 2015

Betreuer Johannes Kirschnick
Erstprüfer Prof. Dr. Volker Markl
Zweitprüfer Prof. Dr. habil. Odej Kao

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Berlin, den 16. November 2017

_____

Unterschrift

Zusammenfassung

Zur Verarbeitung großer Textmengen und automatisierter Modellierung von Sprache gibt es viele Ansätze. Dabei erreicht *Deep Learning* mithilfe neuronaler Netzwerke basierend auf englischer Sprache heutzutage nicht nur hervorragende Ergebnisse, sondern kann Modelle mithilfe geeigneter Architekturen auch unüberwacht trainieren.

Diese Arbeit beschreibt die Modellierung natürlicher Sprache mithilfe von neuronalen Netzwerken basierend auf deutschen Textkorpora. Es soll eine Aussage darüber getroffen werden, wie gut die resultierenden Wort-Vektoren die deutsche Sprache repräsentieren. Dazu werden unter Parametervariation verschiedene Modelle trainiert und mithilfe in dieser Arbeit entwickelter Test-Sets evaluiert. Das resultierende Modell mit den besten Evaluationsergebnissen dient anschließend als Grundlage für die Darstellung komplexer Wortzusammenhänge und zur Visualisierung sprachlicher Konzepte.

Abstract

There are several approaches to process large amounts of text and to model language automatically. *Deep Learning* not only achieves outstanding results in modeling the English language with the help of neural networks, but is also able to train models unsupervised with appropriate architectures.

This thesis describes the modeling of natural language using neural networks with German text corpora. The precision of the resulting German word embeddings is analyzed. Various models are trained under parameter variation and evaluated with the help of test sets, that are generated within this work. The resulting model with optimal parameter configuration is then used for complex semantic word connections and visualization of linguistic concepts.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

NLP	*Natural Language Processing* (linguistische Datenverarbeitung)
IR	*Information Retrieval* (Informationsrückgewinnung)
SGD	*Stochastic Gradient Descent*
NNLM	*(Feedforward) Neural Net Language Model*
RNNLM	*Recurrent Neural Net Language Model*
CBOW	*Continous Bag-of-Words*
PCA	*Principal Component Analysis* (Hauptkomponentenanalyse)

1 Einleitung

Die Analyse und Verarbeitung von Texten natürlicher Sprache stellt die Grundlage für zahlreiche Anwendungen der linguistischen Datenverarbeitung (engl. *Natural Language Processing*, NLP) dar und ist heute ein zentrales Forschungsgebiet von Internetfirmen wie Facebook[1], Google[2] oder Yahoo[3], denen große Mengen an Textdaten zur Verfügung stehen.

NLP möchte im Allgemeinen die natürliche Sprache algorithmisch verarbeiten, sodass automatisiert unstrukturierte Daten in geordnete Informationen umgewandelt und gewünschte Informationen extrahiert werden können. Dieser Vorgang wird als *Information Retrieval* (IR) bezeichnet. Teil des IR ist die Information Extraction, welche gezielt Informationen einer bestimmten Vorgabe gewinnen möchte. Um natürliche Sprache automatisiert zu verarbeiten, müssen Wörter in ein geeignetes maschinenlesbares Format umgewandelt werden. Wort-Vektoren stellen eine Möglichkeit dar, Wörter numerisch zu beschreiben und ihren sprachlichen Zusammenhang durch die Beziehung der Vektoren in einem beschränkten Vektorraum abzubilden (Bengio et al., 2003 [4]).

Sprach-Modellierung kann mithilfe verschiedener Ansätze realisiert werden. Zum einen gibt es einfache Modelle wie *Bag-of-Words* (BOW, vgl. Abschnitt 2.2.2) oder N-Gramme (vgl. Abschnitt 2.2.3). Die Größe des Vokabulars ist hier maßgebend für die Dimension der resultierenden Wort-Vektoren, sodass webbasierte Korpora (wie beispielsweise die freie Enzyklopädie Wikipedia) aufgrund ihrer Größe auch bei hoher Rechenleistung sehr lange Berechnungszeiten verursachen. Da z.B. beim BOW-Modell jedes neue Wort im Vokabular jedem Wort-Vektor eine weitere Dimension hinzufügt, verlieren folglich die Wort-Vektoren bezüglich des repräsentierten Wortes

1 Facebook veröffentlichte 2013 einen wissenschaftlichen Artikel zu *Unicorn* (Curtiss et al. [7]), eines Indexing-System für Facebook's Graph Search, welche es ermöglicht, Suchanfragen der Art *„Restaurants in San Francisco liked by people from Beijing“* zu stellen.

2 Google kündigte Ende September 2013 *Hummingbird* an (Shapiro et al. [24]), einen Algorithmus zur effizienteren Verarbeitung und Sortierung des Suchindex. Damit soll die Bedeutung eines Satzes besser verstanden werden, um präzisere Suchergebnisse komplexer Suchanfragen zu erreichen.

3 Yahoo kaufte Ende 2013 *SkyPhrase* (Miners et al. [16]), ein NLP Start-Up, um die Verarbeitung und Resultate von Benutzeranfragen in vielen Yahoo-Anwendungen zu verbessern.

zunehmend an Bedeutung[4]. Zum anderen gibt es beim *Deep Learning* den Ansatz der neuronalen Netzwerke, welche die Wort-Vektoren durch mehrschichtige Abstraktionen ohne Vorgabe trainieren können. Es wurde gezeigt, dass diese mithilfe von neuronalen Netzwerken gelernten Wort-Vektoren sprachliche Merkmale sehr gut abbilden konnten (Bengio et al., 2003 [4]). Dadurch war es möglich, komplexere Aufgaben zu lösen wie beispielsweise *Relation Detection*, *Relation Classification* (Zeng et al., 2014 [27]) oder *Sentiment Analysis* (Socher et al., 2013 [25]).

Viele Modelle wurden bisher fast ausschließlich für die englische Sprache trainiert, sodass es für die deutsche Sprache kaum Modelle gibt, mit deren Hilfe man Aussagen darüber treffen könnte, wie präzise deutsche Wort-Vektoren sind und welche Parameter genauere Ergebnisse auf einem deutschen Korpus erzielen. Aufgrund der sprachlichen Unterschiede zwischen Deutsch und Englisch können Ergebnisse englischer Sprachmodelle nicht ohne Weiteres für die deutsche Sprache übernommen werden. So gibt es im Deutschen beispielsweise männliche, weibliche und sächliche Artikel, wobei sich biologisches und grammatikalisches Geschlecht unterscheiden können. Gegenstände sind nicht wie im Englischen ausschließlich sächlich, sondern haben oft ein Geschlecht, wie „**der** Stuhl“ oder „**die** Lampe“ (***the*** *chair*, ***the*** *lamp*). Außerdem gibt es Unterschiede in der Wort-Reihenfolge: Während englische Sätze und Nebensätze immer dem Schema ‘Subjekt - Prädikat - Objekt’ folgen, so steht in deutschen Nebensätzen das Verb an letzter Stelle, z.B. „Ich weiß, dass **sie** das Buch **liest**.“ (*I know that* ***she reads*** *the book.*), wobei dem Verb auch längere Beschreibungen vorangestellt werden können. Werden Sprachmodelle aufgrund einer bestimmten Anzahl nebeneinander stehender Wörter (Fenster) trainiert, so ergeben sich für beide Sprachen schon wegen der Unterschiede in der Wort-Reihenfolge verschiedene Ergebnisse.

Ziel dieser Arbeit ist es, die durch Variation verschiedener Parameter beim Modelltraining entstandenen deutschen Wort-Vektoren zu analysieren und zu evaluieren. Dazu wird ein Toolkit für Korpuserstellung, Training, Evaluation und Visualisierung entwickelt und für die Evaluation deutsche Test-Sets generiert. Dieses Toolkit, das finale Sprachmodell und die Test-Sets stehen anschließend als Grundlage für weiterführende Anwendungen deutscher Wort-Vektoren zur Verfügung.

4 Dieses Phänomen ist als *Curse of Dimensionality* – Fluch der Dimensionalität bekannt. Der Begriff wurde 1961 erstmals verwendet von R.E. Bellman [2].

1 Einleitung

Anknüpfen soll diese Arbeit an die Ergebnisse des DIMA Projektes „Exploring semantic word similarities in German News Articles“ [1], welches sich mit unüberwachtem Clustering deutscher Nachrichtenartikel beschäftigte. Basierend auf einem Korpus von 3 Millionen Nachrichtenartikeln wurden in diesem Projekt bereits verschiedene Modelle unter Parameter-Variation mithilfe eines neuronalen Netzwerks auf deutscher Sprache trainiert. Es wurde gezeigt, dass das Trainieren von Wort-Vektoren auf einem Korpus deutscher Sprache prinzipiell funktioniert, sodass darauf aufbauend in dieser Arbeit das beschriebene Toolkit zur Bewertung deutscher Modelle implementiert werden kann.

Diese Arbeit ist in 6 Kapitel gegliedert. In Kapitel 2 werden zunächst die Grundlagen zu Wort-Vektoren allgemein und maschinellem Lernen in der linguistischen Datenverarbeitung dargestellt. Anschließend wird in Kapitel 3 genauer auf den Aufbau neuronaler Netzwerke und darauf basierenden Architekturen und Modellen eingegangen. Das Training der Modelle und die daraus resultierenden verbesserten Wort-Vektoren werden hier erläutert. Die Umsetzung des praktischen Teils, der Implementierung eines Toolkits zur Korpora-Erstellung, Modelltraining und Evaluation trainierter Modelle, ist in Kapitel 4 dargestellt. Dabei werden durch Parametervariation verschiedene Modelle spezifiziert und die Erstellung von Test-Sets erläutert. In Kapitel 5 werden die Ergebnisse der trainierten Modelle sowie das daraus resultierende optimale Modell ausgewertet. Abschließend werden die wichtigsten Erkenntnisse in Kapitel 6 zusammengefasst.

Da es in dieser Arbeit um die Modellierung deutscher Sprache geht, werden weitestgehend die üblicherweise verwendeten deutschen Entsprechungen englischer Fachbegriffe verwendet.

2 Grundlagen linguistischer Datenverarbeitung

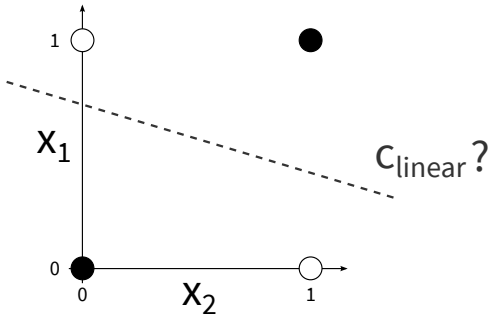
Dieses Kapitel gibt eine Einführung in die geschichtliche Entwicklung von Wort-Vektoren und neuronaler Netzwerke sowie grundlegende Konzepte der Verarbeitung und Modellierung natürlicher Sprache. Die linguistische Datenverarbeitung und neuronale Netzwerke bilden heute eine sehr großes Forschungsfeld, weshalb lediglich ein grober Überblick der Grundlagen dazu gegeben werden soll und der Fokus auf Wort-Vektoren als mögliche Repräsentation von geschriebener Sprache liegt.

2.1 Historischer Kontext von Wort-Vektoren

Das algorithmische Erlernen eines Wort-Vektors (oder allgemein der Darstellung eines Wortes) mittels neuronaler Netzwerke geht auf die Anfänge künstlicher neuronaler Netzwerke zurück. Frank Rosenblatt legte 1958 mit seinem Konvergenztheorem über das Perzeptron[1] den Grundstein künstlicher, an das biologische Nervensystem angelehnter Netzwerke. Er zeigte, dass mit dem angegebenen Lernverfahren jede Funktion gelernt werden kann, die mit diesem Modell repräsentierbar ist [22]. Anknüpfend wurde 1960 von Bernard Widrow das Adaline[2]-Modell eines neuronalen Netzwerks vorgestellt, bei welchem bereits die Gewichte der Eingabe vom Lernalgorithmus angepasst werden konnten [26]. Mit einer Publikation von Marvin Minsky und Seymour Papert im Jahr 1969 wurden die Grenzen des einstufigen Perzeptrons gezeigt [17]. Es war demnach nicht möglich, eine XOR-Funktion zu repräsentieren, da diese nicht linear separierbar ist (XOR-Problem vgl. Abbildung 2.1).

1 Das Perzeptron ist ein von Frank Rosenblatt geprägter Begriff zur Bezeichnung eines einfachen künstlichen neuronalen Netzwerks mit zuerst nur einem einzelnen Neuron [22].

2 Adaline ist ein Akronym für *Adaptive Linear Neuron*.



x_1	x_2	$x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0

Abbildung 2.1: XOR-Problem: Es kann kein linearer Klassifikator c_{linear} für das Ergebnis einer Entweder-Oder-Abfrage gefunden werden.

In den folgenden Jahren nahm das Interesse der weiteren Erforschung künstlicher neuronaler Netzwerke ab. Mögliche Ursache dieser Entwicklung ist neben dem Aufzeigen der Mängel des Pezeptrons durch Minsky auch die Tatsache, dass die zu diesem Zeitpunkt verfügbare Rechenleistung der Computer nicht ausreichte, um repräsentative Modelle zu trainieren. Erst 1982 veröffentliche John Hopfield eine Publikation zu neuronalen Netzwerken mit Rückkopplung (Hopfield-Netze [10]), wodurch Optimierungsprobleme wie das *Travelling Salesman Problem*[3] lösbar wurden. Eine Publikation von Rumelhart et al. zu Backpropagation 1986 [23] (vgl. Abschnitt 3.2.2) und die stetig steigende Rechenleistung[4] führte zur Wiederbelebung der Erforschung künstlicher neuronaler Netzwerke. Trotzdem waren in den 1980er und 1990er Jahren noch zu wenig digitale Daten für ein erfolgreiches Training vorhanden.

Mikolov et al. [15] geben einen kurzen geschichtlichen Überblick über Modelle neuronaler Netzwerke. Demnach wurden 2003 (Bengio et al. [4]) und in den darauffolgenden Jahren durch neuronale Netzwerke trainierte Sprachmodelle vorgestellt, welche die klassische Aufgabe der Vorhersage einer Wahrscheinlichkeitsverteilung für ein Wort aufgrund seiner vorangehenden Wörter lösen sollten. Zunächst wurden diese mit Feedforward-Netzwerken erforscht [4], später mit Recurrent-Netzwerken

3 Das Problem des Handlungsreisenden oder Rundreiseproblem: Die Reihenfolge einmalig zu besuchender Orte soll so gewählt werden, dass der dabei zurückgelegte Weg minimal ist, wobei der letzte Ort wieder der Ausgangsort ist (Rundreise).

4 Die ersten Prozessoren basierend auf Transistortechnik (Mikroprozessoren) kamen 1971 auf den Markt und lösten die zuvor genutzte Röhrentechnik ab. Dies hatte zur Folge, dass Computer kleiner und preiswerter gebaut werden konnten, wobei die Rechenleistung deutlich zunahm.

[12] (vgl. Abschnitt 3.3). Diese Sprachmodelle erzielten sehr gute Ergebnisse, allerdings war dafür auch für heutige Verhältnisse noch immer eine hohe Rechenleistung notwendig. Um mehr Trainingsdaten verarbeiten zu können, wurde folglich nach recheneffizienteren Modellen gesucht. Daraus entstanden Sprachmodelle, welche durch ein neuronales Netzwerk ohne verborgene Ebenen trainiert wurden, wie Continous Bag-of-Words und Skip-Gram (vgl. Abschnitt 3.3).

Nach mehr als 50 Jahren ist das Forschungsgebiet neuronaler Netzwerke heute sehr groß und hat eine Vielfalt von Anwendungsgebieten. So werden neuronale Netzwerke u.a. für Mustererkennung, Bildverarbeitung, Spracherkennung, Medizinische Diagnostik und Klangsynthese verwendet. Diese Arbeit behandelt dabei ausschließlich die Modellierung geschriebener Sprache.

2.2 Maschinelles Lernen für NLP

Im Allgemeinen bezeichnet maschinelles Lernen den Wissenserwerb eines künstlichen Systems, sodass das System anschließend verallgemeinern und das Wissen auf unbekannte Daten anwenden kann. Maschinelle Lernmethoden sind für die Sprachverarbeitung von großer Bedeutung, da es mithilfe entsprechender Modelle möglich ist, semantische und syntaktische Aussagen über Wörter, Wortgruppen oder Dokumente zu treffen.

Maschinelles Lernen wird unterteilt in überwachtes und unüberwachtes Lernen. Überwachung bedeutet dabei, dass es eine Lernvorgabe gibt (z.B. einen Lern-Vektor, engl. *Teaching Vector*). Wird nun ein Modell überwacht trainiert, so werden die Parameter beim Training so gewählt, dass die Ausgabe bezüglich der Lernvorgabe approximiert wird. Da ein richtiges Ergebniss existiert (die Lernvorgabe), ist es möglich, den Fehler der Ausgabe durch den Vergleich mit der Lernvorgabe zu bestimmen und diesen iterativ zu verkleinern, indem die entsprechenden Parameter angepasst werden.

Beim unüberwachten Lernen gibt es keine explizite Lernvorgabe, sodass hier nur aufgrund der Struktur und der Eigenschaften der Eingabe eine Funktion gelernt werden kann. Ein entscheidender Vorteil gegenüber überwachtem Lernen ist, dass Eingabedaten nicht manuell mit Label versehen werden müssen (meist eine sehr

aufwendige und teure Prozedur) und somit viel größere Mengen von Eingabedaten verarbeitet werden können. Ein Nachteil von unüberwachtem Lernen ist die nicht spezifizierbare Lernrichtung. Dadurch ist es aber auch möglich, dass beim unüberwachtem Training Muster in den Eingabedaten erkannt werden, die zuvor noch nicht betrachtet wurden. Da es keine Label zur Überprüfung eines unüberwacht trainierten Modells gibt, kann dieses durch die Beantwortung verschiedener Fragen evaluiert werden (vgl. Mikolov et al. 2012 [15]). Diese wurden im Rahmen dieser Arbeit angelehnt an die Test-Sets von Mikolov et al. für die deutsche Sprache generiert (vgl. Evaluation der Wort-Vektoren in Abschnitt 4.4).

Einige grundlegende Methoden und Modelle zur Überführung von natürlicher Sprache in ein von Maschinen les- und verarbeitbares Format sind im Folgenden aufgeführt.

2.2.1 Wortdarstellung: One-Hot-Kodierung

Funktionen, die mittels Sprachmodellierung gelernt wurden, benötigen numerische Eingaben. Die One-Hot-Kodierung ist eine einfache Möglichkeit der numerischen Darstellung von Wörtern eines Vokabulars. Diese Kodierung wird auch als 1-aus-n-Code bezeichnet, wobei n die Größe des Vokabulars beschreibt.

Hier wird jedem Wort des Vokabulars ein Vektor der Größe n zugeordnet, der genau ein auf '1' gesetztes Bit enthält. Alle anderen Bits sind '0', sodass aus der Position des gesetzten Bits eindeutig das entsprechende Wort ermittelt werden kann.

Es sei beispielsweise angenommen, das Vokabular ist wie folgt definiert:

```
voc = {Beispiel, Dies, ist, ein}
```

Eine mögliche One-Hot-Kodierung (in diesem Fall 1-aus-4-Code) stellen folgende Vektoren für die einzelnen Wörter dar:

```
Beispiel = [1 0 0 0]
Dies     = [0 1 0 0]
ist      = [0 0 1 0]
ein      = [0 0 0 1]
```

Somit entsteht für jedes Wort eine numerische Repräsentation.

2.2.2 Bag-of-Words Modell

Neben der Abbildung einzelner Wörter ist das Bag-of-Words[5] Modell eine einfache Möglichkeit der numerischen Darstellung von Texten (z.B Sätze oder Dokumente) und wird daher u.a. zur Klassifizierung von Dokumenten verwendet.

Nach diesem Modell wird jedem Dokument ein Vektor der Größe des Vokabulars n zu geordnet, welcher sich aus der Summe der One-Hot-kodierten im Dokument enthaltenen Wörtern ergibt.

Es sei angenommen, es existiere ein Korpus mit drei Dokumenten folgenden Inhalts:
Dokument 1: „Hund und Katze sind Tiere."
Dokument 2: „Er hat einen Hund."
Dokument 3: „Sie hat eine Katze."

Daraus ergibt sich das folgende Vokabular, bestehend aus 10 unterschiedlichen Wörtern:

```
voc = {Hund, und, Katze, sind, Tiere, Er, hat, einen, Sie, eine}
```

Nach One-Hot-Kodierung der Wörter des Vokabulars und Addition der Wort-Vektoren eines Dokumentes, ergeben sich folgende Dokumenten-Vektoren:

```
doc1 = [1 1 1 1 1 0 0 0 0 0]
doc2 = [1 0 0 0 0 1 1 1 0 0]
doc3 = [0 0 1 0 0 0 1 0 1 1]
```

Mithilfe dieser Vektoren können die Dokumente nun miteinander verglichen werden. Dazu kann der euklidische Abstand d zweier Vektoren a und b berechnet und die Ähnlichkeit entsprechend abgeleitet werden. Nach Gleichung 2.1 ergeben sich für die Beispielvektoren folgende Abstände:

$$d(a,b) = \|a - b\| = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2} \tag{2.1}$$

$d(doc1, doc2) = 2.646$
$d(doc1, doc3) = 2.646$
$d(doc2, doc3) = 2.450$

5 Der Name „Bag-of-Words" taucht bereits 1954 in einer Publikation von Harris in Zusammenhang mit sprachlicher Verarbeitung auf [9].

Daraus ist ersichtlich dass die Dokumente 2 und 3 den geringsten Abstand also die höchste Ähnlichkeit im Beispielkorpus zueinander aufweisen. Für Vektoren geringer Größe funktioniert dieser Ansatz zwar, aber für hochdimensionale Vektoren, wie sie in der Sprachmodellierung aufgrund der Größe des Vokabulars normalerweise auftreten (vgl. *Curse of dimensionality* in Kapitel 1), sind die Abstände verschiedener Vektoren sehr ähnlich. Mit einer Metrik wie dem euklidischen Abstand kann deshalb keine sinnvolle Aussage mehr über die Ähnlichkeit von Vektoren getroffen werden.

Aus diesem Grund werden Ähnlichkeitsmaße für den Vergleich von hochdimensionalen Vektoren verwendet, wie z.B. die Kosinus-Ähnlichkeit. Diese wird beschrieben als Kosinus des zwischen den Vektoren a und b eingeschlossenen Winkels $\theta_{a,b}$. Je kleiner dieser Winkel ist, desto ähnlicher ist die Richtung beider Vektoren und der Kosinus nähert sich dem Wert 1. Sind beide Vektoren orthogonal zueinander (keine Ähnlichkeit), ist der Kosinus 0. Gleichung 2.2 zeigt die Berechnung der Kosinus-Ähnlichkeit zweier Vektoren a und b mit der Dimension n:

$$\cos(\theta_{a,b}) = \frac{a \cdot b}{\|a\|\|b\|} = \frac{\sum_{i=1}^{n} a_i \cdot b_i}{\sqrt{\sum_{i=1}^{n}(a_i)^2} \cdot \sqrt{\sum_{i=1}^{n}(b_i)^2}} \tag{2.2}$$

Daraus ergeben sich für die drei Beispielvektoren folgende Kosinus-Ähnlichkeiten:

$\cos(\theta_{doc1,doc2}) = 0.2236$
$\cos(\theta_{doc1,doc3}) = 0.2236$
$\cos(\theta_{doc2,doc3}) = 0.2500$

Auch hier haben die Dokumente 2 und 3 eine größere Ähnlichkeit zueinander, als zu Dokument 1.

Wie aus den Beispielen ersichtlich ist, können die Vektoren des Bag-of-Words Modells bereits sprachliche Zusammenhänge abbilden. Dabei werden die Vektoren allerdings ausschließlich aufgrund der Häufigkeit aller im Dokument enthaltenen Wörter gebildet, ohne den Wort-Kontext einzelner Wörter zu berücksichtigen. Dieser ist für das Abbilden von Sprachkonzepten (wie z.B. Verneinung oder Steigerung) und Zusammenhängen von Wörtern aber unbedingt notwendig.

2.2.3 N-Gramm Modell

Ein weiterer Basisansatz zur Sprach-Modellierung ist das N-Gramm. N-Gramme wie auch Bag-of-Words (vgl. Abschnitt 2.2.2) sind Modelle basierend auf Anzahlen (z.B. von Wörtern).

Allgemein entstehen N-Gramme bei der Zerlegung von Texten. Dabei kann ein Text in Wörter oder auch in Buchstaben zerlegt werden. Da es bei Sprachmodellierung aber vorzugsweise um Wörter und ihre Bedeutungen geht, bestehen die hier betrachteten N-Gramme aus Wörtern. Das N steht für die Anzahl der Elemente einer Gruppe. Wird also der Satz „Dies ist ein Beispiel“ in vollständige N-Gramme mit $N = 3$ (auch 3-Gramme oder Trigramme genannt) zerlegt, so lauten diese:

```
{Dies, ist, ein}
{ist, ein, Beispiel}
```

Die folgenden Erklärungen und Formeln basieren auf den Ausführungen von Goodman, 2001 [8]. Hier soll mithilfe von N-Grammen die Wahrscheinlichkeit P einer Wortfolge $w_1...w_i$ errechnet werden. Allgemein geschieht dies mithilfe der bedingten Wahrscheinlichkeiten der einzelnen Wörter in Abhängigkeit des vorherigen Kontextes:

$$P(w_1...w_i) = P(w_1) \times P(w_2 \mid w_1) \times ... \times P(w_i \mid w_1...w_{i-1}) \tag{2.3}$$

Für große i (also lange Wortfolgen) kann es schwierig sein, die Wahrscheinlichkeit zu errechnen, weil die Anzahl der zu berechnenden bedingten Wahrscheinlichkeiten entsprechend groß wird. Aus diesem Grund werden als Vereinfachung oft nur die vorherigen ersten beiden Wörter w_{i-1} und w_{i-2} betrachtet (die Trigramm-Annahme):

$$P(w_1...w_i) \approx P(w_i \mid w_{i-2}w_{i-1}) \tag{2.4}$$

Die Trigramm-Wahrscheinlichkeit kann mithilfe der Anzahl C des Vorkommens der entsprechenden Wortfolgen im Korpus geschätzt werden:

$$P(w_i \mid w_{i-2}w_{i-1}) \approx \frac{C(w_{i-2}w_{i-1}w_i)}{C(w_{i-2}w_{i-1})} \tag{2.5}$$

Diese Schätzung kann unter Umständen (z.B. bei sehr kleinem Vokabular) sehr ungenau werden, wenn beispielsweise ein Trigramm im Trainingskorpus nicht enthalten ist, zwei Wörter daraus aber schon. Die errechnete Wahrscheinlichkeit würde folglich 0 ergeben. Um dieses Problem zu vermeiden, gibt es Techniken zum Ausgleich (engl. *Smoothing*) der Wahrscheinlichkeit einzeln auftretender Sequenzen, wie z.B. das Laplace Smoothing (Peng, 2003 [20]).

2.2.4 Wort-Klassen

Um Sprache zu modellieren, werden charakteristische Merkmale (z.B. ein Merkmalsvektor) benötigt. Eine Möglichkeit ist die Zuordnung von Wörtern zu bestimmten Klassen. Nach Brown et al., 1992 [6] sind Wort-Klassen (engl. *Word Classes* oder *Parts of Speech*) Gruppierungen von Wörtern, die in ihrer Semantik und syntaktischen Funktion ähnlich sind. Beispiele für mögliche Wort-Klassen K sind die Folgenden:

$K_{Farbe} = (rot, blau, gelb, orange)$
$K_{Verb} = (gehen, stehen, laufen, essen)$

Ursprünglich wurden Wort-Klassen manuell erstellt, genauso wie die Zuweisung von Wörtern zu entsprechenden Klassen. Ein Beispiel für einen solchen manuell erstellten Korpus ist der Brown-Korpus[6]. Mit Entwicklung des maschinellen Lernes wurden diese Methoden weitestgehend durch überwachte und unüberwachte Verfahren maschinellen Lernens ersetzt (siehe logistische Regression im folgenden Abschnitt 2.2.5).

2.2.5 Logistische Regression

Die logistische Regression ist eine Möglichkeit der überwachten Klassifizierung von Wort-Vektoren. Ziel der Logistischen Regression ist es, die Klassenwahrscheinlichkeit

6 Der Brown-Korpus ist eine Sammlung von 500 Texten mit rund einer Million Wörtern in englischer Sprache, der von W. N. Francis und H. Kucera in den 1960er Jahren für sprachwissenschaftliche Zwecke zusammengetragen wurde [11].

zu einem Merkmalsvektor vorherzusagen. Gibt es dabei nur zwei mögliche Ergebnisse, spricht man von binärer Klassifizierung. Stehen mehrere Klassen zur Verfügung spricht man von multinomialer logistischer Regression.

Zur Berechnung der Wahrscheinlichkeit nutzt man die logistische Funktion, welche jeden beliebigen reellen Zahlenwert auf einen Wert zwischen 0 und 1 abbildet (Gleichung 2.6, dargestellt in Abbildung 2.2). Das Ergebnis der logistischen Funktion kann dann als die Wahrscheinlichkeit der Zugehörigkeit zu einer Klasse y in Abhängigkeit der unabhängigen Variable x interpretiert werden (Gleichung 2.7).

$$h(x) = \frac{e^x}{e^x + 1} = \frac{1}{1 + e^{-x}} \tag{2.6}$$

$$P(y = 1|x) = h(x) \tag{2.7}$$

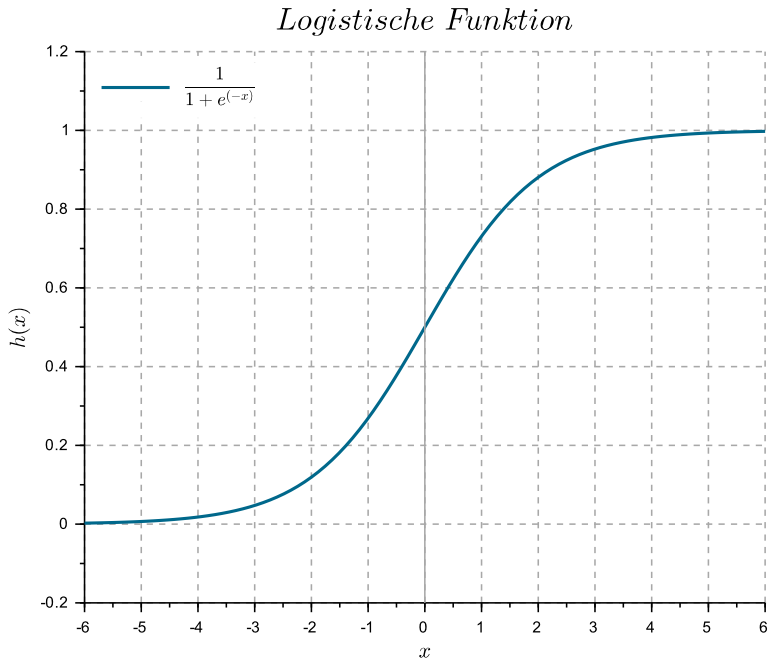


Abbildung 2.2: Logistische Funktion $h(x)$

Um dieses Modell für die Zuordnung des Merkmal-Vektors zu einer von mehreren Klassen zu erweitern, wird x ersetzt durch die Linearkombination $ax_1 + bx_2 + ... + mx_n$ (Bender et al., 2002 [3]).

Prinzipiell funktioniert die logistische Regression wie ein neuronales Netzwerk ohne verborgene Ebenen (vgl. Kapitel 3). Hier wird der Eingabevektor X mithilfe seiner Gewichtung W und der logistischen Funktion h direkt in eine skalare Ausgabe Y umgewandelt (vgl. Abbildung 2.3). Die Verbindungen haben dabei Gewichtungen, welche durch *Stochastic Gradient Descent* (SGD, vgl. Abschnitt 3.2.1) und vorgegebene Merkmale trainiert werden können.

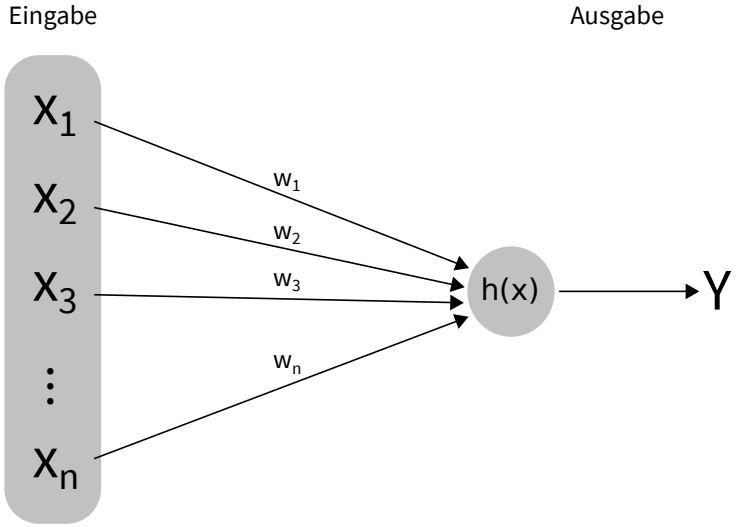


Abbildung 2.3: Modell der Logistischen Regression mit Eingabevektor X, logistischer Funktion $h(x)$ und Ausgabe Y

3 Neuronale Netzwerke: Architektur und Training

Im Allgemeinen lösen neuronale Netzwerke die gleiche Problemstellung wie die in Abschnitt 2.2 dargestellten Basisansätze (Bag-of-Words Modell, N-Gramm Modell, logistische Regression): das Finden einer repräsentativen Darstellung von Sprache. Der Mehrwert liegt hier in der hohen Genauigkeit der Sprachmodelle, ohne dass beim Training eine Lernvorgabe notwendig ist (unüberwachtes Lernen). Im Folgenden wird der Aufbau neuronaler Netzwerke und verschiedene Architekturen zum Modelltraining erläutert.

3.1 Aufbau neuronaler Netzwerke

In Analogie zum Nervensystem eines Organismus ist das Ziel eines künstlichen neuronalen Netzwerks, eingegebene Informationen (Trainingsdaten) derart zu abstrahieren, dass durch Verallgemeinerung (Modellierung) Aussagen über unbekannte Eingabedaten getroffen werden können.

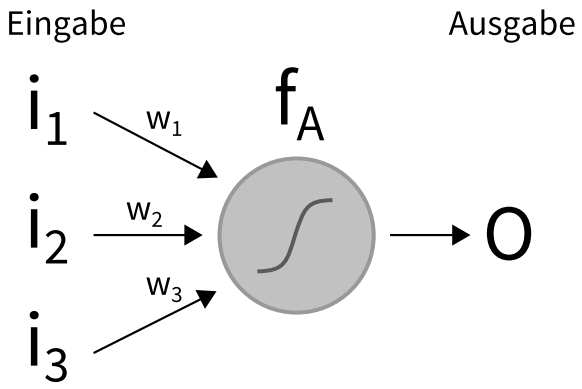
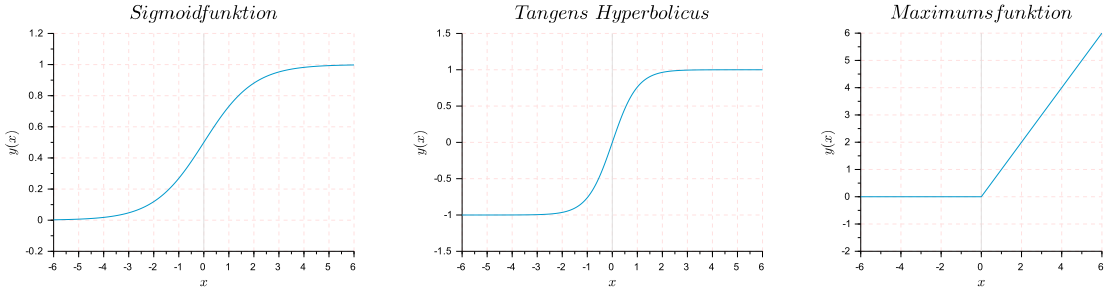


Abbildung 3.1: Künstliches Neuron mit der Eingabe I, der Gewichtung W, der Aktivierungsfunktion $f_A(I, W)$ und der Ausgabe O

Hauptbestandteil eines künstlichen neuronalen Netzwerks sind die Neuronen, welche das Lernen einer Funktion realisieren. In Abbildung 3.1 ist der allgemeine Aufbau eines Neurons dargestellt. Hier gibt es zunächst verschiedene Eingaben $I = (i_1...i_n)$, welche mit den Gewichten $W = (w_1...w_n)$ als Argumente für die Aktivierungsfunktion f_A dienen. Es ist entscheidend, dass die Aktivierungsfunktion eine nicht-lineare Funktion ist, da andernfalls bestimmte Kombinationen von Merkmalen nicht modelliert werden können (vgl. XOR-Problem[1]). Das errechnete Ergebnis O der Aktivierungsfunktion stellt die Ausgabe des Neurons dar (vgl. Gleichung 3.1).

$$O = f_A(I, W) \tag{3.1}$$

Oft werden als Aktivierungsfunktion die Sigmoidfunktion $\frac{1}{1+e^{-I \cdot W}}$, der Tangens Hyperbolicus $tanh(I \cdot W)$ oder die Maximumsfunktion der Form $max(0, I \cdot W)$ verwendet (vgl. Abbildung 3.2). Die Sigmoidfunktion und der Tangens Hyperbolicus bieten sich dann an, wenn eine differenzierbare Funktion mit begrenztem Wertebereich benötigt wird. Das geht mit der Maximumsfunktion nicht, hier wird auf 0 oder einen positiven reellen Wert abgebildet.



(a) Sigmoidfunktion $y(x) = \frac{1}{1+e^{-x}}$

(b) Tangens Hyperbolicus $y(x) = tanh(x)$

(c) Maximumsfunktion $y(x) = max(0, x)$

Abbildung 3.2: Mögliche Aktivierungsfunktionen eines Neurons

Das neuronale Netzwerk ergibt sich aus verschiedenen Ebenen von Neuronen, welche so miteinander verbunden sind, dass die Ausgabe eines Neurons als Eingabe für ein weiteres Neuron der nächsten Ebene dient (vgl. Abbildung 3.3). Allgemein gibt es drei Arten von Ebenen: Die Eingabe-Ebene, die verborgene Ebene und die Ausgabe-Ebene. Die verborgene Ebene repräsentiert dabei gelernte nicht-lineare

1 Beim XOR-Problem können die Ergebnismengen nicht durch einen lineraren Klassifikator separiert werden (vgl. Abbildung 2.1).

Kombinationen der Merkmale des Eingabe-Vektors, sodass damit jede mögliche Funktion darstellbar ist.

Findet die Informationsweitergabe, wie in Abbildung 3.3 dargestellt, nur in Verarbeitungsrichtung (vom Eingang zum Ausgang) statt, so spricht man von Feedforward-Netzwerken. Gibt es zusätzlich zur Vorwärtsrichtung auch rückgerichtete (*rekurrente*) Verbindungen, so spricht man von Recurrent-Netzwerken. Hier ist es durch (oft zeitversetzte) Rückkopplung möglich, Informationen aus vorangegangenen Verarbeitungsschritten wieder als Eingabeinformation zu verarbeiten (vgl. Abschnitt 3.3.2).

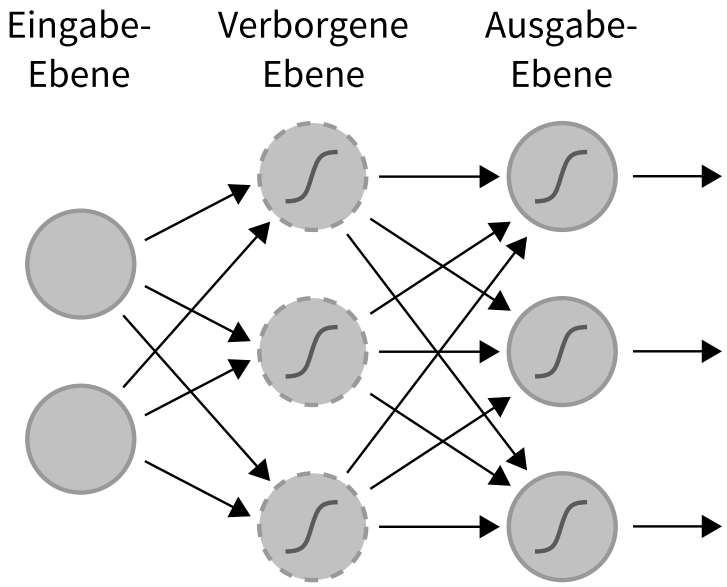


Abbildung 3.3: Künstliches neuronales Netzwerk mit einer verborgenen Ebene

3.2 Training von Modellen

Beim Training eines neuronalen Netzwerk-Modells geht es um das Erlernen einer spezifischen Funktion. Diese Funktion stellt im Allgemeinen die Abbildung bestimmter Merkmale aufgrund gegebener Eingabeinformationen dar. In der Regel geschieht dies durch Modifikation der Gewichtungen zwischen einzelnen Neuronen. Die Modifikation wird im Einzelnen von einem Lernalgorithmus bestimmt, welcher die Größe und Richtung der Gewichtsänderung angibt. Lernalgorithmen sind dabei zwei Kategorien zuzuordnen: überwachtes Lernen und unüberwachtes Lernen (vgl.

Abschnitt 2.2). Im Folgenden wird ein mögliches Lernverfahren zum Training eines neuronalen Netzwerks erläutert.

3.2.1 Fehlerminimierung: Gradientenabstieg

Zunächst ist der Gradientenabstieg eine Möglichkeit, ausgehend von einem Näherungswert Minimierungsprobleme zu lösen. Dabei ist das Ziel die Minimierung einer bestimmten Zielfunktion, welche sich aus der Summe differenzierbarer Funktionen ergibt.

Diese Zielfunktion beschreibt im Zusammenhang mit dem Training neuronaler Netzwerke den Fehler der Ausgabe der einzelnen Neuronen in Abhängigkeit ihrer Gewichtungen. Mithilfe des Gradientenabstiegs kann ausgehend von einem Punkt der Oberfläche der Fehlerfunktion durch iterative Gradientenberechnung solange in Richtung des stärksten Abstiegs gegangen werden, bis ein (lokales) Minimum gefunden wurde. Um den stärksten Abstieg in einem gegebenen Punkt möglichst genau berechnen zu können, werden normalerweise alle zur Verfügung stehenden Trainingsdaten in jeden Berechnungsschritt miteinbezogen. Das bedeutet allerdings einen hohen Aufwand der Gradientenberechnung und damit auch einen entsprechend hohen Aufwand beim Modelltraining.

Um diesen Aufwand zu reduzieren, gibt es das Gradientenverfahren *Stochastic Gradient Descent* (SGD). Die Besonderheit von SGD gegenüber anderen Gradientenverfahren ist, dass nicht alle Trainingsdaten komplett in die Gradientenberechnung der Zielfunktion einbezogen werden, sondern nur ein zufällig gewählter Datensatz[2] pro Iterationsschritt. Dadurch ist die Gradientenberechnung hier erheblich schneller. Außerdem hat SGD neben stabilerer Konvergenz auch den Vorteil, dass aufgrund seiner stochastischen Natur parallelisiert trainiert werden kann, was die Trainingsdauer zusätzlich deutlich verringert.

Abbildung 3.4 zeigt beispielhaft den Gradientenabstieg einer Funktion $J(\Theta_0, \Theta_1)$ mit den Parameter-Funktionen Θ_0 und Θ_1 bis zu einem lokalen Minimum.

2 Aus diesem Grund wurde der Name *stochastischer* Gradientenabstieg gewählt.

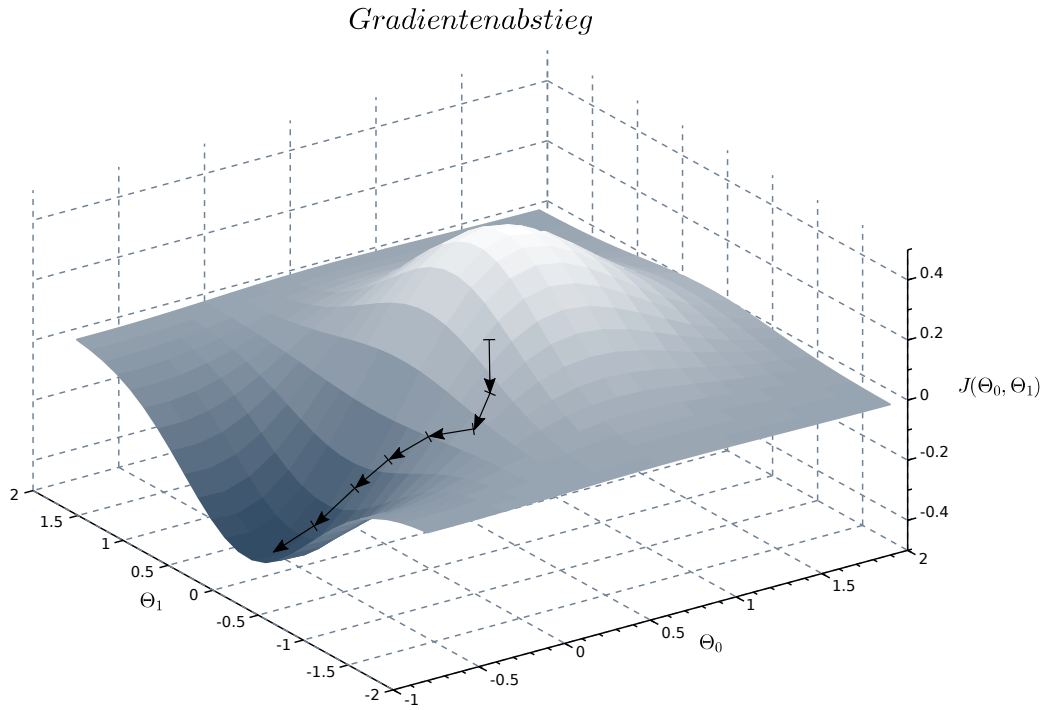


Abbildung 3.4: Beispielhafter Gradientenabstieg einer durch Veränderung der Parameter-Funktionen Θ_0 und Θ_1 zu minimierenden Funktion $J(\Theta_0, \Theta_1)$ von einem Ausgangspunkt bis zu einem (lokalen) Minimum

3.2.2 Lernverfahren: Backpropagation

Ziel beim Training neuronaler Netzwerke ist es, den Fehler der Ausgabe zu minimieren, welcher sich durch die Gewichtungen der Neuronenverbindungen ergibt. Backpropagation gehört zu den überwachten Lernverfahren und dient der Anpassung dieser Gewichtungen durch inverses Durchlaufen eines neuronalen Netzwerks. Das Schema der Backpropagation ist in Abbildung 3.5 dargestellt.

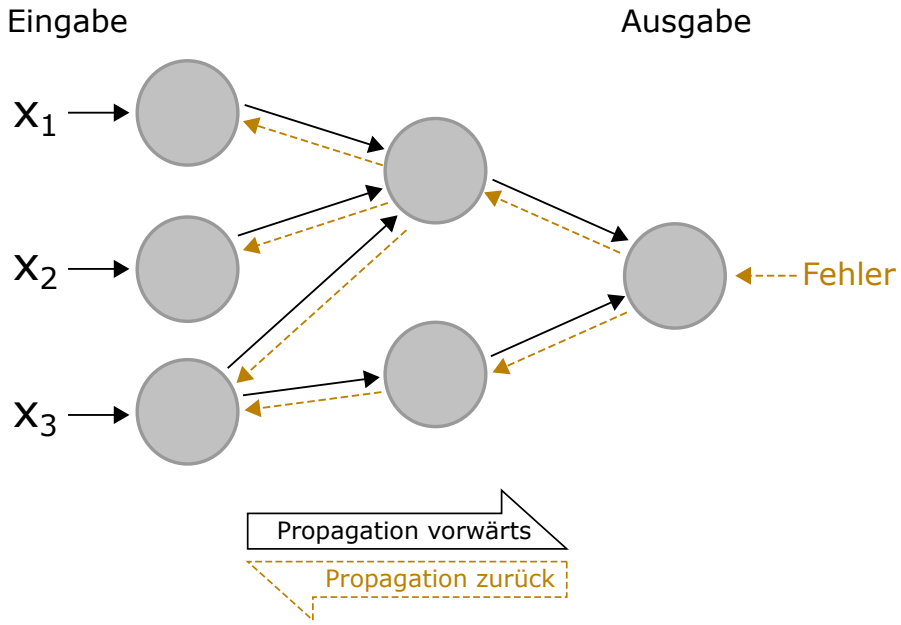


Abbildung 3.5: Backpropagation des bestimmten Fehlers zur Anpassung der Gewichte der Neuronenverbindungen

Zunächst wird ein Eingabe-Vektor am neuronalen Netzwerk angelegt und vorwärts propagiert, sodass ein Ausgabe-Vektor entsteht, der mit dem Ziel-Vektor verglichen werden kann. Aus der quadrierten Differenz ergibt sich der im Netzwerk entstandene Fehler, entsprechend folgender Formel[3]:

$$E_p = \frac{1}{2} \sum_{n=1}^{N} \left(t_{pn} - o_{pn}\right)^2 \tag{3.2}$$

Dabei beschreibt Gleichung 3.2 den Fehler E_p bezüglich eines einzelnen Eingabe-Vektors p von N vorhandenen Eingabe-Vektoren mithilfe des Zielwertes t_p und des aktuellen Ausgabewertes o_n. Der Gesamtfehler ergibt sich daher aus:

$$E = \sum E_p \tag{3.3}$$

Der errechnete Fehler wird nun von der Ausgabe-Ebene zur Eingabe-Ebene hin zurück propagiert, wobei die Gewichtungen der einzelnen Neuronenverbindungen

3 Die Formeln und das Prinzip der Backpropagation wurden von Rumelhart et al. erstmals 1986 vorgestellt [23].

je nach Einfluss auf den Fehler angepasst werden. Möglich ist das mithilfe des Gradientenabstiegs (wie z.B. SGD):

$$\Delta_p w_{ji} = \eta \delta_{pj} i_{pi} \tag{3.4}$$

Die Änderung der Gewichtung $\Delta_p w_{ji}$ zwischen dem i-ten und j-ten Neuron mit Schrittweite η wird dabei durch eine Verallgemeinerung der Delta-Regel beschrieben, wobei das Fehlersignal von Neuron j durch δ_{pj} und die Ausgabe von Neuron i durch i_{pi} dargestellt wird (vgl. Rumelhart et al. 1986 [23]).

Wird nun nach der Zurückpropagierung des Fehlers und Anpassung der Gewichte der Eingabe-Vektor erneut an das Netzwerk angelegt, so ergibt sich ein Ausgabe-Vektor, welcher dem Zielvektor ähnlicher ist. So ist es mithilfe von Backpropagation möglich, ein neuronales Netzwerk zu trainieren.

Beim Zurückpropagieren in sehr großen Netzwerken mit zufällig initialisierten Gewichtungen kann die Gradientenzerstreuung problematisch sein. Durch die hohe Anzahl an Ebenen verteilt sich der zurückpropagierte Fehler auf so viele mögliche Fehlerquellen (Gewichtungen), dass diese kaum mehr geändert werden. Dadurch findet in bestimmten Ebenen kein „Lernenprozess“ mehr statt. Eine Lösung für dieses Problem ist, die Gewichtungen durch ein unüberwachtes Pre-Training einzelner Ebenen zu initialisieren (Bengio et al. 2007 [5]).

3.3 Modelle und Architekturen

Mikolov et al., 2013 [14] erläutert verschiedene Modell-Architekturen basierend auf neuronalen Netzwerken wie Feedforward-Netzwerke (vgl. Abschnitt 3.3.1) und Recurrent-Netzwerke (vgl. Abschnitt 3.3.2). Diese repräsentieren Eingabedaten zwar sehr genau, weisen aber für eine große Menge an Trainingsdaten sehr lange Berechnungszeiten auf. Als Hauptursache für den sich ergebenden Aufwand benennen Mikolov et al. die nichtlineare verborgene Ebene der Modelle. Um den Aufwand der Berechnung während des Trainings zu minimieren, wurden zwei vereinfachte[4]

4 Die Vereinfachung besteht hier im Weglassen der verborgenen Ebene(n) aus dem neuronalen Netzwerk Modell, welche den Hauptteil der Trainingskomplexität ausmacht.

Modell-Architekturen zum Erlernen von Wort-Vektoren vorgestellt: Continuous-Bag-of-Words (CBOW) und Skip-Gram. Diese Architekturen repräsentieren die Eingabedaten zwar weniger präzise, dafür ermöglichen sie aber eine effizientere Verarbeitung der Trainingsdaten und damit die Verarbeitung größerer, webbasierter Korpora. In den folgenden Unterabschnitten werden die eben genannten Modell-Architekturen nach Mikolov et al., 2013 [14] vorgestellt, basierend auf Bengio et al., 2003 [4] und Mikolov et al., 2010 [12].

3.3.1 Feedforward-Netzwerk Modell

Das *Feedforward Neural Net Language Model* (NNLM) besteht aus Eingabe-Ebene, Projektionsebene, verborgener Ebene und Ausgabe-Ebene, welche nur in eine Richtung von der Eingabe- zur Ausgabe-Ebene miteinander verbunden sind (vgl. Abbildung 3.3). In der Eingabe-Ebene werden N vorangehende Wörter One-Hot-kodiert, sodass sie die Größe V des Vokabulars aufweisen. Anschließend findet mithilfe einer gemeinsamen Projektionsmatrix eine Projektion von der Eingabe-Ebene zur Projektionsebene mit der Dimension $N \times D$ statt, wobei D die Größe der Merkmalsvektoren bezeichnet. Die verborgene Ebene mit der Dimension H dient hierbei der Berechnung der Wahrscheinlichkeitsverteilung über alle Wörter des Vokabulars, sodass die Ausgabe-Ebene die Dimension V aufweist. Mikolov et al. geben folgenden Aufwand des Trainings an:

$$Q = N \times D + N \times D \times H + H \times V \tag{3.5}$$

Der Aufwand des dominierenden Terms $H \times V$ kann durch Trainingsoptimierungen wie z.B. Hierarchical Softmax (vgl. Abschnitt 3.4.1) auf $H \times \log_2(V)$ reduziert werden, da in diesem Fall das Vokabular als Binärbaum modelliert wird. Weiterhin wird der Gesamtaufwand von dem Term $N \times D \times H$ bestimmt. Hier ist ersichtlich, dass der Aufwand von der Dimension der verborgenen Ebene H abhängt.

3.3.2 Recurrent-Netzwerk Modell

Das *Recurrent Neural Net Language Model* (RNNLM) wurde als Verbesserung des NNLM vorgestellt, da RNNLM nach Mikolov et al. in der Theorie komplexere

Muster effizienter repräsentieren können. Das RNNLM besteht aus Eingabe-Ebene, verborgener Ebene und Ausgabe-Ebene. Statt einer Projektionsebene verfügt die verborgene Ebene hier über zeitverzögernde Verbindungen (vgl. Abbildung 3.6). Dadurch können Informationen aus vorangegangenen Zeitschritten in das aktuelle Training miteinbezogen werden. Mikolov et al. geben folgenden Aufwand des Trainings an:

$$Q = H \times H + H \times V \tag{3.6}$$

Wie beim Feedforward-Netzwerk kann auch hier $H \times V$ zu $H \times \log_2(V)$ durch Binärbaum-Repräsentation des Vokabulars optimiert werden, wodurch der Aufwand maßgeblich von $H \times H$, also erneut der Dimension der verborgenen Ebene, bestimmt wird.

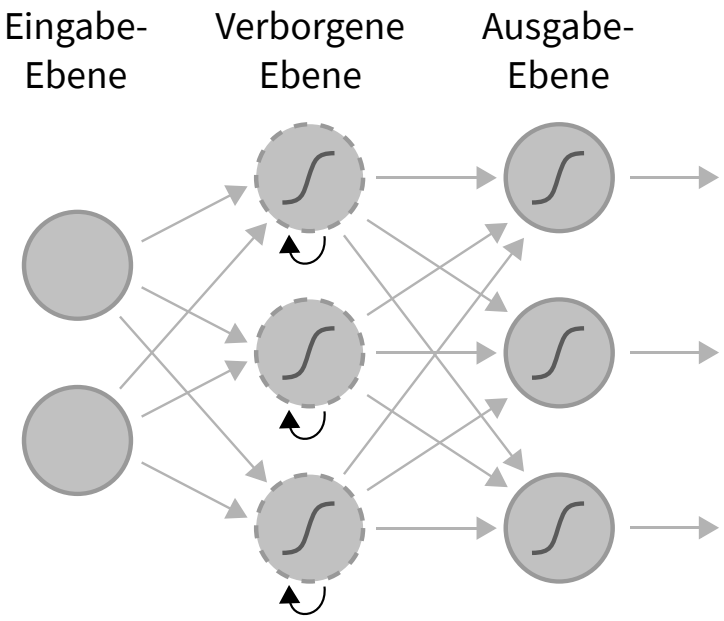


Abbildung 3.6: Architektur des Recurrent-Netzwerk Modell

3.3.3 Continuous-Bag-of-Words Architektur

Ziel der CBOW Architektur ist es, die Vektor-Repräsentation eines Wortes mit geringerem Aufwand als NNLM und RNNLM zu ermitteln. Die Merkmale eines Wortes

sollen hier nicht mehr mithilfe verborgener Ebenen des neuronalen Netzwerks, sondern durch den Kontext repräsentiert werden, in dem das Wort steht. Die Architektur ist in Abbildung 3.7 dargestellt.

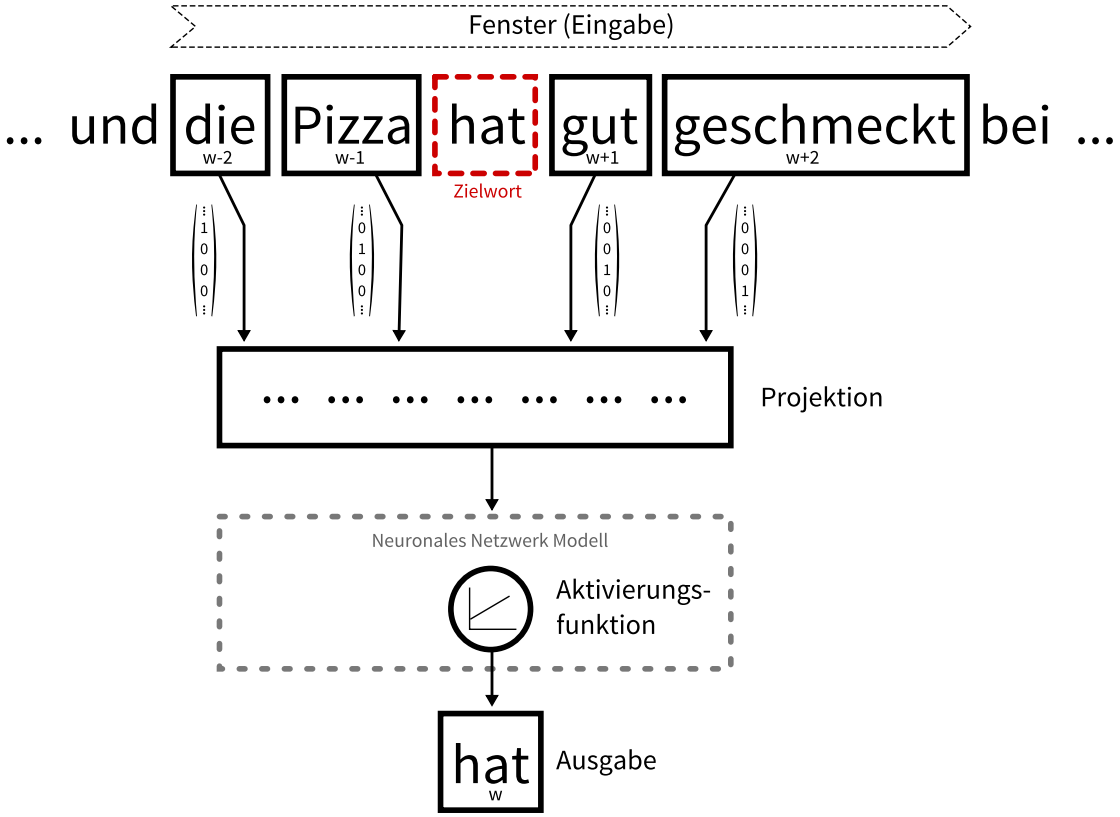


Abbildung 3.7: CBOW-Architektur mit einer Fenstergröße von 2: Lernen des Wort-Vektors zum Wort „hat“ anhand eines Beispielkontextes.

Der Kontext wird durch ein Eingabe-Fenster beschrieben, dessen Breite die Wortanzahl vor und hinter dem Zielwort vorgibt. Die Vektoren der Kontext-Wörter ergeben sich initial aus einer einfachen Repräsentation (wie beispielsweise die One-Hot Kodierung, vgl. Abschnitt 2.2.1) und werden in der Projektionsebene zu einem Vektor vereint (z.B. durch Summen- oder Durchschnittsbildung), welcher anschließend als Eingabe für ein neuronales Netzwerk ohne verborgene Ebene dient. In diesem Modell werden die Gewichte der Neuronenverbindungen trainiert, indem auf den entstehenden Vektor die Aktivierungsfunktion angewendet wird. Anhand des Ergebnisses kann entschieden werden, ob dieser das Zielwort repräsentiert. Falls dies nicht der Fall ist, werden die Gewichte durch Backpropagation (vgl. Abschnitt 3.2.2)

angepasst. Mit dieser Architektur ist demnach ein unüberwachtes Lernen möglich, indem Wörter durch ihren Kontext beschrieben werden.

Durch den Entfall einer verborgenen Ebene ergibt sich nach Mikolov et al. für das CBOW Modell der in Gleichung 3.7 dargestellte Aufwand, welcher deutlich geringer ist als bei NNLM oder RNNLM.

$$Q = N \times D + N \times \log_2(V) \tag{3.7}$$

3.3.4 Skip-Gram Architektur

Wie bereits bei CBOW nutzt die Skip-Gram Architektur die Tatsache aus, dass im Normalfall dicht beieinander stehende Wörter eher miteinander korrelieren als weit von einander entfernte. Anders als CBOW sucht Skip-Gram allerdings Vektor-Repräsentationen von Wörtern, welche dazu geeignet sind, die Wörter im jeweiligen Kontext vorherzusagen. Dazu dient jedes Wort als Eingabe eines log-linearen Klassifizierers, welcher in der Projektionsebene die Wahrscheinlichkeiten der vorhergehenden und nachfolgenden Wörter errechnet. Gleichung 3.8 stellt die durchschnittliche logarithmische Wahrscheinlichkeit für eine Folge von Trainingsworten $w_1, w_2, \cdots, w_T$ dar, welche Skip-Gram maximieren möchten:

$$\frac{1}{T} \sum_{t=1}^{T} \sum_{-C \leq j \leq C} \log p\left(w_{t+j} | w_t\right), j \neq 0 \tag{3.8}$$

Dabei bezeichnet C die Anzahl von Wörtern vor und nach dem gegebenen Wort w_t (Fensterbreite). Mikolov et al. haben gezeigt, dass größere C zwar mehr Trainings-Beispiele liefern und folglich zu höherer Genauigkeit der Wort-Vektoren führen können, allerdings auf Kosten von Rechenkomplexität und Trainingszeit.

Der Aufwand des Trainierens mittels Skip-Gram wird von Mikolov et al. wie folgt angegeben:

$$Q = C \times (D + D \times \log_2(V)) \tag{3.9}$$

3.4 Optimierungen

Im Folgenden werden Methoden zur Optimierung des Rechenaufwands des Trainings und der Verbesserung trainierter Sprachmodelle vorgestellt.

3.4.1 Hierarchical Softmax

Für Modelle basierend auf neuronalen Netzwerken muss die bedingte Wahrscheinlichkeit aller Wörter aus dem Vokabular aufgrund eines gegebenen Kontextes berechnet werden. Anschließend wird eine Normalisierung (engl. *Softmax*) durchgeführt, um den Einfluss von Extremwerten oder Ausreißern zu vermindern, ohne diese zu entfernen. Diese Operationen sind sehr rechenaufwändig, da sie das gesamte Vokabular betreffen.

Hierarchical Softmax wurde in seinem Grundkonzept 2005 von F. Morin und Y. Bengio vorgestellt [18][5] und stellt eine recheneffiziente Möglichkeit der Berechnung einer solchen Normalisierung dar. Hier werden alle Wörter des Vokabulars als Blätter eines Binärbaums repräsentiert, welcher mittels Huffman-Kodierung[6] konstruiert wird. Basierend auf der jeweiligen Worthäufigkeit besitzt jede Verbindung zweier Knoten eine bestimmte Wahrscheinlichkeit. Für jedes Wort existiert jetzt genau ein Pfad von der Wurzel zum Blatt, welcher genutzt werden kann, um die Wahrscheinlichkeit eines Wortes durch Multiplikation der Teil-Wahrscheinlichkeiten zu schätzen. Somit wird der Aufwand des Trainings von $O(V)$ auf $O(\log(V))$ reduziert, wobei V die Anzahl der Wörter im Vokabular bezeichnet.

In Abbildung 3.8 ist die Berechnung der Wahrscheinlichkeit beispielhaft für ein Wort w_2 dargestellt. Der Pfad zu w_2 besteht aus 3 Kanten von der Wurzel n_0 über die Knoten n_1 und n_3, für welche jeweils die Wahrscheinlichkeit in Abhängigkeit vorheriger Kanten berechnet werden kann. Dabei bezeichnet p_{n_k} die Wahrscheinlichkeit des k-ten Knotens, p_{w_i} die Wahrscheinlichkeit des i-ten Wortes und P_{L,n_k} die Wahrscheinlichkeit, dass das Wort ausgehend vom Knoten n_k links liegt.

5 Die Ausführungen in diesem Abschnitt beziehen sich jedoch auf einen wissenschaftlichen Artikel von Xin Rong [21].

6 Von David A. Huffman 1952 entwickelte Kodierungsform, bei welcher jedem Element ein Codewort variabler Länge zugeordnet wird, um dieses möglichst redundanzfrei abzubilden.

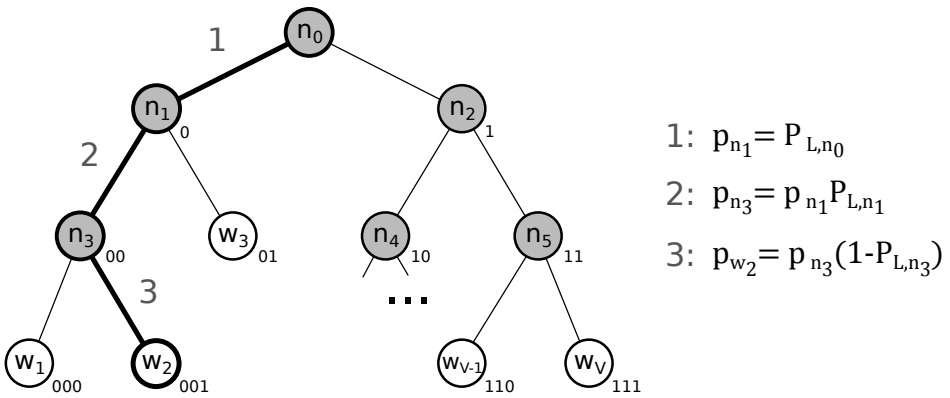


Abbildung 3.8: Hierarchical Softmax: Huffmann-Binärbaum mit einem Beispielpfad für die Wahrscheinlichkeitsberechnung von Wort w_2 eines Vokabulars der Größe V.

3.4.2 Negative Sampling

Neben Hierarchical Softmax ist Negative Sampling eine Möglichkeit der Qualitätsverbesserung trainierter Modelle. Nach Mikolov et al. [13] dient Negative Sampling der Modellverbesserung, indem nicht nur korrekte Fälle aus den Trainingsdaten, sondern auch bewusst nicht in den Trainingsdaten vorkommende, falsche Beispiele (*negative samples*) generiert und beim Training mit einbezogen werden[7]. Diese Art von künstlicher Varianz beim Training sorgt dafür, dass mit Negative Sampling trainierte Modelle weniger fehleranfällig sind und somit bessere Evaluierungsergebnisse liefern. Der modellverbessernde Effekt von Negative Sampling hängt dabei stark von der Anzahl der jeweils verwendeten Negative Samples ab.

7 Da in dieser Arbeit Sprachmodelle auf Textkorpora trainiert werden, sind „negative samples“ hier zufällig aus dem Vokabular gewählte Wörter.

4 Methodik und Implementierung

Nachdem in den vorangegangen Kapiteln die theoretische Grundlage für die Methodik und Umsetzung in NLP allgemein und für diese Arbeit im einzelnen gelegt wurde, präsentiert dieses Kapitel die speziell für diese Arbeit entwickelte und verwendete Methodik. Dies beinhaltet die praktische Umsetzung des Trainierens deutscher Wort-Vekoren: den Bezug und die Erstellung von Korpora und den Aufbau eines Toolkits zum Training und anschließender Evaluierung.

4.1 Erstellung des Korpus

Zum Trainieren von Sprachmodellen sind große Mengen von Trainingsdaten in Textform notwendig. Da es in dieser Arbeit um die Analyse deutscher Sprachmodelle geht, werden deutsche Texte benötigt, welche thematisch vielfältig, dabei jedoch nicht umgangssprachlich sind.

Eine Möglichkeit zum Bezug öffentlicher deutscher Trainingsdaten ist die deutsche Wikipedia. Mit über 1,8 Millionen Artikeln[1] stellt die deutsche Wikipedia eine große Anzahl von Wörtern in natürlicher deutscher Sprache zur Verfügung. Artikelinhalte können per Wikipedia Datenbank Backup Dump[2] bezogen werden. Dieser Dump besteht aus einer komprimierten XML-Datei, welche die Artikel jeweils als einzelne XML-Objekte enthält. Ein solches XML-Objekt besteht neben dem eigentlichen Artikelinhalt auch aus Metainformationen wie Titel, Zeitstempel und Autor. Außerdem ist der Artikelinhalt in MediaWiki Markup[3] notiert, um Textformatierungen und das Einfügen von Tabellen, Formeln, Bildern und HTML zu ermöglichen. Für das Modelltraining müssen die Trainingsdaten in reiner Textform vorliegen, da andernfalls die durch die Notation verfälschten Wörter gelernt werden. Aus diesem Grund müssen sowohl die XML-Tags als auch das MediaWiki Markup entfernt werden.

1 Offizielle Wikipedia-Statistik zur Artikelanzahl (o.J.), URL: http://stats.wikimedia.org/DE/TablesArticlesTotal.htm (Stand: 08.05.2015)

2 Dateiliste deutscher Wikipedia Dumps (o.J.), URL: http://dumps.wikimedia.org/dewiki/latest/ (Stand: 08.05.2015)

3 Formatierungen mittels MediaWiki Markup (o.J.), URL: http://www.mediawiki.org/wiki/Help:Formatting

Der *Wikipedia Extractor* ist ein von Giuseppe Attardi (Universität von Pisa) geschriebenes freies Python Script[4], das die beschriebene Filterung des Wikipedia Dumps bewerkstelligt.

Außerdem gibt es eine zum Buch „Statistical Machine Translation“ von Philipp Koehn gleichnamige Website[5], auf welcher für die automatisierte Übersetzung von Texten öffentliche Trainingsdaten in Form von einsprachigen Nachrichtenartikeln zur Verfügung gestellt werden. Diese sind jahrweise komprimiert und für die deutsche Sprache von 2007 bis 2013 vorhanden[6].

In dieser Arbeit wird entsprechend der Trainingsumgebung (vgl. Abschnitt 4.3.3) der Korpus für die Modellierung aus dem Wikipedia Dump der deutschen Wikipedia und deutschen Nachrichtenartikeln aus dem Jahr 2013 bestehen. Dieser Korpus enthält insgesamt über 1,1 Milliarden Wörter in knapp 60 Millionen Sätzen und ist 7,49 GB groß.

In Tabelle 4.1 sind die einzelnen statistischen Daten der Teilkorpora aufgelistet. Das Erkennen von Sätzen und Wörtern (Tokenisierung) wird im folgenden Abschnitt erläutert.

Korpus	Wortanzahl	Satzanzahl	Vokabulargröße	Korpusgröße
Wikipedia	579.046.421	24.656.073	14.418.176	3,95 GB
Nachrichten	532.348.362	35.014.404	9.256.424	3,54 GB
Gesamt	1.111.394.783	59.670.477	20.705.362	7,49 GB

Tabelle 4.1: Korpus Statistik des deutschen Wikipedia Dumps 2015 und deutscher Nachrichtenartikel von 2013.

4.2 Korpus Preprocessing

Beim Preprocessing wird der Korpus für das Training vorbereitet und bearbeitet, um bei der Evaluation der trainierten Sprachmodelle festzustellen, ob diese ver-

4 Dokumentation und Download des *Wikipedia Extractor* unter GPL3 Lizenz (o.J.), URL: http://medialab.di.unipi.it/wiki/Wikipedia_Extractor

5 Koehn, Philipp: Statistical Machine Translation (o.J.), URL: http://www.statmt.org/ (Stand: 08.05.2015)

6 Dateiliste verschiedensprachiger Nachrichtenartikel aus den Jahren 2007 bis 2013 (o.J.), URL: http://www.statmt.org/wmt14/training-monolingual-news-crawl/ (Stand: 08.05.2015)

bessert wurden oder nicht. Mithilfe des *Natural Language Toolkit* (NLTK[7]) werden die durch den Korpus zur Verfügung stehenden Textdaten in einzelne Sätze und diese wiederum in einzelne Wort-Tokens zerlegt. Diese Tokenisierung geschieht mit der `word_tokenize()`-Funktion von NLTK, basierend auf einem für deutsche Tokenisierung vortrainierten Modell[8].

Anschließend ist es möglich, einzelne Tokens auszuschließen bzw. zu normalisieren, um Aussagen darüber treffen zu können, ob derartiges Preprocessing Sprachmodelle verbessern kann.

Zunächst kann die Interpunktion entfernt werden. Dazu steht die folgende im DIMA-Projekt (Arras et al. [1]) verwendete Zeichenliste zur Verfügung:

```
. .. ... , ; : ( ) " ‘ [ ] { } ? ! - – + * -- “ ”
```

Neben der Interpunktion können auch Stoppwörter entfernt werden. Stoppwörter sind Wörter, welche sehr oft im Korpus auftreten, dabei aber kaum einen inhaltlichen Mehrwert bieten. Deutsche Stoppwörter sind z.B. bestimmte Artikel („der“, „die“, „das“) oder Konjunktionen („und“, „oder“). Listing A.1 zeigt die in dieser Arbeit verwendete Liste deutscher Standard-Stoppwörter von NLTK.

Darüber hinaus ist eine Normalisierung des Korpus bezüglich deutscher Umlaute und ‚ß‘ möglich. Dazu werden die Umlaute in ihre entsprechende Digraph-Repräsentation umgewandelt (ä $\rightarrow$ ae, ß $\rightarrow$ ss etc.). Diese Normalisierung stellt sicher, dass die Umlaute im Korpus einheitlich dargestellt werden und wurde in das Preprocessing in dieser Arbeit mit einbezogen.

Schließlich können Bigramme gebildet werden. Bigramme sind Tokens, bestehend aus zwei Wort-Tokens, welche häufig gemeinsam (hintereinander) auftreten (z.B. „Angela“, „Merkel“ $\rightarrow$ „Angela_Merkel“). Semantische Zusammenhänge sollen dadurch deutlich besser dargestellt werden können. Nach Mikolov et al. [13] werden Bigramme aufgrund der Anzahlen der beiden einzelnen Wörter w_i und w_j und deren gemeinsamen Vorkommen $w_i w_j$ nach folgender Formel gebildet (das δ ist

7 NLTK ist eine Sammlung von Bibliotheken und Programmen in Python zur Verarbeitung natürlicher Sprache wie Klassifizierung, Tokenisierung und Tagging. Natural Language Toolkit (o.J.), URL: http://www.nltk.org

8 Dieses Modell wurde auf der Neuen Züricher Zeitung trainiert und enthält 847k Tokens. Text Mining online (o.J.), URL: http://textminingonline.com/tag/nltk-word-tokenize (Stand: 29.06.2015)

dabei ein Koeffizient zur Vermeidung von Bigrammen zu selten auftretender Wörter):

$$score(w_i, w_j) = \frac{count(w_i w_j) - \delta}{count(w_i) \times count(w_j)} \tag{4.1}$$

Das Ergebnis $score(w_i, w_j)$ wird dabei größer, je öfter zwei Wörter gemeinsam und je seltener beide Wörter einzeln auftreten. Dadurch werden die Wörter „Angela Merkel“ zu einem Bigram, die Wörter „das ist“ aber nicht. Liegt das Ergebnis über einem zuvor festgelegtem Schwellenwert, so werden beide Wörter zu einem Bigramm verbunden. In dieser Arbeit wird dafür der Standardwert der Implementierung von NLTK verwendet.

Tabelle 4.2 zeigt einige Statistiken zu den einzelnen Schritten des Preprocessing.

Preprocessing	Wortanzahl	Vokabulargröße	Datenmenge
RAW	1.111.394.783	20.705.362	7,49 GB
PS	702.510.144	13.197.231	5,65 GB
PSU	714.311.955	13.160.241	5,72 GB
PSUB	651.219.519	13.757.051	5,69 GB

Tabelle 4.2: Statistiken zum Korpus Preprocessing
RAW = kein Preprocessing
PS = Interpunktions-, Stoppwortfilterung
PSU = Interpunktions-, Stoppwortfilterung, Umlaut-Normalisierung
PSUB = Interpunktions-, Stoppwortfilterung, Umlaut-Normalisierung, Bigramme

4.3 Modell Training

Um Modelle auf natürlicher Sprache zu trainieren, werden in dieser Arbeit verschiedene Methoden gezeigt. Insbesondere CBOW und Skip-Gram sind Beispiele für Architekturen, die auf neuronalen Netzwerken ohne verborgene Ebene basieren, um die Rechenkomplexität zu reduzieren und die Menge verarbeitbarer Trainingsdaten zu vergrößern. Das dafür im Rahmen dieser Arbeit entwickelte Toolkit und eine bereits existierende Implementierung dieser beiden Architekturen wird im Folgenden beschrieben und später für das Modelltraining und die anschließende Evaluation verwendet.

4.3.1 Software Toolkit

Das für Forschungszwecke frei verfügbares Open Source Projekt *word2vec* ist ein auf der Arbeit von Mikolov et. al [14] basierendes Software Tool. Es stellt eine effiziente Implementierung der CBOW und Skip-Gram Architektur in C dar[9]. Darauf aufbauend wurden die Trainingsalgorithmen von *word2vec* nach Python übertragen, in die Python Bibliothek *gensim* aufgenommen und mit zusätzlichen Funktionalitäten (wie z.B. Funktionen zum Vergleich der Wort-Vektoren und Bewertung der Modellgenauigkeit) erweitert[10]. Aufgrund dieser Zusatzfunktionen und der Geschwindigkeitsoptimierungen[11] beim Training mit *gensim*, wird das Tool zum Modelltraining in dieser Arbeit verwendet.

4.3.2 Mögliche Parameterkonfiguration

Beim Trainieren mit *gensim* können die im Folgenden beschriebenen Parameter konfiguriert werden[12].

Als **Modell Architektur (A)** kann für das Modelltraining entweder Skip-Gram oder CBOW verwendet werden. Dabei verfügbare Optimierungen sind **Hierarchical Softmax (HS)** und **Negative Sampling (NS)**, wobei für Negative Sampling die Anzahl der „künstlichen Wörter“ (engl. *negative samples*) spezifiziert werden kann. Die **Vektorgröße (D)** bezeichnet die Dimension der Wort-Vektoren, die **Fensterbreite (N)** ist der Maximalabstand des aktuellen Wortes zum vorhergesagten Wort innerhalb eines Satzes. Ein Wort wird nur dann in das Training mit einbezogen, wenn es mindestens so häufig wie die von der **Worthäufigkeit (R)** festgelegte Mindestanzahl (absoluter Zahlenwert) auftritt. Andernfalls tritt das Wort zu selten auf und wird ignoriert, da in diesem Fall ein zu wenig aussagekräftiger Wort-Vektor entstehen würde. Wird CBOW als Architektur gewählt, so kann die **Projektionsart (P)** spezifiziert werden. Diese legt fest, ob aus den Kontext-Vektoren bei der Projektion die Summe oder der Durchschnitt gebildet wird. Die für das parallelisierte Training

9 word2vec Tool (o.J.), URL: https://code.google.com/p/word2vec/ (Stand: 22.06.2015)
10 Rehurek, Radim: Deep learning with word2vec (o.J.), URL: https://radimrehurek.com/gensim/models/word2vec.html (Stand: 22.06.2015)
11 Rehurek, Radim: Machine Learning Blog. (o.J.), URL: http://radimrehurek.com/2013/09/word2vec-in-python-part-two-optimizing/ (Stand: 22.06.2015)
12 In Klammern stehen jeweils die in Tabelle 4.3 zur Parameter-Spezifikation verwendeten Kürzel.

verwendete **Anzahl der Threads** beschleunigt das Training erheblich bei Mehrkern-Prozessoren. Um die Trainingszeit zu minimieren sollte hier die maximale Anzahl der zur Verfügung stehenden Prozessorkerne ausgenutzt werden, in dieser Arbeit 4 (vgl. Abschnitt 4.3.3). Das in Abschnitt 4.2 beschriebene **Preprocessing (P)** beinhaltet die Filterung der Interpunktion `P` und Stoppwörter `S`, die Umwandlung der Umlaute in ihre entsprechende Digraph-Repräsentation `U` oder das zusätzliche Bilden von Bigrammen `B`.

Durch Kombination der unterschiedlichen Parameter ergeben sich die in Tabelle 4.3 dargestellten Parameter-Spezifikationen der trainierten Modelle, unterteil in 8 Evaluationsgruppen.

4.3.3 Trainingsumgebung

Alle Modelle wurden auf einem Desktop-Rechner unter Ubuntu 14.04 mit 16 GB RAM trainiert. Dabei kam eine 4-Kern AMD A10-7850K Radeon R7 CPU zum Einsatz, welche mit 3.7 GHz taktet. Korpora und Modell wurden dabei auf einer SSD-Festplatte gespeichert.

4.4 Evaluation der Wort-Vektoren

Um die trainierten Modelle miteinander vergleichen zu können, ist es notwendig, diese mithilfe geeigneter Test-Sets zu evaluieren. Anhand der Testergebnisse kann anschließend entschieden werden, welches Modell und damit welche Parameter-Konfiguration die besten Wort-Vektoren erzeugt hat.

Angelehnt an Mikolov et al. [15] werden in dieser Arbeit die Modelle durch Test-Sets mit Analogiefragen der Form „`a` verhält sich zu `b`, wie `c` zu `d`“ zum einen zur Syntax und zum anderen zur Semantik evaluiert[13]. Praktisch bedeuten Analogiefragen das Ausführen einer Vektoraddition der entsprechenden Wort-Vektoren, z.B.:
`vector(König) - vector(Mann) + vector(Frau)` $\approx$ `vector(Königin)`
Dabei ergibt sich aus der linken Seite der Gleichung ein Vektor, für welchen mithilfe

13 Zu beachten ist dabei, dass die Modelle zur Nutzung großer Korpora unüberwacht trainiert wurden, sodass diese Fragen nicht Teil des Trainings waren.

Modellname	A	Pr	D	N	HS	NS	R	P
SG-52-5-133M	Skip-Gram	—	52	5	Ja	Nein	5	—
SG-52-5-266M	Skip-Gram	—	52	5	Ja	Nein	5	—
SG-52-5-530M	Skip-Gram	—	52	5	Ja	Nein	5	—
SG-52-5-1B	Skip-Gram	—	52	5	Ja	Nein	5	—
SG-52-5-RAW	Skip-Gram	—	52	5	Ja	Nein	5	—
SG-52-5-PS	Skip-Gram	P, S	52	5	Ja	Nein	5	—
SG-52-5-PSU	Skip-Gram	P, S, U	52	5	Ja	Nein	5	—
SG-52-5-PSUB	Skip-Gram	P, S, U, B	52	5	Ja	Nein	5	—
SG-52-5	Skip-Gram	P, S, U, B	52	5	Ja	Nein	5	—
SG-52-10	Skip-Gram	P, S, U, B	52	10	Ja	Nein	5	—
SG-52-15	Skip-Gram	P, S, U, B	52	15	Ja	Nein	5	—
SG-52-20	Skip-Gram	P, S, U, B	52	20	Ja	Nein	5	—
SG-52-5-R10	Skip-Gram	P, S, U, B	52	5	Ja	Nein	5	—
SG-100-5-R10	Skip-Gram	P, S, U, B	100	5	Ja	Nein	5	—
SG-200-5-R10	Skip-Gram	P, S, U, B	200	5	Ja	Nein	5	—
SG-300-5-R10	Skip-Gram	P, S, U, B	300	5	Ja	Nein	5	—
CB-52-5	CBOW	P, S, U, B	52	5	Ja	Nein	5	$\sum$
CB-52-10	CBOW	P, S, U, B	52	10	Ja	Nein	5	$\sum$
CB-52-15	CBOW	P, S, U, B	52	15	Ja	Nein	5	$\sum$
CB-52-20	CBOW	P, S, U, B	52	20	Ja	Nein	5	$\sum$
SG-52-5	Skip-Gram	P, S, U, B	52	5	Ja	Nein	5	—
SG-52-5-NS10	Skip-Gram	P, S, U, B	52	5	Ja	10	5	—
SG-52-5-NS20	Skip-Gram	P, S, U, B	52	5	Ja	20	5	—
SG-52-5-NS30	Skip-Gram	P, S, U, B	52	5	Ja	30	5	—
SG-52-5	Skip-Gram	P, S, U, B	52	5	Ja	Nein	5	—
SG-52-5-R10	Skip-Gram	P, S, U, B	52	5	Ja	Nein	10	—
SG-52-5-R20	Skip-Gram	P, S, U, B	52	5	Ja	Nein	20	—
SG-52-5-R50	Skip-Gram	P, S, U, B	52	5	Ja	Nein	50	—
CB-52-5-SUM	CBOW	P, S, U, B	52	5	Ja	Nein	5	$\sum$
CB-52-5-MEAN	CBOW	P, S, U, B	52	5	Ja	Nein	5	Ø
SG-52-5-HS	Skip-Gram	P, S, U, B	52	5	Ja	Nein	5	—
SG-52-5-NOHS	Skip-Gram	P, S, U, B	52	5	Nein	Nein	5	—

Tabelle 4.3: Parameter-Spezifikationen trainierter Modelle zur Architektur *A*, Preprocessing *Pr*, Vektordimension *D*, Fensterbreite *N*, Hierarchical Softmax *HS*, Negative Sampling *NS*, Worthäufigkeit *R* und Projektionsart *P*

der Kosinusähnlichkeit geprüft werden kann, welcher Vektor der Ähnlichste im Umfeld ist. Dieser stellt dann die Antwort der Analogiefrage dar. Ist es der Vektor des in der Frage angegebenen Wortes (im Beispiel der Vektor des Wortes „Köni-

gin“), so wurde die Frage korrekt beantwortet. Lag der Ziel-Vektor unter den 10 ähnlichsten Vektoren, so war die korrekte Antwort auf die Frage unter den Top 10 Ergebnissen.

Teil dieser Arbeit ist die Erstellung entsprechender deutscher Test-Sets, welche online zum freien Download zur Verfügung stehen[14]. Die Fragen der erstellten Test-Sets enthalten dabei keine Bigramm-Tokens, um auch Modelle, welche keine Bigramm-Tokens enthalten, mit den gleichen Fragen evaluieren zu können.

4.4.1 Syntaktisches Test-Set

Um syntaktische Zusammenhänge in der deutschen Sprache abfragen zu können, werden Analogiefragen zu unterschiedlichen Formen von Nomen, Adjektiven und Verben verwendet. Dazu wird eine Liste von jeweils 100 gängigen Wörtern zu jeder der drei Wortarten erstellt[15] und durch die nachfolgend beschriebenen Formen ergänzt.

Im Einzelnen sind im syntaktischen Test-Set enthalten: 100 verschiedene Nomen im Singular (`SI`) und Plural (`PL`), 100 verschiedene Adjektive in der Grundform (`GR`), im Komparativ (`KOM`) und im Superlativ (`SUP`) und 100 verschiedene Verben im Infinitiv (`INV`), in der 1. Person Singular Präsens (`1SP`), der 2. Person Plural Präsens (`2PP`), der 3. Person Singular Präteritum (`3SV`) und der 3. Person Plural Präteritum (`3PV`).

Die Analogiefragen werden systematisch generiert, indem jedes der jeweils 100 Wörter mit 5 weiteren, zufällig ausgewählten Wörtern der gleichen Form kombiniert wird. Die möglichen Kombinationen verschiedener Formen (im Folgenden *Muster* genannt) sind in Tabelle 4.4 gezeigt. Das Test-Set zur Syntax enthält demnach 20 Muster mit 500 Fragen pro Muster, also 10k Fragen insgesamt. Listing A.2 zeigt einen Auszug des Test-Sets.

14 Müller, Andreas: GermanWordEmbeddings (o.J.), URL: http://devmount.github.io/GermanWordEmbeddings (Stand: 22.06.2015)

15 Gebräuchliche Wörter sind beispielsweise bei Angeboten für Deutsch-Lernende zu finden. Die Wörter der Listen dieser Test-Sets entstammen *The German Professor* (o.J.), URL: http://www.thegermanprofessor.com/top-100-german-verbs/ (Stand: 18.05.2015) und http://www.thegermanprofessor.com/top-500-german-words/ (Stand: 18.05.2015)

Wortart	Beziehung	Muster	# Fragen	Beispiel
Nomen	Singular/ Plural	SI/PL, PL/SI	1000	Jahr:Jahre Land:__
Adjektive	Grundform/ Komparativ	GR/KOM, KOM/GR	1000	groß:größer kalt:__
	Grundform/ Superlativ	GR/SUP, SUP/GR	1000	groß:größte kalt:__
	Komparativ/ Superlativ	KOM/SUP, SUP/KOM	1000	größer:größte kälter:__
Verben (Präsens)	Infinitiv/ 1.Pers. Singular	INF/1SP, 1SP/INF	1000	sein:bin haben:__
	Infinitiv/ 2.Pers. Plural	INF/2PP, 2PP/INF	1000	sein:seid haben:__
	1.Pers. Singular/ 2.Pers. Plural	1SP/2PP, 2PP/1SP	1000	bin:seid habe:__
Verben (Präteritum)	Infinitiv/ 3.Pers. Singular	INF/3SV, 3SV/INF	1000	sein:war haben:__
	Infinitiv/ 3.Pers. Plural	INF/3PV, 3PV/INF	1000	sein:waren haben:__
	3.Pers. Singular/ 3.Pers. Plural	3SV/3PV, 3PV/3SV	1000	war:waren hatte:__

Tabelle 4.4: Muster (Kombinationen der einzelnen Wortformen) für das syntaktische Test-Set. „Jahr:Jahre Land:__“ meint dabei die Analogiefrage: „Jahr“ verhält sich zu „Jahre“, wie „Land“ zu „Länder“, wobei „Länder“ die korrekte Antwort darstellt.

4.4.2 Semantisches Test-Set

Um Zusammenhänge von Wörtern bzgl. ihrer Bedeutung evaluieren zu können, werden drei Arten semantischer Test-Set Fragen erstellt: Analogiefragen zum Gegenteil eines Wortes, thematische Analogiefragen und Fragen zum inhaltlich nicht passenden Wort einer Wortreihe.

Zum Test auf das korrekte Gegenteil eines gegebenen Wortes (`OPPOSITE`), werden 30 verschiedene Wortpaare (jeweils ein Wort und sein entsprechendes Gegenteil, z.B. *Sommer*, *Winter* oder *Tag*, *Nacht*) gefunden. Anschließend wird jedes Wortpaar mit 10 zufällig ausgewählten weiteren Wortpaaren kombiniert, um so (wie auch schon beim syntaktischen Test-Set) die Analogiefragen zum Gegenteil systematisch zu

generieren. Daraus ergeben sich 300 Analogiefragen, Listing A.5 zeigt einen Auszug dieses Test-Sets.

Ähnlich werden auch die thematischen Analogiefragen (`BESTMATCH`) erstellt. Hier werden zu verschiedenen Gruppen passende Wortpaare erstellt. Insgesamt werden 77 Wortpaare folgenden 7 Gruppen zugeordnet: Land-Währung, Hauptstad-Land, Land-Kontinent, Land-Sprache, Politik, Technik und Geschlecht. Alle Wortpaare werden dabei innerhalb einer Gruppe so miteinander kombiniert, dass keine Kombination mehrfach auftrat. Insgesamt ergeben sich daraus 540 Analogiefragen, Listing A.3 zeigt einen Auszug dieses Test-Sets.

Für den dritten Semantiktest werden schließlich drei inhaltlich zusammengehörende Wörter (wie z.B. die Monate *August*, *April* und *September*) kombiniert mit einem weniger passendem Wort (wie z.B. *Jahr*, *Monat* oder *Kalender*), welches bei der Evaluation als solches gefunden werden soll (`DOESNTFIT`). Es werden 11 Wort-Tripel mit jeweils 10 nicht passenden Wörtern kombiniert, sodass sich insgesamt 110 Fragen ergeben. Listing A.4 zeigt einen Auszug dieses Test-Sets.

Für das semantische Test-Set ergeben sich damit also 950 Fragen. Tabelle 4.5 zeigt die Zusammenfassung der Fragen des Semantiktests.

Testname	# Fragen	Beispiel
`OPPOSITE`	`300`	Sommer:Winter Tag:__
`BESTMATCH`	`540`	Berlin:Deutschland Washington:__
`DOESNTFIT`	`110`	Herz Lunge Leber *Arzt*

Tabelle 4.5: Übersicht des semantischen Test-Sets. Die `OPPOSITE` und `BESTMATCH` Tests werden entsprechend den Analogiefragen des Syntaxtests beantwortet. Beim `DOESNTFIT` Test wird mithilfe der Kosinusähnlichkeit der Vektor gefunden, welcher am wenigsten zu allen anderen passt.

5 Auswertung des Modelltrainings

Mithilfe der in Kapitel 4 beschriebenen Implementierung des Toolkits war es möglich, Sprachmodelle zu trainieren und anschließend zu evaluieren. Die Ergebnisse des Modelltrainings, die Diskussion optimaler Parameter und die Auswertung des Modells mit den besten Ergebnissen werden in diesem Kapitel erläutert.

5.1 Evaluationsergebnisse

In den Folgenden Abschnitten sind die Ergebnisse der einzelnen Evaluationsgruppen der trainierten Modelle dargestellt und erläutert. Dabei wird sowohl auf das korrekte Beantworten der Evaluationsfragen mit der ersten Antwort, als auch mit den ersten 10 Antworten (Top 10) eingegangen (vgl. Berechnung des korrekten Ergebnisses und des Top 10 Ergebnisses in Abschnitt 4.4). Außerdem wurde auch die Deckung[1] des jeweiligen Modells bezüglich der Test-Set-Fragen ausgewertet.

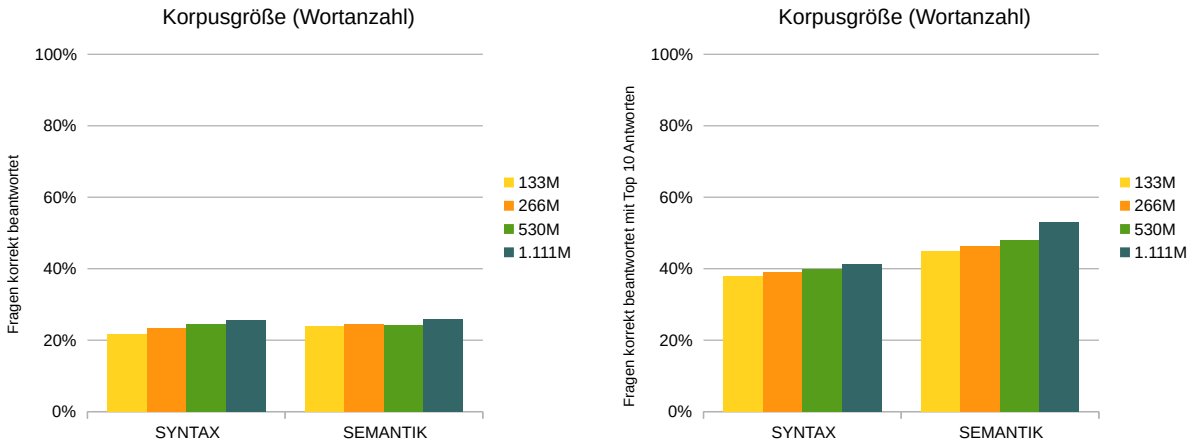
5.1.1 Korpusgröße

Zur Korpusgröße wurden vier Modelle aufgrund von Korpora verschiedener Wortanzahlen trainiert: 133 Millionen Wörter `SG-52-5-133M`, 266 Millionen Wörter `SG-52-5-266M`, 530 Millionen Wörter `SG-52-5-530M` und 1,1 Milliarden Wörter `SG-52-5-1B`. Das Modell des größten Korpus wies dabei die höchste Trainingszeit von etwas mehr als 2 Stunden auf, bei einer Trainingsgeschwindigkeit von 141k Wörtern pro Sekunde.

Abbildung 5.1a zeigt das Gesamtergebnis korrekt beantworteter Fragen gruppiert nach Syntaktik und Semantik, Abbildung 5.1b zeigt das Ergebnis für die korrekte Antwort unter den Top 10 Antworten. Daraus ist ersichtlich, dass mit steigender Korpusgröße, auch die Anzahl korrekt beantworteter Fragen zunimmt. Je mehr

1 Die Deckung bezeichnet die Anzahl beantworteter Fragen im Verhältnis zur Gesamtanzahl vorhandener Fragen. Eine Frage wurde dann beantwortet, wenn alle in der Frage vorkommenden Wörter auch im Modell existierten.

Trainingsdaten demnach zur Verfügung stehen, desto besser können sprachliche Konzepte abgebildet werden.



(a) Korrekt beantwortete Fragen

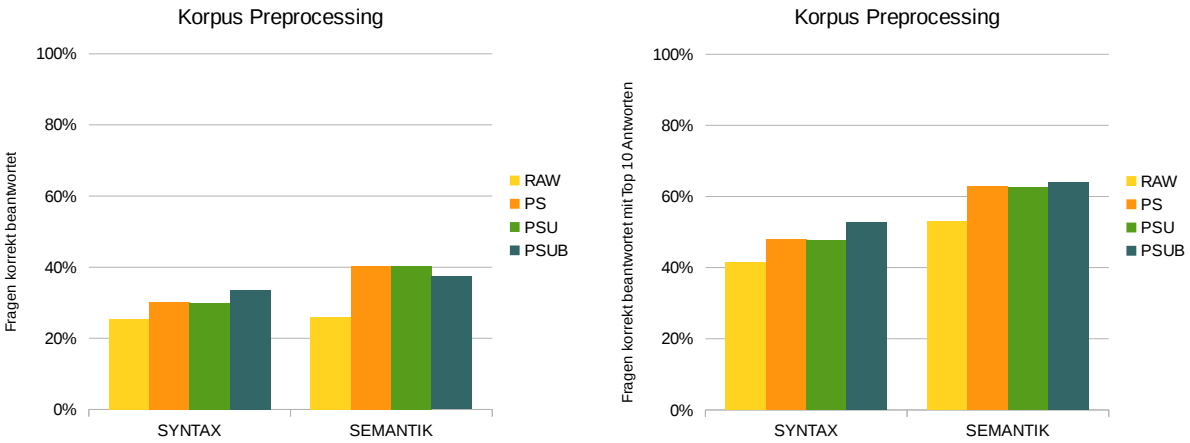
(b) Korrekte Antwort unter den Top 10 Antworten

Abbildung 5.1: Gesamtergebnis der Auswirkung der Korpusgröße (Legende vgl. Tabelle 4.2).

5.1.2 Korpus Preprocessing

Wie in Abschnitt 4.2 erläutert, wurden zum Korpus Preprocessing vier Modelle trainiert: Der reine Korpus ohne zusätzliches Preprocessing `SG-52-5-RAW`, Filterung von Interpunktion und Stoppwörtern `SG-52-5-PS`, zusätzlich dazu die Umwandlung von Umlauten und ß in ihren entsprechenden Digraphen `SG-52-5-PSU` und schließlich das Bilden von Bigramm-Tokens `SG-52-5-PSUB`. Das Training von `SG-52-5-PSUB` ging dabei mit 84 Minuten am schnellsten, bei einer Trainingsgeschwindigkeit von 126k Wörtern pro Sekunde.

Abbildung 5.2a zeigt das Gesamtergebnis korrekt beantworteter Fragen gruppiert nach Syntaktik und Semantik, Abbildung 5.2b zeigt das Ergebnis für die korrekte Antwort unter den Top 10 Antworten.



(a) Korrekt beantwortete Fragen

(b) Korrekte Antwort unter den Top 10 Antworten

Abbildung 5.2: Gesamtergebnis der Auswirkung des Korpus Preprocessing (Legende vgl. Tabelle 4.2).

Aus Abbildung 5.2a ist ersichtlich, dass das Modell `SG-52-5-PSUB` insgesamt das beste Ergebnis erzielt hat. Hier wurden ein Drittel der Syntaxfragen korrekt beantwortet und bei mehr als der Hälfte dieser Fragen war das korrekte Ergebnis unter den Top 10. Bei den Semantikfragen war die korrekte Antwort sogar bei mehr als 60% der Fragen unter den Top 10. Anzumerken ist, dass die Modelle `SG-52-5-PS` und `SG-52-5-PSU` bei der korrekten Antwort von Semantikfragen um etwa 2,5% besser als `SG-52-5 PSUB` abschnitten, dagegen war die Quote der korrekten Antworten unter den Top 10 etwas geringer. Außerdem ergab sich bei beiden Modellen sowohl beim Syntax- als auch beim Semantiktest ein beinahe identisches Ergebnis. Die Umwandlung von Umlauten in ihre Digraph-Repräsentation hatte folglich insgesamt keine Auswirkung auf das Testergebnis.

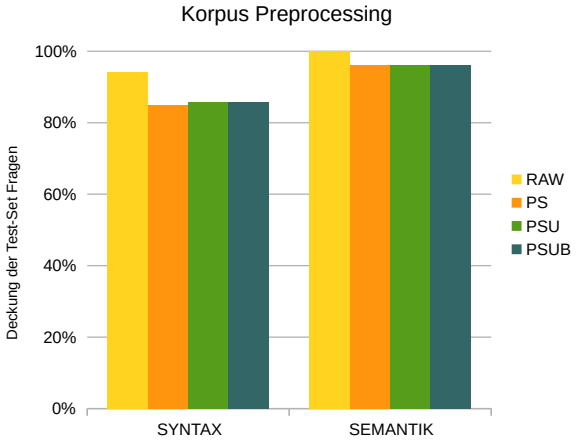
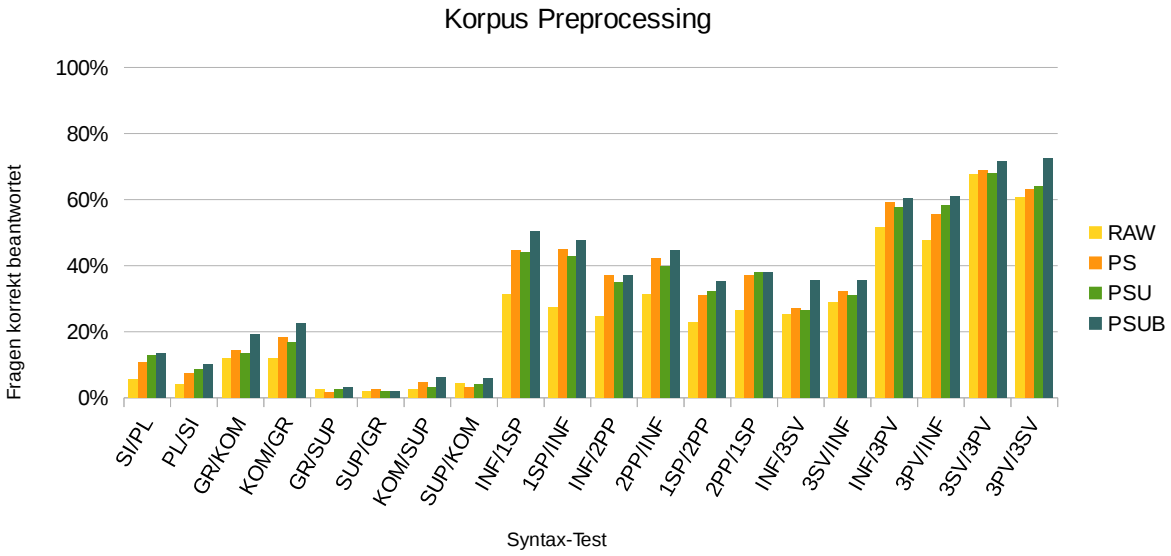


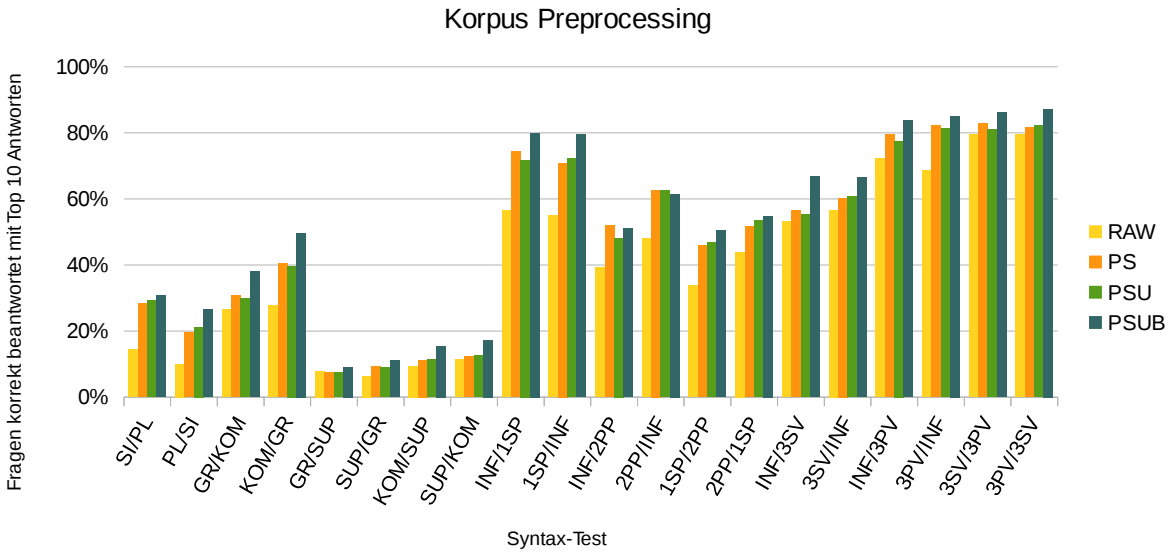
Abbildung 5.3: Gesamtdeckung der Test-Set-Fragen der Modelle zum Korpus Preprocessing)

Die Deckung der Test-Set-Fragen lag bei allen Modellen bei über 85%, das Modell ohne zusätzliches Preprocessing wies dabei jeweils eine höhere Deckung auf als die anderen Modelle. Es ist anzunehmen, dass die Ursache hierfür die Filterung der Stoppwörter ist. Beim Semantiktest wurden hier sogar alle Fragen beantwortet. Allgemein war die Deckung des semantischen Test-Sets höher, da dessen Fragen aufgrund der Ausrichtung auf die Wortbedeutung vermutlich weniger Stoppwörter enthalten, als das syntaktische Test-Set.

Aus der einzelnen Auflistung der syntaktischen Testergebnisse in Abbildung 5.4 geht zunächst hervor, dass allgemein die Syntax-Fragen zu Nomen und Adjektiven am schlechtesten beantwortet wurden. Insbesondere bei Fragen zum Superlativ von Adjektiven wurden maximal 6% der Fragen korrekt beantwortet. Betrachtet man die korrekten Antworten unter den Top 10, so wird die Anzahl der korrekt beantworteten Fragen mehr als verdoppelt (vgl. z.B. `SUP/KOM`), teilweise sogar verdreifacht (vgl. z.B. `SUP/GR`). Ursache für dieses Verhalten könnte die Tatsache sein, dass Deutsche Adjektive in ihren Steigerungsformen abhängig von Singular/Plural sind (z.B. „der **höchste** Turm“, „die **höchsten** Türme“). Im Englischen gibt es diesen Unterschied nicht („the **tallest** tower“, „the **tallest** towers“). Bei der Evaluierung wird nur eine korrekte Antwort geprüft. Aus diesem Grund fällt das Top 10 Ergebnis deutlich besser aus, da sich die gesuchte Form mit höherer Wahrscheinlichkeit unter den besten 10 Ergebnissen befindet.



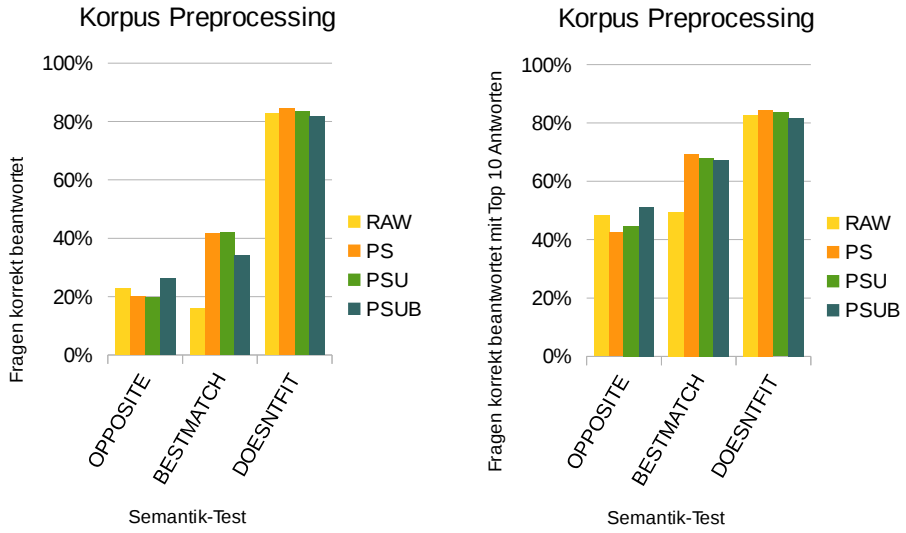
(a) Korrekt beantwortete Fragen



(b) Korrekte Antwort unter den Top 10 Antworten

Abbildung 5.4: Ergebnis der Syntaxfragen Korpus Preprocessing

Die besten Ergebnisse der syntaktischen Fragen wurden bei den Verbformen der dritten Person Plural im Präteritum erreicht. In Kombination mit der dritten Person Singular hatten alle Modelle ein Ergebnis von mehr als 60% und unter den Top 10 mehr als 80% korrekt beantworteter Fragen.



(a) Korrekt beantwortete Fragen **(b)** Korrekte Antwort unter den Top 10 Antworten

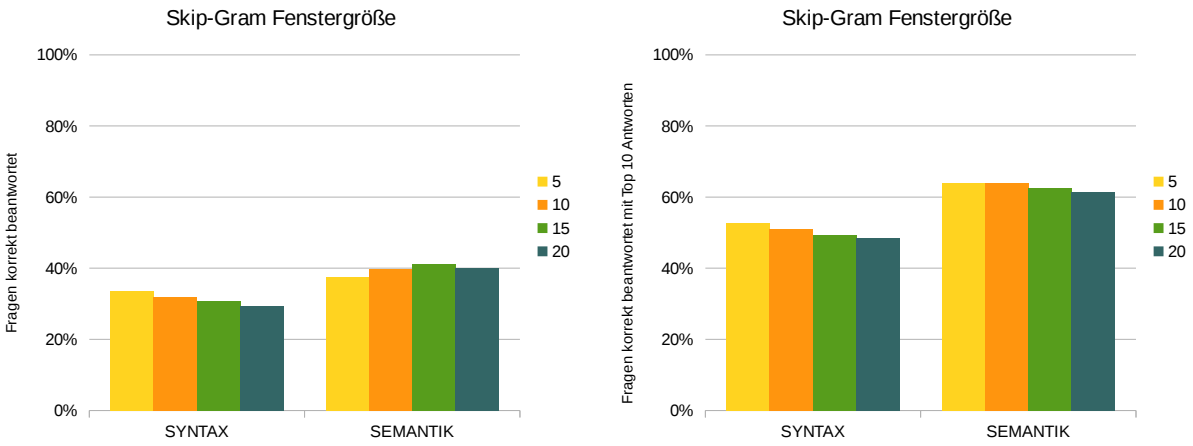
Abbildung 5.5: Ergebnis der Semantikfragen Korpus Preprocessing

Die Ergebnisse der drei Gruppen der Semantik-Fragen (Abbildung 5.5) sind sehr unterschiedlich ausgefallen. Das Gegenteil zu einem gegebenen Wort (`OPPOSITE`) wurde in ca. 20% der Fälle korrekt gefunden, unter den Top 10 war das richtige Ergebnis bei mehr als 42% der Fragen. Ähnlich wie bei den Adjektiven ist der Anteil der korrekten Top 10 Antworten damit mehr als doppelt so hoch. Beim `BESTMATCH` Test unterscheiden sich die einzelnen Modelle deutlich voneinander. `SG-52-5 PS` und `SG-52-5 PSU` schneiden dabei mit über 40% korrekter Antworten mit Abstand am besten ab. Beim Test auf das am wenigsten passende Wort in einer Gruppe von 4 Wörtern (`DOESNTFIT`) beantworten schließlich alle 4 Modelle mehr als 4 von 5 Fragen korrekt. Hier sind das Ergebnis korrekter Antworten und das Top 10 Ergebnis identisch, da bei diesem Test kein korrektes Wort aus dem Vokabular gefunden, sondern ein gegebenes Wort identifiziert werden muss.

5.1.3 Skip-Gram Fensterbreite

Zur Evaluation verschiedener Fensterbreiten wurden bei konstanter Vektorgröße von 52 vier Modelle mittels Skip-Gram trainiert (`SG-52-5`, `SG-52-10`, `SG-52-15` und `SG-52-20`), wobei die Fensterbreite 5, 10, 15 und 20 betrug. Die Trainingsdauer von

84 Minuten bei der Fensterbreite von 5 erhöhte sich dabei mit steigender Fensterbreite nur geringfügig um jeweils etwa 6 Minuten.



(a) Korrekt beantwortete Fragen

(b) Korrekte Antwort unter den Top 10 Antworten

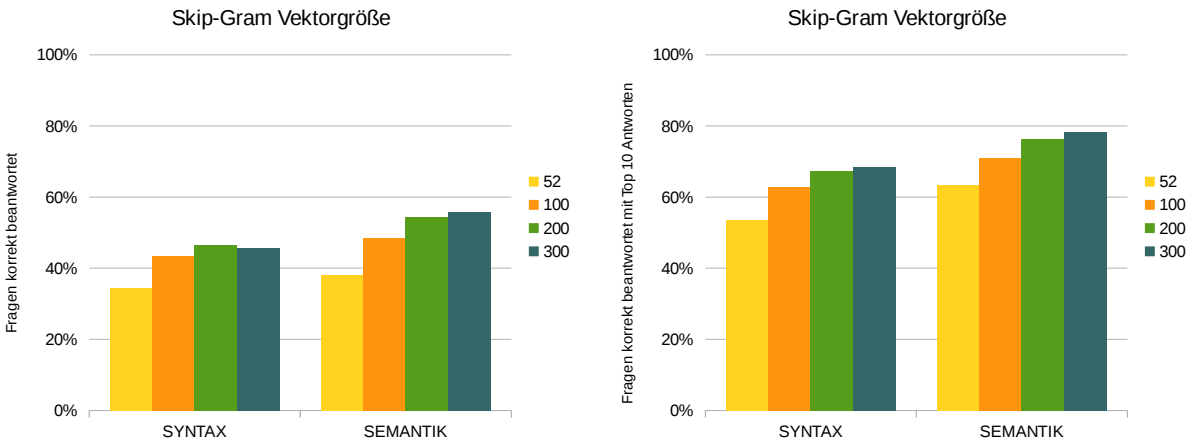
Abbildung 5.6: Gesamtergebnis Skip-Gram Fensterbreite

Aus Abbildung 5.6 ist ersichtlich, dass das Modell `SG-52-5` mit der kleinsten Fensterbreite insgesamt das beste Ergebnis erzielt hat. Im Syntaxtest wurden hier gegenüber den anderen Modellen sowohl die meisten Fragen korrekt beantwortet, als auch die höchste Quote der korrekten Antwort unter den Top 10 Antworten erreicht. Außerdem nimmt die Anzahl korrekter Antworten mit steigender Fensterbreite linear ab. Das gilt auch für die Top 10 Ergebnisse im Semantiktest. Nur beim Test der korrekten Ergebnisse im Semantiktest erzielt das Modell mit Fensterbreite 15 das beste Ergebnis. Folglich könnte eine größere Fensterbreite (zwischen 10 und 20), also das Einbeziehen von Wörtern mit größerem Abstand zum Zielwort, für das korrekte Beantworten semantischer Fragen vorteilhaft sein.

5.1.4 Skip-Gram Vektorgröße

Zur Evaluation verschiedener Vektorgrößen wurden bei konstanter Fensterbreite von 5 und einer Worthäufigkeit von mindestens 10 vier Modelle mittels Skip-Gram trainiert (`SG-52-5-R10`, `SG-100-5-R10`, `SG-200-5-R10` und `SG-300-5-R10`), wobei

die Vektorgröße 52[2], 100, 200 und 300 betrug. Die Trainingsdauer stieg mit steigernder Vektorgröße von 1,4 Stunden (`52`) auf 3,5 Stunden (`300`) stark an.



(a) Korrekt beantwortete Fragen

(b) Korrekte Antwort unter den Top 10 Antworten

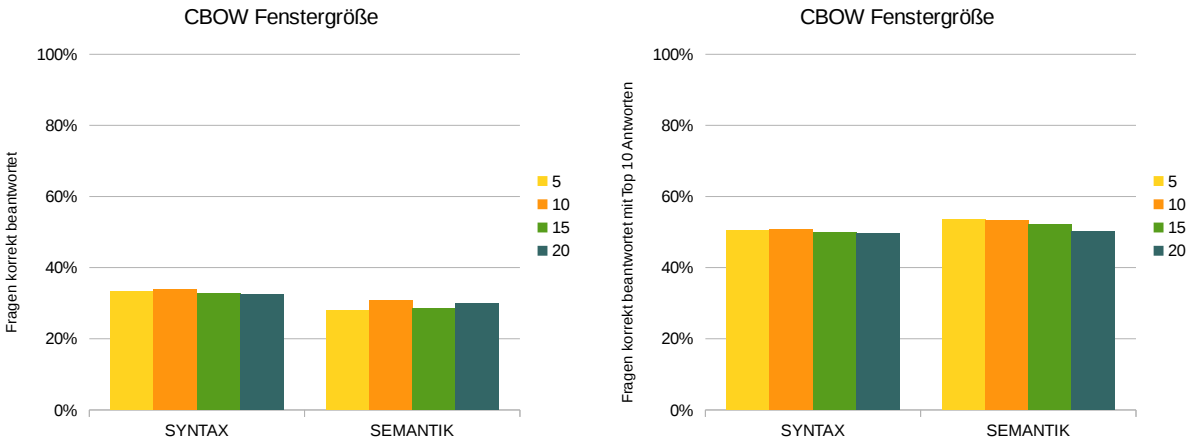
Abbildung 5.7: Gesamtergebnis Skip-Gram Vektorgröße

Aus Abbildung 5.7 ist ersichtlich, dass das Modell `SG-300-5-R10` mit der größten Vektordimension insgesamt das beste Ergebnis erzielt hat. Allgemein nimmt die Anzahl korrekter Antworten mit steigender Vektorgröße zu. Die einzige Ausnahme bildet das Ergebnis der korrekten Antworten beim Syntaxtest (Abbildung 5.7a links): Das Modell mit der Vektorgröße 200 ist hier mit einem knappen Vorsprung von 0,9% das beste Modell.

5.1.5 Continuous-Bag-of-Words Fensterbreite

Auch mit der CBOW Architektur wurden vier Modelle unterschiedlicher Fensterbreiten trainiert (`CB-52-5`, `CB-52-10`, `CB-52-15` und `CB-52-20`), wobei die Fensterbreite wie bei Skip-Gram 5, 10, 15 und 20 betrug. Die Trainingsdauer war im Gegensatz zu Skip-Gram bei allen Modellen mit ca. 100 Minuten bei einer Trainingsgeschwindigkeit von 105k Wörtern pro Sekunde ungefähr gleich.

2 Für die kleinste Vektorgröße wurde 52 statt 50 gewählt, weil die Vektordimension für ein optimales Training ein Vielfaches der verwendeten Threadanzahl (in dieser Arbeit 4) betragen sollte.



(a) Korrekt beantwortete Fragen

(b) Korrekte Antwort unter den Top 10 Antworten

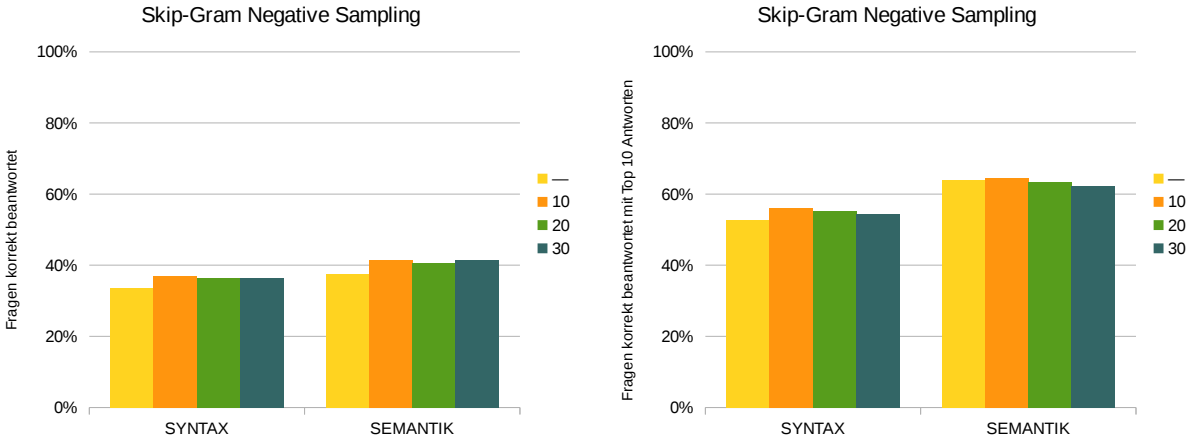
Abbildung 5.8: Gesamtergebnis CBOW Fensterbreite

Aus Abbildung 5.8 ist ersichtlich, dass bei CBOW im Gegensatz zu Skip-Gram das Modell `CB-52-10` mit der Fensterbreite 10 insgesamt das beste Ergebnis erzielt hat. Im Syntaxtest wurden hier gegenüber den anderen Modellen sowohl die meisten Fragen korrekt beantwortet, als auch die höchste Quote der korrekten Antwort unter den Top 10 Antworten erreicht. Tendenziell nimmt aber auch hier die Anzahl korrekter Antworten mit steigender Fensterbreite ab. Das gilt auch für die Top 10 Ergebnisse im Semantiktest. Nur beim Test der korrekten Ergebnisse unter den Top 10 im Semantiktest erzielt das Modell mit Fensterbreite 5 das beste Ergebnis, allerdings mit einem sehr geringen Vorsprung von 0,3%.

5.1.6 Skip-Gram Negative Sampling

Zur Evaluation der Negative Sampling Optimierung wurden bei konstanter Fensterbreite von 5 und konstanter Vektorgröße von 52 drei Modelle mittels Skip-Gram trainiert (`SG-52-5-NS10`, `SG-52-5-NS20` und `SG-52-5-NS30`), wobei die Anzahl der Negative Samples 10, 20 und 30 betrug. Im Vergleich dazu ist das Modell `SG-52-5` ohne Negative Sampling ebenfalls aufgeführt. Die Trainingsdauer der Modelle stieg dabei mit steigender Anzahl der Negative Samples stark an (ca. eine Stunde längeres Training pro zusätzlichen 10 Negative Samples). Das Modell `SG-52-5-NS30` benö-

tigte fast 4,5 Stunden für das Training, bei einer Trainingsgeschwindigkeit von 40k Wörtern pro Sekunde.



(a) Korrekt beantwortete Fragen

(b) Korrekte Antwort unter den Top 10 Antworten

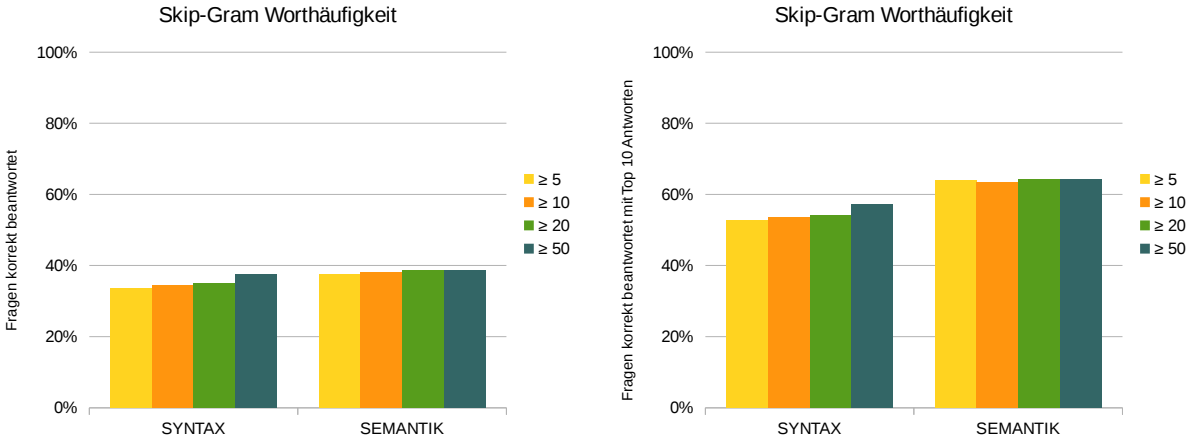
Abbildung 5.9: Gesamtergebnis Skip-Gram Negative Sampling. Das erste Modell (—) ist ohne Negative Sampling trainiert.

Aus Abbildung 5.9 ist ersichtlich, dass das Modell `SG-52-5-NS10` mit der Sample-Anzahl 10 insgesamt das beste Ergebnis erzielt hat. Dabei sind mit Ausnahme der Top 10 Ergebnisse des Semantiktests alle drei Modelle besser als das äquivalente Modell ohne Negative Sampling. Bei der Auswertung der korrekten Antworten in Abbildung 5.9a sind die Ergebnisse für die unterschiedlichen Sample-Anzahlen annähernd konstant, wohingegen bei der Auswertung der korrekten Antwort unter den Top 10 Antworten in Abbildung 5.9b ersichtlich ist, dass sich das Ergebnis mit steigender Sample-Anzahl verschlechtert.

5.1.7 Skip-Gram Worthäufigkeit

Zur Evaluation der Worthäufigkeit, die ein Wort mindestens erreichen muss, um betrachtet zu werden, wurden bei konstanter Fensterbreite von 5 und konstanter Vektorgröße von 52 vier Modelle mittels Skip-Gram trainiert (`SG-52-5-R5` , `SG-52-5-R10` , `SG-52-5-R20` und `SG-52-5-R50`), wobei die Worthäufigkeit mindestens 5, 10, 20 und 50 betrug. Die Trainingsdauer verringerte sich mit steigender Mindestwortanzahl

nur geringfügig von 84 Minuten (`R5`) auf 72 Minuten (`R50`), bei einer Trainingsgeschwindigkeit von ungefähr 130k Wörtern pro Sekunde.



(a) Korrekt beantwortete Fragen

(b) Korrekte Antwort unter den Top 10 Antworten

Abbildung 5.10: Gesamtergebnis Skip-Gram Worthäufigkeit

Aus Abbildung 5.10 ist ersichtlich, dass das Modell `SG-52-5-R50` mit einer Worthäufigkeit von mindestens 50 insgesamt das beste Ergebnis erzielt hat. Dabei sind die Ergebnisse des Semantiktests für alle Modelle beinahe identisch, wohingegen beim Syntaxtest eindeutig das Modell mit einer Worthäufigkeit von mindestens 50 das beste Ergebnis erzielt.

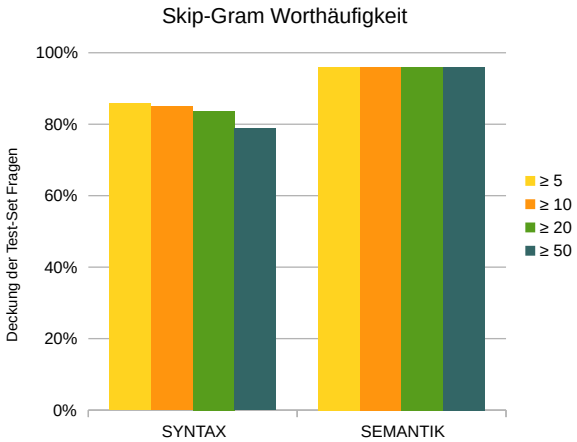
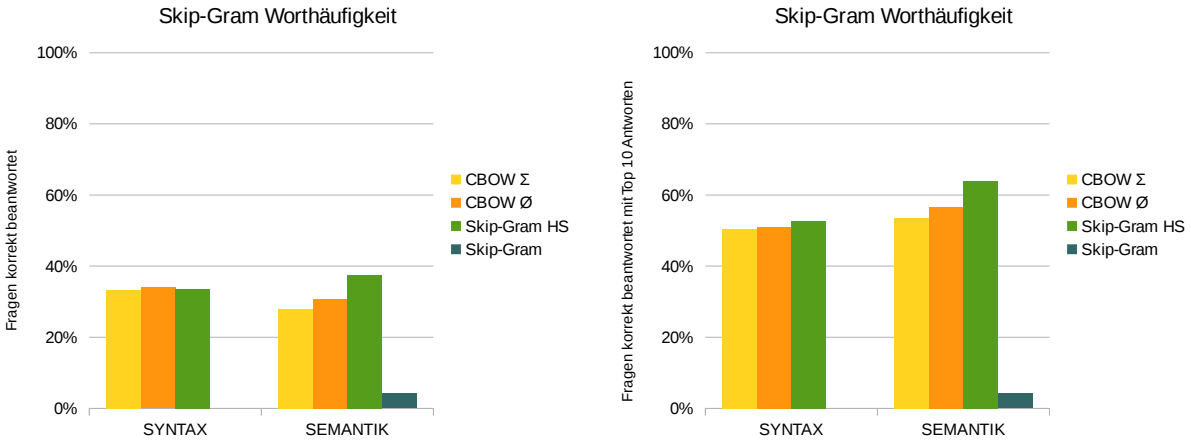


Abbildung 5.11: Gesamtdeckung der Test-Set-Fragen der Modelle zur Worthäufigkeit

Aufgrund des Aussortierens von Wörtern niedrigerer Häufigkeit, verringert sich die Deckung der Test-Set-Fragen (dargestellt in Abbildung 5.10) mit steigender Mindest-Worthäufigkeit geringfügig. Dies wirkt sich allerdings nur auf die Deckung der Syntaxfragen aus. Anzunehmen ist, dass diese Verringerung der Deckung die Anzahl falscher Antworten vermindert und somit für die Verbesserung des qualitativen Ergebnisses des Syntaxtestes verantwortlich ist.

5.1.8 Architektur, Projektion und Hierarchical Softmax

In der letzten Evaluationsgruppe wurden die Architekturen CBOW und Skip-Gram miteinander verglichen. Dabei wurde bei den CBOW-Modellen die Projektionsart variiert (Summe oder Durchschnitt) und die Skip-Gram-Modelle wurden einmal mit und einmal ohne Hierarchical Sampling trainiert. Das Evaluationsergebnis ist in Abbildung 5.12 abgebildet.



(a) Korrekt beantwortete Fragen

(b) Korrekte Antwort unter den Top 10 Antworten

Abbildung 5.12: Gesamtergebnis Architektur, Projektion und Hierarchical Softmax

Zunächst fällt auf, dass das Skip-Gram-Modell ohne Hierarchical Softmax keine einzige Syntaxfrage und nur 4,2% der Semantikfragen korrekt beantworten konnte. Bei den beiden CBOW-Modellen erzielt das Modell mit der Projektion der Wort-Vektoren durch Bilden des Durchschnitts sowohl im Syntax- als auch im Semantiktest das bessere Ergebnis. Im Gesamtergebnis liefert jedoch das Skip-Gram-Modell mit

Hierarchical Softmax das beste Ergebnis, im Semantiktest sogar mit einem Abstand von über 7%.

5.2 Diskussion optimaler Parameter

Basierend auf den Evaluationsergebnissen der einzelnen Evaluationsgruppen, welche in Abschnitt 5.1 erläutert wurden, kann nun die Parameterkonfiguration für das im Rahmen dieser Arbeit optimale Deutsche Sprachmodell spezifiziert werden. Dazu ist die aus der Parameterkonfiguration bekannte Tabelle 4.3 als Übersicht mit dem jeweils besten Modell einer Gruppe (markiert) in Tabelle 5.1 dargestellt.

Daraus sind folgende Spezifikationen ersichtlich:

Der Trainingskorpus sollte möglichst groß sein und vorverarbeitet werden, indem Interpunktion und Stoppwörter gefiltert und Bigramm-Tokens gebildet werden; als Trainingsarchitektur sollte Skip-Gram gewählt werden; die Fensterbreite sollte zwischen 5 und 10 liegen; die Vektordimension sollte je nach verfügbarer Rechenleistung möglichst groß gewählt werden; Negative Sampling sollte bei einer vergleichsweise kleinen Sample-Anzahl von 10 verwendet werden; Wörter mit geringer Häufigkeit (≤ 50) sollten für qualitativ bessere Ergebnisse gefiltert werden[3]; Hierarchical Sampling sollte verwendet werden.

5.3 Auswertung des optimalen Modells

Im Folgenden werden die Ergebnisse des Modells mit der in Abschnitt 5.2 gefundenen optimalen Parameterkonfiguration vorgestellt und mit den Ergebnissen von Mikolov et al. [15] für ein englisches Modell verglichen. Das Training dieses Modells dauerte dabei ungefähr 6 Stunden bei einer Trainingsgeschwindigkeit von knapp 27k Wörtern pro Sekunde.

3 Dabei ist allerdings darauf zu achten, dass die Deckung der Test-Set-Fragen noch ausreichend hoch ist.

Modellname	A	Pr	D	N	HS	NS	R	P
SG-52-5-133M	Skip-Gram	—	52	5	Ja	Nein	5	—
SG-52-5-266M	Skip-Gram	—	52	5	Ja	Nein	5	—
SG-52-5-530M	Skip-Gram	—	52	5	Ja	Nein	5	—
SG-52-5-1B	Skip-Gram	—	52	5	Ja	Nein	5	—
SG-52-5-RAW	Skip-Gram	—	52	5	Ja	Nein	5	—
SG-52-5-PS	Skip-Gram	P, S	52	5	Ja	Nein	5	—
SG-52-5-PSU	Skip-Gram	P, S, U	52	5	Ja	Nein	5	—
SG-52-5-PSUB	Skip-Gram	P, S, U, B	52	5	Ja	Nein	5	—
SG-52-5	Skip-Gram	P, S, U, B	52	5	Ja	Nein	5	—
SG-52-10	Skip-Gram	P, S, U, B	52	10	Ja	Nein	5	—
SG-52-15	Skip-Gram	P, S, U, B	52	15	Ja	Nein	5	—
SG-52-20	Skip-Gram	P, S, U, B	52	20	Ja	Nein	5	—
SG-52-5-R10	Skip-Gram	P, S, U, B	52	5	Ja	Nein	5	—
SG-100-5-R10	Skip-Gram	P, S, U, B	100	5	Ja	Nein	5	—
SG-200-5-R10	Skip-Gram	P, S, U, B	200	5	Ja	Nein	5	—
SG-300-5-R10	Skip-Gram	P, S, U, B	300	5	Ja	Nein	5	—
CB-52-5	CBOW	P, S, U, B	52	5	Ja	Nein	5	$\sum$
CB-52-10	CBOW	P, S, U, B	52	10	Ja	Nein	5	$\sum$
CB-52-15	CBOW	P, S, U, B	52	15	Ja	Nein	5	$\sum$
CB-52-20	CBOW	P, S, U, B	52	20	Ja	Nein	5	$\sum$
SG-52-5	Skip-Gram	P, S, U, B	52	5	Ja	Nein	5	—
SG-52-5-NS10	Skip-Gram	P, S, U, B	52	5	Ja	10	5	—
SG-52-5-NS20	Skip-Gram	P, S, U, B	52	5	Ja	20	5	—
SG-52-5-NS30	Skip-Gram	P, S, U, B	52	5	Ja	30	5	—
SG-52-5	Skip-Gram	P, S, U, B	52	5	Ja	Nein	5	—
SG-52-5-R10	Skip-Gram	P, S, U, B	52	5	Ja	Nein	10	—
SG-52-5-R20	Skip-Gram	P, S, U, B	52	5	Ja	Nein	20	—
SG-52-5-R50	Skip-Gram	P, S, U, B	52	5	Ja	Nein	50	—
CB-52-5-SUM	CBOW	P, S, U, B	52	5	Ja	Nein	5	$\sum$
CB-52-5-MEAN	CBOW	P, S, U, B	52	5	Ja	Nein	5	Ø
SG-52-5-HS	Skip-Gram	P, S, U, B	52	5	Ja	Nein	5	—
SG-52-5-NOHS	Skip-Gram	P, S, U, B	52	5	Nein	Nein	5	—

Tabelle 5.1: Parameter-Spezifikationen trainierter Modelle zur Architektur *A*, Preprocessing *Pr*, Vektordimension *D*, Fensterbreite *N*, Hierarchical Softmax *HS*, Negative Sampling *NS*, Worthäufigkeit *R* und Projektionsart *P*. Markierte Modelle lieferten das beste Evaluationsergebnis ihrer Gruppe.

5.3.1 Evaluationsergebnis

Das Gesamtergebnis des optimalen Modells in Abbildung 5.13 zeigt den Anteil korrekt beantworteter Fragen mit der ersten Antwort und mit den Top 10 Antworten. Im Syntaxtest wurden über die Hälfte der Fragen korrekt beantwortet, im Semantiktest sogar 60%. Die korrekte Antwort lag unter den Top 10 Antworten bei 3 von 4 Fragen im Syntaxtest und bei 4 von 5 Fragen im Semantiktest. Das ist im Hinblick auf das eigenständige Lernen sprachlicher Strukturen (unüberwachtes Training) ein sehr gutes Ergebnis.

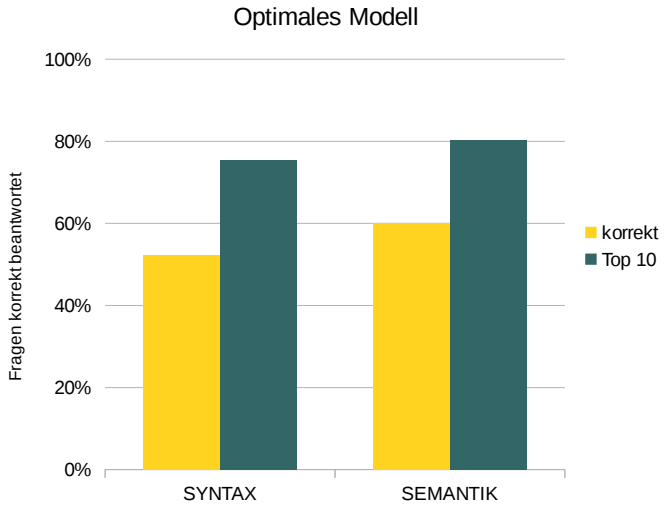


Abbildung 5.13: Gesamtergebnis des optimalen Modells

In Tabelle 5.2 ist der Vergleich der Evaluationsergebnisse des Syntaxtests mit dem englischen Modell `RNN-1600` von Mikolov et al. dargestellt. Dieses Modell wurde mithilfe eines Recurrent-Netzwerks auf englischen Nachrichtenartikeln mit insgesamt 320 Millionen Wörtern trainiert, bei einer Vektordimension von 1600.

Sprach-Modell	Adjektive	Nomen	Verben	Gesamt
Deutsch (diese Arbeit)	**27,4**	23,3	**68,6**	**52,3**
Englisch (RNN-1600)	23,9	**29,2**	62,2	39,6

Tabelle 5.2: Vergleich der Ergebnisse des Syntaxtests des besten deutschen Modells dieser Arbeit mit dem besten englischen Modell von Mikolov et al. [15]. Angaben in Prozent.

Anzumerken ist hier, dass der direkte Vergleich der Evaluationsergebnisse beider Modelle aus unterschiedlichen Gründen nicht vollständig repräsentativ ist. Zum einen wurden unterschiedliche Korpora für das Training verwendet. Dies ist für den sprachlichen Unterschied zwar notwendig, trotzdem können dabei Größe und Bezugsquelle des Korpus übereinstimmen. In dieser Arbeit wurden sowohl deutsche Nachrichtenartikel als auch die deutsche Wikipedia mit insgesamt über 650 Millionen Wörtern als Korpus verwendet, Mikolov et al. nutzen dagegen ausschließlich Nachrichtenartikel bei weniger als der Hälfte der Wörter (320 Millionen). Zum anderen wurden verschiedene Test-Sets für die Evaluation genutzt, wobei sowohl die Gesamtanzahl der Fragen, als auch die Anzahl der Fragen zu den unterschiedlichen Wortarten (Adjektive, Nomen und Verben) verschieden war [4].

Trotz der beschriebenen Einschränkungen des Modellvergleichs ist ersichtlich, dass sprachliche Zusammenhänge ähnlich abgebildet werden. So beantworten beide Modelle die Fragen zu Adjektiven und Nomen zu höchstens 30% korrekt, wobei die Antworten der Fragen zu Verben über 60% korrekt waren. Dies lässt die Schlussfolgerung zu, dass sowohl in englischen als auch in deutschen Modellen Verben besser abgebildet werden als Nomen und Adjektive.

5.3.2 Ausgewählte Wortzusammenhänge

Nachdem die Modelle allgemein ausgewertet wordem somd, beschäftigt sich dieser Abschnitt mit ausgewählten konkreten Beispielen komplexerer Wortzusammenhänge, die mithilfe zusätzlicher Analogiefragen manuell gefunden werden konnten. Damit wird die Fähigkeit der Wort-Vektoren verdeutlicht, die Bedeutung von Wörtern abbilden zu können. Wie bereits bei den Analogiefragen der Evaluation wurde eine Vektoraddition (bzw. -subtraktion) vollzogen und anschließend der Vektor ausgegeben, der dem Ergebnis am ähnlichsten ist. Eine Auswahl davon wird im Folgenden beschrieben. Die Kosinusähnlichkeit (vgl. Abschnitt 2.2.2) ist jeweils in Klammern angegeben.

`Frau + Kind = Mutter` (0,831)
Ein sehr häufig auftretender familiärer Zusammenhang mit einer vergleichsweise hohen Kosinusähnlichkeit.

4 Für den Syntaxtest wurden im englischen Modell 8k Fragen verwendet, jeweils 3k für Adjektive und Verben und 2k für Nomen [15].

`Verwaltungsgebaeude + Buecher = Bibliothek` (0,718)
Der Zusammenhang von Gebäudearten und ihrer Funktion, in diesem Fall ein Gebäude zur Verwaltung von Büchern.

`Haus + Filme + Popcorn = Kino` (0,721)
Eine weiteres Beispiel einer Gebäudeart. Interessant ist hier, dass bereits der Vektor `Haus + Filme` dem Vektor `Kino` am ähnlichsten ist, hier allerdings mit einer etwas geringeren Kosinusähnlichkeit von 0,713. Das bedeutet, dass das Hinzufügen von `Popcorn` den Vektor `Kino` besser beschreibt.

`Planet + Wasser = Erde` (0,717)
Das Hauptmerkmal unseres Planeten gegenüber anderer Planeten wird korrekt erkannt.

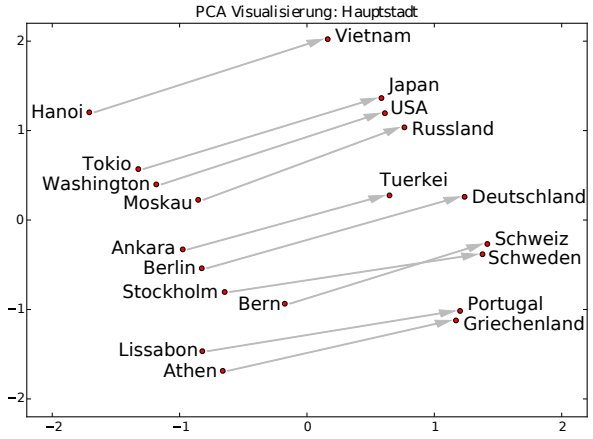
`Kerze + Feuerzeug = brennende_Kerze` (0,768)
Schließlich ein Beispiel für ein Bigramm. Ohne die Umformung der Korpusdaten zu Bigrammen könnte diese Beziehung so nicht dargestellt werden.

5.3.3 Visualisierung mit Principal Component Analysis

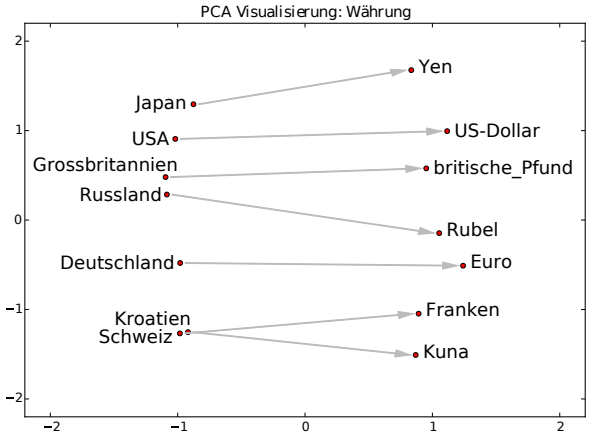
Neben der Darstellung von Wortbedeutungen mithilfe der Analogiefragen ist es auch möglich, Merkmale von Wort-Vektoren zu visualisieren, um gelernte sprachliche Konzepte zu verdeutlichen.

Die Anzahl der Dimensionen der Wort-Vektoren sollte für eine bessere Modellierung von Sprache möglichst groß gewählt werden (vgl. Evaluationsergebnisse der Vektorgröße in Abschnitt 5.1.4). Dabei ist es jedoch unmöglich, einen solch hochdimensionalen Vektor direkt grafisch darzustellen. Aus diesem Grund gibt es die *Principal Component Analysis* (PCA, deutsch: Hauptkomponentenanalyse, begründet von Karl Pearson, 1901 [19]). PCA ist eine statistische Analysemethode, welche die hohe Komponentenanzahl einer Variablen auf einige wenige Hauptkomponenten reduzieren kann, welche die ursprüngliche Variable möglichst aussagekräftig beschreiben. Mikolov et al. [13] haben die Beziehung zwischen Land und Hauptstadt mittels PCA für ein englisches Sprachmodell dargestellt.

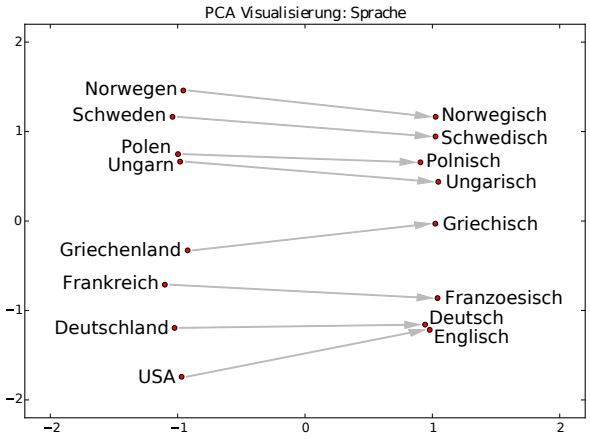
Die Größe der Wort-Vektoren des optimalen Modells in dieser Arbeit ist 300. Diese Größe wurde mithilfe von PCA auf 2 Dimensionen reduziert und für ausgewählte



(a) Abbildung von Ländern und ihren Hauptstädten



(b) Abbildung von Ländern und ihren Währungen



(c) Abbildung von Ländern und ihren Sprachen

Abbildung 5.14: PCA Visualisierung von Wort-Vektoren von Ländern, Hauptstädten, Währungen und Sprachen

Wort-Vektoren in Abbildung 5.14 dargestellt. Dabei wurden manuell Länder mit ihren entsprechenden Hauptstädten, Währungen und Sprachen ausgewählt. Aufgrund der zweidimensionalen Darstellung ist sehr gut ersichtlich, dass das Sprachmodell z.B. in Abbildung 5.14c das Konzept von Landessprachen gelernt hat, ohne dass im Training explizit vorgegeben wurde, welches die Sprache eines Landes ist (unüberwachtes Training). In der Abbildung sind Länder links und entsprechende Sprachen rechts gruppiert zu sehen. Der Abstand und die Verbindungsrichtung eines Landes zu seiner Sprache sind dabei untereinander sehr ähnlich. Genauso ist diese Gruppierung auch für Hauptstädte und Währungen zu beobachten.

5.3.4 Verteilung korrekter Antworten

Ein weiterer interessanter Aspekt der Evaluation des optimalen Modells ist die Anzahl korrekter Antworten in Abhängigkeit von den ersten N als korrekt geltenden Antworten (Top N).

In Abbildung 5.15 ist der Anteil korrekt beantworteter Syntaxfragen in Abhängigkeit der Anzahl N der als korrekt geltenden Antworten abgebildet. Dabei ist ersichtlich, dass für kleine N (≤ 10) der Anteil korrekt beantworteter Fragen mit steigendem N stark zunimmt. Für größere N flacht der Anstieg immer mehr ab, sodass sich das Ergebnis schließlich durch Erhöhung von N nicht mehr verbessert. Das bedeutet, dass sich eine mögliche korrekte Antwort mit hoher Wahrscheinlichkeit auch unter den ersten Ergebnissen befindet, welche das Modell liefert.

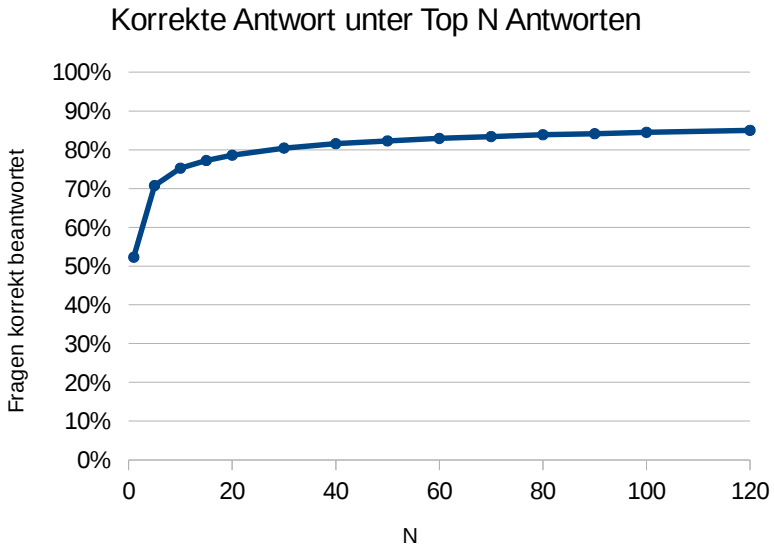


Abbildung 5.15: Anteil korrekt beantworteter Syntaxfragen des optimalen Modells unter Variation der Anzahl N der als korrekt geltenden Antworten (Top N)

In diesem Kapitel wurden die Ergebnisse dieser Arbeit präsentiert und verschiedene Modelle mittels selbst generierter Test-Sets auf allgemeiner Ebene evaluiert. Außerdem wurde das daraus resultierende optimale Modell im Vergleich zu ähnlichen Experimenten und anhand konkreter Beispiele ausgewertet. Im Folgenden wird abschließend zu diesen Ergebnissen Stellung genommen.

6 Fazit

Zur Verarbeitung großer Textmengen und automatisierter Modellierung von Sprache gibt es viele Ansätze. Dabei erreicht *Deep Learning* mithilfe neuronaler Netzwerke basierend auf englischer Sprache heute nicht nur hervorragende Ergebnisse, sondern kann Modelle mithilfe von Architekturen wie Skip-Gram oder CBOW auch unüberwacht trainieren.

Während das Trainieren von Wort-Vektoren bisher hauptsächlich für die englische Sprache durchgeführt wurde, hat diese Arbeit gezeigt, dass eine solche Sprachmodellierung auch für die deutsche Sprache funktioniert. Es wurden mithilfe eines dafür entwickelten Toolkits unter Parametervariation insgesamt 25 unterschiedliche Modelle und daraus resultierend ein optimales Modell trainiert[1], welches bei der Evaluierung syntaktischer Merkmale ein ähnliches Niveau erreicht, wie ein vergleichbares System (vgl. Abschnitt 5.3.1). Dieses Training war dabei aufgrund einer recheneffizienten Implementierung von Skip-Gram und CBOW auch auf großen Korpora mit einem normalen Heimrechner (vgl. Abschnitt 4.3.3) möglich. Beliebige und große Korpora konnten verwendet werden, da das Training unüberwacht stattfand und deshalb keine gelabelten Eingabedaten nötig waren, sondern ausschließlich Text, wie er im normalen Sprachgebrauch auftritt.

Mit der aus der Analyse der Evaluationsergebnisse gefundenen optimalen Parameterkofiguration konnten die im Rahmen dieser Arbeit besten deutschen Wort-Vektoren trainiert werden. Mit diesen Wort-Vektoren war es möglich, syntaktische und semantische Wort-Zusammenhänge darzustellen, komplexe Wortbeziehungen in ihrer Bedeutung korrekt zuordnen (vgl. Abschnitt 5.3.2) und Merkmale mithilfe von PCA zu visualisieren.

Die in dieser Arbeit trainierten Wort-Vektoren können nun als Grundlage für weiterführende Forschungen und Anwendungen der Sprachmodellierung für die deutsche Sprache dienen, wie z.B. *Relation Detection* bzw. *Relation Classification* (das Modellieren und anschließende Klassifizieren der Beziehung von Wörtern zueinander) oder *Sentiment Analysis* (das Erkennen positiver oder negativer Aussagen).

1 Dieses deutsche Sprachmodell, das entwickelte Toolkit und die generierten Test-Set Fragen stehen für weiterführende Anwendungen zum freien Download zur Verfügung. Müller, Andreas (o.J.), URL: http://devmount.github.io/GermanWordEmbeddings (Stand: 30.06.2015)

A Anhang

A.1 Training

Listing A.1: Liste der im Training verwendeten Deutschen Stoppwörter des Natural Language Toolkit (NLTK)

```
aber
alle
allem
allen
aller
alles
als
also
am
an
ander
andere
anderem
anderen
anderer
anderes
anderm
andern
anderr
anders
auch
auf
aus
bei
bin
bis
bist
da
damit
dann
der
den
des
dem
die
das
daß
derselbe
derselben
denselben
desselben
demselben
dieselbe
dieselben
dasselbe
dazu
dein
deine
deinem
deinen
deiner
deines
denn
derer
dessen
dich
dir
du
dies
diese
diesem
diesen
dieser
dieses
doch
dort
durch
ein
eine
einem
einen
einer
eines
einig
einige
einigem
einigen
einiger
einiges
einmal
er
ihn
ihm
es
etwas
euer
eure
eurem
euren
eurer
eures
für
gegen
gewesen
hab
habe
haben
hat
hatte
hatten
hier
hin
hinter
ich
mich
mir
ihr
ihre
ihrem
ihren
ihrer
ihres
euch
im
in
indem
ins
ist
jede
jedem
jeden
jeder
jedes
jene
jenem
jenen
jener
jenes
jetzt
kann
kein
keine
keinem
keinen
keiner
keines
können
könnte
machen
man
manche
manchem
manchen
mancher
manches
mein
meine
meinem
meinen
meiner
meines
mit
muss
musste
nach
nicht
nichts
noch
nun
nur
ob
oder
ohne
sehr
sein
seine
seinem
seinen
seiner
seines
selbst
sich
sie
ihnen
sind
so
solche
solchem
solchen
solcher
solches
soll
sollte
sondern
sonst
über
um
und
uns
unse
unsem
unsen
unser
unses
unter
viel
vom
von
vor
während
war
waren
warst
was
weg
weil
weiter
welche
welchem
welchen
welcher
welches
wenn
werde
werden
wie
wieder
will
wir
wird
wirst
wo
wollen
wollte
würde
würden
zu
zum
zur
zwar
zwischen
```

A.2 Test-Sets

Listing A.2: Syntaktische Analogiefragen (Auszug): Eine Frage (bestehend aus zwei Wortpaaren) pro Zeile, Zeile beginnend mit Doppelpunkt ist das Label des Test-Musters.

```
: nouns: SI/PL
Abbildung Abbildungen Ergebnis Ergebnisse
Abbildung Abbildungen Name Namen
Abbildung Abbildungen Person Personen
Abbildung Abbildungen Ziel Ziele
Abbildung Abbildungen Nacht Nächte
Absatz Absätze Wohnung Wohnungen
Absatz Absätze Boden Böden
Absatz Absätze Straße Straßen
Absatz Absätze Erfahrung Erfahrungen
Absatz Absätze Schule Schulen
[...]
: nouns: PL/SI
Abbildungen Abbildung Schulen Schule
Abbildungen Abbildung Orte Ort
Abbildungen Abbildung Minuten Minute
Abbildungen Abbildung Erfahrungen Erfahrung
Abbildungen Abbildung Bereiche Bereich
Absätze Absatz Herren Herr
Absätze Absatz Nächte Nacht
Absätze Absatz Universitäten Universität
Absätze Absatz Zeiten Zeit
```

A Anhang

```
Absätze Absatz Familien Familie
[...]
: adjectives: GR/KOM
ähnlich ähnlicher faul fauler
ähnlich ähnlicher ruhig ruhiger
ähnlich ähnlicher nett netter
ähnlich ähnlicher eckig eckiger
ähnlich ähnlicher alt älter
alt älter bestimmt bestimmter
alt älter früh früher
alt älter ähnlich ähnlicher
alt älter lang länger
alt älter wichtig wichtiger
[...]
: adjectives: KOM/GR
ähnlicher ähnlich eckiger eckig
ähnlicher ähnlich schärfer scharf
ähnlicher ähnlich trauriger traurig
ähnlicher ähnlich wahrscheinlicher wahrscheinlich
ähnlicher ähnlich langsamer langsam
älter alt genauer genau
älter alt voller voll
älter alt krümmer krumm
älter alt sauberer sauber
älter alt unterschiedlicher unterschiedlich
[...]
: adjectives: GR/SUP
ähnlich ähnlichste kalt kälteste
ähnlich ähnlichste schlank schlankeste
ähnlich ähnlichste nett netteste
ähnlich ähnlichste spannend spannendste
ähnlich ähnlichste schön schönste
alt älteste deutlich deutlichste
alt älteste hart härteste
alt älteste sauber sauberste
alt älteste schmal schmalste
alt älteste nett netteste
[...]
: adjectives: SUP/GR
ähnlichste ähnlich persönlichste persönlich
ähnlichste ähnlich langsamste langsam
ähnlichste ähnlich frischste frisch
ähnlichste ähnlich vollste voll
ähnlichste ähnlich wertvollste wertvoll
älteste alt liebste lieb
älteste alt häufigste häufig
älteste alt weiteste weit
älteste alt kürzeste kurz
älteste alt sonnigste sonnig
[...]
: adjectives: KOM/SUP
ähnlicher ähnlichste größer größte
ähnlicher ähnlichste sonniger sonnigste
ähnlicher ähnlichste schrecklicher schrecklichste
ähnlicher ähnlichste schmutziger schmutzigste
ähnlicher ähnlichste schneller schnellste
älter älteste schärfer schärfste
älter älteste gerader geradeste
älter älteste später späteste
älter älteste gerader geradeste
älter älteste fleißiger fleißigste
[...]
: adjectives: SUP/KOM
ähnlichste ähnlicher schnellste schneller
ähnlichste ähnlicher bestimmteste bestimmter
ähnlichste ähnlicher leichteste leichter
ähnlichste ähnlicher richtigste richtiger
ähnlichste ähnlicher schwierigste schwieriger
älteste älter müdeste müder
älteste älter stärkste stärker
älteste älter stillste stiller
älteste älter frischste frischer
älteste älter ruhigste ruhiger
[...]
: verbs (pres): INF/1SP
ändern ändere schauen schaue
ändern ändere wissen weiß
ändern ändere suchen suche
ändern ändere kommen komme
ändern ändere denken denke
arbeiten arbeite erhalten erhalte
arbeiten arbeite tragen trage
arbeiten arbeite erwarten erwarte
arbeiten arbeite fühlen fühle
arbeiten arbeite wohnen wohne
[...]
: verbs (pres): 1SP/INF
ändere ändern brauche brauchen
ändere ändern werde werden
ändere ändern schaffe schaffen
ändere ändern zeige zeigen
ändere ändern bleibe bleiben
arbeite arbeiten sehe sehen
arbeite arbeiten liege liegen
arbeite arbeiten stelle stellen
arbeite arbeiten muss müssen
arbeite arbeiten schließe schließen
[...]
: verbs (pres): INF/2PP
ändern ändert verstehen versteht
ändern ändert fehlen fehlt
ändern ändert erwarten erwartet
ändern ändert entstehen entsteht
ändern ändert wachsen wachst
arbeiten arbeitet fragen fragt
arbeiten arbeitet sprechen sprecht
arbeiten arbeitet sitzen sitzt
arbeiten arbeitet suchen sucht
arbeiten arbeitet fallen fallt
[...]
: verbs (pres): 2PP/INF
ändert ändern erkennt erkennen
ändert ändern geltet gelten
ändert ändern geltet gelten
ändert ändern vergleicht vergleichen
ändert ändern lebt leben
arbeitet arbeiten erscheint erscheinen
arbeitet arbeiten versucht versuchen
arbeitet arbeiten bekommt bekommen
arbeitet arbeiten ergebt ergeben
arbeitet arbeiten sitzt sitzen
[...]
: verbs (pres): 1SP/2PP
ändere ändert soll sollt
ändere ändert vergleiche vergleicht
ändere ändert treffe trefft
ändere ändert wohne wohnt
ändere ändert rede redet
arbeite arbeitet laufe lauft
arbeite arbeitet entwickle entwickelt
arbeite arbeitet gehöre gehört
arbeite arbeitet rede redet
arbeite arbeitet versuche versucht
[...]
: verbs (pres): 2PP/1SP
ändert ändere besteht bestehe
ändert ändere kommt komme
ändert ändere bietet biete
ändert ändere gewinnt gewinne
ändert ändere arbeitet arbeite
arbeitet arbeite fehlt fehle
arbeitet arbeite vergleicht vergleiche
```

```
arbeitet arbeite erkennt erkenne
arbeitet arbeite führt führe
arbeitet arbeite seht sehe
[...]
: verbs (past): INF/3SV
ändern änderte mögen mochte
ändern änderte gehören gehörte
ändern änderte entwickeln entwickelte
ändern änderte gelten galt
ändern änderte machen machte
arbeiten arbeitete ändern änderte
arbeiten arbeitete bleiben blieb
arbeiten arbeitete wollen wollte
arbeiten arbeitete kommen kam
arbeiten arbeitete brauchen brauchte
[...]
: verbs (past): 3SV/INF
änderte ändern musste müssen
änderte ändern spielte spielen
änderte ändern bot bieten
änderte ändern studierte studieren
änderte ändern redete reden
arbeitete arbeiten betraf betreffen
arbeitete arbeiten hieß heißen
arbeitete arbeiten saß sitzen
arbeitete arbeiten legte legen
arbeitete arbeiten fiel fallen
[...]
: verbs (past): INF/3PV
ändern änderten leben lebten
ändern änderten halten hielten
ändern änderten sagen sagten
ändern änderten erkennen erkannten
ändern änderten gelten galten
arbeiten arbeiteten sagen sagten
arbeiten arbeiteten können konnten
arbeiten arbeiteten lernen lernten
arbeiten arbeiteten müssen mussten
arbeiten arbeiteten gehören gehörten
[...]
: verbs (past): 3PV/INF
änderten ändern verbanden verbinden
änderten ändern handelten handeln
änderten ändern trafen treffen
änderten ändern lasen lesen
änderten ändern erschienen erscheinen
arbeiteten arbeiten folgten folgen
arbeiteten arbeiten saßen sitzen
arbeiteten arbeiten wussten wissen
arbeiteten arbeiten schrieben schreiben
arbeiteten arbeiten halfen helfen
[...]
: verbs (past): 3SV/3PV
änderte änderten sprach sprachen
änderte änderten verband verbanden
änderte änderten fand fanden
änderte änderten zeigte zeigten
änderte änderten nahm nahmen
arbeitete arbeiteten fiel fielen
arbeitete arbeiteten dachte dachten
arbeitete arbeiteten schrieb schrieben
arbeitete arbeiteten trug trugen
arbeitete arbeiteten dachte dachten
[...]
: verbs (past): 3PV/3SV
änderten änderte interessierten interessierte
änderten änderte lebten lebte
änderten änderte bildeten bildete
änderten änderte schlossen schloss
änderten änderte brauchten brauchte
arbeiteten arbeitete mochten mochte
arbeiteten arbeitete hatten hatte
arbeiteten arbeitete setzten setzte
arbeiteten arbeitete handelten handelte
arbeiteten arbeitete fragten fragte
```

Listing A.3: Thematische Analogiefragen (Auszug): Eine Frage (bestehend aus zwei Wortpaaren) pro Zeile.

```
China Yuan Deutschland Euro
China Yuan Dänemark Krone
China Yuan England Pfund
China Yuan Japan Yen
China Yuan Russland Rubel
China Yuan USA Dollar
Deutschland Euro Dänemark Krone
Deutschland Euro England Pfund
Deutschland Euro Japan Yen
Deutschland Euro Russland Rubel
Deutschland Euro USA Dollar
[...]
Berlin Deutschland Bern Schweiz
Berlin Deutschland Hanoi Vietnam
Berlin Deutschland Helsinki Finnland
Berlin Deutschland Kairo Ägypten
Berlin Deutschland Kiew Ukraine
Berlin Deutschland London England
Berlin Deutschland Madrid Spain
Berlin Deutschland Melbourne Australien
Berlin Deutschland Moskau Russland
Berlin Deutschland Oslo Norwegen
Berlin Deutschland Ottawa Kanada
Berlin Deutschland Paris Frankreich
Berlin Deutschland Rom Italien
Berlin Deutschland Stockholm Schweden
Berlin Deutschland Teheran Iran
Berlin Deutschland Tokio Japan
Berlin Deutschland Washington USA
[...]
England Europa Frankreich Europa
England Europa Griechenland Europa
England Europa Indien Asien
England Europa Italien Europa
England Europa Kanada Nordamerika
England Europa Polen Europa
England Europa USA Nordamerika
England Europa Vietnam Asien
England Europa Ägypten Afrika
Frankreich Europa Griechenland Europa
Frankreich Europa Indien Asien
Frankreich Europa Italien Europa
Frankreich Europa Kanada Nordamerika
Frankreich Europa Polen Europa
Frankreich Europa USA Nordamerika
Frankreich Europa Vietnam Asien
Frankreich Europa Ägypten Afrika
[...]
Frankreich Französisch Griechenland Griechisch
Frankreich Französisch Italien Italienisch
Frankreich Französisch Japan Japanisch
Frankreich Französisch Korea Koreanisch
```

```
Frankreich Französisch Norwegen Norwegisch
Frankreich Französisch Polen Polnisch
Frankreich Französisch Russland Russisch
Frankreich Französisch Schweden Schwedisch
Frankreich Französisch Spanien Spanisch
Frankreich Französisch Ukraine Ukrainisch
Griechenland Griechisch Italien Italienisch
Griechenland Griechisch Japan Japanisch
Griechenland Griechisch Korea Koreanisch
Griechenland Griechisch Norwegen Norwegisch
Griechenland Griechisch Polen Polnisch
Griechenland Griechisch Russland Russisch
Griechenland Griechisch Schweden Schwedisch
Griechenland Griechisch Spanien Spanisch
Griechenland Griechisch Ukraine Ukrainisch
[...]
Elisabeth Königin Charles Prinz
Android Google iOS Apple
Android Google Windows Microsoft
iOS Apple Windows Microsoft
[...]
Junge Mädchen König Königin
Junge Mädchen Mann Frau
Junge Mädchen männlich weiblich
Junge Mädchen Neffe Nichte
Junge Mädchen Onkel Tante
Junge Mädchen Papa Mama
Junge Mädchen Partner Partnerin
Junge Mädchen Prinz Prinzessin
Junge Mädchen Vater Mutter
König Königin Mann Frau
König Königin männlich weiblich
König Königin Neffe Nichte
König Königin Onkel Tante
König Königin Papa Mama
König Königin Partner Partnerin
König Königin Prinz Prinzessin
König Königin Vater Mutter
Mann Frau männlich weiblich
Mann Frau Neffe Nichte
Mann Frau Onkel Tante
Mann Frau Papa Mama
Mann Frau Partner Partnerin
Mann Frau Prinz Prinzessin
Mann Frau Vater Mutter
[...]
```

Listing A.4: Fragen zum inhaltlich nicht passenden Wort einer Wortreihe (Auszug): Eine Frage (bestehend aus drei zueinander passenden Wörtern und einem nicht passenden vierten Wort) pro Zeile.

```
August April September Jahr
August April September Monat
August April September Tag
August April September Stunde
August April September Minute
August April September Zeit
August April September Kalender
August April September Woche
August April September Quartal
August April September Uhr
Auto Motorrad Fahrrad Ampel
Auto Motorrad Fahrrad Fahrbahn
Auto Motorrad Fahrrad Fahrer
Auto Motorrad Fahrrad Fußgänger
Auto Motorrad Fahrrad Karte
Auto Motorrad Fahrrad Navigation
Auto Motorrad Fahrrad Polizei
Auto Motorrad Fahrrad Schild
Auto Motorrad Fahrrad Straße
Auto Motorrad Fahrrad Verkehr
Berlin München Frankfurt Amsterdam
Berlin München Frankfurt Brüssel
Berlin München Frankfurt Deutschland
Berlin München Frankfurt Indien
Berlin München Frankfurt Kopenhagen
Berlin München Frankfurt London
Berlin München Frankfurt Luxemburg
Berlin München Frankfurt Paris
Berlin München Frankfurt Washington
Berlin München Frankfurt Wien
Euro Rubel Yen Australien
Euro Rubel Yen China
Euro Rubel Yen Deutschland
Euro Rubel Yen England
Euro Rubel Yen Frankreich
Euro Rubel Yen Indien
Euro Rubel Yen Japan
Euro Rubel Yen Kanada
Euro Rubel Yen Russland
Euro Rubel Yen USA
Frankreich Deutschland England Afrika
Frankreich Deutschland England Amerika
Frankreich Deutschland England Asien
Frankreich Deutschland England Australien
Frankreich Deutschland England Brasilien
Frankreich Deutschland England China
Frankreich Deutschland England Europa
Frankreich Deutschland England Kanada
Frankreich Deutschland England Mexiko
Frankreich Deutschland England USA
Hase Hund Katze Baum
Hase Hund Katze Besitzer
Hase Hund Katze Elefant
Hase Hund Katze Essen
Hase Hund Katze Haus
Hase Hund Katze Mensch
Hase Hund Katze Tier
Hase Hund Katze Tierheim
Hase Hund Katze Wiese
Hase Hund Katze Zoo
Herz Lunge Leber Arzt
Herz Lunge Leber Blut
Herz Lunge Leber Fuß
Herz Lunge Leber Gesundheit
Herz Lunge Leber Hand
Herz Lunge Leber Kopf
Herz Lunge Leber Krankenhaus
Herz Lunge Leber Krankheit
Herz Lunge Leber Körper
Herz Lunge Leber Organ
Montag Mittwoch Freitag Jahr
Montag Mittwoch Freitag Monat
Montag Mittwoch Freitag Tag
Montag Mittwoch Freitag Stunde
Montag Mittwoch Freitag Minute
Montag Mittwoch Freitag Zeit
```

```
Montag Mittwoch Freitag Kalender
Montag Mittwoch Freitag Woche
Montag Mittwoch Freitag Wochentag
Montag Mittwoch Freitag Uhr
[...]
Twitter Facebook Instagram Android
Twitter Facebook Instagram App
Twitter Facebook Instagram Computer
Twitter Facebook Instagram Domain
Twitter Facebook Instagram Internet
Twitter Facebook Instagram iOS
Twitter Facebook Instagram Laptop
Twitter Facebook Instagram Microsoft
Twitter Facebook Instagram Netzwerk
Twitter Facebook Instagram Software
Windows Linux Android App
Windows Linux Android Computer
Windows Linux Android Gerät
Windows Linux Android Laptop
Windows Linux Android Programm
Windows Linux Android Rechner
Windows Linux Android Smartphone
Windows Linux Android Software
Windows Linux Android Tablet
Windows Linux Android Technik
```

Listing A.5: Analogiefragen zum Gegenteil eines Wortes (Auszug): Eine Frage (bestehend aus zwei Wortpaaren) pro Zeile.

```
Frage Antwort stark schwach
Frage Antwort viel wenig
Frage Antwort positiv negativ
Frage Antwort davor danach
Frage Antwort nah fern
Frage Antwort männlich weiblich
Frage Antwort warm kalt
Frage Antwort rechts links
Frage Antwort schnell langsam
Frage Antwort Junge Mädchen
[...]
Leben Tod hoch tief
Leben Tod davor danach
Leben Tod Norden Süden
Leben Tod bekannt unbekannt
Leben Tod rechts links
Leben Tod groß klein
Leben Tod warm kalt
Leben Tod männlich weiblich
Leben Tod Osten Westen
Leben Tod Sommer Winter
Mann Frau groß klein
Mann Frau schnell langsam
Mann Frau davor danach
Mann Frau lang kurz
Mann Frau oben unten
Mann Frau männlich weiblich
Mann Frau warm kalt
Mann Frau Osten Westen
Mann Frau bekannt unbekannt
Mann Frau hell dunkel
Norden Süden hell dunkel
Norden Süden gewinnen verlieren
Norden Süden groß klein
Norden Süden oben unten
Norden Süden Osten Westen
Norden Süden lang kurz
Norden Süden viel wenig
Norden Süden positiv negativ
Norden Süden Mann Frau
Norden Süden stark schwach
[...]
alt jung leicht schwer
alt jung früh spät
alt jung bekannt unbekannt
alt jung rechts links
alt jung Osten Westen
alt jung nah fern
alt jung Norden Süden
alt jung Tag Nacht
alt jung Junge Mädchen
alt jung davor danach
bekannt unbekannt leicht schwer
bekannt unbekannt rechts links
bekannt unbekannt Osten Westen
bekannt unbekannt alt jung
bekannt unbekannt schnell langsam
bekannt unbekannt Leben Tod
bekannt unbekannt viel wenig
bekannt unbekannt Mann Frau
bekannt unbekannt lachen weinen
bekannt unbekannt früh spät
[...]
leicht schwer Sommer Winter
leicht schwer warm kalt
leicht schwer hell dunkel
leicht schwer Junge Mädchen
leicht schwer Mann Frau
leicht schwer Leben Tod
leicht schwer früh spät
leicht schwer oben unten
leicht schwer lachen weinen
leicht schwer Start Ziel
männlich weiblich stark schwach
männlich weiblich Tag Nacht
männlich weiblich bekannt unbekannt
männlich weiblich lang kurz
männlich weiblich hoch tief
männlich weiblich nah fern
männlich weiblich rechts links
männlich weiblich Mann Frau
männlich weiblich Start Ziel
männlich weiblich schnell langsam
rechts links Frage Antwort
rechts links Mann Frau
rechts links hoch tief
rechts links alt jung
rechts links positiv negativ
rechts links früh spät
rechts links Start Ziel
rechts links oben unten
rechts links Junge Mädchen
rechts links lachen weinen
[...]
```

Literaturverzeichnis

[1] Kheira Leila Arras, Jan Heyd, The-Anh Ly, and Andreas Müller. Exploring semantic word similarities in German News Articles. 2014.

[2] R.E. Bellman. *Adaptive control processes: A guided tour*. Princeton University Press (Princeton, NJ), 1961.

[3] R. Bender, A. Ziegler, and St. Lange. Logistische Regression. (14):12–14, 2002.

[4] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.

[5] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy Layer-Wise Training of Deep Networks. *Advances in neural information processing systems*, 19(1):153, 2007.

[6] Peter F. Brown, Peter V. DeSouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. Class-Based n-gram Models of Natural Language. *Computational Linguistics*, (1950), 1992.

[7] Michael Curtiss, Iain Becker, and Tudor Bosman. Unicorn: a system for searching the social graph. Technical report, 2013.

[8] Joshua Goodman. A Bit of Progress in Language Modeling. page 73, 2001.

[9] Zellig Harris. *Distributional Structure*, volume 1. 1954.

[10] J J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America*, 79(8):2554–2558, 1982.

[11] H Kučera and W N Francis. *Computational Analysis of Present-Day American English*. Brown University Press, 1967.

[12] T Mikolov, M Karafiat, L Burget, J Cernocky, and S Khudanpur. Recurrent Neural Network based Language Model. *Interspeech*, (September):1045–1048, 2010.

[13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. pages 1–9, 2013.

[14] Tomas Mikolov, Greg Corrado, Kai Chen, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *Proceedings of the International Conference on Learning Representations (ICLR 2013)*, pages 1–12, 2013.

[15] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic Regularities in Continuous Space Word Representations. 2012.

[16] Zach Miners. Yahoo buys SkyPhrase to better understand natural language. *PCWorld*, 2013.

[17] Marvin Minsky and Seymour Papert. *Perceptron - An Essay in Computational Geometry*. MIT Press, 1969.

[18] Frederic Morin and Y Bengio. Hierarchical probabilistic neural network language model. *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*, pages 246–252, 2005.

[19] Karl Pearson. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2:559–572, 1901.

[20] F Peng. Augmentating Naive Bayes Classifiers with Statistical Language Models. *Computer Science Department Faculty Publication Series*, Paper 91, 2003.

[21] Xin Rong. word2vec Parameter Learning Explained. pages 1–19.

[22] F Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386–408, 1958.

[23] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation, 1986.

[24] David Shapiro, Doug Platts, and Magico Martinez. Google hummingbird explained. *icrossing*, 2013.

[25] Richard Socher, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. 2013.

[26] Bernard Widrow. An Adaptive 'Adaline' Neuron Using Chemical 'Memistors', 1960.

[27] Daojian Zeng, Kang Liu, Siwei Lai, Guangyou Zhou, and Jun Zhao. Relation Classification via Convolutional Deep Neural Network. pages 1–10, 2014.