RUPRECHT-KARLS-UNIVERSITÄT HEIDELBERG

BACHELORARBEIT

# Analyse der Relationsvorhersage im Deutschen mit Wortvektoren

*Autor:*
Dennis ULMER

*Betreuende:*
Dr. Yannick VERSLEY
Dr. Viviana NASTASE

*Eine Arbeit zur Erlangung des Bachelorgrades*

1. Juli 2016

„*I have not failed. I've just found 10,000 ways that won't work.*“

Thomas A. Edison

RUPRECHT-KARLS-UNIVERSITÄT HEIDELBERG

# *Zusammenfassung*

Neuphilologische Fakultät
Institut für Computerlinguistik

Bachelor of Arts

**Analyse der Relationsvorhersage im Deutschen mit Wortvektoren**

von Dennis ULMER

In sog. *Wissensdatenbanken* wird Wissen über die Beziehungen von Entitäten in dieser Welt beschrieben. Dieses Wissen ist für vielerlei nützlich, um Maschinen das fehlende Weltwissen bei Computersystemen zu kompensieren. Die Erstellung solcher Datenbanken ist jedoch sehr arbeitsintensiv, weshalb sich Bemühungen, diese automatisch zu ergänzen, lohnen können. Diese Abschlussarbeit wirft einen Blick auf verschiedene Ansätze in diesem Bereich und versucht darüber hinaus, ähnliche Verfahren auf der Grundlage von Wortvektorenrepräsentationen, die im Zuge des Deep-Learning-Trends in den letzten Jahren erhöhte Aufmerksamkeit in der NLP-Gemeinschaft erfahren haben, zu verwirklichen.

RUPRECHT-KARLS UNIVERSITY HEIDELBERG

# *Abstract*

Faculty of Modern Languages
Department for Computational Linguistics

Bachelor of Arts

**Analysis of relation prediction in German using word vector representations**

by Dennis ULMER

So-called *knowledge bases* contain knowledge about the relations between entities in this world. This knowledge is very useful to compensate the inherent lack of universal knowledge of any computer system. However, the creation of such databases is deemed to be very labour-intensive. Thus, automatic efforts to replenish them might be worthwile.
This thesis aims to compare various approaches in this area and furthermore wants to realize similar procedures based on word embeddings, which gained a lot of popularity in recent years among the NLP community.

# *Danksagung*

The acknowledgments and the people to thank go here, don't forget to include your project advisor...

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Kapitel 1

# Einleitung

> *"Weltwissen beschreibt das einem Individuum verfügbare allgemeine Wissen, Kenntnisse und Erfahrungen über Umwelt und Gesellschaft. [...] Das Weltwissen ermöglicht es, neue Tatsachen einzuordnen und entsprechend zu handeln, auch wenn detaillierte Informationen fehlen. [...]*
>
> *Auch in der Robotik [und KI-Forschung; Anm. des Autors] spielt Weltwissen [...] eine Rolle, da Computer [...] nicht selbst über Weltwissen verfügen."*
>
> EINLEITUNG DES ARTIKELS ÜBER WELTWISSEN, WIKIPEDIA[1]

## 1.1 Knowledge Graph Completion

Computer sind dem Menschen mittlerweile beim Lösen vielerlei Aufgaben überlegen. Sie rechnen schneller und genauer. Sie können riesige Datenmengen in einem Bruchteil der Zeit verarbeiten, die ein Mensch dafür bräuchte. Die menschliche Überlegenheit beginnt auch in Bereichen zu bröckeln, bei denen der Einsatz von Computern lange für unmöglichkeit: Menschliche Champions scheitern nun gegen Maschinen beim Schach. Zuletzt scheiterte auch der Mensch auch beim Spiel Go gegen ein "intelligentes" System. In anderen Bereichen jedoch hinken die Maschinen den Prognosen hinterher. Viele dieser Bereiche haben dabei eines gemeinsam: Die Anforderung an das System, nicht nur einfache Rechenoperationen auszuführen, sondern sich ein Bild von der umgebenden Welt zu machen, Schlüsse zu ziehen und neues Wissen zu Erwerben und passend in den vorhandenen Informationsbestand einzuordnen.
Entwicklungen wie die ersten Fahrten fahrerloser Autos, den Jeopardy-Champion Watson und ähnliche zeigen den Fortschritt in diesem Bereich auf, jedoch ist die Wissenschaft von einer allgemeinen Intelligenz noch weit entfernt. Ein Grund dafür ist das Problem von Computern, dass sie im Gegensatz zum Menschen über kein Weltwissen verfügen, welches letztere sich im Laufe ihres Lebens aneignen. Dabei lernen sie

- wie Objekte in der Realwelt zueinander in Beziehung stehen
- welche Attribute von verschiedenen Objekten besessen werden
- Zusammenhänge zwischen Ereignissen zu verstehen
- Pläne zu schmieden und sich Stragien zurechtzulegen

[1] https://de.wikipedia.org/wiki/Weltwissen (zuletzt abgerufen am 03.03.16)

- ...

Ersteres wird in der Informatik durch sog. *Ontologien* (= formale Darstellung einer Beziehung zwischen Elementen einer Menge) modelliert. Zieht man zwischen den Entitäten in der Welt nun derartige Ontologien, entsteht ein Graph, in dem die Beziehungen der Knoten untereinander mithilfe verschiedener Arten von Kanten kodiert sieht, dem *Knowledge Graph*.
Die Vervollständigung ebendieses ermöglicht Systemen, die langwierige menschliche Erlernung dieses Wissens zu kompensieren. In dieser Arbeit soll deshalb ein Ansatz vorgestellt werden, der die Lösung diesen Problems einen kleinen Schritt näherbringen könnte.

## 1.2 Ansatz

Innerhalb der letzten Jahre haben neurale Netze in der Informatik im Allgemeinen und in der Computerlinguistik im Speziellen eine Renaissance erlebt. Mit diesen konnten eine neue Art von Wortvektoren, auf Englisch "word embeddings" trainiert werden, die semantische Information in sich kodierten. Zwar zeigen einige Untersuchung, dass sich dieser Ansatz älteren durchaus sehr ähnlich ist und nicht unbedingt zu besseren Ergebnissen führt, der Aufruhr hat aber eine neue Welle von Forschungen im Bereich der distributionellen Semantik ausgelöst.
Ein oft zitiertes Beispiel für die Ausdruckskraft dieser Vektoren ist das Entdecken von semantischen Relationen hinter einfachen arithmetischen Operationen:

$$\vec{v}(King) - \vec{v}(Man) + \vec{v}(Woman) \approx \vec{v}(Queen)$$

Zwar lassen sich dadurch nicht alle Arten von Relationen aus Wortvektoren extrahieren und beschränken sich die Beispiele bei dieser Herangehensweise auf 1:1-Relationen (1:N-, N:1- sowie N:N-Relationen lassen sich auf andere Art und Weise finden), jedoch lassen sie schon ein gewisses Potenzial erahnen. Eine Reihe von Papern nutzt nun Methoden des maschinellen Lernens, um mithilfe von Trainingsdaten die Differenzvektoren für bestimmte, im Voraus ausgewählte Relationen zu trainieren und die Ergebnisse in einem Testset anzuwenden.
Die Idee, die in dieser Arbeit angegangen werden soll, fußt auf der Hypthese, dass das Trainieren solcher Relationen nicht möglich wäre, wenn sich die Differenzvektoren von Wortpaaren einzelner Relationen nicht ähneln würden, also z.B.

$$\vec{v}(Berlin) - \vec{v}(Germany) \approx \vec{v}(Paris) - \vec{v}(Frankreich) \approx \vec{v}(Madrid) - \vec{v}(Spain)$$

Betrachtet man die Abstandsvektoren von Wortpaaren als Punkte in einem eigenen Vektorraum, so müssten theoretisch die Punkte, die zu Ländern und deren Hauptstädten gehören, in diesem Raum nahe beieinander liegen. Dies wiederum könnte man dann insofern ausnutzen, indem man ein Clusteringverfahren anwendet, um diese Gruppen zu identifizieren und die Elemente jeder Gruppe danach mit der entsprechenden Relation in einen Knowledge Graph einzuordnen. Dies hätte zwei Vorteile:

1. Teile des kostenintensiven Dateneinpflegens durch menschliche Hilfe fällt weg

2. Es wird möglich abzuschätzen, welche Relationen in einem Raum von Wortvektoren tatsächlich festgehalten werden (darunter vielleicht auch einige, die man bei vorherigen Experimenten nicht berücksichtigt hatte)

Die Ergebnisse, die diese Prozedur zutage fördert sowie die Fallstricke, die sie mit sich bringt, werden in den nächsten Kapiteln beschrieben. Über jene wird nun ein kleiner Überblick gegeben.

## 1.3 Inhalt

In Kapitel 2 dieser Arbeit sollen verwandte Arbeiten zu diesem Thema vorgestellt werden. Darauf folgt eine Beschreibung der ein Vorbereitungsschritte, die erforderlich waren (siehe Kapitel 3) sowie eine Charakterisierung der verwendeten Ressourcen. Dabei wird auch auf das Training verschiedener Wortvektoren eingegangen, die in Kapitel 4 auf qualitative und quantitative Art und Weise auf ihre semantische Aussagekraft evaluiert werden.
Das Kapitel Nummer 5 handelt von Mapping-Vorgang, bei dem aus den Wortvektoren in einem neuen Vektorraum Punkte entstehen, die jeweils einem Wortpaar entsprechen. Dabei wird auf das theoretische Verfahren genauso eingegangen wie auch auf den benötigen Algorithmus und notwendige Einschränkungen.
In Kapitel 6 wird der Clustering-Schritt im Bezug auf die Wahl des Algorithmus, seiner Parameter und seiner Skalierbarkeit beschrieben. Daran schließt in Kapitel 7 die Evaluation der entstehenden Cluster an, bevor in Kapitel 8 die Ergebnisse sowie die Möglichkeiten und Grenzen des Verfahrens ausdrücklich diskutiert werden.
Die Arbeit schließt mit einem Fazit (Kapitel **??**) und einem Ausblick für auf diese Arbeit aufbauende Untersuchungen an (Kapitel **??**).

# Kapitel 2

# Verwandte Arbeiten

## 2.1 Wortvektorrepräsentationen

*Word embeddings* (="Worteinbettungen") wurden zuerst von (Bengio et al., 2006) noch vor dem erneuten Deep-Learning-Boom 2006 präsentiert. Im Vergleich zu vorherigen Arbeiten mit sog. "One-Hot"-Vektoren, bei denen jede Dimension einem Wort des Vokabulars eines Korpus zugeordnet ist und nur eine davon den Wert $1$ besitzt (gewöhnlich besitzen alle anderen den Wert $0$), sind Informationen über ein Wort in dieser Repräsentation für den Menschen uneindeutig und kontinuerlich über den Vektor verteilt ("distributed represenation"), woher sich auch der Name des dazugehörigen Forschungsfeldes - der distributionellen Semantik - herleitet. Dies ist aber eigentlich nur die Folge eines viel grundlegeren Unterschied der Methodik, der von (Baroni, Dinu und Kruszewski, 2014) weiter ausgeführt: Lange hatte man sich mit zählbasierten Methoden beschäftigt, um solche Repräsentationen zu erstellen; mit (Bengio et al., 2006) hielten vorhersagebasierte Methoden schließlich Einzug.
Diese fußt auf der *Distributional hypthesis*, die besagt, dass Worte, die in ähnlichen Kontexten auftauchen, dazu tendieren, eine ähnliche Bedeutung zu haben (Harris, 1954). Diese Besonderheit macht sich (Bengio et al., 2006) zunutze, da das neurale Netz, mit denen die Wortvektorrepräsenationen trainiert werden, sich beim Bewegen durch einen Trainingskorpus ein mehrwortiges Kontextfenster um ein Zielwort herum verwendet. Das System lernt, die Wortvektoren an häufig vorkommende Wortkontexte anzupassen. Selbige besitzen zudem weitere nützliche Eigenschaften, die in Kapitel 3 beschrieben werden.
Diese Forschungsarbeit löste eine Welle weiterer Forschungen in diesem Gebiet aus, da gezeigt werden konnte, dass diese Art der Wortrepräsentationen genutzt werden konnte, die Leistungen von Systemen selbst im Bezug auf lange bekannte und erforschte Probleme der Computerlinguistik signifikant verbessern konnte. Exemplarisch sei hier die Arbeit von (Collobert et al., 2011) genannt, die so neue Ansätze für PoS-Tagging, Named-Entity-Recognition, Chunking und Semantic Role Labeling präsentieren.

Weitere Untersuchungen beschäftigen sich u. A. mit bilingualen Repräsentationen für maschinelle Übersetzung (Zou et al., 2013), deren Training auf der Basis von einem dependenzgeparsten Korpus (Levy und Goldberg, 2014), dem Optimieren der Parameter (Levy, Goldberg und Dagan, 2015) u.v.m.

## 2.2 Vektorrepräsentationen für Wissensdatenbank

Entitäten und Relationen aus Wissensdatenbanken mithilfe eigener Vektoren zu respräsentieren wurde bereits von einigen Forschungsarbeiten in Angriff genommen: Als einer der ersten Ansätze versuchen (Bordes et al., 2011), symbolische Wissensrepräsentationen in einen Vektorraum einzubetten und so für die Künstliche-Intelligenz-Forschung leichter nutzbar zu machen. Diese sog. *Structured Embeddings* (SE) werden mithilfe von neuralen Netzwerken trainiert. Darauf aufbauend präsentieren (Bordes et al., 2013) einen etwas simpleren Ansatz namens *TransE*, der darauf abzieht, Relationen zwischen Entitäten als eine einfach vektorarithmetische Operation (= Übersetzung) zu sehen. Diese Vorgehensweise wird in Kapitel 6 etwas genauer ausgeführt und dere Ergebnisse für einen deutschsprachigen Datensatz repliziert.

| MODELL | SCORING-FUNKTION |
|---|---|
| *TransE* | $f_r(h,t) = \| h + r - t \|_2^2$ |
| *TransH* | $f_r(h,t) = \| h_\perp + r - t_\perp \|_2^2$ |
| *TransR* | $f_r(h,t) = \| h_r + r - t_r \|_2^2$<br>$h_r = hM_r$<br>$t_r = tM_r$ |
| *CTransR* | $f_r(h,t) = \| h_{r,c} + r_c - t_{r,c} \|_2^2 + \alpha \| r_c - r \|_2^2$<br>$h_{r,c} = hM_r$<br>$t_{r,c} = tM_r$ |

**Erklärung der Parameter**

- $(h, r, t)$: Relationstripel bestehend aus einer Kopf- ($h$) und Fußentität ($t$) sowie einer Relation $r$
- $f_r(\cdot, \cdot)$: Scoringfunktion zweier Entitäten für eine Relation $r$
- $h_\perp, t_\perp$: Projektionen zweiter Entitätsvektoren auf eine Ebene
- $M$: Projektionsmatrix
- $(h_{r,c}, r_c, t_{r,c})$: Relationstripel mit auf ein Subcluster einer Relation trainieren Vektorrepräsentationen
- $\alpha$: Gewichtsparameter zur Einschränkung für den Abstand einer Subrelation zur Hauptrelation

**Abbildung 2.1:** Übersicht über verschiedene Arten der Vektorrepräsentationen für Wissensdatenbanken Für TransE (Bordes et al., 2013), TransH (Wang et al., 2014), TransR und CTransR (Lin et al., 2015) werden die verschiedenen Scoringfunktionen gegenübergestellt und vorkommende Parameter erklärt.

Diese Idee wird weiter von (Wang et al., 2014) vorangetrieben: Um nicht nur 1:1- sondern auch 1:n-, n:1- und m:n-Relationen abzubilden, werden Punkte in einem Vektorraum auf eine für jede Relation separat gelernte Ebene projiziert, woher sich der Name des Verfahrens *TransH*, herleitet.
Um die Vektorräume von Entitäten und Relationen zu trennen, stellen (Lin et al., 2015) *TransR* und *CTransR* vor. Für Ersteres wird für jede Relation eine Projektionsmatrix gelernt, die Entitäten in den jeweiligen Relationsvektorraum übersetzt. Bei Letzterem werden für jede Relation mehrere Vektoren trainiert, um der Unterschiedlichkeit im Kontext anderer Entitäten gerecht zu werden. Eine Übersicht über die Berwertungsfunktionen aller Verfahren findet sich in Abbildung 2.1.

# Kapitel 3

# Grundlagen

> *To deal with hyper-planes in a 14-dimensional space, visualize a 3-D space and say "fourteen" to yourself very loudly. Everybody does it.*
>
> UNKNOWN

## 3.1 Neurale Netzwerke

Wie der Name bereits erahnen lässt, fußt die Idee von neuralen Netzwerken auf einer mathematischen Modellierung des menschlichen Gehirns (siehe Abbildung 3.1). Die Grundbausteine bilden dabei einzelne Neuronen, die dabei auf folgende Art und Weise modelliert werden (dieser Teil ist dabei auch als *Perceptron*-Algorithmus bekannt):

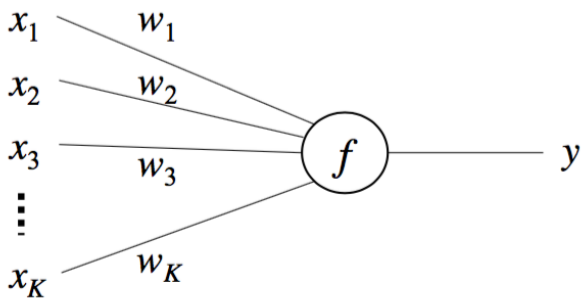



**Abbildung 3.1:** Mathematische Modellierung eines Neurons, auch bekannt als *Perceptron*-Algorithmus. $x$ bezeichnet In-, $y$ den Output des Neurons, $w$ bilden die Gewichte und $f$ die Aktivierungsfunktion. Abbildung aus (Rong, 2014).

Der Analogie der menschlichen Nervenzelle folgend erhält das Neuron mehrere Inputs in Form eines Vektors $x$ mit $K$ Dimensionen. Der Output erhält die Bezeichnung $y$. Um den Output zu bestimmen enthält die Zelle eine Aktivierungsfunktion $f$ (um zu bestimmen, wann das Neuron "feuert"):

$$y = f(u) \tag{3.1}$$

$u$ bezeichnet dabei einen Skalar, dem man durch das Summieren der mit $w$ gewichteten Inputs erhält:

$$u = \sum_{i=0}^{K} w_i x_i = w^T x \tag{3.2}$$

Für die Aktivierungsfunktion $f$ bieten sich mehrere Optionen. Ursprünglich wurde die *Heaviside step function* oder *Rectifier (ReLU)* gewählt:

$$f(u) = \begin{cases} 1 & \text{if } u > 0 \\ 0 & \text{otherwise} \end{cases} \text{ bzw.} \tag{3.3}$$

$$f(u) = max(0, u) = \begin{cases} 0 & \text{if } u < 0 \\ u & \text{otherwise} \end{cases} \tag{3.4}$$

Andere Funktionen sind z.B. *sigmoid* ($\sigma(u) \in [0, 1]$) und *tanh* ($tanh(u) \in [-1, 1]$), im Gegenteil zu den beiden zuvor sind diese durch ihre s-förmige Form kontinuierlich und dadurchableitbar:

$$\sigma(u) = \frac{1}{1 + e^{-u}} \tag{3.5}$$

$$tanh(u) = \frac{e^{2u} - 1}{e^{2u} + 1} \tag{3.6}$$

Um aus einzelnen Neuronen nun ein neurales Netzwerk zu kreieren, werden mehrere Schichten erstellt (i.d.R. eine Eingabe- (*input layer*), eine Ausgabe- (*output layer*) und min. eine "versteckte" Schicht (*hidden layer*). Die Schichten bestehen dann aus mehreren Nervenzellen, wobei jede Zelle mit jeder anderen Zelle der vorhergehenden und folgenden Schicht vernetzt ist (siehe Abbildung 3.2).

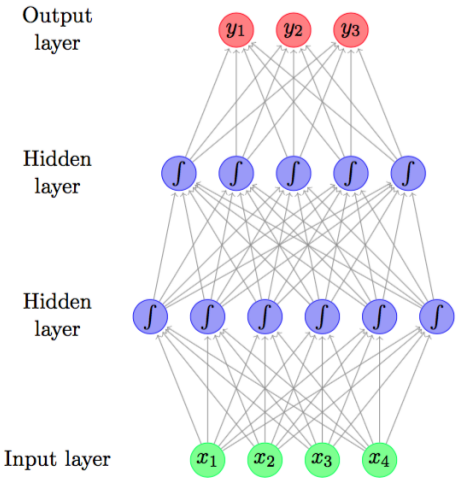



**Abbildung 3.2:** Darstellung eines neuralen Netzwerks mit vier Schichten, davon jeweils eine für In- und Output, sowie zwei "versteckte" Schichten (*hidden layer*). Bild aus (Goldberg, 2015).

Die einzelnen Gewichtsvektoren $w$ aller Neuronen einer Schicht werden dann zu einer Gewichtsmatrix $W$ zusammengefasst, wobei jede Zeile einem Gewichtsvektor entspricht. Zum Training dieser Strukturen wird der sog. *Backpropagation*- Algorithmus verwendet, der mithilfe einer vorher definierten Verlustfunktion (die anzeigt, inwiefern das durch das Netzwerk erzeugt Ergebnis von dem erwünschtne Ergebnis abweicht) einen Fehler errechnet und diesen dann rekursiv durch alle Schichten des Netzwerks zurückgibt und simultan die Werte innerhalb der Gewichtsmatrizen anpasst.

## 3.2 Wortvektoren

Frühere Experimente mit Wortvektoren arbeiteten meist mit sog. “One-Hot”-Vektoren, bei denen jede Dimension i.d.R. einem betimmten Wort zugeordnet wurde. Nehmen wir beispielsweise das Minikorpus *Der Hund beißt den Mann* an. Das Vokabular besteht dann aus V $= \{beit, Der, den, Hund, Mann\}$ (in alphabetischer Reihenfolge). Um jedes dieser Wörter als einen der oben genannten Vektoren zu repräsentieren, können wir Vektoren der Länge $V$ ($V$ steht eigentlich für ||V||, wird der Übersicht halber aber im Folgenden stellvertrentend dafür verwendet) benutzen. Um nun zum Beispiel einen Vektor für *Hund* zu generieren, setzen wir den Wert der Stelle des Vektors (= *Feature*) auf $1$, der dem Index von Hund in V entspricht, also $\vec{v}(Hund) = (0, 0, 0, 1, 0)$.
Diese Art von Wortvektor wird gemeinhin als “sparse”, also spärlich bezeichnet, da sie relativ wenig Information enthält. Wortvektoren, die mithilfe von Neuralen Netzwerken trainiert werden (im Englischen zur Abgrenzung *word embeddings* genannt), beinhalten mehr (semantische) Informationen über das dazugehörige Wort, zudem lassen sich einzelne Features nicht mehr auf eindeutig auf bestimmte Worte zurückführen.

Das Training dieser neuen Wortvektoren läuft folgendermaßen ab: Als Input fungieren die erwähnten “One-Hot”-Vektoren, welche genau so viele Dimensionen wie Worte im Vokabular besitzen ($\vec{v} \in \mathbb{R}^V$). Die Modelle bestehen aus drei Schichten, namentlich *Input*, *Hidden* und *Output*. Input und Output besitzen die Dimensionalität von $V$), Hidden die von $N$.

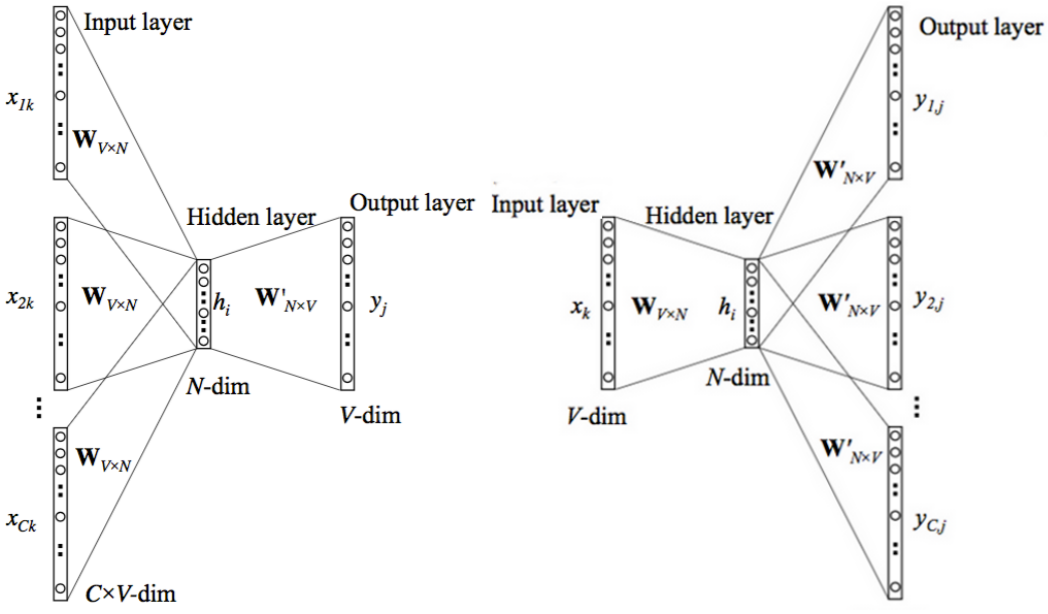



**Abbildung 3.3:** Gegenüberstellung der beiden Trainingsmethoden CBOW (links) und Skip-Gram (rechts). CBOW versucht die Wahrscheinlichkeit eines Wortes gegeben seines Kontexts zu trainieren, Skip-Gram die Wahrscheinlichkeit eines Kontextes gegeben eines Wortes.

Zwischen Input und Hidden liegt die Gewichtsmatrix $W$ und zwischen Hidden und Output die Matrix $W'$. CBOW versucht, die Wahrscheinlichkeit

eines Wortes gegeben eines Kontextes der Größe $C$ zu maximieren (Kontext bezieht sich in diesem Fall auf die Summe der rechts und links vom Eingabewort stehenden Wörter, siehe Abbildung 3.3). Unsere Verlustfunktion, deren Wert es dabei zu minimieren gilt, besteht darin in der negativen logarithmischen Wahrscheinlichkeit eines Wortes gegeben seines Kontextes:

$$E = -log\ p(w_O|w_{I,1}, \ldots, w_{I,C}) \tag{3.7}$$

Beim Skip-gram-Modell verhält sich das Ganze genau umgekehrt, es wird versucht, den Kontext gegeben eines Eingabewortes vorherzusagen:

$$E = -log\ p(w_{I,1}, \ldots, w_O) \tag{3.8}$$

In beiden Fällen wird daraufhin überprüft, ob die Vorhersage mit den tatsächlichen Daten übereinstimmt und die Abweichung errechnet, mit der dann die Parameter der Gewichtsmatrizen $W$ und $W'$ rekursiv angepasst werden, um zukünftige Prognosen zu verbessen, wofür der *Backpropagation*-Algorithmus verwendet wird. Die Wortvektoren, die dann nach dem Abarbeiten aller Trainingsdaten resultieren, sind dann die Zeilen von $W'$, wobei die *i*-te Zeile der Matrix dem Wortvektor des Wortes mit dem Index *i* im Vokabular entspricht.
Eigentlich erfordert das Training eine aufwendige Berechnung über alle Wörter des Vokabels, was der Skalierbarkeit dieses Verfahrens entgegensteht. Der Aufwand kann allerdings durch Techniken wie *Hierarchisches Softmax*, bei dem der Aufwand durch einen binären Baum von $O(V)$ auf $O(log\ V)$ reduziert wird sowie *Negativem Sampling*. Bei letzterem werden “schlechte” (also Negativ-)Beispiele zum Training hinzugezogen, woher sich auch der Name des Verfahrens ableitet.

# Kapitel 4

# Vorbereitung

## 4.1 Vorbereitung

Bla

### 4.1.1 Extraktion von Named Entities

### 4.1.2 Aufbereitung des Korpus

Als Textressource wurde das DECOW14X-Korpus (DE = Deutsch, COW = "**CO**rpus from the **W**eb") verwendet. Dieses Korpus von (Schäfer und Bildhauer, 2012) besteht aus 21 Texten, die in den Jahren 2011 und 2014 von deutschsprachigen Internetseiten gecrawled und aufbereitet wurden. Dies beinhaltet PoS-Tagging, Chunking, Lemmatisierung, das Markieren von Eigennamen (Named Entities) und dem Hinzufügen von Metadaten. Die Sätze liegen darin im CoNLL-Format[1] vor, wobei jedem Wort und dessen Annotationen eine ganze Zeile gewidmet ist, Stazgrenzen werden durch XML-Tags getrennt. Summa summarum enthält das Korpus 624.767.747 Sätze mit 11.660.894.000 Tokens.

Für diese Arbeit wurden auf Basis der Ressource drei Version für das Training der Wortvektoren erstellt:

- Eine Datei mit den originalen Tokens durch Leerzeichen getrennt, je ein Satz pro Zeile.
- Eine Datei mit den lemmatisierten Tokens durch Leerzeichen getrennt, je ein Satz pro Zeile.
- Eine Datei mit dem lemmatisierten Tokens, sortiert nach den für jeden Satz geparsten Dependenzen, ein Satz pro Zeile.

Die Dependenzen wurden dabei mit dem Tool X erzeugt. [Blabla erläutern wenn Punkt erledigt.]

### 4.1.3 Training der Wortvektoren

Wortvektoren werden mithilfe des Tools *word2vec* und zwei verschiedenen Modellen trainiert: Continuous-Bag-of-Words (CBOW) und Skip-Gram. Das CBOW-Model wurde zuerst von (Mikolov et al., 2013) vorgestellt. Die Erklärung der Funktionsweise wird im nachfolgenden Teil recht klein gehalten, für eine ausführlichere und verständliche Ausführung wird beispielsweise

[1] Siehe http://ilk.uvt.nl/conll/ (zuletzt abgerufen am 11.04.16)

die Arbeit von (Rong, 2014) empfohlen.

Zum Training der Vektoren wurde das C-Tool *Word2Vec* von (Mikolov et al.) verwendet. Als Eingabe benötigt es eine Textressource, die einen Satz pro Zeile enthält, Tokens durch Leerzeichen getrennt und gibt die Wortvektoren entweder einem einfach Text- oder Binärformat aus.
Das Tool lässt zudem dem Nutzer offen, einige Parameter zu verändern. Jene, die in dieser Arbeit berücksichtigt wurden, sollen dabei näher erläutert werden:

- `-sample`
  Die Wahrscheinlichkeit, mit der hochfrequente Worte
- `-cbow`
  Bestimmt, welche Trainingsmethode verwendet wird ($0 \mathrel{\hat{=}}$ Skip-gram, $1 \mathrel{\hat{=}}$ Continuous-Bag-of-Words)
- `-negative`
  Anzahl von negativen Beispielen beim Training.

Zwar bietet das Tool auch noch andere Parameter, jedoch soll aufgrund mit der Empfehlungen in (Levy, Goldberg und Dagan), in der eine große Anzahl von Konfigurationen ausprobiert wurde, im Rahmen dieser Arbeit nur mit den oben genannten Werten experimentiert werden.

# Kapitel 5

# Evaluation der Wortvektoren

> *[Y]ou'll see that there are a lot of pairings where words with similar meanings are nearby. [...] On the other hand, there's a lot of junk. [...] You'd want the closest word to "grandma" to be "grandpa", not "gym."*
>
> BEN SCHMIDT ÜBER DAS PLOTTEN VON WORD EMBEDDINGS IN ZWEI DIMENSIONEN[1]

## 5.1 Evaluation der Wortvektoren

An dieser Stelle sollen die verschiedenen Ansätze zum Trainieren von Wortvektoren, die im vorherigen Kapitel vorgestellt werden, miteinander verglichen werden. Zu diesem Zweck sollte zuerst eine Frage gestellt werden: Was macht eine Menge von Wortvektoren "besser" bzw. "schlechter" als andere? Da der Vorteil von Wortvektoren darin besteht, semantische Informationen zu beinhalten, wird diese Frage meist dahingehend beantwortet, dass Vektoren dann als überlegen an zu sehen sind, wenn sie eine höhere semantische Ausdruckskraft besitzen. Um dies festzustellen, haben sich in Veröffentlichungen zu diesem Thema bestimmte Vorgehensweisen durchgesetzt, die in dne folgenden Abschnitten, vorgestellt, erläutert, angewendet und kritisch reflektiert werden sollen.

### 5.1.1 Qualitative Evaluation

Qualitative Verfahren zur Evaluation sind meist recht simple Ansätze, die für das menschliche Auge leicht zu interpetierbare Ergebnisse liefern. Deshalb sind sie für einen ersten Ausdruck auch durchaus geeignet, sollten wenn möglich aber nicht als alleinige Kriterium für eine Bewertung hinzugezogen werden, da sie meisten nie die Gesamtheit aller in den Ergebnissen enthaltenen Informationen darstellen können.
Im Beispiel der Wortvektoren werden beispielsweise einige Wörter des Vokabulars stellvertretend ausgewählt und zu diesen die $k$ nächsten Nachbarn im Vektorraum gesucht. Unter der Annahme, dass in Vektorräumen von Wortvektoren ähnliche Wörter nahe zusammenliegen, sollte diese Liste nah verwandte Begriffe zutage fördern (siehe Abbildung 5.1).

[1] Blogeintrag "Word Embeddings for the digital humanities" von Ben Schmidt (2015), online unter http://bookworm.benschmidt.org/posts/2015-10-25-Word-Embeddings.html (zuletzt abgerufen am 21.04.15)

| | WORT 1 | WORD 2 | WORT 3 | WORT 4 | WORT 5 |
|---|---|---|---|---|---|
| Datenset 1 | | | | | |
| Datenset 2 | | | | | |
| Datenset 3 | | | | | |

**Abbildung 5.1:** Listen der *k* nächsten Nachbarn von Wörtern in verschiedenen Datensets.

Das Problem bei dieser Methode liegt in der menschlichen Subjektivität: Die präsentierte Auswahl der Begriffe muss nicht zwangsläufig repräsentativ für die restlichen Daten sein und könnte theoretisch aus den wenigen, gut funktionierenden Beispielen bestehen. Darüber hinaus bleibt es in einigen Fällen schwierig, die Ergebisse verschiedener Datensets zu vergleichen, da sich die Qualität der *k* Nachbarn nicht quantifizieren lässt: Es lässt sich vielleicht erkennen, das diese in einem Fall wenig Sinn machen und im anderen Fall die Erwartungen erfüllen; an anderer Stelle scheinen die Resultate für den Betrachter jedoch nicht unbedingt schlechter, sondern einfach nur anders.
Darum ist wiederum festzuhalten, dass sich qualitative Methoden in diesem Fall eher für den ersten Eindruck eignen, weiterhin aber Prozeduren mit quantifizierbaren Ergebnissen verwendet werden sollten, wie z.B. nächsten Abschnitt beschrieben werden.

### 5.1.2 Quantitative Evaluation

Bei der quantitativen Evaluation von Wortvektoren werden die folgenden Ideen aufgegriffen:

1. **Benchmark-Tests**
   Bei dieser pragmatischen Art der Bewertung werden wird das Datenset als Grundlage für eine einfach Aufgabe wie Sentiment-Klassifikation oder Part-of-Speech-Tagging verwendet. Die Qualität der Daten wird dann anhand der Ergebnisse des Systems gemessen.
   Diese extrinische Evaluationsmethode macht ergo nur dann Sinn, wenn man mehr als ein Datenset miteinander vergleicht. Dabei muss sichergestellt werden, dass die Tests immer unter den selben Bedingungen ablaufen, damit eine Vergleichbarkeit gewährleistet bleibt.

2. **Wortähnlichkeit**
   Hierbei werden Wortpaaren Ähnlichkeitswerte von menschlichen Annotatoren zugeordnet. Anschließend werden mit den zu Verfügung stehenden Vektoren Ähnlichkeitswerte für die gleichen Paare berechnet, in der Regel mithilfe der Cosinus-Ähnlichkeit. Diese beschreibt die Ähnlichkeit zweier Vektoren als den Winkel zwischen ihnen, mit einem Wert von $-1$ ($\hat{=}$ komplett unterschiedlich) über 0 ($\hat{=}$ orthogonal) und $+1$ ($\hat{=}$ Equivalenz):

$$cosine\_similarity(\vec{a}, \vec{b}) = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\|\|\vec{b}\|} \tag{5.1}$$

Danach kann mit *Spearman's* $\rho$ bzw. *Spearman's rank correlation coefficient* anschließend festgestellt werden, ob die beiden Werte für die Wortpaare korrelieren, sprich ob das System Paaren, denen von Menschen ein hoher Ähnlichkeitswert zugewiesen wurde auch eine hohe Ähnlichkeit zuschreibt. Dabei ist $\rho \in [-1, 1]$ den Grad der Korrelation anzeigt, wobei $-1$ einer starken negativen, $+1$ einer starken positiven Korrelation entspricht.

3. **Analogien**
   Die dritte Methode basiert auf Analogien der Form *a verhält sich zu a* wie b zu b**. Die Daten werden nun dahingehend getestet, indem unter Gebrauch der Cosinus-Ähnlichkeit das $b*$ aus dem Vokabular $\mathcal{V}$ gesucht wird, welches besonders ähnlich zu $b$ und $a*$ aber unähnlich zu $a$ ist:

$$\underset{\tilde{b}^* \in \mathcal{V}}{argmax}\ cos(\tilde{b}^*, b - a + a^*) \tag{5.2}$$

   Sind alle Vektoren der Länge eins, so kann diese Gleichung umformuliert werden:

$$\underset{\tilde{b}^* \in \mathcal{V}}{argmax}\ cos(\tilde{b}^*, b) - cos(\tilde{b}^*, a) + cos(\tilde{b}^*, a^*) \tag{5.3}$$

   Diese Methode wird gemeinhin als 3COSADD bezeichnet. (Autor) etablierten dazu jedoch eine Alternative namens 3COSMUL, die bei Tests bessere Ergebnisse produziert:

$$\underset{\tilde{b}^* \in \mathcal{V}}{argmax} \frac{cos(\tilde{b}^*, b)\ cos(\tilde{b}^*, a^*)}{cos(\tilde{b}^*, a) + \epsilon} \tag{5.4}$$

   Dabei ist $\epsilon = 0,001$, um die Divison durch Null zu verhindern.
   Der Erfolg der Evaluation kann dann als Anteil der richtig vervollständigten Analogien (bei denen $\tilde{b}^* = b^*$) gemessen werden.

In dieser Arbeit sollen die Datensets durch die zweit- und drittgenannte Methode evaluiert werden. Ein weiterer Fallstrick liegt allerdings in der Zusammenstellung der Datensets: So liefern die genannten nur dann Aussagekräftige Ergebnisse, wenn bei der Wortähnlichkeit die menschlichenen Annotatoren zuverlässig und sinnvoll die Paare bewertet haben (zu messen z.B. mit $Cohen's\ \kappa$) und bei den Analogien aus der Zusammenstellung ebendieser (welche Entitäten sind enthalten, wie oft kommen diese vor, welche semantische Relationen wurden ausgewählt, wurden diese maschinell oder per Hand erzeugt).
Aus diesem Grund sollen die benutzten Evaluationssets im hierauf folgenden Abschnitt näher beleuchtet werden.

### 5.1.3 Evaluationsdaten

#### Wortpaarähnlichkeit

Im Englischen wird für die Wortähnlichkeitsevaluation häufig das WORDSIM353-Datenset verwendet. Dieses wurde unter dem Namen SCHM280 in deutsche portiert, wobei die Paare nicht nur einfach übersetzt, sondern die

Ähnlichkeit auch noch von deutschen Muttersprachlern neu bewertet wurde. Es enthält insgesamt 280 Wortpaare.

WORTPAARE65, WORTPAARE222 und WORTPAARE350 entstammen der Arbeit von [REFERENZ]. Dabei werden Wortpaaren Werte von 0 ($\hat{=}$ vollkommen unzusammenhängend) bis 4 ($\hat{=}$ stark zusammenhängend) bewertet. Die Anzahl der menschlichen Annotatoren sowie deren Übereinstimmung sind in Abbildung 5.2 festgehalten.

| Datenset | #Annotatoren | $\kappa$ |
|---|---|---|
| WORTPAARE65 | 24 | 0,81 |
| WORTPAARE222 | 21 | 0,49 |
| WORTPAARE350 | 8 | 0,69 |

**Abbildung 5.2:** Anzahl der Annotatoren und Agreement (als $Cohen's\ \kappa$) der WORTPAAR-Evaluationsdatensets.

#### Analogien

Die GOOGLE SEMANTIC/SYNTACTIC ANALOGY DATASETS wurden von Mikolov et al. (2013) [REFERENZ] eingefügt und bestehen aus Analogien der Form *a verhält sich zu a* wie b zu b**. [REFERENZ] haben diese manuell übersetzt und durch drei menschliche Prüfer validieren lassen. Dabei wurde die Kategorie “adjektiv - adverb” fallengelassen, da sie im Deutschen nicht existiert, weshalb 18.552 Analogien übrigbleiben. Diese werden im Folgenden einfach als GOOGLE bezeichnet.

SEMREL wurde aus Synonomie-, Antonomie- und Hypernomie-Beziehungen von [REFERENZ] für das Deutsche und Englische konstruiert. Dabei werden Substantive, Verben und Adjektive berücksichtigt. In der deutschen Variante sind 2.462 (recht schwierige) Analogien enthalten, die aus teilweise sehr seltenen Wörter kreiert wurden.

### 5.1.4 Evaluationsergebnisse

Im Folgenden sollen die Ergebnisse der Wortvektor-Evaluation diskutiert werden. Am Ende dieses Abschnitts werden dabei nur die Datensets mit den besten Ergebnissen und deren Konfiguration von Parametern vorgestellt, eine ausführliche Übersicht findet sich jedoch im Appendix B.

| DATENSET | WORTPAARE65 | WORTPAARE222 | WORTPAARE350 | SCHM280 |
|---|---|---|---|---|

| DATENSET | GOOGLE | SEMREL |
|---|---|---|

**Abbildung 5.3:** Evaluationsergebnisse der fünf besten Vektordatensetz. Oben: Spearman's $\rho$ für die Wortähnlichkeit mit von menschen gewerteten Wortpaaren. Unten: Treffer beim Vervollständigen von Analogien in Prozent. Nicht gefundene Wortvektoren klein in Klammern hinter dem Wert. Für eine vollständige Liste der Trainingsparameter jedes Datensets siehe A. Für die gesamte Liste aller Ergebnisse siehe B.

# Kapitel 6

# Experiment A: TransE für deutsche Wissensdaten

In diesem Kapitel soll der Ansatz von (Bordes et al., 2013) für deutsche Daten nachvollzogen werden. Dabei wird erst erklärt, wie die Daten erstellt wurden. Danach wird die Idee zum Training der Daten ausgeführt und auf die neuen Datensätze angewendet, bevor schlußendlich eine Evaluation und eine Gegenüberstellung zu den Originaldaten erfolgt.

## 6.1 Datenerzeugung

In (Bordes et al., 2013) werden mehrere Datensets erstellt darunter eines namens *FB15k*. Dieses besteht aus Relationstripeln der Form $(h, l, t)$ $(= (head, link, tail))$. Diese stammen aus *Freebase*, einer community-gepflegten Datenbank, in der mehr als 23 Millionen Entitäten durch Relationen miteinander verknüpft sind. Mittlerweile ist die Seite offline; das Projekt wurde sukzessive in *Wikidata* [1] integriert. Auch die Freebase API, die als Programmierschnittstelle zum Abfragen von Informationen dient wird langsam abgeschaltet [2].

Die FB15k-Daten enthalten 592.213 Tripeln mit 14.951 einzigartigen Entitäten und 1.345 einzigartigen Relationen. In Freebase sind Entitäten sprachlich unabhängig gehalten. So wird die Entität mit dem Kürzel "/m/02vk52z" im Englischen mit dem Begriff *World Bank* und im Deutschen mit *Weltbank* bezeichnet. Somit sind ist auch FB15k zumindest theoretisch vielsprachig. Jedoch sind die Entitäten darin oft hauptsächlich englische bzw. amerikanische Entitäten, die im deutschen Sprachraum teils nicht sehr bekannt sind. Das lässt daraus schließen, dass das Attribut einer Entität in Freebase, dass den deutschen Namen enthält, nicht immer verwendet wurde. Dies könnte drei Gründ haben:

1. Es gibt keine deutsche Übersetzung
2. Die Entität ist für den deutschen Sprachraum nicht relevant genug
3. Bisher hat einfach noch kein Nutzer einen deutschen Begriff hinzugefügt

[1] Siehe https://www.wikidata.org/wiki/Wikidata:Main_Page(zuletztabgerufenam20.05.16)

[2] Siehe https://en.wikipedia.org/wiki/Freebase(zuletztabgerufenam(20.05.16))

Die Plausibilität dieser Gründe ist diskutabel: Nicht alle Begriffe brauchen eine Übersetzung, so sind beispielsweise Personennamen i.d.R. durch alle Sprachen hinweg gleich. Schwieriger wird es bei Ortsnamen oder Namen von Organisationen: Intuitiv drängt sich der Anschein auf, dass populäre Bezeichnungen eher übersetzt werden als unpopulärere (*United States of America* → *Vereinigte Staaten von Amerika / University of Denver* → *University of Denver*).
Im Falle der Relevanz ist davon auszugehen, dass diese mit der Bearbeitung durch Nutzer einher geht: Bei einer großen Nutzerbasis ist davon auszugehen, dass diese primär Einträge von Entitäten bearbeiten, die im Kontext der Geschichte, des Tagesgeschehens o. Ä. relevant sind. Gegeben genug Zeit und aktive Nutzer ist also anzunehmen, dass eine immer größer werdende Prozentzahl von Relevanten Entitäten an das Deutsche angepasst wird. Bedenkt man die Laufzeit von Freebase seit 2007 (also zum Zeitpunkt des Schreibens dieser Arbeit rund 9 Jahre), so nehmen wir an, dass dieses Bedenken zwar nicht ganz aus der Welt zu räumen, aber zumindest zu vernachlässigen ist.

Basierend auf dieser Argumentation werden korrespondierende "deutsche" Daten folgendermaßen erzeugt: Es wird bei allen Relationstripeln eine Prüfung durchgeführt, ob beide Entitäten $h$ und $t$ eine deutsche Bezeichnung besitzen. Falls nicht, wird dieses Tripel ausgelassen. Danach wird ggf. noch die inverse Relation ergänzt (diese wird später beim Training benötigt), z.B. bei */location/location/contains* und */location/location/containedby*.

| DATENSET | #TRIPEL | #ENTITÄTSTYPEN | #RELATIONSTYPEN |
|---|---|---|---|
| FB15k | 592.213 | 14.951 | 1.345 |
| GER14k | 459.724 | 14.334 | 1.236 |

**Abbildung 6.1:** Daten über die Relationsdatensets FB15k und GER19k. Aufgelistet ist die Anzahl der Tripel (Datensätze), Entitäts- und Relationstypen.

Somit gilt für die Menge englischer Relationstripel $S_{EN}$ und die Menge deutscher Tripel $S_{DE}$, dass letztere eine echte Teilmenge ist: $S_{EN} \supsetneq S_{DE}$ [3]. Verschiedene Informationen über die beiden Datensets sind in Abbildung 6.1 aufgelistet.

## 6.2 Training

Gegeben ist eine Menge von Relationstripeln $S$ der Form $(h, l, t) \in S$. Zusätzliche gibt es noch eine Entitätsmenge $E$ sodass und $h, t \in E$ und eine Menge von Relationen $L$ mit $h \in L$. Den Entitäten und Relationen werden zuästzlich noch Vektoren aus $\mathbb{R}^k$ zugewiesen. Idealerweise gelten nach dem Training für eine gültige Relation $(h, l, t)$ dann $h + l \approx t$. Hinzu wird noch eine Menge aus korrumpierten bzw. "falschen" Tripeln $S'$ erstellt, indem für jedes Tripel in $S$ entweder $h$ oder $t$ durch eine andere, zufällig gewählte Entität ersetzt wird:

$$S' = \{(h', l, t) | h' \in E\} \cup \{(h, l, t') | t' \in E\} \tag{6.1}$$

[3] $\forall e \in S_{DE} : e \in S_{EN} \wedge S_{DE} \neq S_{EN}$

Um die für das Training benötigte Verlustfunktion du definieren wird das Unähnlichkeitsmaß $d_p$ für ein Tripel bestimmt, welcher entweder aus der $L_1$- oder der $L_2$-Norm abgeleitet wird, also:

$$d_1(h+l,t) = \sum_{i=1}^{k} \|h_i + l_i - t_i\| \tag{6.2}$$

$$d_2(h+l,t) = \sqrt{\sum_{i=1}^{k} \|h_i + l_i - t_i\|^2} \tag{6.3}$$

Nun kann die Verlustfunktion $\mathcal{L}$ erstellt werden:

$$\mathcal{L} = \sum_{(h,l,t)\in S} \sum_{(h',l',t')\in S'} max(0, \gamma + d_p(h+l,t) - d(h'+l,t')) \tag{6.4}$$

Der Parameter $\gamma$ bezeichnet hier einen Hyperparameter, der dem System einen gewissen Spielraum lässt. Indem versucht wird, den durch diese Funktion errechneten Verlust zu minimieren, wird beim Training sichergestellt, dass die Vektorrepräsentationen von korrekten Tripeln die Gleichung $h + l \approx t$ bestmöglichst erfüllen und für korrumpierte Tripel möglichst weit verfehlen.
Vor dem Training werden die Repräsentationen der Entitäten und Relationen mit einer uniformen Verteilung im Intervall $[-\frac{6}{\sqrt{k}}, \frac{6}{\sqrt{k}}]$ initialisiert und im Falle von $l \in L$ zusätzlich normiert. Während jedes Trainingsschritt werden für die aktuellen Repräsentation der Verlust berechnet und deren Werte mithilfe des Minibatch-Stochastic-Gradient-Descent angepasst. Der Minibatch enthält dabei ebenso viele gültige wie korrumpierte Tripel. Das Training wird solange ausgeführt, bis die Fehlerrate auf dem Validationsset konvergiert[4].

Für das Reproduzieren der Ergebnisse wurde den Empfehlen von (Bordes et al., 2013) gefolgt: So wird das Training nach maximal 1000 Epochen beendet, außerdem gilt $k = 50$, $\gamma = 1$, $\lambda = 0,01$ und $d_1$ als Unähnlichkeitsmaß.

## 6.3 Evaluation

Um die erstellten Daten zu evaluieren, werden bei den Relationen im Testset jeweils die Head- und Tailentitäten ersetzt. Danach werden alle bekannten Entitäten eingesetzt und mithilfe des Dissimilaritätsmaßes ansteigend sortiert. Der Rang der korrekten (ursprünglich entfernten) Entität wird dann gespeichert. Dieses Verfahren wurde zuerst von (Bordes et al., 2011) vorgeschlagen.
Daraus lassen sich dann zwei Evaluationsmaße ableiten: Den gemittelten Rang der korrekten Entität sowie die den Anteil der Evaluationsläufe, bei denen sich die richtige Entität innerhalb der Top 10 befand (Hits@10). Zusätzlich wurde noch zwischen roh und gefiltert unterschieden: Bei letzterem werden mögliche Tripel ignoriert, die in dieser Form schon in den Trainingsdaten vorkamen und so auf “unfaire” Art und Weise vor allen anderen

[4] Konvergenz auf dem Trainingsset könnte ein Zeichen für Overfitting sein.

Tripeln bevorzugt werden. Die Ergebnisse für die auf FB15k und GER14k trainierten Daten sind in Abbildung 6.2 festgehalten.

| Datenset | Gemittelter Rang | | Hits@10 | |
|---|---|---|---|---|
| | *Roh* | *Gefiltert* | *Roh* | *Gefiltert* |
| FB15k | 575,54 | **197,02** | **34,8** | **48,79** |
| GER14k | **234,73** | 222,05 | **34,07** | 41,83 |

**Abbildung 6.2:** Resultate für TransE, trainiert auf dem FB15k- und dem GER14k-Datenset. Hits@10 gibt den Anteil der Vorkommen an, bei denen sich die gesuchte Entität in der Relation links oder rechts in den top zehn von dem System vorhergesagten Kandidaten befand (*raw*). *Filtered* bezieht sich auf denselben Rang, wenn die extra erstellten korrumpierten Tripel zum Training aus den Trainingsdaten entfernt wurden. *mean* bezeichnet den gemittelten Rang der richtigen Entität in der vom System erstellten Rangliste der wahrscheinlichsten Vorhersagen.

An Ergebnissen ist zuerst zu erkennen, dass sie von den Ergebnissen in (Bordes et al., 2013) für FB15k etwas abweichen. Dies könnte daran liegen, das bei der Beschreibung des Trainingsverfahren angegeben wurde, dass das Trainingsverfahren nicht unbedingt 1000 Epochen lang ausgeführt wurde. In dem Code, der von den Autoren online zur Verfügung gestellt wurde [5], wurde dazu leider keine Option gefunden. Weshalb vermutlich davon auszugehen ist, dass dies manuell ausgeführt wurde. Deshalb ist zu schließen, dass die hier aufgeführten Ergebnisse schlechter sind als die, die im Paper angegeben wurde, weil die Vektorrepräsentationen zu sehr auf die Trainingsdaten angepasst wurden und dann schlecht auf den Testdaten generalisieren. Da die Daten aus GER14k gewissermaßen eine Untermenge von FB15k sind, sind deren Ergebnisse eher i.d.R. gleichwertig oder schlechter. Die Frage, warum der rohe gemittelte Rang bei GER14k signifikant besser ist als bei FB15k, könnte dahingehend beantwortet werden, dass sich in der Trainingsphase zu sehr an die Übungsdaten angepasst wurde und zuviele Trainingstripel vor der eigentlich richtigen Lösung rangieren. Deshalb ist der Wert für die gefilterten Tripel auch umso viel besser.

[STATISTISCHE SIGNIFIKANZ HITS@10 ROH]

## 6.4 Fazit

In diesem Kapitel wurde ein Verfahren vorgestellt, dass es erlaubt, Vektoren zu trainieren, die Entitäten und Relationen aus Wissendatenbanken modellieren und zur Relationsvorhersage zu nutzen. Dabei bestehen die Daten aus Relationstripeln, also 3-Tupeln mit einer Kopf- und einer Fußentität und einer dazugehörigen semantischen Relation. Es werden außerdem noch egativbeispiele erzeugt, indem einer der beiden Entitäten durch eine andere in den Trainingsdaten vorkommende Entität zufällig ausgetauscht wird. Der Trainingsvorgang wird hierbei mithilfe eines auf Vektorarithmetik basierenden Unähnlichkeitsmaß betrieben, mit der die “Stimmigkeit” korrekter Tripel versucht wird zu maximieren. Das Gegenteil stimmt für korrumpierte Tripel.

[5] https://github.com/glorotxa/SME

Es wurde gezeigt, dass sich die (Bordes et al., 2011) präsentierten Ergebnisse im Rahmen der gegebenen Möglichkeit annähernd replizieren lassen. Darüber hinaus wurde die Methodik auch auf eine Menge deutscher Daten angewendet, die eine echte Teilmenge der englischen Daten darstellt. Für diese konnten ähnlich gute Ergebnisse erziehlt werden, wobei die qua definition kleinere Menge an Übungsbeispielen dem Unterschied in Performanz Rechnung trägt.

Bei dieser Vorgehensweise wurden die Vektorrepräsentationen für Entitäten und Relationen gleichermaßen trainiert. In den folgenden Kapiteln soll ein Versuch unternommen werden, mithilfe von distributioneller Semantik erstellte Wortvektorrepräsentationen für die Entitäten zu nutzen und darauf aufbauend Relationen vorherzusagen.

# Kapitel 7

# Experiment B: Relationsvorhersage mit Wortvektorrepräsentationen

*Q: Why did the multithreaded chicken cross the road? A: to To other side . get the*

JASON WHITTINGTON

## 7.1 Idee

Gegeben ist ein Vektorraum $V$ mit Wortvektoren $\vec{u}, \vec{v} \in V$ und $\vec{u}, \vec{v} \in \mathbb{R}^d$. Gesucht ist eine Funktion $\phi$, die ein Vektorenpaar in einen Relationsraum abbildet: $\phi : \mathbb{R}^d \to \mathbb{R}^e$, wobei nicht zwangsläufig $d = e$.
In ihrer einfachsten Form bildet sie einfach die Differenz $\vec{d}$ der beiden Vektoren:

$$\phi(\vec{u}, \vec{v}) = \vec{v} - \vec{u} = \vec{d} \tag{7.1}$$

## 7.2 Algorithmus

Der Grundalgorithmus (siehe Abbildung 7.1) versucht nun, alle Kombinationen von Wortvektoren zu bilden und über diese und den Differenzvektor zu bilden. Alle Kombinationen würde bei $n$ Vektoren in $n * n = n^2$ Vektoren resultieren. Zwar ist $\phi(\vec{u}, \vec{v}) \neq \phi(\vec{v}, \vec{u})$, jedoch wäre die Berechnung beider Differenzvektoren redundant, da sie lediglich Spiegelungen voneinander im Raum sind und so diesselbe Information enthalten. Die Berechnung der Differenz in nur eine Richtung reduziert die Anzahl der Vektoren dadurch zu $\frac{n*(n-1)}{2}$.

**Data** : Menge von Vektorpaaren $\mathcal{C}$
**for** $(\vec{u}, \vec{v}) \in \mathcal{C}$ **do**
  $\vec{d} = \vec{v} - \vec{u}$;
**end**

**Abbildung 7.1:** Einfacher Projektionsalgorithmus.

Gegeben einer Menge relevanter Kombinationen $\mathcal{C}$ mit Vektorpaaren gilt also:

$$\forall(\vec{u}, \vec{v}) \in \mathcal{C} : (\vec{v}, \vec{u}) \notin \mathcal{C} \tag{7.2}$$

Eine Modifikation des Algorithmus besteht daran, das Berechnen von $\vec{d}$ ab eine Bedingung zu knüpfen. Gegeben sei eine Menge von Sätzen (ein Korpus) $\mathcal{K}$ mit $n$ Sätzen $s_i$ sodass $\mathcal{K} = \{s_i\}_{i=1}^{n}$ mit $m$ Wörtern $w_{ij}$ pro Satz $s_i = \{w_{ij}\}_{j=1}^{m}$. Eine Kookkurrenz von zwei Begriffen (*types*) $t_1$ und $t_2$ besteht demnach, falls im Korpus mindestens ein Satz existiert, in dem beide gleichermaßen vorkommen. Dazu können wir eine Funktion $\Lambda$ definieren, die die Anzahl der Kookkurrenzen bestimmt:

$$\Lambda(t_1, t_2) = |\{s_i | \exists w_{ij} = t_1 \wedge \exists w_{ij} = t_2 \wedge t_1 \neq t_2\}_{i=1}^{n}| \tag{7.3}$$

Eine mögliche Einschränkung besteht darin, in einem geänderten Algorithmus (siehe Abbildung 7.2) das Berechnen von $\vec{d}$ nur dann zu erlauben, wenn die Anzahl der Kookkurrenzen der zu den Vektoren $\vec{v}(t_1), \vec{v}(t_2)$ gehörenden Begriffe $t_1, t_2$ einen bestimmten Schwellenwert $\gamma$ überschreitet, also $\Lambda(t_1, t_2) > \gamma$.

**Data** : Menge von Vektorpaaren $\mathcal{C}$
**for** $(\vec{v}(t_1), \vec{v}(t_2)) \in \mathcal{C}$ **do**
  **if** $\Lambda(t_1, t_2) > \gamma$ **then**
    $\vec{d} = \vec{v}(t_2) - \vec{v}(t_1)$;
  **end**
**end**

**Abbildung 7.2:** Modifizierter Projektionsalgorithmus, bei dem die zu den Vektoren gehörigen Begriffe über $\gamma$ Mal im Korpus im gleichen Satz aufgetreten sein müssen, damit $\vec{d}$ errechnet wird.

## 7.3 Parallelisierter Algorithmus

Doch selbst mit den erwähnten Einschränkungen skaliert dieser Algorithmus nur bedingt. Um dem entgegenzuwirken, soll nun versucht werden, diesen mithilfe von *Multithreading* zu parallelisieren. Bei dieser Technik beschäftigt ein Rechenprozess mehrere Prozessstränge (*Threads*), die miteinander kommunizieren und bei Bedarf synchronisiert werden können, während sie Teile eines Problems lösen.
Ein beliebtes Muster, das Verhältis zwischen mehreren Threads zu definieren besteht im *Master-Slave*-Muster. Dabei fungiert ein Subprozess als Aufseher, der seine Arbeiterprozesse überwacht, sie mit Daten versorgt und in manchen Fällen untereinander koordiniert.

In diesem konkreten Fall ist der Master-Thread dafür verantwortlich, alle benötigten Daten in entsprechende Datenstrukturen zu laden, sie den Worker-Threads bereitzustellen und letztere gemeinsam zu starten und zu beenden, sobald ein bestimmtes Abschlusskriterium der Aufgabe erfüllt ist. Zusätzlich wird eine Menge eingeführt, in die Paare von Vektoren hinzugefügt werden, sofern für sie von einem Thread ein Differenzvektor ausgerechnet wurde (siehe Abbildung 7.3). Damit nun keiner der Threads redundante

**Data** : Menge von Vektorpaaren $\mathcal{C}$
**Data** : Menge von erledigten Vektorpaaren $\mathcal{C}'$ (Anfangs $\mathcal{C}' = \varnothing$)
**Data** : Menge von Threads $\mathcal{T} = \{th\}_i^n$

**Klasse** MASTERTHREAD
data = ladeDaten();
**for** $th \in \mathcal{T}$ **do**
  starteThread($th$, data);
**end**
**for** $th \in \mathcal{T}$ **do**
  beendeThread($th$);
**end**

**Klasse** WORKERTHREAD
**for** $(\vec{v}(t_1), \vec{v}(t_2)) \in \mathcal{C}$ **do**
  **if** $(\vec{v}(t_1), \vec{v}(t_2)) \notin \mathcal{C}'$ **then**
    **if** $\Lambda(t_1, t_2) > \gamma$ **then**
      $\vec{d} = \vec{v}(t_2) - \vec{v}(t_1)$;
    **end**
    $\mathcal{C}' = \mathcal{C}' \cup (\vec{v}(t_1), \vec{v}(t_2))$;
  **end**
**end**

**Abbildung 7.3:** Parallelisierter Projektionsalgorithmus, bei dem die zu den Vektoren gehörigen Begriffe über $\gamma$ Mal im Korpus im gleichen Satz aufgetreten sein müssen, damit $\vec{d}$ errechnet wird. Ein Master-Thread verteilt zudem die Aufgaben an Worker-Threads, die diese Berechnungen übernehmen und erledigt Vektorpaare in einer Menge ablegen.

Berechnungen durchführt, prüft er, ob sein aktuelles Vektorpaar sich in dieser Menge befindet und schon abgehakt wurde[1].

Als Grundlage der Daten wurde das Datenset Mark 0 [BESTES EINFÜGEN] gewählt, da es in den meisten der Evaluationsaufgaben als bestes Abschnitt. Mit $\gamma = 100$ resultierte das Procedere in einem neuen Menge an Daten mit insgesamt X Differenzvektoren [GENAUE ZAHL EINFÜGEN] mit einer Größe von X,X GB [GENAUE GRÖSSE EINFÜGEN].
Um den Gewinn durch Multithreading zu verdeutlichen, wurden darüber hinaus die Rechenzeiten gemessen (siehe Abbildung 7.4). Gerechnet wurde auf eine Hardware mit 40 Prozessorkernen und [RESTLICHE SPECS EINFÜGEN].

## 7.4 Clustering

*Clustering* bezeichnet eine Art von unüberwachten maschinellen Lernen, die versucht, Elemente, die nach vorher definierten Maßgaben als ähnlich erachtet werden sollen, zu gruppieren.
Die Literatur zu diesem Thema bietet dabei eine Fülle von Algorithmen mit

[1] Damit dies möglichst schnell funktioniert, wird das Vektorpaar durch eine Hashfunktion in einen Wert umgewandelt. Diese wird so gewählt, sodass $h(\vec{v}(t_1), \vec{v}(t_2)) = h(\vec{v}(t_2), \vec{v}(t_1))$.

| Anzahl der Threads | Benötigte Zeit (in $m$) | Beschleunigung (in %) |
|---|---|---|
| 1 | | - |
| 2 | | |
| 8 | | |
| 32 | | |
| 64 | | |
| 128 | | |

**Abbildung 7.4:** Rechenzeiten für den Mappingschritt nach Anzahl von Threads auf einem System mit 40 Prozessorkernen und [RESTLICHE SPECS EINFÜGEN]. Die Beschleunigung wird im Vergleich zum Ausführen des Programms mit nur einem Thread gemessen.

verschiedenen Grundannahmen, aus denen es zu wählen gilt. Für die Aufgaben in dieser Arbeit wurde der DBSCAN-Algorithmus (**D**ensity-**B**ased **S**patial **C**lustering of **A**pplications with **N**oise) gewählt, da er folgende Kriterien erfüllt:

- DBSCAN erlaubt es, dass nicht alle Datenpunkte bei der Terminierung des Algorithmus einem Cluster zugeordnet sein müssen (*Outlier detection*)
- Die Anzahl der Cluster muss bei Beginn des Algorithmus nicht festgelegt werden.
- Cluster werden nicht nach der räumlichen Distanz zwischen den Punkten gebildet, sondern nach deren Dichte im Raum. Dies lässt auch nicht-sphärische Clusterformen zu.

Zur Erklärung von DBSCAN müssen zuerst einige Definitionen erstellt werden (siehe Abbildung 7.5 zur visuellen Darstellung):

- Ein Punkt $p$ in den Daten wird dann als Kernpunkt (*core point*) bezeichnet, sofern mindesten $minPts$ Punkte innerhalb einem Radius $\epsilon$ vorhanden sind.
  Diese Punkte sind von $p$ *direkt erreichbar*.
- Ein Punkt $q$ ist von $p$ erreichbar, sofern es einen Pfad $p_1, \ldots, p_n$ mit $p_1 = p$ und $p_n = q$ gibt, wobei jeder Punkt $p_{i+1}$ von einem Punkt $p_i$ direkt erreichbar ist.
- Alle Punkte, die nicht von einem anderen Punkt aus direkt erreichbar sind, sind Ausreißer (*Outlier*).
- Zwei Punkte $p$ und $q$ sind *direkt verbunden*, sofern es einen Punkt $o$ gibt, von dem aus beide Punkte direkt erreichbar sind.
- Cluster bestehen aus Punkten, die alle miteinander direkt verbunden sind.

[2]Abbildung von Chire - Eigenes Werk, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=17045963 (zuletzt abgerufen am 25.04.16).

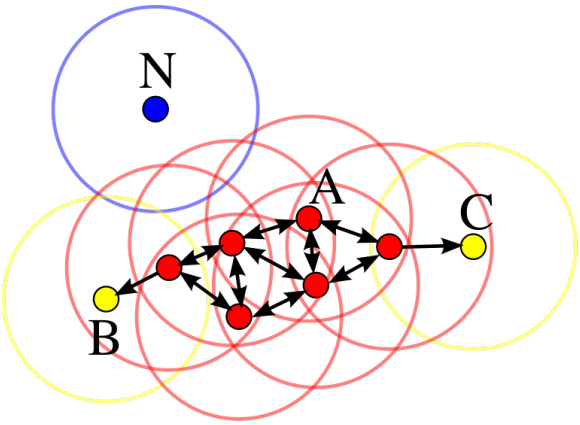



**Abbildung 7.5:** Darstellung der Funktionsweise von DBSCAN. Punkt A ist ein Kernpunkt, die Punkte B und C sind von A aus erreichbar. Punkt N ist nicht erreichbar und damit ein Ausreißer.[2]

Der Algorithmus ist von linearer Komplexität, sofern ihm eine geeignete Indexierung der Daten zugrunde gelegt wird, ansonsten verhält sich die Komplexität quadratische zur Anzahl der Datenpunkte[3].

## 7.5 Ergebnisse

## 7.6 Zwischendiskussion

Wie der letzte Abschnitt der gezeigt hat, entsprechen die Ergebnisse nicht den vorher gefassten Hoffnungen. Im Folgenden soll ein Versuch unternommen zu erklären, welche Probleme zu diesen Resultaten geführt haben und welche Implikationen diese besitzen.

### 7.6.1 Daten

Will man das Scheitern unmittelbar auf einen Faktor zurückführen, so liegen die zugrunde liegenden Daten am nächsten. Es lassen sich im Bezug auf jene folgende Punkte feststellen:

- **Menge**
  Selbst mit dir Reduzierung der resultierenden Daten von $n^2$ auf $\frac{n*(n-1)}{2}$ ist die Datenmenge noch sehr groß, gerade wenn das anfänglich Wortvektorset auf einem großen Vokabular wie dem des Decow-Korpus aufbaut. Dies stellt hohe Anforderung an die Skalierbarkeit aller involvierter Algorithmen und fordert Einschränkungen auf verschiedener Ebene, wodurch mögliche spätere Entdeckung eventuell vorenthalten werden könnten.

- **Rauschen**
  In den Enddaten ist der überwiegende Teil der Datenpunkte keiner sinnvollen Relation zuzuordnen. Dadurch verrauschen diese mögliche sinnvolle Relationscluster, die darin untergehen (vgl. Figure 7.6).

[3] Bei der `scikit-learn`-Implementation wird beispielsweise ein Nearest-Neighbour-Graph verwendet: http://scikit-learn.org/stable/modules/generated/sklearn.cluster.dbscan.html (zuletzt abgerufen am 25.04.16)

Die resultierenden Cluster sind sehr schwammig, da sie sich schlecht vom Hintergrundrauschen abgrenzen. Dies ist beispielsweise durch den Wert des Silhouettenkoeffizienten [4] erkennbar, der bei den Experimenten immer kurz unter Null lag[5].

- **Ähnliche Distanzvektoren**
  Der beschriebene Ansatz wurde unter der Annahme verfolgt, dass für Wortpaare einer Relation $R = \{(h_j, t_j\}_{j=0}^{m}$ folgendes gilt:

$$\vec{v}(t_0) - \vec{v}(h_0) \approx \vec{v}(t_1) - \vec{v}(h_1) \approx \ldots \approx \vec{v}(t_m) - \vec{v}(h_m) \tag{7.4}$$

  Anders ausgedrückt wurde aufgrund vorheriger Arbeiten die Hypothese aufgestellt, dass sich die Distanzvektoren von Wortpaaren derselben Relation ähneln. Durch die Ergebnisse wurde diese Hypothese nicht widerlegt (im Gegenteil scheinen andere Arbeiten wie (Bordes et al., 2013) oder (Lin et al., 2015) diese Annahmen zu bestätigen), jedoch gilt diese Eigenschaft nicht **exklusiv** für diese Untermenge an Vektorenpaaren.
  Nehmen wir beispielsweise im ursprünglichen Vektorraum mit Wortvektoren das folgende Szenario an: Wir finden dort zwei Cluster $C_1$ und $C_2$ vor. Die zu den Vektoren zugehörigen Cluster weisen eine semantische Ähnlichkeit auf und liegen deshalb nahe beieinander, z.B. Vornamen wie *Julia* und *Hans* in $C_1$ und Städte wie *Barcelona* und *Paris* in $C_2$. Daraus ergibt sich Folgendes:

$$\begin{aligned} \vec{v}(Julia) \approx \vec{v}(Hans) \wedge \vec{v}(Barcelona) \approx \vec{v}(Paris) \implies \\ \vec{v}(Julia) - \vec{v}(Barcelona) \approx \vec{v}(Hans) - \vec{v}(Paris) \end{aligned} \tag{7.5}$$

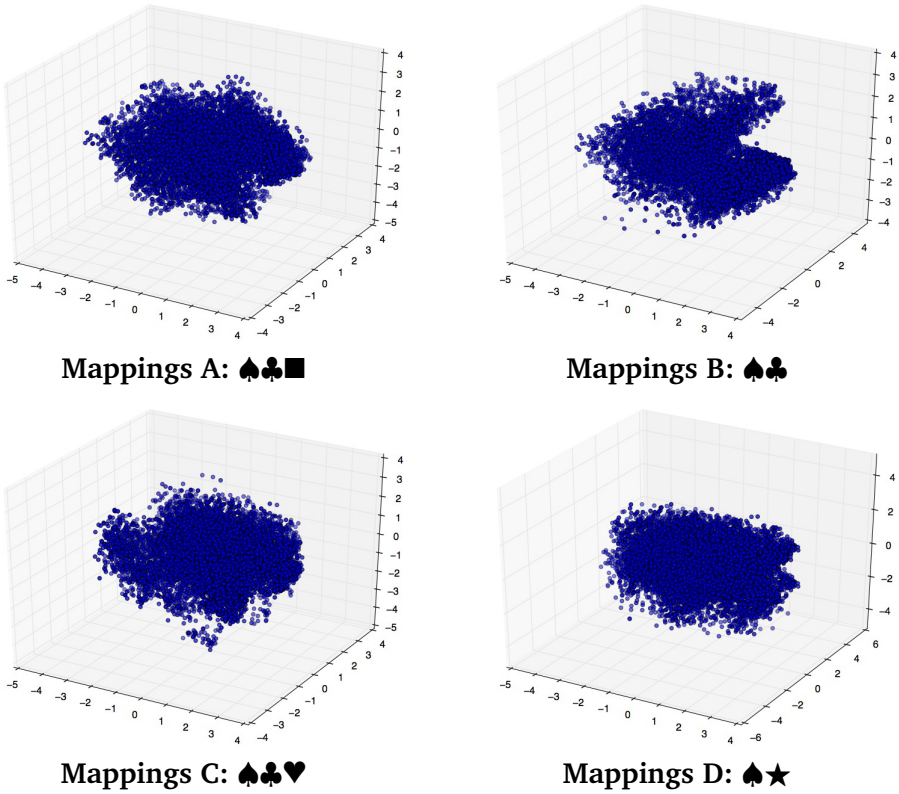
  Anders ausgedrückt: Auch wenn sich semantische Information in den Wortvektoren dadurch manifestiert, dass ähnliche Ausdrücke in ihren Clustern nahe beinander liegen, können Distanzvektoren aus Wortenpaaren der gleichen Cluster ähnlich sein, ohne semantisch in irgendeinem Zusammenhang zu stehen. Dies führt schlussendlich dazu, dass die wenigen Cluster, die im Relationsraum gefunden werden, zwar aus Wortpaaren mit ähnlichem Differenzvektor stehen, jedoch keine "sinnvolle" Relation bilden, was sicht mit dem in 6.1 eingeführtern Parameter $\gamma$ zwar verringern, allerdings nicht gänzlich verhindern lässt. Das Ziel, neue und legitime Relationen zu finden, wird durch diesen Trugschluss ad absurdum geführt.

### 7.6.2 Ansatz

Das Fehlschlagen des Vorgehens kann auch auf einer theoretischen Ebene festgestellt werden: Das Ziel bestand darin, Wissen über Relationen zwischen Wortpaaren zu extrahieren. Um dieses Wissen gewissermaßen "freilegen", muss es aber auf eine Art und Weise innerhalb der Daten "kodiert" sein, wenn auch versteckt (so wie die semantische Ähnlichkeit zwischen Wörtern durch die räumliche Nähe ihrer Vektoren und deren Dimension kodiert ist).

[4] Der Silhoettekoeffizient $s_C$ beschreibt das durchschnittliche Verhältnis vom Abstand eines Punktes zu seinem Clusterzentrum gegnüber des nächsten Clusterzentrums. $-1 \leq s_C \leq 1$.

[5] Für ein gutes Clustering wird ein Wert von $s_C \geq 0,75$ erwartet.



**Mappings A: ♠♣■** **Mappings B: ♠♣**

**Mappings C: ♠♣♥** **Mappings D: ♠★**

**Feature-Legende**

- ♠ $\vec{v}(t_2) - \vec{v}(t_1)$
- ★ $\Lambda(t_1, t_2) \geq 1$
- ♣ $\Lambda(t_1, t_2) \geq 100$
- ■ $cos(\vec{v}(t_2) - \vec{v}(t_1))$
- ♥ $\parallel \vec{v}(t_2) - \vec{v}(t_1) \parallel$

**Abbildung 7.6:** Dreidimensionale Projektionen einiger durch das Mappingverfahren resultierender Vektorräume. Jeweils dargestellt: 10000 Vektoren. Symbole zeigen die jeweils genutzten Feature an.

Wie beim Aufzeigen des Trugschlusses am Ende des vorherigen Abschnitts gezeigt wurde, sind semantische Relationen nicht eindeutig innerhalb der Daten aufzuzeigen bzw. nur dann, wenn bereits vorher bruchstückhaftes Wissen darüber vorliegt. Ohne dieses Vorwissen sind richtige von nur scheinbaren Relationen nicht zu trennen.

# Kapitel 8

# Fazit

*Mit dem Wissen wächst der Zweifel.*

JOHANN WOLFGANG VON GOETHE

## 8.1 Zusammenfassung

In dieser Arbeit wurden verschiedene Möglichkeiten präsentiert, Wissen aus Wissensdatenbanken durch kontinuierliche Vektoren zu repräsentieren. In Kapitel 6 wurden unterschiedlich komplexe Herangehesweisen vorgestellt, Repräsentationen für Entitäten und Relationen einer solchen Datenbank gleichzeitig zu mithilfe von Methoden des Maschinellen Lernens zu trainieren.
Für das TransE genannte Verfahren wurden zusätzlich Ergebnisse mit einer deutschsprachigen Untermenge der Daten (GER14k) durchgeführt, die zwar bei der Evaluation etwas schlechter ausfielen, aber dennoch vergleichbar waren.

In Kapitel 7 wurde ein eigener Versuch unternommen, ähnliche Ergebnisse mithilfe von Wortvektorrepräsentationen zu erreichen, zur deren Erstellung das Tool *word2vec* verwendet wurde, welches auf großen Korpora arbeitet. Dazu wurde der deutschsprachige Internetkorpus DECOW14X aufbereitet.
Im Folgenden wurde angestrebt, Wortvektorpaare durch ihre Differenzvektoren verschiedenen Relationen zuzuteilen. Dies schlug jedoch fehl, da der Ansatz auf einem Trugschluss fußte. Deshalb konnten keine evaluierbaren Ergebnisse erzeugt werden. Eine ausführliche Diskussion darüber findet sich im letzten Abschnitt von Kapitel 7.

## 8.2 Diskussion

Die Implikationen der in dieser Arbeit gewonnenen Erkenntnisse sollte mit Bedacht diskutiert werden. Das Fehlschlagen eines Ansatzes auf der Basis von Wortvektorrepräsentationen heißt im Umkehrschluss nicht, dass alle Vorstöße in dieser Richtung zum Scheitern verurteilt sind (im Gegenteil soll darauf im nächsten Abschnitt eingegangen werden).
Es scheint jedoch leichter, Repräsentationen gleich im Kontext der Struktur einer Wissensdatenbank als im Kontext eines großen Korpus zu trainieren, da diese so von Anfang an auf dafür wichtige Eigenschaften getrimmt werden (nämliche die Beziehung zu anderen Relevanten Entitäten).

Das Fehlschlagen des vorgestellten Ansatzes in Kapitel 7 scheint vor allem auf einen Fehler in der Grundannahme und die schlechte Skalierbarkeit zurückzuführen. Um Letzteres zu verhindern, könnten in zukünftigen Ansätzen weitere Informationsquellen hinzugezogen werden, um “sinnvolle” von “sinnlosen” Wortpaaren zu trennen. Da davon auszugehen ist, dass in der Menge aller Möglichen Wortpaare nur ein sehr kleiner Prozentsatz tatsächlich Sinn ergeben dürfte, sollte dieses Vorgehen die totale Rechenzeit signifikaten verringern.
Der der Hypothese zugrunde liegende logische Fehler scheint im Nachhinein offensichtlich. Jedoch war dieser schwer vorherzusehen, nachdem in vielen anderen Arbeiten die semantischen Eigenschaften von arithmentischen Rechnungen mit Wortvektoren so stark hervorgehoben werden. Zudem wird dieser Aspekt immer nur anhand von in einer offensichtlichen Beziehung zueinander stehenden Wortpaaren illustriert. Dadurch fiel dieses Versäumnis erst auf, als es aus zeitliche Gründen nicht mehr möglich war, im Rahmen dieser Abschlussarbeit andere Verfahren zu testen und zudem Versuche, auf Basis der vorliegenden Daten diesen Fehler zu korrigieren fehlgeschlagen waren.

## 8.3 Ausblick

# Anhang A

# Übersicht über die Parameter zum Trainieren der Wortvektoren

| NAME | KORPUS | PREP | TRAINING | NEG | SAMPLING |
|---|---|---|---|---|---|
| *Mark I* | Decow | - | Skip-gram | 5 | $1^-5$ |
| *Mark II* | Decow | - | Skip-gram | 5 | $1^-4$ |
| *Mark III* | Decow | - | Skip-gram | 5 | $1^-3$ |
| *Mark IV* | Decow | - | Skip-gram | 5 | $0,01$ |
| *Mark V* | Decow | - | Skip-gram | 5 | $0,1$ |
| *Mark VI* | Decow | - | Skip-gram | 5 | $1$ |
| *Mark VII* | Decow | - | CBOW | 5 | $1^-5$ |
| *Mark VIII* | Decow | - | CBOW | 5 | $1^-4$ |
| *Mark IX* | Decow | - | CBOW | 5 | $1^-3$ |
| *Mark X* | Decow | - | CBOW | 5 | $0,01$ |
| *Mark XI* | Decow | - | CBOW | 5 | $0,1$ |
| *Mark XII* | Decow | - | CBOW | 5 | $1$ |
| *Mark XIII* | Decow | Lemmatisiert | Skip-gram | 5 | $1^-5$ |
| *Mark XIV* | Decow | Lemmatisiert | Skip-gram | 5 | $1^-4$ |
| *Mark XV* | Decow | Lemmatisiert | Skip-gram | 5 | $1^-3$ |
| *Mark XVI* | Decow | Lemmatisiert | Skip-gram | 5 | $0,01$ |
| *Mark XVII* | Decow | Lemmatisiert | Skip-gram | 5 | $0,1$ |
| *Mark XVIII* | Decow | Lemmatisiert | Skip-gram | 5 | $1$ |
| *Mark XIX* | Decow | Lemmatisiert | CBOW | 5 | $1^-5$ |
| *Mark XX* | Decow | Lemmatisiert | CBOW | 5 | $1^-4$ |
| *Mark XXI* | Decow | Lemmatisiert | CBOW | 5 | $1^-3$ |
| *Mark XXII* | Decow | Lemmatisiert | CBOW | 5 | $0,01$ |
| *Mark XXIII* | Decow | Lemmatisiert | CBOW | 5 | $0,1$ |
| *Mark XXIV* | Decow | Lemmatisiert | CBOW | 5 | $1$ |

**Abbildung A.1:** Quelle und Trainingsparameter für verschiedenen Sets von Wortvektoren. PREP = Ggf. Aufbereitung des Korpus vor dem Training; TRAINING = Verwendete Trainingsmethode; NEG = Anzahl der Negativbeispiele beim Training; SAMPLING = Ausmaß des Downsamplings häufiger Wörter.

# Anhang B

# Evaluationen der Wortvektoren

Evaluationsergebnisse bei Wortähnlichkeit und Analogien für die verschiedenen Datensets. Für weitere Informationen über die Grundlage der Vektoren siehe Appendix A. Wörter außerhalb des Vokabulars wurden entweder als Fehler gerechnet, oder werden, falls anders nicht möglich, als Zahl im Index in runden Klammern angegeben.

| | Wortähnlichkeit ($\rho \in [-1, 1]$) | | | | Analogien (in %) | |
|---|---|---|---|---|---|---|
| **Datenset** | WORTPAARE65 | WORTPAARE222 | WORTPAARE350 | SCHM280 | GOOGLE | SEMREL |
| Mark I | **-0,8096** | **-0,4640**$_{(13)}$ | **-0,7302**$_{(13)}$ | **-0,7094**$_{(2)}$ | **44,56** | 1,71 |
| Mark II | -0,7856 | -0.4409$_{(13)}$ | -0,7156$_{(13)}$ | -0,6928$_{(2)}$ | 40,37 | 1,83 |
| Mark III | -0,7718 | -0,4242$_{(13)}$ | -0,6886$_{(13)}$ | -0,6778$_{(2)}$ | 27,42 | 1.87 |
| Mark IV | -0,7681 | -0,3902$_{(13)}$ | -0,6834$_{(13)}$ | -0,6545$_{(2)}$ | 25,48 | 1,83 |
| Mark V | -0,7725 | -0,3889$_{(13)}$ | -0,6738$_{(13)}$ | -0,6529$_{(2)}$ | 25,54 | 1,67 |
| Mark VI | -0,7697 | -0,3907$_{(13)}$ | -0,6784$_{(13)}$ | -0,6541$_{(2)}$ | 25,52 | 1,46 |
| Mark VII | -0,7255 | -0,4198$_{(13)}$ | -0,6924$_{(13)}$ | -0,6361$_{(2)}$ | 30,60 | 1,50 |
| Mark VIII | -0,6149 | -0,4321$_{(13)}$ | -0,6530$_{(13)}$ | -0,5983$_{(2)}$ | 26,98 | 1,83 |
| Mark IX | -0,6784 | -0,4288$_{(13)}$ | -0,6090$_{(13)}$ | -0,5553$_{(2)}$ | 22,26 | 1,62 |
| Mark X | -0,5949 | -0,4185$_{(13)}$ | -0,5791$_{(13)}$ | -0,5003$_{(2)}$ | 19,22 | 1,22 |
| Mark XI | -0,5890 | -0,4245$_{(13)}$ | -0,5700$_{(13)}$ | -0,4993$_{(2)}$ | 18,60 | 1,38 |
| Mark XII | -0,5874 | -0,4251$_{(13)}$ | -0,5706$_{(13)}$ | -0,5021$_{(2)}$ | 8,62 | 1,50 |
| Mark XIII | **-0,8247**$_{(1)}$ | **-0,5066**$_{(102)}$ | **-0,7494**$_{(132)}$ | -0,7097$_{(48)}$ | 15,51 | **3,01** |
| Mark XIV | -0,8106$_{(1)}$ | -0,4953$_{(102)}$ | -0,7362$_{(132)}$ | **-0,7205**$_{(48)}$ | 14,51 | 2,56 |
| Mark XV | -0,7786$_{(1)}$ | -0,4991$_{(102)}$ | -0,7136$_{(132)}$ | -0,6956$_{(48)}$ | 13,15 | 2,44 |
| Mark XVI | -0,7576$_{(1)}$ | -0,4991$_{(102)}$ | -0,6990$_{(132)}$ | -0,6702$_{(48)}$ | 11,76 | 2,56 |
| Mark XVII | -0,7578$_{(1)}$ | -0,5074$_{(102)}$ | -0,6986$_{(132)}$ | -0,6638$_{(48)}$ | 11,41 | 2,80 |
| Mark XVIII | -0,7551$_{(1)}$ | -0,5102$_{(102)}$ | -0,6981$_{(132)}$ | -0,6715$_{(48)}$ | 11,38 | **3,01** |
| Mark XIX | -0,7589$_{(1)}$ | -0,4899$_{(102)}$ | -0,7476$_{(132)}$ | -0,6849$_{(48)}$ | 14,41 | 2,23 |
| Mark XX | -0,7519$_{(1)}$ | -0,5144$_{(102)}$ | -0,7239$_{(132)}$ | -0,6731$_{(48)}$ | 12,82 | 2,15 |
| Mark XXI | -0,7188$_{(1)}$ | -0,5371$_{(102)}$ | -0,6875$_{(132)}$ | -0,6520$_{(48)}$ | 9,68 | 1,75 |
| Mark XXII | -0,6926$_{(1)}$ | **-0,5377**$_{(102)}$ | -0,6429$_{(132)}$ | -0,6138$_{(48)}$ | 6,85 | 1,71 |
| Mark XXIII | -0,6851$_{(1)}$ | -0,5373$_{(102)}$ | -0,6359$_{(132)}$ | -0,5882$_{(48)}$ | 6,11 | 1,95 |
| Mark XXIV | -0,6913$_{(1)}$ | -0,5349$_{(102)}$ | -0,6369$_{(132)}$ | -0,5930$_{(48)}$ | 6,15 | 2,07 |

# Literatur


Baroni, Marco, Georgiana Dinu und Germán Kruszewski (2014). „Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors." In: *ACL (1)*, S. 238–247.

Bengio, Yoshua et al. (2006). „Neural probabilistic language models". In: *Innovations in Machine Learning*. Springer, S. 137–186.

Bordes, Antoine et al. (2011). „Learning structured embeddings of knowledge bases". In: *Conference on Artificial Intelligence*. EPFL-CONF-192344.

Bordes, Antoine et al. (2013). „Translating embeddings for modeling multi-relational data". In: *Advances in Neural Information Processing Systems*, S. 2787–2795.

Collobert, Ronan et al. (2011). „Natural language processing (almost) from scratch". In: *The Journal of Machine Learning Research* 12, S. 2493–2537.

Goldberg, Yoav (2015). „A Primer on Neural Network Models for Natural Language Processing". In: *CoRR* abs/1510.00726. URL: http://arxiv.org/abs/1510.00726.

Harris, Zellig S (1954). „Distributional structure". In: *Word* 10.2-3, S. 146–162.

Levy, Omer und Yoav Goldberg (2014). „Dependency-Based Word Embeddings." In: *ACL (2)*, S. 302–308.

Levy, Omer, Yoav Goldberg und Ido Dagan (2015). „Improving distributional similarity with lessons learned from word embeddings". In: *Transactions of the Association for Computational Linguistics* 3, S. 211–225.

Lin, Yankai et al. (2015). „Learning Entity and Relation Embeddings for Knowledge Graph Completion." In: *AAAI*, S. 2181–2187.

Mikolov, Tomas et al. (2013). „Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781*.

Rong, Xin (2014). „word2vec parameter learning explained". In: *arXiv preprint arXiv:1411.2738*.

Schäfer, Roland und Felix Bildhauer (2012). „Building Large Corpora from the Web Using a New Efficient Tool Chain." In: *LREC*, S. 486–493.

Wang, Zhen et al. (2014). „Knowledge Graph Embedding by Translating on Hyperplanes." In: *AAAI*. Citeseer, S. 1112–1119.

Zou, Will Y et al. (2013). „Bilingual Word Embeddings for Phrase-Based Machine Translation." In: *EMNLP*, S. 1393–1398.