

RUPRECHT-KARLS-UNIVERSITÄT HEIDELBERG

BACHELORARBEIT

**Analyse von Relationsvorhersagen im
Deutschen mit
Wortvektorrepräsentationen**

Author:
Dennis ULMER

Supervisor:
Dr. Yannick VERSLEY
Dr. Viviana NASTASE

*Eine Arbeit zur Erlangung
des Bachelorgrades*

4. Juni 2016

Eidesstattliche Erklärung

Ich, Dennis ULMER, gebe hiermit die eidesstattliche Erklärung ab, dass ich meine Bachelorarbeit mit dem Titel „Analyse von Relationsvorhersagen im Deutschen mit Wortvektorrepräsentationen“

- selbständig angefertigt
- keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und alle wörtlichen oder sinngemäß übernommenen Textstellen als solche kenntlich gemacht habe
- Mir ist bekannt, dass die ungekennzeichnete Übernahme fremder Texte – auch aus dem Internet – als Täuschung gewertet wird und die entsprechende Prüfungsleistung als nicht erbracht gilt (Bachelor-Prüfungsordnung § 8, 4 und § 21, 4; Master-Prüfungsordnung § 8, 4 und § 22, 4; Magisterprüfungsordnung, Allgemeiner Teil § 22).

Unterschrift:

Datum:

„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 von Relationsvorhersagen im Deutschen mit
Wortvektorrepräsentationen**

von Dennis ULMER

Deutscher Satz...

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

English sentence...

Danksagung

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

Inhaltsverzeichnis

Eidesstattliche Erklärung	iii
Zusammenfassung	vii
Abstract	ix
Danksagung	xi
1 Einleitung	1
1.1 Knowledge Graph Completion	1
1.2 Ansatz	2
1.3 Inhalt	3
2 Verwandte Arbeiten	5
2.1 Verwandte Arbeiten	5
3 Grundlagen	7
3.1 Neurale Netzwerke	7
3.2 Wortvektoren	9
3.3 Wortvektoren aus Abhängigkeiten	10
4 Vorbereitung	13
4.1 Vorbereitung	13
4.1.1 Extraktion von Named Entities	13
4.1.2 Aufbereitung des Korpus	13
4.1.3 Training der Wortvektoren	13
5 Evaluation der Wortvektoren	15
5.1 Evaluation der Wortvektoren	15
5.1.1 Qualitative Evaluation	15
5.1.2 Quantitative Evaluation	16
5.1.3 Evaluationsdaten	17
Wortpaarähnlichkeit	17
Analogien	18
5.1.4 Evaluationsergebnisse	18
6 Experiment A	21
6.1 Idee	21
6.2 Algorithmus	21
6.3 Parallelisierter Algorithmus	22
6.4 Clustering	24
6.5 Ergebnisse	25
6.6 Zwischendiskussion	25
6.6.1 Daten	25
6.6.2 Ansatz	26

7 Experiment B	27
7.1 Datenerzeugung	27
7.2 Training	28
7.3 Evaluation	29
8 Evaluation der Cluster	31
8.1 Evaluation der Cluster	31
9 Diskussion	33
10 Fazit	35
10.1 Fazit	35
11 Ausblick	37
11.1 Ausblick	37
A Übersicht über die Parameter zum Trainieren der Wortvektoren	39
Literatur	41

Abbildungsverzeichnis

3.1	Mathematische Modellierung eines Neurons	7
3.2	Darstellung eines neuronalen Netzwerks	8
3.3	Gegenüberstellung von Skip-Gram und CBOW	9
3.4	Erstellung von Dependenzkontexten beim Wortvektortraining	11
5.1	Liste der k nächsten Nachbarn von Wörtern in verschiedenen Datensets.	16
5.2	Anzahl der Annotatoren und Agreement (als <i>Cohen's κ</i>) der WORTPAAR-Evaluationsdatensets.	18
5.3	Evaluationsergebnisse bei Wortähnlichkeit und Analogien	19
6.1	Einfacher Projektionsalgorithmus	21
6.2	Modifizierter Projektionsalgorithmus	22
6.3	Parallelisierter Projektionsalgorithmus	23
6.4	Rechenzeiten für den Mappingschritt nach Anzahl von Threads	23
6.5	Darstellung der Funktionsweise von DBSCAN	25
7.1	Daten über die Relationsdatensets FB15k und GER14k	28
7.2	Resultate für TransE mit FB15k und GER14k	29
A.1	Quelle und Trainingsparameter für Wortvektoren	39

Abkürzungsverzeichnis

PoS	P art of S peech
SGD	S tochastic G radient D escent
DECOW	German C Orpus from the W eb
CBOW	C ontinuous- B ag- O f- W ords
DBSCAN	D ensity- B ased S patial C lustering of A pplications with N oise

Symbolverzeichnis

w	Ein nicht näher spezifiziertes Wort
w_1, w_2, w_3 / a, b, c	Mehrere nicht näher spezifizierte Wörter
$\vec{v}, \vec{y}, \vec{z}$	Vektoren
$\vec{v}(\text{Hund})$ / $\vec{v}(w)$	Der zu einem Wort zugehörige Wortvektor
\vec{v}' / $\vec{v}'(w)$.	Projektion eines (Wort-)Vektors
\oplus	Vektorkonkatenation
$\mathcal{U}, \mathcal{V}, \mathcal{W}$	Mengen
x^*, y^*, z^*	Gewünschte Ausprägung einer Variablen
$\tilde{x}, \tilde{y}, \tilde{z}$	Möglicher Bestwert einer Variablen

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, WIKIPEDIA¹

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:

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

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 Hypothese, dass das Trainieren solcher Relationen nicht möglich wäre, wenn sich die Differenzvektoren von Wortpaaren einzelner Relationen nicht ähneln würden, also z.B.

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

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 Dateneinpflagens 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 Vorbereitungs-schritte, 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 benötigten 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 9) und einem Ausblick für auf diese Arbeit aufbauende Untersuchungen an (Kapitel 10).

Kapitel 2

Verwandte Arbeiten

2.1 Verwandte Arbeiten

Bla

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 neuronalen Netzwerken auf einer mathematischen Modellierung des menschlichen Gehirns. Die Grundbausteine bilden dabei einzelne Neuronen, die dabei auf folgende Art und Weise modelliert werden (dieser Teil ist dabei auch als *Perceptron*-Algorithmus bekannt):

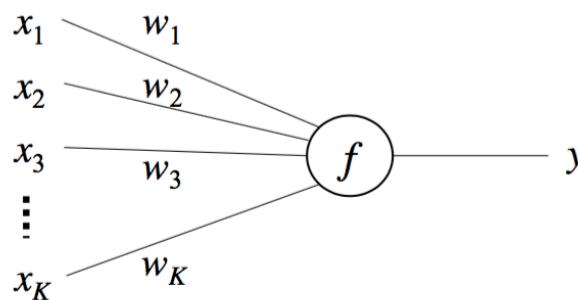


Abbildung 3.1: Mathematische Modellierung eines Neurons, auch bekannt als *Perceptron*-Algorithmus. x bezeichnet In-, y den Output des Neurons, w bilden die Gewichte und f die Aktivierungsfunktion. Abbildung aus (Rong, 2014).

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"):

$$y = f(u) \quad (3.1)$$

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

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

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

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

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

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 dadurch ableitbar:

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

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

Um aus einzelnen Neuronen nun ein neuronales 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 Abb. X).

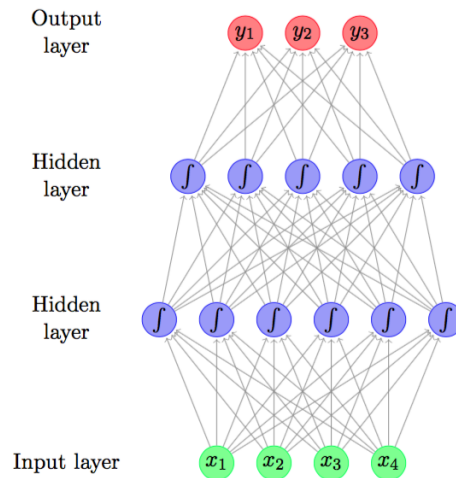


Abbildung 3.2: Darstellung eines neuronalen Netzwerks mit vier Schichten, davon jeweils eine für In- und Output, sowie zwei "versteckte" Schichten (*hidden layer*). Bild aus (Goldberg, 2015).

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 erzeugte Ergebnis von dem erwünschten 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 bestimmten Wort zugeordnet wurde. Nehmen wir beispielsweise das Minikorpus *Der Hund beißt den Mann* an. Das Vokabular besteht dann aus $V = \{\text{beißt}, \text{Der}, \text{den}, \text{Hund}, \text{Mann}\}$ (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 stellvertretend 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 $\vec{v}(\text{Hund}) = (0, 0, 0, 1, 0)$.

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 .

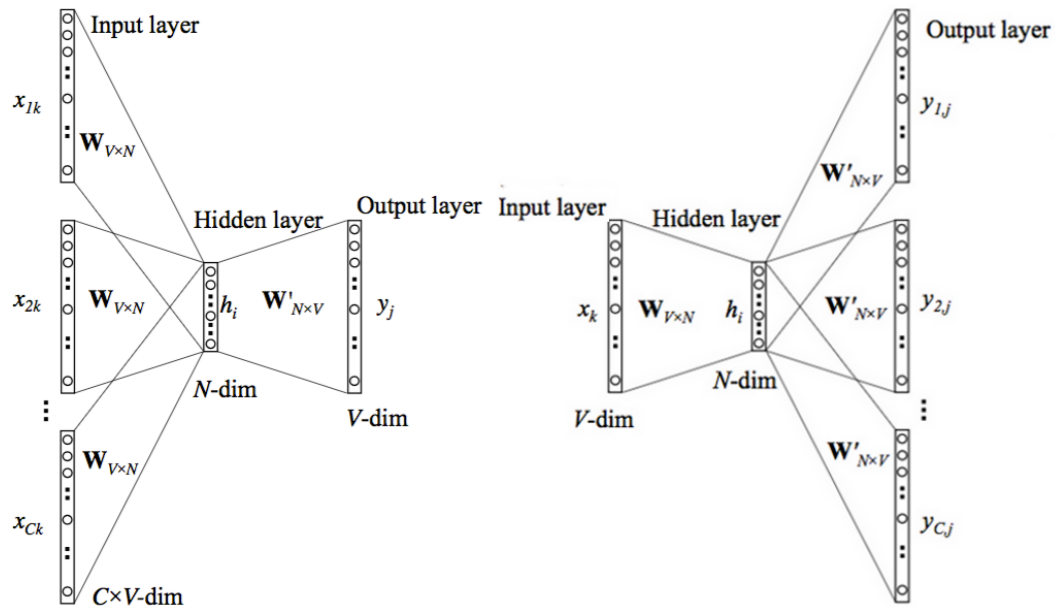


Abbildung 3.3: Gegenüberstellung der beiden Trainingsmethoden CBOW (links) und Skip-Gram (rechts). CBOW versucht die Wahrscheinlichkeit eines Wortes gegeben seines Kontexts zu trainieren, Skip-Gram die Wahrscheinlichkeit eines Kontextes gegeben eines Wortes.

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). Unsere Verlustfunktion, deren Wert es dabei zu minimieren gilt, besteht darin in der negativen logarithmischen Wahrscheinlichkeit eines Wortes gegeben seines Kontextes:

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

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

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

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 verbessern, 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.

3.3 Wortvektoren aus Abhängigkeiten

Abhängigkeitsgrammatiken untersuchen die Abhängigkeiten zwischen Wörtern eines Satzes und fügen diese in eine Abhängigkeitsstruktur ein. Anders als in der Phrasenstrukturgrammatik entsteht dabei kein Syntaxbaum mit Knoten. Worte stehen in Abhängigkeitsverhältnissen, wobei das das die Abhängigkeit verursachende Wort als *Regens*, das davon abhängige als *Abhängigkeit* bezeichnet wird.

(Levy und Goldberg) machen sich dies zunutze, um den Kontext beim Training von Wortvektoren neu zu definieren: Er besteht nun nicht mehr aus den umgebenden Wörtern im Satz, sondern aus den Abhängigkeiten: Für ein Wort w mit den Modifizierern m_1, \dots, m_k und den Kopf h besteht der Kontext nun aus $(m_1, lbl_1), \dots, (m_k, lbl_k), (h, lbl_h^{-1})$, wobei lbl stellvertretend für eine Abhängigkeitsrelation steht, ein -1 im Exponenten zeigt das Inverse einer solchen Relation an. Ein Beispiel dafür ist in Abb. X zu sehen.

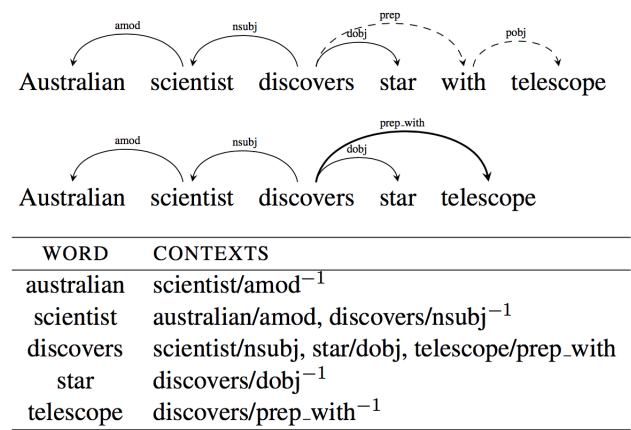


Abbildung 3.4: Beispiel der Erstellung von Wortkontexten aus Abhängigkeiten. Abhängigkeiten mit Präposition werden zu einer Abhängigkeit zusammengefasst. **Oben:** Abhängenzstruktur. **Unten:** Extrahierte Kontexte.

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 = “**C**orpus 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-Format¹ vor, wobei jedem Wort und dessen Annotationen eine ganze Zeile gewidmet ist, Satzgrenzen 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 Abhängigkeiten, ein Satz pro Zeile.

Die Abhängigkeiten 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-Modell wurde zuerst von (Mikolov u. a., 2013) vorgestellt. Die Erklärung der Funktionsweise wird im nachfolgenden Teil recht klein gehalten, für eine ausführlichere und verständliche Ausführung wird beispielsweise

¹Siehe <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 u. a.) 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 ($0 \triangleq$ Skip-gram, $1 \triangleq$ 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 DIMENSIONEN¹

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 den 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 interpretierbare 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 meistens 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 Abb. X).

¹Blogeintrag "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 Ergebnisse verschiedener Datensets zu vergleichen, da sich die Qualität der k Nachbarn nicht quantifizieren lässt: Es lässt sich vielleicht erkennen, dass 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 einfache Aufgabe wie Sentiment-Klassifikation oder Part-of-Speech-Tagging verwendet. Die Qualität der Daten wird dann anhand der Ergebnisse des Systems gemessen.

Diese extrinsische 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 $+1$ ($\hat{=}$ Äquivalenz):

$$\text{cosine_similarity}(\vec{a}, \vec{b}) = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \|\vec{b}\|} \quad (5.1)$$

Danach kann mit *Spearman's ρ* 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:

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

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

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

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

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

Dabei ist $\epsilon = 0,001$, um die Division 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 menschlichen Annotatoren zuverlässig und sinnvoll die Paare bewertet haben (zu messen z.B. mit *Cohen's κ*) 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 WORD-SIM353-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 Fig. X festgehalten.

Datenset	#Annotatoren	κ
WORTPAARE65	24	0,81
WORTPAARE222	21	0,49
WORTPAARE350	8	0,69

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

Analogen

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 Wörtern kreiert wurden.

5.1.4 Evaluationsergebnisse

Datenset	Wortähnlichkeit ($\rho \in [-1, 1]$)			Analogien (in %)		
	WORTPAARE65	WORTPAARE222	WORTPAARE350	SCHM280	GOOGLE	SEMREL
Mark I	-0,8096		-0,7302 ₍₁₃₎		44,56	
Mark II	-0,7856		-0,7156 ₍₁₃₎		40,37	
Mark III	-0,7718		-0,6886 ₍₁₃₎		27,42	35,11
Mark IV	-0,7681		-0,6834 ₍₁₃₎		25,48	32,03
Mark V	-0,7725		-0,6738 ₍₁₃₎		25,54	
Mark VI	-0,7697		-0,6784 ₍₁₃₎		25,52	
Mark VII	-0,7255		-0,6924 ₍₁₃₎		30,60	1,50
Mark VIII	-0,6149		-0,6530 ₍₁₃₎		26,98	
Mark IX	-0,6784		-0,6090 ₍₁₃₎		22,26	1,62
Mark X	-0,5949		-0,5791 ₍₁₃₎		19,22	1,22
Mark XI	-0,5890		-0,5700 ₍₁₃₎		18,60	1,38
Mark XII	-0,5874		-0,5706 ₍₁₃₎		8,62	
Mark XIII	-0,8247 ₍₁₎		-0,7494 ₍₁₃₂₎		15,51	3,01
Mark XIV	-0,8106 ₍₁₎		-0,7362 ₍₁₃₂₎		14,51	2,56
Mark XV	-0,7786 ₍₁₎		-0,7136 ₍₁₃₂₎		13,15	2,44
Mark XVI	-0,7576 ₍₁₎		-0,6990 ₍₁₃₂₎		11,76	2,56
Mark XVII	-0,7578 ₍₁₎		-0,6986 ₍₁₃₂₎		11,41	2,80
Mark XVIII	-0,7551 ₍₁₎		-0,6981 ₍₁₃₂₎	11,38		3,01
Mark XIX	-0,7589 ₍₁₎		-0,7476 ₍₁₃₂₎			2,23
Mark XX	-0,7519 ₍₁₎		-0,7239 ₍₁₃₂₎			2,15
Mark XXI	-0,7188 ₍₁₎		-0,6875 ₍₁₃₂₎		9,68	1,75
Mark XXII	-0,6926 ₍₁₎		-0,6429 ₍₁₃₂₎			1,71
Mark XXIII	-0,6851 ₍₁₎		-0,6359 ₍₁₃₂₎			1,95
Mark XXIV	-0,6913 ₍₁₎		-0,6369 ₍₁₃₂₎			2,07

Abbildung 5.3: 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.

Kapitel 6

Experiment A

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

JASON WHITTINGTON

6.1 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 ϕ , die ein Vektorenpaar in einen Relationsraum abbildet: $\phi : \mathbb{R}^d \rightarrow \mathbb{R}^e$, wobei nicht zwangsläufig $d = e$.

In ihrer einfachsten Form bildet sie einfach die Differenz \vec{d} der beiden Vektoren:

$$\phi(\vec{u}, \vec{v}) = \vec{v} - \vec{u} = \vec{d} \quad (6.1)$$

6.2 Algorithmus

Der Grundalgorithmus (siehe Fig. X) 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 6.1: Einfacher Projektionsalgorithmus.

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

$$\forall (\vec{u}, \vec{v}) \in \mathcal{C} : (\vec{v}, \vec{u}) \notin \mathcal{C} \quad (6.2)$$

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 Λ definieren, die die Anzahl der Kookkurrenzen bestimmt:

$$\Lambda(t_1, t_2) = |\{s_i | \exists w_{ij} = t_1 \wedge \exists w_{ij} = t_2 \wedge t_1 \neq t_2\}_{i=1}^n| \quad (6.3)$$

Eine mögliche Einschränkung besteht darin, in einem geänderten Algorithmus (siehe Fig. X) 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 γ 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 6.2: Modifizierter Projektionsalgorithmus, bei dem die zu den Vektoren gehörigen Begriffe über γ Mal im Korpus im gleichen Satz aufgetreten sein müssen, damit \vec{d} errechnet wird.

6.3 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 Verhältnis 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. Damit nun keiner der Threads redundante Berechnungen durchführt, prüft er, ob sein aktuelles Vektorpaar sich in dieser Menge befindet und schon abgehakt wurde¹.

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

Data : Menge von Vektorpaaren \mathcal{C}
Data : Menge von erledigten Vektorpaaren \mathcal{C}' (Anfangs $\mathcal{C}' = \emptyset$)
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 6.3: Parallelisierter Projektionsalgorithmus, bei dem die zu den Vektoren gehörigen Begriffe über γ 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.

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 Fig. X). Gerechnet wurde auf eine Hardware mit 40 Prozessorkernen und [RESTLICHE SPECS EINFÜGEN].

ANZAHL DER THREADS	BENÖTIGTE ZEIT (in m)	Beschleunigung (in %)
1		-
2		
8		
32		
64		
128		

Abbildung 6.4: 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.

6.4 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 verschiedenen Grundannahmen, aus denen es zu wählen gilt. Für die Aufgaben in dieser Arbeit wurde der DBSCAN-Algorithmus (Density-Based Spatial Clustering of Applications with Noise) 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:

- Ein Punkt p in den Daten wird dann als Kernpunkt (*core point*) bezeichnet, sofern mindesten $minPts$ Punkte innerhalb einem Radius ϵ vorhanden sind.
Diese Punkte sind von p *direkt erreichbar*.
- Ein Punkt q ist von p erreichbar, sofern es einen Pfad p_1, \dots, 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.

Der Algorithmus ist von linearer Komplexität, sofern ihm eine geeignete Indexierung der Daten zugrunde gelegt wird, ansonsten verhält sich die Komplexität quadratisch zur Anzahl der Datenpunkte³.

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

³Bei 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)

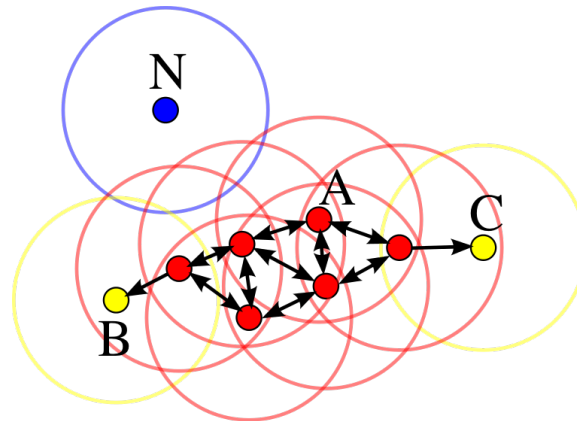


Abbildung 6.5: Darstellung der Funktionsweise von DBSCAN. Punkt A ist ein Kernpunkt, die Punkte B und C sind von A aus erreichbar. Punkt N ist nicht erreichbar und damit ein Ausreißer.²

6.5 Ergebnisse

6.6 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.

6.6.1 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 der 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. Die resultierenden Cluster sind sehr schwammig, da sie sich schlecht vom Hintergrundrauschen abgrenzen. Dies ist beispielsweise durch den Wert des Silhouettenkoeffizienten⁴ erkennbar, der bei den Experimenten immer kurz unter Null lag⁵.

⁴Der Silhouettenkoeffizient s_C beschreibt das durchschnittliche Verhältnis vom Abstand eines Punktes zu seinem Clusterzentrum gegenüber des nächsten Clusterzentrums. $-1 \leq s_C \leq 1$.

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

- **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:

$$\vec{v}(t_0) - \vec{v}(h_0) \approx \vec{v}(t_1) - \vec{v}(h_1) \approx \dots \approx \vec{v}(t_m) - \vec{v}(h_m) \quad (6.4)$$

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 u. a., 2013) oder (Lin u. a., 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:

$$\begin{aligned} \vec{v}(\text{Julia}) \approx \vec{v}(\text{Hans}) \wedge \vec{v}(\text{Barcelona}) \approx \vec{v}(\text{Paris}) &\implies \\ \vec{v}(\text{Julia}) - \vec{v}(\text{Barcelona}) \approx \vec{v}(\text{Hans}) - \vec{v}(\text{Paris}) &\end{aligned} \quad (6.5)$$

Anders ausgedrückt: Auch wenn sich semantische Information in den Wortvektoren dadurch manifestiert, dass ähnliche Ausdrücke in ihren Clustern nahe beieinander liegen, können Distanzvektoren aus Wortpaaren 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 sich mit dem in 6.1 eingeführten Parameter γ zwar verringern, allerdings nicht gänzlich verhindern lässt. Das Ziel, neue und legitime Relationen zu finden, wird durch diesen Trugschluss ad absurdum geführt.

6.6.2 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).

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 7

Experiment B

In diesem Kapitel soll der Ansatz von (Bordes u. a., 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 schließlich eine Evaluation und eine Gegenüberstellung zu den Originaldaten erfolgt.

7.1 Datenerzeugung

In (Bordes u. a., 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 abgeschaltet².

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 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 Gründe 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

Die Plausibilität dieser Gründe ist diskutabel: Nicht alle Begriffe brauchen eine Übersetzung, so sind beispielsweise Personennamen i.d.R. durch

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

²Siehe <https://en.wikipedia.org/wiki/Freebase> (zuletztabgerufenam(20.05.16))

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*.

DATENSATZ	#TRIPEL	#ENTITÄTSTYPEN	#RELATIONSTYPEN
FB15k	592.213	14.951	1.345
GER14k	459.724	14.334	1.236

Abbildung 7.1: Daten über die Relationsdatensets FB15k und GER19k. Aufgelistet ist die Anzahl der Tripel (Datensätze), Entitäts- und Relationstypen.

7.2 Training

Gegeben ist eine Menge von Relationstripeln S der Form $(h, l, t) \in S$. Zusätzlich gibt es noch eine Entitätsmenge E sodass $h, t \in E$ und eine Menge von Relationen L mit $l \in L$. Den Entitäten und Relationen werden zusätzlich 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 korruptierten bzw. “falschen” Tripeln S' erstellt, indem für jedes Tripel in S entweder h oder t durch eine andere, zufällig gewählte Entität ersetzt wird:

$$S' = \{(h', l, t) | h' \in E\} \cup \{(h, l, t') | t' \in E\} \quad (7.1)$$

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

$$d_1(h + l, t) = \sum_{i=1}^k \|h_i + l_i - t_i\| \quad (7.2)$$

$$d_2(h + l, t) = \sqrt{\sum_{i=1}^k \|h_i + l_i - t_i\|^2} \quad (7.3)$$

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

$$\mathcal{L} = \sum_{(h,l,t) \in S} \sum_{(h',l',t') \in S'} \max(0, \gamma + d_p(h + l, t) - d(h' + l', t')) \quad (7.4)$$

Der Parameter γ 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 korruptierte 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 korruptierte Tripel. Das Training wird solange ausgeführt, bis die Fehlerrate auf dem Validationsset konvergiert³.

Für das Reproduzieren der Ergebnisse wurde den Empfehlungen von (Bordes u. a., 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.

7.3 Evaluation

DATENSET	MEAN RANK		HITS@10	
	<i>Raw</i>	<i>Filtered</i>	<i>Raw</i>	<i>Filtered</i>
FB15k	575,54	197,02	34,8	48,79
GER14k	234,73	222,05	34,07	41,83

Abbildung 7.2: Resultate für TransE, trainiert auf dem FB15k- und dem GER14k-Datenset. Hits10 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 korruptierten 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.

³Konvergenz auf dem Trainingsset könnte ein Zeichen für Overfitting sein.

Kapitel 8

Evaluation der Cluster

8.1 Evaluation der Cluster

Bla

Kapitel 9

Diskussion

Mit dem Wissen wächst der Zweifel.

JOHANN WOLFGANG VON GOETHE

Kapitel 10

Fazit

10.1 Fazit

Bla

Kapitel 11

Ausblick

11.1 Ausblick

Bla

Anhang A

Übersicht über die Parameter zum Trainieren der Wortvektoren

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

Literatur

- Bordes, Antoine u. a. (2013). „Translating embeddings for modeling multi-relational data“. In: *Advances in Neural Information Processing Systems*, S. 2787–2795.
- 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>.
- 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 u. a. (2015). „Learning Entity and Relation Embeddings for Knowledge Graph Completion.“ In: *AAAI*, S. 2181–2187.
- Mikolov, Tomas u. a. (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.