Julius-Maximilians-Universität Würzburg
Philosophische Fakultät
Institut für deutsche Philologie/Computerphilologie
und Neuere Deutsche Literaturgeschichte
Seminar: Word Embeddings organisieren
Dozent: Dr. Steffen Pielström
Sommersemester 2020

Word Embeddings: Ein Überblick
(Teil 1, Rechercheaufgabe)

Name: Sophia Ackermann
E-Mail: sophia.ackermann@stud-mail.uni-wuerzburg.de
Matrikelnummer: 2125355
Studiengang: MA 45/45 (Digital Humanities/
Allgemeine und angewandte Sprachwissenschaft)
Prüfungsordnung: PO 2016
Teilmodul: 04-DH-K-1-161-m01 (Digitale Objekte verwalten 1)
Prüfungsnummer: 315328

Word Embeddings - Ein Überblick

August 7, 2020

1 Word Embeddings

Word Embeddings (WE) sind eine mächtige Technik im Bereich des Natural Language Processing (NLP), bei der Wörter als Vektoren in einem n-dimensionalen Raum dargestellt werden (vgl. Osinga 2018). Diese Vektoren werden Embeddings genannt, da die Wörter gewissermaßen im Vektorraum eingebettet sind (vgl. Jurafsky, Martin 2019: 6). Vektoren, die Wörter mit einer ähnlichen semantischen Bedeutung repräsentieren, liegen in diesem Raum näher beieinander als welche, die sich auf Wörter mit sehr unterschiedlichem Sinn beziehen (vgl. Osinga 2018). Die Verbindung zwischen der Ähnlichkeit in der Verteilung von Wörtern und der Ähnlichkeit in ihrer Bedeutung nennt man in der Semantik Verteilungshypothese (distributional hypothesis), eine Theorie, welche erstmals in den 1950er Jahren von den Linguisten Joos, Harris und Firth formuliert wurde und die besagt, dass Wörter mit ähnlicher Bedeutung auch in ähnlichem Kontext vorkommen (vgl. Jurafsky, Martin 2019: 1). WE arbeiten deshalb mit kleinen Kontexten des zu untersuchenden Wortes wie Sätzen oder Teilsequenzen eines Satzes (vgl. Thomas 2020).

Trotz ihrer jüngsten Popularität sind WE ein überraschend alter Ansatz: Bereits in den 1950er Jahren wurden in der psychologischen Forschung zur Wortassoziation stabile Muster bei verschiedenen Probanden vorgefunden, die als eine Art semantischer Raum interpretiert werden können (vgl. Hellrich 2019: 34). In den 60er und 70er Jahren kam es dann durch verschiedene Forscher zu ersten Versuchen, Beziehungen zwischen Wort und Text in (meist analogen) Document-Term-Matrizen darzustellen, die im Information Retrieval (IR) Research verwendet wurden (vgl. ebd. 34f.). Eine erste Bemühung der Wortrepräsentation durch Vektoren in Bezug zu anderen Wörtern (anstatt zu Dokumenten) wurde im Jahr 1965 durch Switzer vorgenommen (vgl. Switzer 1965: 163–171). Laut Hellrich kann Switzer deshalb als (einer der) Erfinder von WE angesehen werden, auch wenn er keine Implementierung durchführte (vgl. Hellrich 2019: 35).

2 LSA

Anfang der 1990er Jahre entwickelten einige Forscher um Deerwester die sogenannte Latent Semantic Analysis (LSA), manchmal auch Latent Semantic Indexing (LSI) (vgl. Deerwester et al. 1990: 391), „a fully automatic mathematical/statistical technique for extracting and inferring relations of expected contextual usage of words in passages of discourse" (Landauer et al. 1998: 8). LSA verarbeitet ein gegebenes Textkorpus und repräsentiert die darin enthaltenen Wörter und Wortgruppen als Punkte in einer vieldimensionalen semantischen Matrix. LSA ähnelt neuronalen Netzen, basiert jedoch auf der Singular Value Decomposition (SVD) (vgl. ebd. 3).

Zunächst wird dabei der zu untersuchende Text als Document-Term-Matrix dargestellt, in der jede Zeile für ein eindeutiges Wort und jede Spalte für einen Kontext, zum Beispiel eine Textpassage, steht. Jede Zelle enthält die Häufigkeit, mit der das Wort in der entsprechenden Passage erscheint.

Anschließend werden die Zelleneinträge einer vorläufigen Gewichtung unterzogen, die sowohl die Bedeutung des Wortes in der jeweiligen Passage als auch die Wortart berücksichtigt (vgl. ebd. 8). Darauffolgend wendet LSA SVD auf die Matrix an, eine mathematische Matrixzerlegungstechnik, bei der die rechteckige Document-Term-Matrix in das Produkt von drei weiteren Matrizen aufgeteilt wird. Die erste Matrix beschreibt die ursprünglichen Zeilenentitäten als Vektoren von abgeleiteten orthogonalen Faktorwerten, die zweite Matrix beschreibt die ursprünglichen Spaltenentitäten auf dieselbe Weise und die dritte ist eine Diagonalmatrix, die Skalierungswerte enthält, sodass, wenn die drei Komponenten multipliziert werden, die ursprüngliche Matrix rekonstruiert wird, wobei die Dimensionen zur Vereinfachung reduziert werden (vgl. ebd. 8f.). Abbildung 1 zeigt schematisch, wie SVD funktioniert (vgl. Anhang 1).

Schließlich werden die beiden zu untersuchenden Begriffe miteinander verglichen, indem der Kosinus zwischen den beiden Vektoren im reduzierten Dimensionsraum als Ähnlichkeitsmaß hergenommen wird (vgl. Deerwester et al. 1990: 398f.). Manchmal wird darüber hinaus die Länge der Vektoren als Maß verwendet, um abzubilden, wie häufig ein bestimmtes Thema behandelt wurde (vgl. Landauer et al. 1998: 16).

LSA wurde ursprünglich für den Bereich des IR entwickelt. Die meisten Ansätze im IR hängen von einer lexikalischen Übereinstimmung zwischen den Wörtern in der Benutzeranfrage und denen im Dokument ab. Das Problematische hierbei ist die große Menge an irrelevanten Informationen, welche bei einer derartigen Suche abgerufen werden, die die relevanten Treffer überlagern. (vgl. Dumais 2005: 189). „The key insight in LSA was to reduce the dimensionality of th[is] information retrieval problem" (ebd. 189). Ein großer Vorteil von LSA beim Abrufen und Filtern von Daten ist, dass Dokumente angesprochen werden können, auch wenn keine exakte Übereinstimmung mit den Abfragewörtern vorhanden ist (vgl. ebd. 220f.). Abgesehen vom IR kommt LSA auch bei verwandten Aufgabenbereichen wie Textklassifikation oder Clustering zum Einsatz (vgl. ebd. 220).

3 Word2Vec

Das Lernen von niedrigdimensionalen Vektoren aus Textkorpora kann alternativ durch Neuronale Netze (NN) erfolgen. „Neural networks [are] suitable candidates for this purpose due to their efficiency and speed in processing large amounts of texts and for their ability in learning dense representations" (Pilehvar, Camacho-Collados 2020: 31). Das im Jahr 2013 von Mikolov und seinem Team bei Google entwickelte Word2Vec basiert auf einer einfachen, aber effizienten Feed-Forward-NN-Architektur, die mit dem Ziel der Sprachmodellierung trainiert wurde.

Word2Vec-WE sind den älteren SVD-Ansätzen bei einer Vielzahl von Aufgaben überlegen. Sie werden nicht durch Dimensionsreduktion aus einer Koexistenz-Matrix abgeleitet, sondern mit zufälligen Werten initialisiert und dann so trainiert, dass wahrscheinliche Kontextwörter für ein fragliches Wort vorhergesagt werden (vgl. Hellrich 2019: 29). Für beide Ansätze werden Kontexte mit einem Wahrscheinlichkeitsfenster und einem Frequenzfaktor heruntergerechnet, im Gegensatz zu SVD verarbeiten die Word2Vec-Modelle den Text aber ohne vorherige Erstellung einer Document-Term-Matrix (vgl. ebd. 29).

Word2Vec besteht aus zwei verschiedenen, aber verwandten Modellen: Continuous Bag-Of-Words (CBOW) und Sikp-gram (SG) (vgl. Pilehvar, Camacho-Collados 2020: 31). „The CBOW model aims at predicting the current word using its surrounding context, minimizing the following loss function:

$$E = -log\ (p((w_t)\ |(W_t)))$$

where w_t is the target word and $W_t = w_(t-n), ..., w_t, ..., w_(t+n)$ represents the sequence of words in context. The [SG] model is similar to the CBOW model but in this case the goal is to predict the words in the surrounding context given the target word, rather than predicting the target word itself" (ebd. 31). Genauer gesagt wird bei SG jedes Wort als Eingabe für einen logarithmisch linearen Klassifikator mit kontinuierlichem Projection-Layer verwendet und Wörter innerhalb eines bestimmten Bereichs vor und nach dem aktuellen Wort vorhergesagt (vgl. Mikolov et al. 2013: 4). Abbildung 2 zeigt die Architektur des CBOW und SG Modells von Word2Vec vereinfacht auf (vgl. Anhang 2). Sie besteht aus je einem Input-, Hidden- oder auch Projection- und Output-Layer (vgl. Mikolov et al. 2013: 4f.). Der Input-Layer ist so groß wie das Vokabular und kodiert den Kontext als eine Kombination von One-Hot-Vektor-Darstellungen der umliegenden Wörter eines bestimmten Zielworts. Der Output-Layer hat dieselbe Größe wie der Input-Layer und enthält während der Trainingsphase einen One-Hot-Vektor des Zielwortes (vgl. Pilehvar, Camacho-Collados 2020: 31f.).

Ein interessanter Anwendungsfall von SG-WE ist neben Wortsimilarität die Wortanalogie, das heißt die Übertragung der semantischen Beziehung zwischen zwei Wörtern auf andere Wörter durch Verrechnung der Vektoren: „The common example is king – man + woman ~ queen. This means that you subtract the man vector from the king vector, then add the woman vector" (Thomas 2020).

Folgender Code zeigt eine exemplarische Implementierung des SG-Word2Vec-Modells mithilfe der Python-Bibliothek Gensim.

[1]:
```
import gensim.downloader as api
```

[2]:
```
# Pre-trained Word2Vec-Modell über Gensim erhältlich
# (auf einen Teil des Google News Dataset trainiert,
# enthält ca. 3 Million Wörter und Phrasen)

wv = api.load('word2vec-google-news-300')
```

[3]:
```
# Gibt die Similaritätsrelation der beiden Vektoren aus
# (je größer die Zahl, desto ähnlicher die Wörter)

pairs_king = [('king', 'queen'),
              ('king', 'prince'),
              ('king', 'princess'),
              ('king', 'palace'),
              ('king', 'apple')]
for w1, w2 in pairs_king:
    print((w1, w2, wv.similarity(w1, w2)))
```

```
('king', 'queen', 0.6510957)
('king', 'prince', 0.61599934)
('king', 'princess', 0.5161999)
('king', 'palace', 0.4878291)
('king', 'apple', 0.10826096)
```

[4]:
```
# Gibt die 5 ähnlichsten Vektoren des Vektors 'king' aus

wv.most_similar(positive=['king'], topn=5)
```

[4]:
```
[('kings', 0.7138046026229858),
 ('queen', 0.6510956883430481),
 ('monarch', 0.6413194537162781),
 ('crown_prince', 0.6204220056533813),
 ('prince', 0.6159993410110474)]
```

[5]:
```
# Vergleicht 2 Vektoren

wv.similarity('knight', 'horse')
```

[5]:
```
0.3580188
```

[6]:
```
# Wortanalogie: Welches Wort hat dieselbe Beziehung zu 'woman',
# wie 'king' zu 'man'?

wv.most_similar(positive=['king', 'woman'], negative=['man'], topn=1)
```

[6]:
```
[('queen', 0.7118192911148071)]
```

[7]:
```
wv.most_similar(positive=['Berlin', 'Italy'], negative=['Germany'], topn=1)
```

[7]:
```
[('Rome', 0.7297912836074829)]
```

4 GloVe

Ein weiteres beliebtes WE-Modell ist GloVe, das im Jahr 2014 von einem Team um Pennington in Stanford entwickelt wurde. „[It is] GloVe, for Global Vectors, because the global corpus statistics are captured directly by the model" (Pennington et al. 2014: 2).

GloVe basiert auf der Idee, dass das Verhältnis der Wahrscheinlichkeiten für das gleichzeitige Auftreten von zwei Wörtern, w_i und w_j, mit einem dritten Wort w_k, das heißt $P(w_i, w_k)/P(w_j, w_k)$, eher ein Hinweis auf ihre semantische Assoziation als eine direkte Wahrscheinlichkeit des gleichzeitigen Auftretens, also $P(w_i, w_j)$, ist (vgl. Pilehvar, Camacho-Collados 2020: 33). „To achieve this, they propose an optimization problem which aims at fulfilling the following objective

$$w_i^T w_k + b_i + b_k = log\ (X_i k)$$

where b_i and b_k are bias terms for word w_i and probe word w_k and $X_i k$ is the number of times w_i co-occurs with w_k" (ebd. 33). Durch das Erreichen dieses Ziels wird der Unterschied zwischen dem Skalarprodukt von w_i und w_k und dem Logarithmus ihrer Anzahl von Co-Vorkommen minimiert, was zur Konstruktion der Vektoren w_i und w_k führt, deren Skalarprodukt einen verwendbaren Wert ihrer transformierten Koexistenzzahlen angibt (vgl. ebd. 33).

GloVe ist ein neues globales logarithmisch bilineares Regressionsmodell, welches die Vorteile der beiden Hauptvorgängermodelle miteinander kombiniert: GloVe verwendet einerseits globale Matrix-Faktorisierung wie in LSA und baut andererseits auf den mathematischen Strukturen von Word2Vec auf, wobei ein strukturell verbesserter Vektorraum produziert wird (vgl. Pennington et al. 2014: 1). Das Modell nutzt effizient statistische Informationen durch zufälliges Training der Nicht-Null-Elemente in einer Term-Term-Matrix, anstatt der gesamten Matrix wie bei LSA oder der individuellen Kontextelement eines großen Korpus wie bei SG oder CBOW (vgl. ebd. 1). Dabei ist wichtig anzumerken, dass es sich bei GloVe nicht um eine NN-Architektur handelt, wie bei den Word2Vec-Modellen (vgl. Thomas 2020).

Ein Vorteil von GloVe ist, dass es das Ergebnis der direkten Modellierung von Beziehungen ist, anstatt diese nur als Nebeneffekt beim Trainieren eines Sprachmodells zu erhalten (vgl. ebd.). „The result, GloVe, is a new global log-bilinear regression model for the unsupervised learning of word representations that outperforms other models on word analogy, word similarity, and named entity recognition tasks" (Pennington et al. 2014: 10).

Im Folgenden wird GloVe anhand eines vortrainierten Modells der Stanford-University vorgeführt.

[1]:
```
import numpy as np
from scipy import spatial
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
```

[2]:
```
# Pre-trained GloVe-Modell kann hier heruntergeladen werden:
# https://nlp.stanford.edu/projects/glove/
# (Wikipedia 2014 + Gigaword 5 (6B tokens, 400K vocab, uncased,
# 50d, 100d, 200d, & 300d vectors, 822 MB download): glove.6B.zip)

glove = {}

with open('glove.6B/glove.6B.50d.txt',
          'r', encoding='utf-8',
          errors='ignore') as corpus:
    for line in corpus:
        values = line.split()
        word = values[0]
        vector = np.asarray(values[1:], 'float32')
        glove[word] = vector
```

[3]:
```
def most_similar(embedding):
    return sorted(glove.keys(), key=lambda word: spatial.distance.
 ↪euclidean(glove[word], embedding))
```

[4]:
```
# Gibt die 3 ähnlichsten Vektoren des Vektors 'strawberry' aus

most_similar(glove['strawberry'])[:3]
```

[4]:
```
['strawberry', 'cherry', 'shortcake']
```

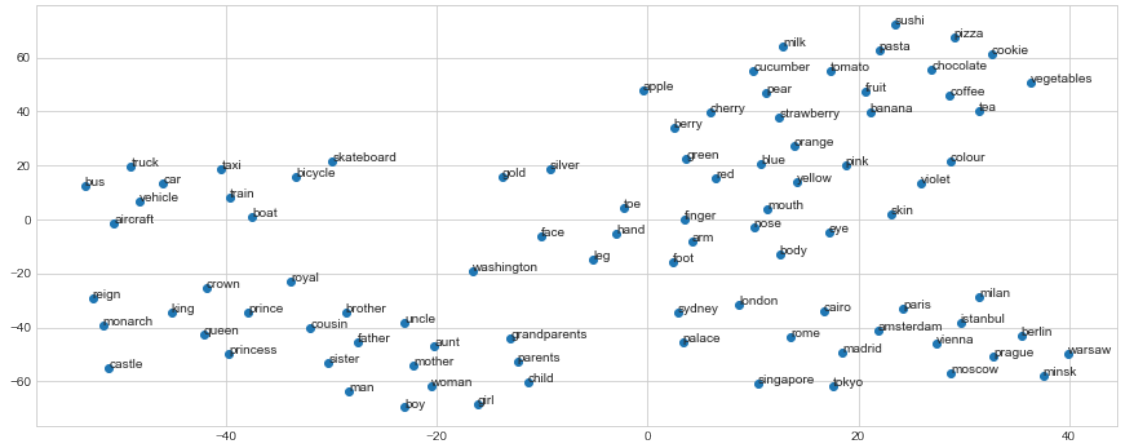
[5]:
```
# Wortanalogie: Welches Wort hat dieselbe Beziehung zu 'germany',
# wie 'paris' zu 'france'?

most_similar(glove['paris'] - glove['france'] + glove['germany'])[:1]
```

[5]:
```
['berlin']
```

[6]:
```
# Visualisierung einer Reihe von Wortvektoren im 2-dimensionalen Raum

tsne = TSNE(n_components=2, random_state=0)
words =  ['king', 'queen', 'prince', 'princess', 'palace', 'monarch', 'crown',
          'reign', 'castle', 'royal', 'man', 'woman', 'boy', 'girl', 'mother',
          'father', 'sister', 'brother', 'uncle', 'aunt', 'cousin', 'parents',
          'grandparents', 'child', 'apple', 'banana', 'pear', 'strawberry',
          'cherry', 'fruit', 'berry', 'vegetables', 'tomato', 'cucumber',
          'pizza', 'pasta', 'sushi', 'cookie', 'chocolate', 'milk', 'coffee',
          'tea', 'car', 'boat', 'aircraft', 'train', 'vehicle', 'skateboard',
          'bicycle', 'bus', 'taxi', 'truck', 'berlin', 'amsterdam', 'paris',
          'rome', 'london', 'washington', 'cairo', 'tokyo', 'singapore',
          'sydney', 'prague', 'istanbul', 'moscow', 'madrid', 'vienna', 'minsk',
          'warsaw', 'milan', 'face', 'body', 'finger', 'arm', 'leg', 'foot',
          'hand', 'toe', 'nose', 'eye', 'mouth', 'skin', 'red', 'blue', 'green',
          'yellow', 'orange', 'pink', 'violet', 'gold', 'silver', 'colour']
vectors = [glove[word] for word in words]
plt.figure(figsize=[15,6])
plt.style.use('seaborn-whitegrid')
Y = tsne.fit_transform(vectors[:200])
plt.scatter(Y[:, 0], Y[:, 1])
for label, x, y in zip(words, Y[:, 0], Y[:, 1]):
    plt.annotate(label, xy=(x, y), xytext=(0, 0), textcoords="offset points")
plt.show()
```



5 fastText

Im Jahr 2015 entwickelte Facebook Research eine Erweiterung von Word2Vec namens fastText (vgl. Thomas 2020). FastText „[is] an open-source library for computing word embeddings by summing embeddings of the bag of character n-grams that make up a word" (Jurafsky, Martin 2019: 26).

Ein häufig auftretendes Problem bei Word2Vec ist die Art und Weise, wie unbekannte Wörter behandelt werden, das heißt Wörter, die nicht im Vokabular des Trainingskorpus enthalten sind (vgl. Thomas 2020). Bei manchen Problemstellungen kann ein Weglassen der unbekannten Wörter, unter der Annahme, dass sie zu selten sind, um einen signifikanten Einfluss auf das Ergebnis nachgeschalteter Prozesse zu haben, Sinn machen. In Korpora mit fachspezifischem Vokabular ist es allerdings nicht ungewöhnlich, ein für das Dokument wichtiges Wort zu finden, das möglicherweise nicht in den Trainingsdaten enthalten ist. Dieses sogenannte Out-of-vocabulary-Problem macht eine Durchführung des Transfer learning schwierig (vgl. ebd.). Beim Transfer learning wird ein Teil des Modells oder das gesamte Modell, das für einen Datensatz und eine Aufgabe trainiert wurde, auf einen anderen Datensatz und eine andere Aufgabe transferiert. Word2Vec beherrscht Transfer learning, indem es ein Modell für ein häufig auftretendes Sprachmodellierungsproblem anfertigt und es anschließend teilweise für andere NLP-bezogene Aufgaben wiederverwendet (vgl. ebd.).

FastText erleichtert das Transfer learning mithilfe von WE durch das Erlernen von Informationen auf Zeichenebene. Anstatt also eine higher-level Darstellung von Tokens zu lernen, merkt es sich eine higher-level Repräsentation der Zeichensequenzen, die in Vektoren umgewandelt und dann verrechnet werden (vgl. ebd. 2020). „Each word in fasttext is represented as itself plus a bag of constituent n-grams, with special boundary symbols < and > added to each word. For example, with n = 3 the word where would be represented by the character n-gram [...] <wh, whe, her, ere, re> plus the sequence [.] Then a skipgram embedding is learned for each constituent n-gram, and the word where is represented by the sum of all of the embeddings of its constituent n-grams" (Jurafsky, Martin 2019: 28).

Auch fastText soll im Folgenden beispielhaft implementiert werden. Hierbei wird, ähnlich wie bei der Implementierung von Word2Vec, ein vortrainiertes Modell der Gensim-Bibliothek verwendet.

[1]:
```
# Pre-trained fastText-Modell über Gensim erhältlich

import gensim.downloader as api
ft = api.load('fasttext-wiki-news-subwords-300')
```

[2]:
```
# Gibt die Similaritätsrelation der beiden Vektoren aus

pairs_fruit = [('fruit', 'apple'),
               ('fruit', 'pear'),
               ('fruit', 'banana'),
               ('fruit', 'strawberry'),
               ('fruit', 'tomato')]
for w1, w2 in pairs_fruit:
    print((w1, w2, ft.similarity(w1, w2)))
```

```
('fruit', 'apple', 0.65969634)
('fruit', 'pear', 0.6272378)
```

```
('fruit', 'banana', 0.635579)
('fruit', 'strawberry', 0.617207)
('fruit', 'tomato', 0.5951181)
```

[3]:
```
# Gibt die 5 ähnlichsten Vektoren des Vektors 'fruit' aus

ft.most_similar(positive=['fruit'], topn=5)
```

[3]:
```
[('fruits', 0.8579899072647095),
 ('fruit-bearing', 0.7444858551025391),
 ('fruit-', 0.7373621463775635),
 ('starfruit', 0.7342027425765991),
 ('fresh-fruit', 0.7309085130691528)]
```

[4]:
```
# Vergleicht 2 Vektoren

ft.similarity('fruit', 'vegetable')
```

[4]:
```
0.7002144
```

[5]:
```
# Wortanalogie: Welches Wort hat dieselbe Beziehung zu 'girl',
# wie 'prince' zu 'boy'?

ft.most_similar(positive=['prince', 'girl'], negative=['boy'], topn=1)
```

[5]:
```
[('princess', 0.8643317818641663)]
```

6 Contextualized Word Embeddings

In den letzten Jahren kam ein neuer Trend im Bereich der WE auf: Die sogenannten Contextualized Word Embeddings (CWE) können als neue Generation von WE betrachtet werden, deren Innovation vor allem in der Empfindlichkeit der Darstellung eines Wortes für den Kontext liegt (vgl. Pilehvar, Camacho-Collados 2020: 74). Die Einbettung eines Zielwortes kann sich je nach seinem Kontext ändern, das WE ist also dynamisch. Diese Dynamik verringert viele Probleme, die mit der statischen WE-Architektur der Vorgängermodelle einherkommen und erfassen zuverlässig die semantischen und syntaktischen Eigenschaften der natürlichen Sprache im Kontext (vgl. ebd. 74). Trotz ihres jungen Alters brachten CWE bereits in nahezu jeder Downstream-NLP-Aufgabe erhebliche Verbesserungen im Vergleich zu den statischen Vorgängermodellen wie Word2Vec oder GloVe (vgl. ebd. 74).

Kontextunabhängige (statische) WE sind Fixpunkte im semantischen Raum, die sich nicht ändern, unabhängig vom Kontext, in dem das Zielwort erscheint. Die WE werden als Features eingeführt, normalerweise im Input-Layer des Modells (vgl. ebd. 75). Im kontextbasierten Modell wird der Kontext des Zielwortes bei der Verarbeitung mitberücksichtigt (vgl. ebd. 83).

Abhängig von dem bei der Sprachmodellierung verwendeten Sequence Encoder können die CWE-Modelle in zwei große Kategorien eingeteilt werden: Recurrent Neural Networks (RNN) und Transformer Networks (vgl. ebd. 85).

RNNs sind eine spezielle Art von NN, die sich durch ihre Wiederholung auszeichnen: Im Gegensatz zu Feed-Forward-Netzwerken verfügen RNNs über Rückkopplungsschleifen, die dem NN eine zeitliche Dynamik ermöglichen und sich an die Vergangenheit ‚erinnern' (vgl. ebd. 13). Eine verbreitete Erweiterung des RNN ist das bidirektionale RNN (BiRNN), bei dem die Eingabesequenz von Anfang bis Ende und auch von Ende bis Anfang eingespeist wird, was einen Zugriff auf ‚zukünftige' Zeitschritte gewährleistet, das heißt auf die nächsten Wörter in der Eingabesequenz (vgl. ebd. 14f.). Long Short-Term Memory (LSTM) ist eine Variation von RNN. „[T]he main difference with simple RNNs [...] lies in the 'carry' track which transfers cell states across timesteps. The carry track works as the memory of this network. An internal mechanism, called gate, allows the memory in LSTMs to last longer" (ebd. 16). LSTMs können die Wortreihenfolge erfassen sowie Wortrepräsentationen auf vernünftige Weise kombinieren und den Wörtern, die im Kontext eine zentrale semantische Rolle einnehmen, eine höhere Gewichtung geben (vgl. ebd. 85). Bei den meisten RNN-basierten CWE-Modellen liegt eine bidirektionale LSTM-Architektur (BiLSTM) vor.

Mit dem Transformer-Modell wurde 2017 eine leistungsfähigere Alternative zu RNNs eingeführt, die vor allem das Gebiet der maschinellen Übersetzung revolutionierte (vgl. ebd. 23). Die Transformer-Architektur ist ein Feed-Forward-NN, das aus einem Encoder und einem Decoder besteht. Im Gegensatz zu RNNs, die Eingabe-Tokens nacheinander einzeln empfangen, nimmt der Transformer alle Tokens in der Sequenz gleichzeitig auf, was ihn für die parallele Verarbeitung effizienter macht. Darüber hinaus gewährleistet das Modell eine Erfassung von Fernabhängigkeiten (vgl. ebd. 23).

Im Folgenden soll auf zwei der aktuell populärsten CWE-Verfahren näher eingegangen werden, die auf den eben beschriebenen Modellen basieren: ELMo und BERT.

7 ELMo

Embeddings from language models (ELMo) sind ein CWE-Modell, das 2018 am Allen Institute for Artificial Intelligence entwickelt wurde (vgl. Peters et al. 2018). Das Sprachmodell lernt ein Wort aufgrund der Wörter, die im Textkorpus davor und danach kommen. Das Modell lernt zwar auf Zeichenebene, die WE sind aber wortbasiert (vgl. Thomas 2020). Dies ermöglicht ELMo, auch Darstellungen von Wörtern außerhalb des Wortschatzes des Trainingskorpus zu bilden (vgl. Taylor 2019). Die Art und Weise, wie ELMo verwendet wird, unterscheidet sich somit stark von Word2Vec oder fastText: Anstatt eine Art Wörterbuch zum Nachschlagen von Wörtern und ihren entsprechenden Vektoren zu haben, erstellt ELMo stattdessen Vektoren im laufenden Betrieb, indem Text durch das NN-Modell geleitet wird (vgl. ebd.).

ELMo liegt ein mehrschichtiges BiLSTM zugrunde: Die ELMo-Darstellung für das Zielwort ist eine Kombination der hidden states der beiden BiLSTM-Layer h_k1 und h_k2, die die kontextsensitive Darstellung jedes Wortes codieren, und der statischen, zeichenbasierten Darstellung des Wortes x_k (vgl. Pilehvar, Camacho-Collados 2020: 86f.). Abbildung 3 stellt die Funktionsweise von ELMo vereinfacht dar (vgl. Anhang 3). „A residual connection between the LSTM layers allows the deeper layer(s) to have a better look at the original input and to allow the gradients to better backpropagate to the initial layers" (vgl. Pilehvar, Camacho-Collados 2020: 87).

ELMo wird auf große Textmengen trainiert mit dem Ziel der Sprachmodellierung, das heißt anhand gegebener Tokens sollen die nächsten vorausgesagt werden (vgl. Alammar 2018). Die ELMo-Darstellungen verbessern den Stand der Technik zudem bei weiteren herausfordernden NLP-Problemen erheblich, einschließlich Question Answering und Sentiment Analyse (vgl. Peters et al. 2018: 1).

8 BERT

Mit der Veröffentlichung von Bidirectional Encoder Representations from Transformers (BERT), ein von einem Team um Devlin bei Google Research entwickeltes CWE-Modell, wurde 2018 das NLP-Feld revolutioniert (vgl. Devlin et al. 2018). BERT basiert auf einem mehrschichtigen bidirektionalen Transformer inklusive der vollständigen Encoder-Decoder-Architektur (vgl. Pilehvar, Camacho-Collados 2020: 88). Die Bidirektionalität ermöglicht es BERT, in allen Layern eine gemeinsame Konditionierung sowohl im linken als auch im rechten Kontext durchzuführen. Dies wird erreicht, indem das herkömmliche Ziel der Sprachmodellierung für die Vorhersage des nächsten Wortes in eine modifizierte Version geändert wird, die als Masked Language Modeling (MLM) bezeichnet wird (vgl. ebd. 88). MLM maskiert zufällig einige der Tokens aus dem Input mit dem Ziel, die ursprüngliche Vokabular-ID des maskierten Wortes nur basierend auf seinem Kontext vorherzusagen (vgl. Devlin et al. 2018: 1f.). „Unlike left-to-right language model pre-training, the MLM objective enables the representation to fuse the left and the right context, which allows […] to pretrain a deep bidirectional Transformer. In addition to the masked language model, [it] also [uses] a 'next sentence prediction' task that jointly pretrains text-pair representations" (ebd. 2).

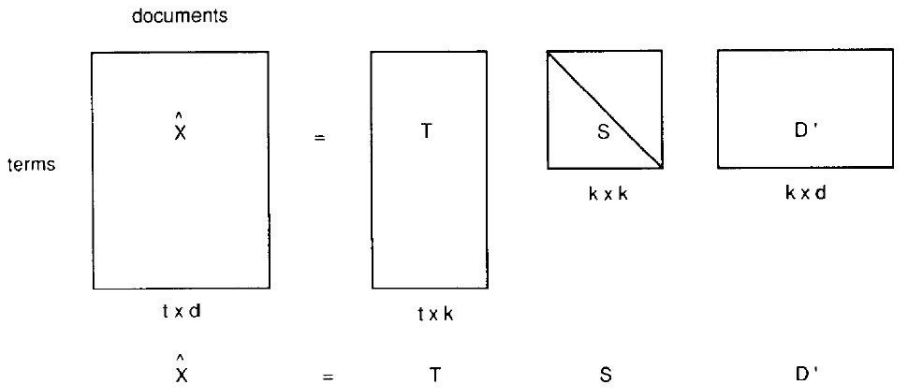
Das BERT-Framework besteht aus zwei Schritten: Pre-Training und Fine-Tuning. „During pre-training, the model is trained on unlabeled data over different pre-training tasks. For finetuning, the BERT model is first initialized with the pre-trained parameters, and all of the parameters are fine-tuned using labeled data from the downstream tasks" (ebd. 3). Eine schematische Funktionsweise von BERT kann Abbildung 4 entnommen werden (vgl. Anhang 4).

BERT ist das erste auf Fine-Tuning basierende Darstellungsmodell, das bei einer großen Reihe von Aufgaben auf Satz- und Token-Ebene eine State-of-the-art-perfomance erreicht und dabei andere aufgabenspezifische Architekturen in vielen Bereich des NLP, wie beispielweise Next Sentence Prediction, Question Answering oder Sentiment Analyse übertrifft (vgl. Devlin et al. 2018: 2ff.). BERT ist aktuell das bewährteste und deshalb auch beliebteste CWE-Modell im Bereich NLP.

9 Anhang

Anhang 1

Abbildung 1: SVD bei LSA (vgl. Deerwester et al. 1990: 401).

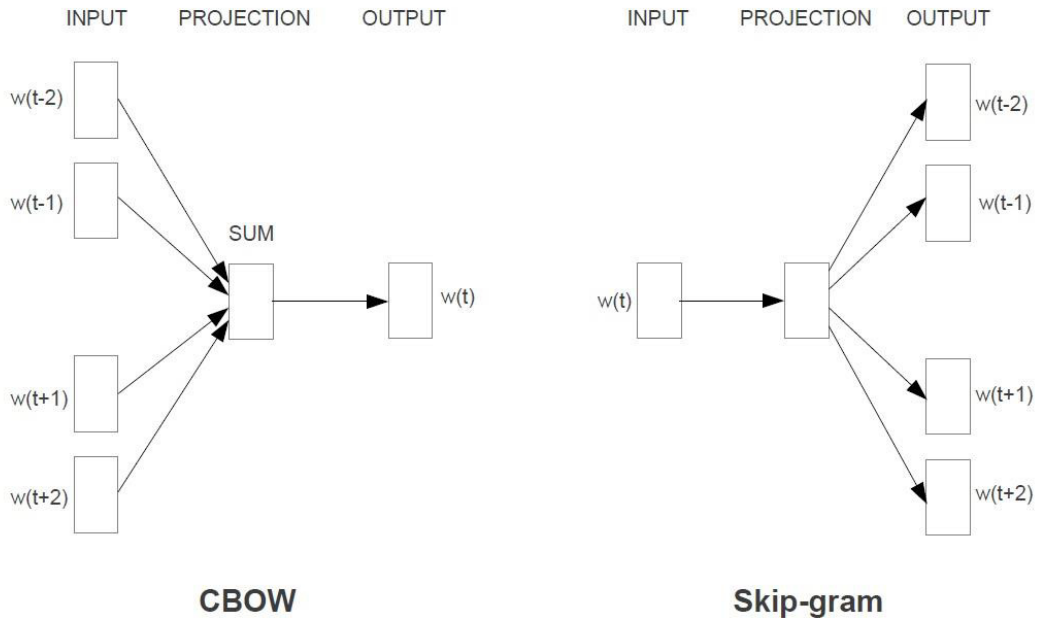


Reduced singular value decomposition of the term x document matrix, X. Where:

T has orthogonal, unit-length columns (T' T = I)
D has orthogonal, unit-length columns (D' D = I)
S is the diagonal matrix of singular values
t is the number of rows of X
d is the number of columns of X
m is the rank of X (≤ min(t,d))
k is the chosen number of dimensions in the reduced model (k ≤ m)

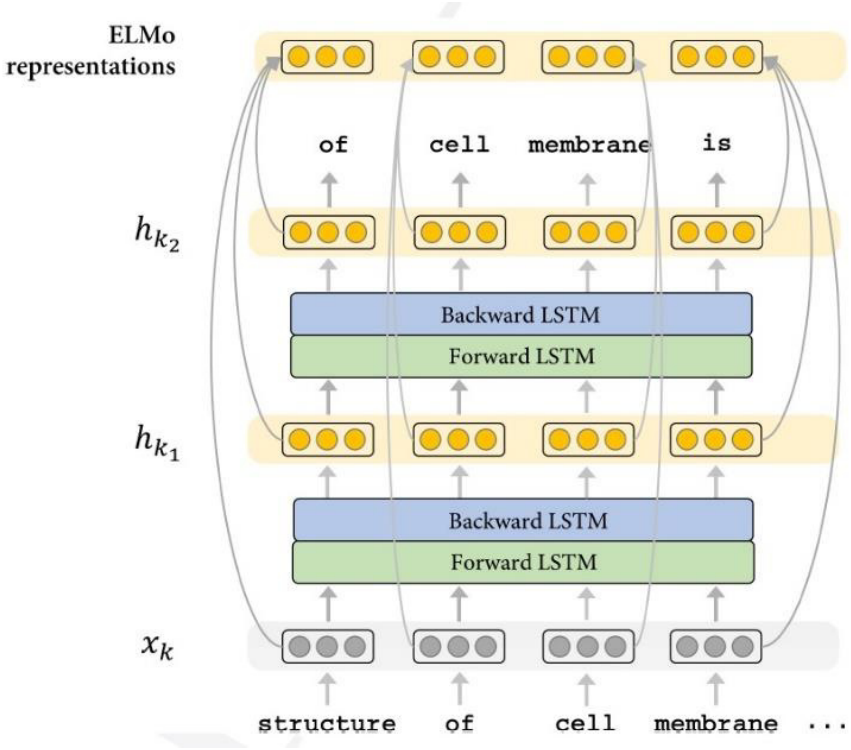
Anhang 2

Abbildung 2: Word2Vec-Modelle CBOW und SG im Vergleich (vgl. Mikolov et al. 2013: 5).



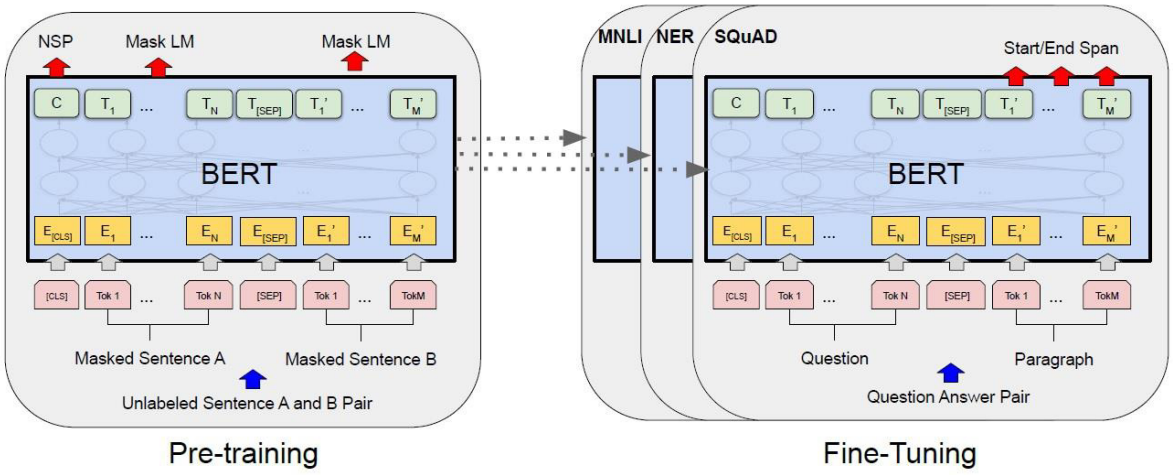
Anhang 3

Abbildung 3: ELMo (vgl. Pilehvar, Camacho-Collados 2020: 86).



Anhang 4

Abbildung 4: BERT (vgl. Devlin et al. 2018: 3).



10 Literaturverzeichnis

Alammar, Jay (2018): The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning). Online verfügbar unter http://jalammar.github.io/illustrated-bert/, zuletzt geprüft am 04.08.2020.

Deerwester, Scott; Dumais, Susan T.; Furnas, George W.; Landauer, Thomas K.; Harshman, Richard (1990): Indexing by latent semantic analysis. In: *Journal of the American Society for Information Science* (Vol. 41, Issue 6), S. 391–407. Online verfügbar unter https://doi.org/10.1002/(SICI)1097-4571(199009)41:6%3C391::AID-ASI1%3E3.0.CO;2-9, zuletzt geprüft am 26.07.2020.

Devlin, Jacob; Chang, Ming-Wei; Lee, Kenton; Toutanova, Kristina (2018): BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Online verfügbar unter https://arxiv.org/abs/1810.04805, zuletzt geprüft am 05.08.2020.

Dumais, Susan T. (2005): Latent semantic analysis. In: *Annual Review of Information Science and Technology* (Vol. 38, Issue 1), S. 188–230. Online verfügbar unter https://doi.org/10.1002/aris.1440380105, zuletzt geprüft am 25.07.2020.

Hellrich, Johannes (2019): Word Embeddings: Reliability and Semantic Change. IOS Press (Dissertations in Artificial Intelligence Ser., 347). Online verfügbar unter http://ebookcentral.proquest.com/lib/ub-wuerzburg/detail.action?docID=5899578, zuletzt geprüft am 25.07.2020.

Jurafsky, Daniel; Martin, James H. (2019): Speech and Language Processing. Chapter 6: Vector Semantics and Embeddings. Online verfügbar unter https://web.stanford.edu/~jurafsky/slp3/, zuletzt geprüft am 25.07.2020.

Landauer, Thomas K.; Foltz, Peter W.; Laham, Darrell (1998): An Introduction to Latent Semantic Analysis. Online verfügbar unter http://lsa.colorado.edu/papers/dp1.LSAintro.pdf, zuletzt geprüft am 25.07.2020.

Mikolov, Tomas; Chen, Kai; Corrado, Greg; Dean, Jeffrey (2013): Efficient Estimation of Word Representations in Vector Space. Online verfügbar unter https://arxiv.org/pdf/1301.3781.pdf, zuletzt geprüft am 27.07.2020.

Osinga, Douwe (2018): Deep Learning Cookbook. Chapter 3: Calculating Text Similarity Using Word Embeddings. O'Reilly. Online verfügbar unter https://learning.oreilly.com/library/view/deep-learning-cookbook/9781491995839/?ar, zuletzt geprüft am 20.07.2020.

Pennington, Jeffrey; Socher, Richard; Manning, Christopher D. (2014): GloVe: Global Vectors for Word Representation. Online verfügbar unter https://nlp.stanford.edu/pubs/glove.pdf, zuletzt geprüft am 28.07.2020.

Peters, Matthew E.; Neumann, Mark; Iyyer, Mohit; Gardner, Matt; Clark, Christopher; Lee, Kenton; Zettlemoyer, Luke (2018): Deep contextualized word representations. Online verfügbar unter https://arxiv.org/pdf/1802.05365.pdf, zuletzt geprüft am 29.07.2020.

Pilehvar, Mohammad Taher; Camacho-Collados, Jose (2020): Embeddings in Natural Language Processing. Theory and Advances in Vector Representation of Meaning. Morgan & Claypool Publishers. Online verfügbar unter http://josecamachocollados.com/book_embNLP_draft.pdf, zuletzt geprüft am 25.07.2020.

Switzer, Paul (1965): Vector images in document retrieval. In: Mary Elizabeth Stevens, Vincent E. Giuliano und Laurence B. Heilprin (Hg.): Statistical Association Methods For Mechanized Documentation. Washington: Symposium Proceedings, S. 163–171. Online verfügbar unter http://nvlpubs.nist.gov/nistpubs/, zuletzt geprüft am 26.07.2020.

Taylor, Josh (2019): ELMo: Contextual language embedding. Towards data science. Online verfügbar unter https://towardsdatascience.com/elmo-contextual-language-embedding-335de2268604, zuletzt geprüft am 29.07.2020.

Thomas, Alex (2020): Natural Language Processing with Spark NLP. Chapter 11: Word Embeddings. Boston: O'Reilly. Online verfügbar unter https://learning.oreilly.com/library/view/-/9781492047759/?ar, zuletzt geprüft am 20.07.2020.