CAS Big Data and Machine Learning

word2vec Schritt für Schritt

31.05.2018

Thomas Briner
thomas.briner@gmail.com
Ergon Informatik AG

Supervisor:
Dr. Simon Clematide

Institute of Computational Linguistics
University of Zurich

Universität Zürich[UZH]

Abstract

Mit der Veröffentlichung von word2vec 2013 änderte sich die Ausgangslage im Gebiet des Natural Language Processing (NLP) grundsätzlich. Mittels word2vec konnten dichte kontinuierliche *Word Embeddings* effizient erzeugt werden, die für viele Anwendungen hervorragende Eigenschaften haben. Die Resultate von word2vec übertrafen die bisherigen Ansätze bezüglich der Qualität wie auch der Performance deutlich. word2vec ist ein Framework, das zwei unterschiedliche Modelle und verschiedene Optimierungen für die Erstellung der *Word Embeddings* bietet.

Das Grundprinzip von word2vec, dass basierend auf einem Kontext die passenden Wörter vorhergesagt werden, ist einleuchtend und intuitiv. Um die Details der Realisierung zu verstehen, ist aber ein vertiefter Blick auf die Modelle notwendig. In dieser Arbeit wird die Funktionsweise von word2vec Schritt für Schritt durchleuchtet und anhand eines konkreten Beispiels greifbar gemacht.

Inhaltsverzeichnis

Abstract i

Inhaltsverzeichnis ii

1 Einführung 1

1.1 Ansätze zur Erzeugung von *Word Embeddings* . . . 2

1.2 Ziel dieser Arbeit . . . 3

1.3 Verwandte Arbeiten . . . 5

2 word2vec 6

2.1 Vereinfachtes Modell mit Single-Word-Kontext . . . 7

2.1.1 Vorwärtsberechnung (Forward Pass) . . . 9

2.1.2 Rückwärtsberechnung (Backward Pass) . . . 15

2.2 word2vec-Modelle: CBOW und Skip-Gram . . . 18

2.2.1 CBOW . . . 18

2.2.2 Skip-Gram . . . 19

2.3 Optimierungen . . . 21

2.3.1 Hierarchical Softmax . . . 21

2.3.2 Negative Sampling . . . 22

2.3.3 Subsampling von häufigen Wörtern . . . 24

2.3.4 Behandlung von festen Ausdrücken . . . 24

3 Evaluation 26

4 Zusammenfassung 29

Literaturverzeichnis 30

1 Einführung

Die Abbildung von Wörtern in eine numerische Repräsentation ist eine Voraussetzung für verschiedenste Aufgaben rund um die Verarbeitung von natürlicher Sprache. Die Bearbeitung einer Suchabfrage im Web ist ein Beispiel für eine solche Aufgabe.

Wenn ich in einer Suchmaschine nach den Begriffen „harddisk grösse notebook apple“ suche, so erwarte ich, dass als Suchresultate auch Seiten aufgelistet werden, welche dieser Abfrage sinngemäss entsprechen. Tatsächlich ist der oberste Treffer bei dieser Suche auf einer der gängigen Suchmaschinen eine Seite mit dem Titel „Interne Festplatte - Kapazität - Mac Zubehör“.[1] Offensichtlich ist die Suchmaschine in der Lage, die Ähnlichkeit der Begriffe „Harddisk“ und „Festplatte“ sowie „Grösse“ und „Kapazität“ in diesem Kontext zu erkennen.

Eine Repräsentation für Wörter bildet die *atomic representation* in Form eines *One-Hot-Encodings*. Für diese Repräsentation wird das gesamte Vokabular, d.h. alle Wörter welche im entsprechenden Korpus vorkommen, aufgelistet. Die Repräsentation besteht aus einem Vektor, welcher an der Position des entsprechenden Wortes den Wert 1 stehen hat, während an allen anderen Positionen eine 0 steht.

$$\text{Harddisk: } [0, 0, 0, \ldots, 0, 1, 0]$$

$$\text{Festplatte: } [0, 1, 0, \ldots, 0, 0, 0]$$

Einen groben Indikator für die Ähnlichkeit zweier Vektoren stellt gemäss Manning [2017] das innere Produkt der beiden Vektoren dar. Angewendet auf die Repräsentation als *One-Hot-Encoding* liefert dieser Indikator aber kein sinnvolles Resultat. Die Vektoren zweier unterschiedlicher Wörter stehen immer orthogonal zueinander und das innere Produkt ist dementsprechend immer gleich 0.

$$[0, 0, \ldots, 0, 1, 0]^{\mathrm{T}} \cdot [0, 1, 0, \ldots, 0, 0] = 0$$

Diese Repräsentation ist deshalb für die erwähnten Anwendungen nicht von Nutzen.[2]

[1] https://www.google.ch in Incognito Browser Window, Stand 01.05.2018

[2] Darüber hinaus ist sie aufgrund der Länge der Vektoren, welche der Anzahl der Wörter im gesamten Vokabular entspricht, für praktische Anwendungen schlecht geeignet.

Um Rückschlüsse über die Bedeutung eines Wortes zu erhalten, ist es notwendig, den Kontext eines Wortes zu betrachten. Dies wird durch das Zitat von J.R. Firth von 1957

> You shall know a word by the company it keeps.

auf den Punkt gebracht. Die Information über den Kontext muss in die Repräsentation einfliessen, damit sie nutzbringend eingesetzt werden kann. Die Hypothese ist dabei, dass Wörter mit einer ähnlichen Bedeutung häufig in ähnlichen Kontexten auftauchen.

Diese Betrachtung des Kontextes zu einem bestimmten Wort ist die Basis für word2vec, wie es in Mikolov u. a. [2013a] präsentiert wurde. word2vec ist ein Framework mit zwei unterschiedlichen Modellen, welches das Gebiet des *Natural Language Processing* revolutioniert hat.

1.1 Ansätze zur Erzeugung von *Word Embeddings*

Die Methoden für die Abbildung von Wörtern[3] lassen sich in zwei unterschiedliche Ansätze unterteilen: häufigkeitsbasiert (count-based) und vorhersagebasiert (prediction-based).

Die häufigkeitsbasierten Methoden waren lange die dominante Vorgehensweise für die Bildung von Wortrepräsentationen. Die Ausgangslage stellt dabei eine Kookkurrenz-Matrix dar, in welcher ersichtlich ist, wie Wörter gemeinsam im gleichen Kontext genutzt werden. Diese Matrix kann als Wort-zu-Wort-Abbildung oder auch als Wort-zu-Dokument-Matrix definiert werden. Bei der Wort-zu-Dokument-Matrix wird die Anzahl gemeinsamer Vorkommen von Wörtern innerhalb eines Dokuments gezählt und in der Matrix als globale Statistik über den ganzen Korpus aufgeführt.

Die Abbildung in einer Kookkurrenz-Matrix führt zu riesigen, schwach besetzten Matrizen. Um mit diesen Informationen effizient arbeiten zu können, wurde mit verschiedenen Methoden die Anzahl der Dimensionen reduziert, um die Daten zu verdichten, wobei möglichst wenig Information durch die Reduktion verlorengehen soll. Eine typische Vorgehensweise ist hierbei die Eigenwertzerlegung, die zu dichten Vektoren für jedes Wort führt, welche bezüglich Dimensionalität sehr viel kleiner sind. Ein erfolgreicher Vertreter dieser Kategorie ist das LSA-Verfahren (Latent Semantic Analysis), das bereits 1990 präsentiert wurde.

Die vorhersagebasierten Methoden, zu denen word2vec gehört, gehen einen anderen Weg, der im Gegensatz zur globalen statistischen Sicht auf den lokalen Kontext eines Wortes fokussiert. Dabei wird mit dem Prinzip eines *Sliding Windows* jedes Dokument schrittweise verarbeitet, wobei immer aufgrund eines aktuellen Kontextes im *Sliding Window* ein oder mehrere Zielwörter vorhergesagt werden. Als Nebenprodukt dieser Verfahren entstehen dabei *Word Embeddings*,

[3] Diese Abbildungsmethoden sind auch auf andere Sprachkonstrukte wie Sätze, Dokumente gut übertragbar.

also Repräsentationen für die verschiedenen Wörter. Sie werden aus den Gewichtsmatrizen eines neuronalen Netzes extrahiert, welche während des Trainings über den gesamten Korpus mit den bekannten Techniken von *Stochastic Gradient Descent* und *Backpropagation* optimiert wurden.

Das word2vec Framework war der Vorreiter dieser vorhersagebasierten Methoden. Mit diesem Verfahren wurden auch neue Möglichkeiten für die Anwendung der *Word Embeddings* populär, die danach in die entsprechenden Qualitätsmetriken einflossen, wie sie in Baroni u. a. [2014] angewendet und in Levy und Goldberg [2014] genauer untersucht werden. Auf die entsprechenden Ergebnisse wird in Kapitel 3 eingegangen.

Den vorhersagebasierten Methoden wird vorgeworfen, dass der Trainingsaufwand unnötig hoch ist, weil er mit der Korpusgrösse proportional wächst und darüber hinaus die globalen statistischen Informationen nicht genutzt werden, welche auf effiziente Art ermittelt werden könnten.

Diese Punkte werden im Besonderen von den Vertretern des GloVe[4] Algorithmus eingebracht. Der Kern von GloVe besteht darin, dass die bedingten Wahrscheinlichkeiten für das Vorkommen eines Wortes w_i gegeben das Vorkommen eines zweiten Wortes w_k $P(w_i|w_k)$ im Verhältnis zur Wahrscheinlichkeit $P(w_j|w_k)$ für ein anderes Wort w_j betrachtet werden. Basierend auf diesem Verhältnis wird ein Regressionsproblem formuliert, mit welchem Wortrepräsentationen gelernt werden können. GloVe nimmt für sich in Anspruch, für die Synthese der beiden Methoden zu stehen und die Nutzung der globalen Statistiken der häufigkeitsbasierten Methoden mit der Qualität der vorhersagebasierten Methoden zu verschmelzen.

1.2 Ziel dieser Arbeit

Innerhalb des CAS Kurses Big Data and Machine Learning der Universität Zürich[5] wurde im Modul Text Analytics das Thema der *Word Embeddings* behandelt und das Framework word2vec vorgestellt.

Das Grundprinzip, basierend auf einer bestimmten Eingabe eine Vorhersage für Wörter in deren Kontext zu machen, wie sie in den Dokumenten des Korpus vorkommen, ist eine einleuchtende Idee. Dass das trainierte Modell, welches in der Lage ist, solche Wörter mit einer hohen Treffsicherheit vorherzusagen, aber gar nicht das eigentliche Resultat des ganzen Prozesses ist, sondern lediglich ein Mittel, um so *Word Embeddings* zu erhalten, ist aus meiner Sicht raffiniert.

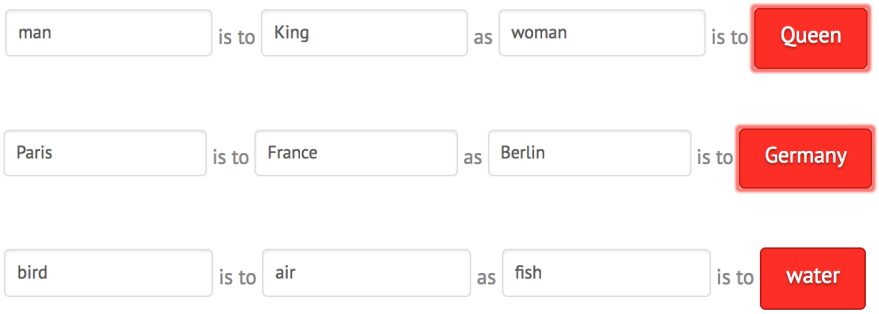
Die Ergebnisse, die sich für verschiedene Aufgaben mit diesen Repräsentationen erzielen lassen, sind beeindruckend. So können basierend auf diesen *Word Embeddings* sowohl syntaktische als

[4] Die Abkürzung GloVe steht für „Global Vector“.

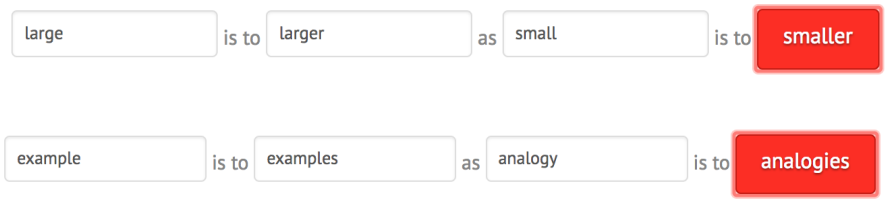
[5] http://www.ifi.uzh.ch/de/studies/cas.html

auch semantische Fragestellungen beantwortet werden. Die bekanntesten Beispiele sind dabei die Wortanalogien vom Typ „Welches Wort ist für z dasselbe wie x für y?“

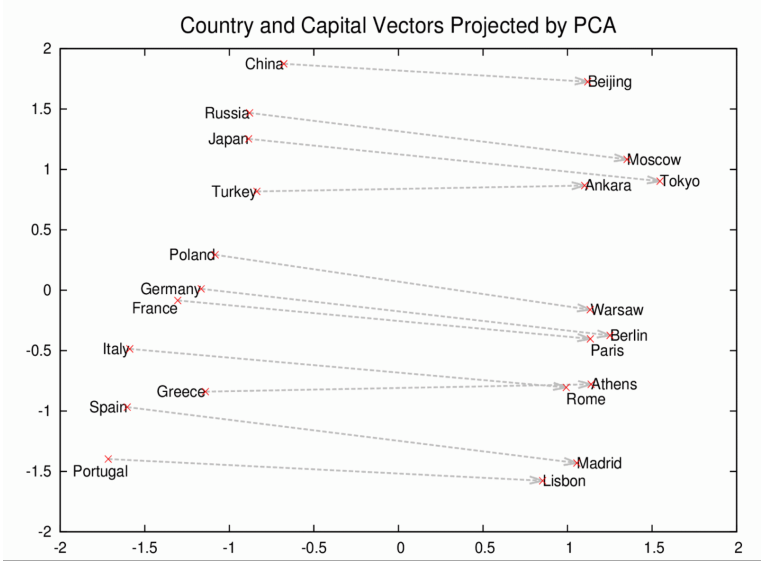
Auf verschiedenen Webseiten[6] gibt es die Möglichkeit, selber solche Analogien zu testen. Die Ergebnisse sind dabei verblüffend.



Dieses Prinzip funktioniert wie erwähnt auch für syntaktische Eigenschaften:



Basierend auf den *Word Embeddings* ist es möglich, mittels Dimensionsreduktionsverfahren 2D-Visualisierungen davon zu erzeugen.[7]



[6]Beispielsweise auf der Seite https://rare-technologies.com/word2vec-tutorial unter „Bonus App“

[7]Aus „Learning the meaning behind words“, https://opensource.googleblog.com/2013/08/learning-meaning-behind-words.html

Aus der Faszination für diese *Word Embeddings* heraus habe ich mich entschieden, einen Blick hinter die Kulissen dieses Frameworks zu werfen, um die Details zu verstehen, die ihm zugrundeliegen. Das Ziel des vorliegenden Dokuments ist es, die Funktionsweise in einer Form zu präsentieren, welche es erlaubt, basierend auf den Informationen aus den Vorlesungen zur Einführung in Machine Learning, zu Deep Learning und Text Analytics, die Funktionsweise von word2vec als Ganzes zu verstehen und anhand des konkreten Beispiels Schritt für Schritt die einzelnen Puzzleteile zusammenfügen zu können.

Die einzelnen Schritte, wie sie in diesem Dokument präsentiert werden, stehen als Berechnungen in einem iPython Notebook[8] zur Verfügung. Dafür wird die Library pytorch[9] verwendet. Mit diesem Notebook ist es möglich, interaktiv Schritt für Schritt die Berechnungen von word2vec nachzuvollziehen.

1.3 Verwandte Arbeiten

Aufgrund der Tatsache, dass die initialen Publikationen zu word2vec sehr knapp gehalten und verschiedene Bereiche nur kurz erwähnt wurden, gibt es inzwischen eine erhebliche Menge von Dokumenten, die die verschiedenen Aspekte von word2vec ausführlicher beleuchten.

Das Thema *Word Embeddings* und word2vec wird in verschiedenen Lehrbüchern behandelt, so beispielsweise in Goldberg [2017], in welchem die beiden Modelle von word2vec wie auch zwei Optimierungen behandelt werden. Das entsprechende Kapitel gibt einen Überblick, wobei das Verständnis für die Grundprinzipien vorausgesetzt wird.

In Rong [2014] wird das word2vec Framework von Grund auf analysiert, indem von einer Vereinfachung ausgegangen wird, die als Eingabekontext wie auch als Ausgabe ein einzelnes Wort benutzt. Basierend auf diesem Aufbau werden die Schritte konzeptionell erklärt und mathematisch hergeleitet und anschliessend die Resultate auf die Struktur übertragen, wie sie in word2vec effektiv benutzt wird. Dieser Aufbau hat mich so überzeugt, dass ich für diese Arbeit ebenfalls den gleichen Weg gewählt habe.

Die Methode des *Negative Sampling* wird in Goldberg und Levy [2014] detailliert hergeleitet, wobei besonders interessant ist, dass die Frage, weshalb dieses Vorgehen zu guten *Word Embeddings* führt, auch für die Autoren unklar bleibt.

Basierend auf den erwähnten Publikationen bietet dieses Dokument eine Einführung in das Framework word2vec, welche nahtlos auf dem vermittelten Stoff des CAS aufsetzt.

[8] Das Notebook ist als Jupyter Notebook im Repository unter https://github.com/thomasbriner/word2vec-Schritt-fuer-Schritt zugänglich. Mit dem entsprechenden Link im Readme kann es direkt als interaktives Dokument geöffnet werden.

[9] https://pytorch.org/

2 word2vec

Das Ziel von word2vec ist es, dichte *Word Embeddings* zu berechnen, so dass die Ähnlichkeit von Wörtern auch in den Repräsentationen erhalten bleibt. Der Schlüssel zu dieser Ähnlichkeit liegt in der Nutzung des Kontexts, in welchem ein Wort vorkommt.

word2vec benützt ein flaches neuronales Netz, das aus einem Input Layer, einem einzigen Hidden Layer und einem Output Layer besteht. Dieses Netzwerk wird mit den verschiedenen Dokumenten des Korpus trainiert. Die Grundidee ist dabei, dass das Netzwerk aufgrund eines bestimmten Kontextes vorherzusagen versucht, was das gesuchte Wort bzw. die gesuchten Wörter im Bezug auf diesen Kontext sein könnten.

Diese Problemmodellierung ist aber nur eine Surrogat-Aufgabe, da am Schluss nicht das gesamte gelernte Modell von Interesse ist, sondern nur ein Teil der gelernten Gewichte innerhalb des neuronalen Netzes. Diese Gewichte bilden direkt die *Word Embeddings*.

word2vec beinhaltet zwei unterschiedliche Modelle, mit welchen diese Gewichte und damit die *Word Embeddings* gelernt werden können, nämlich CBOW[10] und Skip-Gram[11].

Das Prinzip, dass basierend auf bestimmten Eingabewörtern eine Menge von Zielwörtern vorhergesagt wird, ist für beide Modelle gültig. Es variert lediglich, wieviele Wörter als Kontext benutzt bzw. wieviele als Ziel vorausgesagt werden.

[10] für **c**ontinuous **b**ag **o**f **w**ords

[11] Der Name stammt von der Idee, N-Gramme mit Lücken darin zu verwenden, wie in https://en.wikipedia.org/wiki/N-gram#Skip-gram beschrieben

2.1 Vereinfachtes Modell mit Single-Word-Kontext

Um das Grundprinzip Schritt für Schritt zu besprechen, habe ich eine Vereinfachung gewählt. Es wird lediglich ein einziges Wort als Kontext verwendet und basierend darauf soll ein einziges - nämlich das Folgewort - vorausgesagt werden.

Die Prinzipien des Algorithmus sind analog zu den tatsächlich verwendeten Modellen CBOW und Skip-Gram. Die Differenzen liegen ausschliesslich in der Wahl des Kontext- bzw. Zielfensters. Die Übertragung der Mechanismen auf CBOW und Skip-Gram wird in den Kapiteln 2.2.1 und 2.2.2 präsentiert.

Die Funktionsweise wird an einem konkreten Beispiel illustriert. Schritt für Schritt werden an diesem reduzierten Beispiel die Prinzipien wie auch die effektiven Berechnungen aufgezeigt. Das Beispiel wurde minimal gehalten, um die Verständlichkeit und Übersichtlichkeit zu erhöhen, ohne aber an den Mechanismen Veränderungen vorzunehmen.

Der Korpus für das Beispiel besteht aus zwei Dokumenten, welche wiederum je einen einzigen Satz enthalten.

$$\begin{aligned} \text{Dokument 1} &= \text{'unsere katze heisst mimo'} \\ \text{Dokument 2} &= \text{'die katze klettert auf den baum'} \\ \text{Korpus} &= \{\text{Dokument 1}, \text{Dokument 2}\} \end{aligned}$$

Das Vokabular wird aus allen Dokumenten des Korpus aufgebaut. Dabei werden die Wortformen so übernommen, wie sie in den Dokumenten auftreten.

$$\begin{aligned} \text{Vokabular} &= [\text{'unsere', 'katze', 'heisst', 'mimo', 'die', 'klettert', 'auf', 'den', 'baum'}\ ] \\ V &= \text{Länge des Vokabulars} = 9 \end{aligned}$$

Als Kontext wird in dieser Vereinfachung nur ein einziges Wort verwendet. Basierend auf diesem Kontextwort w_c[12] wird ein einziges Zielwort w_t[13] vorhergesagt.

Kontext-wort w_c	Zielwort w_t

Das *Sliding Window* wird Schritt für Schritt über jedes Dokument geschoben und der Algorithmus auf das entsprechende Kontextwort und Zielwort angewendet.

[12] c für „Context“.
[13] t für „Target“.

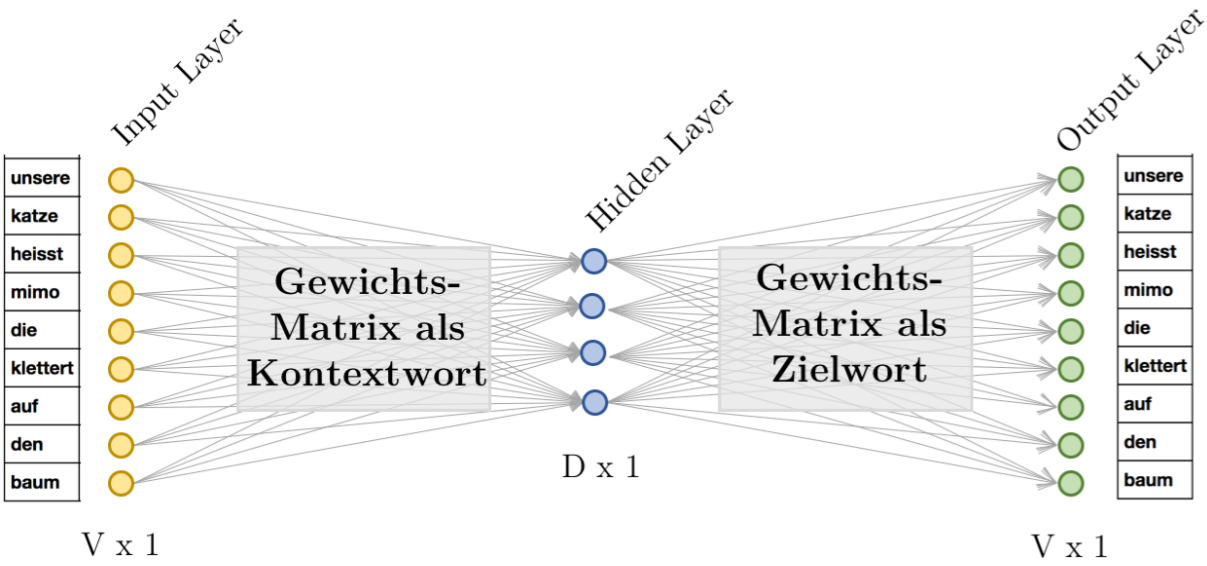
Position 1:	die	katze	klettert	auf	den	baum
Position 2:	die	katze	klettert	auf	den	baum

...

Das Beispiel wird für die Position 2 durchgerechnet, an welcher als Kontextwort 'katze' und als Zielwort 'klettert' steht.

Die Dimensionalität der *Word Embeddings* ist ein Modellparameter und wird in der Realität typischerweise in der Grössenordnung von 50 - 1000 gewählt. Für dieses Beispiel wird mit der Dimensionalität $D = 4$ gearbeitet, um die Darstellungen wie auch die Berechnungen übersichtlich und leicht nachvollziehbar zu halten.

Bei word2vec wird für jedes Wort nicht nur mit einer Repräsentation gearbeitet, sondern mit zwei: Einer für die Wörter als Kontextwort und mit einer zweiten für die Wörter als Zielwort. Damit wird die Berechnung erheblich vereinfacht und auch das Problem gelöst, dass ansonsten jedes Wort zu sich selber am ähnlichsten ist, was aber nicht der Realität entspricht, da nur in seltenen Fällen ein Wort direkt zweimal hintereinander in einem Text vorkommt. Der grössere Teil der Anwendungen benutzt dabei die *Word Embeddings* als Kontextwort aus der Gewichtsmatrix vom Input- zum Hidden Layer.



2.1.1 Vorwärtsberechnung (Forward Pass)

Bei der Vorwärtsberechnung wird für die aktuelle Position des *Sliding Windows* der Vorhersagefehler (Prediction Error) berechnet. Dabei sind folgende Operationen notwendig:

<table>
<tr>
<td>Kontextwort als One-Hot-Encoding</td>
<td>Das Kontextwort w_c an der aktuellen Position des Sliding Windows wird als One-Hot-Encoding codiert.
$$x_i = \begin{cases} 1 & \text{für Kontextwort} \\ & \text{an dieser Position} \\ 0 & \text{überall sonst} \end{cases}$$
Der Vektor $\mathbf{x}$ hat die Dimension $V \times 1$</td>
<td>$\mathbf{x}$
<table><tr><th></th><th>val</th></tr><tr><td>unsere</td><td>0</td></tr><tr><td>katze</td><td>1</td></tr><tr><td>heisst</td><td>0</td></tr><tr><td>mimo</td><td>0</td></tr><tr><td>die</td><td>0</td></tr><tr><td>klettert</td><td>0</td></tr><tr><td>auf</td><td>0</td></tr><tr><td>den</td><td>0</td></tr><tr><td>baum</td><td>0</td></tr></table>V x 1</td>
</tr>
</table>

<table>
<tr>
<td>Matrix mit Word Embeddings für Kontextwort</td>
<td>$\mathbf{W}$ ist die Matrix mit Word Embeddings für Kontextwörter und hat die Dimension $V \times D$. Die Matrix $\mathbf{W}$ wird mit Zufallswerten initialisiert. Jede Zeile $\mathbf{v}_i$ entspricht dem Word Embedding für das i-te Wort des Vokabulars.</td>
<td>$\mathbf{W}$
<table>
<tr><th></th><th>dim1</th><th>dim2</th><th>dim3</th><th>dim4</th></tr>
<tr><td>unsere</td><td>0.88</td><td>0.92</td><td>0.38</td><td>0.96</td></tr>
<tr><td>katze</td><td>0.39</td><td>0.60</td><td>0.26</td><td>0.79</td></tr>
<tr><td>heisst</td><td>0.94</td><td>0.13</td><td>0.93</td><td>0.59</td></tr>
<tr><td>mimo</td><td>0.87</td><td>0.57</td><td>0.74</td><td>0.43</td></tr>
<tr><td>die</td><td>0.89</td><td>0.57</td><td>0.27</td><td>0.63</td></tr>
<tr><td>klettert</td><td>0.27</td><td>0.44</td><td>0.30</td><td>0.83</td></tr>
<tr><td>auf</td><td>0.11</td><td>0.27</td><td>0.36</td><td>0.20</td></tr>
<tr><td>den</td><td>0.55</td><td>0.01</td><td>0.95</td><td>0.08</td></tr>
<tr><td>baum</td><td>0.89</td><td>0.58</td><td>0.34</td><td>0.81</td></tr>
</table>
V x D</td>
</tr>
<tr>
<td>Berechnung Hidden Layer</td>
<td>Die Multiplikation $\mathbf{h} = \mathbf{W}^{\mathrm{T}}\mathbf{x}$ der Gewichtsmatrix mit dem One-Hot-Encoding des Kontextworts entspricht der Selektion der entsprechenden Zeile für das Kontextwort aus der Gewichtsmatrix. Der Hidden Layer enthält also genau die Gewichte für das jeweilige Kontextwort.</td>
<td>$\mathbf{W}^{\mathrm{T}} \cdot \mathbf{x} := \mathbf{h}$
<table>
<tr><th>x</th><th>W</th><th>h</th></tr>
<tr><td>
<table>
<tr><th></th><th>val</th></tr>
<tr><td>unsere</td><td>0</td></tr>
<tr><td>katze</td><td>1</td></tr>
<tr><td>heisst</td><td>0</td></tr>
<tr><td>mimo</td><td>0</td></tr>
<tr><td>die</td><td>0</td></tr>
<tr><td>klettert</td><td>0</td></tr>
<tr><td>auf</td><td>0</td></tr>
<tr><td>den</td><td>0</td></tr>
<tr><td>baum</td><td>0</td></tr>
</table>
V x 1</td>
<td>
<table>
<tr><th></th><th>dim1</th><th>dim2</th><th>dim3</th><th>dim4</th></tr>
<tr><td>unsere</td><td>0.88</td><td>0.92</td><td>0.38</td><td>0.96</td></tr>
<tr><td>katze</td><td>0.39</td><td>0.60</td><td>0.26</td><td>0.79</td></tr>
<tr><td>heisst</td><td>0.94</td><td>0.13</td><td>0.93</td><td>0.59</td></tr>
<tr><td>mimo</td><td>0.87</td><td>0.57</td><td>0.74</td><td>0.43</td></tr>
<tr><td>die</td><td>0.89</td><td>0.57</td><td>0.27</td><td>0.63</td></tr>
<tr><td>klettert</td><td>0.27</td><td>0.44</td><td>0.30</td><td>0.83</td></tr>
<tr><td>auf</td><td>0.11</td><td>0.27</td><td>0.36</td><td>0.20</td></tr>
<tr><td>den</td><td>0.55</td><td>0.01</td><td>0.95</td><td>0.08</td></tr>
<tr><td>baum</td><td>0.89</td><td>0.58</td><td>0.34</td><td>0.81</td></tr>
</table>
V x D</td>
<td>
<table>
<tr><th></th><th>val</th></tr>
<tr><td>dim1</td><td>0.39</td></tr>
<tr><td>dim2</td><td>0.60</td></tr>
<tr><td>dim3</td><td>0.26</td></tr>
<tr><td>dim4</td><td>0.79</td></tr>
</table>
D x 1</td>
</tr>
</table>
</td>
</tr>
</table>

<table>
<tr>
<td>Matrix mit Word Embeddings für das Zielwort</td>
<td>Die Matrix W’ beinhaltet die Word Embeddings für alle Wörter als mögliche Zielwörter.
Sie hat die Dimension $D \times V$ und wird ebenfalls mit Zufallszahlen initialisiert.</td>
<td>

W‘

	unsere	katze	heisst	mimo	die	klettert	auf	den	baum
dim1	0.45	0.20	0.92	0.35	0.15	0.09	0.59	0.07	0.75
dim2	0.63	0.94	0.13	0.52	0.53	0.54	0.71	0.43	0.28
dim3	0.84	0.16	0.11	0.73	0.32	0.89	0.13	0.84	0.56
dim4	0.41	0.86	0.28	0.58	0.33	0.68	0.70	0.92	0.23

D x V
</td>
</tr>
<tr>
<td>Berechnung Ähnlichkeit</td>
<td>Um eine Vorhersage für das Zielwort zu machen, wird die Gewichtsmatrix für das Zielwort W’ mit dem Vektor h multipliziert:
$\mathbf{u} = \mathbf{W'}^{\mathrm{T}} \times \mathbf{h} = \mathbf{W'}^{\mathrm{T}} \cdot (\mathbf{W}^{\mathrm{T}} \cdot \mathbf{x})$
So wird mittels des Punkt-Operators die Ähnlichkeit des Embeddings des Kontextwortes mit jedem möglichem Zielwort berechnet.
Der Vektor u ist nicht normiert und stellt deshalb noch keine Wahrscheinlichkeitsverteilung dar.</td>
<td>

$\mathbf{W'}^{\mathrm{T}} \cdot \mathbf{h} = \mathbf{W'}^{\mathrm{T}} \cdot (\mathbf{W}^{\mathrm{T}} \cdot \mathbf{x}) := \mathbf{u}$

h **W‘** **u**

	val
dim1	0.39
dim2	0.60
dim3	0.26
dim4	0.79

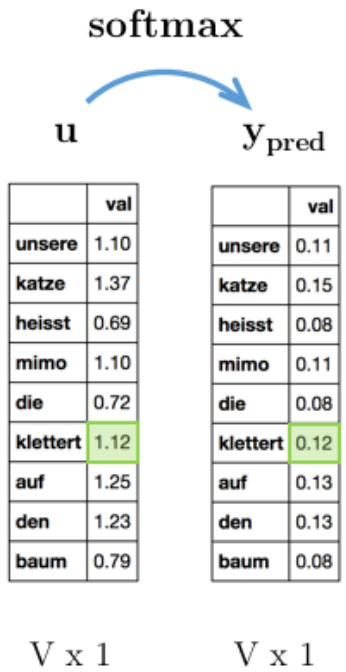
D x 1

	unsere	katze	heisst	mimo	die	klettert	auf	den	baum
dim1	0.45	0.20	0.92	0.35	0.15	0.09	0.59	0.07	0.75
dim2	0.63	0.94	0.13	0.52	0.53	0.54	0.71	0.43	0.28
dim3	0.84	0.16	0.11	0.73	0.32	0.89	0.13	0.84	0.56
dim4	0.41	0.86	0.28	0.58	0.33	0.68	0.70	0.92	0.23

D x V

	val
unsere	1.10
katze	1.37
heisst	0.69
mimo	1.10
die	0.72
klettert	1.12
auf	1.25
den	1.23
baum	0.79

V x 1
</td>
</tr>
</table>

<table>
<tr>
<td>Anwenden der Softmax-Funktion für die Berechnung des Vorhersage-Vektors $\mathbf{y}_{pred}$</td>
<td>Um die Ergebnisse der Ähnlichkeitsberechnung im Vektor $\mathbf{u}$ vergleichbar zu machen, wird die Softmax-Funktion darauf angewendet.
Jedes Element des berechneten Vektors y_{pred_i} zeigt die Wahrscheinlichkeit für das i-te Element, mit welcher der Algorithmus basierend auf dem bekannten Kontext vorhersagt, dass dieses Wort das Zielwort ist.
$$p(w_i|w_c) = y_{pred_i} = \frac{e^{u_i}}{\sum_{k=1}^{V} e^{u_j}}$$
Die Elemente $\mathbf{y}_{pred}$ summieren zu 1.
$$\sum_{i=1}^{V} y_{pred_i} = 1$$</td>
<td> 
</td>
</tr>
</table>

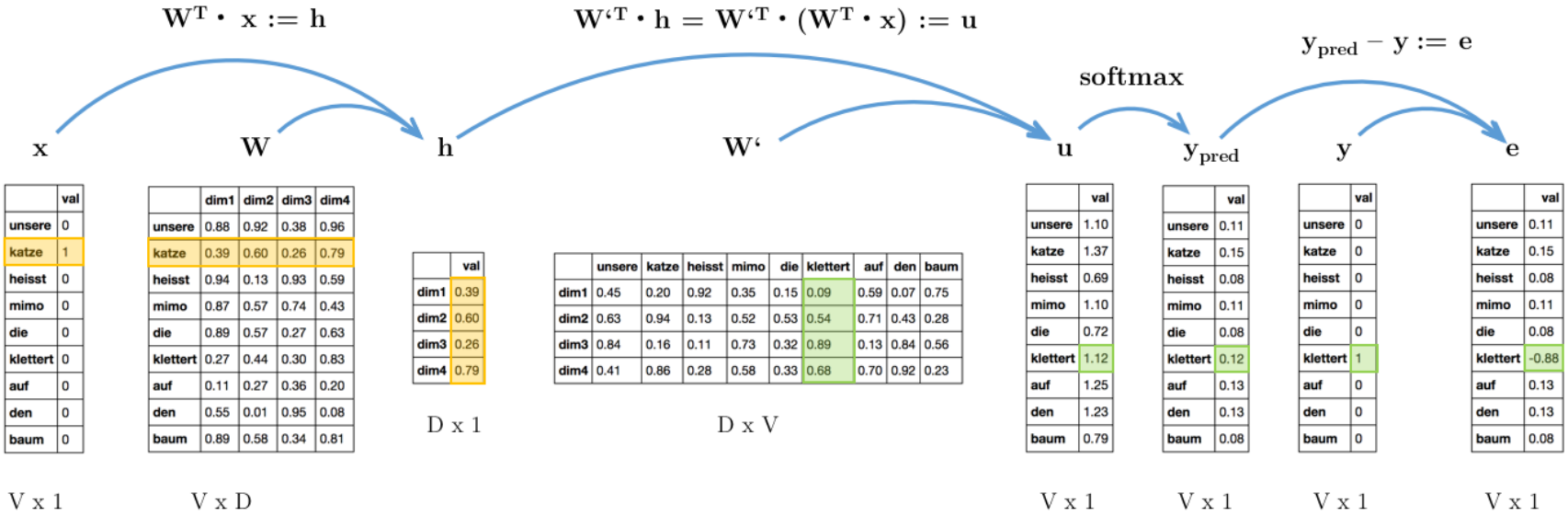
<table>
<tr>
<td>Zielwort als One-Hot-Encoding y</td>
<td>Der Vektor y ist das One-Hot-Encoding für das Zielwort. Es wird für die Berechnung des Fehlervektors benötigt.</td>
<td>$\mathbf{y}$
<table>
<tr><th></th><th>val</th></tr>
<tr><td>unsere</td><td>0</td></tr>
<tr><td>katze</td><td>0</td></tr>
<tr><td>heisst</td><td>0</td></tr>
<tr><td>mimo</td><td>0</td></tr>
<tr><td>die</td><td>0</td></tr>
<tr><td>klettert</td><td>1</td></tr>
<tr><td>auf</td><td>0</td></tr>
<tr><td>den</td><td>0</td></tr>
<tr><td>baum</td><td>0</td></tr>
</table>
V x 1</td>
</tr>
<tr>
<td>Fehlervektor e</td>
<td>Der Fehlervektor berechnet sich aus der Differenz zwischen der Vorhersage $\mathbf{y}_{pred}$ und dem tatsächlichen Zielwort y als One-Hot-Encoding.</td>
<td>$\mathbf{y}_{\text{pred}} - \mathbf{y} := \mathbf{e}$
$\mathbf{y}_{\text{pred}}$
<table>
<tr><th></th><th>val</th></tr>
<tr><td>unsere</td><td>0.11</td></tr>
<tr><td>katze</td><td>0.15</td></tr>
<tr><td>heisst</td><td>0.08</td></tr>
<tr><td>mimo</td><td>0.11</td></tr>
<tr><td>die</td><td>0.08</td></tr>
<tr><td>klettert</td><td>0.12</td></tr>
<tr><td>auf</td><td>0.13</td></tr>
<tr><td>den</td><td>0.13</td></tr>
<tr><td>baum</td><td>0.08</td></tr>
</table>
V x 1
$\mathbf{y}$
<table>
<tr><th></th><th>val</th></tr>
<tr><td>unsere</td><td>0</td></tr>
<tr><td>katze</td><td>0</td></tr>
<tr><td>heisst</td><td>0</td></tr>
<tr><td>mimo</td><td>0</td></tr>
<tr><td>die</td><td>0</td></tr>
<tr><td>klettert</td><td>1</td></tr>
<tr><td>auf</td><td>0</td></tr>
<tr><td>den</td><td>0</td></tr>
<tr><td>baum</td><td>0</td></tr>
</table>
V x 1
$\mathbf{e}$
<table>
<tr><th></th><th>val</th></tr>
<tr><td>unsere</td><td>0.11</td></tr>
<tr><td>katze</td><td>0.15</td></tr>
<tr><td>heisst</td><td>0.08</td></tr>
<tr><td>mimo</td><td>0.11</td></tr>
<tr><td>die</td><td>0.08</td></tr>
<tr><td>klettert</td><td>-0.88</td></tr>
<tr><td>auf</td><td>0.13</td></tr>
<tr><td>den</td><td>0.13</td></tr>
<tr><td>baum</td><td>0.08</td></tr>
</table>
V x 1</td>
</tr>
</table>

Übersicht Vorwärtsberechnung

Das *Sliding Windows* steht an Position 2.

		Kontext-wort w_c	Zielwort w_t			
Position 2:	die	katze	klettert	auf	den	baum

Hier sind alle Schritte der Vorwärtsberechnung im Zusammenhang dargestellt.



	val
unsere	0
katze	1
heisst	0
mimo	0
die	0
klettert	0
auf	0
den	0
baum	0

	dim1	dim2	dim3	dim4
unsere	0.88	0.92	0.38	0.96
katze	0.39	0.60	0.26	0.79
heisst	0.94	0.13	0.93	0.59
mimo	0.87	0.57	0.74	0.43
die	0.89	0.57	0.27	0.63
klettert	0.27	0.44	0.30	0.83
auf	0.11	0.27	0.36	0.20
den	0.55	0.01	0.95	0.08
baum	0.89	0.58	0.34	0.81

	val
dim1	0.39
dim2	0.60
dim3	0.26
dim4	0.79

	unsere	katze	heisst	mimo	die	klettert	auf	den	baum
dim1	0.45	0.20	0.92	0.35	0.15	0.09	0.59	0.07	0.75
dim2	0.63	0.94	0.13	0.52	0.53	0.54	0.71	0.43	0.28
dim3	0.84	0.16	0.11	0.73	0.32	0.89	0.13	0.84	0.56
dim4	0.41	0.86	0.28	0.58	0.33	0.68	0.70	0.92	0.23

	val
unsere	1.10
katze	1.37
heisst	0.69
mimo	1.10
die	0.72
klettert	1.12
auf	1.25
den	1.23
baum	0.79

	val
unsere	0.11
katze	0.15
heisst	0.08
mimo	0.11
die	0.08
klettert	0.12
auf	0.13
den	0.13
baum	0.08

	val
unsere	0
katze	0
heisst	0
mimo	0
die	0
klettert	1
auf	0
den	0
baum	0

	val
unsere	0.11
katze	0.15
heisst	0.08
mimo	0.11
die	0.08
klettert	-0.88
auf	0.13
den	0.13
baum	0.08

2.1.2 Rückwärtsberechnung (Backward Pass)

Das Ziel des Trainings ist es, die vorausgesagte Wahrscheinlichkeit für das tatsächliche Zielwort zu optimieren. Sei $\mathbf{v}_{w_i}$ die i-te Zeile der Matrix $\mathbf{W}$, in welcher das *Embedding* für Wort w_i als Kontextwort steht und $\mathbf{v}'_{w_j}$ die j-te Spalte der Matrix **W'**, in welcher das *Embedding* für Wort w_j als Zielwort steht. Die Wahrscheinlichkeit, dass ein bestimmtes Wort i als Zielwort gewählt wird gegeben das aktuelle Kontextwort w_c, lässt sich damit formulieren als

$$p(w_i|w_c) = y_{pred_i} = \frac{e^{u_i}}{\sum_{k=1}^{V} e^{u_k}} = \frac{e^{\mathbf{v}'^{\mathrm{T}}_{w_i}\mathbf{v}_{w_c}}}{\sum_{k=1}^{V} e^{\mathbf{v}'^{\mathrm{T}}_{w_k}\mathbf{v}_{w_c}}}$$

Die Loss-Funktion unter Anwendung der Negative-Log-Likelihood-Methode heisst

$$\begin{aligned}\mathcal{L} &= -\log p(w_t|w_c) = -\log y_{pred_{w_t}} = -\log \frac{e^{u_{w_t}}}{\sum_{k=1}^{V} e^{u_k}} \\ &= -\log e^{u_{w_t}} + \log \sum_{k=1}^{V} e^{u_k} = -u_{w_t} + \log \sum_{k=1}^{V} e^{u_k} = \mathbf{v}'^{\mathrm{T}}_{w_t}\mathbf{v}_{w_c} - \log \sum_{k=1}^{V} e^{\mathbf{v}'^{\mathrm{T}}_{w_k}\mathbf{v}_{w_c}}\end{aligned}$$

Der aktuelle Loss im obigen Beispiel ist also

$$\begin{aligned}\mathcal{L} &= -\log p(\text{"}klettert\text{"}katze\text{"}) = -u_{klettert} + \log \sum_{k=1}^{V} e^{u_k} \\ &= -1.12 + \log(1.10 + 1.37 + 0.69 + 1.10 + 0.72 + 1.12 + 1.25 + 1.23 + 0.79) = 1.11\end{aligned}$$

$\mathcal{L}$ ist abhängig von $\mathbf{u}$, welches wiederum abhängig von den beiden Gewichtsmatrizen $\mathbf{W}$ und $\mathbf{W}'$ ist.

Für die Rückwärtspropagierung (Backpropagation) werden mittels *Gradient Descent* die Gewichte von $\mathbf{W}$ und $\mathbf{W}'$ angepasst, um die Loss-Funktion zu minimieren. Die Herleitung der Ableitungen von $\frac{\partial \mathcal{L}}{\partial \mathbf{W}}$ und $\frac{\partial \mathcal{L}}{\partial \mathbf{W}'}$ wird in Rong [2014] detailliert beschrieben. Die Gleichung für die Anpassung der Gewichte von $\mathbf{W}'$ lautet

$$\mathbf{v}'^{(new)}_{w_i} = \mathbf{v}'^{(old)}_{w_i} - \eta \cdot e_i \cdot \mathbf{h} \qquad \text{für } i = 1, 2, \ldots, V$$

Falls das Wort i also nicht das gesuchte Zielwort war, ist e_i positiv. In diesem Fall wird abhängig von der Schrittweite η ein Bruchteil von $\mathbf{h}$ vom Vektor $\mathbf{v}'_{w_i}$ subtrahiert und die Distanz zwischen den beiden *Embeddings* $\mathbf{v}'_{w_i}$ und $\mathbf{h} = \mathbf{v}^{\mathrm{T}}_{w_c}$ vergrössert sich. Falls das Wort i aber das gesuchte Zielwort war, ist e_i negativ. Damit nähert sich $\mathbf{v}'^{(new)}_{w_i}$ an $\mathbf{v}_{w_c}$ an und die beiden *Embeddings* von Kontextwort und Zielwort werden 'ähnlicher' [14].

[14] Mit Ähnlichkeit ist hier das innere Produkt der beiden Vektoren gemeint.

Für das konkrete Beispiel sehen die Anpassungen an W' folgendermassen aus:

W‘ (old)

	unsere	katze	heisst	mimo	die	klettert	auf	den	baum
dim1	0.45	0.20	0.92	0.35	0.15	0.09	0.59	0.07	0.75
dim2	0.63	0.94	0.13	0.52	0.53	0.54	0.71	0.43	0.28
dim3	0.84	0.16	0.11	0.73	0.32	0.89	0.13	0.84	0.56
dim4	0.41	0.86	0.28	0.58	0.33	0.68	0.70	0.92	0.23

W‘ (new)

	unsere	katze	heisst	mimo	die	klettert	auf	den	baum
dim1	0.45	0.18	0.92	0.34	0.14	0.15	0.58	0.06	0.74
dim2	0.61	0.92	0.12	0.51	0.52	0.64	0.69	0.41	0.27
dim3	0.83	0.15	0.10	0.72	0.32	0.93	0.12	0.83	0.56
dim4	0.40	0.84	0.27	0.57	0.32	0.82	0.68	0.90	0.22

W‘ delta

	unsere	katze	heisst	mimo	die	klettert	auf	den	baum
dim1	-0.01	-0.01	-0.01	-0.01	-0.01	0.07	-0.01	-0.01	-0.01
dim2	-0.01	-0.02	-0.01	-0.01	-0.01	0.11	-0.02	-0.02	-0.01
dim3	-0.01	-0.01	-0.00	-0.01	-0.00	0.05	-0.01	-0.01	-0.00
dim4	-0.02	-0.02	-0.01	-0.02	-0.01	0.14	-0.02	-0.02	-0.01

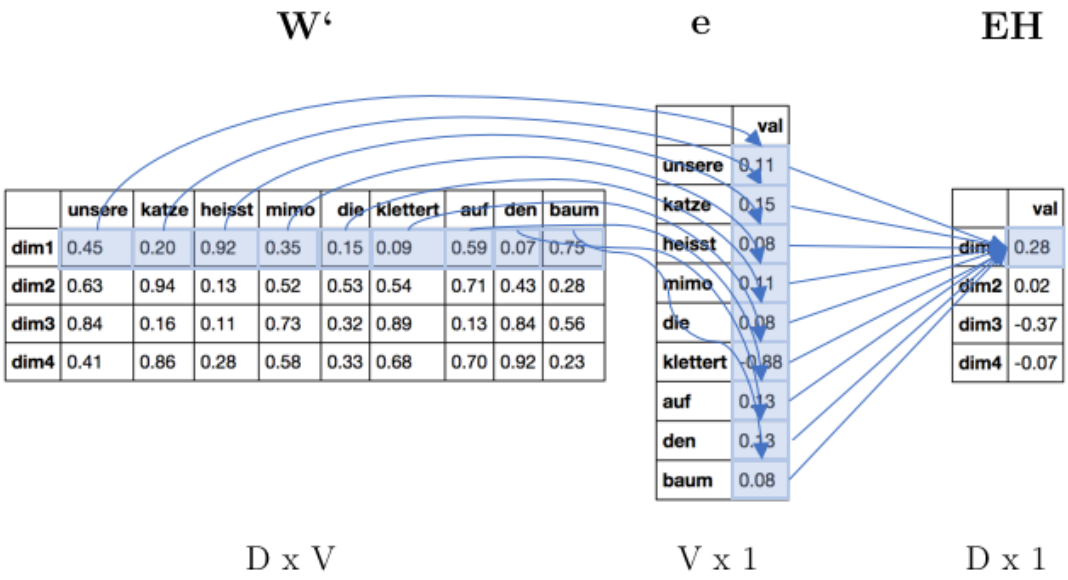
Es werden sämtliche Werte der Matrix $\mathbf{W}'$ angepasst, da auch jeder Wert auf die Berechnung von $\mathbf{y}_{pred}$ Einfluss hat. Am stärksten werden die Werte für das effektive Zielwort 'klettert' verändert, da der Fehler an dieser Position mit Abstand am grössten ist.

Die Gleichung für die Anpassung der Gewichte von $\mathbf{W}$ lautet

$$\mathbf{v}_{w_c}^{(new)} = \mathbf{v}_{w_c}^{(old)} - \eta \mathbf{EH}^{\mathrm{T}}$$

wobei $\mathbf{EH}$ die Summe der Gewichte aller Zielwörter im Vokabular ist, gewichtet mit ihrem Prediction Fehler $e_j = y_{pred_j} - y_j$

$$\mathbf{EH}_i = \sum_{j=1}^{V} e_j \times \mathbf{v}'_{w_{ij}}$$



Diese Anpassung kann als Korrektur des *Word Embeddings* des aktuellen Kontextworts interpretiert werden, in welche von jedem möglichen Zielwort ein Bruchteil seines *Embeddings* als Zielwort einfliesst.

Je grösser der Fehler in der Vorhersage eines bestimmten Wortes ist, desto stärker fliesst sein *Embeddings* als Zielwort ein. Je nachdem, ob es sich dabei um das effektive Zielwort handelt, ist die Korrektur negativ oder positiv.

Falls es sich bei der Vorhersage für ein Zielwort um ein falsches handelt, d.h. die Wahrscheinlichkeit wurde überschätzt, so ist der Fehler positiv und führt dazu, dass die Ähnlichkeit zwischen den beiden *Embeddings* reduziert wird. Handelt es sich dabei um das aktuelle Zielwort, so ist der Fehler $<= 0$ und dies führt dazu, dass das Kontext-Embedding des aktuellen Eingabewortes sich an das Ziel-Embedding des Zielworts annähert.

Die Anwendung dieses Schritts auf das Beispiel zeigt, dass nur das *Embedding* für das konkrete Kontextwort beeinflusst wird. Das macht Sinn, da die übrigen Gewichte gar nicht in die Berechnungen eingeflossen sind, da durch die Multiplikation mit dem *One-Hot-Encoding* des Kontextwortes diese Zeile als einzige selektiert wurde.

$\mathbf{W}^{(\text{old})}$

	dim1	dim2	dim3	dim4
unsere	0.88	0.92	0.38	0.96
katze	0.39	0.60	0.26	0.79
heisst	0.94	0.13	0.93	0.59
mimo	0.87	0.57	0.74	0.43
die	0.89	0.57	0.27	0.63
klettert	0.27	0.44	0.30	0.83
auf	0.11	0.27	0.36	0.20
den	0.55	0.01	0.95	0.08
baum	0.89	0.58	0.34	0.81

$\mathbf{W}^{(\text{new})}$

	dim1	dim2	dim3	dim4
unsere	0.88	0.92	0.38	0.96
katze	0.33	0.60	0.33	0.81
heisst	0.94	0.13	0.93	0.59
mimo	0.87	0.57	0.74	0.43
die	0.89	0.57	0.27	0.63
klettert	0.27	0.44	0.30	0.83
auf	0.11	0.27	0.36	0.20
den	0.55	0.01	0.95	0.08
baum	0.89	0.58	0.34	0.81

$\mathbf{W}^{(\text{delta})}$

	dim1	dim2	dim3	dim4
unsere	0.00	0.00	0.00	0.00
katze	-0.06	-0.00	0.07	0.01
heisst	0.00	0.00	0.00	0.00
mimo	0.00	0.00	0.00	0.00
die	0.00	0.00	0.00	0.00
klettert	0.00	0.00	0.00	0.00
auf	0.00	0.00	0.00	0.00
den	0.00	0.00	0.00	0.00
baum	0.00	0.00	0.00	0.00

Durch diese Korrekturen an den Gewichten haben sich die *Embeddings* für 'katze' bei den Kontext-Embeddings und 'klettert' bei den Zielwort-Embeddings angenähert.

$$\mathbf{v}_{katze}^{(old)} \cdot \mathbf{v}_{klettert}^{\prime(old)} = 1.12$$
$$\mathbf{v}_{katze}^{(new)} \cdot \mathbf{v}_{klettert}^{\prime(new)} = 1.41$$

Durch die Iterationen über sämtliche Dokumente und das Verschieben der Positionen innerhalb des Dokuments über die entsprechenden Wörter werden die *Embeddings* so gelernt, dass das Netzwerk mit der Zeit immer besser in der Lage ist, das entsprechende Zielwort vorherzusagen.

2.2 word2vec-Modelle: CBOW und Skip-Gram

word2vec, wie es in Mikolov u. a. [2013a] vorgestellt wurde, bietet zwei verschiedene Modelle. Beide basieren auf dem gleichen Grundprinzip. Sie varieren aber bezüglich der unterschiedlichen Grössen des *Sliding Windows* auf Seiten des Kontexts wie auch auf Seiten der Zielwörter, welche vorhergesagt werden.

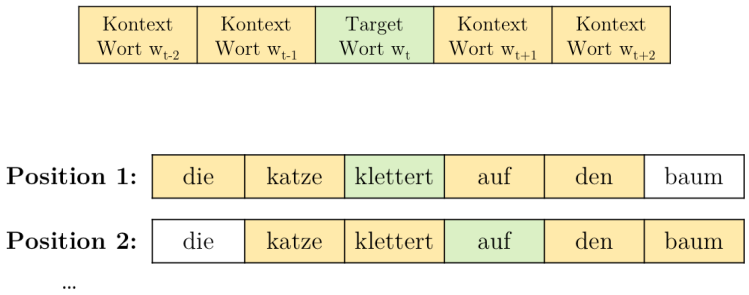
In den folgendenden Abschnitten werden die beiden Modelle kurz erklärt und die Auswirkungen der Änderungen gegenüber dem vereinfachten Modell mit einem Single-Word-Kontext aufgezeigt.

2.2.1 CBOW

Das Modell 'Continuous Bag of Words', kurz CBOW, weicht vom vereinfachten Beispielmodell insofern ab, dass der Kontext, welcher für die Vorhersage des Zielwortes verwendet wird, nicht nur aus einem einzigen Wort besteht.

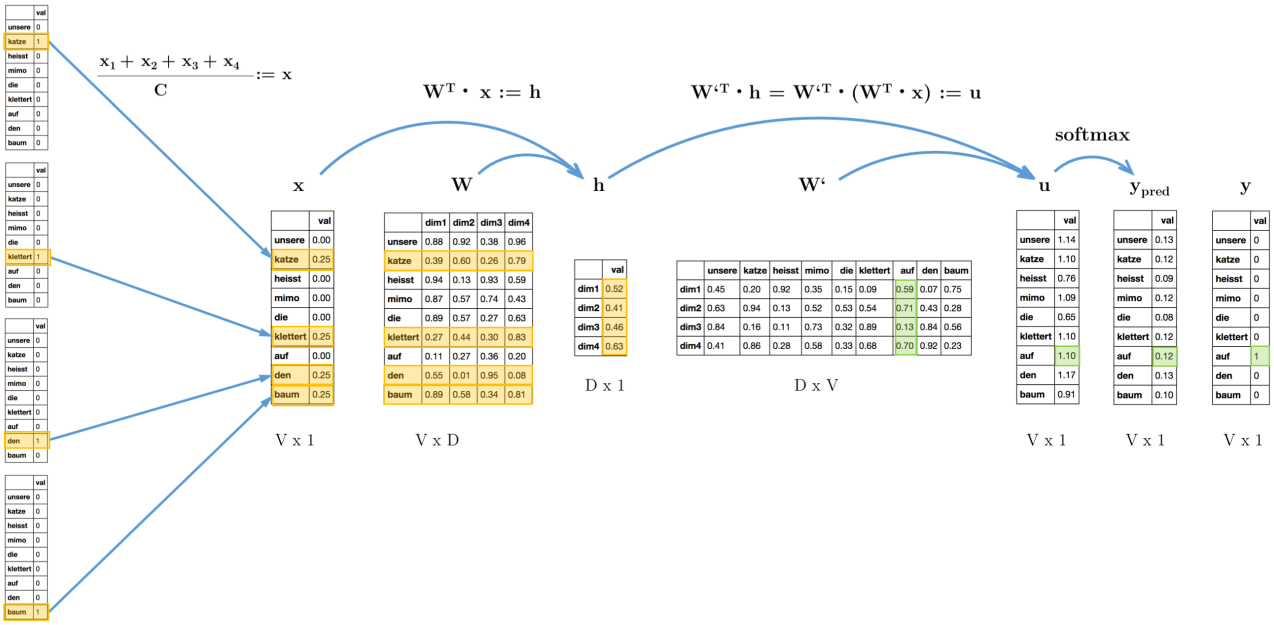
Die Grundidee ist, dass innerhalb des Dokuments die c Wörter vor und nach dem Zielwort als Kontext benutzt werden. Die Gesamtgrösse des Fenster ist damit $C = 2 \cdot c$. Gegeben diesen Kontext ist es die Aufgabe des Algorithmus, das mittlere Wort als Zielwort vorauszusagen. Die Reihenfolge der Kontext-Wörter wird hierbei nicht berücksichtigt.

Bei jedem Trainingsschritt werden sämtliche Ziel-Embeddings sowie diejenigen Kontext-Embeddings der Wörter im *Sliding Window* optimiert.



Die einzige Änderung der Berechnung verglichen mit dem vereinfachten Single-Word-Kontext-Modell ist ein zusätzlicher Schritt, der benötigt wird, um das Mittel über die *One-Hot-Encodings* der Kontext-Wörter zu bilden.

Ansonsten ist der Algorithmus und damit auch die Logik für die Vorwärts- wie auch die Rückwärtsberechnung identisch wie beim Single-Word-Kontext-Modell.



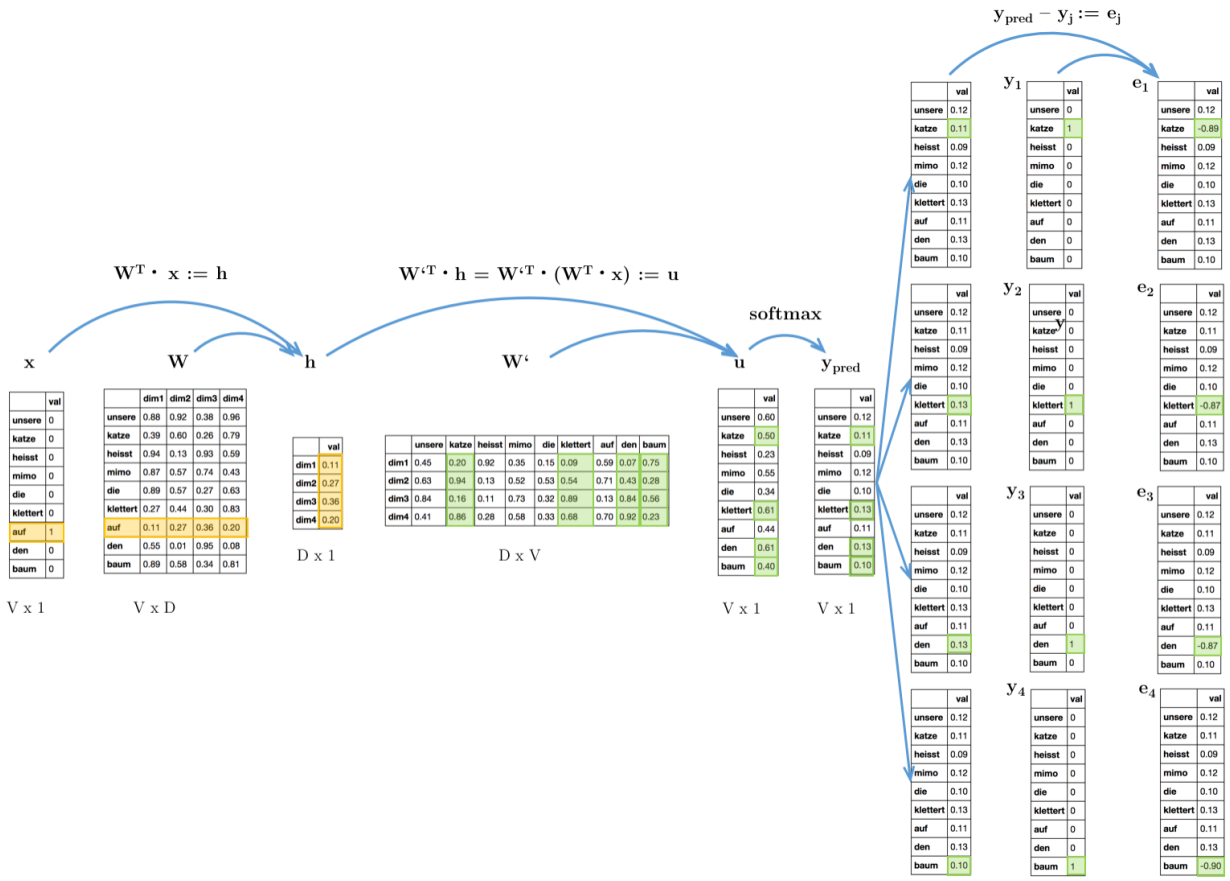
2.2.2 Skip-Gram

Das Skip-Gram Modell stellt eine Umkehrung des CBOW-Modells dar. Als Kontext wird wieder – gleich wie beim Single-Word-Kontext-Modell – mit einem einzigen Kontextwort gearbeitet. Der Algorithmus sagt aber nicht ein einzelnes Wort voraus, sondern ein Fenster von Wörtern rund um das Kontextwort. Der Begriff Kontextwort ist für dieses Modell etwas verwirrend, weil die Wörter, welche der Algorithmus voraussagen soll, eigentlich den Kontext des Eingabeworts darstellen. Ich verwende deshalb in diesem Zusammenhang die Terminologie des Eingabeworts und der Ausgabewörter, welche vorhergesagt werden sollen.

Output Wort w_{t-2}	Output Wort w_{t-1}	Input Wort w_t	Output Wort w_{t+1}	Output Wort w_{t+2}

Position 1:	die	katze	klettert	auf	den	baum
Position 2:	die	katze	klettert	auf	den	baum
...						

Dies bedingt bei der Berechnung Anpassungen im Vergleich zum Single-Word-Kontext-Modell, indem der Vorhersage-Vektor $\mathbf{y}_{pred}$ konzeptionell so oft repliziert wird, wie es für die Grösse des Fensters notwendig ist. Daraus errechnet sich ein Fehlervektor pro Ausgabewort $\mathbf{e}_j$, für alle $j = 1, 2, \ldots, C$.



Die Berechnungen für die Rückwärtsberechnung ändern sich insofern, als die verschiedenen Fehlervektoren benutzt werden müssen. So ergibt sich für die Anpassung der Gewichte von $\mathbf{W}'$ neu die Gleichung

$$\mathbf{v}'^{(new)}_{w_i} = \mathbf{v}'^{(old)}_{w_i} - \eta \cdot \mathbf{EI}_i \cdot \mathbf{h} \qquad \text{für } i = 1, 2, \ldots, V$$

wobei EI für die Summe der Fehlervektoren aller vorauszusagenden Wörter steht:

$$\mathbf{EI}_j = \sum_{c=1}^{C} e_{c,j}$$

Die Berechnung der neuen Gewichte W basiert ebenfalls auf dieser Summe der Fehlervektoren und lautet damit

$$\mathbf{v}^{(new)}_{w_c} = \mathbf{v}^{(old)}_{w_c} - \eta \mathbf{EH}^T$$

wobei EH nun folgendermassen definiert ist.

$$\mathbf{EH}_i = \sum_{j=1}^{V} \mathbf{EI}_j \times v'_{w_{ij}}$$

2.3 Optimierungen

Für das Lernen der Gewichte in den Matrizen $\mathbf{W}$ bzw. $\mathbf{W}'$ und damit der *Word Embeddings*, wird als Loss-Funktion

$$\mathcal{L} = -\log p(w_t|w_c) = \mathbf{v}'^{\mathrm{T}}_{w_t}\mathbf{v}_{w_c} - \log \sum_{k=1}^{V} e^{\mathbf{v}'^{\mathrm{T}}_{w_k}\mathbf{v}_{w_c}}$$

verwendet, welche mittels *Gradient Descent* minimiert wird.

Die Berechnung des zweiten Terms der Loss-Funktion ist bezüglich des Berechnungsaufwandes problematisch, da die Funktion $e^{\mathbf{v}'^{\mathrm{T}}_{w_k}\mathbf{v}_{w_c}}$ für alle Wörter des Vokabulars evaluiert werden muss, um die Summe bilden zu können. Das ist so nicht praktikabel für ein Vokabular in der Grössenordnung von Millionen von Wörtern, wie sie für word2vec typischerweise verwendet werden.

Eine weitere Problematik besteht in der Grösse der Gewichtsmatrizen, welche in jedem Trainingsschritt angepasst werden müssen. Die Matrix $\mathbf{W}'$ hat die Dimension $V \times D$. Das verwendete Vokabular liegt typischerweise in der Grössenordnung von 10^5 Wörtern, die Embeddings haben meist eine Dimension in der Grössenordnung von 50 bis 1000.

Bei einem konkreten Setting mit $V = 50'000$ und $D = 1000$ haben die beiden Gewichtsmatrixen also je 50 Millionen Werte. Um dieses Netzwerk zu trainieren und dabei ein *Overfitting* zu vermeiden, wird eine riesige Menge von Trainingsdaten benötigt, was auf die Laufzeit entsprechende Auswirkungen hat.

Es sind also Optimierungen notwendig, um die Algorithmen überhaupt praktikabel zu machen. In diesem Kapitel werden die Optimierungen erläutert, wie sie in Mikolov u. a. [2013b] präsentiert werden.

2.3.1 Hierarchical Softmax

Um die Berechnung der Softmax-Funktion performanter zu machen, kann mit einer hierarchischen Softmax-Funktion gearbeitet werden. Diese Optimierung basiert darauf, dass die Resultate der Softmax-Funktion nicht in einem flachen Layer abgelegt werden, sondern innerhalb einer Baumstruktur. Die Blätter enthalten dabei die Wörter des Vokabulars.

Das Ziel ist es, den Berechnungsaufwand von $O(V)$ auf $O(\log V)$ zu reduzieren.

Die Wahrscheinlichkeit für ein bestimmtes Blatt, d.h. das Resultat der Vorhersage, mit welcher Wahrscheinlichkeit dieses Wort w_i als Zielwort gewählt wird, wird aufgrund des Pfades durch die inneren Knoten berechnet. Jeder Knoten im Baum wird durch einen eindeutigen Pfad erreicht.

Die Wahrscheinlichkeit, dass dieses Wort gewählt wird, entspricht der Wahrscheinlichkeit, dass dieser Pfad im Baum von der Wurzel her durchlaufen wird.

Für jede Verzweigung stellt sich eine binäre Entscheidung, ob der Pfad nach links oder rechts weiterverfolgt wird. Diese binäre Entscheidung wird als Sigmoid-Funktion modelliert.

Jede Spalte in der Matrix $\mathbf{W}'$ entspricht nicht mehr einem Zielwort, sondern einem inneren Knoten innerhalb des Baumes. Von der Grösse der Matrix her gibt dies keine Verbesserung, da ein Baum mit V Blättern $V-1$ innere Knoten hat, aber der Berechnungsaufwand ist nur noch proportional zur Tiefe des Baumes in der Ordnung von $O(\log V)$ und nicht mehr zur Grösse des Vokabulars V.

Als weitere Optimierung wird in der Implementation von word2vec ein Huffman Baum verwendet, wobei die kürzesten Pfade für die häufigsten Wörter verwendet werden. Dadurch wird für die häufigsten Worte der Berechnungsaufwand noch einmal reduziert.

2.3.2 Negative Sampling

Wie in Kapitel 2.1 aufgezeigt wird die Wahrscheinlichkeit für das Auftreten eines Zielworts basierend auf der Eingabe mittels der Softmax-Funktion berechnet:

$$p(w_i|w_c) = y_{pred_i} = \frac{e^{u_i}}{\sum_{k=1}^{V} e^{u_k}}$$

Diese Funktion ist für ein grosses Vokabular sehr berechnungsintensiv. Die Optimierung durch *Negative Sampling* beschreitet den Weg, diese Berechnungsfunktion durch eine performantere Variante zu ersetzen. Dafür wird das Problem umformuliert: Anstelle der bisherigen Frage „Wie hoch ist die Wahrscheinlichkeit für dieses Zielwort gegeben das Eingabewort?“ lautet die Fragestellung neu: „Stammt diese Kombination von Eingabe- und Zielwort tatsächlich aus dem Korpus oder nicht?“ Es wird also als binäres Klassifikationsproblem formuliert. Die Aufgabe des Algorithmus ist es, die Wahrscheinlichkeit abzuschätzen, dass ein solches Tupel aus dem Korpus stammt. Weil es sich um eine binäre Entscheidung handelt, kann die Wahrscheinlichkeit mittels einer Sigmoid-Funktion modelliert werden:

$$P(y=1|w_I, w_O) = \sigma(\mathbf{v}'^{\mathrm{T}}_{w_I} \mathbf{v}_{w_O})$$

Diese Wahrscheinlichkeit kann basierend auf dem aktuellen Eingabewort w_I für jedes Zielwort w_O berechnet werden. Damit diese Performance-Optimierung aber den gewünschten Effekt erzielt, wird die Wahrscheinlichkeit nicht für jedes Wort des Vokabulars berechnet, sondern nur für eine kleine Menge.

Die Menge von Zielwörtern, welche als Tupel zusammen mit dem Eingabewort für das Training benützt werden, wird folgendermassen konstruiert. Das korrekte Tupel mit dem aktuellen Eingabewort und dem aktuellen Zielwort wird für das Training benützt und mit dem Label 1 versehen.

Zusätzlich werden Tupel konstruiert mit Zielwörtern, die nicht dem korrekten aktuellen Zielwort entsprechen, und mit 0 als Label gekennzeichnet. Die Anzahl von solchen *Negative Samples* k ist ein Modellparameter. Die Empfehlung gemäss Mikolov u. a. [2013b] schlägt dafür Werte zwischen 5 und 20 für kleine Trainingsdatensets vor und 2 bis 5 für grosse Datensets.

Eingabewort w_I	Zielbewort w_O	Label
katze	klettert	1
katze	tisch	0
katze	der	0
katze	sand	0
katze	stellt	0
katze	mond	0

Während des Trainings werden die Wahrscheinlichkeiten für die Klassifikation in eine der beiden Klassen gelernt und die Gewichte in den Matrizen $\mathbf{W}$ und $\mathbf{W}'$ entsprechend angepasst. Die Auswahl der *Negative Samples* erfolgt zufällig aus dem Vokabular gewählt, wobei die Häufigkeit eines Wortes im Korpus berücksichtigt wird.

Die Wahrscheinlichkeit für die Verwendung eines Wortes als *Negative Sample* in word2vec ist

$$P(w_i) = \frac{f(w_i)^{\frac{3}{4}}}{\sum_{k=1}^{V} f(w_k)^{\frac{3}{4}}}$$

Die Potenzierung mit $\frac{3}{4}$ wurde dabei empirisch als nützlich erkannt und nicht systematisch hergeleitet.

Dank dieser Optmierung kann durch die Berechnung der Sigmoid-Funktion für $k + 1$ Tupel anstelle der Berechnung der Softmax-Funktion über das gesamte Vokabular die Performance deutlich gesteigert werden. Gemäss Ruder [2016] wird dadurch eine Performance Verbesserung um einen Faktor von 50 - 100 erreicht werden.

Interessanterweise wird die Qualität der erzeugten *Word Embeddings* durch dieses Verfahren auch noch verbessert. Aus diesem Grund wird in der Praxis sehr häufig das Skip-Gram Modell mit *Negative Sampling* kombiniert eingesetzt.

2.3.3 Subsampling von häufigen Wörtern

Eine weitere Optimierung beschäftigt sich mit Wörtern, die in einem Korpus sehr häufig vorkommen, wie das Wort „die“ in unserem Beispiel. Typischerweise handelt es sich dabei um sogenannte Stoppwörter.

Es ergeben sich zwei Schwierigkeiten bei der Anwendung von word2vec mit solchen sehr häufigen Wörtern:

1. Die Tatsache, dass ein anderes Wort zusammen mit einem sehr häufigen Wort vorkommt, sagt wenig über das Wort aus und trägt deshalb wenig zur Verbesserung des *Word Embeddings* bei.

2. Ein solches sehr häufiges Wort kommt beim Training viel öfter vor, als es für die Ausprägung des entsprechenden *Word Embeddings* notwendig wäre und verlängert unnötigerweise die Laufzeit.

Um das Ungleichgewicht bezüglich der Häufigkeit der Wörter des Vokabulars auszugleichen, wird ein Subsampling angewendet. Das bedeutet, jedes Wort im Trainingsset wird mit einer gewissen Wahrscheinlichkeit weggelassen, wobei die Wahrscheinlichkeit grösser ist, je häufiger ein Wort vorkommt.

Konkret wird ein Wort mit Wahrscheinlichkeit

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$$

weggelassen, wobei $f(w_i)$ die Häufigkeit des Wortes w_i und t ein frei wählbarer Threshold ist. t wird meist in der Grössenordnung von 10^{-5} gewählt.

Diese Funktion hat die Eigenschaft, dass sie Wörter mit einer Häufigkeit $f(w_i) > t$ sehr aggressiv weglässt, während die Verteilung für die übrigen Wörter gut erhalten bleibt.

Auch diese Optimierung hat neben dem Effekt auf die Performance des Algorithmus auch auf die Qualität der *Word Embeddings* einen positiven Einfluss.

2.3.4 Behandlung von festen Ausdrücken

Eine Limitierung von *Word Embeddings*, die sich ausschliesslich auf einzelne Wörter beziehen, ist ihre Unfähigkeit, mit zusammengesetzten Phrasen umgehen zu können. So wird der Begriff „Neue Zürcher Zeitung“in die einzelnen Wörter zerlegt und es werden *Word Embeddings* für die drei einzelnen Wörter gelernt, aber kein Embedding für den ganzen Begriff.

Diese Limitierung kann behoben werden, indem in einem Vorbereitungssschritt solche Phrasen erkannt und die Wörter als ein gemeinsames Token behandelt werden und dem Vokabular hinzugefügt.

Die Erkennung erfolgt über das Verhältnis der Häufigkeit, in welchem zwei Wörter zusammen auftreten, gegenüber der Häufigkeit, in welcher sie in anderen Kontexten vorkommen. Wenn dieses Verhältnis einen bestimmten Threshold übersteigt, werden sie zu einem Token gebündelt.

Um auch Wortgruppen mit mehr als zwei Wörtern zu erkennen und zu Phrasen zu bündeln, werden mehrere Durchgänge über die Trainigsdaten durchgeführt, wobei der Threshold bei jedem folgenden Durchgang herabgesetzt wird.

Dank dieser Optimierung erhalten fixe Wortgruppen ein eigenes *Word Embedding*.

3 Evaluation

Wie in Kapitel 1 beschrieben ist es das Ziel von *Word Embeddings*, Vektoren als Wortrepräsentationen zu definieren, so dass die Vektoren die sprachliche Beziehung zwischen den Wörtern widerspiegeln. Um die Qualität eines *Word Embeddings* zu evaluieren, gibt es gemäss Schnabel u. a. [2015] zwei grundsätzlich unterschiedliche Ansätze: die extrinsische und die intrinsische Evaluierung.

Die extrinsische Evaluierung benutzt die generierten *Word Embeddings* als Inputs für weiterführende Aufgaben und misst die Qualität der Ergebnisse dieser weiterführenden Aufgaben. Damit können Rückschlüsse auf die Qualität der *Embeddings* gezogen werden. Beispiele für solche extrinsischen Evaluierungen sind part-of-speech tagging oder named-entity recognition Aufgaben. Das Problem an diesen Messungen liegt darin, dass die Übertragbarkeit nicht gegeben ist.

Intrinsische Evaluierung untersucht die Eigenschaften der *Word Embeddings* selber beispielsweise bezüglich ihrer syntaktischen oder semantischen Ähnlichkeit. Die Messungen, welche auf diesen intrinsischen Evaluierungen basieren, sind sehr gut vergleichbar um verschiedene Ansätze der Erstellung von *Word Embeddings* gegenüberzustellen. In Baroni u. a. [2014] sind verschiedene Metriken aufgelistet, mit denen unterschiedliche *Word Embeddings* evaluiert werden können. Einige Methoden existieren dabei bereits seit über 50 Jahren, andere wurden erst mit den neuen, vorhersagebasierten Ansätzen für die Erstellung von *Word Embeddings* definiert.

Semantische Verwandtheit Diese Evaluierungsmethode basiert auf Wortpaaren, welche von Menschen bezüglich ihrer Verwandtheit beurteilt wurden. Eines der ersten Testsets von solchen Wortpaaren, welches grosser Verwendung fand, ist WordSim353. Ein Ausschnitt aus der Testbasis von im Gesamten einigen hundert Wortpaaren ist in der folgenden Tabelle dargestellt.

Wort 1	Wort 2	Mittelwert der Beurteilungen	Person 1	Person 2	...	Person 13
love	sex	6.77	9	6		8
tiger	cat	7.35	9	7		7
tiger	tiger	10.00	10	10		10
book	paper	7.46	8	8		9
computer	keyboard	7.62	8	7		9
computer	internet	7.58	8	6		9
plane	car	5.77	6	6		7
train	car	6.31	7	7.5		8

Die Güte der erzeugten *Word Embeddings* wird gemessen, indem die Kosinus-Ähnlichkeit der Vektoren der beiden Wörter berechnet und mit den Beurteilungen der Testpersonen verglichen werden.

Synonymerkennung Die Aufgabe bei der Synonymerkennung besteht darin, aus jeweils vier Synonymkandidaten zu einem bestimmten Wort dasjenige auszuwählen, welches bezüglich der Bedeutung am nächsten liegt. Auch für diese Methode wird die Kosinus-Ähnlichkeit der Vektoren benutzt und das Wort mit dem höchsten Wert ausgewählt.

Konzeptkategorisierung Bei der Konzeptkategorisierung müssen Wörter in Kategorien gruppiert werden, z.B. „Helikopter“ und „Motorrad“ in die Kategorie „Fahrzeuge“ oder „Elefant“ und „Hund“ in die Kategorie „Säugetiere“ . Für die Bewertung der *Word Embeddings* wird hierfür ein Clustering Algorithmus angewendet und die Zuteilung in die Clusters mit der korrekten Kategorisierung verglichen.

Selectional Preference Diese Evaluierungsmethode arbeitet mit zwei Datensets, wobei das erste aus Nomen besteht und das zweite aus Verben. Die Aufgabe ist es, die typischsten Kombinationen zu finden. Dabei werden für die *Word Embeddings* je nach Methode die n ähnlichsten Nomen zu einem bestimmten Verb bestimmt und das Mittel dieser Vektoren mit dem Vektor des effektiv gesuchten Wortes verglichen.

Analogie Während die vorhergehenden Evaluierungsmethoden bereits seit längerer Zeit etabliert sind, ist die Methode der Analogie erst mit der Publikation von word2vec dazugekommen. Dabei wurde ein Set von je ungefähr 10′000 semantischen und syntaktischen Analogien definiert. Die semantischen Fragen bestehen dabei jeweils aus einem Wortpaar (z.B. „Bruder“-„Schwester“) und einem Testwort (z.B. „Enkel“). Die Aufgabe besteht darin, ein weiteres Wort zu finden, das die Beziehung aus dem Wortpaar auf das Testwort überträgt. In diesem Beispiel wäre das gesuchte Wort „Enkelin“. Um diese Aufgabe zu lösen, benützt Mikolov die Vektoroperation und formuliert das Problem als $\vec{Bruder} - \vec{Schwester} + \vec{Enkel}$.

Interessant ist sicherlich, dass alle diese Metriken auf ausgewählten Stichproben beruhen, welche explizit zusammengestellt und durch Testpersonen beurteilt wurden. Zudem ist die Menge der gemessenen Beispiele häufig recht klein. Trotzdem sind diese Evaluierungsmethoden die Vorgehensweisen, welche benutzt werden, um die verschiedenen Modelle gegenüberzustellen. [15]

In Baroni u. a. [2014] stellen die Autoren verschiedene häufigkeitsbasierte und vorhersagebasierte Modelle bezüglich dieser Evaluierungsmethoden gegenüber. Zu ihrem Erstaunen, wie sie selber sagen, schneiden die vorhersagebasierten Modelle in fast allen Qualitätsmetriken klar besser ab als die *Word Embeddings*, die mit häufigkeitsbasierten Modellen erstellt wurden. Das ist im Besonderen deshalb überraschend, da die häufigkeitsbasierten Modelle zum Teil unter Beizug von externen Quellen und manuellen Regeln arbeiteten, während word2vec einfach in der Standardausführung mit vorgeschlagenen Parameterwerten eingesetzt wurde. Die einzige Evaluierungsmethode, bei welcher die häufigkeitsbasierten Modelle word2vec überlegen waren, ist die Selectional-Preference-Metrik. Dieses Ergebnis hat zum klaren Verdikt geführt, welches sich im Titel der erwähnten Publikation widerspiegelt: „Don't count, predict!".

Seit dieser Vergleichsstudie sind neuere Modelle dazugekommen, so dass diese Resultate nicht mehr uneingeschränkte Gültigkeit haben. Besonders erwähnenswert ist sicher das Modell GloVe, welches einen neuen Ansatz verfolgt, der auf globalen statistischen Information aufsetzt (also häufigkeitsbasiert) und basierend auf diesen Informationen mittels linearer Regression die *Word Embeddings* lernt. Die in Pennington u. a. [2014] aufgeführten Ergebnisse zu den Evaluierungsmethoden semantische Verwandtheit, semantische und syntaktische Analogie sowie die Anwendung für die Bestimmung von Named Entities zeigen Vorteile für GloVe bei vielen der verwendeten Metriken. Insofern wäre eine erneute unabhängige Gegenüberstellung der verschiedenen Ansätze spannend.

[15] In einem Vortrag von Christopher Manning äussert ein Teilnehmer die kritische Frage, ob es wirklich legitim ist, basierend auf einzelnen erfolgreichen Ergebnissen davon auszugehen, dass die erzeugten Embeddings als Ganzes tatsächlich gut sind (https://youtu.be/nFCxTtBqF5U?t=21m26s). Die Antwort von Manning ist, dass keine Alternative dazu existiert, die Gesamtqualität exemplarisch an einigen (wenigen) Instanzen zu überprüfen.

4 Zusammenfassung

Die Modelle von word2vec sind verblüffend bezüglich der Qualität der Resultate und der Einfachheit der Architektur der zugrundeliegenden neuronalen Netze.

Bei der vertieften Analyse der Modelle wird aber deutlich sichtbar, dass die Komplexität sehr wohl vorhanden ist. Dies gilt im Besonderen sobald es darum geht, die Modelle nicht nur konzeptionell zu definieren, sondern auch bezüglich der Performance auf die benötigten Volumina anwendbar zu machen. Die Betrachtung der Optimierungen zeigt in den Details, dass sehr viel empirische Arbeit darin steckt, indem Parameterwerte oder Entscheide für bestimmte Berechnungsschritte nicht theoretisch begründet werden, sondern mit den damit erzielten besseren Resultaten.

Die initialen Veröffentlichungen von Mikolov et al. haben sich sehr stark auf die erzielten Resultate fokussiert und viel offen gelassen bezüglich der Funktionsweise der Modelle. Dieser grosse Bedarf an Erklärungen wurde durch eine Vielzahl von Publikationen, Blogs und Tutorials gedeckt, welche jeweils einzelne Aspekte detailliert beleuchten. Die Zusammenstellung „Awesome Embedding Models“[16] bietet dafür einen ausgezeichneten Ausgangspunkt.

Für mein Verständnis von word2vec haben Erläuterungen an konkreten Beispielen mit den expliziten Berechnungen sehr viel beigetragen. So ist auch das Herzstück dieser Arbeit ein solches Beispiel, wobei mir neben den Details der einzelnen Berechnungsschritte auch die Übersichten über den Ablauf eines ganzen Modells viel zum Verständnis geholfen haben.

Basierend auf den Grundlagen, welche im CAS Big Data und Machine Learning der Universität Zürich[17] in der Einführungsvorlesung und an den Kurstagen zu Deep Learning und Text Analytics vermittelt werden, bietet diese Arbeit die Möglichkeit, für Interessierte einen vertieften Blick hinter die Kulissen von word2vec zu werfen. Das dazugehörende iPython Notebook[18] liefert für eine vertiefte hands-on Auseinandersetzung einen guten Ausgangspunkt.

[16] https://github.com/Hironsan/awesome-embedding-models

[17] http://www.ifi.uzh.ch/de/studies/cas.html

[18] https://github.com/thomasbriner/word2vec-Schritt-fuer-Schritt

Literaturverzeichnis

[Baroni u. a. 2014] Baroni, Marco ; Dinu, Georgiana ; Kruszewski, Germán: Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Baltimore, Maryland : Association for Computational Linguistics, Juni 2014, S. 238–247. – URL `http://www.aclweb.org/anthology/P14-1023`

[Goldberg 2017] Goldberg, Yoav: *Neural Network Methods for Natural Language Processing*. Morgan & Claypool Publishers, 2017 (Synthesis Lectures on Human Language Technologies 10)

[Goldberg und Levy 2014] Goldberg, Yoav ; Levy, Omer: word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method. In: *CoRR* abs/1402.3722 (2014). – URL `http://arxiv.org/abs/1402.3722`

[Levy und Goldberg 2014] Levy, Omer ; Goldberg, Yoav: Linguistic Regularities in Sparse and Explicit Word Representations. In: *Proceedings of the Eighteenth Conference on Computational Natural Language Learning, CoNLL 2014, Baltimore, Maryland, USA, June 26-27, 2014*, URL `http://aclweb.org/anthology/W/W14/W14-1618.pdf`, 2014, S. 171–180

[Manning 2017] Manning, Christopher: *Lecture 2 Word Vector Representations: word2vec*. 2017. – URL `https://youtu.be/ERibwqs9p38?t=32m27s`

[Mikolov u. a. 2013a] Mikolov, Tomas ; Sutskever, Ilya ; Chen, Kai ; Corrado, Greg ; Dean, Jeffrey: Distributed Representations of Words and Phrases and their Compositionality. In: *CoRR* abs/1310.4546 (2013). – URL `http://arxiv.org/abs/1310.4546`

[Mikolov u. a. 2013b] Mikolov, Tomas ; Sutskever, Ilya ; Chen, Kai ; Corrado, Greg S. ; Dean, Jeff: Distributed Representations of Words and Phrases and their Compositionality. In: Burges, C. J. C. (Hrsg.) ; Bottou, L. (Hrsg.) ; Welling, M. (Hrsg.) ; Ghahramani, Z. (Hrsg.) ; Weinberger, K. Q. (Hrsg.): *Advances in Neural Information Processing Systems 26*. Curran Associates, Inc., 2013, S. 3111–3119. – URL `http://papers.nips.cc/paper/`

5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf

[Pennington u. a. 2014] Pennington, Jeffrey ; Socher, Richard ; Manning, Christopher D.: Glove: Global Vectors for Word Representation. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, URL http://aclweb.org/anthology/D/D14/D14-1162.pdf, 2014, S. 1532–1543

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

[Ruder 2016] Ruder, Sebastian: *On word embeddings - Part 2: Approximating the Softmax*. 2016. – URL http://ruder.io/word-embeddings-softmax

[Schnabel u. a. 2015] Schnabel, Tobias ; Labutov, Igor ; Mimno, David M. ; Joachims, Thorsten: Evaluation methods for unsupervised word embeddings. In: Màrquez, Lluís (Hrsg.) ; Callison-Burch, Chris (Hrsg.) ; Su, Jian (Hrsg.) ; Pighin, Daniele (Hrsg.) ; Marton, Yuval (Hrsg.): *EMNLP*, The Association for Computational Linguistics, 2015, S. 298–307. – URL http://dblp.uni-trier.de/db/conf/emnlp/emnlp2015.html#SchnabelLMJ15. – ISBN 978-1-941643-32-7