



**Universität  
Zürich<sup>UZH</sup>**

**Institut für Computerlinguistik**

## word2vec Schritt für Schritt

Schriftliche Arbeit zum CAS Big Data und Machine Learning der  
Universität Zürich

Thomas Briner  
thomas.briner@gmail.com

31.05.2018

## Zusammenfassung

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

# Contents

<b>Abstract</b>	<b>i</b>
<b>Contents</b>	<b>ii</b>
<b>1 Einführung</b>	<b>1</b>
1.1 Ansätze zur Erzeugung von Word Embeddings . . . . .	2
1.2 Ziel dieser Arbeit . . . . .	3
1.3 Verwandte Arbeiten . . . . .	5
<b>2 word2vec</b>	<b>7</b>
2.1 Intuition anhand einer Single-Word Kontext Architektur . . . . .	8
2.1.1 Forward Pass . . . . .	10
2.1.2 Backward Pass . . . . .	15
2.2 word2vec Architekturen: CBOW und Skip-Gram . . . . .	18
2.2.1 CBOW . . . . .	19
2.2.2 Skip-Gram . . . . .	20
2.3 Optimierungen . . . . .	22
2.3.1 Hierarchical Softmax . . . . .	22
2.3.2 Negative Sampling . . . . .	23
2.3.3 Subsampling von häufigen Wörtern . . . . .	25
2.3.4 Behandlung von festen Ausdrücken . . . . .	26
<b>3 Evaluation</b>	<b>27</b>
<b>4 Zusammenfassung</b>	<b>30</b>
<b>References</b>	<b>31</b>

# 1 Einführung

Wörter in eine numerische Repräsentation abzubilden 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 obiger Suche auf einer der gängigen Suchmaschinen eine Seite mit dem Titel "Interne Festplatte - Kapazität - Mac Zubehör".<sup>1</sup> Offensichtlich ist die Suchmaschine in der Lage, die Ähnlichkeit der Begriffe "Harddisk" und "Festplatte" sowie "Grösse" und "Kapazität" in diesem Kontext zu erkennen.

Eine intuitive 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 dann in einem Vektor, welcher an der Position des entsprechenden Wortes den Wert 1 stehen hat, während an allen anderen Positionen eine 0 steht.

Harddisk:  $[0, 0, 0, \dots, 0, 1, 0]$

Festplatte:  $[0, 1, 0, \dots, 0, 0, 0]$

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 0.

$$[0, 0, \dots, 0, 1, 0] \cdot [0, 1, 0, \dots, 0, 0] = 0$$

Diese Repräsentation ist deshalb für die erwähnten Anwendungen nicht von Nutzen.<sup>2</sup>

---

<sup>1</sup><https://www.google.ch> in Incognito Browser Window, Stand 01.05.2018

<sup>2</sup>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. Diese Erkenntnis wird durch das Zitat von J.R. Firth von 1957 "You shall know a word by the company it keeps" bestens auf den Punkt gebracht. Diese Information muss in die Repräsentation einfließen, damit sie nutzbringend eingesetzt werden kann. Die Hypothese ist dabei, dass Wörter mit einer ähnlichen Bedeutung tendenziell in ähnlichen Kontexten auftauchen.

Diese Betrachtung des Kontextes zu einem bestimmten Wort ist die Basis für word2vec, wie es in Mikolov et al. [2013a] präsentiert wurde. word2vec ist ein Framework mit zwei unterschiedlichen zugrundeliegenden Architekturen, welcher das Gebiet des Natural Language Processing grundlegend revolutioniert hat.

## 1.1 Ansätze zur Erzeugung von Word Embeddings

Die Methoden für die Abbildung von Wörtern<sup>3</sup> lassen sich in zwei unterschiedliche Ansätze unterteilen: count-based und prediction-based.

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

Die Abbildung in einer Cooccurrence-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 prediction-based Methoden, zu denen word2vec gehört, gehen dabei einen anderen Weg, der im Gegensatz zur globalen statistischen Sicht auf einen lokalen Kontext fokussiert. Dabei wird mit dem Prinzip eines Sliding Windows der Text gescannt, wobei immer aufgrund eines aktuellen Kontextes im Sliding Window ein Target - ein einzelnes oder mehrere Wörter - vorhergesagt werden. Als Nebenprodukt dieser Verfahren entstehen dabei Word Embeddings,

---

<sup>3</sup>Diese Abbildungsmethoden sind auch auf andere Sprachkonstrukte wie Sätze, Dokumente sind diese Abbildungsmethoden gut übertragbar.

also Repräsentationen für die verschiedenen Wörter. Sie werden als Gewichte eines neuronalen Netzes modelliert und während des Trainings über den gesamten Corpus mit den bekannten Techniken von Stochastic Gradient Descent und Backpropagation justiert.

Das word2vec Framework war der Vorreiter dieser prediction-based Methoden und wurde in Mikolov et al. [2013a] präsentiert. Mit diesem Verfahren wurden auch neue Möglichkeiten für die Anwendung der Word Embeddings populär, die danach auch in die entsprechenden Qualitätsmetriken einfließen, wie sie in Baroni et al. [2014] angewendet und in Levy and Goldberg [2014] genauer untersucht werden. Auf die entsprechenden Ergebnisse wird in Kapitel ?? eingegangen.

Den prediction-based Methoden wird vorgeworfen, dass der Trainingsaufwand mit der Korpusgrösse proportional wächst und sie darüber hinaus die globalen statistischen Informationen nicht nutzen, welche auf effiziente Art ermittelt werden könnten.

Diese Punkte werden im Besonderen von den Vertretern des GloVe Algorithmus vorgebracht. Die Abkürzung GloVe steht für "Global Vector". Der Kern von GloVe besteht darin, dass die bedingten Wahrscheinlichkeiten für das Vorkommen eines Wortes  $w_i$  gegeben das Vorkommen eines 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 count-based Methoden mit der Qualität der prediction-based Methoden zu verschmelzen.

## 1.2 Ziel dieser Arbeit

Innerhalb des CAS Kurses Big Data and Machine Learning der Universität Zürich <sup>4</sup> wurde im Modul Text Analytics das Thema der Word Embeddings und im Besonderen das Framework word2vec vorgestellt.

Das Grundprinzip, basierend auf einem bestimmten Input eine Voraussage für Wörter in dessen Kontext zu machen, wie sie in den Dokumenten des Korpus vorkommen, ist eine einleuchtende Idee. Dass das trainierte Modell, welches nach diesem Training in der Lage ist, solche Wörter mit einer hohen Treffsicherheit vorauszusagen, aber gar nicht das eigentliche Resultat des ganzen Prozesses ist, sondern lediglich ein Mittel, um so Word Embeddings zu berechnen, ist aus meiner Sicht raffiniert.

---

<sup>4</sup><http://www.ifi.uzh.ch/de/studies/cas.html>

Die Ergebnisse, die sich für verschiedene Aufgaben mit diesen Embeddings erzielen lassen, sind aber schlicht überwältigend. So können basierend auf diesen Word Embeddings sowohl syntaktische als auch semantische Fragestellungen beantwortet werden. Die bekanntesten Beispiele sind dabei die Wortanalogien vom Typ "Welches Wort ist für x dasselbe wie y für z ist?"

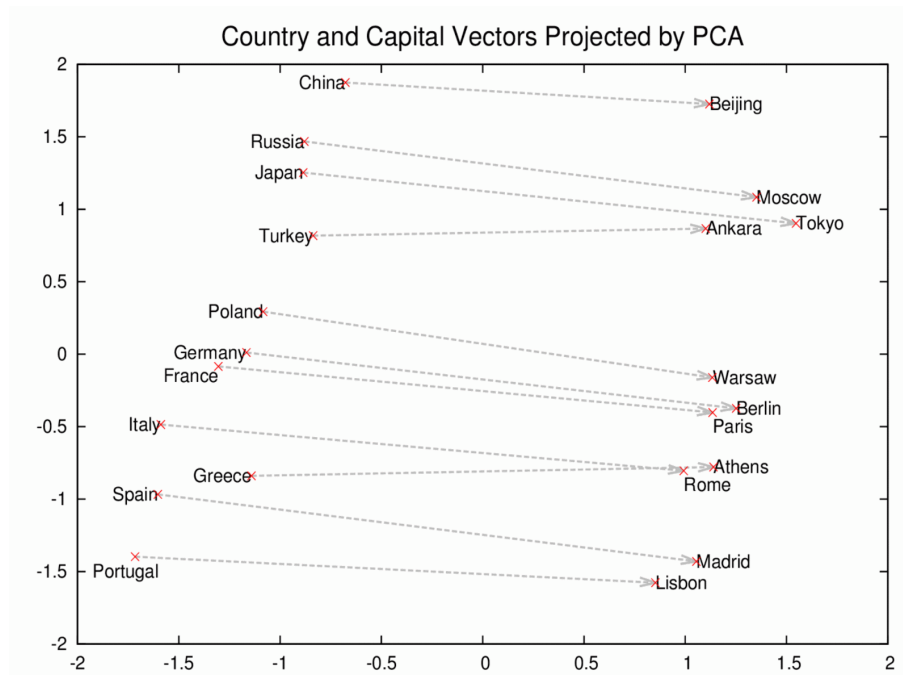
Auf der Seite <https://rare-technologies.com/word2vec-tutorial> unter "Bonus App" gibt es die Möglichkeit, selber solche Analogien zu testen. Die Ergebnisse sind dabei verblüffend.

man	is to	King	as	woman	is to	Queen
Paris	is to	France	as	Berlin	is to	Germany
bird	is to	air	as	fish	is to	water

Dieses Prinzip funktioniert wie bereits erwähnt auch für syntaktische Eigenschaften:

large	is to	larger	as	small	is to	smaller
example	is to	examples	as	analogy	is to	analogies

Basierend auf den Word Embeddings ist es auch möglich, mittels Dimensionsreduktionsverfahren Visualisierungen davon zu erzeugen.



Aus dieser Faszination heraus habe ich mich entschieden, einen Blick hinter die Kulissen dieses Frameworks zu werfen, um die Details zu verstehen, die diesem zugrundeliegen.

Das Ziel des vorliegenden Dokuments ist es, das Prinzip in einer Form zu präsentieren, welches es erlaubt, basierend auf den Informationen aus den Vorlesungen zur Einführung in Machine Learning und der Vorlesung zu Deep Learning, die Funktionsweise von word2vec als Ganzes zu verstehen und anhand des konkreten Beispiels Schritt für Schritt die einzelnen Puzzleteile zusammenfügen zu können.

Im folgenden Kapitel werde ich mit einem Überblick über verschiedene Herangehensweisen starten und das Framework word2vec im Bezug auf andere Methoden einordnen.

## 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 im Besonderen auch word2vec wird in verschiedenen Lehrbüchern behandelt, so beispielsweise in Goldberg [2017], in welchem die beiden Architekturen von word2vec wie auch zwei Optimierungen behandelt werden. Das entsprechende Kapitel gibt in kurzer Zeit einen ersten Überblick, wobei das Verständnis für die Grundprinzipien vorausgesetzt werden.



In Rong [2014] wird das word2vec Framework von Grund auf analysiert, indem von einer Vereinfachung ausgegangen wird, die als Input wie auch als Output 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. Im Anhang ist noch eine grundsätzliche Abhandlung über Back Propagation enthalten und der Hinweis auf eine Visualisierung, welche online unter <https://ronxin.github.io/wevi/> zur Verfügung steht.

Die Methode des Negative Sampling wird in Goldberg and Levy [2014] detailliert hergeleitet, wobei für mich im Besonderen auch der abschliessende Abschnitt interessant war, dass die Frage, weshalb dies zu guten Word Embeddings führt, auch für die Autoren unklar bleibt.

## 2 word2vec

Das Ziel von word2vec ist es, dichte Word Embeddings zu berechnen, so dass die Ähnlichkeit von Wörtern auch in den Word Embeddings erhalten bleibt.

Dafür macht es sich den Leitsatz von J.R.R. Firth zu Nutze, dass die Bedeutung eines Wortes deutlich wird durch den Kontext, in welchem es eingesetzt wird.

word2vec benützt dafür ein flaches neuronales Netz, das aus einem Input Layer, nur einem Hidden Layer und einem Output Layer besteht. Dieses Netzwerk wird mit den verschiedenen Dokumenten des Korpus trainiert. Die Grundidee ist dabei, dass aufgrund einer bestimmten Kontextes das Netzwerk voraussagen versucht, was das gesuchte Wort (bzw. je nachdem auch die gesuchten Wörter) im Bezug auf diesen Kontext sein könnte.

Diese ganze Problemmodellierung ist aber eigentlich 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 beinhalten direkt die Word Embeddings.

word2vec beinhaltet zwei unterschiedliche Algorithmen, mit welchen diese Gewichte und damit die Word Embeddings gelernt werden können, nämlich CBOW<sup>5</sup> und Skip-Gram<sup>6</sup>.

Die beiden Algorithmen basieren auf dem identischen Prinzip, dass basierend auf einem Kontext ein Set von 1 bis n Target Wörtern vorausgesagt werden. Es variieren dabei lediglich die Definitionen, wieviele Wörter als Kontext benutzt bzw. wieviele als Target vorausgesagt werden.

---

<sup>5</sup>für continuous **bag of words**

<sup>6</sup>Der Name stammt von Idee, N-Gramme mit Lücken darin zu verwenden, wie in <https://en.wikipedia.org/wiki/N-gram#Skip-gram> beschrieben

## 2.1 Intuition anhand einer Single-Word Kontext Architektur

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 CBOW und Skip-Gram. Die Differenzen liegen ausschliesslich in der Wahl des Kontext- bzw. Target-Fensters. Die Übertragung der Mechanismen auf CBOW und Skip-Gram wird in den Kapiteln ?? und ?? 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 maximieren, ohne aber an den Mechanismen Veränderungen vorzunehmen.

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

Menge der Dokumente =  $\{D_1, D_2\}$

Dokument  $D_1$  = 'Unsere Katze heisst Mimo'

Dokument  $D_2$  = 'Die Katze klettert auf den Baum'

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

Vokabular = ['unsere', 'katze', 'heisst', 'mimo', 'die', 'klettert', 'auf', 'den', 'baum']

$V$  = Länge des Vokabulars = 9

Die wesentliche Vereinfachung gegenüber den realen Architekturen von word2vec liegt in der Wahl des Kontextes. Als Kontext wird in dieser Variante nur ein einziges Wort verwendet. Basierend auf diesem Kontextwort  $w_c$  wird als prediction ebenfalls ein einziges Target Wort  $w_t$  vorausgesagt.

Kontext	Target
Wort $w_c$	Wort $w_t$

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

<b>Position 1:</b>	die	katze	klettert	auf	den	baum
<b>Position 2:</b>	die	katze	klettert	auf	den	baum
...						

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

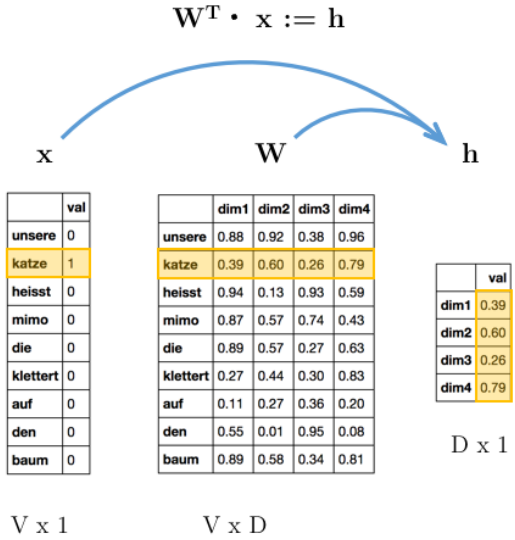
Die Dimensionalität der Word Embeddings ist ein Modellparameter und wird in der Realität typischerweise in der Grössenordnung von 200 - 1000 gewählt.

Für dieses Beispiel wird mit der Dimensionalität  $D = 4$  gearbeitet, um die Darstellungen wie auch die Berechnungen übersichtlich und leicht nachvollziehbar zu machen.

Bei word2vec wird für jedes Wort nicht nur mit einem Embedding gearbeitet, sondern mit zwei Embeddings: Einem für die Wörter als Kontextwort und mit einem zweiten für die Wörter als Target Wort. 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 sehr seltenen Fällen ein Wort direkt zweimal hintereinander in einem Text vorkommt.

### 2.1.1 Forward Pass

Kontextwort als One-Hot- Encoding	<p>Das Kontextwort <math>w_c</math> an der aktuellen Position des Sliding Windows wird als One-Hot-Encoding codiert.</p> $x_i = \begin{cases} 1 & \text{für Kontextwort} \\ & \text{an dieser Position} \\ 0 & \text{überall sonst} \end{cases}$ $x = w_c$ <p>Der Vektor <math>x</math> hat die Dimension <math>V \times 1</math></p>	<p><b>x</b></p> <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> <p><math>V \times 1</math></p>		val	unsere	0	katze	1	heisst	0	mimo	0	die	0	klettert	0	auf	0	den	0	baum	0																														
	val																																																			
unsere	0																																																			
katze	1																																																			
heisst	0																																																			
mimo	0																																																			
die	0																																																			
klettert	0																																																			
auf	0																																																			
den	0																																																			
baum	0																																																			
Matrix mit Word Em- beddings für Kontextwort	<p><math>W</math> ist die Matrix mit Word Embeddings für Kontextwörter und hat die Dimension <math>V \times D</math>. Die Matrix <math>W</math> wird mit Zufallswerten initialisiert. Jede Zeile <math>v_i</math> entspricht dem Word embedding für das <math>i</math>-te Wort des Vokabulars.</p>	<p><b>W</b></p> <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> <p><math>V \times D</math></p>		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
	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																																																

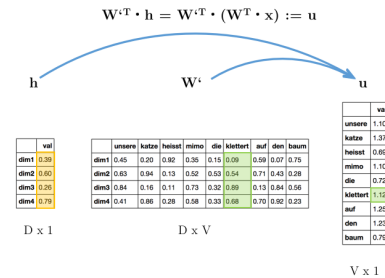
<div>Berechnung Hidden Layer</div>	<div>Die Multiplikation <math>\mathbf{h} = \mathbf{W}^T \mathbf{x}</math> der Gewichtsmatrix <math>\mathbf{W}^T</math> mit dem One-Hot-Encoding des Kontextworts <math>S^T x</math> entspricht der Selektion der entsprechenden Zeile für das Kontextwort aus der Gewichtsmatrix.</div> <div>Der Hidden Layer enthält also genau die Gewichte für das jeweilige Kontextwort.</div>	<div><math display="block">\mathbf{W}^T \cdot \mathbf{x} := \mathbf{h}</math></div> <div><table><tr><td></td><td>val</td></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><p><math>V \times 1</math></p><table><tr><td></td><td>dim1</td><td>dim2</td><td>dim3</td><td>dim4</td></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><p><math>V \times D</math></p><table><tr><td></td><td>val</td></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><p><math>D \times 1</math></p></div>		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
	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																																																																																	
<div>Matrix mit Word Em- beddings für Target Wort</div>	<div>Die Matrix <math>\mathbf{W}'</math> beinhaltet die Word Embeddings für alle Wörter als mögliche Target Wörter. Sie hat die Dimension <math>D \times V</math> und wird ebenfalls mit Zufallszahlen initialisiert.</div>	<div><math display="block">\mathbf{W}'</math><table><tr><td></td><td>unsere</td><td>katze</td><td>heisst</td><td>mimo</td><td>die</td><td>klettert</td><td>auf</td><td>den</td><td>baum</td></tr><tr><td>dim1</td><td>0.45</td><td>0.20</td><td>0.92</td><td>0.35</td><td>0.15</td><td>0.09</td><td>0.59</td><td>0.07</td><td>0.75</td></tr><tr><td>dim2</td><td>0.63</td><td>0.94</td><td>0.13</td><td>0.52</td><td>0.53</td><td>0.54</td><td>0.71</td><td>0.43</td><td>0.28</td></tr><tr><td>dim3</td><td>0.84</td><td>0.16</td><td>0.11</td><td>0.73</td><td>0.32</td><td>0.89</td><td>0.13</td><td>0.84</td><td>0.56</td></tr><tr><td>dim4</td><td>0.41</td><td>0.86</td><td>0.28</td><td>0.58</td><td>0.33</td><td>0.68</td><td>0.70</td><td>0.92</td><td>0.23</td></tr></table><p><math>D \times V</math></p></div>		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																														
	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																																																																									

Berechnung  
Ähnlichkeit

Um eine Prediction für das Target Wort zu machen, wird nun die Gewichtsmatrix für das Target Wort  $\mathbf{W}'$  mit dem Vektor  $\mathbf{h}$  multipliziert:  $\mathbf{u} = \mathbf{W}'^T \times \mathbf{h} = \mathbf{W}'^T \cdot (\mathbf{W}^T \cdot \mathbf{x})$

So wird mittels des Dot-Operators die Ähnlichkeit des Embeddings des Kontextwortes mit jedem möglichem Target Wort berechnet.

Der Vektor  $\mathbf{u}$  ist nicht normiert und stellt deshalb noch keine Wahrscheinlichkeitsverteilung dar.

Anwenden  
der Softmax  
Funktion

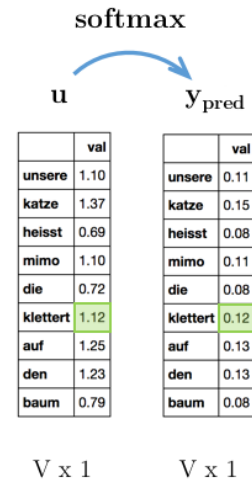
Um die Ergebnisse der Ähnlichkeitsberechnung im Vektor  $\mathbf{u}$  vergleichbar zu machen, wird die Softmax Funktion darauf angewendet.

Jedes Element des berechneten Vektors  $y_{pred_i}$  zeigt nun die Wahrscheinlichkeit für das  $i$ -te Element, mit welcher der Algorithmus basierend auf dem bekannten Kontext voraussagt, dass dieses Wort das Target Wort ist.

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

Die Summe aller Elemente  $y_{pred_i}$  ergibt nun 1.

$$\sum_{i=1}^V y_{pred_i} = 1$$

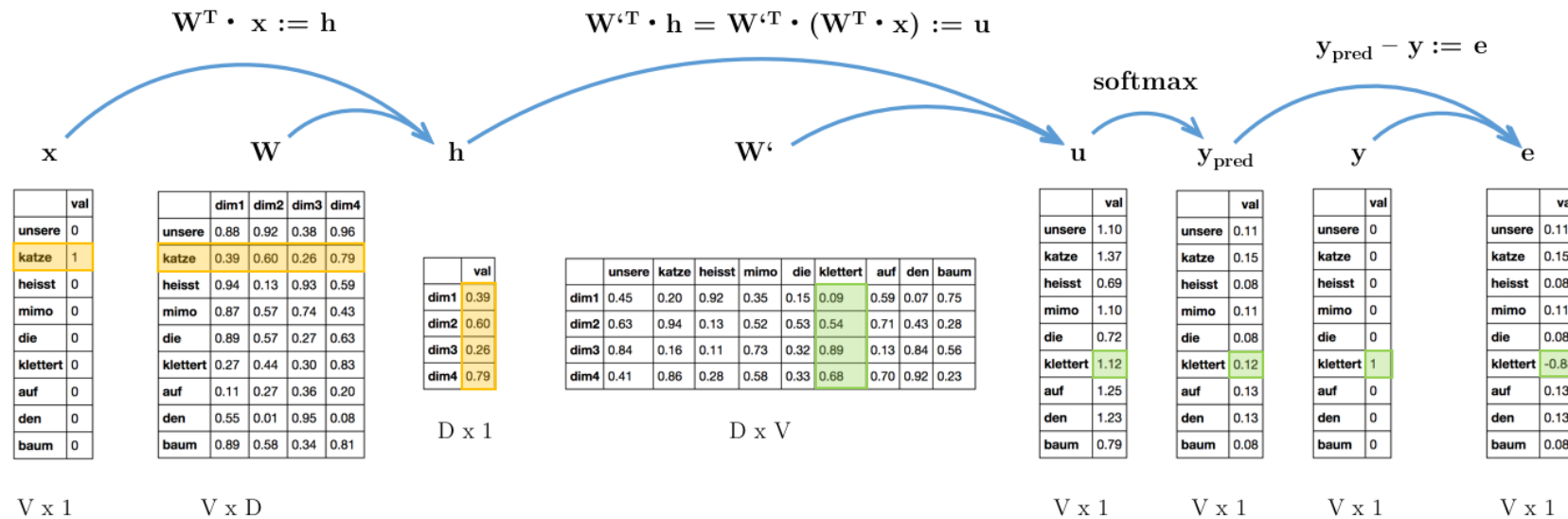


Target Wort als One-Hot- Encoding $y$	Der Vektor $y$ ist das One-Hot- Encoding für das Target Wort. Es wird für die Berechnung des Fehlervektors benötigt.	<div><math>y</math></div> <table><tr><td></td><td>val</td></tr><tr><td>unsere</td><td>0</td></tr><tr><td>katze</td><td>0</td></tr><tr><td>heisst</td><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> <div><math>V \times 1</math></div>		val	unsere	0	katze	0	heisst	0	mimo	0	die	0	klettert	1	auf	0	den	0	baum	0																																								
	val																																																													
unsere	0																																																													
katze	0																																																													
heisst	0																																																													
mimo	0																																																													
die	0																																																													
klettert	1																																																													
auf	0																																																													
den	0																																																													
baum	0																																																													
Fehlervektor $e$	Der Fehlervektor berechnet sich aus der Differenz zwischen der Prediction $y_{pred}$ und dem tatsächlichen Target Wort $y$ als One-Hot-Encoding.	<div><math>y_{pred} - y := e</math></div> <div><div><math>y_{pred}</math></div><table><tr><td></td><td>val</td></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><div><math>V \times 1</math></div></div> <div><div><math>y</math></div><table><tr><td></td><td>val</td></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><div><math>V \times 1</math></div></div> <div><div><math>e</math></div><table><tr><td></td><td>val</td></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><div><math>V \times 1</math></div></div>		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
	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																																																													



## Übersicht Forward Pass

Hier sind alle Schritte des Forward Pass im Zusammenhang dargestellt. Eine Durchgang des Forward Pass bezieht sich hierbei auf eine bestimmte Position des Sliding Windows.



### 2.1.2 Backward Pass

Das Ziel des Trainings ist es, für die vorausgesagte Wahrscheinlichkeit für das tatsächliche Target Wort zu optimieren.

Sei  $\mathbf{v}_{w_i}$  die  $i$ -te Zeile der Matrix  $\mathbf{W}$ , in welcher das Encoding für Wort  $w_i$  als Kontextwort steht und  $\mathbf{v}'_{w_j}$  die  $j$ -te Spalte der Matrix  $\mathbf{W}'$ , in welcher das Encoding für Wort  $w_j$  als Target Wort steht.

Die Wahrscheinlichkeit, dass ein bestimmtes Wort  $j$  als Target Wort gewählt wird gegeben das aktuelle Kontextwort  $w_c$ , lässt sich damit formulieren als

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

Die Loss Funktion unter Anwendung der Negative Log Likelihood Methode heisst

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

Der aktuelle Loss im obigen Beispiel ist also

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

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

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

Die Gleichung für die Anpassung der Gewichte von  $W'$  lautet

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

Falls das Wort  $w_i$  also nicht das gesuchte Target Wort war, ist  $e_i$  positiv. In diesem Fall wird abhängig von der Schrittweite  $\eta$  ein Bruchteil von  $\mathbf{h}$  vom Vektor  $\mathbf{v}'_{w_i}$  subtrahiert und die Distanz

zwischen den beiden Embeddings  $\mathbf{v}'_{w_i}$  und  $\mathbf{h} = \mathbf{v}_{w_c}^T$  vergrößert sich.

Falls das Wort  $w_i$  aber das gesuchte Target Wort war, ist  $e_i$  negativ. Damit nähert sich  $\mathbf{v}_{w_i}^{(new)}$  an  $\mathbf{v}_{w_c}$  an und die beiden Embeddings von Kontextwort und Target Wort werden 'ähnlicher' <sup>7</sup>.

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

$W'^i$ (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

$W'^i$ (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  $W'$  angepasst, da auch jeder Wert auf die Berechnung von  $y_{pred}$  Einfluss hat. Am stärksten werden die Werte für das Wort effektive Target Wort 'klettert' verändert, da der Fehler an dieser Position mit Abstand am grössten ist.

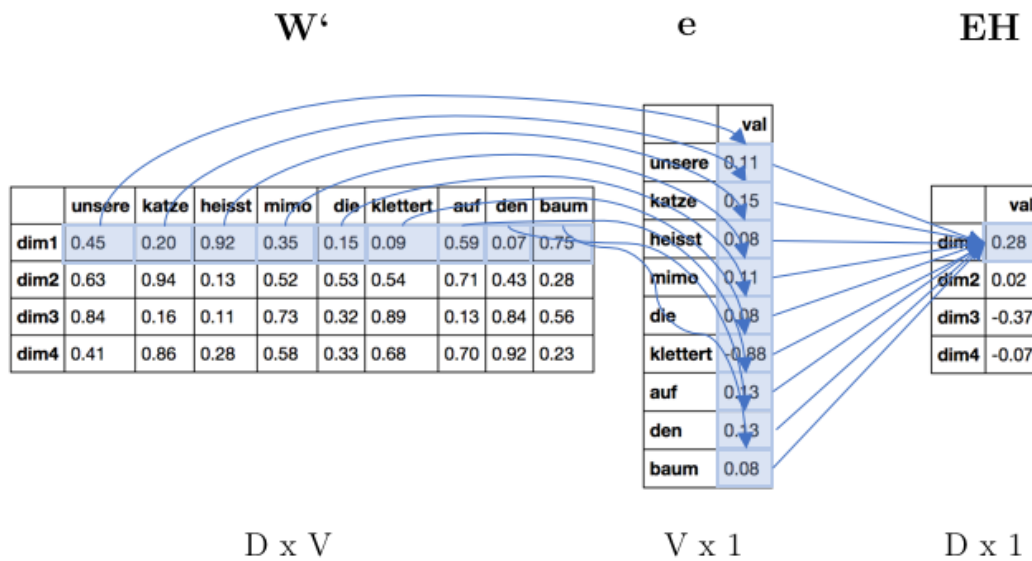
Die Gleichung für die Anpassung der Gewichte von  $W$  lautet

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

für  $W$ , wobei  $\mathbf{E} \mathbf{H}$  die Summe der Vektoren aller Worte im Vokabular ist, gewichtet mit ihrem Prediction Fehler  $e_j = y_{pred_j} - t_j$

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

<sup>7</sup>Mit Ähnlichkeit ist hier das innere Produkt der beiden Vektoren gemeint.



Diese Anpassung kann als Korrektur des Kontext Embeddings des aktuellen Input Worts interpretiert werden, in welche von jedem möglichen Output Wort ein Bruchteil seines Target Embeddings einfließt.

Je grösser der Fehler in der Prediction eines bestimmten Wortes ist, desto stärker fließt sein Gewicht des Embeddings als Target Wort ein. Je nachdem, ob es sich dabei um das effektive Target Wort handelt, ist die Korrektur negativ oder positiv.

Falls es sich bei der Prediction für ein Target Wort also um eine falsche, 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 Target Wort, so ist der Fehler  $\leq 0$  und dies führt dazu, dass die das Context Embedding des aktuellen Inputworts sich an das Target Embedding des Target Worts 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.

$\mathbf{W}^{(old)}$					$\mathbf{W}^{(new)}$					$\mathbf{W}^{(delta)}$				
	dim1	dim2	dim3	dim4		dim1	dim2	dim3	dim4		dim1	dim2	dim3	dim4
unsere	0.88	0.92	0.38	0.96	unsere	0.88	0.92	0.38	0.96	unsere	0.00	0.00	0.00	0.00
katze	0.39	0.60	0.26	0.79	katze	0.33	0.60	0.33	0.81	katze	-0.06	-0.00	0.07	0.01
heisst	0.94	0.13	0.93	0.59	heisst	0.94	0.13	0.93	0.59	heisst	0.00	0.00	0.00	0.00
mimo	0.87	0.57	0.74	0.43	mimo	0.87	0.57	0.74	0.43	mimo	0.00	0.00	0.00	0.00
die	0.89	0.57	0.27	0.63	die	0.89	0.57	0.27	0.63	die	0.00	0.00	0.00	0.00
klettert	0.27	0.44	0.30	0.83	klettert	0.27	0.44	0.30	0.83	klettert	0.00	0.00	0.00	0.00
auf	0.11	0.27	0.36	0.20	auf	0.11	0.27	0.36	0.20	auf	0.00	0.00	0.00	0.00
den	0.55	0.01	0.95	0.08	den	0.55	0.01	0.95	0.08	den	0.00	0.00	0.00	0.00
baum	0.89	0.58	0.34	0.81	baum	0.89	0.58	0.34	0.81	baum	0.00	0.00	0.00	0.00

Durch diese Korrekturen an den Gewichten hat sich im Endeffekt die Ähnlichkeit zwischen den Embeddings für 'katze' bei den Kontext Embeddings und 'klettert' bei den Target Embeddings angenähert.

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

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

Durch die Iterationen über sämtliche Dokumente und das Verschieben der Positionen innerhalb des Dokuments über die entsprechenden Tokens werden die Embeddings so gelernt, dass das Netzwerk mit der Zeit immer besser in der Lage ist, das entsprechende Target Wort vorauszusagen.

## 2.2 word2vec Architekturen: CBOW und Skip-Gram

word2vec, wie es in Mikolov et al. [2013a] vorgestellt wurde, unterstützt zwei verschiedene Architekturen. Die beiden Architekturen basieren auf dem gleichen Grundprinzip. Sie variieren aber bezüglich der unterschiedlichen Grössen des Sliding Windows auf Seiten des Kontexts wie auch auf Seiten der Target Wörter, welche vorausgesagt werden.

In den folgenden Abschnitten werden die beiden Architekturen kurz erklärt und die Auswirkungen der Änderungen verglichen mit der vereinfachten Beispielarchitektur mit einem Single Word Kontext aufgezeigt.

## 2.2.1 CBOW

Die Architektur Continuous Bag of Words, kurz CBOW, weicht von der vereinfachten Beispielarchitektur insofern ab, dass der Kontext, welcher für die die Prediction des Target Wortes verwendet wird, nicht aus einem einzigen Wort besteht.

Die Grundidee ist, dass innerhalb des Dokuments die  $c$  Wörter vor und nach dem Target Wort 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 Target Wort vorausszusagen. Die Reihenfolge der Kontext Wörter wird hierbei nicht berücksichtigt.

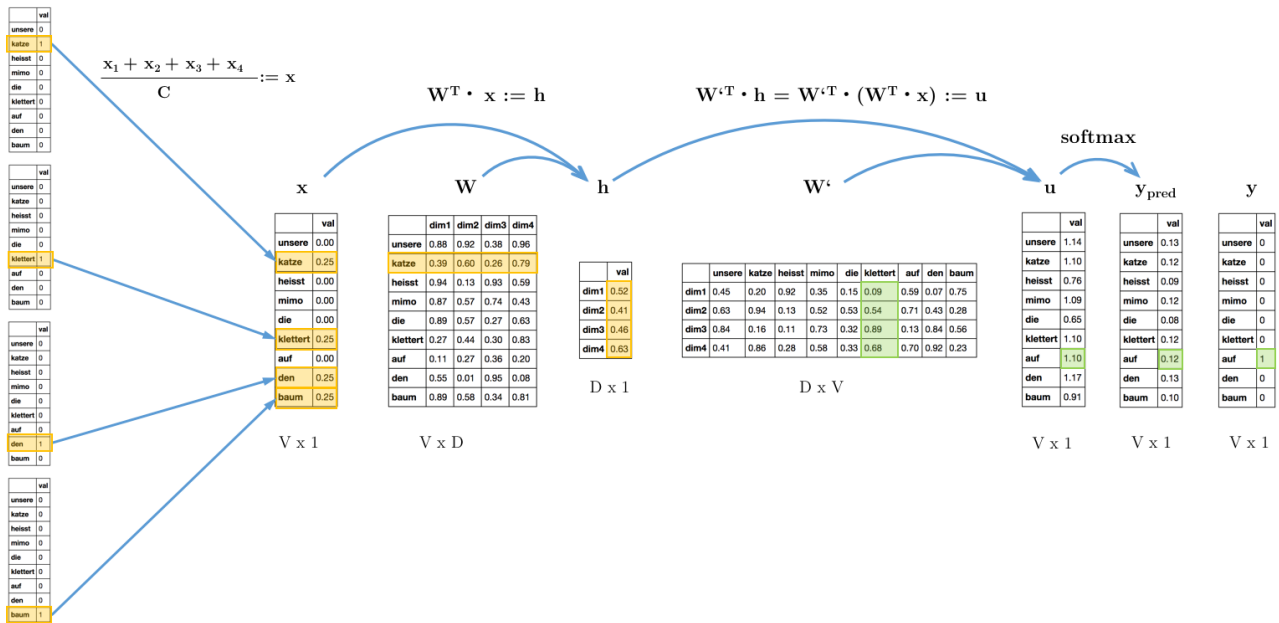
Kontext Wort $w_{t-2}$	Kontext Wort $w_{t-1}$	Target Wort $w_t$	Kontext Wort $w_{t+1}$	Kontext Wort $w_{t+2}$
---------------------------	---------------------------	----------------------	---------------------------	---------------------------

Auch hier wird mit einem Sliding Window gearbeitet, das die entsprechende Grösse hat. Die Grösse des Fensters ist ein Modellparameter der CBOW Architektur, wobei window size  $C = 2$  bedeutet, dass vor und nach dem Target Wort je zwei Wörter als Kontext verwendet werden.

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

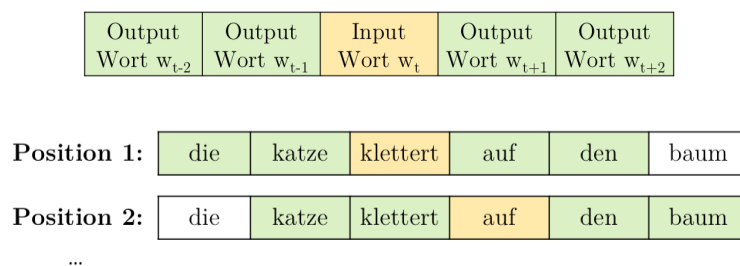
Die einzige Änderung der vereinfachten Single-Word-Context Architektur ist ein zusätzlicher Schritt, der benötigt wird, um das Mittel über die One-Hot-Encodings der Kontext Wörter zu bilden.

Ansonsten ist der Algorithmus und damit auch die Logik für den Forward- wie auch den Backward Pass identisch wie bei der Single-Word-Context Architektur.

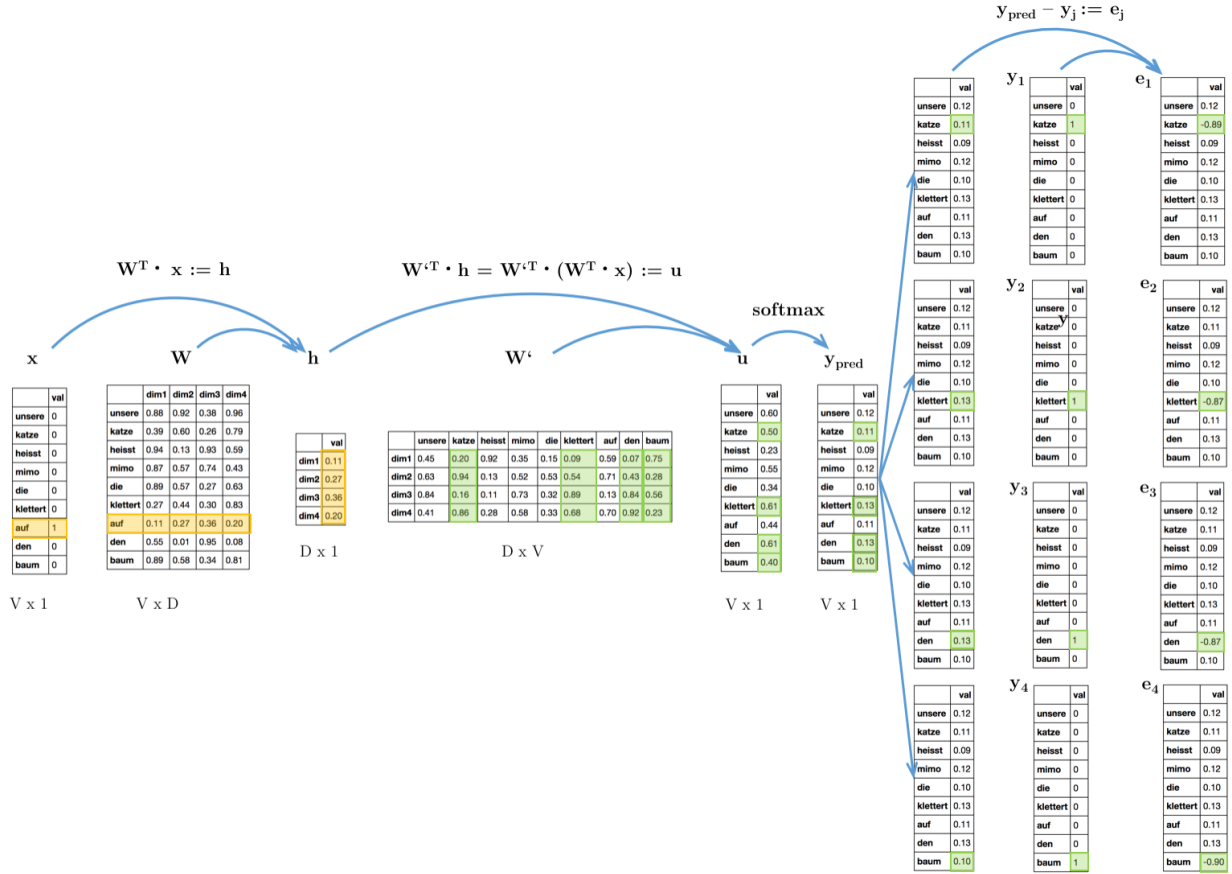


## 2.2.2 Skip-Gram

Die Skip-Gram Architektur stellt im Prinzip eine Umkehrung der CBoW Architektur dar. Als Kontext wird nun wieder - gleich wie bei der Single-Word-Kontext Architektur - mit einem einzigen Kontextwort gearbeitet. Der Algorithmus sagt nun aber nicht ein einzelnes Wort voraus, sondern wiederum ein Fenster von Wörtern rund um das Kontextwort. Der Begriff Kontextwort ist für diese Architektur etwas verwirrend, weil die Wörter, welche der Algorithmus voraussagen soll, eigentlich den Kontext des Input Worts darstellen. Ich verwende deshalb in diesem Zusammenhang die Terminologie des Input Worts und der Output Wörter, welche vorausgesagt werden sollen.



Dies bedingt Anpassungen im Vergleich zur Single-Word-Context Architektur, indem der Prediction Vektor  $y_{pred}$  konzeptionell so oft repliziert wird, wie es für die Grösse des Fensters notwendig ist. Daraus errechnen sich dann sovieler Fehlervektoren  $e_j$ , für alle  $j = 1, 2, \dots, C$ .



Die Berechnungen für die Backward Propagation ändern sich insofern, als nun die verschiedenen Fehlervektoren benutzt werden müssen. So ergibt sich für die Anpassung der Gewichte von  $W'$  neu die Gleichung

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

wobei  $EI$  für die Summe der Fehlervektoren aller vorausszusagenden 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

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



für  $W$ , 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  $W$  bzw.  $W'$  und damit der Word Embeddings, wird als Loss Funktion

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

verwendet, welche dann mittels Gradient Descent minimiert wird.

Die Berechnung des zweiten Terms der Lossfunktion ist bezüglich des Berechnungsaufwandes problematisch, da die Funktion  $e^{v_{w_k}'^T v_{w_c}}$  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 ajustiert werden müssen. Die Matrix  $W'$  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  $10^2$ .

Bei einem konkreten Setting mit  $V = 50'000$  und  $D = 1000$  haben die beiden Gewichtsmatrizen 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 Performance entsprechende Auswirkungen hat.

Es sind also Optimierungen notwendig, um die Algorithmen überhaupt praktikabel zu machen. In diesem Kapitel werden die Optimierungen präsentiert, wie sie in Mikolov et al. [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 Funktion basiert darauf, dass die Resultate der Softmax Funktion nicht in einem flachen Layer abgelegt werden, sondern innerhalb einer

Baumstruktur. Die Knoten enthalten dabei die Wörter des Vokabulars.

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

Die Wahrscheinlichkeit für einen bestimmten Knoten, d.h. das Resultat der Prediction, mit welcher Wahrscheinlichkeit dieses Wort  $w_i$  als Output Wort 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 nun 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  $W'$  entspricht nun nicht mehr einem Wort des Outputs, 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 nun 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 Output Worts basierend auf dem Input wird durch die Softmax Funktion berechnet:

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

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 Output Wort gegeben das Input Wort?”

lautet die Fragestellung neu

”Stammt diese Kombination von Input und Output Wort 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 zum Korpus gehört. Weil es sich um eine binäre Entscheidung handelt, kann die Wahrscheinlichkeit mittels einer Sigmoid Funktion modelliert werden:

$$P(y = 1|w_I, w_O) = \sigma(v_{w_I}^T v_{w_O})$$

Diese Wahrscheinlichkeit kann nun basierend auf dem aktuellen Input Wort  $w_I$  für jedes Output Wort  $w_O$  berechnet werden. Damit diese Performance Optimierung aber den gewünschten Effekt erzielt, wird die Wahrscheinlichkeit nicht für jedes Wort des Vokabulars berechnet, sondern nur für ein bestimmtes Set.

Das Set von Output Wörtern, welche als Tupel zusammen mit dem Input Wort für das Training benützt werden, wird folgendermassen konstruiert. Das korrekte Tupel mit dem aktuellen Input Wort und dem aktuellen Output Wort wird für das Training benützt und mit dem Label 1 versehen.

Zusätzlich werden Tupels konstruiert mit Output Wörtern, die nicht dem korrekten aktuellen Output Wort entsprechen, und mit 0 als Label gekennzeichnet. Die Anzahl von solchen "Negative Samples"  $k$  ist ein Modellparameter. Die Empfehlung gemäss Mikolov et al. [2013b] schlägt für  $k$  Werte zwischen 5 und 20 für kleine Trainingsdatensets vor und 2 bis 5 für grosse Datensets.

Input Wort $w_I$	Output Wort $w_O$	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  $W$  und  $W'$  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 Optimierung kann durch die Berechnung dieser 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 die Skip-Gram Architektur 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. Im Beispiel wäre das sicher das Wort "die".

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. Für 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  $f(w_i) > k$  sehr aggressiv weglässt, während die Verteilung für die übrigen Wörter sehr 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 Vorbereitungsschritt solche Phrasen erkannt und die Wörter dann 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 Trainingsdaten 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 et al. [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 dann 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 et al. [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, prediction-based 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 der ähnlichste 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"), 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  $\vec{Brüder} - \vec{Schwester} + \vec{Enkel}$ .

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.<sup>8</sup>

In Baroni et al. [2014] stellen die Autoren verschiedene count-based und prediction-based Modelle bezüglich dieser Evaluierungsmethoden gegenüber. Zu ihrem Erstaunen, wie sie selber sagen, schneiden die prediction-based Modelle in fast allen Qualitätsmetriken klar besser gegenüber den Word Embeddings, die mit count-based Modellen erstellt wurden. Die einzige Evaluierungsmethode, bei welcher die count-based Modelle word2vec überlegen waren, ist die Selectional Preference Metrik. Das ist im Besonderen deshalb überraschend, da die count-basierten Modelle zum Teil unter Beizug von externen Quellen und manuellen Regeln arbeiteten, während word2vec einfach in der Standardausführung mit vorgeschlagenen Parameterwerten eingesetzt wurde.

Dieses Ergebnis hat dann schliesslich auch 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 count-based) und basierend auf diesen Informationen mittels linearer Regression die Word Embeddings lernt.

Die in Pennington et al. [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.

---

<sup>8</sup>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 Maning ist, dass keine Alternative dazu existiert, die Gesamtqualität exemplarisch an einigen (wenigen) Instanzen zu überprüfen.



## **4 Zusammenfassung**

# References

- M. Baroni, G. Dinu, and G. Kruszewski. 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)*, pages 238–247, Baltimore, Maryland, June 2014. Association for Computational Linguistics. URL <http://www.aclweb.org/anthology/P14-1023>.
- Y. Goldberg. *Neural Network Methods for Natural Language Processing*. Number 10 in Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2017. doi: 10.2200/S00762ED1V01Y201703HLT037.
- Y. Goldberg and O. Levy. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *CoRR*, abs/1402.3722, 2014. URL <http://arxiv.org/abs/1402.3722>.
- O. Levy and Y. Goldberg. 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*, pages 171–180, 2014. URL <http://aclweb.org/anthology/W/W14/W14-1618.pdf>.
- C. Manning. Lecture 2 word vector representations: word2vec, 2017. URL <https://youtu.be/ERibwqs9p38?t=32m27s>.
- T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013a. URL <http://arxiv.org/abs/1310.4546>.
- T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013b. URL <http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf>.

- J. Pennington, R. Socher, and C. D. Manning. 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*, pages 1532–1543, 2014. URL <http://aclweb.org/anthology/D/D14/D14-1162.pdf>.
- X. Rong. word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*, 2014.
- S. Ruder. On word embeddings - part 2: Approximating the softmax, 2016. URL <http://ruder.io/word-embeddings-softmax>.
- T. Schnabel, I. Labutov, D. M. Mimno, and T. Joachims. Evaluation methods for unsupervised word embeddings. In L. Màrquez, C. Callison-Burch, J. Su, D. Pighin, and Y. Marton, editors, *EMNLP*, pages 298–307. The Association for Computational Linguistics, 2015. ISBN 978-1-941643-32-7. URL <http://dblp.uni-trier.de/db/conf/emnlp/emnlp2015.html#SchnabelLMJ15>.