

Syntax natürlicher Sprachen

Parsing mit neuronalen Netzwerken

Martin Schmitt

Ludwig-Maximilians-Universität München

29.01.2019

Acknowledgment

Inspiriert von ESSLLI 2018 Kurs „Neural Dependency Parsing of morphologically rich languages“ von Daniël de Kok & Erhard Hinrichs.

European Summer School of Language, Logic, and Information
esslli2018.folli.info

[http://esslli2018.folli.info/
neural-dependency-parsing-of-morphologically-rich-languages/](http://esslli2018.folli.info/neural-dependency-parsing-of-morphologically-rich-languages/)

- Probabilistisches Modell für die Wahl der nächsten Operation
- Merkmale müssen definiert und als Vektoren kodiert werden
- Durch gelernte Gewichte werden Scores für jede Operation berechnet (z. B. lineares Modell $\mathbf{W}\vec{x}$)
- Softmax-Funktion normalisiert Scores zu Wahrscheinlichkeiten
- Training durch Minimierung einer Kostenfunktion (z. B. negative log likelihood)

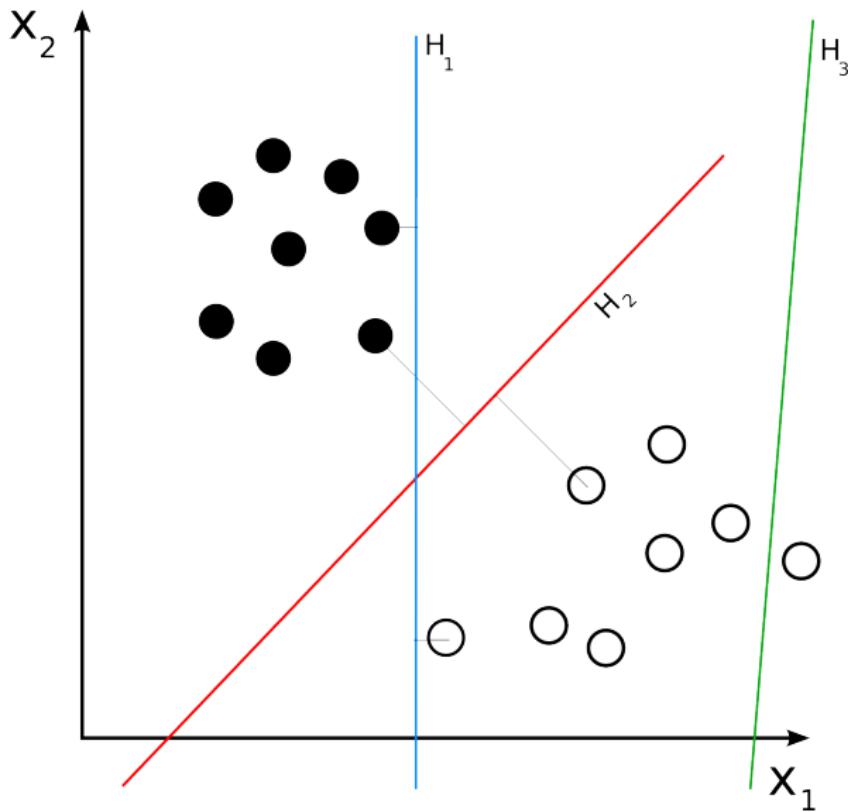
Themen der heutigen Vorlesung

1 Grenzen von linearen Klassifikatoren

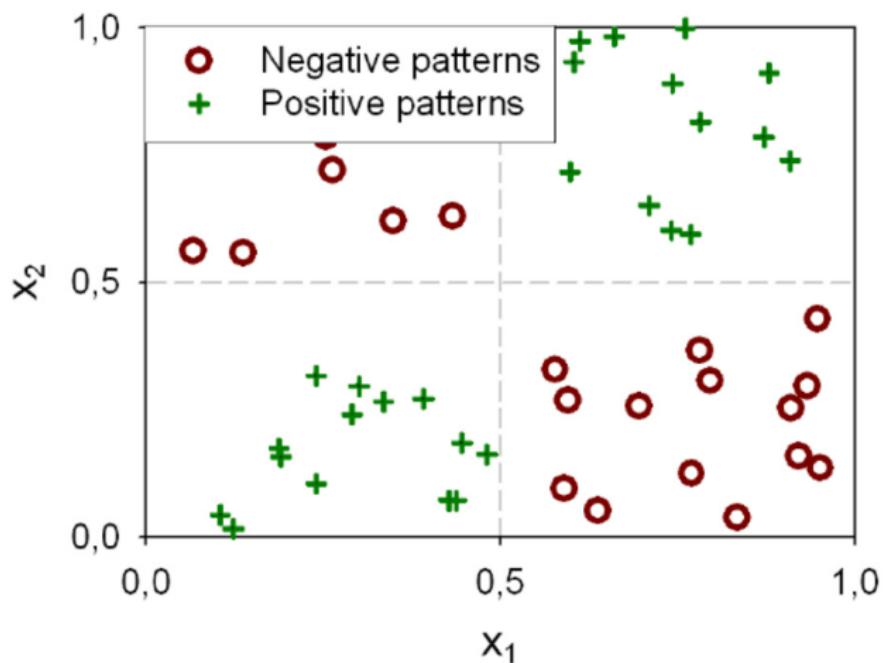
2 Feed-forward Neural Networks

3 Word Embeddings

Beispiel: Klassifikation mit zwei Features



Klassifikation von xor (Minski und Papert, 1969)



Grafik aus (Ferrari und Muselli, 2008)

Linguistisches Beispiel

Nehmen wir an, wir haben einen Shift-Reduce Dependency-Parser mit folgenden Merkmalen:

- ① $stack = [\dots, \text{Autos}, _]$
- ② $stack = [\dots, \text{hupen}]$
- ③ $stack = [\dots, \text{wir}, _]$
- ④ $stack = [\dots, \text{kaufen}]$

Typische Transitionen:

- $[1 \ 1 \ 0 \ 0]$: Left-Arc_{subj}
- $[0 \ 0 \ 1 \ 1]$: Left-Arc_{subj}

Gelernte Gewichtung ist z. B. $w = [1 \ 1 \ 1 \ 1]$

Linguistisches Beispiel (II)

① $stack = [\dots, \text{Autos}, _]$

② $stack = [\dots, \text{hupen}]$

③ $stack = [\dots, \text{wir}, _]$

④ $stack = [\dots, \text{kaufen}]$

$$w = [1 \ 1 \ 1 \ 1]$$

Was passiert bei folgendem Feature-Vektor?

$$[1 \ 0 \ 0 \ 1]$$

- Hohe Wahrscheinlichkeit für Left-Arc_{subj}
 - Das ist allerdings klar falsch!
- ⇒ Lineares Modell nicht ausdrucksstark genug für diese Abhängigkeiten.

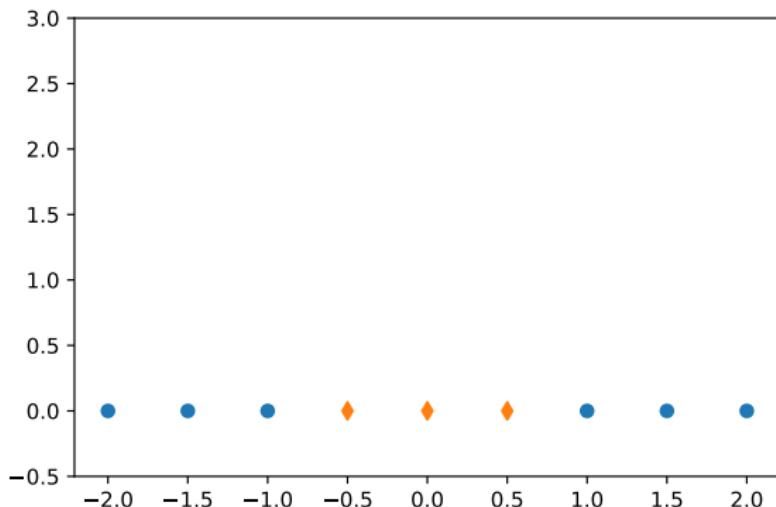
Workaround: Higher Order Features

- ① $stack = [\dots, \text{Autos}, \text{hupen}]$
- ② $stack = [\dots, \text{Autos}, \text{kaufen}]$

Probleme:

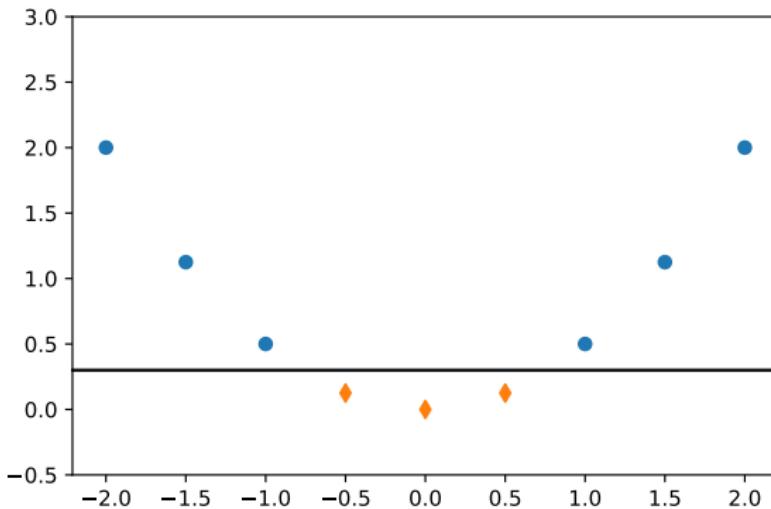
- Wesentliche Vergrößerung des Merkmalsraums
⇒ schwieriger gute Modelle zu trainieren
- Mehr Feature Engineering nötig: welche Merkmale sollten kombiniert werden, welche nicht?
- Ein besseres Modell würde solche Interaktionen automatisch erkennen.

Einführung



- Daten nicht linear separierbar (nicht durch eine Linie trennbar)
- Können wir eine Transformation anwenden, um die Daten separierbar zu machen?

Einführung (II)



- Transformation mittels einer quadratischen Funktion
⇒ Linear separierbar!

Non-lineare Transformationen

$g(\mathbf{W}\vec{x})$ ist eine non-lineare Transformation von \vec{x} , wobei

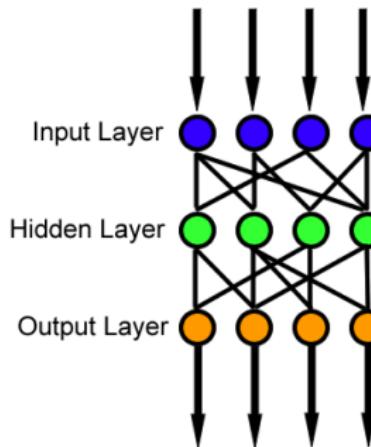
- $\vec{x} \in \mathbb{R}^i$
- $\mathbf{W} \in \mathbb{R}^{h \times i}$: Gewichtsmatrix (wie bei linearem Modell), h ist Hyperparameter (*hidden size*)
- $g : \mathbb{R}^n \rightarrow \mathbb{R}^n$: nicht-lineare Funktion (elementweise)

$g(\mathbf{W}\vec{x})$ wird bei neuronalen Netzwerken versteckte Schicht (*hidden layer*) genannt.

Feed-forward Neural Network

Ein *Feed-forward Neural Network* kombiniert eine oder mehrere von diesen Schichten und eine Ausgabeschicht (wie *softmax*).

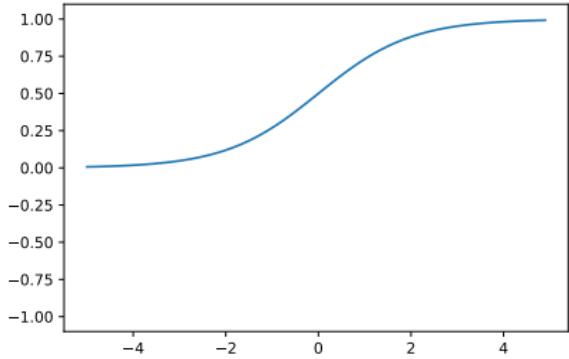
- 1 hidden layer : $\text{softmax}(\mathbf{W}_0 g(\mathbf{W}_1 \vec{x}))$
- 2 hidden layers : $\text{softmax}(\mathbf{W}_0 g(\mathbf{W}_1 g(\mathbf{W}_2 \vec{x})))$
- 3 hidden layers : $\text{softmax}(\mathbf{W}_0 g(\mathbf{W}_1 g(\mathbf{W}_2 g(\mathbf{W}_3 \vec{x}))))$



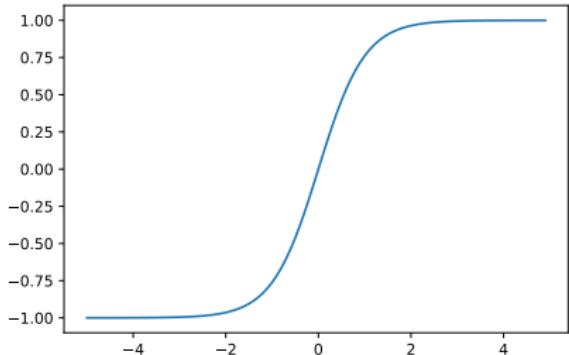
Credits: Wikipedia User Paskari. Lizenz: CC BY-SA 3.0

Non-lineare Funktionen (Aktivierungsfunktionen)

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

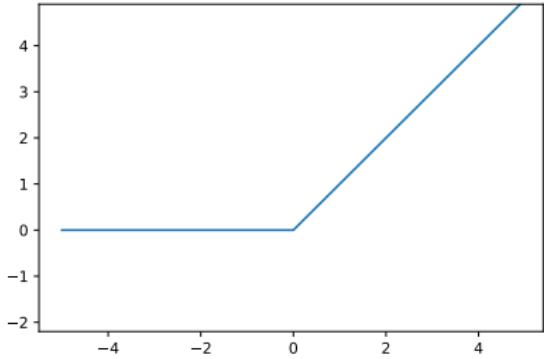


$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

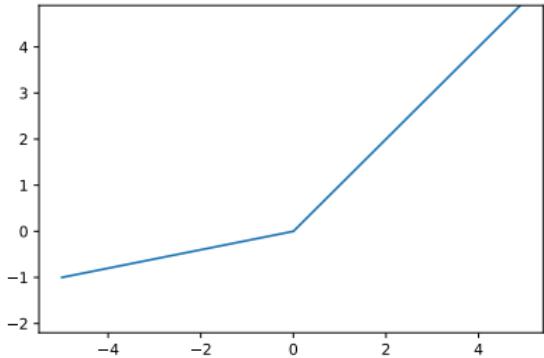


Non-lineare Funktionen (Aktivierungsfunktionen)

$$\text{ReLU}(x) = \max(0, x)$$



$$\text{Leaky ReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ ax & \text{if } x < 0 \end{cases}$$



Non-lineare Klassifikation und Parsing

- ① $stack = [\dots, \text{Autos}, _]$
- ② $stack = [\dots, \text{hupen}]$
- ③ $stack = [\dots, \text{wir}, _]$
- ④ $stack = [\dots, \text{kaufen}]$

Können wir mit einem nicht-linearen Klassifikator unser Problem lösen?

Betrachte ein einfaches Modell $softmax(\mathbf{W}_0 g(\mathbf{W}_1 \vec{x}))$ mit folgenden Gewichten für die versteckte Schicht:

$$\mathbf{W}_1 = \begin{bmatrix} 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \end{bmatrix}$$

Autos hupen

- ① $stack = [\dots, \text{Autos}, _]$
- ② $stack = [\dots, \text{hupen}]$
- ③ $stack = [\dots, \text{wir}, _]$
- ④ $stack = [\dots, \text{kaufen}]$

$$\begin{aligned} & \text{ReLU} \left(\begin{bmatrix} 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \right) \\ &= \text{ReLU} \left(\begin{bmatrix} 2 \\ -2 \end{bmatrix} \right) \\ &= \begin{bmatrix} 2 \\ 0 \end{bmatrix} \end{aligned}$$

wir kaufen

- ① $stack = [\dots, \text{Autos}, _]$
- ② $stack = [\dots, \text{hupen}]$
- ③ $stack = [\dots, \text{wir}, _]$
- ④ $stack = [\dots, \text{kaufen}]$

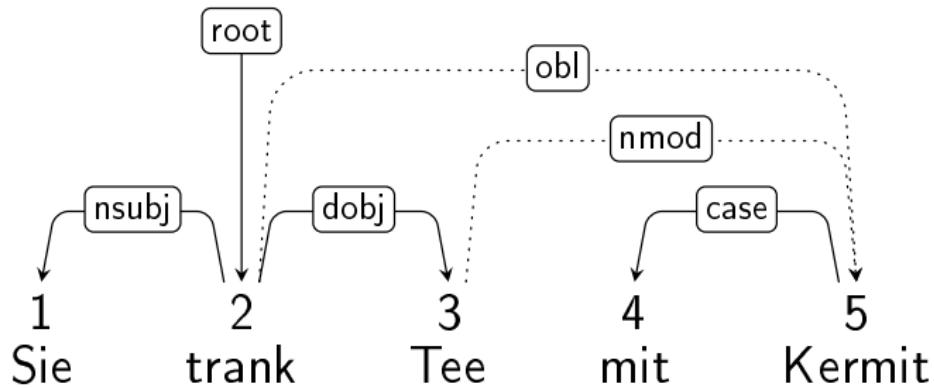
$$\begin{aligned} & \text{ReLU} \left(\begin{bmatrix} 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \right) \\ &= \text{ReLU} \left(\begin{bmatrix} -2 \\ 2 \end{bmatrix} \right) \\ &= \begin{bmatrix} 0 \\ 2 \end{bmatrix} \end{aligned}$$

Autos kaufen

- ① $stack = [\dots, \text{Autos}, _]$
- ② $stack = [\dots, \text{hupen}]$
- ③ $stack = [\dots, \text{wir}, _]$
- ④ $stack = [\dots, \text{kaufen}]$

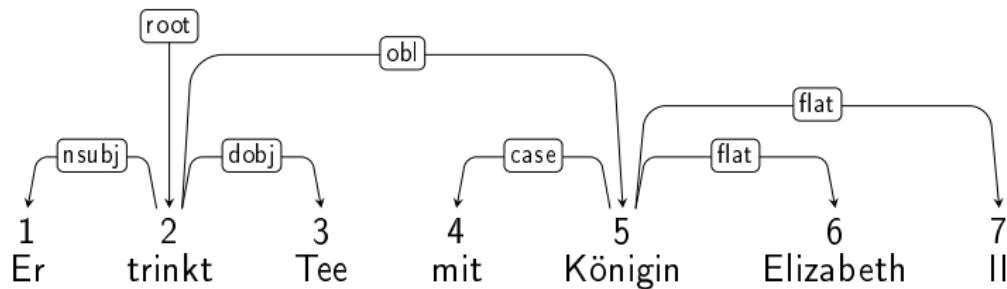
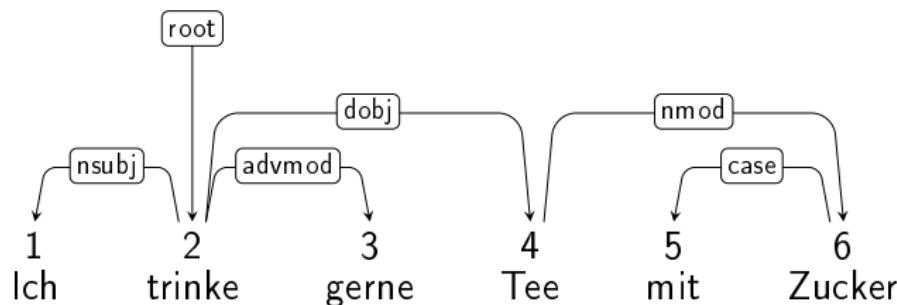
$$\begin{aligned} & \text{ReLU} \left(\begin{bmatrix} 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \right) \\ &= \text{ReLU} \left(\begin{bmatrix} 0 \\ 0 \end{bmatrix} \right) \\ &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} \end{aligned}$$

Motivation



Trainingsdaten

Können wir von ähnlichen Sätzen lernen, welche Variante korrekt ist?



Wie lernen wir, dass *Kermit* und *Elizabeth* ähnlicher sind als *Kermit* und *Zucker* ?

Wörter als Features (traditionell)

- Wörter wurden traditionell als unabhängige Symbole repräsentiert
- Beispiel: *one-hot* Vektoren
 - Vektoren aus $\{0, 1\}^{|V|}$
 - Alle Einträge außer einem (dem Wortindex) sind 0

$$v_{\text{Tee}} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{bmatrix} \quad v_{\text{Zucker}} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{bmatrix} \quad v_{\text{Königin}} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{bmatrix} \quad \dots \quad v_{\text{Elizabeth}} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ 1 \end{bmatrix}$$

⇒ Alle Wortvektoren sind orthogonal zueinander!

Probleme des traditionellen Ansatzes

- Sehr große Vektoren (normalerweise mehrere 100k)
 - benötigen viel Speicher.
 - machen Berechnungen aufwändig.
- Modelle benötigen viele Parameter; daher auch viele Trainingsinstanzen, um zu lernen.
- Morphologische Formen des gleichen Lemmas sind völlig unabhängig voneinander.
- Ähnliche Wörter (syntaktisch, semantisch) sind völlig unabhängig voneinander.
- Es ist unmöglich, von ähnlichen Beispielen aus den Trainingsdaten zu lernen.

Word Embeddings

- *Word Embeddings* sind Wortvektoren mit weniger Dimensionen.
- Sie werden so gelernt, dass (syntaktisch, semantisch) ähnliche Wörter ähnliche Vektoren haben.
- Sie basieren auf distributionellen Informationen (Kookkurrenz).
- *Distributional Hypothesis: You shall know a word by the company it keeps.* — J. R. Firth

The Distributional Hypothesis: Beispiel

Zum Schluss gibt ihr noch die Milch und die Eier in das *Suzifie* und röhrt alles gut durch.



Da tauchte plötzlich ein kleines, haariges *Suzifie* auf, rannte zu der Nuss und verschwand damit in seiner Höhle.



Kookkurrenz von Milch und Zucker

Milch		Zucker	
Nachbarwort	Anzahl	Nachbarwort	Anzahl
Wasser	48	Wasser	51
Käse	42	Stärke	41
Zucker	40	Milch	32
Butter	39	Salz	30
Fleisch	36	Liter	28
Sahne	28	Honig	28
Eiern	27	Mehl	27
...		...	

(Quelle: dt. Wikipedia, Januar 2018; Fenstergröße: 5, eingeschränkt auf Nomen)

⇒ Nahrungsmittel-Kontexte !

Kookkurrenz von Kermit und Elizabeth

Kermit		Elizabeth	
Nachbarwort	Anzahl	Nachbarwort	Anzahl
Roosevelt	11	Taylor	184
Frosch	10	Queen	149
Robert	6	Tochter	132
Driscoll	6	Königin	90
Sohn	5	Port	85
Ruffins	4	Frau	82
Star	3	Mary	75
...		...	

(Quelle: dt. Wikipedia, Januar 2018; Fenstergröße: 5, eingeschränkt auf Nomen)

⇒ Personen-Kontexte !

Word Embedding Modelle (I)

Es gibt viele verschiedene Modelle, die Word Embeddings aus großen Textkorpora lernen können:

- word2vec
 - CBOW: ausgehend von allen Kontextwörtern versuche, das Wort in der Mitte vorherzusagen
 - Skip-gram: ausgehend vom Wort in der Mitte versuche, alle Kontextwörter (einzelnen) vorherzusagen
- GloVe: (1) Stelle globale Kookkurrenzmatrix auf. (2) Optimiere Wortvektoren, sodass sie so gut wie möglich diese Kookkurrenzen wiedergeben.

Salz , Pfeffer und Zucker werden noch Butter und
5 Token (Fenstergröße)

Word Embedding Modelle (II)

FastText

Wie word2vec; zusätzlich lerne auch Vektoren für Untersequenzen der Wörter (*character n-grams*). Finales Embedding ist dann Kombination aus Gesamtwort-Vektor und Vektoren der Untersequenzen. (Morphologie)

Beispiel mit 3-5-grams:

- <Fr, Fre, rei, eis, isi, sin, ing, ng>
- <Fre, Frei, reis, eisi, isin, sing, ing>
- <Frei, Freis, reisi, eising, ising>

Word Embedding Modelle (III)

Es gibt noch viele Varianten!

- Beziehe Position mit ein: Wang2vec !
- Nutze syntaktischen Kontext: word2vecf !
- Beziehe auch Ton und Bilder mit ein: multimodale Embeddings !
- Trainiere Embeddings über mehrere Sprachen hinweg:
multilinguale Embeddings !

⇒ Offenes Thema für weitere Forschung!

Zusammenfassung: Neuronale Netzwerke

- Können Merkmale selbst lernen (weniger Feature Engineering nötig)
- Word Embeddings generalisieren über unbekannte, aber zu bekannten Wörtern ähnliche Wörter und Wortformen
- Können komplexe (nicht-lineare) Muster und Wechselwirkungen zwischen Merkmalen erkennen
- Benötigen tendenziell sehr große Datenmengen
- Viele Hyperparameter zu wählen (nicht einfach zu optimieren)
- Ausblick: Rekurrente neuronale Netze (RNN) beziehen den Sequenzcharakter eines Satzes mit ein!
- Es gibt noch (viel) mehr Architekturen!

Literaturverzeichnis

- ① Minski, Marvin, & Papert, Seymour. (1969). Perceptrons: An Introduction to Computational Geometry.
- ② Ferrari, Enrico & Muselli, Marco. (2008). A Multivariate Algorithm for Gene Selection Based on the Nearest Neighbor Probability.