# Syntax natürlicher Sprachen

## Parsing with neural networks

Lütfi Kerem Senel

Ludwig-Maximilians-Universität München

02.02.2022

# Acknowledgment

Inspired by ESSLLI 2018 course „Neural Dependency Parsing of morphologically rich languages“ by 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/

# Topics of today's lecture

1. Statistical Parsing
2. Limits of linear classifiers
3. Feed-forward Neural Networks
4. Word Embeddings

Section 1

# Statistical Parsing

# Deriving oracles from data I

## oracle

A function that always returns the correct next operation.

## Probabilistic Model as an Oracle

A probability distribution over operations given the current state (of the stack, the read buffer, the analysis so far): $P(op \mid state)$

# Deriving oracles from data I

## Deriving a probabilistic model from data

- Specify the features of the parser state to be used
- given a corpus of sentences and syntax trees, for each sentence create pairs of features and correct operations
  ($\Rightarrow$ training data)
- Optimize the parameters of a statistical probability model in such a way that it makes the right prediction as often as possible.

# Coding of Features

## Binary Features

A binary feature is either present (1) or absent (0).

## Examples

- $f_1 \equiv stack = [\ldots|\text{in}|\text{München}]$
- $f_2 \equiv stack = [\ldots|\text{München}]$ und $buffer = [\text{gewesen}|\ldots]$

## Feature Vectors

- represent all possible characteristics as a sequence of 0 and 1
- enable methods of linear algebra ($\Rightarrow$ model)
- e.g.: $\vec{x} = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix}$

# Softmax-Regression

## Simple Linear Model

Let matrix $W \in \mathbb{R}^{|Op| \times |F|}$ with $Op$ operations and $F$ features.
Each line (operation) weights the features differently.
We calculate scores for each operation by multiplication: $W\vec{x}$

## Normalization by Softmax

$$softmax(\vec{v})_i = \frac{e^{v_i}}{\sum_j e^{v_j}}$$

- Values are between 0 and 1
- $\sum_j softmax(\vec{v})_j = 1$
- $softmax(\vec{v})_{op}$ is the probability for operation $op$

# Training

## Cost/Objective function

Negative log likelihood

$$\mathcal{L}(\mathrm{W}) = -\sum_{i=1}^{n} \log P(y^i|\vec{x}^i)$$

- is trained by minimizing the cost function
- Maximizing goodness is equivalent to minimizing cost
- The difference is a minus sign
- There are numerous libraries that allow automatic optimization of such a function: e.g. Pytorch, Tensorflow, Dynet.

# Statistical vs. Formal Grammar

Statistical

- always returns a result
- often finds a good (probable) solution
- annotators can focus on real linguistic examples
- Very efficient calculations possible

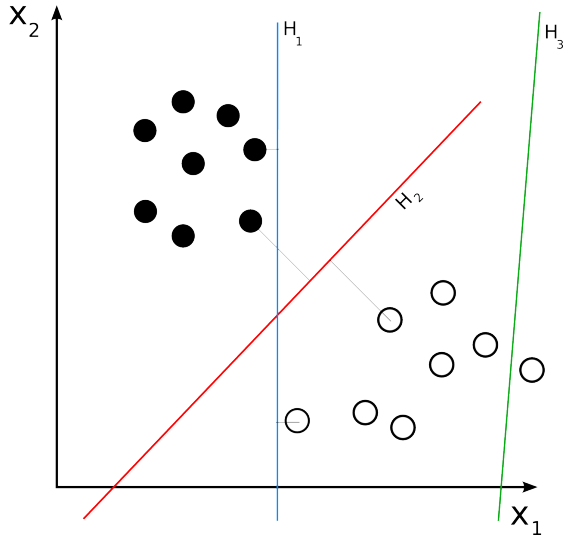
Formal Grammar

- recognizes ungrammatical sentences; but it is also naturally incomplete
- always finds all correct solution(s) licensed by the grammar
- generalization must be planned for by the grammarian
- Exact algorithms require polynomial running time

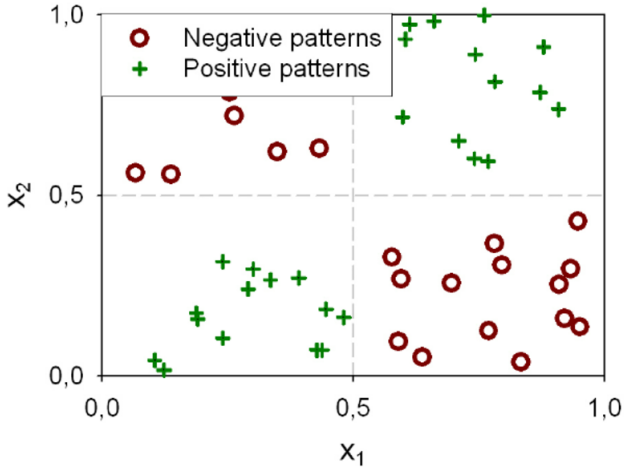
Section 2

# Limits of linear classifiers

# example: classification with two features



# Classification of xor (Minski and Papert, 1969)



# Linguistic Example

Let's say we have a shift-reduce dependency parser with the following characteristics:

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

Typical transitions:

- $\begin{bmatrix} 1 & 1 & 0 & 0 \end{bmatrix}$ : Left-$\text{Arc}_{\text{subj}}$
- $\begin{bmatrix} 0 & 0 & 1 & 1 \end{bmatrix}$ : Left-$\text{Arc}_{\text{subj}}$

Learned weighting is e.g. $\text{w} = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}$

# Linguistic Example (II)

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

$\mathbf{w} = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}$

What happens with the following feature vector?
$\begin{bmatrix} 1 & 0 & 0 & 1 \end{bmatrix}$

- High probability for Left-$\text{Arc}_{\text{subj}}$
- However, this is clearly wrong!

$\Rightarrow$ Linear model not expressive enough for these dependencies.

# Workaround: Higher Order Features

1. $stack = [\ldots, \text{Autos}, \text{hupen}]$
2. $stack = [\ldots, \text{Autos}, \text{kaufen}]$

Problems:

- Significant enlargement of the feature space

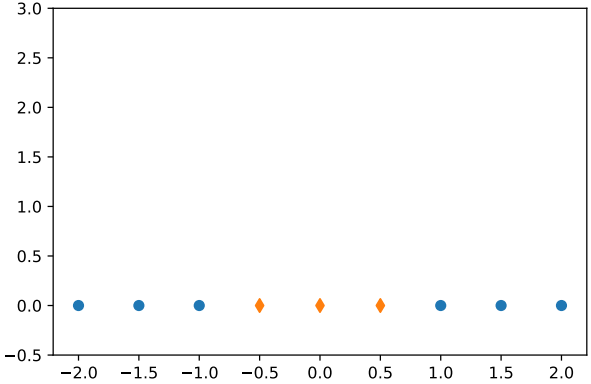
$\Rightarrow$ harder to train good models

- More feature engineering needed: which features should be combined, which not?
- A better model would automatically detect such interactions.

Section 3

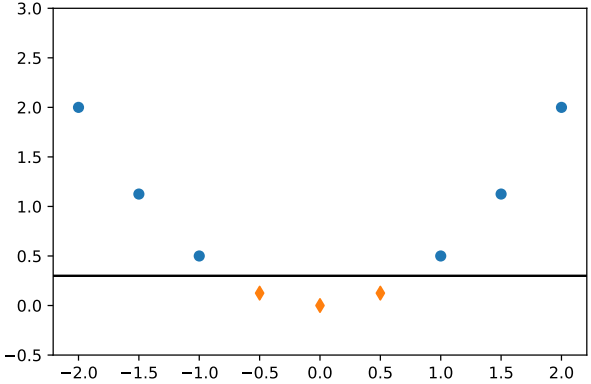
# Feed-forward Neural Networks

# Introduction




- Data not linearly separable (cannot be separated by a line)
- Can we apply a transformation to make the data separable?

# Introduction (II)




- Transformation using a quadratic function

$\Rightarrow$ Linearly separable!

# Non-lineare Transformationen

$g(W\vec{x})$ is a non-linear transformation of $\vec{x}$, where

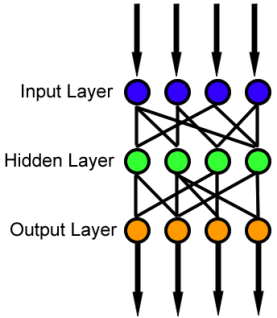
- $\vec{x} \in \mathbb{R}^i$
- $W \in \mathbb{R}^{h \times i}$ : Weight matrix (like linear model), $h$ is hyperparameter (*hidden size*)
- $g : \mathbb{R}^n \rightarrow \mathbb{R}^n$ : non-linear function (element-wise)

$g(W\vec{x})$ is called the hidden layer (*hidden layer*) in neural networks.

# Feed-forward Neural Network

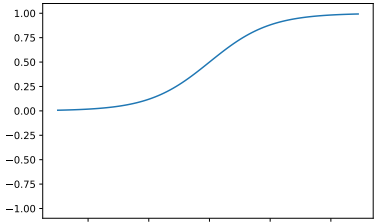
A *Feed-forward Neural Network* combines one or more of these layers and an output layer (like *softmax*).

- 1 hidden layer : $softmax(W_0 g(W_1 \vec{x}))$
- 2 hidden layers : $softmax(W_0 g(W_1 g(W_2 \vec{x})))$
- 3 hidden layers : $softmax(W_0 g(W_1 g(W_2 g(W_3 \vec{x}))))$

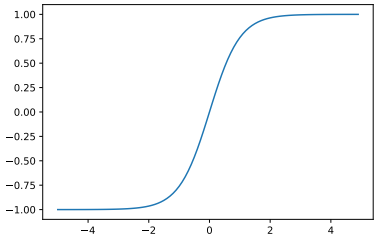


# Non-Linear Functions (Activation Functions)

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

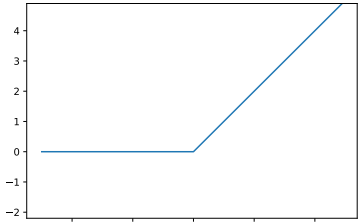



$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

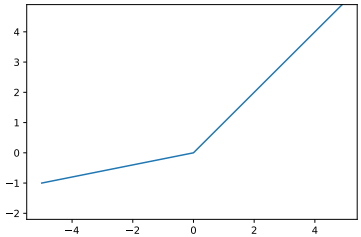


# Non-Linear Functions (Activation Functions)

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




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



# Non-linear classification and parsing

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

Can we solve our problem with a non-linear classifier?

Consider a simple model $softmax(W_0 g(W_1 \vec{x}))$ with the following weights for the hidden layer:

$$W_1 = \begin{bmatrix} 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \end{bmatrix}$$

# Autos hupen

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

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

# wir kaufen

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

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

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

$$= \begin{bmatrix} 0 \\ 2 \end{bmatrix}$$

# Autos kaufen

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

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

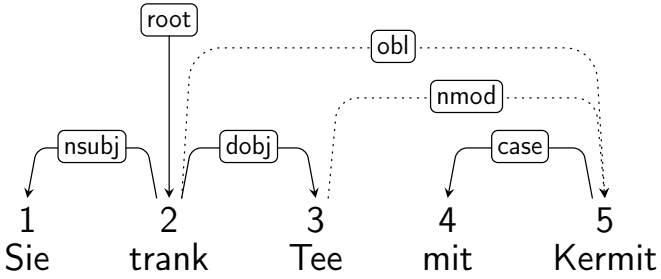
$$= \text{ReLU}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}\right)$$

$$= \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Section 4

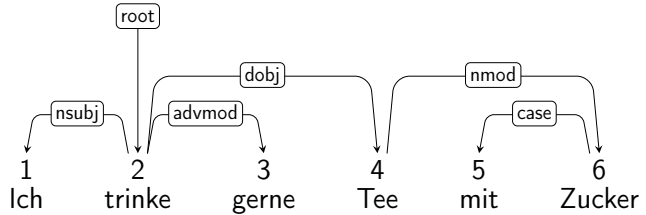
# Word Embeddings

# Motivation



# Rraining data

Can we learn from similar sentences which variant is correct?




How do we learn that *Kermit* and *Elizabeth* are more similar than *Kermit* and *Zucker* ?

# Words as features (traditional)

- Words have traditionally been represented as independent symbols
- Example: *one-hot* vectors
  - Vectors from $\{0,1\}^{|V|}$
  - All items except one (the word index) are 0

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

$\Longrightarrow$ All word vectors are orthogonal to each other!

# Problems of the traditional approach

- Very large vectors (usually several 100k)
  - take up a lot of memory.
  - makes calculations difficult.
- Models take many parameters; $\Rightarrow$ many training instances needed to learn.
- Morphological forms of the same lemma are completely independent of each other.
- Similar words (syntactically, semantically) are completely independent of each other.
- It is impossible to learn from similar examples from the training data.

# Word Embeddings

- *Word Embeddings* are word vectors with fewer dimensions.
- Similar words (syntactically, semantically) should have similar vectors.
- They are based on distributional information (co-occurrence).
- *Distributional Hypothesis*: *You shall know a word by the company it keeps.* — J. R. Firth

# The Distributional Hypothesis: Example

Finally, add the milk and the eggs to the *Suzifie* and stir well.

Suddenly a small, hairy *Suzifie* appeared, ran to the nut and disappeared into his cave with it.

# Co-occurrence of Milch and Zucker

**Milch**

| Neighbor Word | Frequency |
|---|---|
| Wasser | 48 |
| Käse | 42 |
| Zucker | 40 |
| Butter | 39 |
| Fleisch | 36 |
| Sahne | 28 |
| Eiern | 27 |
| ... | |

**Zucker**

| Neighbor Word | Frequency |
|---|---|
| Wasser | 51 |
| Stärke | 41 |
| Milch | 32 |
| Salz | 30 |
| Liter | 28 |
| Honig | 28 |
| Mehl | 27 |
| ... | |

(Source: German Wikipedia, January 2018; window size: 5, limited to nouns)

$\Longrightarrow$ food contexts !

# Kermit and Elizabeth co-occurrence

**Kermit**

| Nachbarwort | Frequenz |
|---|---|
| Roosevelt | 11 |
| Frosch | 10 |
| Robert | 6 |
| Driscoll | 6 |
| Sohn | 5 |
| Ruffins | 4 |
| Star | 3 |
| ... | |

**Elizabeth**

| Nachbarwort | Frequenz |
|---|---|
| Taylor | 184 |
| Queen | 149 |
| Tochter | 132 |
| Königin | 90 |
| Port | 85 |
| Frau | 82 |
| Mary | 75 |
| ... | |

(Source: German Wikipedia, January 2018; Window size: 5, limited to nouns)

$\Longrightarrow$ Person contexts !

# Word embedding models (I)

There are many models that learn word embeddings from large text corpora:

- word2vec
  - CBOW: based on all context words, determine the word in the middle
  - Skip-gram: starting from the word in the middle, determine all context words (individually)
- GloVe:
  (1) Set up global co-occurrence matrix.
  (2) Optimize word vectors so that they reflect these co-occurrences as well as possible.

... außer Salz , Pfeffer und *Zucker* werden noch Butter und zwei Eier ...

5 Token (Fenstergröße)

# Word embedding models (II)

## FastText

Like word2vec; in addition, also learn vectors for subsequences of words (*character n-grams*). The final embedding is then a combination of the complete word vector and the vectors of the subsequences. (Morphology)

Example with 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 models (III)

There are still many variants!

- Include position: Wang2vec !
- Use syntactic context: word2vecf !
- Also include sound and images: multimodal embeddings !
- Train embeddings across multiple languages: multilingual embeddings !

$\Longrightarrow$ Open topic for further research!

# Abstract: Neural Networks

- can learn features themselves (less feature engineering needed)
- generalize with word embeddings about unknown words and word forms that are similar to known words
- can recognize complicated (non-linear) patterns and interactions between features
- tend to require very large amounts of data
- have many hyperparameters (not easy to optimize)
- Outlook: Recurrent neural networks (RNN) include the sequence character of a sentence!
- There are (many) more architectures!

# bibliography


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.