VL Deep Learning for Natural Language Processing

07. Word Embeddings I

Prof. Dr. Ralf Krestel

AG Information Profiling and Retrieval

Semester Schedule

Week	1st Date	2nd Date	Hand-in at start of week	Hand-out at end of Week
1	Introduction	Python/Keras/Colab		
2	NLP	Neural Networks Recap		1. Assignment
3	Text Mining	Text Classification		
4	Word Embeddings I	Word Embeddings II		
5	Word Embeddings III	Assignment 1 Discussion	1. Assignment	
6	Convolutional Models	Named Entity Recognition		2. Assignment
7	Recurrent Models I	Holiday		
8	Recurrent Models II	Deep Learning in Practice		
9	Contextual Word Embeddings	Assignment 2 Discussion	2. Assignment	
10	Sequence-to-Sequence Models	Transformer Models		3. Assignment
11	Question Answering	Enhanced Language Models		
12	Neural Topic Models	Deep Generative Models		
13	Natural Language Generation	Assignment 3 Discussion	3. Assignment	

Popular DL Applications for Text Mining

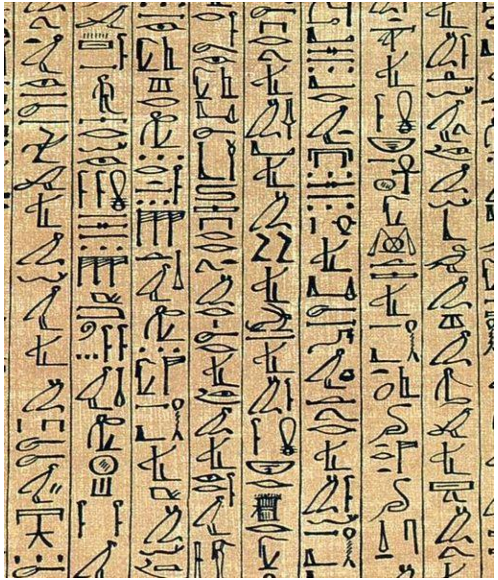
- Text is a sequence of characters or words (tokens)
- DL algorithm for sequences
  - Recurrent neural networks (RNN)
  - 1D Convolutional neural networks (CNN)
- Applications
  - Document classification
    - Topic, category, author, gender, etc.
  - Document clustering (similartity, comparison)
  - Sequence-to-sequence learning
    - Maschine translation
  - Sentiment analysis
    - Classification of tweets, reviews, etc. in positive and negative
  - Language modeling
  - Natural language generation
  - And many more…

What is Meaning

- What‘s the meaning of this sentence?
  - *The weather is nice.*
  - *Time flies like an arrow; fruit flies like a banana.*

- What‘s the meaning of these words?
  - *Bank*
  - *Jaguar*
  - *Java*

- Is cold positive or negative?
  - *Cold beer*
  - *Cold coffee*



https://www.historyonthenet.com/the-egyptians-hieroglyphs

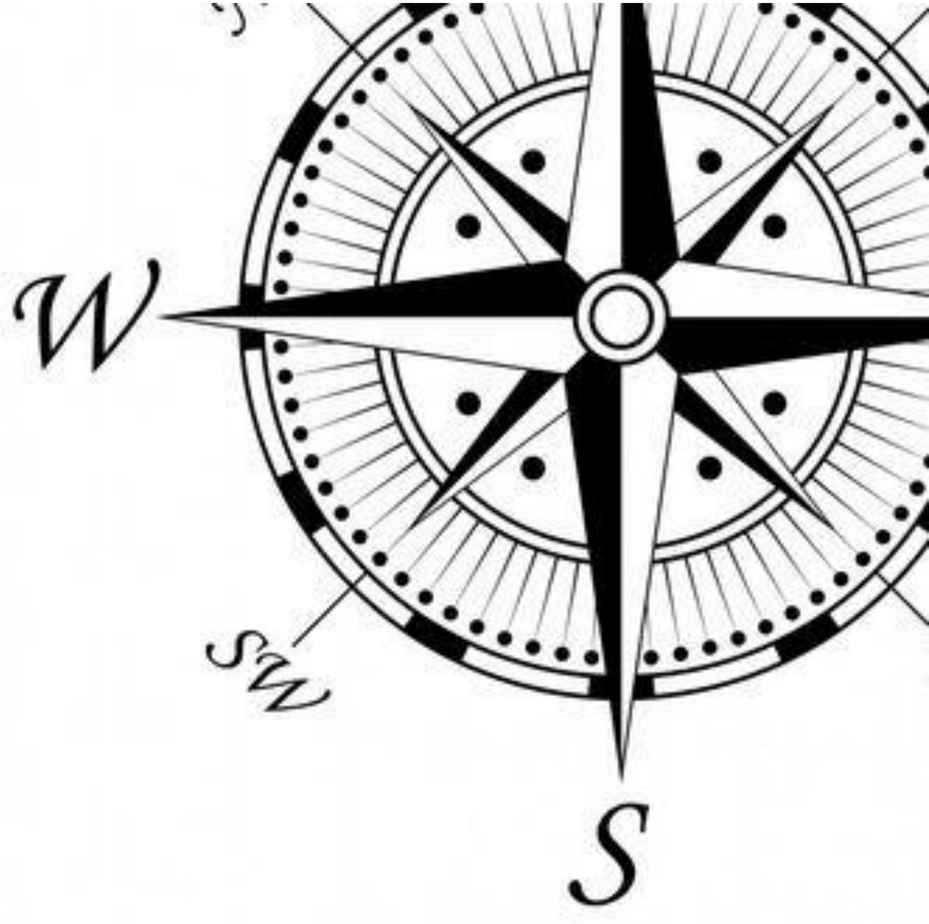
Lerning Goals for this Chapter

- Whats the meaning of meaning?
- Know representations for words
  - Pros and cons
- Understand word2vec
  - CBOW/skip-gram
- Understand alternative: GloVe

- Relevant Chapters:
  - P6.1, S1 + S2
    - http://web.stanford.edu/class/cs224n/

Topics Today

1. **Word Meaning**
2. LSA
3. Word2vec
4. GloVe



What‘s the Meaning of Meaning?

- Merriam-Webster (https://www.merriam-webster.com/dictionary/meaning)
  - **Meaning** noun (mean·ing | ˈmē-niŋ)
  - **Definition of *meaning***
    - 1a **:** the thing one intends to convey especially by language **:** purport
      // Do not mistake my meaning.
    - 1b **:** the thing that is conveyed especially by language **:** import
      // Many words have more than one meaning.
    - 2 **:** something meant or intended **:** aim
      // a mischievous meaning was apparent
    - 3 **:** significant quality
      especially **:** implication of a hidden or special significance
      // a glance full of meaning
    - 4a **:** the logical connotation of a word or phrase
    - 4b **:** the logical denotation or extension of a word or phrase

How do Humans Understand Words?

- Wikipedia (https://de.wikipedia.org/wiki/Bedeutung_(Sprachphilosophie))
  - Meaning means the knowledge of the usual **use** of a word or phrase within a **language community** and given **context**.
  - Meaning means what someone **understands** based on a sign or a linguistic expression.
  - In **reference** theory, meaning is the **object** denoted by a word.
  - In **lexical** meaning theory, the meaning is expressed by a **list of properties** that includes a term.

- Decisive for determining the meaning are
  - word and sentence structure (**syntax**),
  - content of the expression (**semantics**), and
  - the context of use of an utterance (**pragmatics**)

How Can a Computer Understand Words?

- Easy, using a dictionary!
  - E.g. WordNet, SentiWordNet
    - Collection of words organized in synsets (synonym sets) and in „is-a“-Hierarchy

```
fox = wn.synset("fox.n.01")
hyper = lambda s: s.hypernyms()
list(fox.closure(hyper))

[Synset('canine.n.02'),
 Synset('carnivore.n.01'),
 Synset('placental.n.01'),
 Synset('mammal.n.01'),
 Synset('vertebrate.n.01'),
 Synset('chordate.n.01'),
 Synset('animal.n.01'),
 Synset('organism.n.01'),
 Synset('living_thing.n.01'),
 Synset('whole.n.02'),
 Synset('object.n.01'),
 Synset('physical_entity.n.01'),
 Synset('entity.n.01')]
```

```
import nltk
nltk.download('wordnet')
from nltk.corpus import wordnet as wn
for synset in wn.synsets("bank"):
    print ("(%s)" % synset.pos(), ", ".join([l.name() for l
    in synset.lemmas()]), "Def: %s " % synset.definition())
>>>(n) bank
    Def: sloping land (especially the slope beside a body of water)
>>>(n) depository_financial_institution, bank, banking_concern, banking_company
   Def: a financial institution that accepts deposits and channels the money into lending activities
>>>(v) bank
    Def: tip laterally
```

Problems with Dictionaries such as WordNet

- In general useful, e.g. to answer the questions
  - Is this word ‚positive‘?
  - Is this word related to a particular topic?
- But:
  - Not very sophisticated
    - *proficient* is only in certain context a synonym for *good*
  - New words not included at all
    - *Ninja, wizard, genius*
  - Subjective, since man made
  - Very costly to create and maintain
  - Meaning of n-grams unclear
  - Similartiy of words hard to measure
    - Distance in WordNet-Hierarchy

Words as Discrete Symbols

- Bag-of-words, vector space model
  - Words and n-grams correspond to positions in a vector
    - Also possible on character level: character and character n-grams
  - Length of vector = vocabulary size
    - Between 20k and 500k
  - Advantage:
    - Very successul in IR
    - Input data are vectors
    - Can be processed by a DNNs
    - Documents can be represented as sum of all their words‘ vectors

Web 1T corpus: one billion tokens; 13.5 million unigrams

$$\text{aunt}= \begin{array}{c} c \\ \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \end{array} \qquad \text{uncle}= \begin{array}{c} c \\ \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \end{array}$$

Vectorization

- **One-hot-encoding**
  - Vectorization of words

$$\begin{aligned} house &= [0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \ldots 0] \\ mouse &= [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0 \ldots 0] \\ home &= [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0 \ldots 0] \end{aligned}$$

```
from keras.preprocessing.text import Tokenizer
samples = ['The cat sat on the mat.', 'The dog ate my homework.']

tokenizer = Tokenizer(num_words=1000)
tokenizer.fit_on_texts(samples)
sequences = tokenizer.texts_to_sequences(samples)
one_hot_results = tokenizer.texts_to_matrix(samples, mode='binary')
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
```

Only the 1k most frequent words

Index generation

Words to indices

Or in one go

Generated index

- Problem:
  - Vectors of similar words are orthogonal
  - One-hot-encoding has no concept of similarity
- Solution:
  - Make use of Wordnet to find synonyms?
  - **Better: include similarity already in vector representation**

Represent Words by their Context

- Idea: The meaning of a word is defined by the words that appear often in its vicinity.
  - *“You shall know a word by the company it keeps”* (Firth 1957)
  - Most successful concept in modern, statistical NLP

- The context of a word *w* in a text are the words in its vicinity.
  - Typically a window of fixed size
    - Between two and five on both sides
- One word is then represented by all its contexts:

*…proud to own a **house** in this neighborhood…*
*The **house** of cultures offers…*
*…on main street, the **house** of her parents was…*

These contexts then represents the word *house*

John Rupert Firth (1957). "A synopsis of linguistic theory 1930-1955." In *Special Volume of the Philological Society.* Oxford: Oxford University Press.

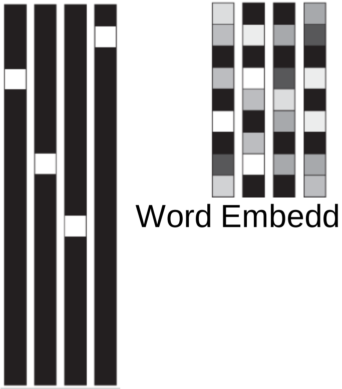
Word Vectors

- Idea: each word is represented by a **dense** vector.
  - Similar words have similar vector representations
  - $house = \begin{bmatrix} -0.345 \\ 0.422 \\ -0.114 \\ -0.225 \\ 0.135 \\ 0.452 \\ -0.164 \\ -0.398 \\ 0.145 \end{bmatrix} mouse = \begin{bmatrix} 0.441 \\ 0.125 \\ -0.514 \\ 0.156 \\ 0.532 \\ -0.216 \\ -0.379 \\ 0.294 \\ 0.542 \end{bmatrix} home = \begin{bmatrix} -0.465 \\ 0.222 \\ -0.165 \\ -0.123 \\ 0.415 \\ 0.392 \\ -0.183 \\ -0.382 \\ 0.1 \end{bmatrix}$
  - These word vectores are called **word embeddings**.
- Two types of word embeddings
  - Learning of the vectors as **part of the machine learning problem**
  - **Pretrained vectors** as input for a DNN
    - Pretrained word embeddings: word2vec, GloVe, fastText

Sparse vs. Dense

High vs. Low dimensional

Hard-coded vs. Learned from data



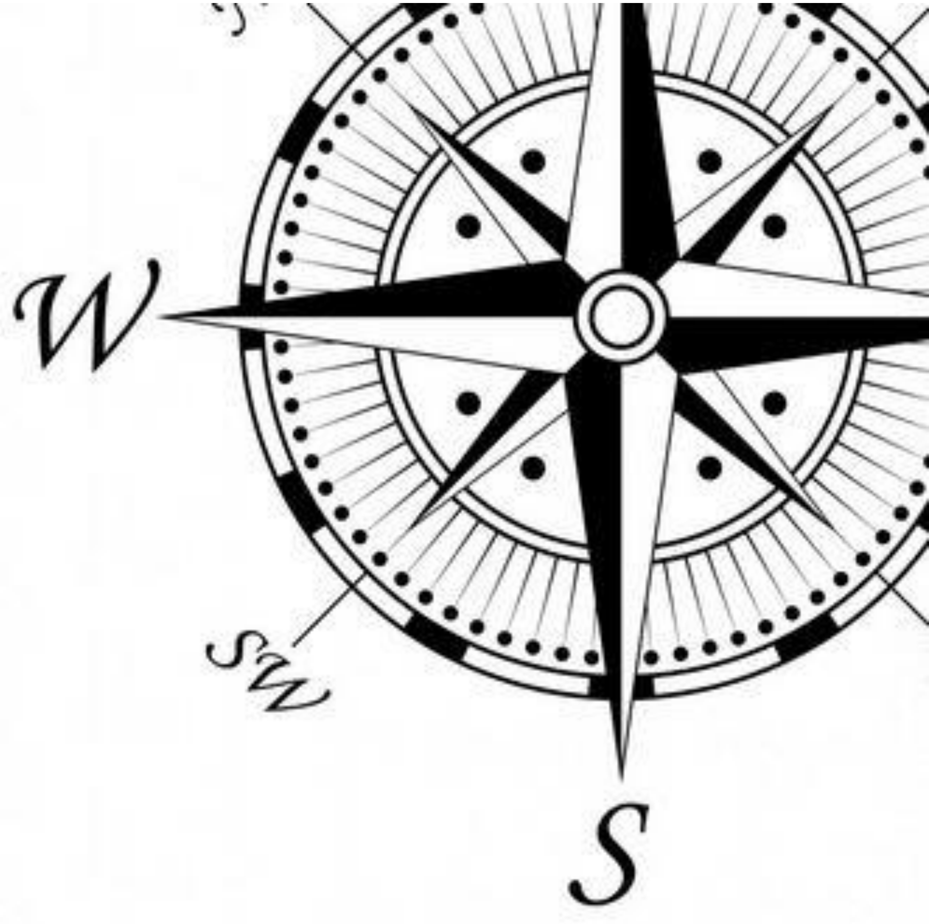
Word Meaning

- How could you determine the sentiment of a sentence using word vectors?
  - E.g movie reviews: „This movie was boring“ vs. „Great movie, highly recommended“

Start 5 4 3 2 1 End

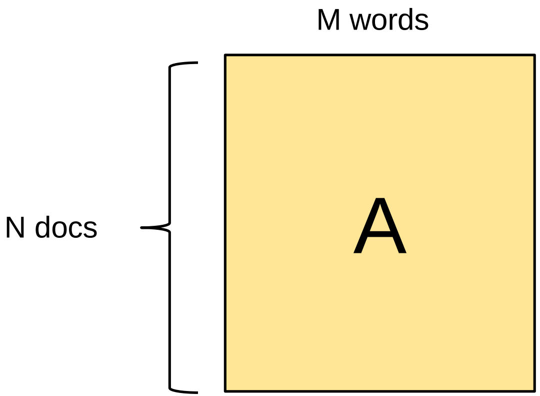
Topics Today

1. Word Meaning
2. **LSA**
3. Word2vec
4. GloVe



Latent Semantic Analysis

- Aka latent semantic indexing (LSA/LSI)
- Based on **bag-of-words model**
- Given a matrix ***A*** encoding some documents:
  - A_{ij} is the count of word ***j*** in document ***i***.
  - Often tf-idf or other “squashing” functions of the count are used
  - Most entries are 0.



Scott Deerwester et al. “Indexing by latent semantic analysis”.
Journal of the American society for information science (1990).

Latent Semantic Analysis

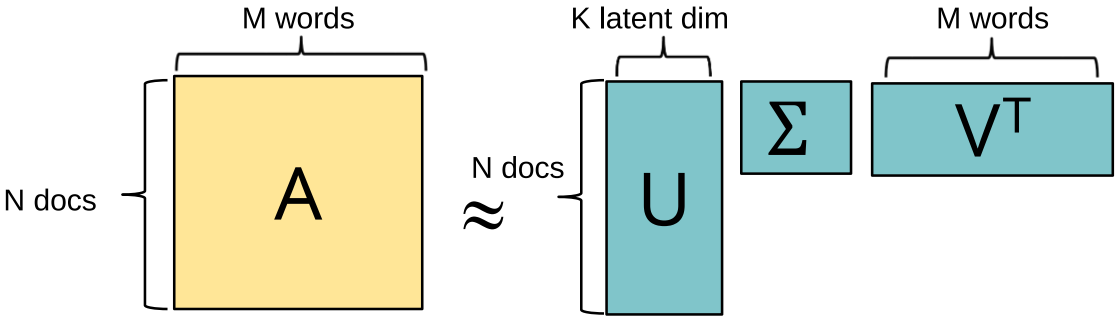
- Low-rank **singular value decomposition (SVD)**:

$$A_{[m\times n]} = U_{[m\times r]}\Sigma_{[r\times r]}(V_{[n\times r]})^T$$

  - U : document-to-concept similarities matrix (orthogonal matrix)
  - V : word-to-concept similarities matrix (orthogonal matrix)
  - Σ : strength of each concept

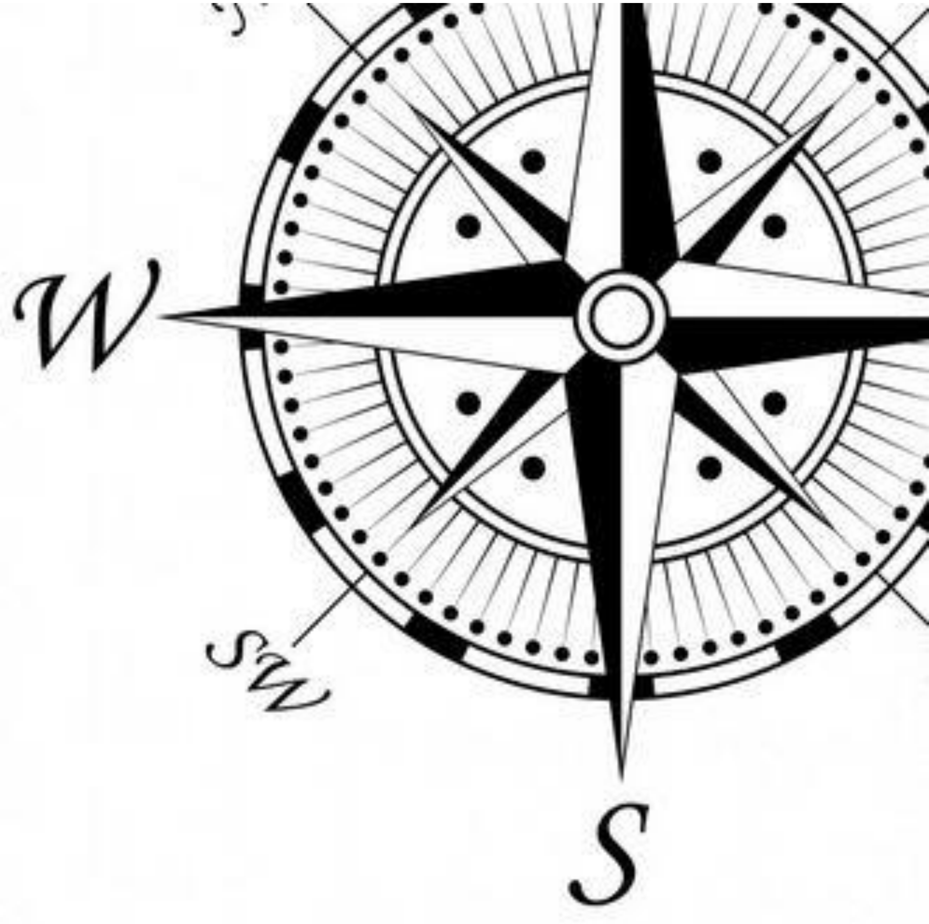
An SVD factorization gives the best possible reconstructions of a word *w* from its embedding

- Then given a word ***w*** (column of ***A***):
  - $\varsigma = w^T \times U$ is the embedding (encoding) of the word ***w*** in the latent space
  - $w \approx U \times \varsigma^T = U \times (w^T \times U)^T$ is the decoding of the word ***w*** from its embedding



Topics Today

1. Word Meaning
2. LSA
3. **Word2vec**
4. GloVe



Word Embeddings

- **Word2vec** is a framework to to compute word vectors (**word embeddings**) using a deep neural network architecture.
- Idea:
  1. Given a very, very large text corpus
     - E.g. the web, large digital libraries, at least Wikipedia
  2. Represent each word of the vocabulary as a dense vector.
     - Number of dimensions: 50-300
  3. Iterate over each word *c* (center) in the text and look at its context *o* (outside).
  4. Learn to predict the center word *c*, given the context word vectors.
     - $P(c|o)$; Multi-class classification problem; or vice versa $P(o|c)$
  5. Change/adapt the word vectors in a way to maximize these probabilities.
     - Similar context results in similar word vectors!

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems (NIPS).* pp. 3111-3119.

Word2vec

- Two variants of Word2vec
  1. **Skip-grams** (SG)
     - Predict the context words given the center word
  2. **Continuous Bag of Words** (CBOW)
     - Predict the center word given the bag-of-words of ist context

Independent of exact position

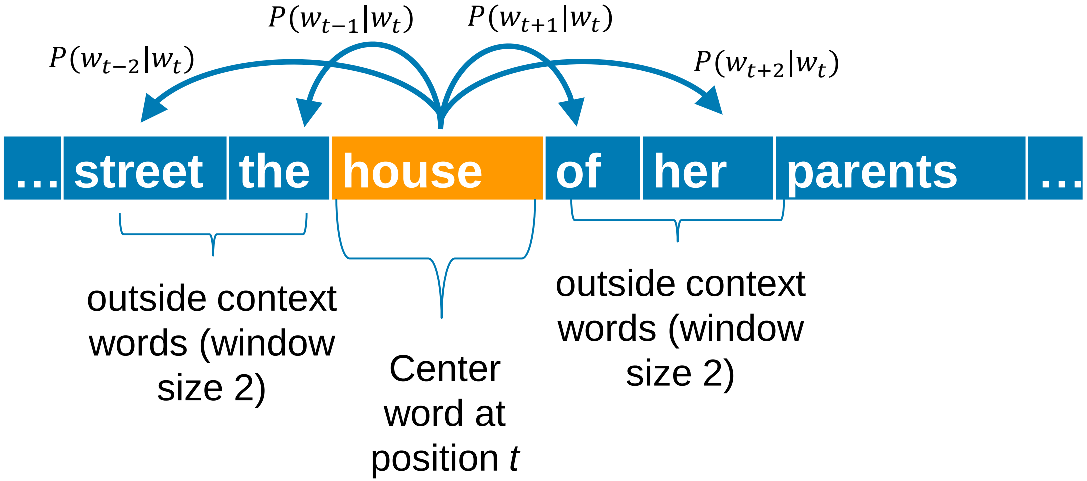
- Two more or less efficient methods to learn the word vectors
  1. Hierarchical softmax
  2. Negative sampling

  – We look at naive softmax
     - VERY inefficient ;) but easier to understand ☺

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems (NIPS).* pp. 3111-3119.

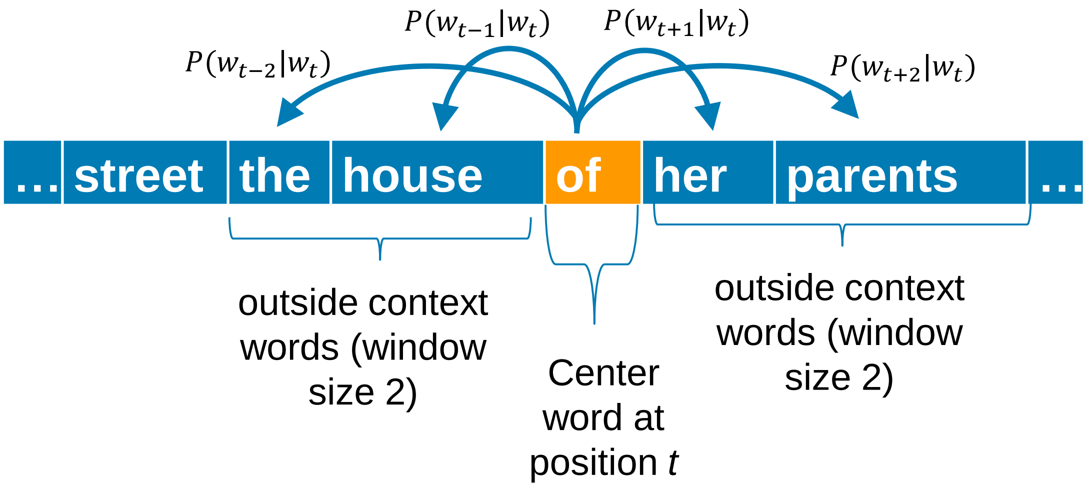
Example

- Calculation of $P(w_{t+x}|w_t)$ for window size x

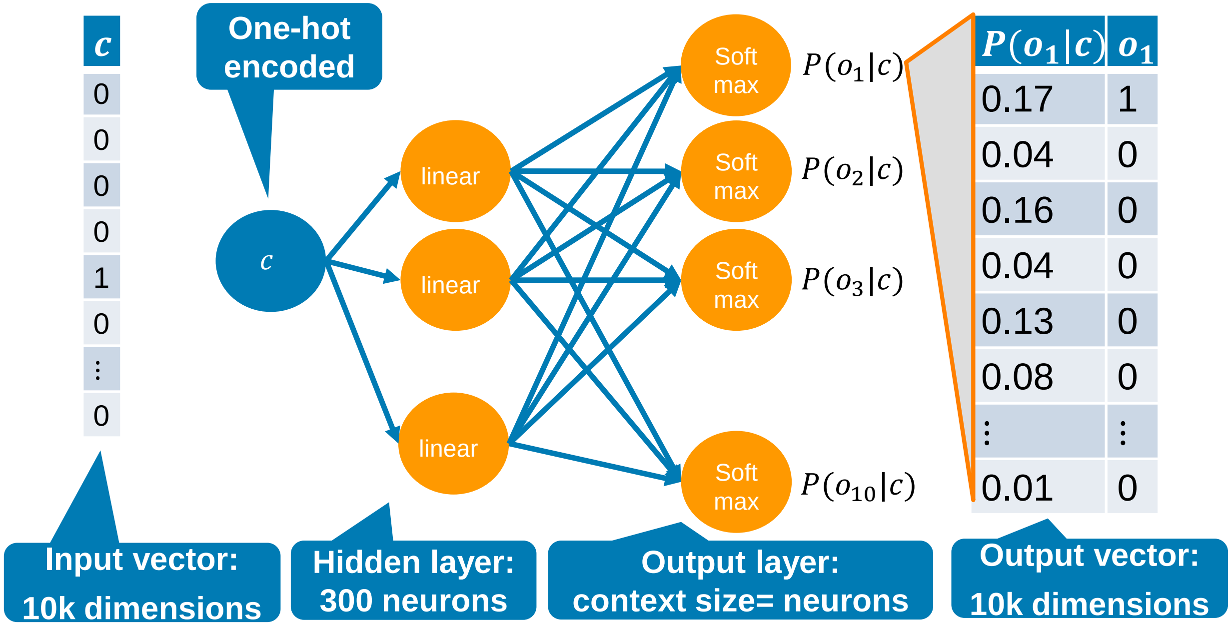


Example

- Calculation of $P(w_{t+x}|w_t)$ for window size x



Network Architecture



Loss Function

- For each position $t = 1, \dots, T$ predict the context words in window of size m for a given center word w_t.

$$J'(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \ne 0}} P(w_{t+j} | w_t; \theta)$$

- The loss function $J(\theta)$ is then the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T} \log J'(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \ne 0}} log P(w_{t+j} | w_t; \theta)$$

- Minimizing the loss function $\Leftrightarrow$ Maximizing the prediction accuracy

Computing the Loss Function I

- Minimizing the loss function

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\leq j\leq m\\ j\neq 0}} logP(w_{t+1}|w_t;\theta)$$

- How to compute $P(w_{t+1}|w_t;\theta)$?
- For each word w there are two vectors:
  - v_w if w is a center word
  - u_w if w is a context word
- Then, for a given center word c and a given context word o you can compute:

$$P(o|c) = \frac{\mathrm{e}^{u_o^T v_c}}{\sum_{w\in V}\mathrm{e}^{u_w^T v_c}}$$

Scalar product compares the similarity of *o* and *c*.

Normalization across the complete vocabulary

Computing the Loss Function II

$$P(o|c) = \frac{e^{u_o^T v_c}}{\sum_{w \in V} e^{u_w^T v_c}}$$

- A version of the softmax function
- Reminder:

$$softmax(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} = p_i$$

  – The softmax function maps arbitrary values x_i to a probability distribution p_i.
    - „max“, because large values are maped to excessively large probability values
    - „soft“, because also very small values get mapped to a small probability value

Parameters of the Model

- Optimize all parameters so that loss fuction is minimized!

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\leq j\leq m\\ j\neq 0}} log\,\frac{e^{u_o^T v_c}}{\sum_{w\in V} e^{u_w^T v_c}}$$

- Find two vector representations for each word in a way that similar words have similar vectors.
- Compute gradients per window for all parameters
- Why two vectors per word?
  - Easier to optimize
  - Use average at the end!

$$\theta = \begin{bmatrix} v_{aachen} \\ v_{aake} \\ \vdots \\ v_{zuse} \\ u_{aachen} \\ u_{aake} \\ \vdots \\ u_{zuse} \end{bmatrix} \in \mathbb{R}^{2dV}$$

Gradients

- How to change each vector v and u to minimize loss function?
- For each window and context word, we need to compute $\frac{\partial J}{\partial v_c}$ and $\frac{\partial J}{\partial u_o}$:

$$\frac{\partial}{\partial v_c} log \frac{\mathrm{e}^{u_o^T v_c}}{\sum_{w \in V} \mathrm{e}^{u_w^T v_c}} = \underbrace{\frac{\partial}{\partial v_c} log \mathrm{e}^{u_o^T v_c}}_{\text{A}} - \underbrace{\frac{\partial}{\partial v_c} log \sum_{w \in V} \mathrm{e}^{u_w^T v_c}}_{\text{B}}$$

- A: $\frac{\partial}{\partial v_c} u_o^T v_c = u_o^T$
- B: $\frac{\partial}{\partial v_c} log \sum_{w \in V} \mathrm{e}^{u_w^T v_c} = \frac{1}{\sum_{w \in V} \mathrm{e}^{u_w^T v_c}} \cdot \frac{\partial}{\partial v_c} \sum_{x \in V} \mathrm{e}^{u_w^T v_c} = \frac{1}{\sum_{w \in V} \mathrm{e}^{u_w^T v_c}} \cdot \sum_{x \in V} \frac{\partial}{\partial v_c} \mathrm{e}^{u_x^T v_c}$

$$= \frac{1}{\sum_{w \in V} \mathrm{e}^{u_w^T v_c}} \cdot \sum_{x \in V} \mathrm{e}^{u_x^T v_c} \frac{\partial}{\partial v_c} u_x^T v_c = \frac{1}{\sum_{w \in V} \mathrm{e}^{u_w^T v_c}} \cdot \sum_{x \in V} \mathrm{e}^{u_x^T v_c} u_x^T$$

$$= \sum_{x \in V} \frac{\mathrm{e}^{u_x^T v_c}}{\sum_{w \in V} \mathrm{e}^{u_w^T v_c}} u_x^T = \sum_{x \in V} P(x|c)\, u_x^T$$

- A-B: $\mathrm{u_o} - \sum_{x \in V} P(x|c)\, u_x^T$

Learning Word Embeddings for Preprocessing

- Goal of word2vec: Learn word vectors
  - Iterate through all words in a corpus
  - Predict which words occur in their contexts
  - Thus: Similar context results in similar word vectors!

- But **every** DNN learns a representation of the input data!
  - Word vectors can be learnt for a specific task.
  - Representation is then optimized towards the problem that needs to be solved.
    - For word2vec this problem is to predict the context (meaning).
    - Other nets predict other things, e.g., polarity (sentiment).

Embedding Example: IMDB

```
from keras.datasets import imdb
from keras import preprocessing

max_features = 10000
maxlen = 20

(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

x_train = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)
```

Only first 20 words of a text

Transforms a list of integers into a 2d integer tensor of shape (samples, maxlen)

Embedding Example: IMDB

```
from keras.models import Sequential
from keras.layers import Flatten, Dense

model = Sequential()
model.add(Embedding(10000, 8, input_length=maxlen))
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])

model.summary()

history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_split=0.2)
```

Vocabulary size, embedding dimensions

Transformation of the 3d-tensor into a 2d-tensor of shape (samples, maxlen*8)

- Validation accuracy $\approx 76\%$
  - Not bad for only looking at the 20 first words

Learning Embeddings

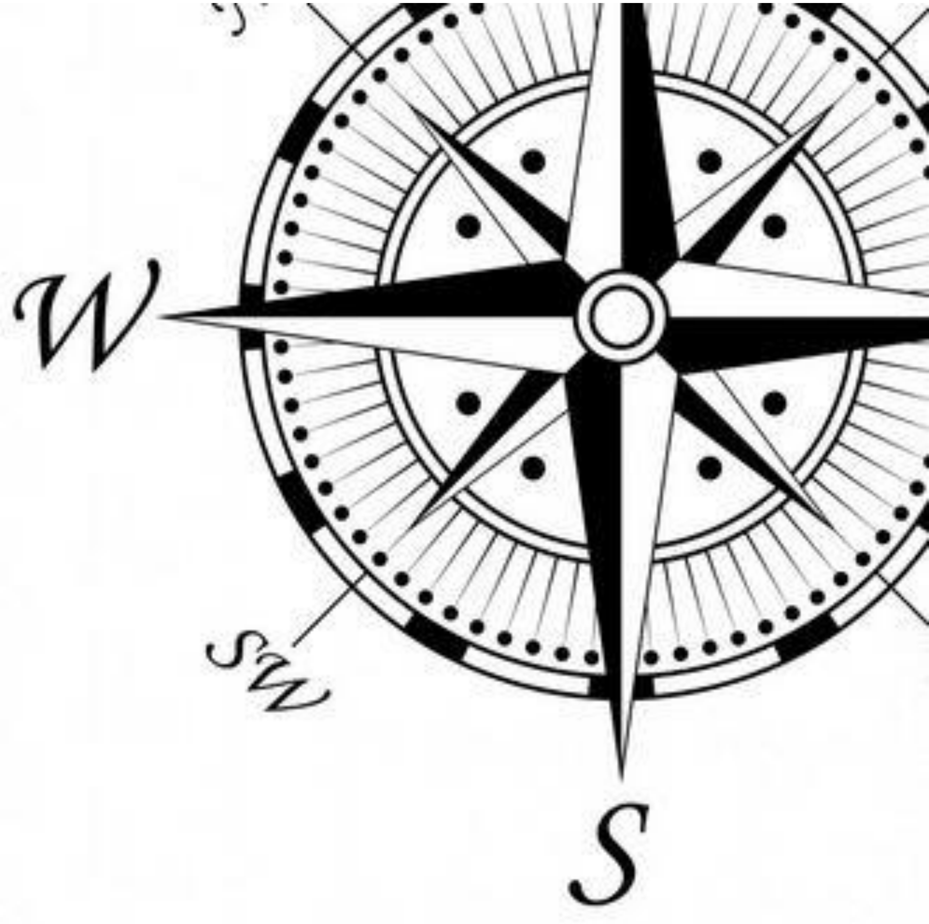
- Imagine a corpus with only three (meaningful) words
  - house, home, mouse
  - Further there are stop words and verbs.
  - Assume the corpus contains 10k sentences

- Go through the process of learning word embeddings manually using the network architecture and the loss function
  - How does backpropagation changes the weights?

Start 5 4 3 2 1 End

Topics Today

1. Word Meaning
2. LSA
3. Word2vec
4. **GloVe**



GloVe

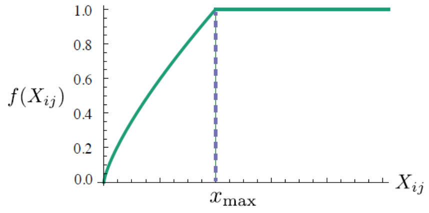
- Latent Semantic Analysis (1988)
  - Term-weighting-based model
- Word2Vec (2013)
  - Prediction-based model
- GloVe (2014)
  - Count-based model

- Count-based models learn vectors by doing dimensionality reduction on a **co-occurrence counts matrix**.
  - Factorize this matrix to yield a lower-dimensional matrix of words and features, where each row yields a vector representation for each word.
  - The counts matrix is preprocessed by normalizing the counts and log-smoothing them.

GloVe

The ratio of co-occurrence probabilities can indicate meaning

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96



- Log-bilinear-model: $w_i \cdot w_j = \log P(i|j)$
- Ratio of co-occurrence probabilities: $w_x \cdot (w_a - w_b) = \log \frac{P(x|a)}{P(x|b)}$
- Loss function

$$J(\theta) = \frac{1}{2} \sum_{i,j=1}^{W} f(X_{ij}) \left(u_i^T v_j - logX_{ij}\right)^2$$

- Final word vector for word x: $w_x = u_x + v_x$

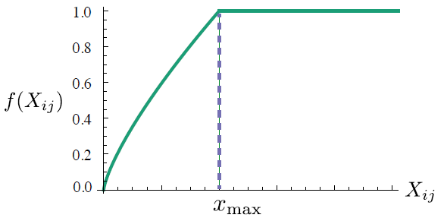
weighting function to penalize rare co-occurrences

Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation. In *EMNLP* (pp. 1532-1543)

GloVe

- Loss function

$$J(\theta) = \frac{1}{2} \sum_{i,j=1}^{W} f(P_{ij}) \left(u_i^T v_j - log P_{ij}\right)^2$$



- Final word vector for word x: $w_x = u_x + v_x$

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

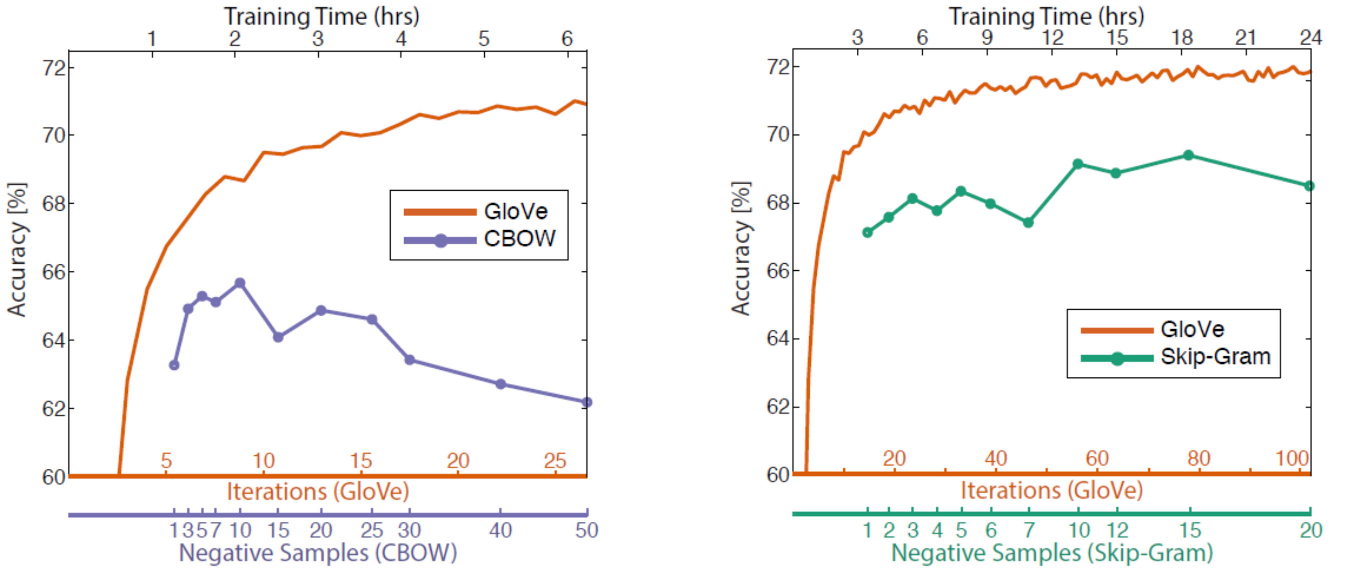
Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1532-1543).

GloVe Example

- Which word is closest to *frog*?
  1. frogs
  2. toad
  3. litoria
  4. leptodactylidae
  5. rana
  6. lizard
  7. eleutherodactylus

GloVe Results

- Overall accuracy on the word analogy task
  - 300-dimensional vectors trained on a 6B token corpus (Wikipedia 2014 + Gigaword 5) with a 400,000 word vocabulary, and a symmetric context window of size 10.



Lerning Goals for this Chapter

- Whats the meaning of meaning?
- Know representations for words
  - Pros and cons
- Understand word2vec
  - CBOW/skip-gram
- Understand alternative: GloVe

- Relevant Chapters:
  - P6.1, S1 + S2
    - http://web.stanford.edu/class/cs224n/

Literature

- Efficient Estimation of Word Representations in Vector Space
  – (original word2vec paper)
- Distributed Representations of Words and Phrases and their Compositionality
  – (negative sampling paper)
- GloVe: Global Vectors for Word Representation
  – (original GloVe paper)
- Improving Distributional Similarity with Lessons Learned from Word Embeddings
- word2vec parameter learning explained