Recap

- Logistic/linear regression
  - Predict probability of a class or some number
  - Optimize parameters in function $h(x) = \frac{1}{e^{-(b+ax)}}$ or $h(x) = ax + b$
- Gradient descent
  - Initialise parameters randomly
  - Measure wrongness with the loss function $J(\vec{w})$
  - Improve iteratively

Word2Vec

Sprachverarbeitung (VL + Ü)

Nils Reiter

June 20, 2023

INSTITUT FÜR
DIGITAL HUMANITIES
UNIVERSITÄT ZU KÖLN

AI Revolution

Three components

- Hardware: GPUs, storage, ...
- Input representation
- Neural network architecture

Exercise

This exercise is to be done on the server.

- The directory `/teaching/summer-2023/sprachverarbeitung/data` contains three corpora of different sources in three large plain text files. 'bt.txt' contains text from parliamentary debates from the German Bundestag, 'wp.txt' contains text from the German Wikipedia and 'st.txt' a corpus of steam reviews. Choose one of these as you like and find interesting. They have very different sizes.
- Train a word2vec model using the script `train-word2vec.py` in `/teaching/summer-2023/sprachverarbeitung/word2vec`. You'll need to change the filename (which is hardcoded in the script).
- Load and explore the module using the script 'use-word2vec.py' (again, change the filename within the script). Try to find a few good and a few bad examples. Run the script with 'python3 -i use-word2vec.py' to enter the Python interpreter. The function `model.wv.most_similar("universität")` will give you the ten most similar words to the word 'universität'. Beware: All tokens are lower-cased.
- Optional: Do all the above for a second corpus and compare the resulting similarities between word pairs.

Introduction

- Embedding: Words are *embedded* into a high-dimensional vector space

Introduction

- Embedding: Words are *embedded* into a high-dimensional vector space
- Word2Vec
  - A method to represent words in a (high-dimensional) vector space
  - No end-user task

Introduction

- Embedding: Words are *embedded* into a high-dimensional vector space
- Word2Vec
  - A method to represent words in a (high-dimensional) vector space
  - No end-user task
- A vector representation for »köln«

0.0539 -0.0030 0.0203 -0.1084 -0.0099 0.0705 -0.0546 -0.0433 -0.0096 0.0561 -0.0095 0.0280 0.1726 0.0190 0.0369 0.0217 -0.0002 -0.0309 0.0347 -0.0749 -0.0202 0.0151 -0.0195 0.0001 0.0232 0.0243 -0.0170 -0.0090 -0.0108 -0.0943 0.0376 0.1118 -0.0324 0.0148 -0.0033 0.0537 -0.0681 -0.0733 -0.0201 -0.0329 0.1242 0.0324 -0.0744 -0.0149 -0.0047 -0.0484 -0.0483 0.0481 0.0107 0.0101 -0.0704 0.0500 0.0112 -0.0227 0.0499 -0.0259 -0.0441 0.0712 -0.0157 -0.1271 0.0407 -0.0495 -0.0359 0.0202 0.0024 0.0764 0.0196 0.0267 -0.0117 0.0026 0.0171 -0.0121 -0.1374 -0.0370 0.0247 -0.0113 -0.0094 0.0322 -0.0347 -0.0866 0.0042 -0.0014 0.0067 0.0591 0.0009 0.0085 0.0310 0.0479 -0.0511 0.0198 -0.0886 -0.0274 -0.1364 0.0322 -0.1638 -0.0689 0.0016 -0.1039 0.0059 0.0757 -0.0034 0.1013 -0.0034 -0.0065 -0.0468 0.1577 -0.0065 -0.0478 -0.0004 0.0682 0.0045 -0.0607 -0.0590 0.0343 0.0036 -0.1014 -0.0136 -0.0063 0.0801 0.0360 0.0579 -0.0039 0.0975 0.0500 -0.0558 -0.0095 0.0057 -0.0246 0.1070 -0.0186 0.0669 -0.0781 -0.0569 -0.1286 -0.0834 0.0106 -0.0672 -0.0205 0.0613 0.0290 -0.0545 -0.0481 -0.0882 -0.0489 0.0622 -0.0730 -0.0192 -0.0415 -0.0287 0.0218 -0.0427 -0.0046 0.0255 -0.1164 0.0077 -0.0546 -0.0786 0.0000 -0.0456 0.0943 0.0157 -0.0117 -0.0441 -0.0015 -0.0556 -0.0508 0.0088 0.0418 0.0030 -0.1450 -0.0663 0.0800 0.0172 -0.0289 0.1178 -0.0973 0.0888 0.0637 -0.0295 0.0212 0.0100 -0.0860 0.0035 0.0730 0.0425 -0.0080 0.0885 -0.0166 -0.0765 0.0004 -0.0118 0.0138 -0.0093 -0.0606 -0.0447 -0.0746 0.0131 -0.0447 -0.0763 0.0032 0.1181 0.0542 0.0431 -0.0273 0.0547 0.0135 0.0006 -0.0241 -0.0418 0.0278 -0.0821 -0.0572 -0.0039 0.0214 -0.0196 0.0449 -0.0286 0.0204 0.0681 -0.0901 -0.0266 -0.0287 -0.0874 0.0797 -0.0784 -0.0920 0.0380 0.0411 0.0859 0.0369 0.0595 0.0446 0.0363 -0.0353 -0.0044 -0.0061 0.1134 0.1420 -0.0026 -0.0013 0.0033 0.0508 0.0096 -0.0757 0.0085 -0.0099 -0.0384 0.0218 -0.0259 -0.0112 -0.0212 0.0273 0.0532 -0.0278 -0.0634 0.0317 -0.0022 0.0882 -0.0240 0.0031 -0.0370 0.0747 -0.0097 -0.0315 0.0405 0.0124 -0.1416 -0.0768 0.0363 -0.1248 -0.0134 0.0702 -0.0905 -0.0387 0.0683 -0.0784 0.0886 0.0640 0.0611 -0.0199 -0.0447 -0.1331 -0.1247 0.0540 0.0499 -0.0212 -0.0544 -0.1161 -0.0729 0.0894 0.0532 0.0164 -0.0039 -0.0108 -0.0248 -0.1021 -0.0549 -0.0318 0.0309 -0.0691

Embeddings

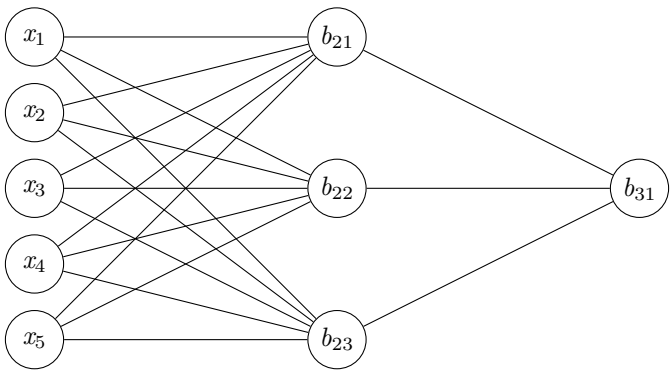
Why is that useful?

1. Input Representation for Neural Networks
   - Example Task: Sentiment Analysis
   - Take a sentence, classify it as positive or negative

Embeddings

Why is that useful?

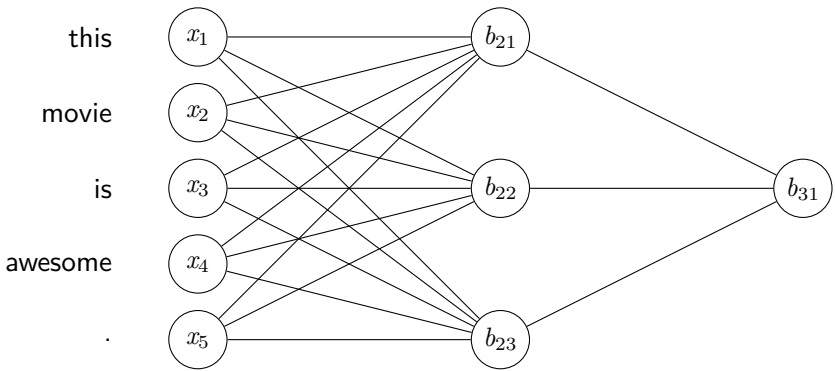
1. Input Representation for Neural Networks
   - Example Task: Sentiment Analysis
   - Take a sentence, classify it as positive or negative



Embeddings

Why is that useful?

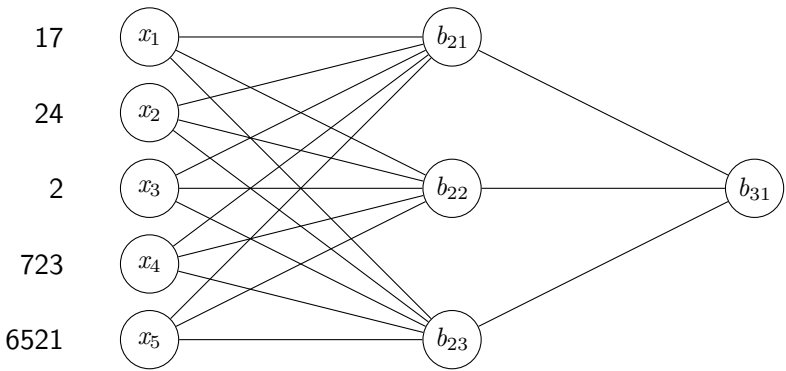
1. Input Representation for Neural Networks
   - Example Task: Sentiment Analysis
   - Take a sentence, classify it as positive or negative



Embeddings

Why is that useful?

1. Input Representation for Neural Networks
   - Example Task: Sentiment Analysis
   - Take a sentence, classify it as positive or negative



Embeddings

Why is that useful?

1. Input Representation for Neural Networks
   - Example Task: Sentiment Analysis
   - Take a sentence, classify it as positive or negative

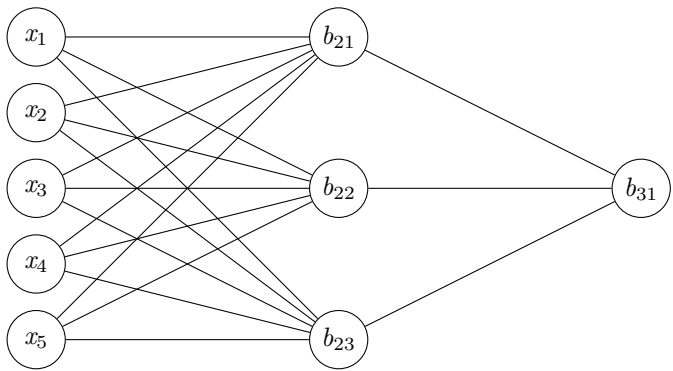
$\langle 0.0088, 0.0418, 0.0030, -0.1450 \rangle$

$\langle 0.0683, -0.0784, 0.0886, 0.0640 \rangle$

$\langle -0.0353, -0.0044, -0.0061, 0.1134 \rangle$

$\langle -0.0278, -0.0634, 0.0317, -0.0022 \rangle$

$\langle -0.0689, 0.0016, -0.1039, 0.0059 \rangle$

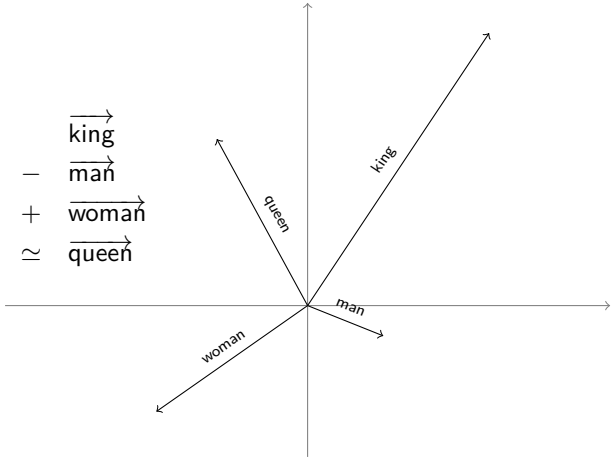


Embeddings

Why is that useful?

2. For semantic calculations

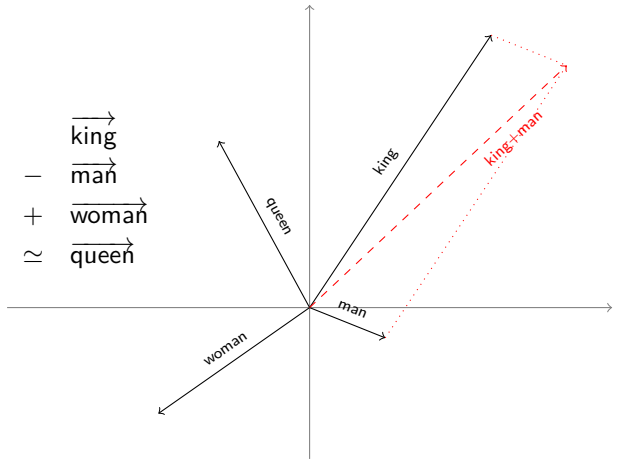
$$\overrightarrow{\text{king}} - \overrightarrow{\text{man}} + \overrightarrow{\text{woman}} \simeq \overrightarrow{\text{queen}}$$



Embeddings

Why is that useful?

2. For semantic calculations

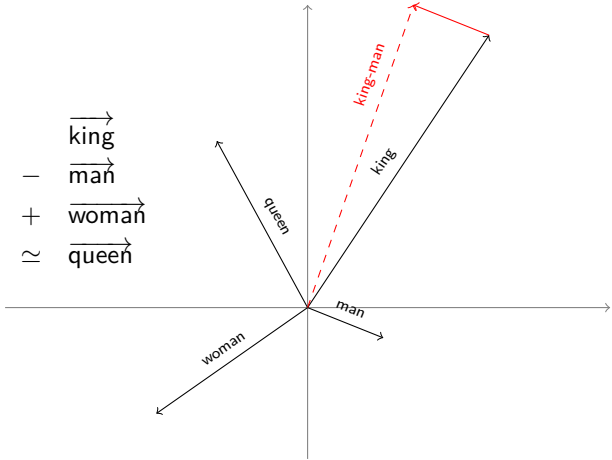


Embeddings

Why is that useful?

2. For semantic calculations

$$\overrightarrow{\text{king}} - \overrightarrow{\text{man}} + \overrightarrow{\text{woman}} \simeq \overrightarrow{\text{queen}}$$

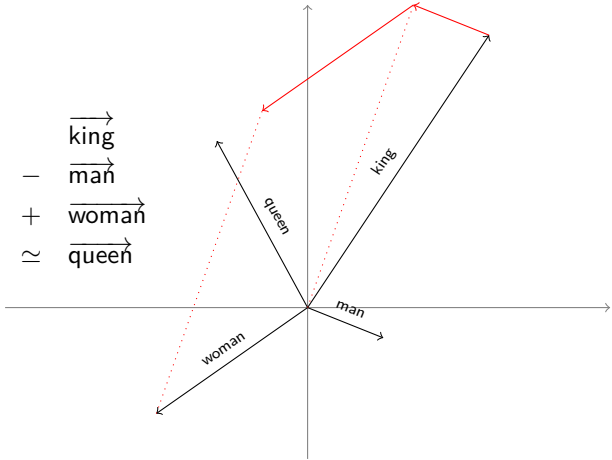


Embeddings

Why is that useful?

2. For semantic calculations

$$\overrightarrow{\text{king}} - \overrightarrow{\text{man}} + \overrightarrow{\text{woman}} \simeq \overrightarrow{\text{queen}}$$

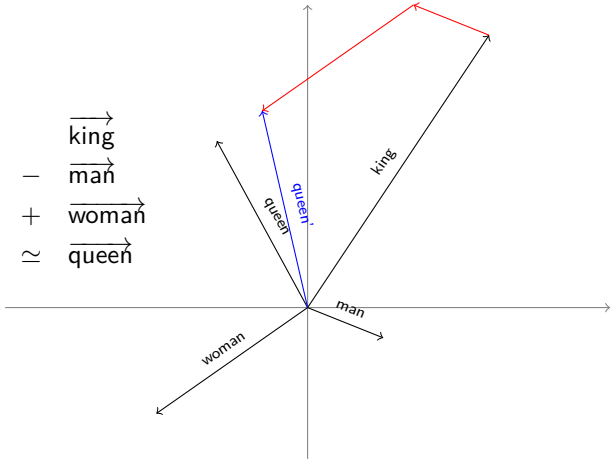


Embeddings

Why is that useful?

2. For semantic calculations

$$\overrightarrow{\text{king}} - \overrightarrow{\text{man}} + \overrightarrow{\text{woman}} \simeq \overrightarrow{\text{queen}}$$

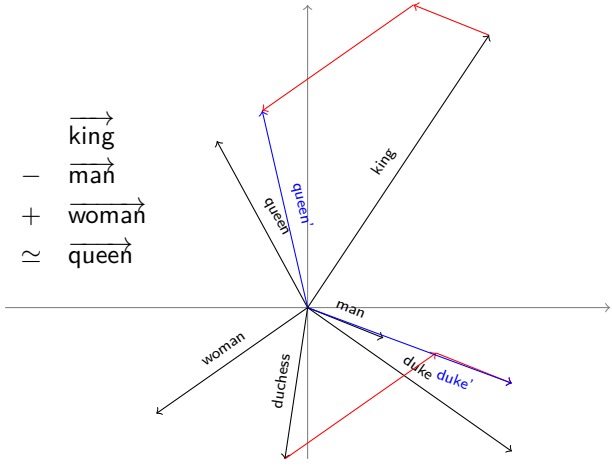


Embeddings

Why is that useful?

2. For semantic calculations

$$\overrightarrow{\text{king}} - \overrightarrow{\text{man}} + \overrightarrow{\text{woman}} \simeq \overrightarrow{\text{queen}}$$



Subsection 1

Generating Word Embeddings with Word2Vec

Literature basis

Two very influential papers by Mikolov et al.

T. Mikolov/K. Chen/G. Corrado/J. Dean (2013). »Efficient Estimation of Word Representations in Vector Space«. In: *ArXiv e-prints*

Tomas Mikolov/Ilya Sutskever/Kai Chen/Greg S Corrado/Jeff Dean (2013). »Distributed Representations of Words and Phrases and their Compositionality«. In: *Advances in Neural Information Processing Systems 26*. Ed. by C. J. C. Burges/L. Bottou/M. Welling/Z. Ghahramani/K. Q. Weinberger. Curran Associates, Inc., pp. 3111–3119

Software package

word2vec – `https://github.com/tmikolov/word2vec`
(other implementations do exist)

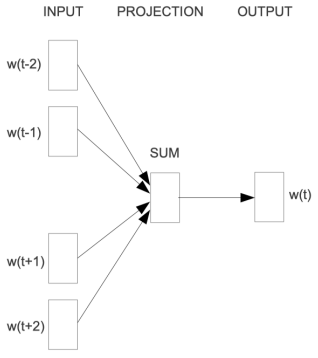
Textbook recommendation

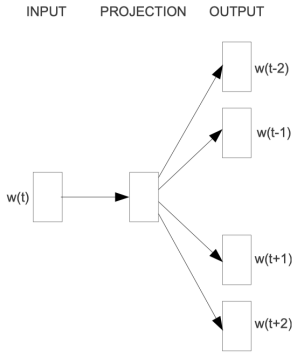
Dan Jurafsky/James H. Martin (2020). *Speech and Language Processing*. 3rd ed. draft

Core Idea

- Define a classification task for which we have huge training data sets
  - Given a word, predict predict possible context words
  - Training data: Any text collection (e.g., Wikipedia)
- Train a neural network
- Extract learned weights and use as embeddings ☑

Two tasks





Continuous Bag of Words (CBOW)

Context words used to predict a single word

Skip-Gram

One word used to predict its context

Skip-gram

- Context: ± 2 words around target word t

```
... lemon, a [tablespoon of apricot jam, a] pinch ...
             c1          c2 t       c3   c4
```

Skip-gram

- Context: ± 2 words around target word t

```
... lemon, a [tablespoon of apricot jam, a] pinch ...
             c1          c2 t       c3   c4
```

- Classifier:
  - Predict for (t, c) wether c are *really* context words for t
  - Probability of $\vec{t}$ and $\vec{c}$ being positive examples: $p(+|\vec{t}, \vec{c})$

Skip-gram

- Context: ± 2 words around target word t

```
... lemon, a [tablespoon of apricot jam, a] pinch ...
             c1          c2 t       c3   c4
```

- Classifier:
  - Predict for (t, c) wether c are *really* context words for t
  - Probability of $\vec{t}$ and $\vec{c}$ being positive examples: $p(+|\vec{t}, \vec{c})$
- Probability is based on similarity
  - »a word is likely to occur near the target if its embedding is similar to the target embedding«

Jurafsky/Martin (JM20, 112)

Skip-gram

- Context: ± 2 words around target word t

```
... lemon, a [tablespoon of apricot jam, a] pinch ...
             c1          c2 t        c3   c4
```

- Classifier:
  - Predict for (t, c) wether c are *really* context words for t
  - Probability of $\vec{t}$ and $\vec{c}$ being positive examples: $p(+|\vec{t}, \vec{c})$
- Probability is based on similarity
  - »a word is likely to occur near the target if its embedding is similar to the target embedding«

    Jurafsky/Martin (JM20, 112)
  - Similarity of vectors? Dot product / cosine! ➔ Next week

Skip-gram

- Context: ± 2 words around target word t

```
... lemon, a [tablespoon of apricot jam, a] pinch ...
             c1          c2 t       c3   c4
```

- Classifier:
  - Predict for (t, c) wether c are *really* context words for t
  - Probability of $\vec{t}$ and $\vec{c}$ being positive examples: $p(+|\vec{t}, \vec{c})$
- Probability is based on similarity
  - »a word is likely to occur near the target if its embedding is similar to the target embedding«

    Jurafsky/Martin (JM20, 112)
  - Similarity of vectors? Dot product / cosine! ➔ Next week
  - Similarity $\rightarrow$ probability? Sigmoid / logistic function! ➔ Last week

Workflow

1. Get embeddings
   - Train them yourself
   - Download them from someone else
     - E.g., `http://vectors.nlpl.eu/repository/`
2. Use embeddings
   - ML Training: Replace all your words by the corresponding vectors
     - Let the neural network figure out the rest
   - Inspection
     - Look at similarities between words (i.e., word vectors)
     - Make ›semantic calculations‹

Section 1

Exercise

Exercise Material

train-word2vec.py

```
from gensim.models import Word2Vec

# Train the model on the file "wp.txt"
model = Word2Vec(corpus_file="wp.txt",
  vector_size=100, window=5, min_count=1, workers=4)

# Save it to a file
model.save("w2v-wikipedia.model")
```

Exercise Material

use-word2vec.py

```
from gensim.models import Word2Vec

# Load the model from a file
model = Word2Vec.load("w2v-wikipedia.model")
# Display the embedding for a single word
model.wv["europa"]
# Display the top most similar words to "europa"
model.wv.most_similar("europa", topn=10)
```

Exercise Material

Usage

```
$ python3 train-word2vec.py
$ python3 -i use-word2vec.py
>>> model.wv.most_similar('köln')
[('düsseldorf', 0.9235410094261169), ('dresden', 0.9070231914520264), ('zürich',
    0.9018810987472534), ('hannover', 0.8999517560005188), ('münchen',
    0.8994168639183044), ('hamburg', 0.8964424729347229), ('wien', 0.8960092663764954)
    , ('nürnberg', 0.89522385597229), ('leipzig', 0.8951458930969238), ('basel',
    0.8885967135429382)]
>>> model.wv.most_similar('rot', topn=15)
[('blau', 0.9638434052467346), ('grün', 0.9334064722061157), ('schwarz',
    0.8903344869613647), ('rot,', 0.8900261521339417), ('gelb', 0.8864821195602417), (
    'blau,', 0.8806263208389282), ('leuchtend', 0.8547627925872803), ('schwarzer',
    0.8546361923217773), ('schwarz,', 0.8540391325950623), ('grau',
    0.8489606976509094), ('seitlich', 0.848198413848877), ('gelben',
    0.8423659801483154), ('grün,', 0.8415961265563965), ('braunen',
    0.8389317393302917), ('streifen', 0.8379982113838196)]
```

Exercise

This exercise is to be done on the server.

- The directory `/teaching/summer-2023/sprachverarbeitung/data` contains three corpora of different sources in three large plain text files. 'bt.txt' contains text from parliamentary debates from the German Bundestag, 'wp.txt' contains text from the German Wikipedia and 'st.txt' a corpus of steam reviews. Choose one of these as you like and find interesting. They have very different sizes.
- Train a word2vec model using the script `train-word2vec.py` in `/teaching/summer-2023/sprachverarbeitung/word2vec`. You'll need to change the filename (which is hardcoded in the script).
- Load and explore the module using the script 'use-word2vec.py' (again, change the filename within the script). Try to find a few good and a few bad examples. Run the script with 'python3 -i use-word2vec.py' to enter the Python interpreter. The function `model.wv.most_similar("universität")` will give you the ten most similar words to the word 'universität'. Beware: All tokens are lower-cased.
- Optional: Do all the above for a second corpus and compare the resulting similarities between word pairs.