Tutorial: Word Embeddings

Niels Scholten | JM2050-M-6 Natural Language Processing

JADS is powered by

Why word embeddings?

- We need to represent words as numbers do to calculations.

Challenges of conventional methods:

- High-dimensionality (n=vocabulary)
- No representation of similar meaning between words.

“It turns out that dense vectors work better in every NLP task than sparse vectors” (Jurafsky, D. & Martin, J., 2024)

Word Embeddings

- **Word Embedding:**
  The transformation of words into vectors in a continuous vector space.

- Useful property: Semantically related words are located nearby each other.

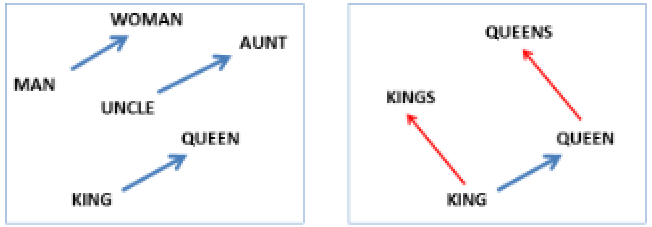
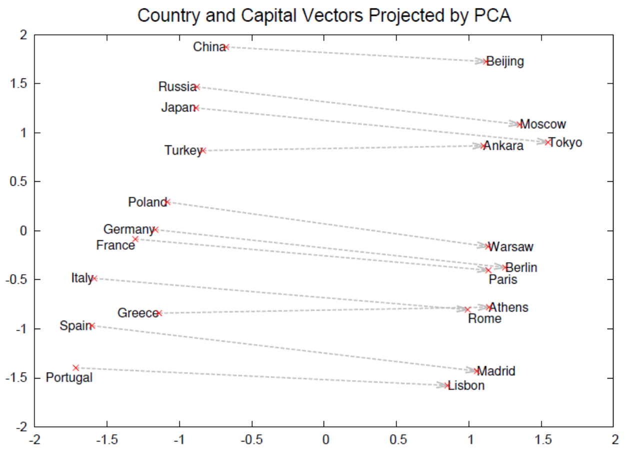


Figure 2: Left panel shows vector offsets for three word pairs illustrating the gender relation. Right panel shows a different projection, and the singular/plural relation for two words. In high-dimensional space, multiple relations can be embedded for a single word.

"Linguistic Regularities in Continuous Space Word Representations" (Mikolov et al, NAACL 2013)



"Distributed Representations of Words and Phrases and their Compositionality" (Mikolov et al, 2013)

Example

pepperoni_pizza
Pizza
pizzas
sandwich
Domino_pizza
commuter_train
train
freight_train
Train
locomotive
salmon
striped_bass
trout
fishes
catfish
Kremlin
Russia
France
Hopital_Europeen_Georges_Pompidou
EURASIAN_NATURAL_RESOURCES_CORP
cars
Tbilisi
Pantheon Sorbonne
Spyker_D##_Peking
vehicle
Kiev
Parisian
Islamabad_Slyvia_Hui
truck
SUV
Minsk
Londons
minivan
Wandsworth
Canary Wharf
social_networking
Facebook.com
tweets
Facebook
tweet
Twitter
FaceBook
facebook
twitter
Twitter.com
Paris
London
Moscow
Twitter
Facebook
pizza
fish
train
car
100
50
0
-50
-100
-150
-100
-50
0
50
100
150

Common Algorithms

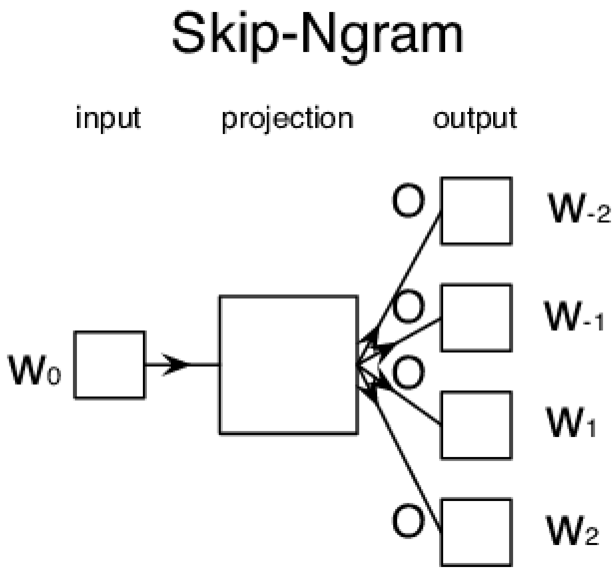
- **Word2Vec:**
  Uses shallow neural networks to create word embeddings. Two variants: Skip-gram and Continuous Bag of Words (CBOW).

- **GloVe:**
  Generates embeddings by factorizing the word co-occurence matrix. It is a count-based model that captures both local and global semantics.

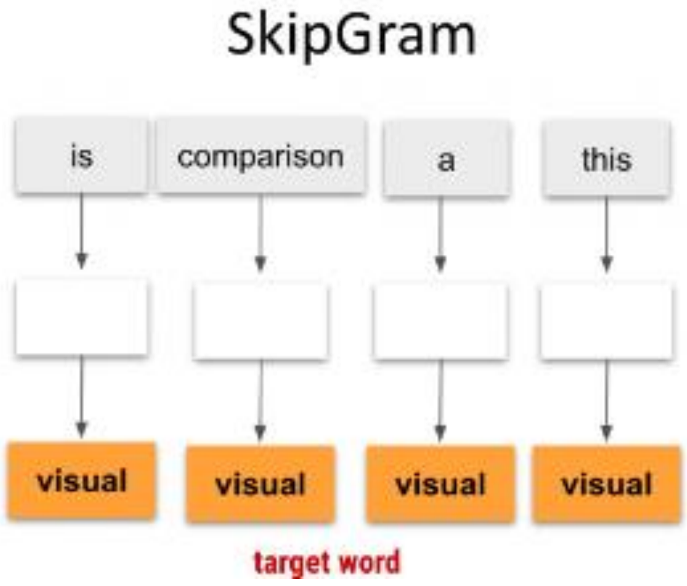
- **BERT and ELMo:**
  Bidirectional models that translates words in context to vectors. ELMo uses LSTMs and BERT uses the Transformer architecture.

Word2Vec: Skip-Gram

- **Task:** Given a word, predict the context



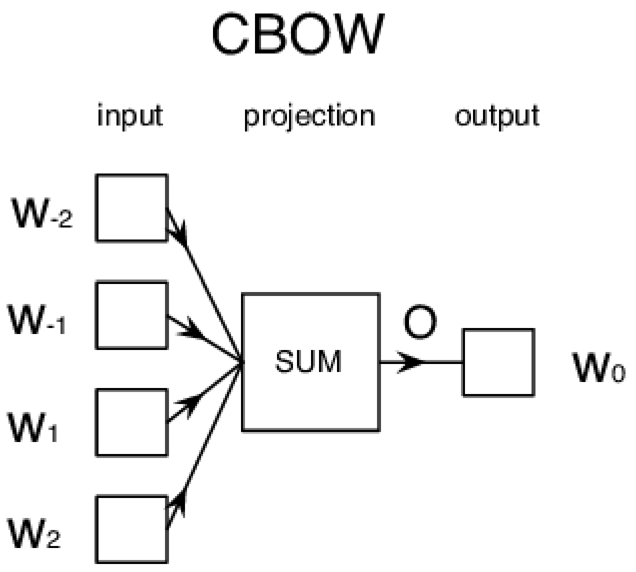
This is a visual comparison



https://kavita-ganesan.com/comparison-between-cbow-skipgram-subword/

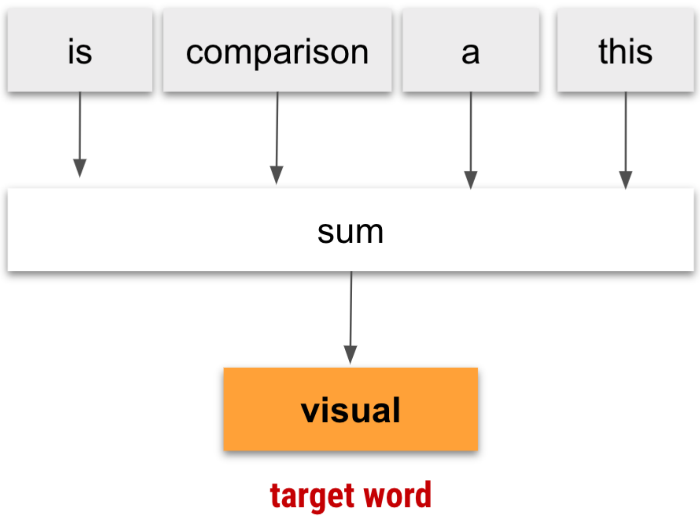
Word2Vec: Continuous Bag of Words

- **Task:** Given a context, predict the word



This is a visual comparison

CBOW



https://kavita-ganesan.com/comparison-between-cbow-skipgram-subword/

Semantics vs. Syntax

- Show the most similar words to the word.
- Similarity by these models is both based on semantics and syntax
- Research shows that CBOW is better at modeling syntax and Skip-gram is better at modeling semantics.
  But the difference is very small.

Word: Negative

CBOW	Skip-gram
Positive	Positive
Logical	Psychological
Rational	Particulate
Superstitious	Substance
Dangerous	Severity
Subjective	Damaging
Meaningless	Wildly
Weak	Definite
Trickier	Promiscuity
Significant	Harmful

Similarity Concepts

- Similarity is hard to define.
- Different Word2Vec models excess at different types of similarities

	a_word	b_word	concept_type	score_cbow	score_skipgram
0	friendly	staff	neighboring	0.114944	0.749117
1	shower	curtain	neighboring	0.262860	0.717065
2	very	clean	neighboring	0.397924	0.678147
3	hotel	property	synonymous	0.807957	0.667862
4	dirty	filthy	synonymous	0.865373	0.878625
5	washroom	bathroom	synonymous	0.766167	0.801310
6	staff	staffs	near_duplicates	0.830538	0.623231
7	calendar	calender	near_duplicates	0.209005	0.497958
8	bathroom	bathrooms	near_duplicates	0.165828	0.506214

https://kavita-ganesan.com/comparison-between-cbow-skipgram-subword/

Jheronimus Academy of Data Science

Sentence Similarity

phrase1	phrase2	similar	cbow_sim	skipgram_sim
polite staff	rude staff	0	1	1
friendly manager	rude manager	0	1	1
room was huge	large rooms	1	0	1
staff was friendly	very polite manager	1	1	1
bathroom was very dirty	filthy bathroom	1	1	1
clean and tidy rooms	the room was a mess	0	0	1
the views were awesome	the breakfast was nice	0	0	1
what lovely breakfast	friendly staff	0	0	0
would recommend	highly recommended	1	0	1
the manager was rude	staff were arrogant and rude	1	1	1
good breakfast selection	variety of breakfast items	1	1	1

https://kavita-ganesan.com/comparison-between-cbow-skipgram-subword/

Pre-trained vs self-trained word embeddings

<table>
<tr>
<td>

Pre-trained embeddings

Advantages:

+ Captures broad range of language features, as it is trained on a large corpus.
+ Good for tasks requiring a broad language understanding

Disadvantages:

- Not tailored to specific domain
- Out-of-vocabulary words
- Memory inefficient

</td>
<td>

Self-trained embeddings

Advantages:

+ Domain specific representations
+ Little out-of-vocabulary words

Disadvantages:

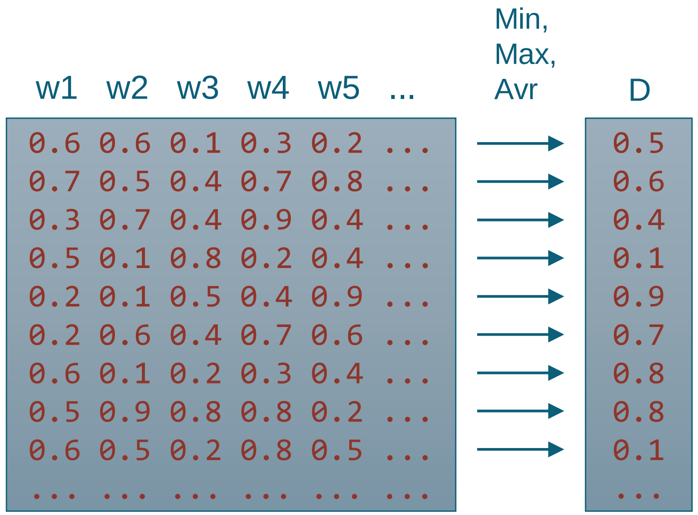
- Trained on less data
- More computationally expensive

</td>
</tr>
</table>

Creating Document Embeddings

- For many applications, document embeddings are needed instead of word embeddings.

- Averaging of word embeddings is the most popular and often effective method to calculate document embeddings, but min-pooling and max-pooling are sometimes used.
- In min- and max-pooling, the lowest or highest value for each dimension is taking.



Jheronimus Academy of Data Science

Practical Session

Niels Scholten | JM2050-M-6 Natural Language Processing

JADS is powered by TU/e Eindhoven University of Technology Tilburg University

JADS Jheronimus Academy of Data Science

Today

- Train Word2Vec Embeddings on the 20Newsgroup dataset

```
from sklearn.datasets
import fetch_20newsgroups
from nltk.tokenize import word_tokenize
from gensim.models import Word2Vec
import re
# Fetch the 20 Newsgroups dataset
newsgroups = fetch_20newsgroups(subset='all', remove=('headers', 'footers',
'quotes'))
```

Preprocessing

```
def preprocess_text(text):
        text = text.strip()
        words = word_tokenize(text.lower())
        alpha_words = [word for word in words if word.isalpha()]
        return alpha_words

preprocessed_data = [preprocess_text(document) for document in newsgroups.data]
```

Training the Word2Vec Model

```
cbow_model = Word2Vec(
          sentences=preprocessed_data,        # Input tokens

          vector_size=500,                    # Embedding dimensions

          window=5,                           # Considering the number of
                                              # words on both sides of the word

          min_count=1,                        # Minimum frequency of words

          sg=0,                               # Indicates method
          )                                   # Skip-gram (sg=1) CBOW (sg=0)
```

For training TF-IDF embeddings, look at
`sklearn.feature_extraction.text.TfidfVectorizer`

Accessing Embeddings

All embeddings can be obtained as follows:

`Embeddings = cbow_model.wv.vectors`

To get specific embeddings: `model.wv[“computer”]`

`Wv.index_to_key and wv.key_to_index map index to words`

Index to key:

```
i2k= cbow_model.wv.index_to_key
for i in [0, 167, 503, 78527]:
     print(f'word {i} in vocabulary: {i2k[i]}’)
>>> word 0 in vocabulary: the
>>> word 167 in vocabulary: data
>>> word 503 in vocabulary: science
>>> word 78527 in vocabulary: definate
```

Key to index:

```
k2i= cbow_model.wv.key_to_index
for i in ['data', 'science', 'the','definate’]:
    print(f'word {i} in vocabulary: {k2i[i]}’)
>>> word data in vocabulary: 167
>>> word science in vocabulary: 503
>>> word the in vocabulary: 0
>>> word definate in vocabulary: 78527
```

JADS Jheronimus Academy of Data Science

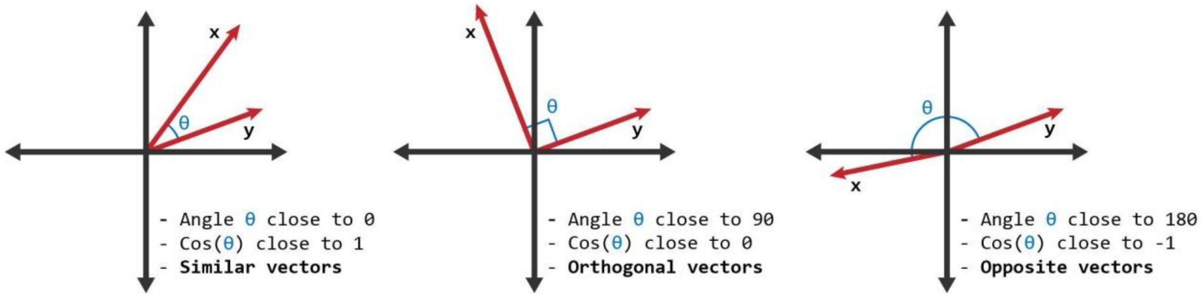
Calculating Similarity

- Similarity of vectors can be calculated on various ways, but cosine similarity is the most popular.

```
from sklearn.metrics.pairwise import cosine_similarity

vector_computer = cbow_model.wv['computer'].reshape(1,-1)
vector_network = cbow_model.wv["network"].reshape(1, -1)
vector_bird = cbow_model.wv['bird'].reshape(1,-1)

print(cosine_similarity(vector_computer, vector_network))
print(cosine_similarity(vector_computer, vector_bird))
>>> [[0.8103024]]
>>> [[0.2295769]]
```



- Cosine similarity can be used to calculate the similarity of all kinds of vectors. TF-IDF & Word2Vec, Word & Document Embeddings.

JADS Jheronimus Academy of Data Science

Gensim implementation

```
cbow_model.wv.most_similar(“car”)

[
('bike', 0.8462114334106445),
('battery', 0.7564215660095215),
('dealer', 0.7380539774894714),
('snazzy', 0.7085026502609253),
('oil', 0.7017979025840759),
('helmet', 0.69758540391922),
('bought', 0.6896640658378601),
...
]
```

Jheronimus Academy of Data Science

Other Gensim Features

Vector Arithmetics:

```
Model.vw.most_similar(positive=[‘woman’, ‘king’], negative=[“man”])
```

Closer than:

```
Model.wv.words_closer_than(“lion”, “cat”)
```

Does not match:

```
Model.wv.doesnt_match([“breakfast”, “lunch”, “frog”])
```

Exercise

Niels Scholten | JM-6-2024 Natural Language Processing

JADS is powered by TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY TILBURG UNIVERSITY

JADS Jheronimus Academy of Data Science

Overview

- Train Word2Vec models on the provided datasets. The goal is to familiarize yourself with the periously discussed concepts.

- You will work with the following datasets:
  - AGNews
  - IMBD

Loading the data

```
import re
def read_and_preprocess_txt(file_path):
    """
    Reads a .txt file, preprocesses each line by lowercasing,
    keeping only alphabetic characters and filtering out words with length less than 3.
    Tokenizes the preprocessed line into a list of words.
    Returns a list of lists where each inner list is a tokenized and preprocessed line from the file.
    Parameters:
    - file_path (str): The path to the .txt file to be read.
    Returns:
    - list: A list of lists, where each inner list is a tokenized and preprocessed line from the
    file.
    """
    with open(file_path, 'r', encoding='utf-8') as file:
        return [[word for word in re.sub('[^a-zA-Z\s]', '', line.lower().strip()).split() if len(word) >= 3] for line in file]
```

Exercise

- Start training a word2vec model on the preprocessed AGNews dataset using CBOW and Skipgram.
  - For both models, what are the 10 words most similar to the words ‘amsterdam’?
  - How do the results differ from each other?

- Now train a word2vec with both models on the imdb dataset and time your training.
  - Which training went faster?
  - In the IMDB dataset what is the resulting vector if you subtract ‘man’ from ‘uncle’ and add ‘woman’? What about doing the same on the AGNews dataset? What causes the difference?
  - What are the differences between CBOW and Skipgram?

- Create a document embedding, using max-pooling, for the first 10 documents in the AGNews dataset.
  - Which document is most similar to the first document?

JADS Jheronimus Academy of Data Science

Contact

Niels Scholten

n.c.scholten@tue.nl