



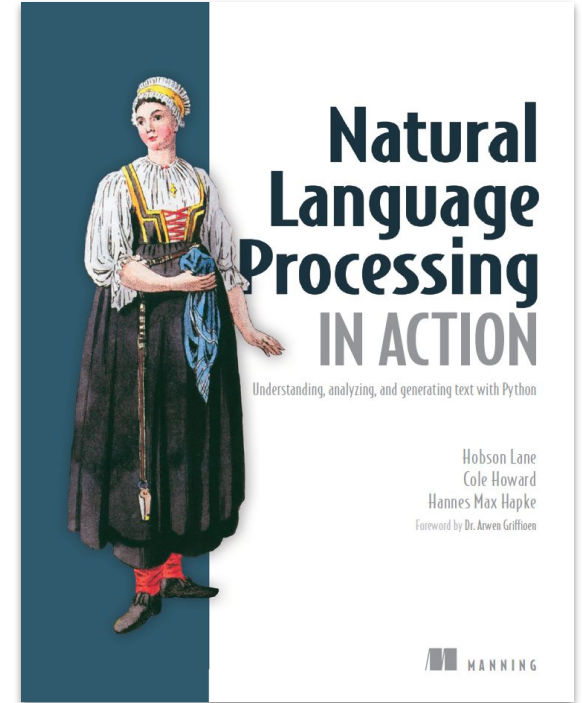
NLP

TEXT REPRESENTATION

BOOK RECOMMENDATION

Lane, Hobson, et al. *Natural Language Processing in Action: Understanding, Analyzing, and Generating Text with Python*.
Manning Publications Co, 2019.

You can find the example code in [this notebook](#).



BAG OF WORDS

RAW WORD COUNTS

A **bag of words** is a dictionary with counts of the occurrence of the single words in a text.

```
{  
  "wander": 31,  
  "turn": 5,  
  "focus": 4,  
  "work": 4,  
  "call": 3  
  ...  
}
```

```
%sql  
select  
  verb,  
  occurrences  
from  
  ted_top_verbs  
where  
  youtube_id = "Qy5A8dVYU3k"
```

► (2) Spark Jobs

	verb	occurrences
1	wander	31
2	turn	5
3	focus	4
4	work	4
5	call	3

BAG OF WORDS

NORMALIZED TERM FREQUENCY

Instead of raw counts, it is useful to calculate a relative occurrence to the length of the text.
This is called the **normalized term frequency**.

```
{  
  "work": 0.2384,  
  "turn": 0.0384,  
  "focus": 0.0307,  
  "work": 0.0307,  
  "call": 0.0230  
  ...  
}
```

```
%sql  
select  
  verb,  
  occurrences / total_verbs as norm_term_freq  
from  
  ted_top_verbs ttv  
  inner join (  
    select  
      youtube_id,  
      sum(occurrences) as total_verbs  
    from  
      ted_top_verbs  
    group by  
      youtube_id  
  ) s on s.youtube_id = ttv.youtube_id  
where  
  ttv.youtube_id = "QySABdVU3k"  
order by occurrences desc
```

▶ (3) Spark Jobs

	verb	norm_term_freq
1	wander	0.23846153846153847
2	turn	0.038461538461538464
3	focus	0.03076923076923077
4	work	0.03076923076923077
5	call	0.023076923076923078

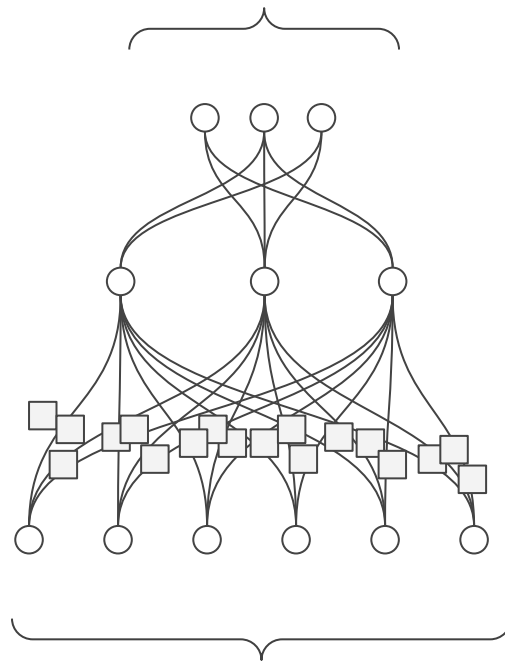
BAG OF WORDS

APPLICATION AND LIMITS

- Good basis for **rule-based analysis**, such as sentiment, topic identification, spam-filters
- Not suitable as input for machine learning algorithms; they require numeric values

How can we represent text numerically?

The output (prediction) is numeric, too



ML models like neural networks are purely mathematical objects and require numeric input

ONE HOT ENCODED VECTORS

SPARSE REPRESENTATION

A **one hot encoded vector** is a sparse vector with only 0 and a single 1 for the index of the word it represents.

“This is my absolute
undisputable favorite tea
right now”



absolute	0	0	0	1	0	0	0	0	0
favorite	0	0	0	0	0	1	0	0	0
is	0	1	0	0	0	0	0	0	0
my	0	0	1	0	0	0	0	0	0
now	0	0	0	0	0	0	0	0	1
right	0	0	0	0	0	0	0	1	0
tea	0	0	0	0	0	0	1	0	0
this	1	0	0	0	0	0	0	0	0
undisputable	0	0	0	0	1	0	0	0	0

The length of each vector depends on the size of the vocabulary. Large vectors are >99% filled with zeroes, which makes them inefficient.

WORD EMBEDDINGS

A VECTOR OF NUMBERS

Word embeddings are learned using a machine learning algorithm such as **word2vec**

A **word embedding** is the representation of a word through a vector of numbers (floats).

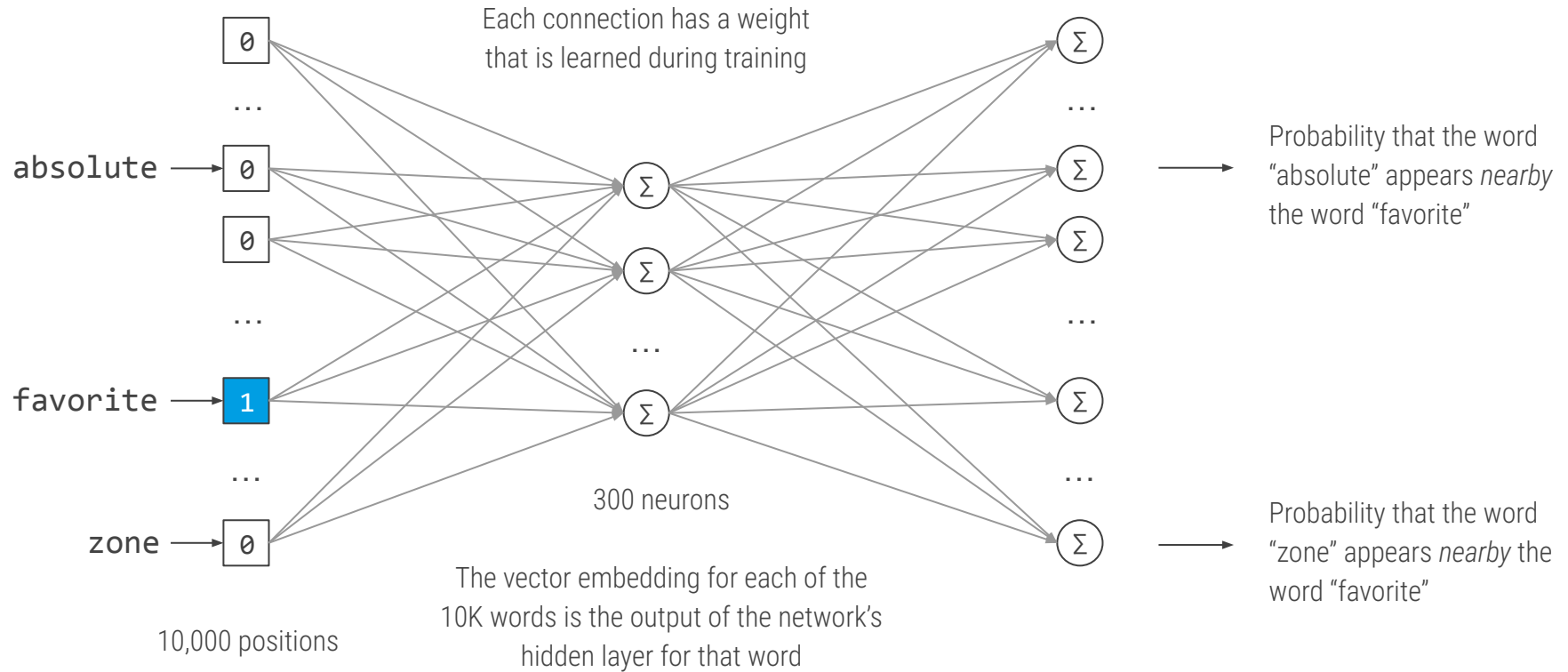
Vectors of contextually similar words are closer to each other in the euclidean space than others.

```
doc = nlp("This is my absolute undisputed favorite tea right now.");  
print("The token '{}' has the following word2vec vector embedding:".format(doc[3].text,  
doc[3].vector)
```

```
The token 'absolute' has the following word2vec vector embedding:  
Out[112]: array([-1.4459e-01,  2.2050e-01,  8.6909e-02,  3.3820e-01,  2.2789e-01,  
                -2.4581e-01,  1.3967e-01,  2.6703e-01,  9.2204e-02,  1.6055e+00,  
                8.6824e-02,  1.7958e-01, -2.6495e-01,  6.3712e-01,  3.9218e-01,  
                -2.6489e-01, -4.7509e-01,  1.5260e+00,  9.0911e-02,  4.9902e-01,  
                -1.4884e-01, -7.6880e-01,  1.4809e-01,  3.5931e-02, -6.2046e-02,  
                -2.8647e-01,  1.9036e-01,  5.6531e-02,  1.1770e-02,  7.0728e-02,  
                2.9888e-01,  6.4778e-01,  2.2893e-01,  1.0843e+00, -1.2830e-01,  
                -3.7705e-01, -1.8517e-01, -2.4000e-01, -1.0283e-01, -3.6733e-01,  
                1.3703e-01, -4.2059e-02, -2.1249e-01,  3.4027e-01,  1.7061e-01,  
                -8.6444e-02,  2.2293e-02,  5.2327e-01, -2.5780e-01, -1.0093e-01,  
                1.8023e-01,  4.0808e-01, -1.6114e-01, -8.7858e-02,  4.2435e-01,  
                1.3158e-03, -2.1900e-01, -8.7514e-02,  1.5678e-01, -1.5575e-01,  
                -7.5321e-03,  2.8298e-01, -2.2250e-01, -2.2584e-01, -1.0050e-01,  
                3.3866e-01,  3.1441e-01, -2.1194e-01,  2.4665e-02, -3.3567e-01,
```

WORD EMBEDDINGS

LEARNING THE VECTORS



WORD EMBEDDINGS

VIDEO RECOMMENDATION



WORD EMBEDDINGS

EXAMPLE spaCy

spaCy's medium English model has a **vocabulary of > 700,000 words** with vector embeddings.

A single vector has **300 dimensions**

```
doc = nlp("This is my absolute undisputed favorite tea right now.");
print("The token '{}' has the following word2vec vector embedding:".format(doc[3].text))
print("Each vector has {} dimensions".format(len(doc[3].vector)))
doc[3].vector
```

```
The token 'absolute' has the following word2vec vector embedding:
Each vector has 300 dimensions
Out[117]: array([-1.4459e-01,  2.2050e-01,  8.6909e-02,  3.3820e-01,  2.2789e-01,
        -2.4581e-01,  1.3967e-01,  2.6703e-01,  9.2204e-02,  1.6055e+00,
         8.6824e-02,  1.7958e-01, -2.6495e-01,  6.3712e-01,  3.9218e-01,
        -2.6489e-01, -4.7509e-01,  1.5260e+00,  9.0911e-02,  4.9902e-01,
        -1.4884e-01, -7.6880e-01,  1.4809e-01,  3.5931e-02, -6.2046e-02,
        -2.8647e-01,  1.9036e-01,  5.6531e-02,  1.1770e-02,  7.0728e-02,
         2.9888e-01,  6.4778e-01,  2.2893e-01,  1.0843e+00, -1.2830e-01,
        -3.7705e-01, -1.8517e-01, -2.4000e-01, -1.0283e-01, -3.6733e-01,
         1.3703e-01, -4.2059e-02, -2.1249e-01,  3.4027e-01,  1.7061e-01,
```

```
import spacy
nlp = spacy.load("en_core_web_md")

vocab_size = len(nlp.vocab.strings)
print("The English model in medium size has a vocabulary of {} words.".format(vocab_size))

n_keys = nlp.vocab.vectors.n_keys
print("The English model in medium size has {} unique word embeddings (vectors)".format(n_keys))
```

```
The English model in medium size has a vocabulary of 701570 words.
The English model in medium size has 684830 unique word embeddings (vectors)
```

With word embeddings, we can calculate **similarities** between words and documents.

```
tea = nlp("I love tea")
coffee = nlp("I love coffee")
pizza = nlp("I love pizza")
pasta = nlp("I love pasta")

print("Tea and coffee: {}".format(tea.similarity(coffee)))
print("Tea and pizza: {}".format(tea.similarity(pizza)))
print("Tea and pasta: {}".format(tea.similarity(pasta)))
print("Pizza and pasta: {}".format(pizza.similarity(pasta)))
```

```
Tea and coffee: 0.9411059275089753
Tea and pizza: 0.8494171567369873
Tea and pasta: 0.8388414815173865
Pizza and pasta: 0.9358318464113806
```

WORD EMBEDDINGS

ARITHMETIC

We can even do arithmetic based on learned vector embeddings!

King - Man + Woman = ?

The difference vector of
(King - Man + Woman) - Queen
contains only numbers close to zero.

```
nlp = spacy.load('en_core_web_md')
doc = nlp('queen king woman man')
queen, king, woman, man = doc[0].vector, doc[1].vector, doc[2].vector, doc[3].vector
vec = king - man + woman
vec - queen
```

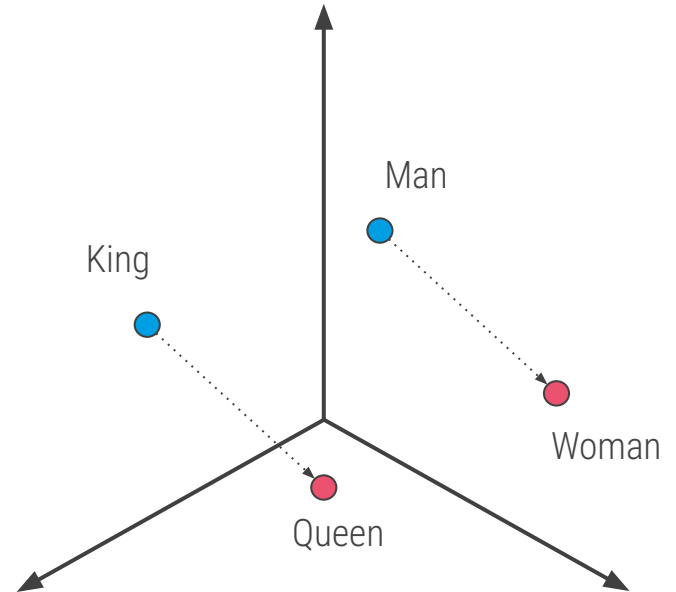
```
Out[147]: array([ 0.10458702, -0.05152999, -0.01085299,  0.40603995,  0.111525 ,
  0.03181005, -0.18277001,  0.10793996,  0.22586 ,  0.42549992,
 -0.620518 ,  0.09305897, -0.0758817 , -0.29067168, -0.297841 ,
 -0.43369 , -0.44859397,  0.21168 , -0.172735 ,  0.24211 ,
  0.20211 , -0.15502006, -0.04844499, -0.202636 , -0.21129996,
  0.457768 ,  0.03138995,  0.13294101, -0.534806 , -0.07134694,
 -0.157518 , -0.05403006, -0.14246997, -0.773906 ,  0.15866998,
 -0.12601201, -0.19204 , -0.40347007,  0.05978 ,  0.5203604 ,
  0.37192 , -0.252379 , -0.097138 , -0.40504098,  0.25123 ,
 -0.03785798, -0.11933102, -0.00672996,  0.40258 ,  0.02721703,
 -0.29956898,  0.34834102, -0.15371901, -0.14056298,  0.17291501,
  0.73967993, -0.0257776 , -0.28438202, -0.337454 ,  0.12431702,
```

WORD EMBEDDINGS

ARITHMETIC

The **relative position** of “King” and “Queen” in the multidimensional vector space is similar to the one of “Man” and “Woman”.

$$\text{King} - \text{Man} + \text{Woman} = \text{Queen}$$



WORD EMBEDDINGS

SIMILARITY

BUT: The same words have the same vector embeddings, no matter the context :-)

```
city = nlp("Berlin is the capital of Germany.")
money = nlp("The firm needed more capital to invest.")
print(city[3])
print(money[4])
print(city[3].similarity(money[4]))
```

```
capital
capital
1.0
```

The word "capital" can have two meanings; static word embeddings do not account for context.

TAKE CONTEXT INTO ACCOUNT

LSTM, SEQ2SEQ, TRANSFORMERS

Long short-term memory (LSTM) networks, sequence-to-sequence models and attention-based transformer networks take context into account and extend the NLP capabilities.

Transformer networks became particularly famous through the release of BERT in 2019 and GPT-3 in 2020

GPT-3 facts:

> 170 billion parameters

~ 500 billion tokens of training data

~ 4.7 million USD training costs