Natural Language Processing

Prof. Dr. Jannik Strötgen
jannik.stroetgen@h-ka.de
Summer 2024

Hochschule Karlsruhe
University of
Applied Sciences

Fakultät für
Informatik und
Wirtschaftsinformatik

Recap

Similarity and Search

Types of Similarity in Language

Surface form similarity:

- Phonological similarity (e.g., `brake | break`)
- Morphological similarity (e.g., `respect | respectful`)
- **Spelling similarity** (e.g., `theater | theatre`)

Semantic similarity:

- Synonymy (e.g., `verbose | wordy`)
- Hyponymy (e.g., `color | red`)

Content similarity:

- Sentence similarity (e.g., paraphrases)
- Document similarity (e.g., two news stories on the same event)

The Term-Document Matrix

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Mcbeth	...
Antony	1	1	0	0	0	1	...
Brutus	1	1	0	1	0	0	...
Caesar	1	1	0	1	1	0	...
Calpurnia	0	1	0	0	0	0	...
Cleopatria	1	0	0	0	0	0	...
mercy	1	0	1	1	1	1	...
worser	1	0	1	1	1	1	...

Corpus representation:

- Columns in the matrix represent documents, rows represent terms
- Non-zero values indicate that a given term occurs in a given document

TF-IDF Weighting

We combine the term frequency and the inverse document frequency to assign a `tf-idf` weight `w` to each term `t` in each document `d`:

$$w_{t,d} = (1+\log_{10} tf_{t,d}) \log_{10} \frac{N}{df_t}$$

The weight:

- Increases with the number of occurrences of a term within a document
- Increases with the rarity of the term in the collection

Cosine Similarity

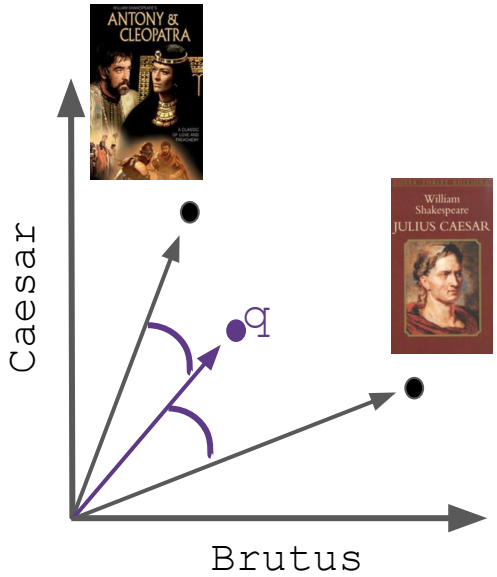
We can use the cosine function to measure the similarity of two documents in tf-idf vector representation (or between a query and a document):

$$\cos(\vec{q}, \vec{d}) = \text{SIM}(\vec{q}, \vec{d}) = \frac{\vec{q} \cdot \vec{d}}{|\vec{q}||\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2} \sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

q_i is the tf-idf weight of term i in the query.

d_i is the tf-idf weight of term i in the document.

$|\vec{q}|$ and $|\vec{d}|$ are the lengths of $\vec{q}$ and $\vec{d}$.



Outline

Outline of the Course

1. Foundations and Pre-processing
2. Part-of-speech Tagging
3. Parsing
4. Named Entity Recognition and Linking
5. Similarity and Search
6. **Language Models: Static Word Embeddings**
7. Contextual Language Models
8. Text Mining (Classification, Clustering, and Topic Models)
9. Opinion Mining and Sentiment Analysis
10. Relation Extraction and Question Answering

Outline for Today

Language Models: Static Word Embeddings

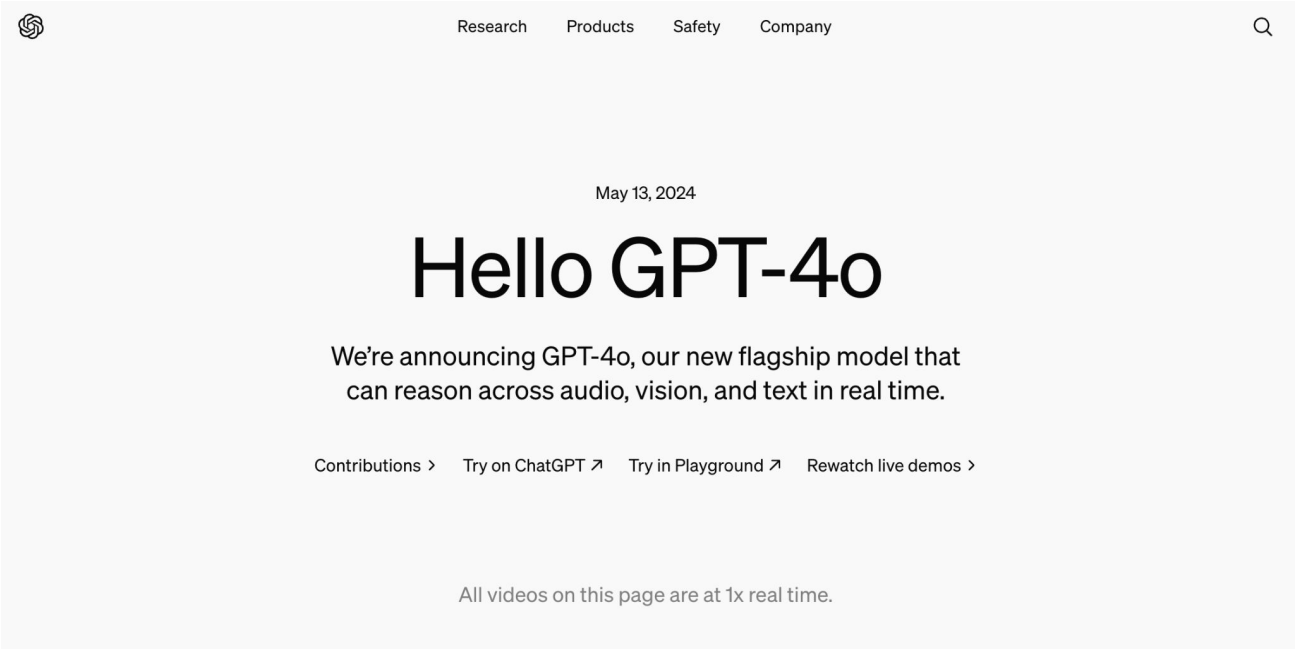
1. Language Models
2. Limitations of Discrete Models of Language
3. Word Embeddings
4. Latent Semantic Analysis (LSA)
5. word2vec
6. Compositional Semantics

Language Models

Let’s Aim High: Talking to Computers

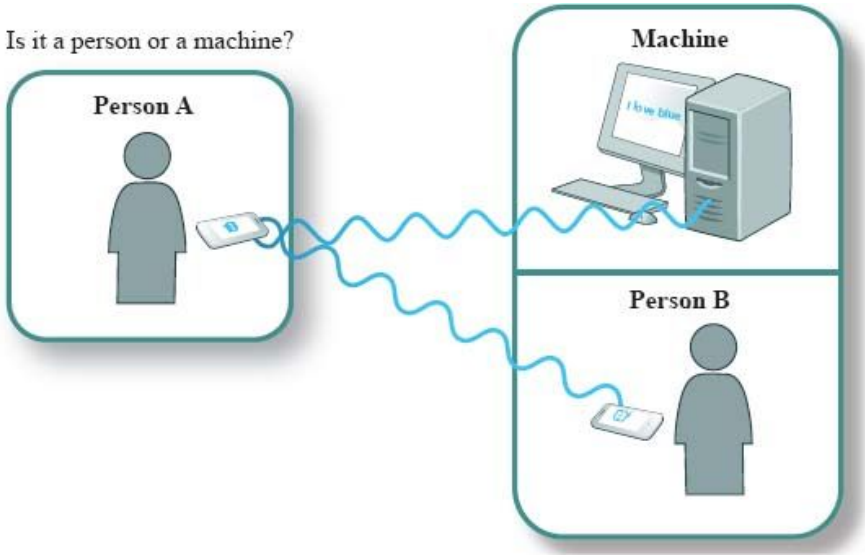
What do we need to make these happen?

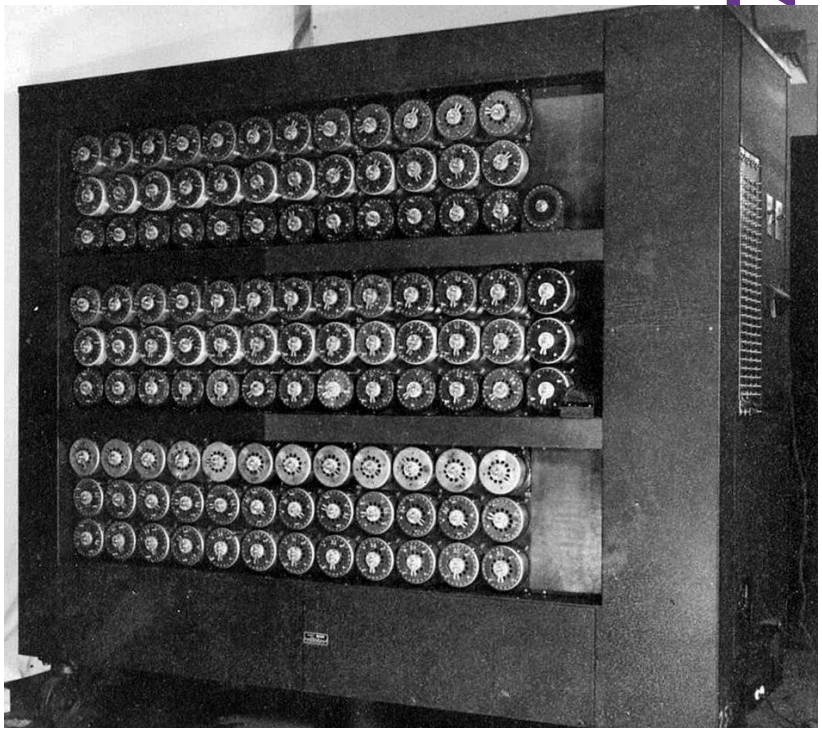
Let’s Aim High: Talking to Computers



What did we need to make this happen?

Historical Excursion: The Turing Test





Turing test:
Can a machine pretend to be a human well enough in a (blind) conversation to fool a human into thinking it is human?

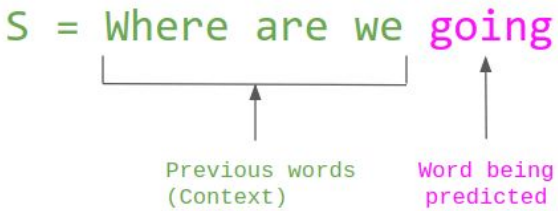
Applications of Language Models

- Natural language understanding (NLU)
  - Question answering systems
  - Chatbots
- Natural language generation (NLG)
  - Text summarization
  - Automated journalism
  - Chatbots
- Machine translation
- Text classification
- Named entity recognition
- Opinion mining
- Sentiment analysis
- ...

Language Models: Definition

Language model:
A language model is a probability distribution over sequences of words.

By using a language model, we can assign probabilities to words, given a sequence of other words. That is, we can **predict** the occurrence of words.



$$P(S) = P(\text{Where}) \times P(\text{are} \mid \text{Where}) \times P(\text{we} \mid \text{Where are}) \times P(\text{going} \mid \text{Where are we})$$

Language model examples:

- n-grams
- Static embeddings (today)
  - Word2vec
  - GloVe
  - FastText
  - etc.
- Contextualized word embeddings
  - GPT-2 / GPT-3 / …
  - BERT
  - WuDao 2.0
  - T5
  - etc.

Limitations of Discrete Models of Language

A Core NLP Task: Word Similarity

Many applications in text processing and information retrieval rely on **word similarity** as a core task that needs to be solved.

- Spell checking
  - Similarity between **individual words**
- Search
  - Similarity between a **sentence** and the content of a **document**
- Duplicate detection
  - Similarity between two **documents**
- Summarization
  - Removal of redundant (= similar) **sentences** in a document
- Translation
  - Finding a similar **word** in a different language

A similarity between sentences or documents can often be derived from **word** similarities.

Word Similarity in Manually Created Models

We already know that manually curated models of language like **WordNet** are a great resource that can be used to compute word similarities. But:

- WordNet is missing nuance
  - For example, `proficient` is a synonym for `good`.
  - `Proficient cop, bad cop` is different from `Good cop, bad cop`.
  - This type of similarity is only correct in some contexts.
- WordNet is missing **new meanings** of words until they are added
  - `Streaming` (on Twitch), `salty` (= angry), etc.
  - Impossible to keep manually curated resources up-to-date
- WordNet is subjective and biased by the annotators' perspectives
- Requires **constant human labor** to create and update

Similarity in the Vector Space Model

We also know how to compute similarities between a query sentence and a document in the **vector space model**:

- Represent the **query** as a tf-idf encoded vector
- Represent each **document** as a tf-idf encoded vector
- Compute cosine similarities between query and documents

Can we use this to handle word similarities? Not really…

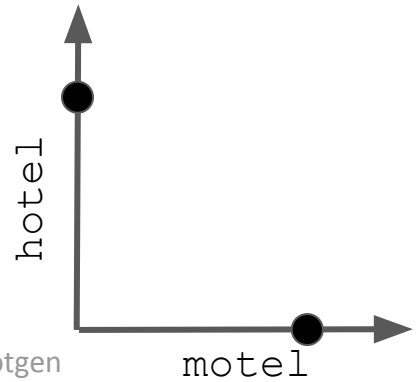
- Word 1: `Hotel`, w_1 = `[0, 0, 0, 0, 1, 0, 0]`
- Word 2: `Motel`, w_2 = `[0, 0, 0, 1, 0, 0, 0]`
- `CosineSim(`w_1`,`w_2`)` = 0
- Normalizing the values of w_1 and w_2 with tf-idf does not change the similarity.

One-Hot Vector Encodings

In the vector space model, we are using a **localist** representation:

- The **dimension** of the vector space is equal to the **vocabulary** size
- Each word (or lemma, or stem) corresponds to exactly one dimension
- A single word is encoded by a **one-hot** vector:
  - All vector components are equal to 0 (these components are “cold”)
  - Only the component in the dimension that corresponds to the word itself has a value of 1 (the component is “hot”)
- Individual word vectors are orthogonal by definition

⇨ There is **no notion of word similarity** in the vector space model



Fundamental Limitations of the Vector Space Model

The vector space model is designed to compare sequences of text based on the words they contain. But even that is problematic. Consider the search queries:

- Query: `Karlsruhe Motel` $q_1 = [0, 1, 0, 0, 1, 0, 0]$
- Doc 1: `...Karlsruhe Hotel...` $d_1 = [0, 1, 0, 1, 0, 0, 0]$
- Doc 2: `...Hochschule Karlsruhe...` $d_2 = [0, 1, 0, 0, 0, 0, 1]$

A traveler from the U.S. who is searching for a motel in Karlsruhe is equally likely to find hotel websites as they are to find websites related to the university.

Historical solution in information retrieval: **Query expansion**

- Keep a dictionary of similar words and use it to expand the queries
- E.g., if a user searches for **`motel`**, also add **`hotel`** to the query

Towards Semantically Meaningful Word Representations

Query expansion is a (set of) heuristic(s) specifically designed to work around the inability of the vector space model to represent semantic meaning.

For handling word similarities, the vector space model won't do! Can we do better?

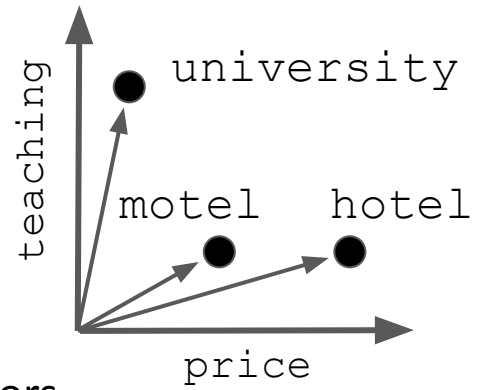
Ideas?

Dense Word Vector Representations

In theory:
Instead of a **term-document matrix**, we want a **term-concept matrix**, in which each word is ranked according to the strength of its relation to a semantic concept. For example:

	price	teaching
hotel	0.9	0.1
motel	0.5	0.1
university	0.1	0.8

- `Hotel` and `motel` are very similar and differ mostly in room prices and the quality of service.
- They are both similarly low on a scale that rates the relevance for teaching, while a `university` is higher on that scale.



The rows of such a matrix could then be used as vector representations of the words. Similar words would have similar vectors.

Dense Word Vector Representations

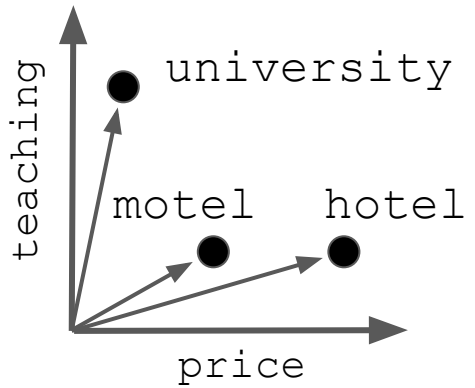
In practice:

How can we even create such a matrix? Manual curation is entirely infeasible:

	price	teaching
hotel	0.9	0.1
motel	0.5	0.1
university	0.1	0.8

- How do we decide on the values?
- How do we choose the concepts?
- We still have to update the model with new words
- We also have to update in case of semantic shifts (e.g., `gay` in the 1930s vs. today)

⇨ We need a way of learning dense vector representations from data.

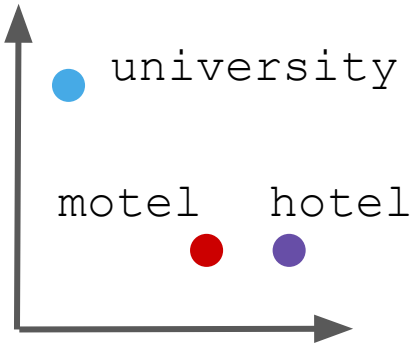


Word Embeddings

Word Embeddings: Definition

Embedding of words:
Given a training corpus, the embedding of words is the **process** of assigning a vector (in a **latent space**) to each word, based on the content of the corpus (by using some algorithm).

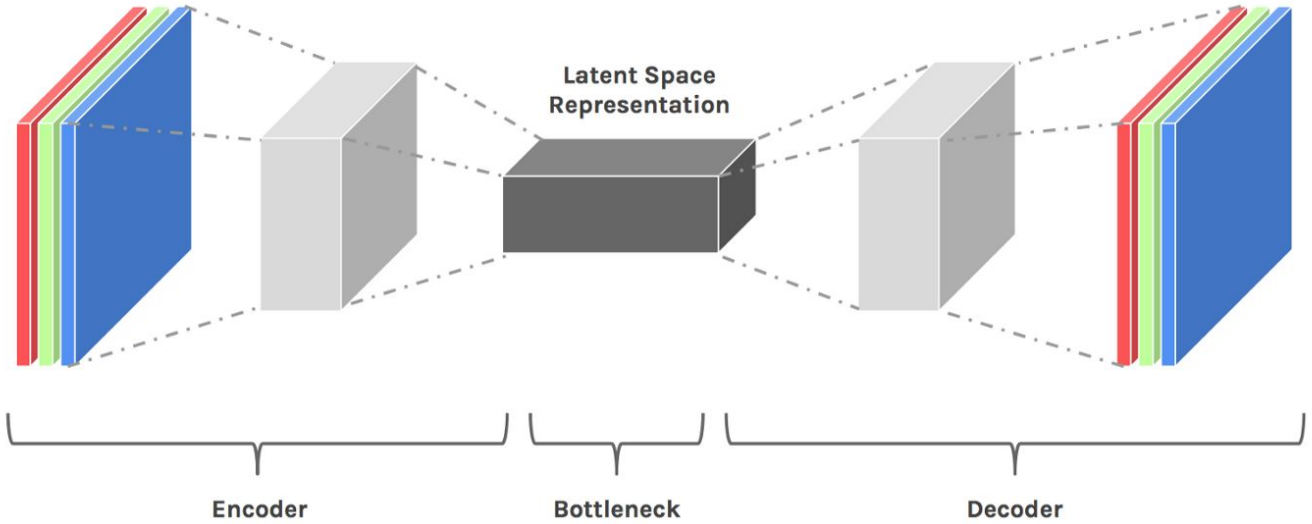
Word embedding:
The **vector** that is assigned to a word as the result of the above process is often called the embedding of the word.



Karlsruhe is a German city that has a university and several hotels, but no motels.

Latent Space

A **latent space** (also called embedding space) is a vector space in which items are placed in such a way that **similar items are placed in proximity**. Typically, latent spaces have a lower rank (dimensionality) than the original space of the data.



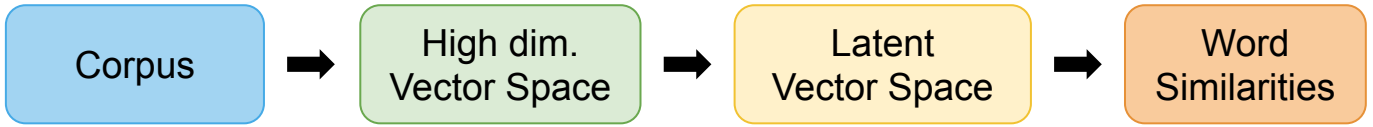
https://towardsdatascience.com/understanding-latent-space-in-machine-learning-de5a7c687d8d

Embedding Spaces for Words

For creating word embeddings, the embedding algorithm should:

- Place similar and/or related words in close proximity in the latent space
- Use word (co-)occurrence information from the corpus
- Work without supervision (labeling data at this this scale is infeasible)

Embedding Pipeline:



Latent Semantic Analysis

Dimensionality Reduction of Term-Document Matrices

Corpus Size ⟹

Vocabulary Size ⇓

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Mcbeth	...
Antony	5.25	3.18	0	0	0	0.35	...
Brutus	1.21	6.10	0	1.0	0	0	...
Caesar	8.59	2.54	0	1.51	0.25	0	...
Calpurnia	0	1.54	0	0	0	0	...
Cleopatria	2.85	0	0	0	0	0	...
mercy	1.51	0	1.90	0.12	5.25	0.88	...
worser	1.37	0	0.11	4.15	0.25	1.95	...

⟸

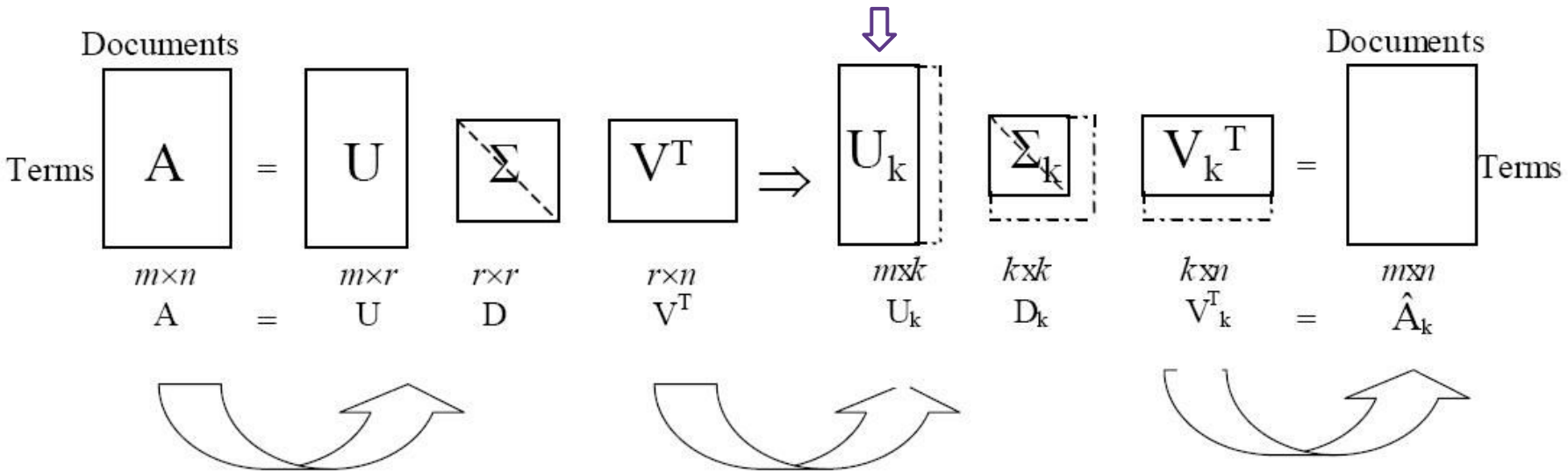
Core idea:

Use **dimensionality reduction** techniques to reduce the corpus dimension to a manageable size (e.g., `n=300`) but leave the vocabulary dimension as it is. The resulting row vectors are **dense** and capture information regarding the occurrence of words in the documents.

Latent Semantic Analysis (LSA)

Formal approach:

- Compute a singular value decomposition $A = UDV$ of the term-document matrix A
- Reduce the number of dimensions to retain only the k most important dimensions
- Use the **row vectors of U_k** as **word embeddings**



https://liqiangguo.wordpress.com/2011/06/09/latent-semantic-analysis

Disadvantages of LSA

Latent Semantic Analysis has a few downsides:

- It works solely based on **global cooccurrence** statistics
  (it only knows about words occurring in the same documents)
- It does not enable us to use **compositional semantics** (more on that later today)

How can we do better?

Distributional Semantics to the Rescue

“You shall know a word by the company it keeps”

J. R. Firth, 1957

Distributional Semantics:

- The meaning of words is indicated by the context in which they occur.
- So let’s use the context of a word to represent it!

⇨ We are using word **co**occurrence information.

Word embeddings derived from this principle are called **distributed representations** (in contrast to **localist representations**).

Cooccurrence: Context and Context Windows

Core idea:

- When a word `w` appears in a text, its context is the set of words that appear nearby
- We use the combined contexts to learn the representation of word `w`

Example (window size = 2) for the word **`banking`**:

```
...decisions [causing a banking crises as] in 2009...
...Europe [needs unified banking regulations to] replace...
...India has just [given its banking system a] shot in the arm...
```

word2vec

The Cloze Task

Cloze test:
The cloze test (also called cloze deletion test) is a fill-in-the-blanks style examination task. For example, we can easily fill in this blank:

```
Natural language processing is all about _________ models.
```

When using word2vec to create word embeddings, we are essentially training a computer to solve this task in two variations:

- Given some context, learn to predict a missing word.
  (this is called **continuous bag of words** or CBOW)
- Given a word, learn to predict the context (this is called **skipgram**)

From a Corpus to Training Data: Skipgram

We generate training data by:

- Iterating over the corpus step-by-step with a fixed window size
- Extracting **pairs** of the input word in the center of the window and **one** target word

Thou shalt not make a machine in the likeness of a human mind

thou	shalt	not	make	a	machine	in	the	…

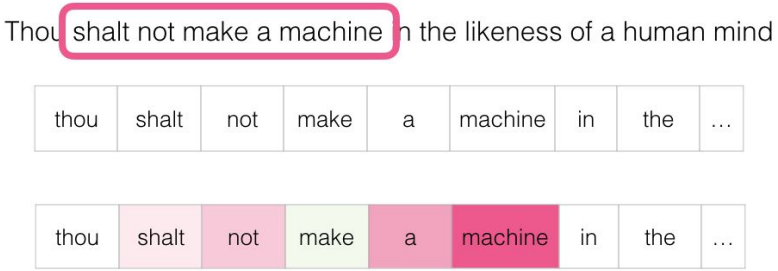
input word	target word
not	thou
not	shalt
not	make
not	a

Examples from https://jalammar.github.io/illustrated-word2vec/

From a Corpus to Training Data: Skipgram

We generate training data by:

- Iterating over the corpus step-by-step with a fixed window size
- Extracting **pairs** of the input word in the center of the window and **one** target word



input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine

Generating training data for CBOW works the same way, except input and target words are reversed

From a Corpus to Training Data: Negative Sampling

We have so far generated only **positive** example (words that actually cooccur)

- Training a classifier on this data would be pointless!
- Always predicting cooccurrence and learning nothing produces perfect accuracy

⇨ We need to add **negative** examples by picking random output words

Pick randomly from vocabulary
(random sampling)

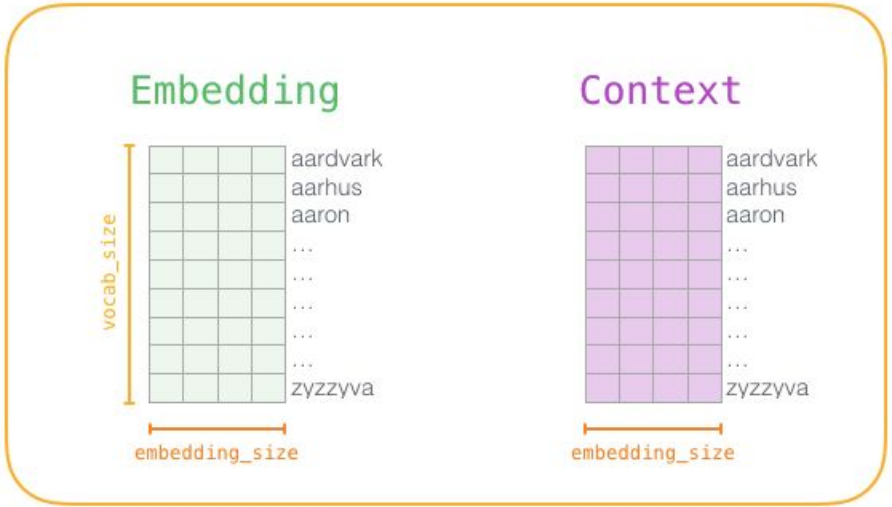
input word	output word	target
not	thou	1
not	aaron	0
not	taco	0
not	shalt	1
not	make	1

Word	Count	Probability
aardvark		
aarhus		
aaron		
taco		
thou		
zyzzyva		

Word2vec: Model Initialization

Word2vec is a **shallow** neural network architecture with just two layers. The layers are an embedding layer (used for learning representations of the input words) and a context layer (used for learning representations of the output words.

The matrices are initialized with random values. The **embedding size** is the selected dimensionality of our latent space (typically 300 dimensions).

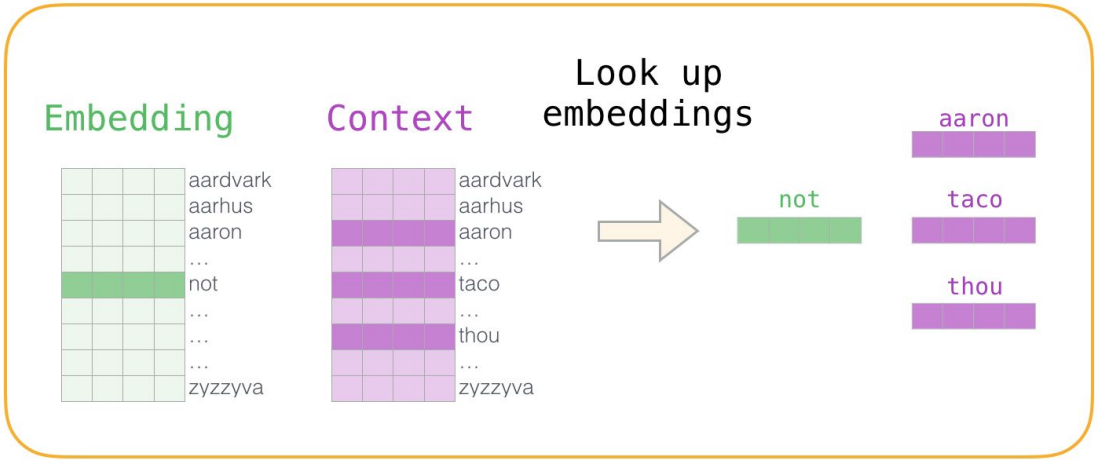


Word2vec: Model Training (1)

When training word2vec, we repeatedly **iterate over the training data**.

- For each input word, we find the corresponding row in the embedding matrix and retrieve the current embedding
- For the positive and negative output words, we find the corresponding rows in the context matrix and retrieve the current embeddings

input word	output word	target
not	thou	1
not	aaron	0
not	taco	0
not	shalt	1
not	mango	0
not	finglonger	0
not	make	1
not	plumbus	0
...	...	...

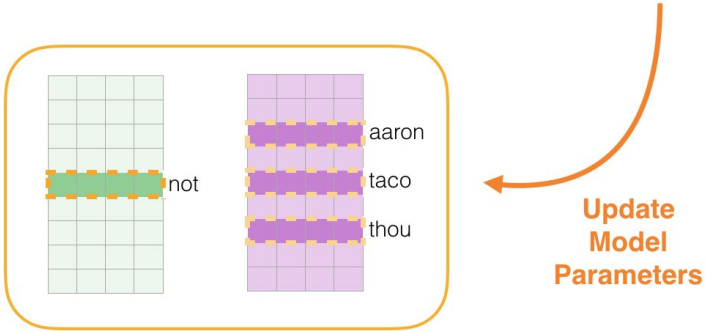


Word2vec: Model Training (2)

For each embedding-context pair, we compute the dot product (≈ cosine similarity)

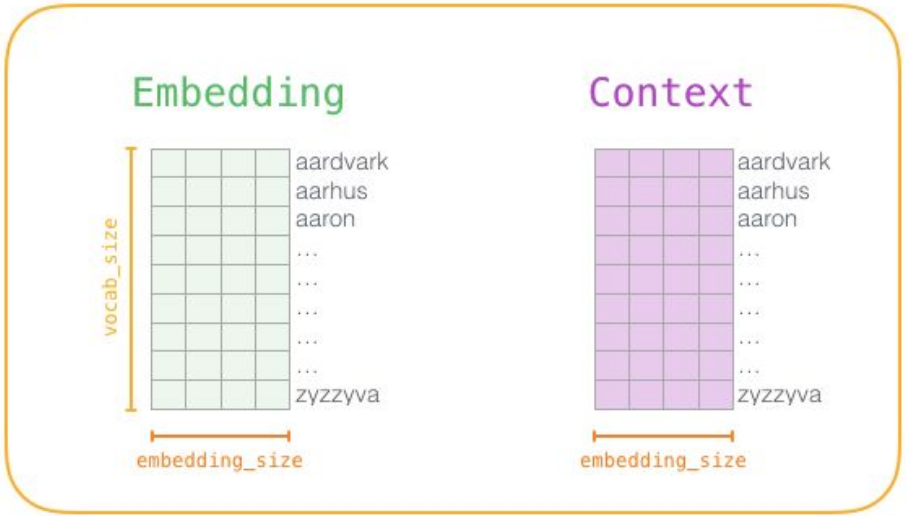
- The similarity is used to predict the likelihood of the target occurring in the context
- We use the error to update the embeddings in both matrices
- This process is repeated for multiple cycles over all data points until it converges

input word	output word	target	input • output	sigmoid()	Error
not	thou	1	0.2	0.55	0.45
not	aaron	0	−1.11	0.25	−0.25
not	taco	0	0.74	0.68	−0.68



Word2vec: Output

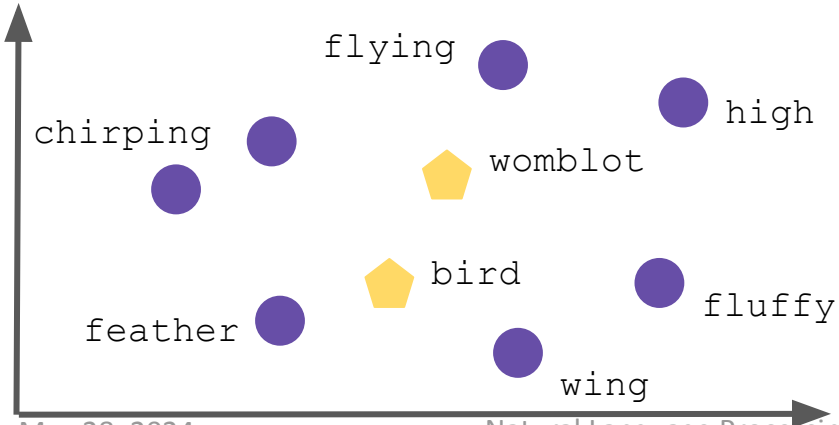
After convergence is reached, we discard the context matrix and use the **embedding matrix** for our word embeddings. Each row contains the embedding of the corresponding word in the vocabulary.



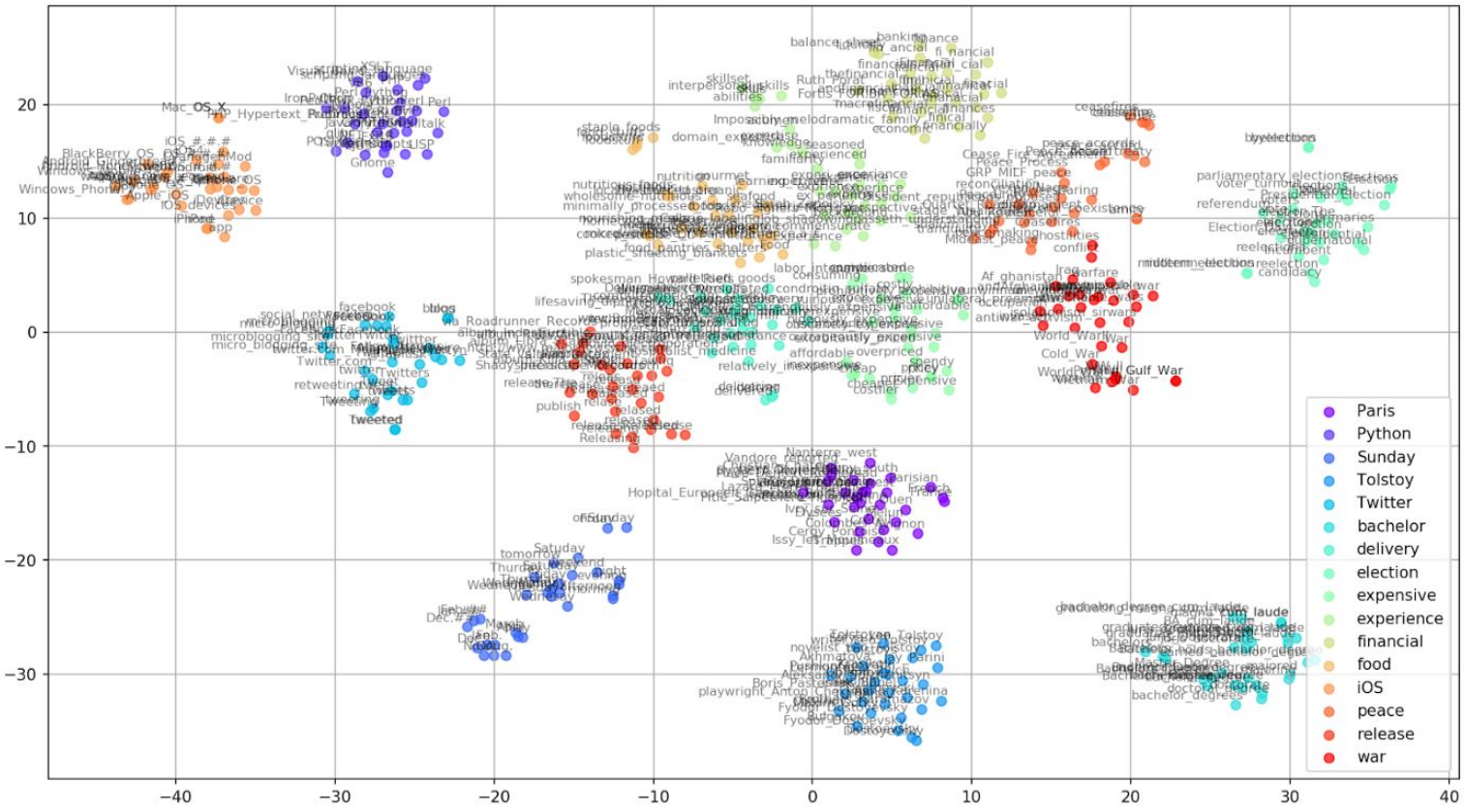
How word2vec Works Intuitively

Why does word2vec produce word embeddings that produce meaningful word similarity?

- During initialization, each word starts at a random location
- During training, each word is pulled closer towards frequently cooccurring words
- During training, each word is pushed away from non-cooccurring words
- Words end up in the proximity of other words that occur in similar contexts
- Think of it as simultaneous high-dimensional tug-of-war



Dense Word Embeddings: Visualized



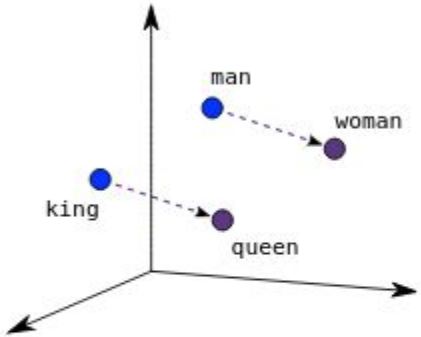
https://towardsdatascience.com/google-news-and-leo-tolstoy-visualizing-word2vec-word-embeddings-with-t-sne-11558d8bd4d

Compositional Semantics

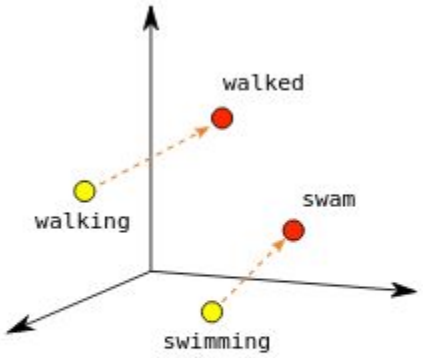
The Word Analogy Task

Man is to **king** as **woman** is to **[?]**

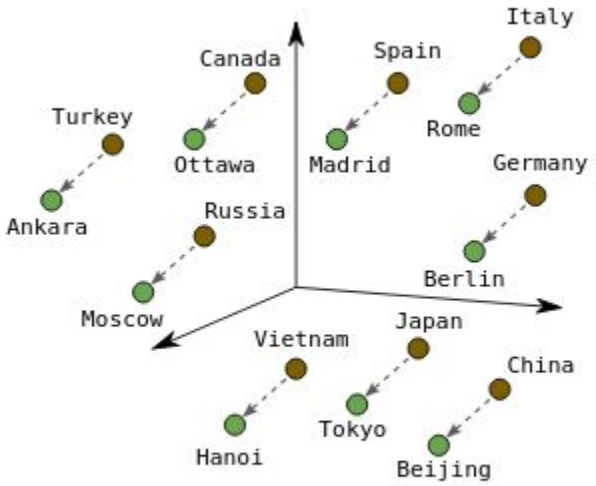
Additive Compositionality



Male-Female



Verb Tense



Country-Capital

Word2vec was created with the idea of supporting arithmetic solution of analogy tasks:

```
king - man + woman = queen
```

Compositional Semantics in a Nutshell

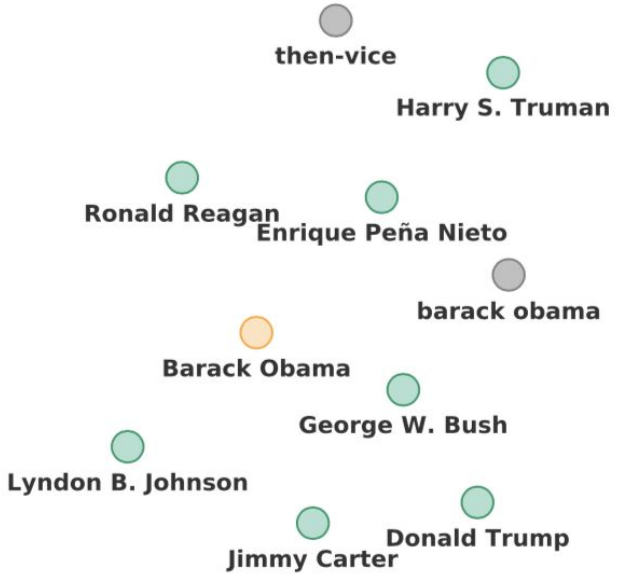
The idea behind compositional semantics:

- Common semantic concepts are implicitly encoded as directions in the latent space
  - Gender
  - Verb tenses
  - Country/capital relations
  - etc.
- The relevance of this is hard to overstate: Being able to **arithmetically** solve **semantic** challenges on data from an **unsupervised** model is a huge step towards machine intelligence!

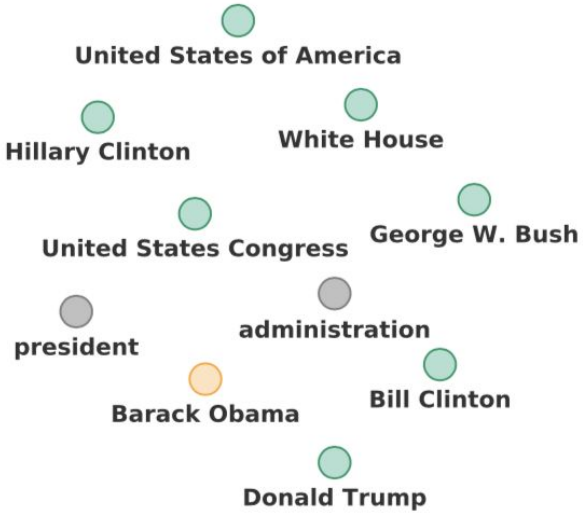
But this ability of language models was (and is) severely overhyped...
⇨ More on that in the exercises.

Word of Caution: Similarity ≠ Relatedness

Word embeddings can be used for many downstream applications. Make sure that the assumptions used when applying them match the assumptions made during training!



word2vec word embeddings
(trained on entity-annotated news documents)



VERSE graph-node embeddings
(trained on an implicit network of the documents)

Further-Reading Material

Latent Semantic Analysis explained

https://moj-analytical-services.github.io/NLP-guidance/LSA.html

The basics of word embeddings

https://towardsdatascience.com/why-do-we-use-embeddings-in-nlp-2f20e1b632d2

The illustrated word2vec

https://jalammar.github.io/illustrated-word2vec/

The math behind GloVe

https://towardsdatascience.com/light-on-math-ml-intuitive-guide-to-understanding-glove-embeddings-b13b4f19c010

Compositional semantics in word2vec revisited

https://blog.esciencecenter.nl/king-man-woman-king-9a7fd2935a85