# Chapter NLP:II

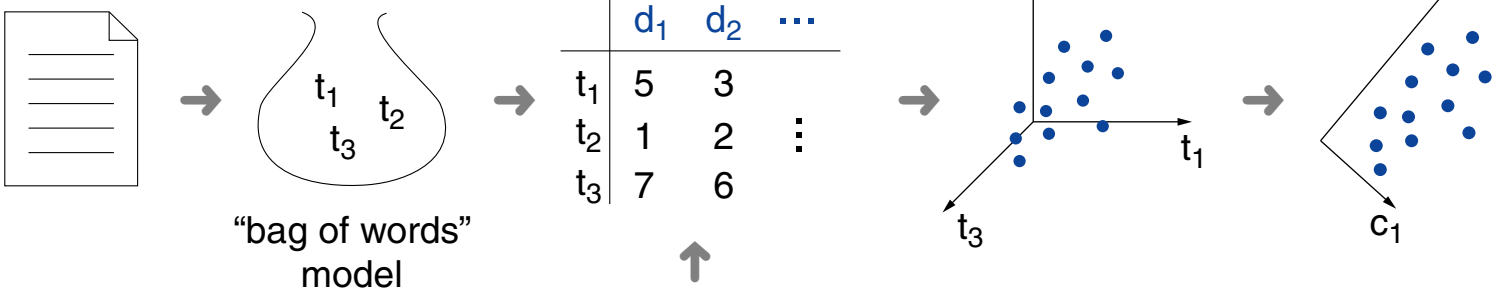
## II. Text Models

- Text Preprocessing
- Text Representation
- Text Similarity
- Text Classification
- Language Modeling
- Sequence Modeling

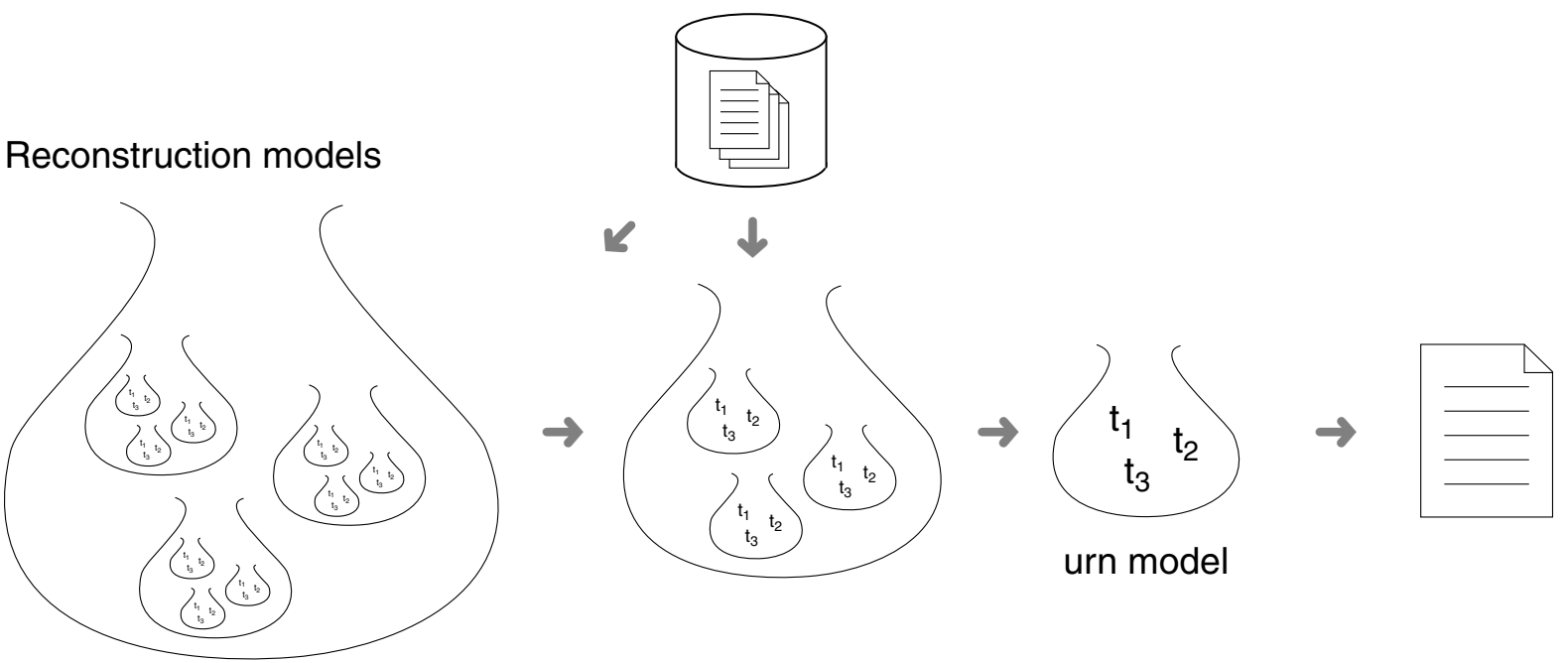
# Text Representation

## How to represent Text? Digitally available texts

Deconstruction models




Reconstruction models



# Text Representation

## Sequence Representation: $n$-grams

Word $n$-grams represent text as overlapping $n$-length subsequences.

- For a sequence of $m \geq n$ tokens, the number of $n$-grams is $(m - n) + 1$.

**Example:** "The quick brown fox jumps over the lazy dog."

- 1-grams: "The", "quick", "brown", "fox", ..., "dog", "."

# Text Representation

## Sequence Representation: $n$-grams

Word $n$-grams represent text as overlapping $n$-length subsequences.

- For a sequence of $m \geq n$ tokens, the number of $n$-grams is $(m - n) + 1$.

**Example:** "The quick brown fox jumps over the lazy dog."

- 1-grams: "The", "quick", "brown", "fox", ..., "dog", "."
- 2-grams: "The quick", "quick brown", ..., "lazy dog", "dog."
- 3-grams: "The quick brown", "quick brown fox", ..., "lazy dog."

# Text Representation

## Sequence Representation: $n$-grams

Word $n$-grams represent text as overlapping $n$-length subsequences.

- For a sequence of $m \geq n$ tokens, the number of $n$-grams is $(m - n) + 1$.

**Example:** "The quick brown fox jumps over the lazy dog."

- 1-grams: "The", "quick", "brown", "fox", ..., "dog", "."
- 2-grams: "The quick", "quick brown", ..., "lazy dog", "dog."
- 3-grams: "The quick brown", "quick brown fox", ..., "lazy dog."

Example $n$-gram counts from *Google n-grams*:

| 1-grams | 2-grams | 3-grams | 4-grams | 5-grams | Tokens | Sentences |
|---|---|---|---|---|---|---|
| 13.6 million | 314.8 million | 977.1 million | 1.3 billion | 1.2 billion | 1.0 trilion | 95.1 billion |

- The most frequent 3-gram on the English web: "all rights reserved".
- $n$-grams with less than 40 occurrences are not included.

# Text Representation

## Document Representation: Bag of Words

Bag of words hypothesis: “The frequencies of words in a document tend to indicate the relevance of the document to a query” [Turney, Pantel 2010]

Bag metaphor

- frequency is important
- order can be neglected

Example word list from presidential speech: `shall(12), amendment(7), states(7), constitution (6), congress (4), united (4)`

# Text Representation

## Document Representation: Vector Space Model [Salton et. al. 1975]

Idea: Encode textual

- documents in **vectors**
- corpora in **matrices**

**Data = event counts for applications like machine learning and statistics**

Dimensionality of vector space

- $|D| \times |V|$ where $|D|$: Number of documents, $|V|$: Vocabulary
- Example matrix dimensions: $3 \times 7$

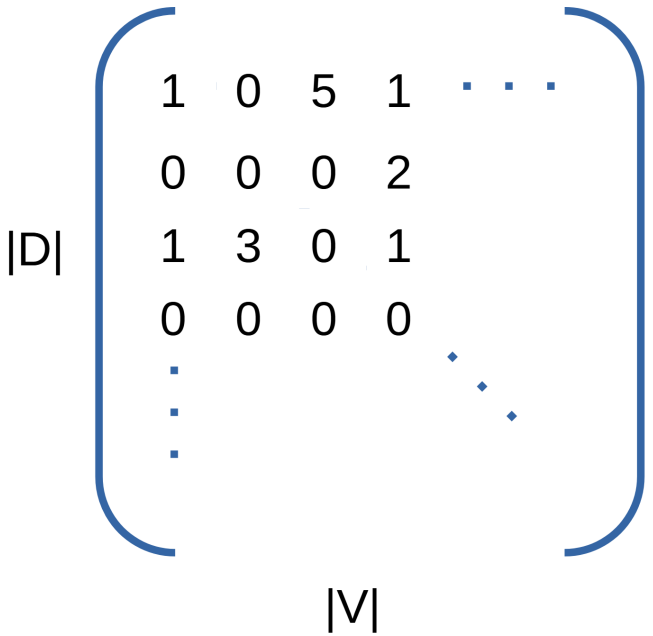
Example corpus:

- $D_1$: Kim is leaving home.
- $D_2$: Kim is at home.
- $D_3$: Karen is leaving.

| | Kim | is | leaving | home | . | at | Karen |
|---|---|---|---|---|---|---|---|
| D_1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| D_2 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| D_3 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |

# Text Representation

## Document Representation: Document-Term-Matrix




- DTM's may get very large
- Events: Frequency counts of word types in each document
- 100% Bag-of-Words
- Very sparse (contains approx. 95% zeros)
- variations:
  - Binary event counts
  - paragraphs as documents
  - sentences as documents
  - additional n-grams (n > 1) as events
  - . . .

Remarks:

- The set of index terms $T = \{t_1, \ldots, t_m\}$ is typically composed of the word stems of the vocabulary of a document collection, excluding stop words.
- The representation $\mathbf{d}$ of a document $d$ is a $|T|$-dimensional vector, where the $i$-th vector component of $\mathbf{d}$ corresponds to a term weight $w_i$ of term $t_i \in T$, indicating its importance for $d$. Various term weighting schemes have been proposed.

# Text Representation

## Term Weighting: *tf* · *idf*

To compute the weight $w$ for a term $t$ from document $d$ under the vector space model, the most commonly employed term weighting scheme $\omega(t)$ is *tf* · *idf*:

- *tf*$(t, d)$ denotes the normalized *term frequency* of term $t$ in document $d$.
  The basic idea is that the importance of term $t$ is proportional to its frequency in document $d$. However, $t$'s importance does not increase linearly: the raw frequency must be normalized.
- *df*$(t, D)$ denotes the *document frequency* of term $t$ in document collection $D$.
  It counts the number of documents that contain $t$ at least once.
- *idf*$(t, D)$ denotes the *inverse document frequency*:

$$\mathit{idf}(t, D) = \log \frac{|D|}{\mathit{df}(t, D)}$$

  The importance of term $t$ in general is inversely proportional to its document frequency.

A term weight $w$ for term $t$ in document $d \in D$ is computed as follows:

$$\omega(t) = \mathit{tf}(t, d) \cdot \mathit{idf}(t, D).$$

# Text Representation

## Term Weighting: *tf* · *idf*

To compute the weight $w$ for a term $t$ from document $d$ under the vector space model, the most commonly employed term weighting scheme $\omega(t)$ is *tf* · *idf*:

- *tf*$(t, d)$ denotes the normalized *term frequency* of term $t$ in document $d$.
  The basic idea is that the importance of term $t$ is proportional to its frequency in document $d$. However, $t$'s importance does not increase linearly: the raw frequency must be normalized.

- *df*$(t, D)$ denotes the *document frequency* of term $t$ in document collection $D$.
  It counts the number of documents that contain $t$ at least once.

- *idf*$(t, D)$ denotes the *inverse document frequency*:

$$\mathit{idf}(t, D) = \log \frac{|D|}{\mathit{df}(t, D)}$$

  The importance of term $t$ in general is inversely proportional to its document frequency.

A term weight $w$ for term $t$ in document $d \in D$ is computed as follows:

$$\omega(t) = \mathit{tf}(t, d) \cdot \mathit{idf}(t, D).$$

# Text Representation

## Term Weighting: *tf* · *idf*

To compute the weight $w$ for a term $t$ from document $d$ under the vector space model, the most commonly employed term weighting scheme $\omega(t)$ is *tf* · *idf*:

- *tf*$(t, d)$ denotes the normalized *term frequency* of term $t$ in document $d$.
  The basic idea is that the importance of term $t$ is proportional to its frequency in document $d$. However, $t$'s importance does not increase linearly: the raw frequency must be normalized.

- *df*$(t, D)$ denotes the *document frequency* of term $t$ in document collection $D$.
  It counts the number of documents that contain $t$ at least once.

- *idf*$(t, D)$ denotes the *inverse document frequency*:

$$\textit{idf}(t, D) = \log \frac{|D|}{\textit{df}(t, D)}$$

  The importance of term $t$ in general is inversely proportional to its document frequency.

A term weight $w$ for term $t$ in document $d \in D$ is computed as follows:

$$\omega(t) = \textit{tf}(t, d) \cdot \textit{idf}(t, D).$$

# Text Representation

## Term Weighting: *tf* · *idf*

To compute the weight $w$ for a term $t$ from document $d$ under the vector space model, the most commonly employed term weighting scheme $\omega(t)$ is *tf* · *idf*:

- *tf*$(t, d)$ denotes the normalized *term frequency* of term $t$ in document $d$.
  The basic idea is that the importance of term $t$ is proportional to its frequency in document $d$. However, $t$'s importance does not increase linearly: the raw frequency must be normalized.
- *df*$(t, D)$ denotes the *document frequency* of term $t$ in document collection $D$.
  It counts the number of documents that contain $t$ at least once.
- *idf*$(t, D)$ denotes the *inverse document frequency*:

$$\textit{idf}(t, D) = \log \frac{|D|}{\textit{df}(t, D)}$$

  The importance of term $t$ in general is inversely proportional to its document frequency.

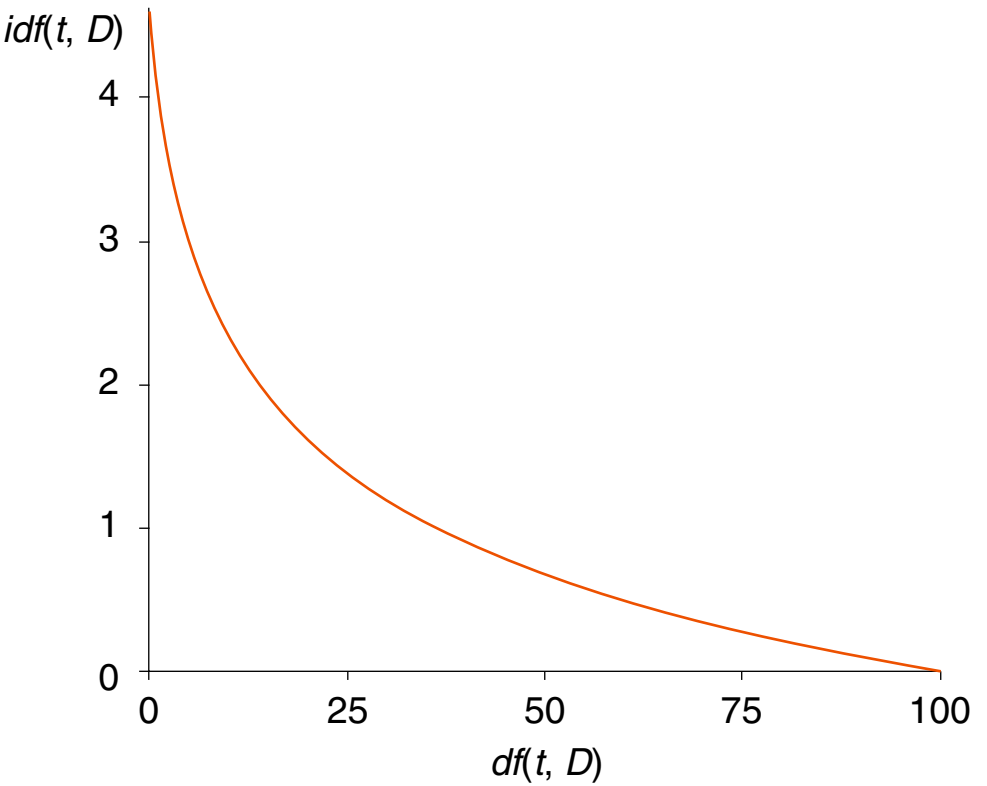
A term weight $w$ for term $t$ in document $d \in D$ is computed as follows:

$$\omega(t) = \textit{tf}(t, d) \cdot \textit{idf}(t, D).$$

# Text Representation

## Term Weighting: $tf \cdot idf$

Plot of the function $idf(t, D) = \log \frac{|D|}{df(t, D)}$ for $|D| = 100$.



Remarks:

- Term frequency weighting was invented by Hans Peter Luhn: “There is also the probability that the more frequently a notion and combination of notions occur, the more importance the author attaches to them as reflecting the essence of his overall idea.” [Luhn 1957]
- The importance of a term $t$ for a document $d$ is not linearly correlated with its frequency. Several normalization factors have been proposed [Wikipedia]:
  - $\mathit{tf}(t,d)/|d|$
  - $1+\log(\mathit{tf}(t,d))$ for $\mathit{tf}(t,d)>0$
  - $k+(1-k)\frac{\mathit{tf}(t,d)}{\max_{t'\in d}(\mathit{tf}(t',d))}$, where $k$ serves as smoothing term; typically $k=0.4$
- Inverse document frequency weighting was invented by Karen Spärck Jones: “it seems we should treat matches on non-frequent terms as more valuable than ones on frequent terms, without disregarding the latter altogether. The natural solution is to correlate a term’s matching value with its collection frequency.” [Spärck Jones 1972]
- Spärck Jones gives little theoretical justification for her intuition. Given the success of *idf* in practice, over the decades, numerous attempts at a theoretical justification have been made. A comprehensive overview has been compiled by [Robertson 2004].

# Text Representation

## Vocabulary Pruning

Pruning = filtering the vocabulary of a collection by minimum / maximum thresholds of token occurrence

Vocabularies even of small collections can get very large (5.000 German newspaper documents $\rightarrow |V| >$ 300.000 types)

- performance issue in machine learning tasks
- meaningful semantics

Very useful preprocessing step to reduce vocabulary size:

- Count occurrence of types in the complete corpus
- keep only those terms which occur above / below a well-defined threshold

# Text Representation

## Vocabulary Pruning

### Term Frequency

- absolute pruning
- sum all term occurrences in all documents filter terms which occur e.g. $count(term) > 1$ AND $count(term) < 1000$

### Document Frequency

- relative pruning
- for each term count number of documents in which it is contained allows for filters like: terms which occur e.g. in
  - more than 99% OR
  - less than 1%

  of all documents

Remarks:

- Linguistic Preprocessing shall reduce / unify data for application specific purpose (See slides about text preprocessing)
- May contain various steps in row:
  - Data cleaning: encoding, spelling correction, duplicate filtering
  - Removing uninformative data: noise, duplicates, stopwords, dictionary lists
  - Unification: punctuation, capitalization, stemming, lemmatization
  - Pruning: remove low/high frequent terms
- Best setup dependent on final analysis task requirements
  - to be derived experimentally
  - or by analyst experience
- **Caution: order of steps influences results!**

# Text Representation

## Distributional Representation of Words

*"You shall know a word by the company it keeps!"* [Firth, 1957]

- If A and B have almost identical environments, they are synonyms.
- Two words are similar if they have similar word contexts (i.e., if they have similar words around them).

"Everybody likes tesgüino." "A bottle of tesgüino is on the table."
"Tesgüino makes you drunk." "We make tesgüino out of corn."

$\rightarrow$ An alcoholic beverage like beer.

# Similarity Measures

## Similarity between Strings: Distributional Hypothesis

### Word-context matrix

- Co-occurrences of words in a corpus within a window of some number of words (say, 20).

| | computer | data | pinch | result | sugar |
|---|---|---|---|---|---|
| apricot | 0 | 0 | 1 | 0 | 1 |
| pineapple | 0 | 0 | 1 | 0 | 1 |
| digital | 2 | 1 | 0 | 1 | 0 |
| information | 1 | 6 | 0 | 4 | 0 |

# Text Representation

## Word Embeddings

### Classical semantic representation with DTM's

- Can be very large and unhandy
- Example → Sentences: 26,142,898, Types: 5,876,655 Term-Term-Matrix of Dimension 5,876,655 × 5,876,655, **3.8TB of data with 32 Bit Integer**
- How to compare term similarity?
- How to find word context efficiently?
- **Idea: Dimension reduction of semantic space!**

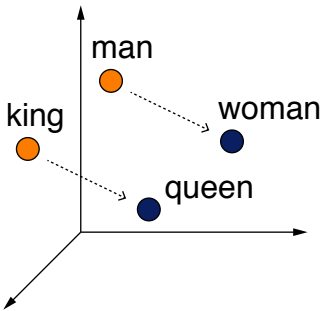
| | dog | plays | children | playing | sun | ... | shining | burning | fire | moon | candle | agree | fact |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| dog | 0 | 1 | 0 | 0 | 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| plays | 1 | 0 | 0 | 0 | 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| children | 0 | 0 | 0 | 1 | 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| playing | 0 | 0 | 1 | 0 | 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| sun | 0 | 0 | 0 | 0 | 0 | | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| ... | | | | | | | | | | | | | |
| shining | 0 | 0 | 0 | 0 | 1 | | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| burning | 0 | 0 | 0 | 0 | 1 | | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| fire | 0 | 0 | 0 | 0 | 0 | | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| moon | 0 | 0 | 0 | 0 | 0 | | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| candle | 0 | 0 | 0 | 0 | 0 | | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| agree | 0 | 0 | 0 | 0 | 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| fact | 0 | 0 | 0 | 0 | 0 | | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

# Text Representation

## Word Embeddings

### Extension of the distributional idea

- Representation of a word by the context it occurs in.
- To do so, words are mapped to an *embedding space* where contextually related words are similar.

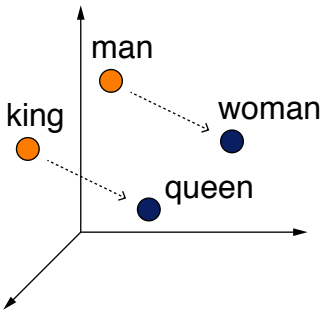


# Text Representation

## Word Embeddings

### Extension of the distributional idea

- Representation of a word by the context it occurs in.
- To do so, words are mapped to an *embedding space* where contextually related words are similar.




### Word embedding (aka word vector)

- A real-valued vector that represents the *distributional semantics* of a particular word in the embedding space.

$$\text{“}king\text{”} \quad \rightarrow \quad v_{king} = (0.13, 0.02, 0.1, 0.4, \ldots, 0.22)$$

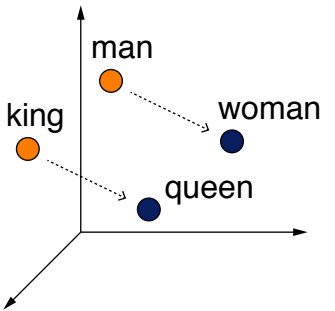
- The longer the vector, the more variance is kept (typical: 100–500).

# Text Representation

## Word Embeddings

### Extension of the distributional idea

- Representation of a word by the context it occurs in.
- To do so, words are mapped to an *embedding space* where contextually related words are similar.




### Word embedding (aka word vector)

- A real-valued vector that represents the *distributional semantics* of a particular word in the embedding space.

$$\text{“}king\text{”} \quad \rightarrow \quad v_{king} = (0.13, 0.02, 0.1, 0.4, \ldots, 0.22)$$

- The longer the vector, the more variance is kept (typical: 100–500).

### Some properties of embedding spaces

- Similar context results in similar embeddings. projector.tensorflow.org
- Analogies are arithmetically represented. turbomaze.github.io/word2vecjson

$$v_{king} - v_{man} + v_{woman} \approx v_{queen} \qquad v_{france} - v_{paris} + v_{berlin} \approx v_{germany}$$

# Text Representation

## Embedding Models

### Word embedding model

- A function that maps each known word to its word embedding.
- Such mappings are created unsupervised based on huge corpora, capturing the likelihood of words occurring in sequence.

  The technical details are beyond the scope of this course.

### Several software libraries and pre-trained models exist

- Libraries. Glove, word2vec, Fasttext, Flair, Bert, ...
- Models. GoogleNews-vectors, ConceptNet Numberbatch, ...

# Text Representation

## Embedding Models

### Word embedding model

- A function that maps each known word to its word embedding.
- Such mappings are created unsupervised based on huge corpora, capturing the likelihood of words occurring in sequence.

  The technical details are beyond the scope of this course.

### Several software libraries and pre-trained models exist

- Libraries. Glove, word2vec, Fasttext, Flair, Bert, ...
- Models. GoogleNews-vectors, ConceptNet Numberbatch, ...

### From word embeddings to text embeddings

- Simple. Average the embeddings of each word in a text.
- More sophisticated. Learn embeddings for sentences or similar.
- In general, the longer the text, the harder it is to capture its semantics in an embedding.

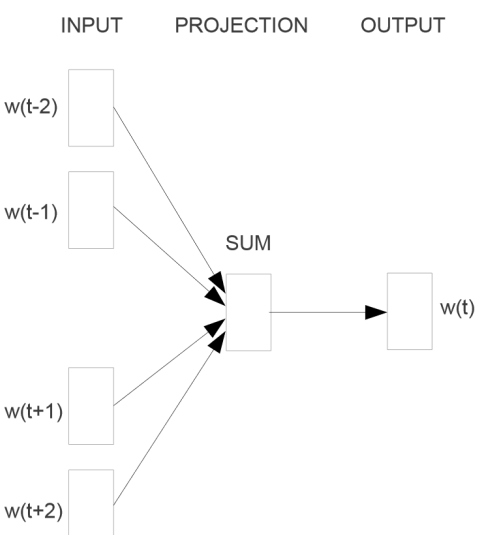
# Text Representation

## Word2Vec Example

### Main Idea [Mikolov et al. 2013]

- Shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words.
- The projection layer transforms the input to the output: **Similar words or contexts need a similar weight vector** in order to be transformed to the same target.

- **Continuous bag-of-words architecture (CBOW)**: the model predicts the current word from a window of surrounding context words.
- CBOW is faster while skip-gram is slower but does a better job for infrequent words




CBOW

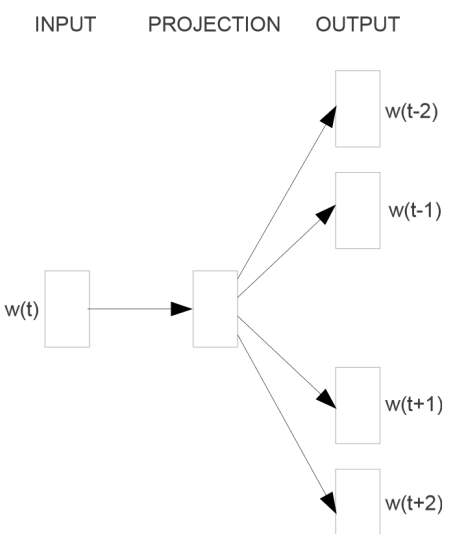
# Text Representation

## Word2Vec Example

### Main Idea [Mikolov et al. 2013]

- Shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words.
- The projection layer transforms the input to the output: **Similar words or contexts need a similar weight vector** in order to be transformed to the same target.

- **Continuous skip-gram architecture (SKIPGRAM)**: the model uses the current word to predict the surrounding window of context words.




**Skip-gram**

# Text Representation

## Word Vectors as Feature

### Main Idea

- Replacement of vectors among vocabulary by semantic embedding vectors
- Better capturing of semantic properties, composition and ambiguities
- Modern language model embeddings (Bert, Elmo, Flair, FastText) use character sequence embeddings → no (O)ut (O)f (V)ocabulary Problem in prediction, robust to typos

Example: Convolutional Neural Networks for Sentence Classification [Yoon Kim 2014]