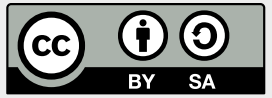
Representing Text (TF-IDF, Word2vec)

Jindřich Libovický (reusing materials by Milan Straka)

November 7, 2023

Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics



Today's Lecture Objectives

After this lecture you should be able to

- Use TF-IDF for representing documents and explain its information-theoretical interpretation.
- Explain training of Word2Vec as a special case of logistic regression.
- Use pre-trained word embeddings for simple NLP tasks.

Metrics for Exemplary Tasks

- **Part-of-speech tagging**: assign a part-of-speech to every word in the input text.
  - **accuracy** on such a task is the same as micro-averaged precision, recall, and F_1-score, because exactly one class is predicted for every word (i.e., TP+FP = TP+FN).
- **Named entity recognition**: recognize personal names, organizations, and locations in the input text.
  - **accuracy** is artificially high, because many words are not a named entity;
  - **micro-averaged** F_1 considers all named entities, with classes used only to decide if a prediction is correct; *"how good are we at recognizing all present named entities"*;
  - **macro-averaged** F_1 *"how good are we at recognizing all named entities* ***types****"*.

Consider **multi-label classification**, where you can generate any number of classes for an input example (while in the multiclass classification you generate always exactly one).

- For example **text classification**: choose domains (sports/politics/...) for input documents.
- Can be solved analogously to softmax classification, only using sigmoid activation.
- Accuracy is very strict (all predicted classes must be exactly the same).
- Commonly evaluated using micro-averaged or macro-averaged F_1-score.

Document Representation

We already know how to represent images and categorical variables (classes, letters, words, ...).

Now consider the problem of representing a whole *document*.

An elementary approach is to represent a document as a **bag of words** – we create a feature space with a dimension for every unique word (or for character sequences), called a **term**.

However, there are many ways in which the values of the terms can be set.

Term Frequency – Inverse Document Frequency

Commonly used ways of setting the term values:

- **binary indicators**: 1/0 depending on whether a term is present in a document or not;
- **term frequency (TF)**: relative frequency of a term in a document;

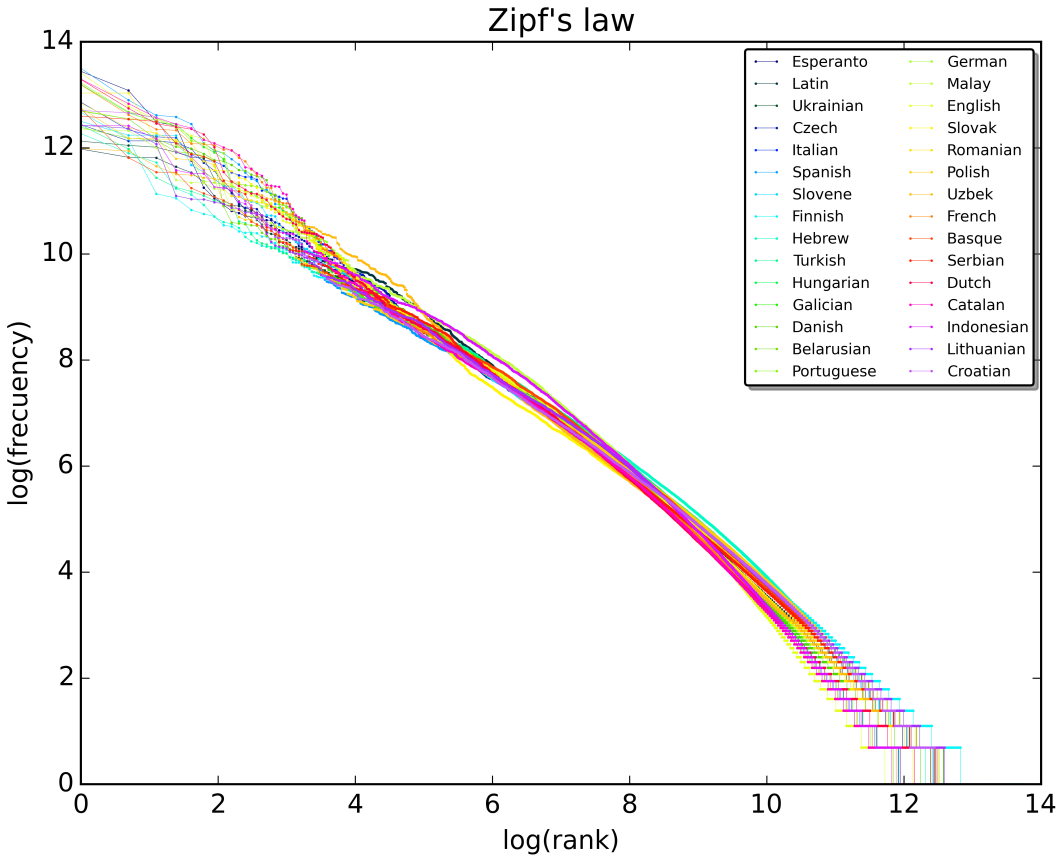
$$TF(t;d) = \frac{\text{number of occurrences of } t \text{ in the document } d}{\text{number of terms in the document } d}$$

- **inverse document frequency (IDF)**: we could also represent a term using self-information of a probability of a random document containing it (therefore, terms with lower document probability have higher weights);

$$IDF(t) = \log \frac{\text{number of documents}}{\text{number of documents containing } t \ (\text{optionally} + 1)} = I\big(P(d \ni t)\big)$$

- **TF-IDF**: empirically, product $TF \cdot IDF$ is a feature reflecting quite well how important a term is to a document in a corpus (used by 83% text-based recommender systems in 2015).

Original Motivation



Jones (1972) provided only intuitive justification.

Zipf's law: empirically, word frequencies follow

$$\text{word frequency} \propto \frac{1}{\text{word rank}}$$

I.e., $\frac{|\mathcal{D}|}{|\{d \in \mathcal{D} : t \in d\}|}$ would be extremely low for frequent words, and high for infrequent ones. Logarithm normalizes that.

https://en.wikipedia.org/wiki/Tf%E2%80%93idf

Mutual Information

Consider two random variables $\mathbf{x}$ and $\mathbf{y}$ with distributions $\mathbf{x} \sim X$ and $\mathbf{y} \sim Y$.

The conditional entropy $H(Y|X)$ can be naturally considered an expectation of a self-information of $Y|X$, so in the discrete case,

$$H(Y|X) = \mathbb{E}_{\mathbf{x},\mathbf{y}}\big[I(y|x)\big] = -\sum_{x,y} P(x,y) \log P(y|x).$$

In order to assess the amount of information *shared* between the two random variables, we might consider the difference

$$H(Y) - H(Y|X) = \mathbb{E}_{\mathbf{x},\mathbf{y}}\big[-\log P(y)\big] - \mathbb{E}_{\mathbf{x},\mathbf{y}}\big[-\log P(y|x)\big] = \mathbb{E}_{\mathbf{x},\mathbf{y}}\left[\log \frac{P(x,y)}{P(x)P(y)}\right].$$

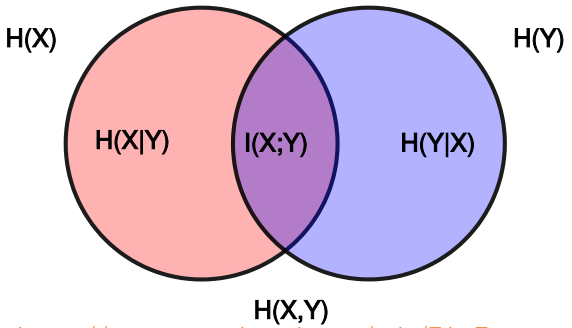
We can interpret this value as

How many bits of information will we learn about Y when we find out X?

Mutual Information

Let us denote this quantity as the **mutual information** $I(X;Y)$:

$$I(X;Y) = \mathbb{E}_{\mathrm{x,y}}\left[\log \frac{P(x,y)}{P(x)P(y)}\right].$$



https://commons.wikimedia.org/wiki/File:Entropy-mutual-information-relative-entropy-relation-diagram.svg

- The mutual information is symmetrical, so

$$I(X;Y) = I(Y;X) = H(Y) - H(Y|X) = H(X) - H(X|Y).$$

- It is easy to verify that

$$I(X;Y) = D_{\mathrm{KL}}\big(P(X,Y) \| P(X)P(Y)\big).$$

Therefore,
  - $I(X;Y) \geq 0$,
  - $I(X;Y) = 0$ iff $P(X,Y) = P(X)P(Y)$ iff the random variables are independent.

TF-IDF as Mutual Information

Let $\mathcal{D}$ be a collection of documents and $\mathcal{T}$ a collection of terms.

We assume that whenever we need to draw a document, we do it uniformly randomly. Then,

- $P(d) = 1/|\mathcal{D}|$ and $I(d) = H(\mathcal{D}) = \log |\mathcal{D}|$,
- $P(d|t \in d) = 1/|\{d \in \mathcal{D} : t \in d\}|$,
- $I(d|t \in d) = H(\mathcal{D}|t) = \log |\{d \in \mathcal{D} : t \in d\}|$, assuming $0 \cdot \log 0 = 0$ in H as usual,
- $I(d) - I(d|t \in d) = H(\mathcal{D}) - H(\mathcal{D}|t) = \log \frac{|\mathcal{D}|}{|\{d \in \mathcal{D} : t \in d\}|} = IDF(t)$.

Finally, we can compute the mutual information $I(\mathcal{D}; \mathcal{T})$ as

$$I(\mathcal{D}; \mathcal{T}) = \sum_{d,\, t \in d} P(d) \cdot P(t|d) \cdot \big(I(d) - I(d|t)\big) = \frac{1}{|\mathcal{D}|} \sum_{d,\, t \in d} TF(t; d) \cdot IDF(t).$$

Therefore, summing all TF-IDF terms recovers the mutual information between $\mathcal{D}$ and $\mathcal{T}$, and we can say that each TF-IDF carries a "bit of information" attached to a document-term pair.

Representation Learning

- We interpreted MLP as automatic feature extraction for a generalized linear model.
- Representation learning: learning using a proxy task that leads to reusable features.

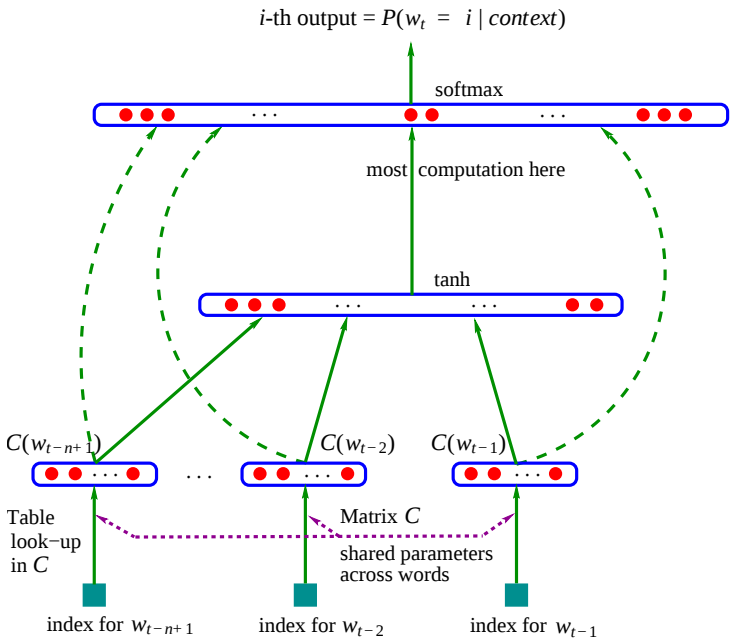
Famous examples: pre-training image representations using **object classification**, pre-training using **language models**.

Word Embeddings

- Assuming a limited vocabulary: an input word can be represented as a one-hot vector
- Multiplying a matrix with a one-hot vector = picking a vector from the weight matrix
- This matrix column = **word embedding**

The oldest neural language model

- Language model predicts a probability of the next token – it used to be a component of machine translation and speech recognition.
- In 2003: Bengio et al. used MLP for language modeling



Bengio, Yoshua, Réjean Ducharme, and Pascal Vincent. "A neural probabilistic language model." Advances in neural information processing systems 13 (2000). Figure 1.

- The embedding matrix is reused for all input words

Properties of Word Embeddings

- Collobert et al. (2011) first reused embeddings from language modelling as features for other NLP tasks
- Geometric properties: Neighbors in the space are semantically similar words

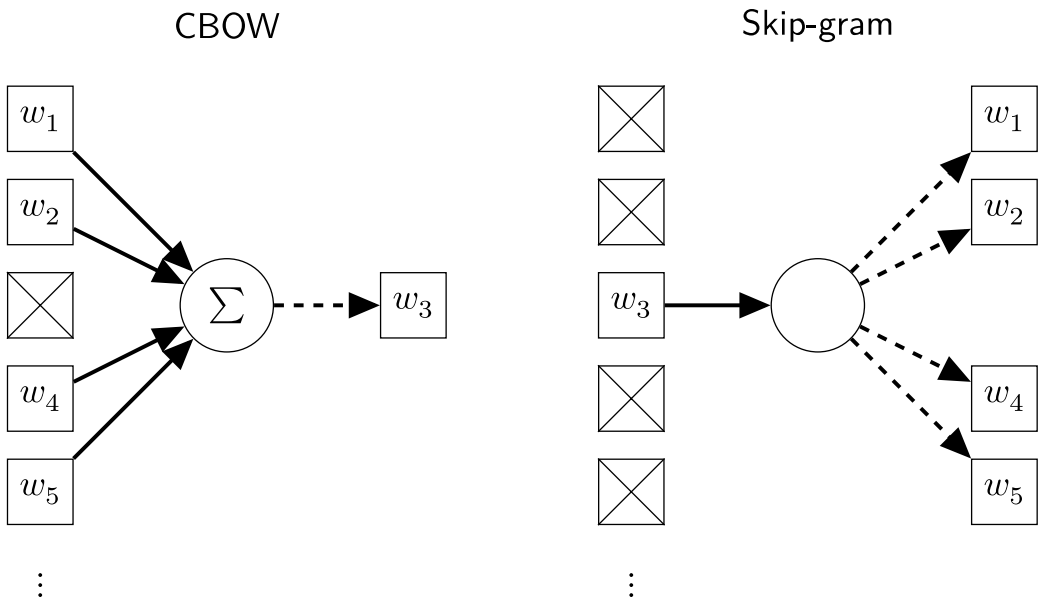
FRANCE	JESUS	XBOX	REDDISH	SCRATCHED	MEGABITS
454	1973	6909	11724	29869	87025
AUSTRIA	GOD	AMIGA	GREENISH	NAILED	OCTETS
BELGIUM	SATI	PLAYSTATION	BLUISH	SMASHED	MB/S
GERMANY	CHRIST	MSX	PINKISH	PUNCHED	BIT/S
ITALY	SATAN	IPOD	PURPLISH	POPPED	BAUD
GREECE	KALI	SEGA	BROWNISH	CRIMPED	CARATS
SWEDEN	INDRA	PSNUMBER	GREYISH	SCRAPED	KBIT/S
NORWAY	VISHNU	HD	GRAYISH	SCREWED	MEGAHERTZ
EUROPE	ANANDA	DREAMCAST	WHITISH	SECTIONED	MEGAPIXELS
HUNGARY	PARVATI	GEFORCE	SILVERY	SLASHED	GBIT/S
SWITZERLAND	GRACE	CAPCOM	YELLOWISH	RIPPED	AMPERES

Collobert, Ronan, et al. "Natural language processing (almost) from scratch." Journal of machine learning research 12.(2011): 2493-2537. Table 6.

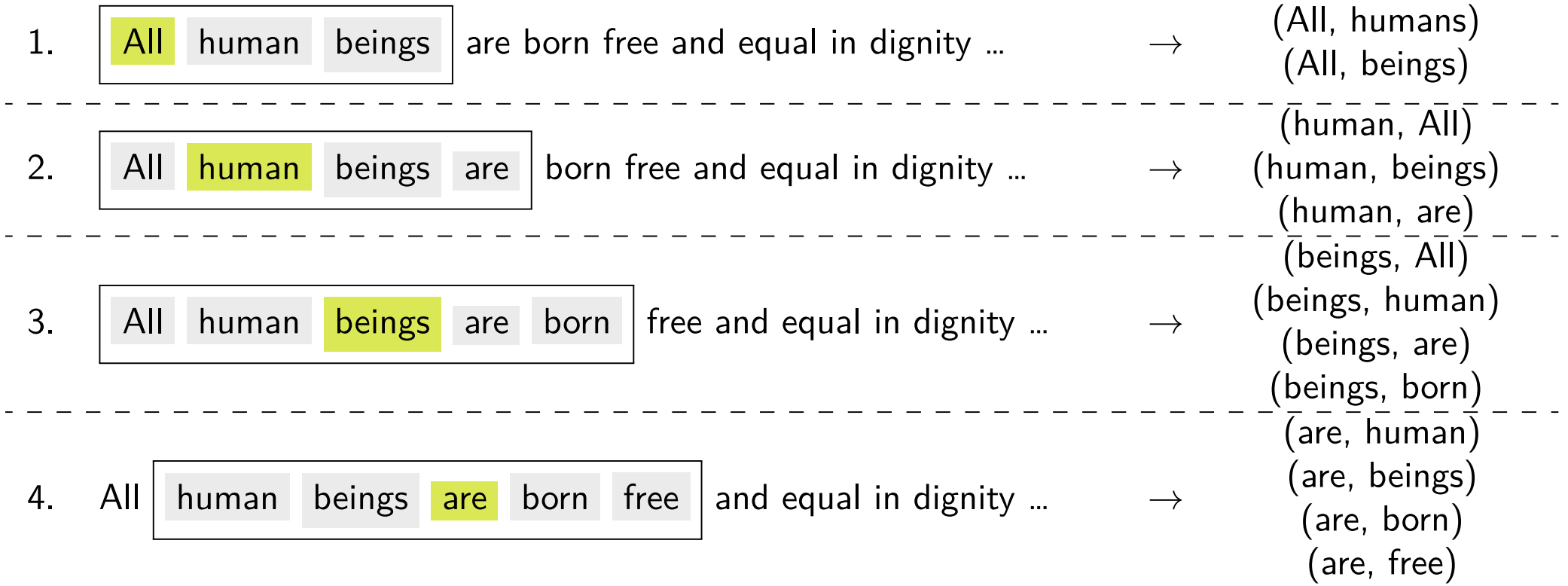
- In a downstream task, we learn something also for words that **were not in training data** but might be similar to some that were.

How to get vectors for as many words as possible without training a huge model?

1. Simplify the context – treat it as a bag of words
2. Simplify the architecture – turn it to a linear model



Word2Vec: Skip-gram Sampling



SkipGram: Idea of the Computation

- For each word from vocabulary V, we want to learn a d-dimension embedding vector $\boldsymbol{e}$
- For an embedding vector $\boldsymbol{e} \in \mathbb{R}^d$, we predict probability distribution of words that may appear in the context: $\mathrm{softmax}\left(\boldsymbol{e}^T \boldsymbol{W}\right)$
- $\boldsymbol{W} \in d \times |\mathbb{V}|$ is a parameter matrix shared for all embeddings $\boldsymbol{e}$
- Vocabularies of $10^5 - 10^6$ word forms $\Rightarrow$ computing the $\mathrm{softmax}$ would be expensive.
- Turn it into **multi-label classification** and use negative sampling for loss function – we say about each word independently if it appears in the word context.

$\sigma\left(EW\right)_{I,J}$ should estimate a table of $|V| \times |V|$ with probabilities that j-th word is in the neighbor window of i-th word

SkipGram: Negative Sampling

- Split the output matrix $\boldsymbol{W}$ into output embeddings $\boldsymbol{v}$
- For input word w with embeddings $\boldsymbol{e}_w$ and context word c with output embedding $\boldsymbol{v}_c$, we do logistic regression and the following to the loss function:

$$-\log \sigma(\boldsymbol{e}_w^T \boldsymbol{v}_c)$$

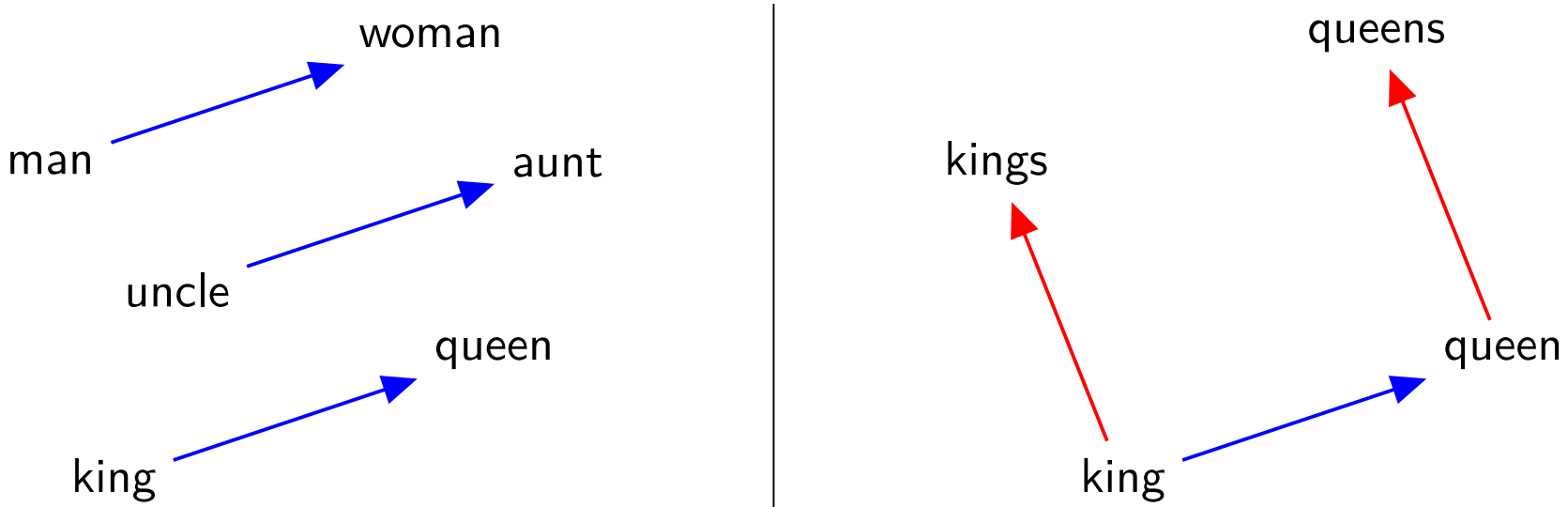
- Then we sample negative K word sample c_i that are **not** in the context window and add them negatively to the loss function:

$$-\sum_{i=1}^{K} \log \sigma(-\boldsymbol{e}_w^T \boldsymbol{v}_{c_i})$$

- The distribution we sample from is heuristically modified categorical distribution based on word frequencies

Interesting Properties of Word2Vec

It seems that vector arithmetics captures nicely lexical semantics.



Mikolov, Tomáš, Wen-tau Yih, and Geoffrey Zweig. "Linguistic regularities in continuous space word representations." Proceedings of NAACL-HLT. 2013. Adapted from Figure 2.

Using Word Embeddings in Our Models

- Single label per text:
  - Problem: text of variable length → most frequent solution
  - Just compute the average over the sequence (and think what should go into the average)
- Sequence labeling = assign a label per token
  - Many NLP tasks can be formulated as sequence labeling (POS tagging, named entity recognition, extractive summarization or QA)
  - Use a sliding window of embedding and classify the middle one

More Recent Word Embeddings

- FastText: Word embedding is a sum of substring embeddings (can be trained by backpropagation)
- Distillation of word embeddings from larger neural language models
- SoTA: finetuning with contextual embeddings (*BERT*), large language models

Today's Lecture Objectives

After this lecture you should be able to

- Use TF-IDF for representing documents and explain its information-theoretical interpretation.
- Explain training of Word2Vec as a special case of logistic regression.
- Use pre-trained word embeddings for simple NLP tasks.