Intelligent Systems,
MIPT

Natural Language Processing

L2 / 17.02.26

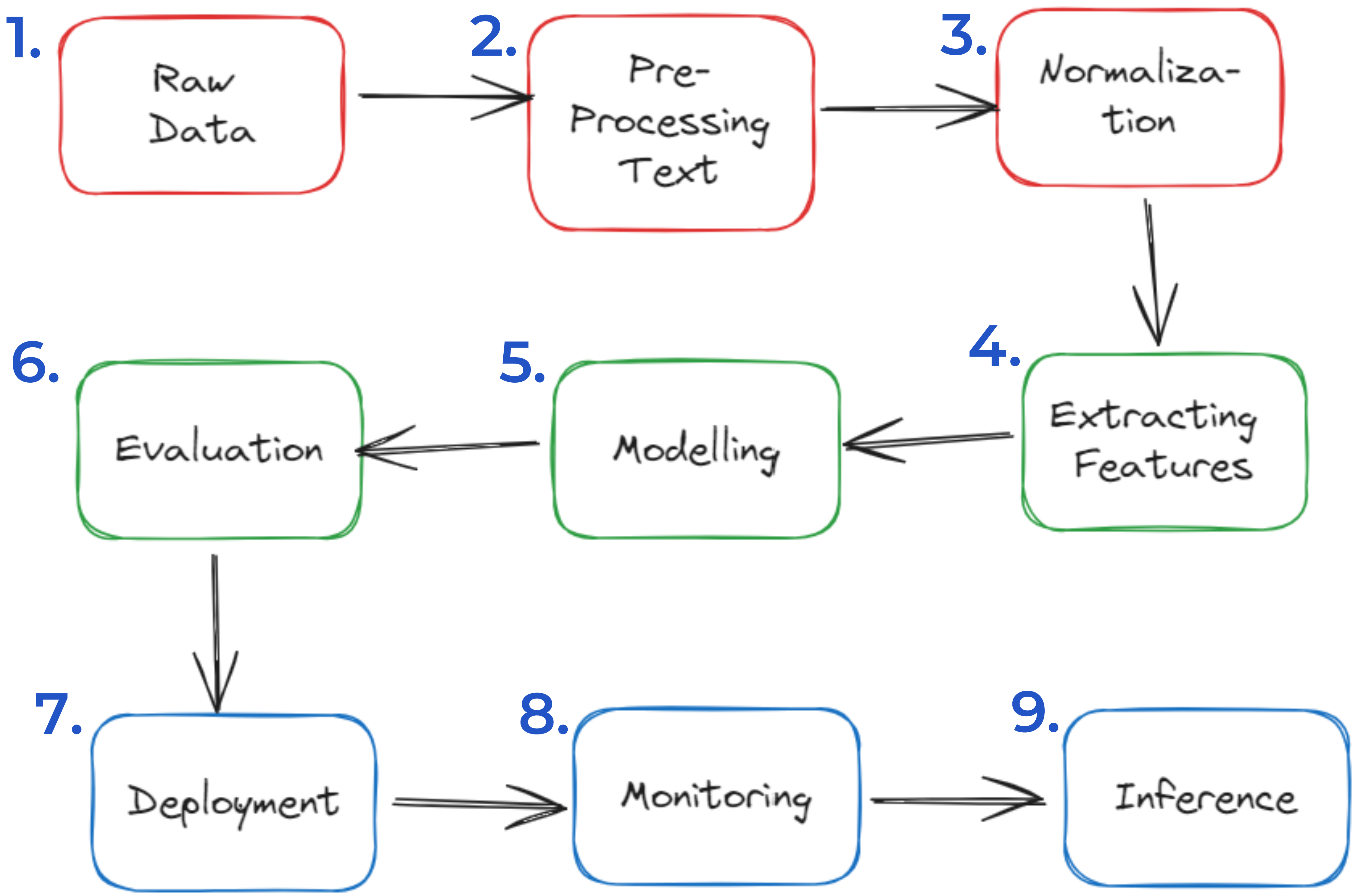
- Course page
- GitHub

Content

Lecture 2: Feature Extraction and Word Representations

- Bag of Words, TF-IDF
- One-hot vectors, Count-Based Methods
- Word2Vec, Glove, FastText
- Evaluation and Current State

NLP pipeline



Bag of Words

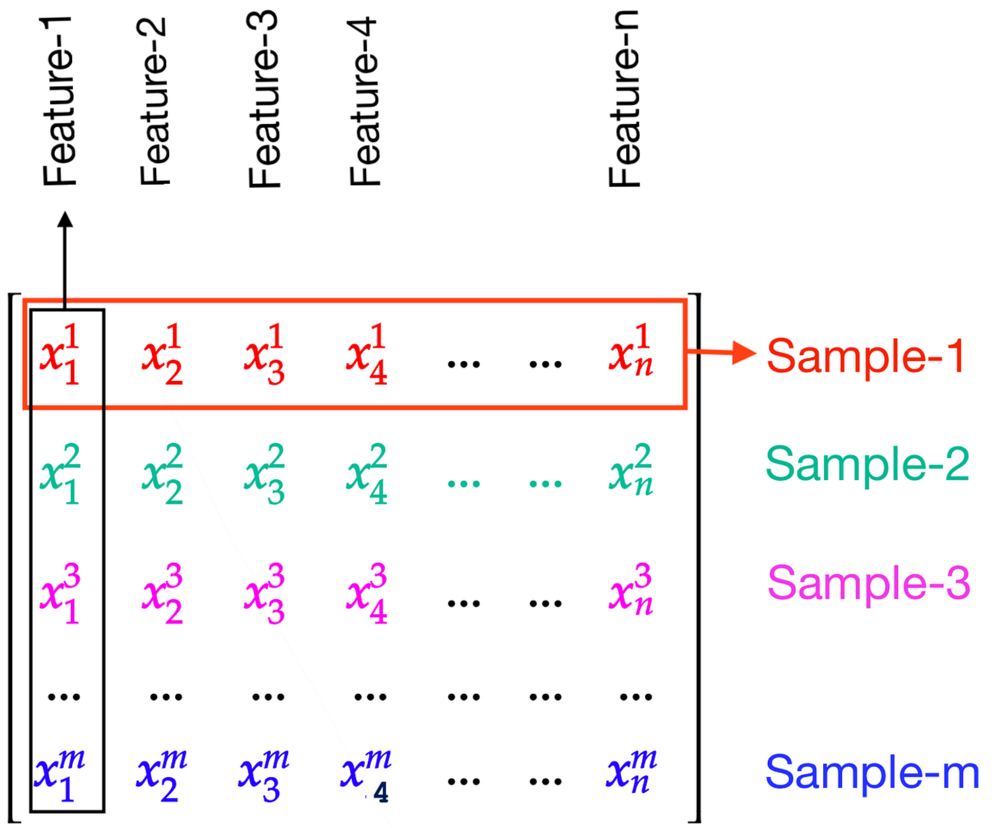
- Feature matrix from classical ML

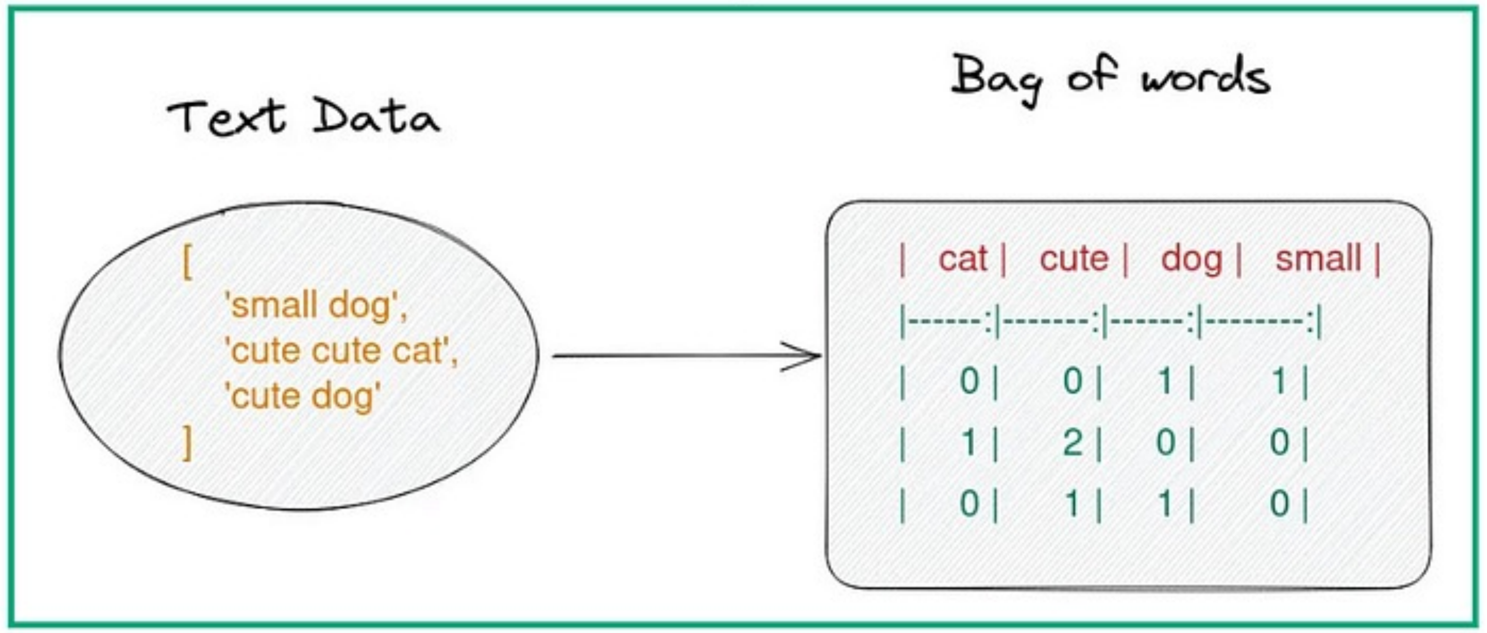
$$
\begin{array}{c}
\begin{array}{ccccccc}
\text{Feature-1} & \text{Feature-2} & \text{Feature-3} & \text{Feature-4} & & & \text{Feature-n}
\end{array} \\
\begin{array}{cc}
\begin{bmatrix}
x_1^1 & x_2^1 & x_3^1 & x_4^1 & \dots & \dots & x_n^1 \\
x_1^2 & x_2^2 & x_3^2 & x_4^2 & \dots & \dots & x_n^2 \\
x_1^3 & x_2^3 & x_3^3 & x_4^3 & \dots & \dots & x_n^3 \\
\dots & \dots & \dots & \dots & \dots & \dots & \dots \\
x_1^m & x_2^m & x_3^m & x_4^m & \dots & \dots & x_n^m
\end{bmatrix}
&
\begin{array}{l}
\text{Sample-1} \\
\text{Sample-2} \\
\text{Sample-3} \\
\\
\text{Sample-m}
\end{array}
\end{array}
\end{array}
$$

Bag of Words

- Feature matrix from classical ML

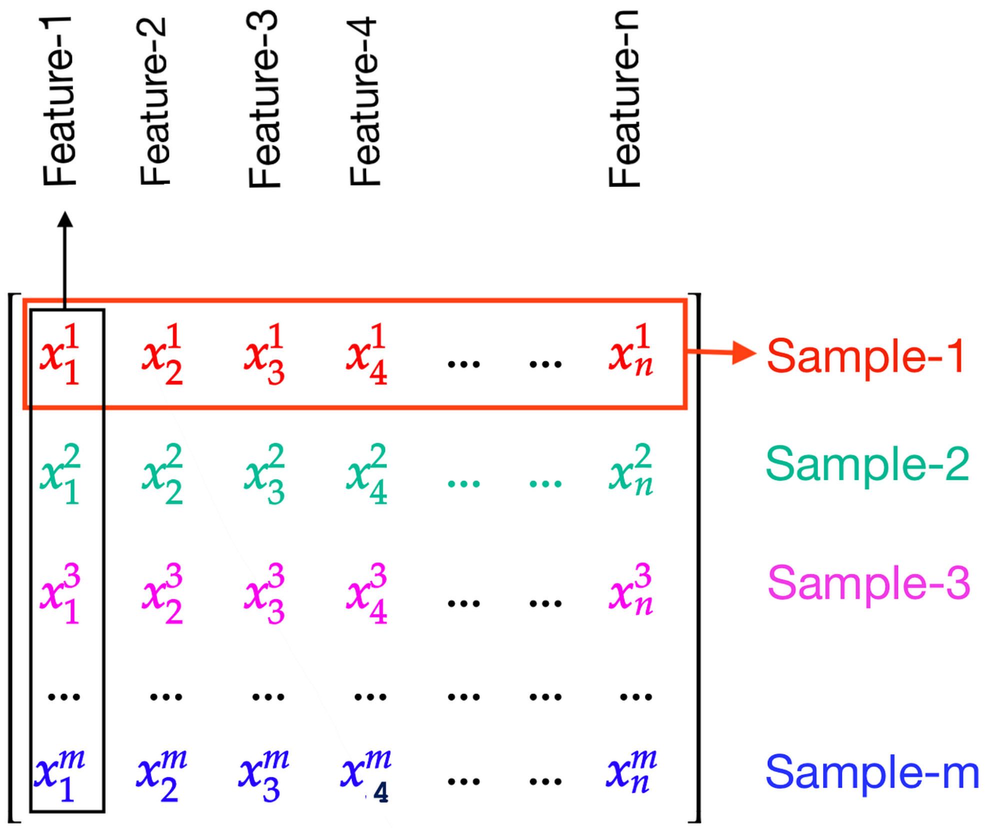
- A text, such as a paragraph or document, is presented as a set of words, and the frequency with which each word appears in the text is counted

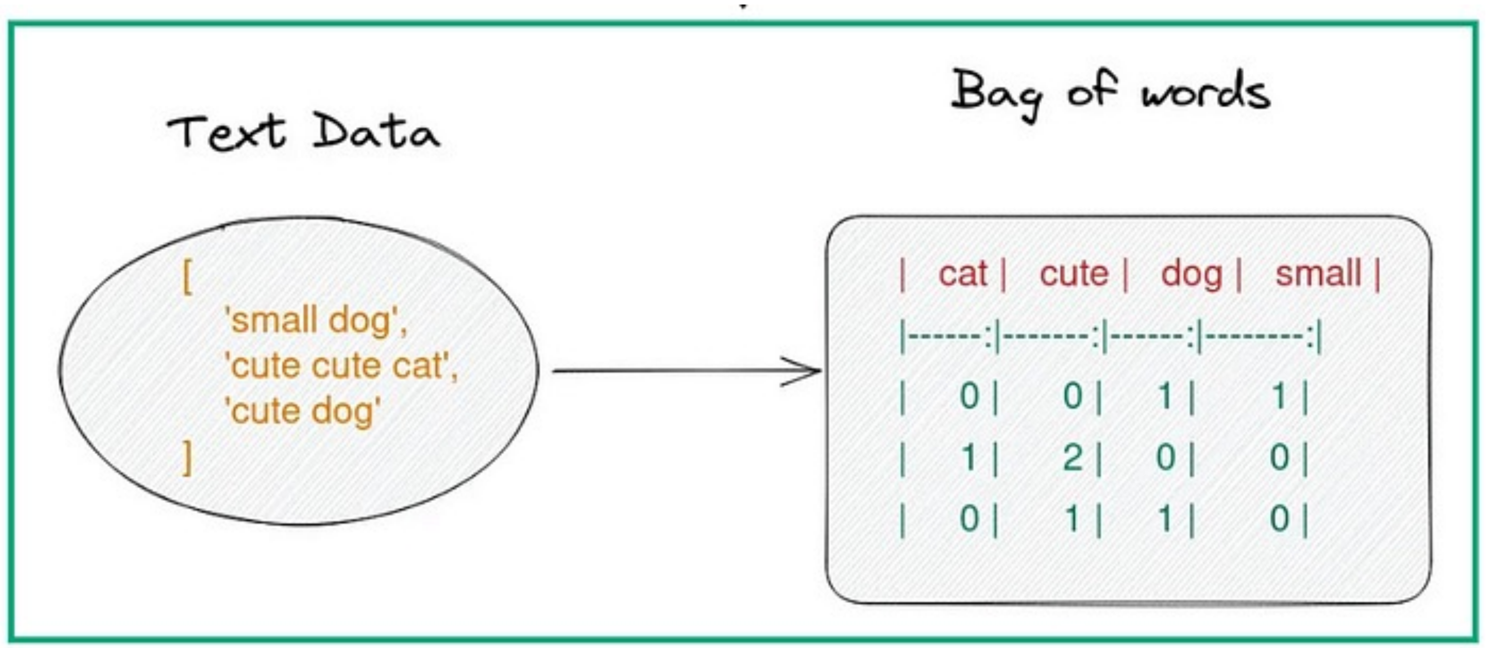




Bag of Words

- Feature matrix from classical ML
- A text, such as a paragraph or document, is presented as a set of words, and the frequency with which each word appears in the text is counted





Problems:

➢ Loss of word order → Identical vectors
"The dog bit the man", "The man bit the dog"

➢ Ignoring semantics → No connection between synonyms
"I drive a car", "I drive an automobile"

➢ The curse of dimensionality

TF-IDF

- Idea: highlight words that appear frequently in this text, but rarely in other texts

TF-IDF

- **Idea:** highlight words that appear frequently in this text, but rarely in other texts

- Term Frequency - Inverse Document Frequency (TF-IDF):

$$\mathrm{tf}(t,d) = \frac{n_t}{\sum_k n_k}$$

TF-IDF

- Idea: highlight words that appear frequently in this text, but rarely in other texts

- Term Frequency - Inverse Document Frequency (TF-IDF):

$$\mathrm{tf}(t,d) = \frac{n_t}{\sum_k n_k}$$

$$\mathrm{idf}(t,D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

TF-IDF

- **Idea: highlight words that appear frequently in this text, but rarely in other texts**

- **Term Frequency - Inverse Document Frequency (TF-IDF):**

$$\mathrm{tf}(t,d) = \frac{n_t}{\sum_k n_k}$$

$$\mathrm{idf}(t,D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

$$\text{tf-idf}(t,d,D) = \mathrm{tf}(t,d) \times \mathrm{idf}(t,D)$$

where - n_t is the number of occurrences of word t in the document d, $N = |D|$

- **Usage** → sklearn.feature_extraction.text import TfidfVectorizer

Example: TF-IDF

Sentence A: The car is driven on the road.
Sentence B: The truck is driven on the highway.

Word	TF		IDF	TF * IDF	
	A	B		A	B
The	1/7	1/7	log(2/2)=0		
Car	1/7	0	log(2/1)=0.3		
Truck	0	1/7	log(2/1)=0.3		
Is	1/7	1/7	log(2/2)=0		
Driven	1/7	1/7	log(2/2)=0		
On	1/7	1/7	log(2/2)=0		
The	1/7	1/7	log(2/2)=0		
Road	1/7	0	log(2/1)=0.3		
Highway	0	1/7	log(2/1)=0.3		

Example: TF-IDF

Sentence A: The car is driven on the road.
Sentence B: The truck is driven on the highway.

Word	TF		IDF	TF * IDF	
	A	B		A	B
The	1/7	1/7	log(2/2)=0		
Car	1/7	0	log(2/1)=0.3		
Truck	0	1/7	log(2/1)=0.3		
Is	1/7	1/7	log(2/2)=0		
Driven	1/7	1/7	log(2/2)=0		
On	1/7	1/7	log(2/2)=0		
The	1/7	1/7	log(2/2)=0		
Road	1/7	0	log(2/1)=0.3		
Highway	0	1/7	log(2/1)=0.3		

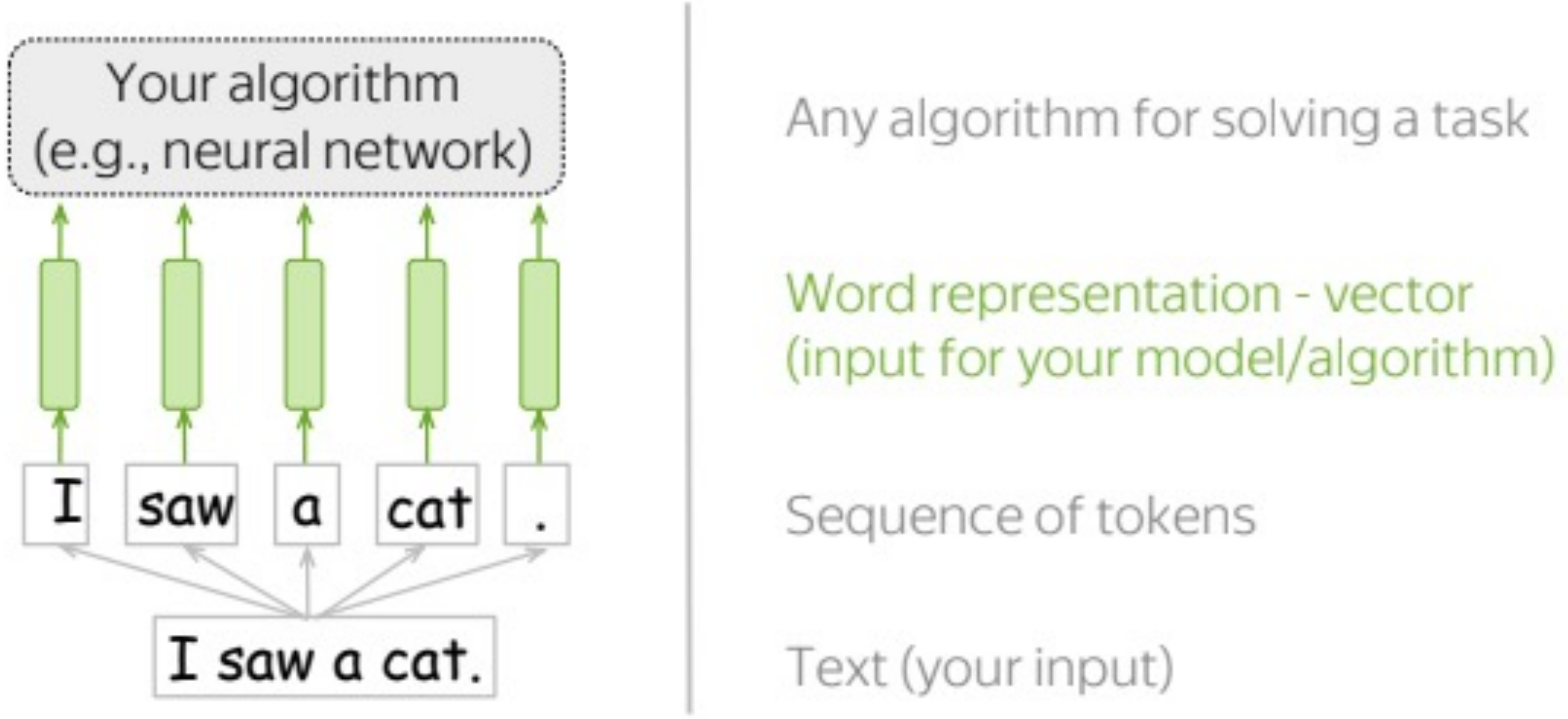
Word	TF		IDF	TF * IDF	
	A	B		A	B
The	1/7	1/7	log(2/2)=0	0	0
Car	1/7	0	log(2/1)=0.3	0.043	0
Truck	0	1/7	log(2/1)=0.3	0	0.043
Is	1/7	1/7	log(2/2)=0	0	0
Driven	1/7	1/7	log(2/2)=0	0	0
On	1/7	1/7	log(2/2)=0	0	0
The	1/7	1/7	log(2/2)=0	0	0
Road	1/7	0	log(2/1)=0.3	0.043	0
Highway	0	1/7	log(2/1)=0.3	0	0.043

Aggregation of word representations

- Traditional methods treat words as indexes rather than concepts
- Each word $\boldsymbol{w} \in \boldsymbol{W}$ is associated with a vector $\boldsymbol{v_w} \in \mathbb{R}^{\boldsymbol{m}}$ — representation of the word (word embedding), m — dimension of the space

Aggregation of word representations

- Traditional methods treat words as indexes rather than concepts
- Each word $w \in W$ is associated with a vector $v_w \in \mathbb{R}^m$ — representation of the word (word embedding), m — dimension of the space
- We will calculate the representation of the document as an aggregate function of the vectors of the document's words



One-hot Vectors

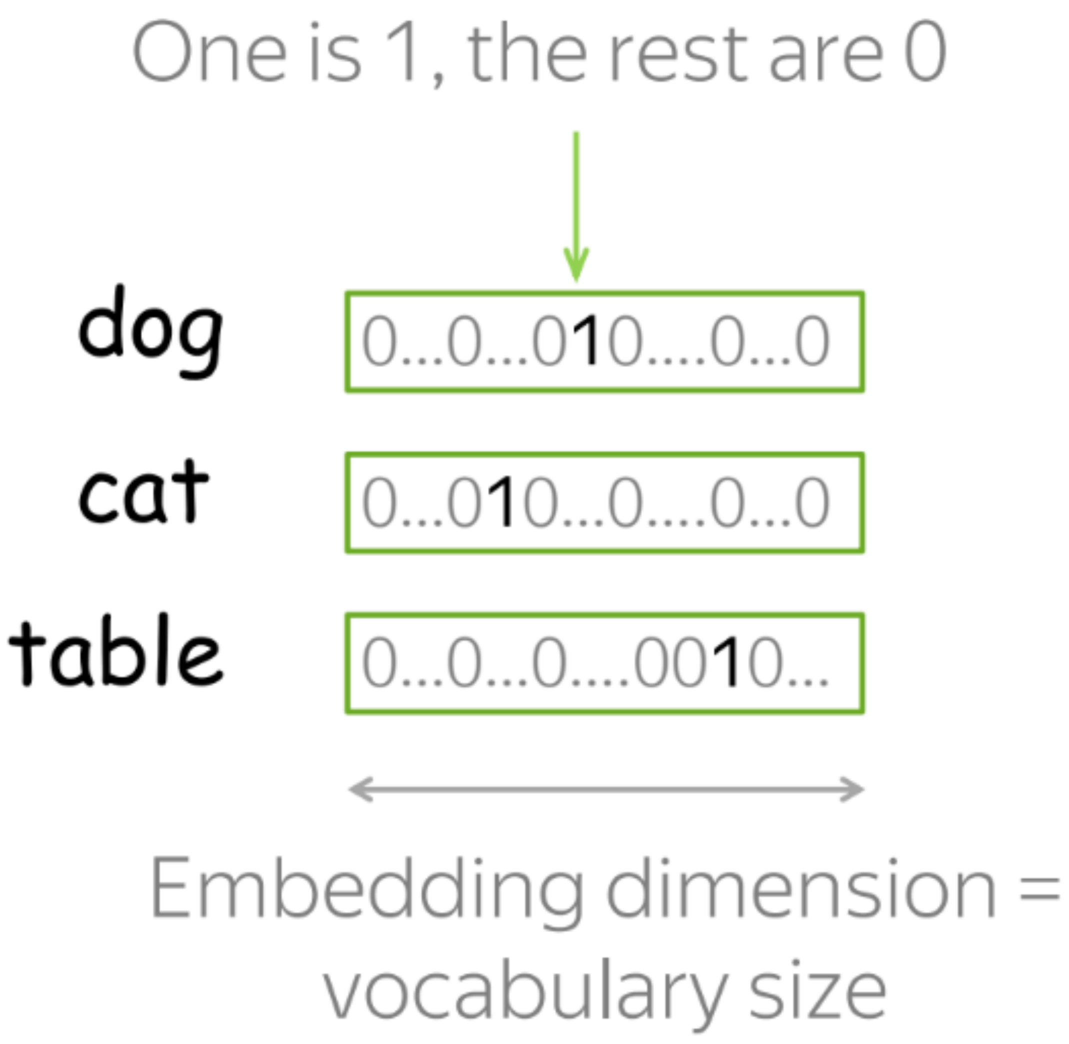
- Each word $\boldsymbol{w} \in \boldsymbol{W}$ corresponds to a one-hot vector:

  $$\boldsymbol{v_w} = [\mathbf{0}, \dots, \mathbf{0}, \mathbf{1}, \mathbf{0}, \dots, \mathbf{0}] \in \mathbb{R}^{|\boldsymbol{W}|}$$

One-hot Vectors

- **Each word $w \in W$ corresponds to a one-hot vector:**

$$v_w = [\mathbf{0}, \dots, \mathbf{0}, \mathbf{1}, \mathbf{0}, \dots, \mathbf{0}] \in \mathbb{R}^{|W|}$$



Problems:

- Sparseness
- Large dimensionality
- Orthogonality of all word representations
- No mechanism for processing unfamiliar words (out of vocabulary, OOV) on the test
- Vectors know nothing about meaning
  "cat is as close to dog as it is to table!"

What is meaning?

- Do you know what the word tezgüino means ?

A bottle of tezgüino is on the table.
Everyone likes tezgüino.
Tezgüino makes you drunk.
We make tezgüino out of corn.

What is meaning?

- Do you know what the word tezgüino means ?

A bottle of tezgüino is on the table.
Everyone likes tezgüino.
Tezgüino makes you drunk.
We make tezgüino out of corn.

Tezgüino is a kind of alcoholic beverage made from corn.

What is meaning?

- Do you know what the word tezgüino means ?

A bottle of tezgüino is on the table.
Everyone likes tezgüino.
Tezgüino makes you drunk.
We make tezgüino out of corn.

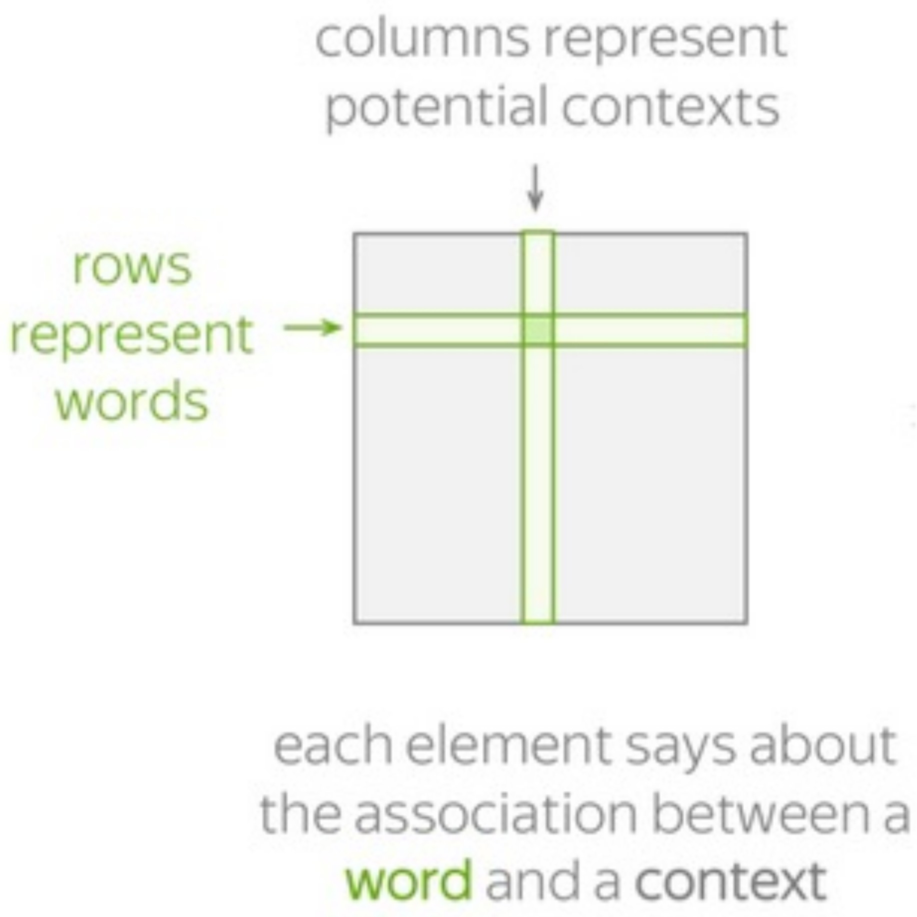
→ Tezgüino is a kind of alcoholic beverage made from corn.

- Distributional Hypothesis (Harris 1954, Firth 1957)

  > Words which frequently appear in similar contexts have similar meaning.

Count-based methods

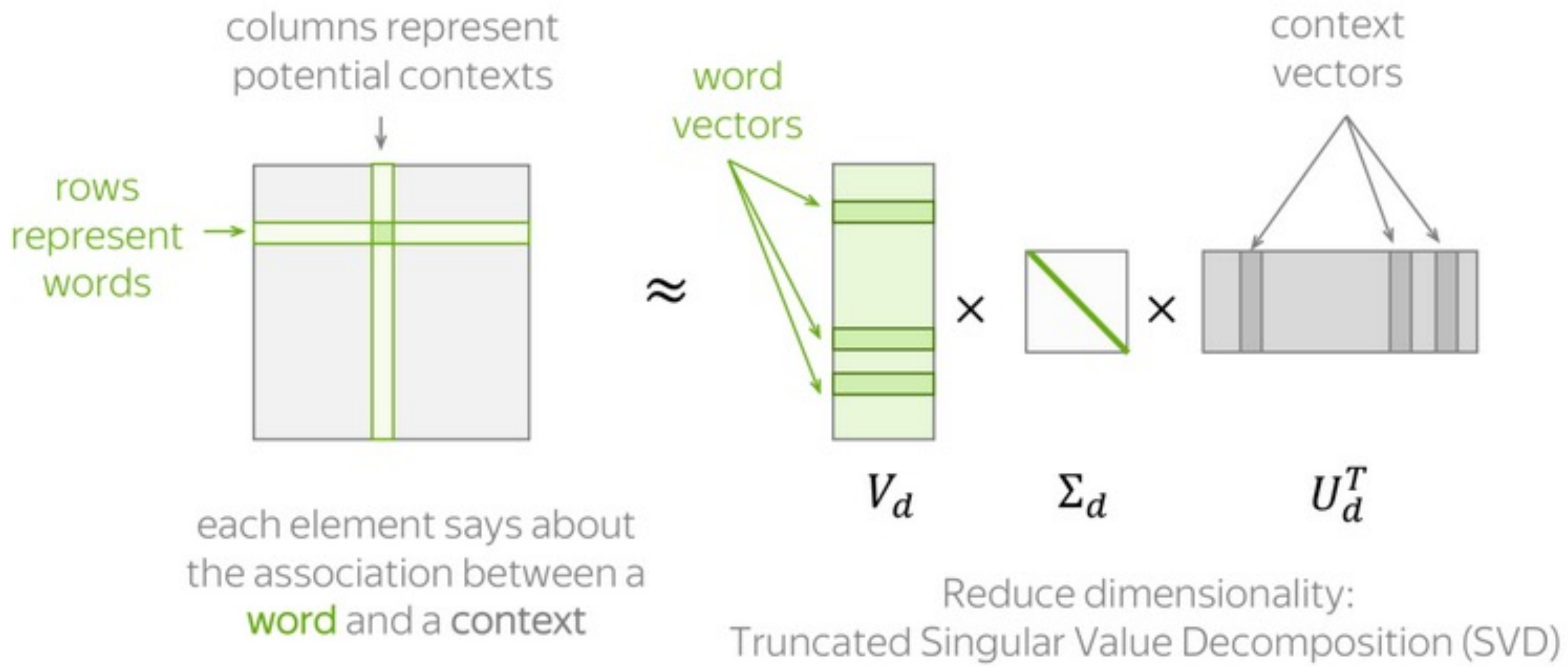
- We have to put information about contexts into word vectors



- How to obtain $v_w \in \mathbb{R}^m$?

SVD for representations

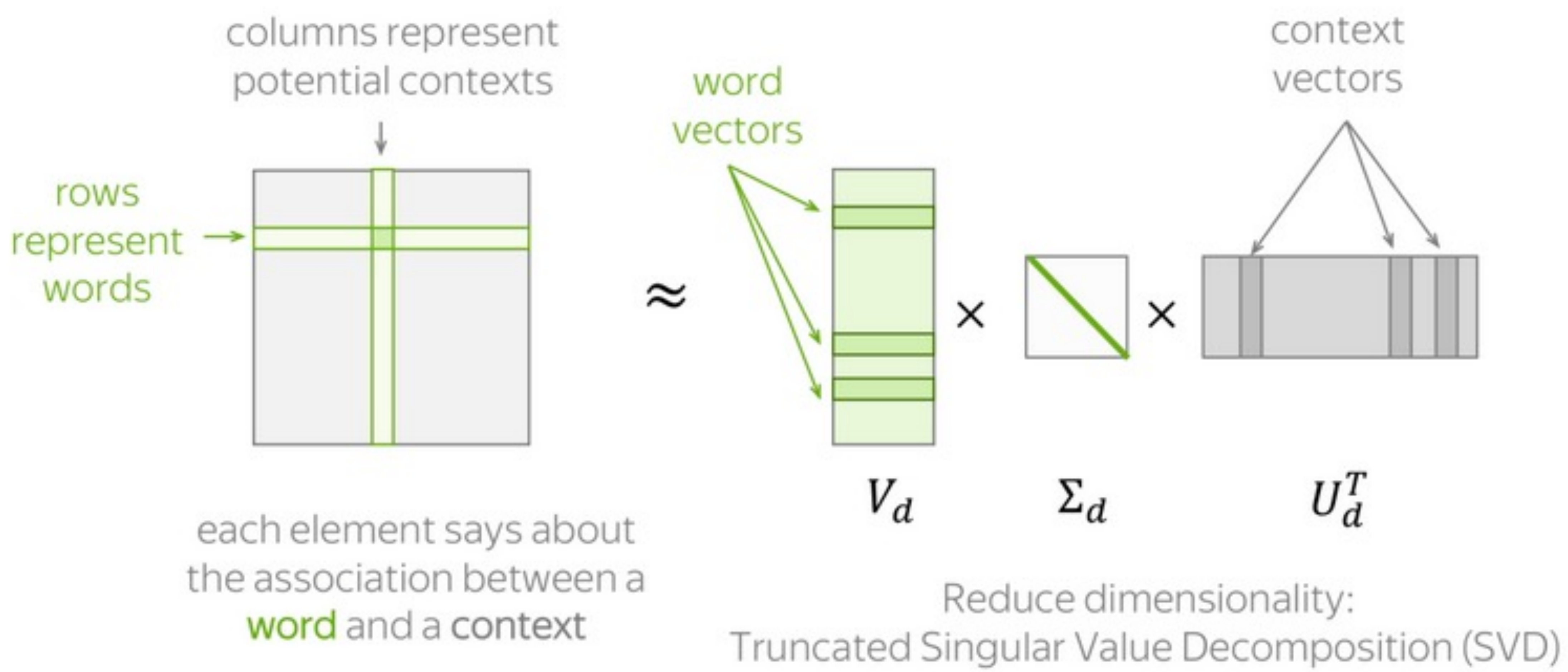
- Word representations via matrix factorization



Initial matrix for the collection $X \in \mathbb{R}^{|W|\times|W|}$, approximation $X \approx \underbrace{V_d}_{\text{word vectors}} \cdot \underbrace{\Sigma_d}_{\text{diagonal}} \cdot \underbrace{U_d^T}_{\text{context vectors}} \rightarrow V_d \approx X U_d \Sigma_d^{-1}$

SVD for representations

- **Word representations via matrix factorization**

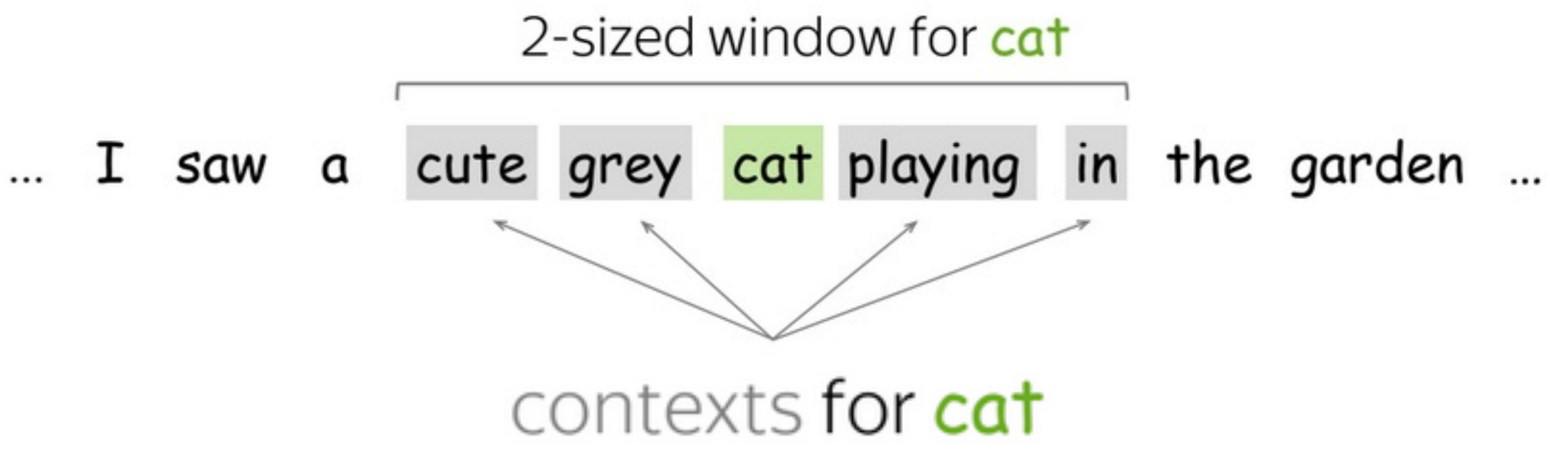


- What is context?
- What is matrix element?

Initial matrix for the collection $X \in \mathbb{R}^{|W|\times|W|}$, approximation $X \approx \underbrace{V_d}_{\text{word vectors}} \cdot \underbrace{\Sigma_d}_{\text{diagonal}} \cdot \underbrace{U_d^T}_{\text{context vectors}} \rightarrow V_d \approx X U_d \Sigma_d^{-1}$

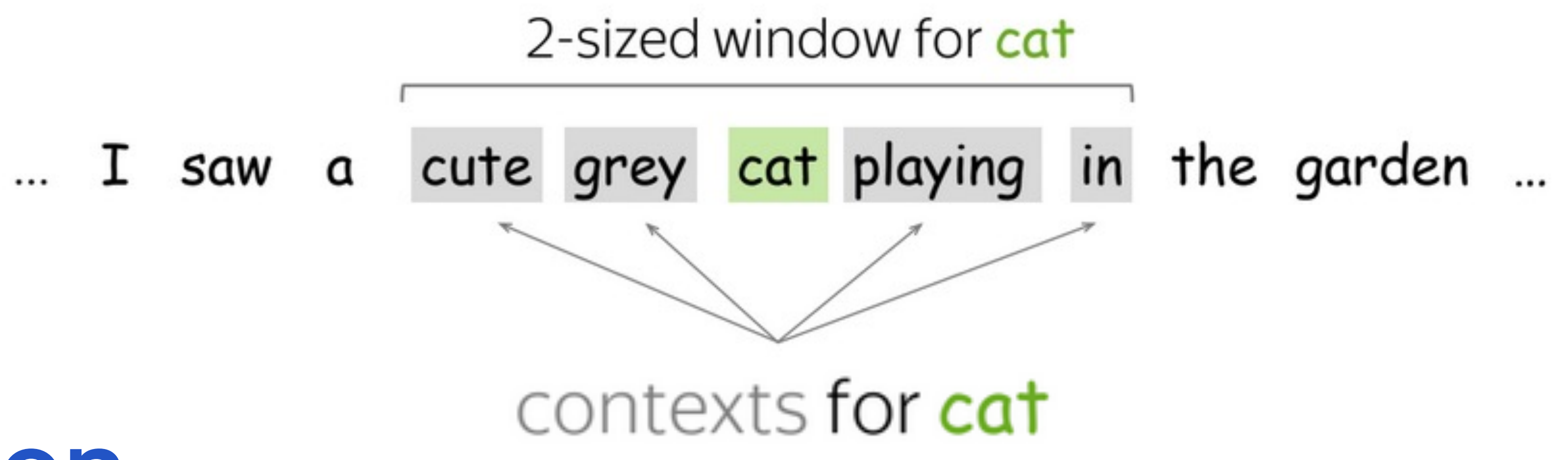
Collocations

- Co-Occurrence Counts: context – surrounding window, matrix element - number of times word appears in context



Collocations

- Co-Occurrence Counts: context – surrounding window, matrix element - number of times word appears in context

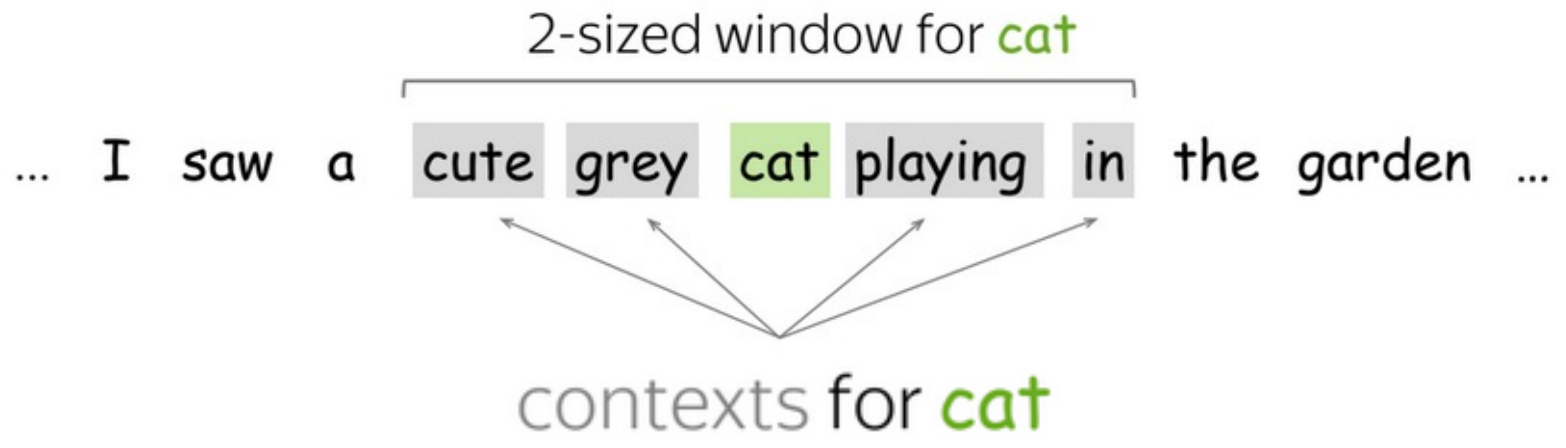


- Positive Pointwise Mutual Information (PPMI): context – surrounding window, matrix element:

$$\mathrm{PPMI}(w, c) = \max(0, \mathrm{PMI}(w, c)), \quad \mathrm{PMI}(w, c) = \log\frac{P(w, c)}{P(w)P(c)} = \log\frac{N(w, c)|(w,c)|}{N(w)N(c)}$$

Collocations

- Co-Occurrence Counts: context – surrounding window, matrix element - number of times word appears in context



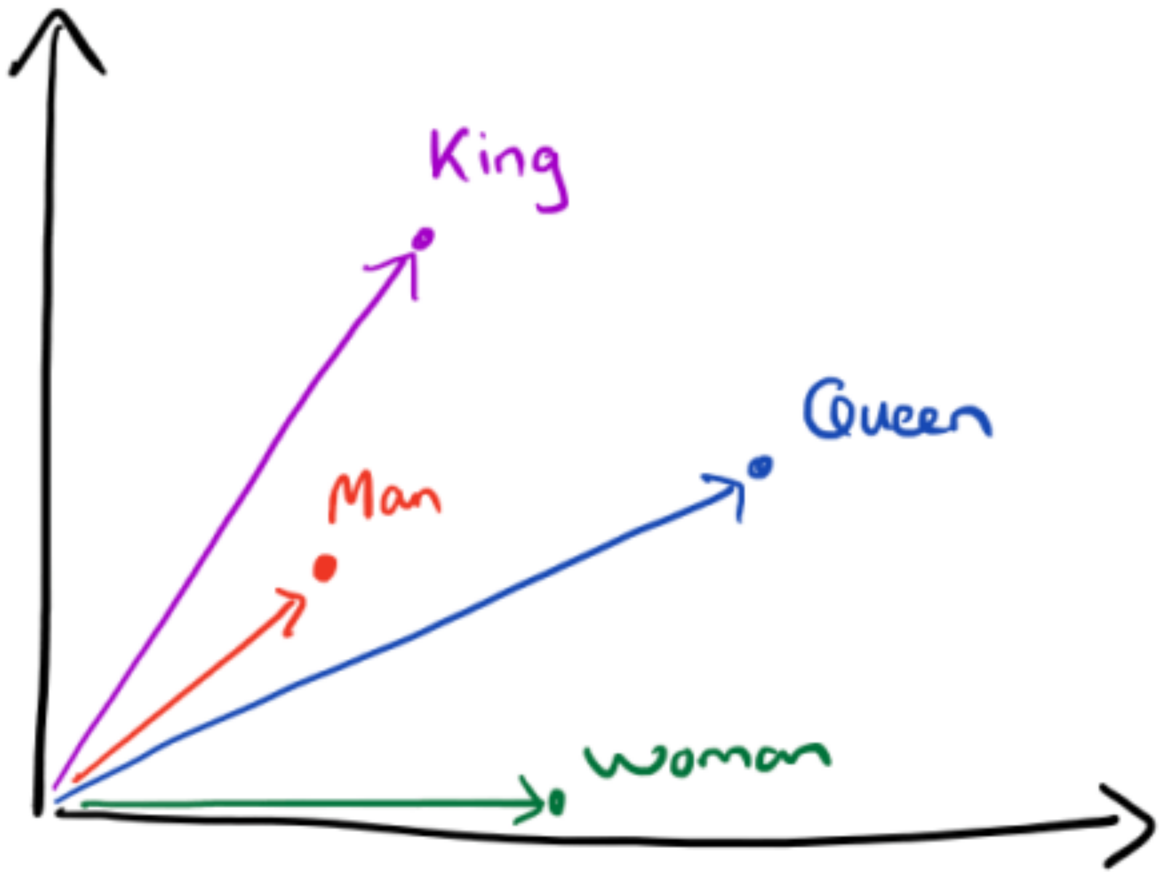
- Positive Pointwise Mutual Information (PPMI): context – surrounding window, matrix element:

$$\mathrm{PPMI}(w, c) = \max(0, \mathrm{PMI}(w, c)), \quad \mathrm{PMI}(w, c) = \log \frac{P(w, c)}{P(w)P(c)} = \log \frac{N(w, c)|(w,c)|}{N(w)N(c)}$$

- Latent Semantic Analysis (LSA): context – document from collection, matrix element – TF-IDF(w, d, D)

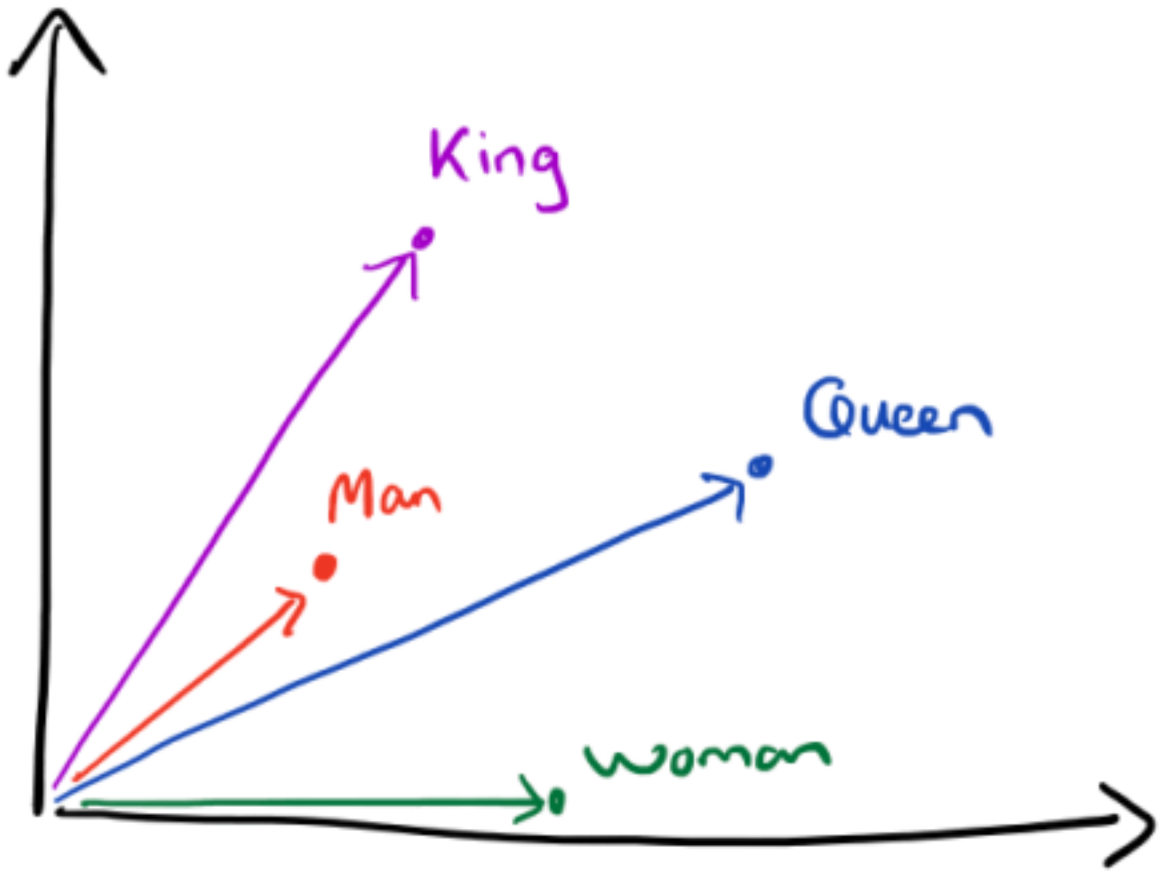
Prediction-based methods

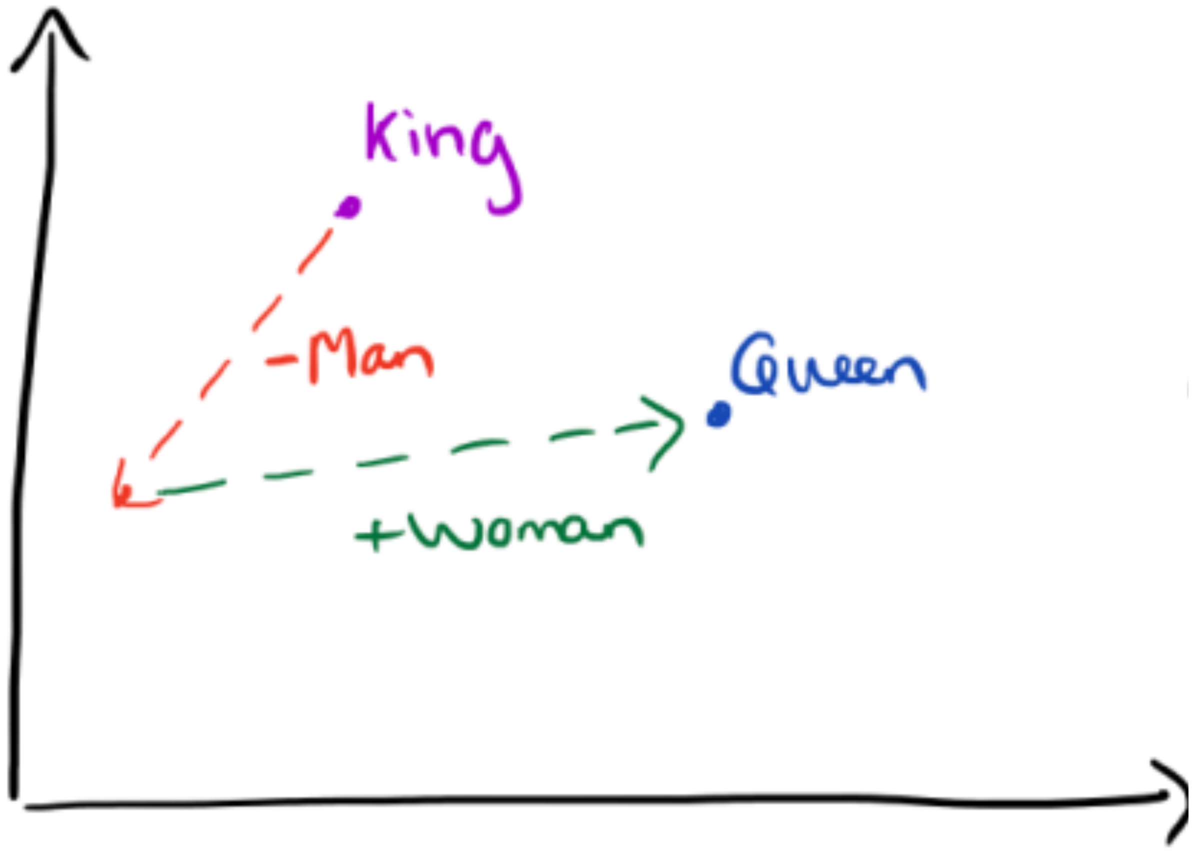
- It is possible to learn word vectors that are able to capture the relationships between words



Prediction-based methods

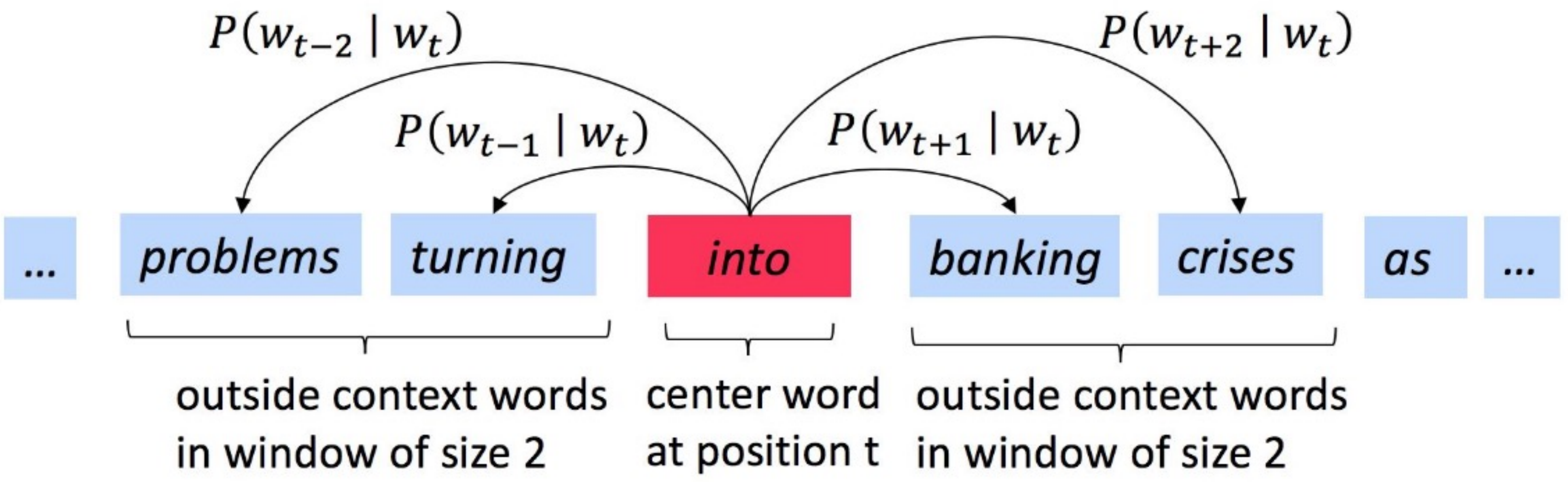
- It is possible to learn word vectors that are able to capture the relationships between words



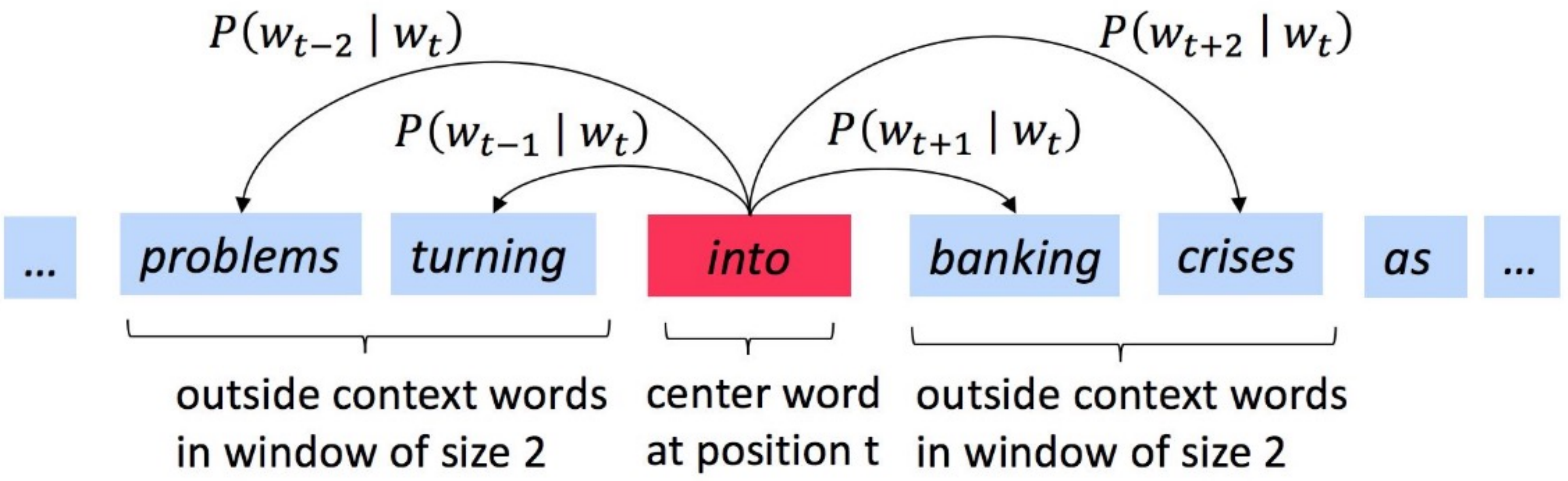


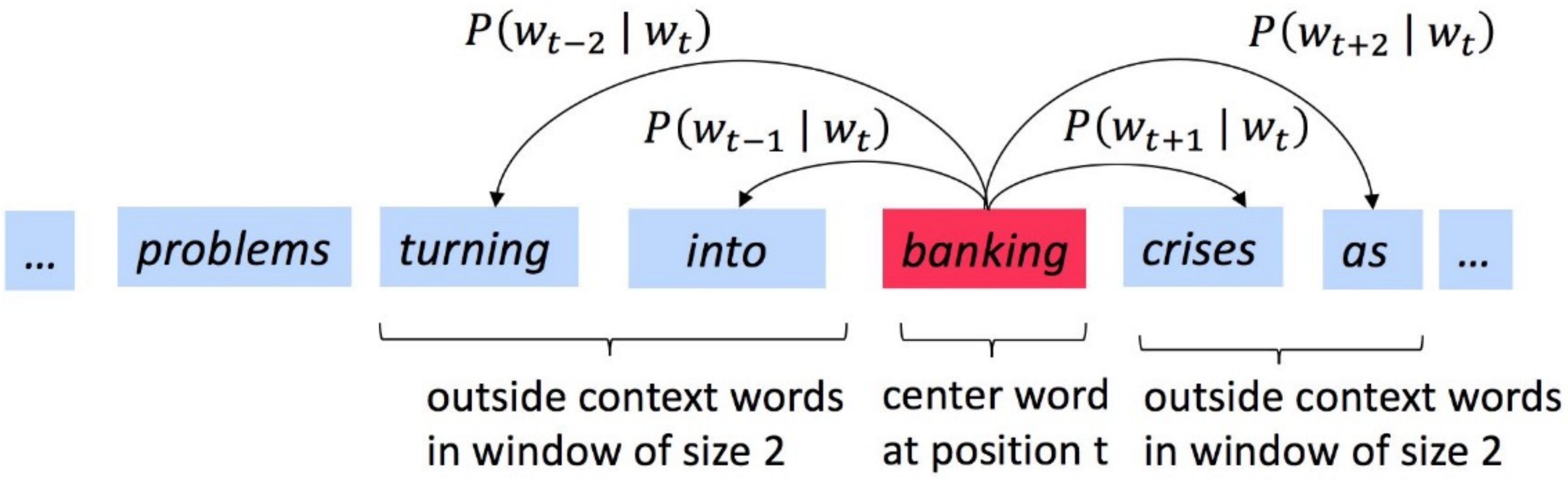
- **Word2vec** (Mikolov et al. 2013) - a framework for learning word embeddings

Word2Vec: Idea



Word2Vec: Idea





Word2Vec: Objective

> Maximize the data likelihood: Likelihood = $L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \leq j \leq m, \\ j \neq 0}} P(w_{t+j} | w_t, \theta)$

Word2Vec: Objective

> Maximize the data likelihood:

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m, \\ j \ne 0}} P(w_{t+j} | w_t, \theta)$$

> Uses negative (log-)likelihood as its loss function:

$$\text{Loss} = J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m, \\ j \ne 0}} \log P(w_{t+j} | w_t, \theta)$$

Word2Vec: Objective

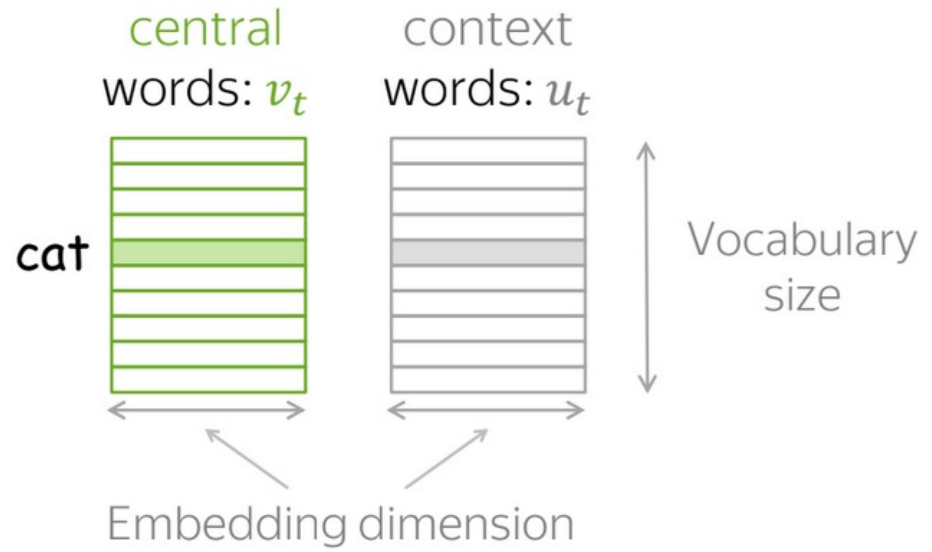
> Maximize the data likelihood: Likelihood $= L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m, \\ j \ne 0}} P(w_{t+j} | w_t, \theta)$

> Uses negative (log-)likelihood as its loss function:

$$\text{Loss} = J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m, \\ j \ne 0}} \log P(w_{t+j} | w_t, \theta)$$

> For each word w, we will have two vectors: v_w when it is a central word, u_w when it is a context word:

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$



Word2Vec: Objective

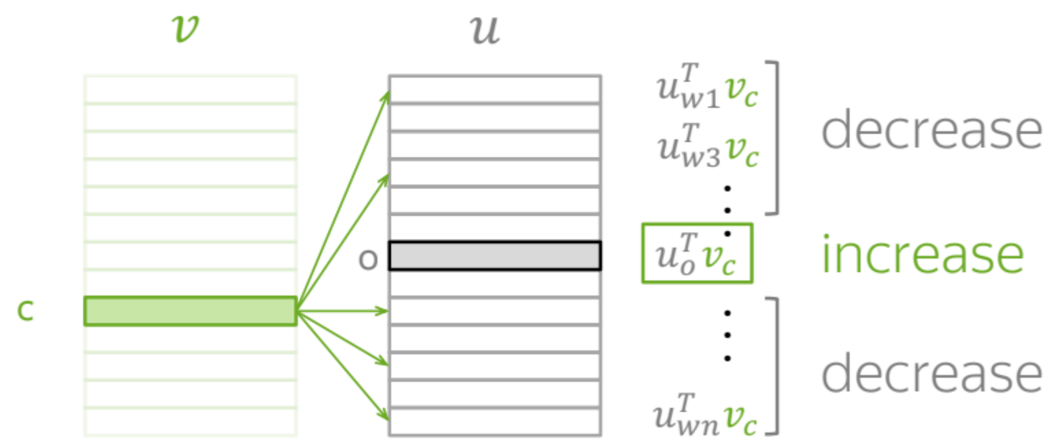
> Maximize the data likelihood:

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m, \\ j \neq 0}} P(w_{t+j} | w_t, \theta)$$

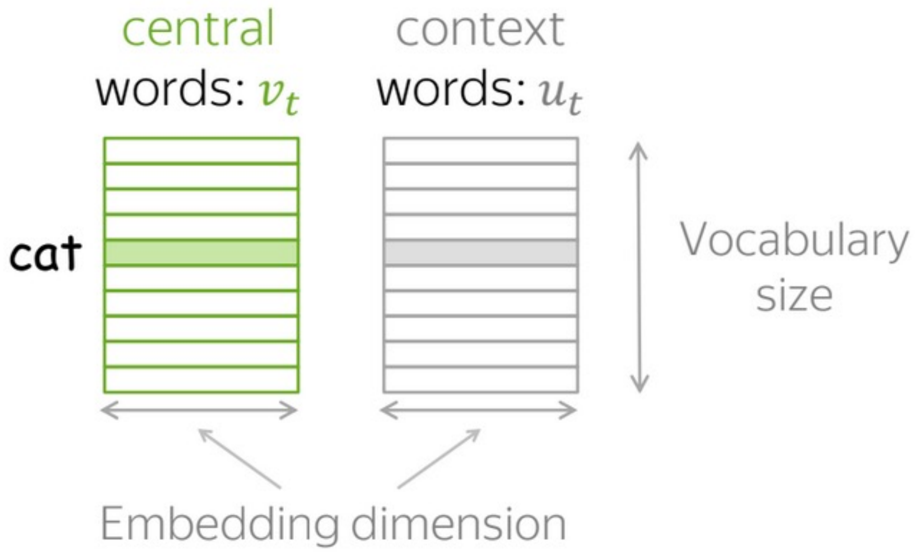
> Uses negative (log-)likelihood as its loss function:

$$\text{Loss} = J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m, \\ j \neq 0}} \log P(w_{t+j} | w_t, \theta)$$

> For each word w, we will have two vectors: v_w when it is a central word, u_w when it is a context word:

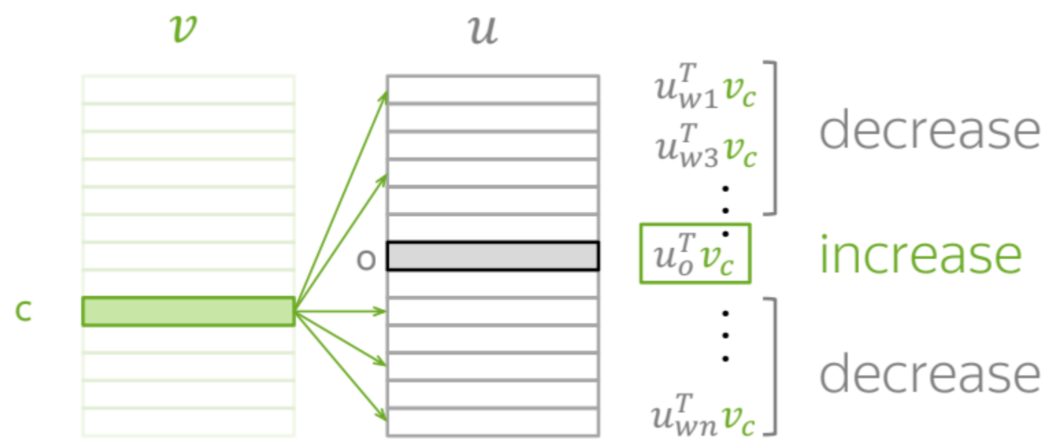


$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$



Word2Vec: Improvements

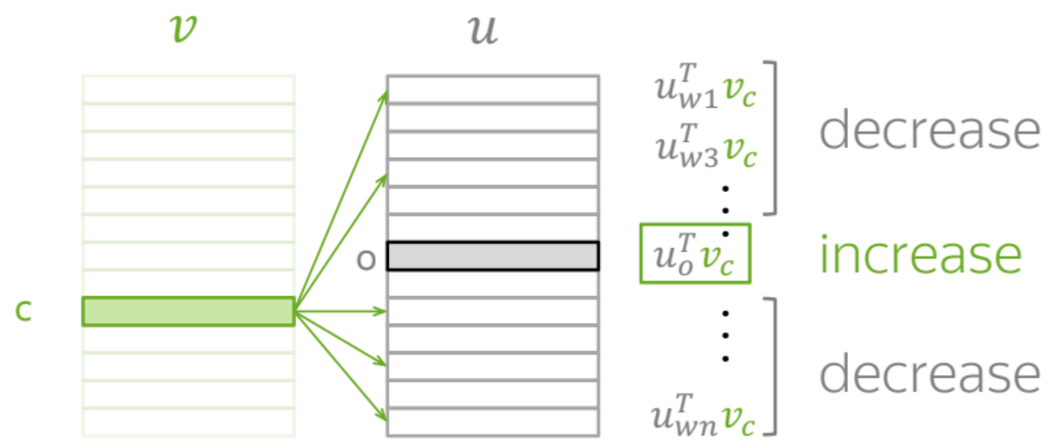
> **Negative sampling:**



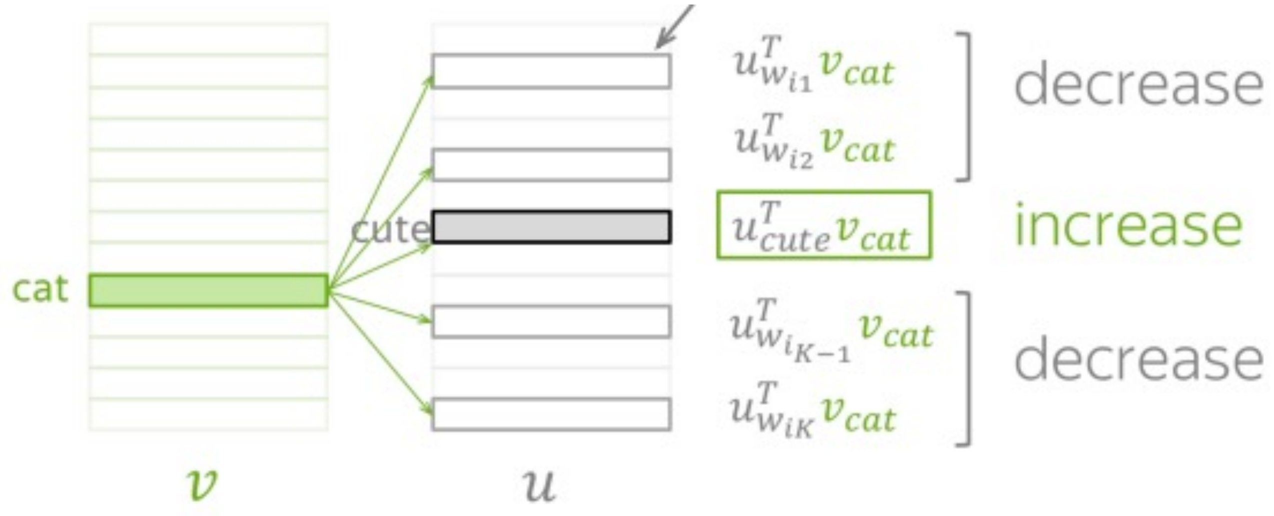
Need to update |W| + 1 vector

Word2Vec: Improvements

> Negative sampling:



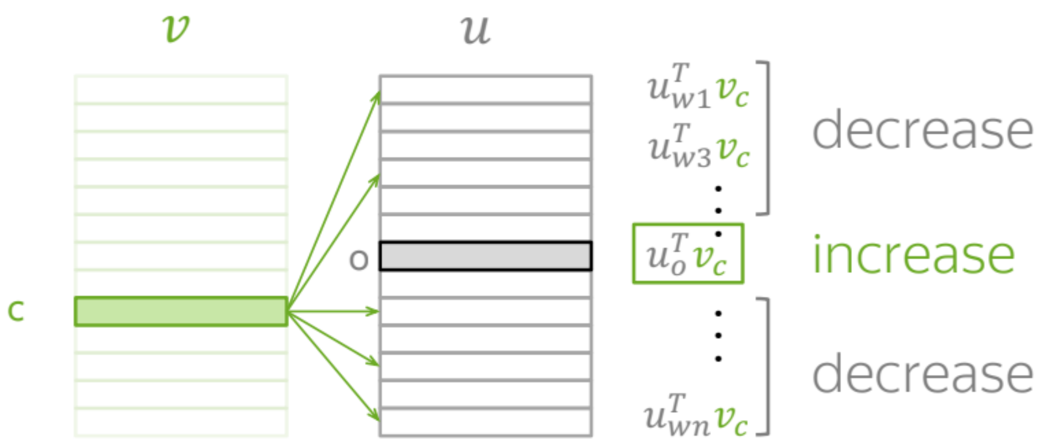
Need to update |W| + 1 vector



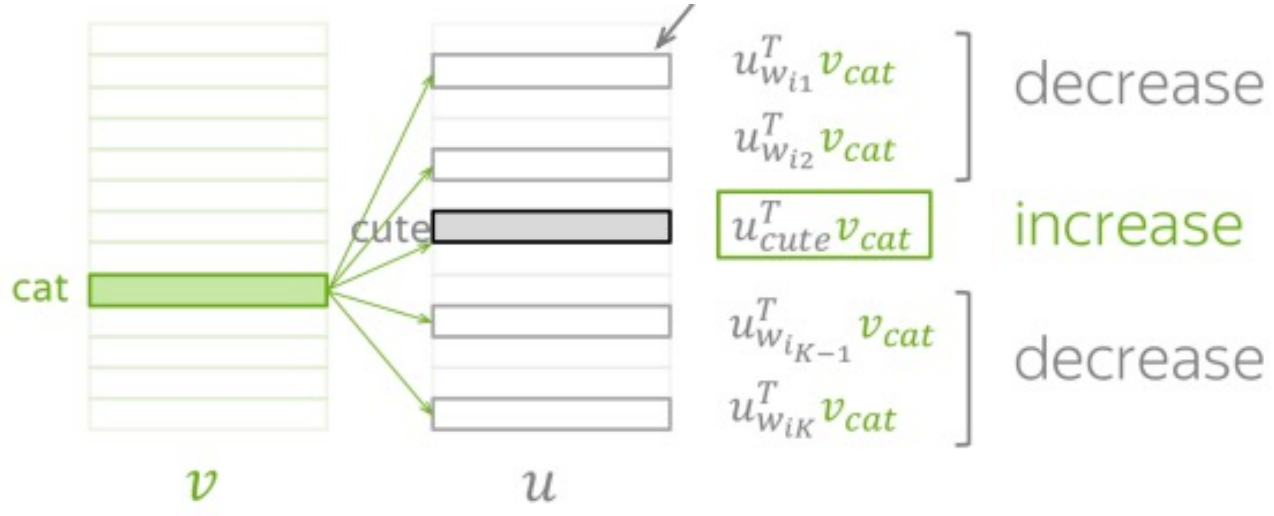
Randomly select K words

Word2Vec: Improvements

> Negative sampling:



Need to update |W| + 1 vector



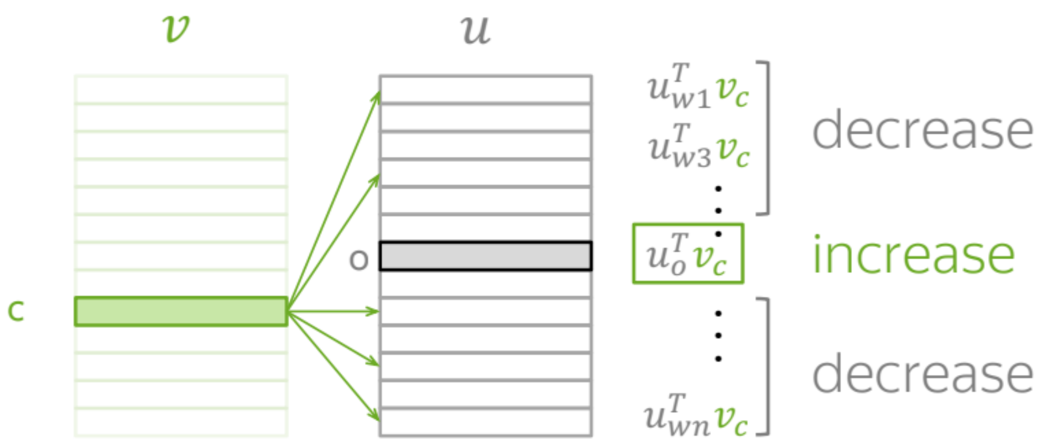
Randomly select K words

> Hierarchical softmax:

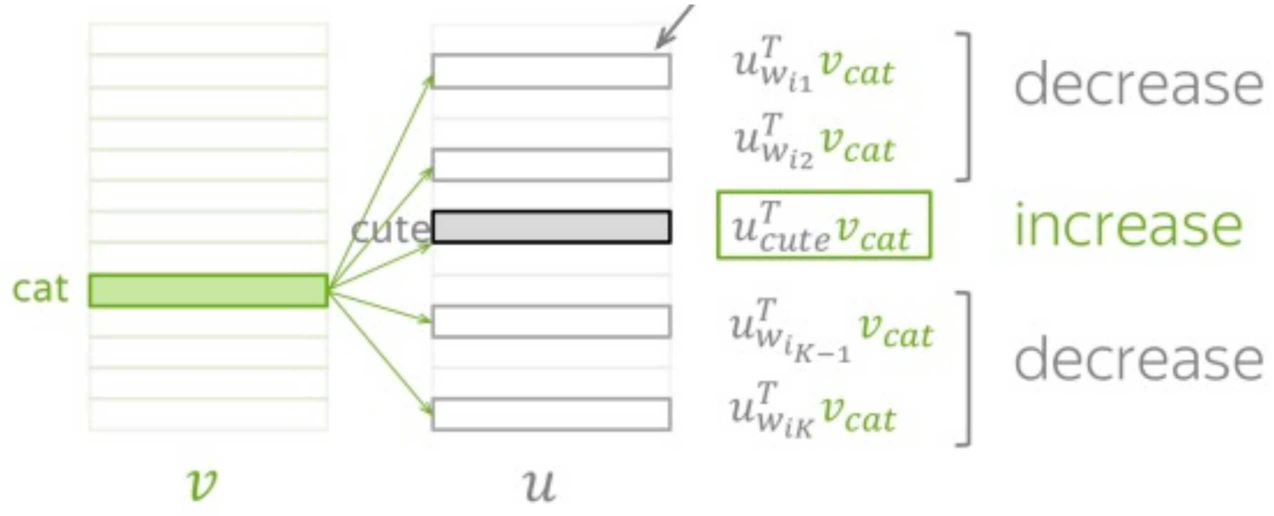
Idea: replace softmax with another function whose optimisation will have a complexity of $O(\log |W|)$.

Word2Vec: Improvements

> Negative sampling:



Need to update |W| + 1 vector



Randomly select K words

> Hierarchical softmax:

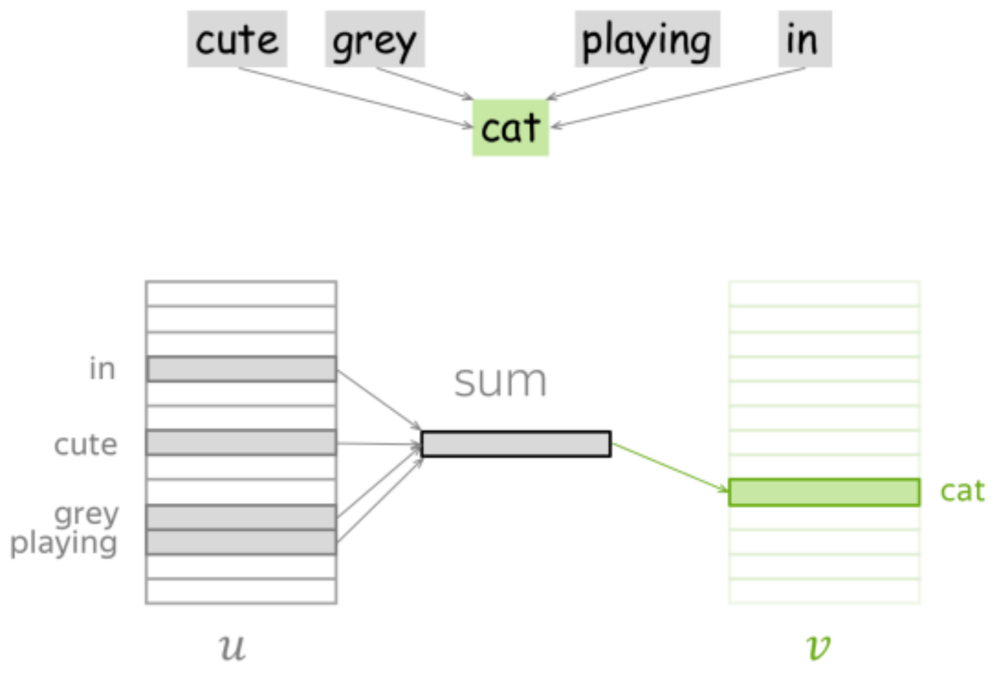
Idea: replace softmax with another function whose optimisation will have a complexity of $O(\log |W|)$.

Before training the model on a set of word pairs and their frequencies, a Huffman binary tree is constructed.

Each node of the tree corresponds to a trainable representation. The leaves of the tree correspond to words. The representations in the leaves are the desired representations for the words.

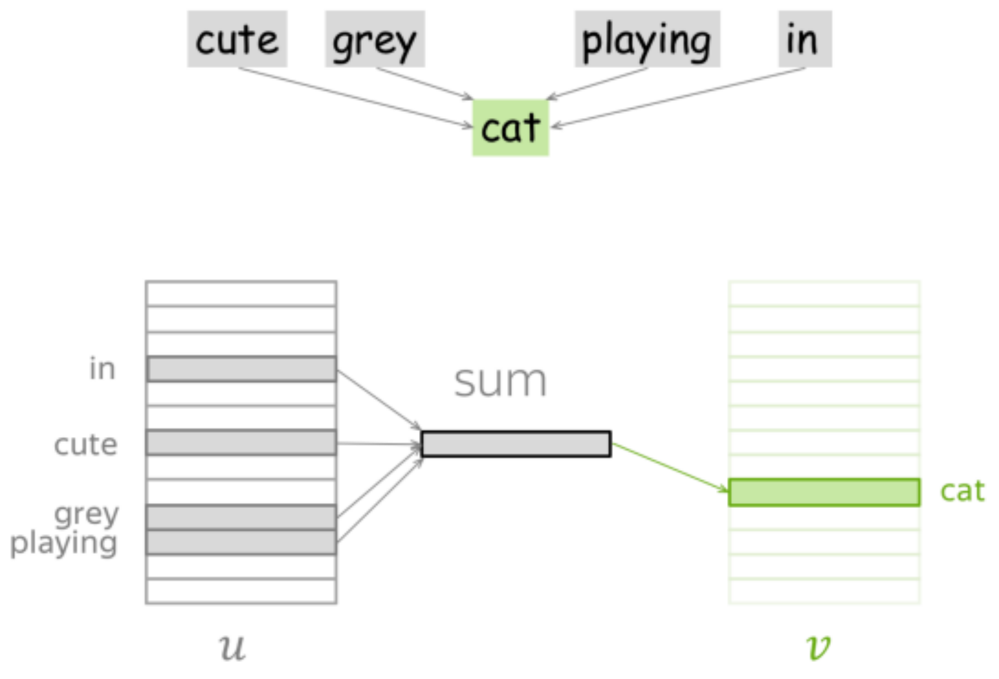
Word2Vec: Two methods

Continuous BOW (CBOW)

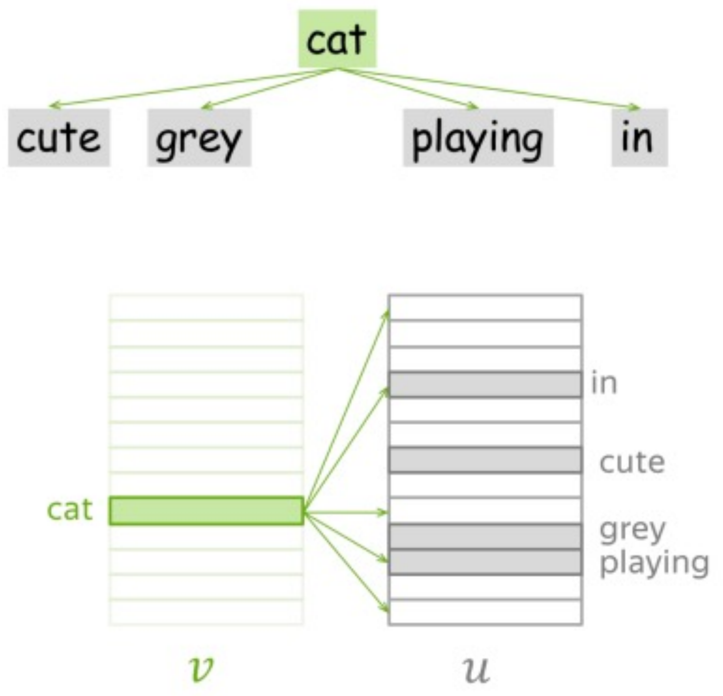


Word2Vec: Two methods

Continuous BOW (CBOW)

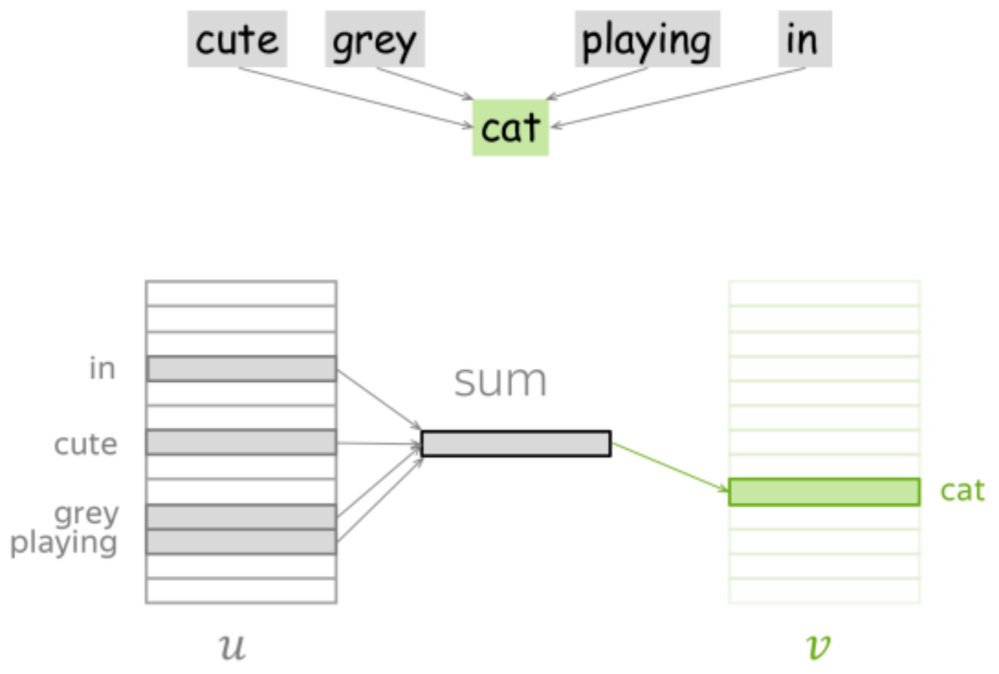


Skip-gram



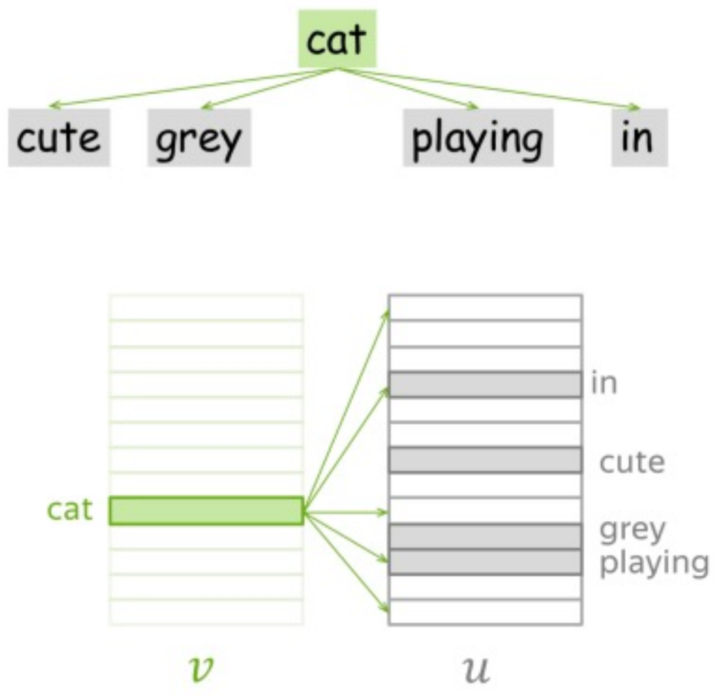
Word2Vec: Two methods

Continuous BOW (CBOW)



> From sum of context predict central
> Predicting one word each time
> Relatively fast

Skip-gram



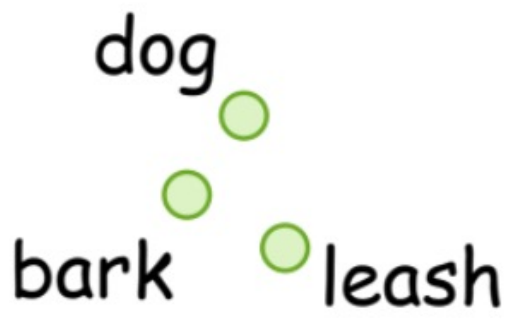
> From central predict context (one at a time)
> Much slower
> Better with infrequent words

Word2Vec: Additional

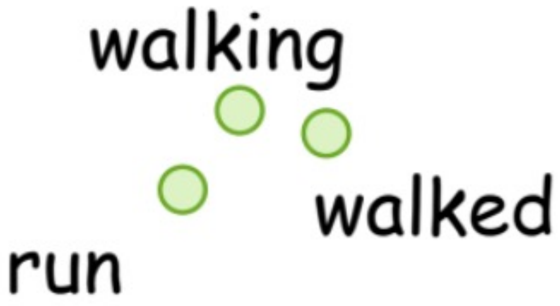
- **Dynamic window** — random selection of context size at each iteration or 5-10
- **Most popular:** Skip-Gram with negative sampling
- Number of **negative examples**:
  - for smaller datasets 15-20
  - for huge datasets it can be 2-5
- **Embedding dimensionality**: frequently used value is 300, but other variants (e.g., 100 or 50) are also possible

Word2Vec: Practical tips

- Larger windows – more topical similarities

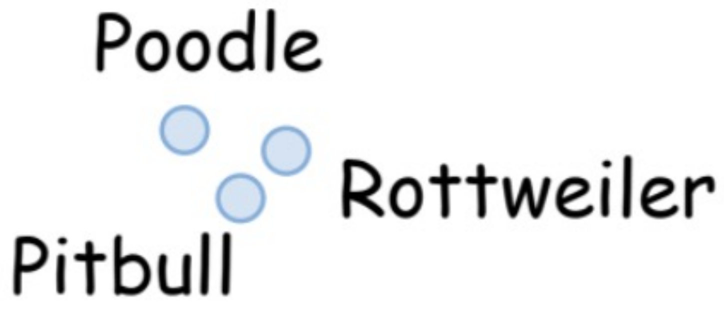


(grouped together)



(grouped together)

- Smaller windows – more functional and syntactic similarities



(grouped together)

(grouped together)

GloVe

	Count-based	Prediction-based
Information comes from:	global corpus statistics	“reading” text corpora
Vectors are:	obtained via dimensionality reduction	learned by gradient descent

GloVe

Global Vectors for word representations

	Count-based	GloVe	Prediction-based
Information comes from:	global corpus statistics →	global corpus statistics	"reading" text corpora
Vectors are:	obtained via dimensionality reduction	learned by gradient descent	← learned by gradient descent

GloVe

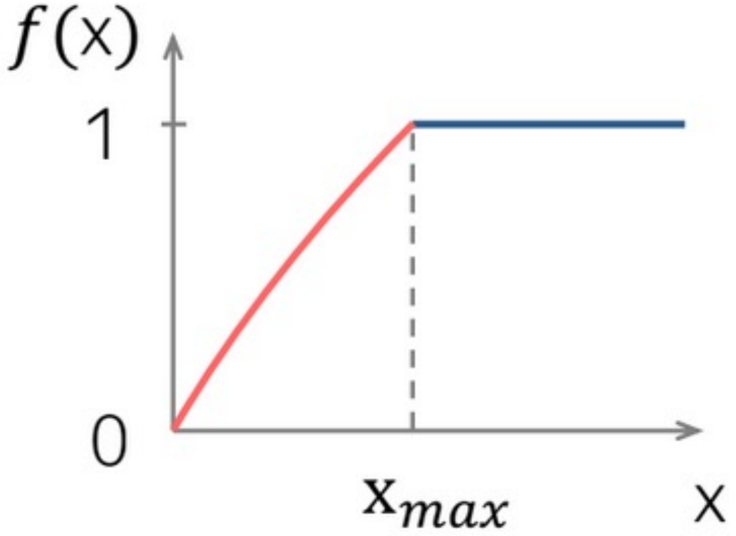
> Loss function:

context vector

word vector

bias terms (also learned)

$$J(\theta) = \sum_{w,c \in V} f(\mathrm{N}(w,c)) \cdot (u_c^T v_w + b_c + \overline{b_w} - \log \mathrm{N}(w,c))^2$$



$$\begin{cases} (x/x_{max})^{\alpha} & \text{if } x < x_{max}, \\ 1 & \text{otherwise.} \end{cases}$$

$\alpha = 0.75$, $x_{max} = 100$

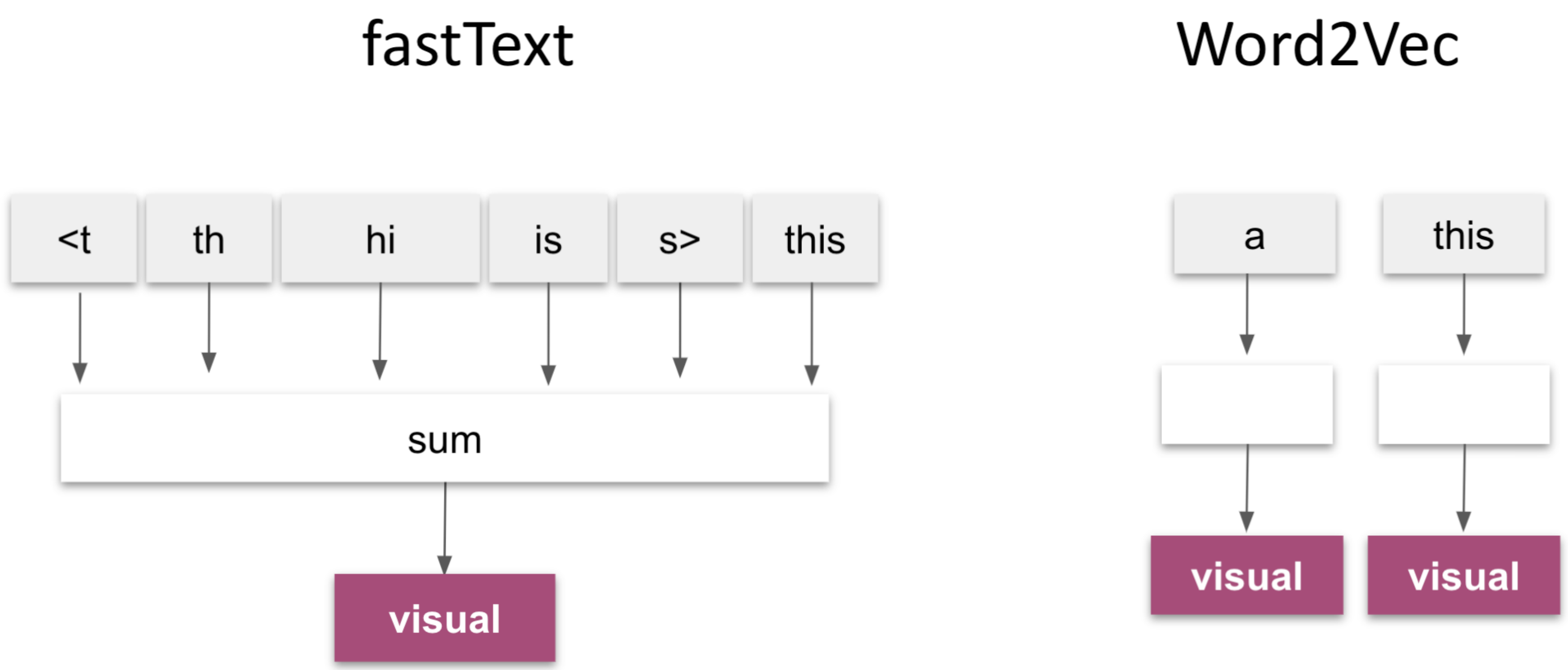
> Popular, but in practice usually worse than Word2vec

FastText

- Breaks words into subword components (n-grams) and learns embeddings for these subwords

FastText

- Breaks words into subword components (n-grams) and learns embeddings for these subwords
- Can represent the word that was not presented in training set using the existing subword embeddings for its subwords



Evaluation

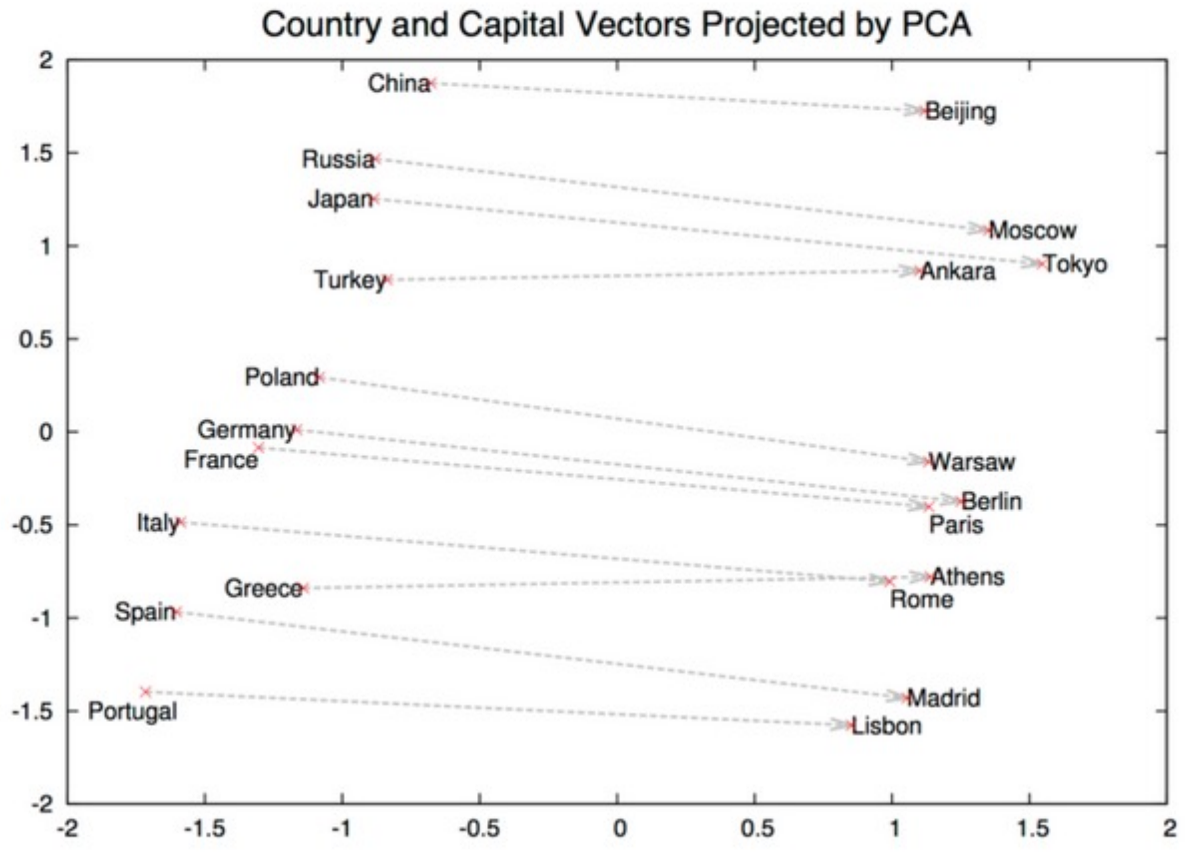
- ## Which representations should be considered good?

  > Words that are close in meaning correspond to vectors that are close in distance
  > Small dimension
  > Interpreted arithmetic operations in vector space
  > Quality of the solution to the finite problem.

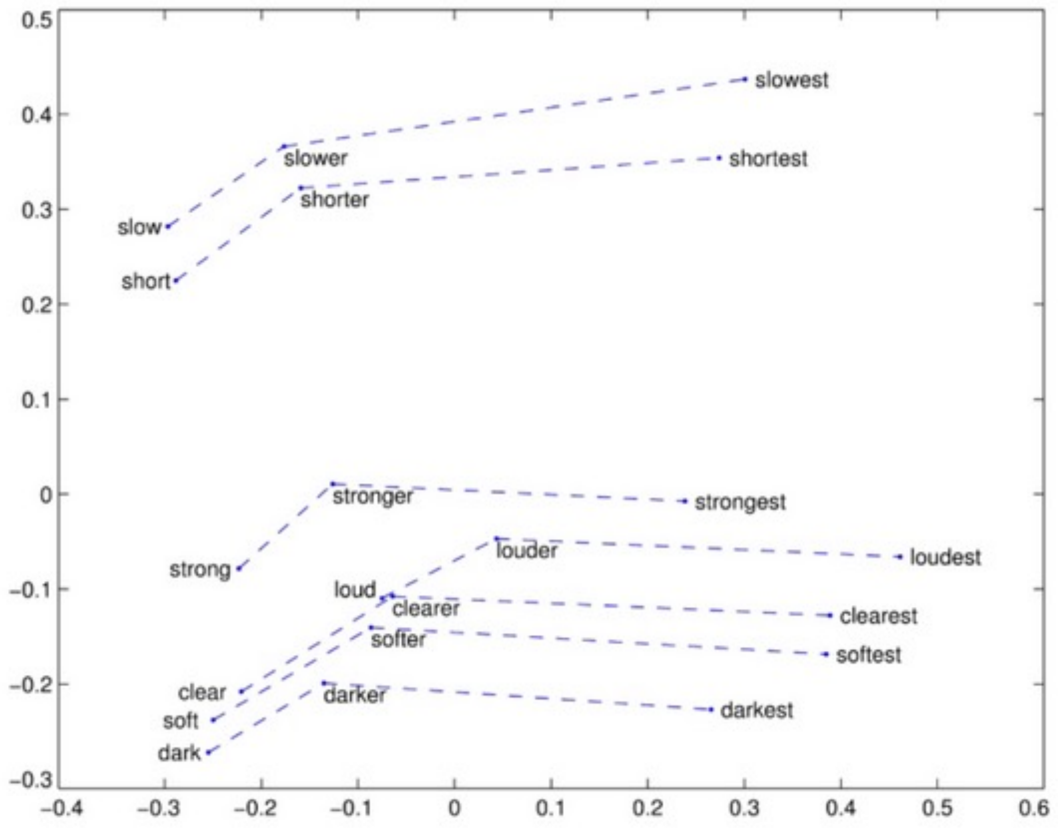
Evaluation

- ## Which representations should be considered good?

  > Words that are close in meaning correspond to vectors that are close in distance
  > Small dimension
  > Interpreted arithmetic operations in vector space
  > Quality of the solution to the finite problem.



Word2Vec



GloVe

Evaluation

- **Which representations should be considered good?**

  > Words that are close in meaning correspond to vectors that are close in distance
  > Small dimension
  > Interpreted arithmetic operations in vector space
  > Quality of the solution to the finite problem.

- **How to use word embeddings?**

  > Solve tasks involving searching for similar words, synonyms, etc.
  > Obtain a representation of a document/sentence that can be used to solve a machine learning task
  > Use the word representation as a fixed representation in a complex architecture (e.g., a recurrent network)
  > Use to initialize representations in a complex architecture

Current state for embeddings

- **Lexical matching**
  > TF-IDF, BM-25 [fast, interpretable]
- **Semantic similarity**
  > Sentence Transformers (bi-encoder), BERT-like models
- **Reranking**
  > Cross-encoder [slower, higher quality]
- **Generation & reasoning**
  > LLMs [prompting / fine-tuning]

Recap

- Sparse / count-based representations
- Predictive embeddings
- Extensions & alternatives as FastText, GloVe for OOV and speed
- Representation choice = trade-off between interpretability, compute, and semantic power

Next:

- Language modelling & Attention