Word embeddings

Based on lecture by Murat Apishev

- **Word embeddings idea.**
  One-hot vectors. SVD

- **Word2vec model, training methods**

- **Word2vec training optimization: hierarchical softmax and SGNS**

- **GloVe model**

- **FastText model, Hashing Trick**

- **Word embeddings models’ quality evaluation**

Word embeddings idea. One-hot vectors. SVD

Input data may have different format

Visual content

- Images
- Video

Texts

- Unstructured documents
- HTML / XML

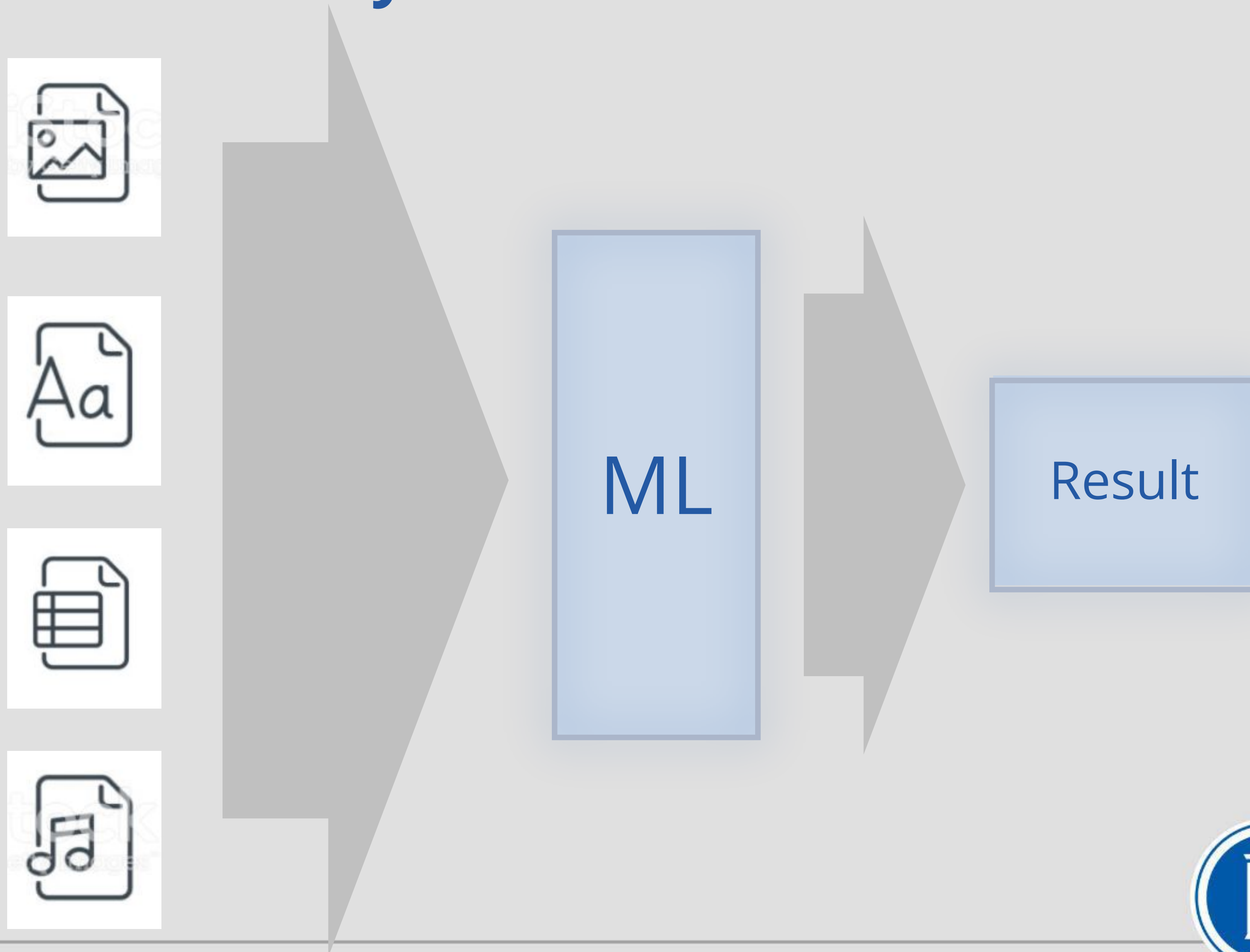
Structured documents

- Tables with features

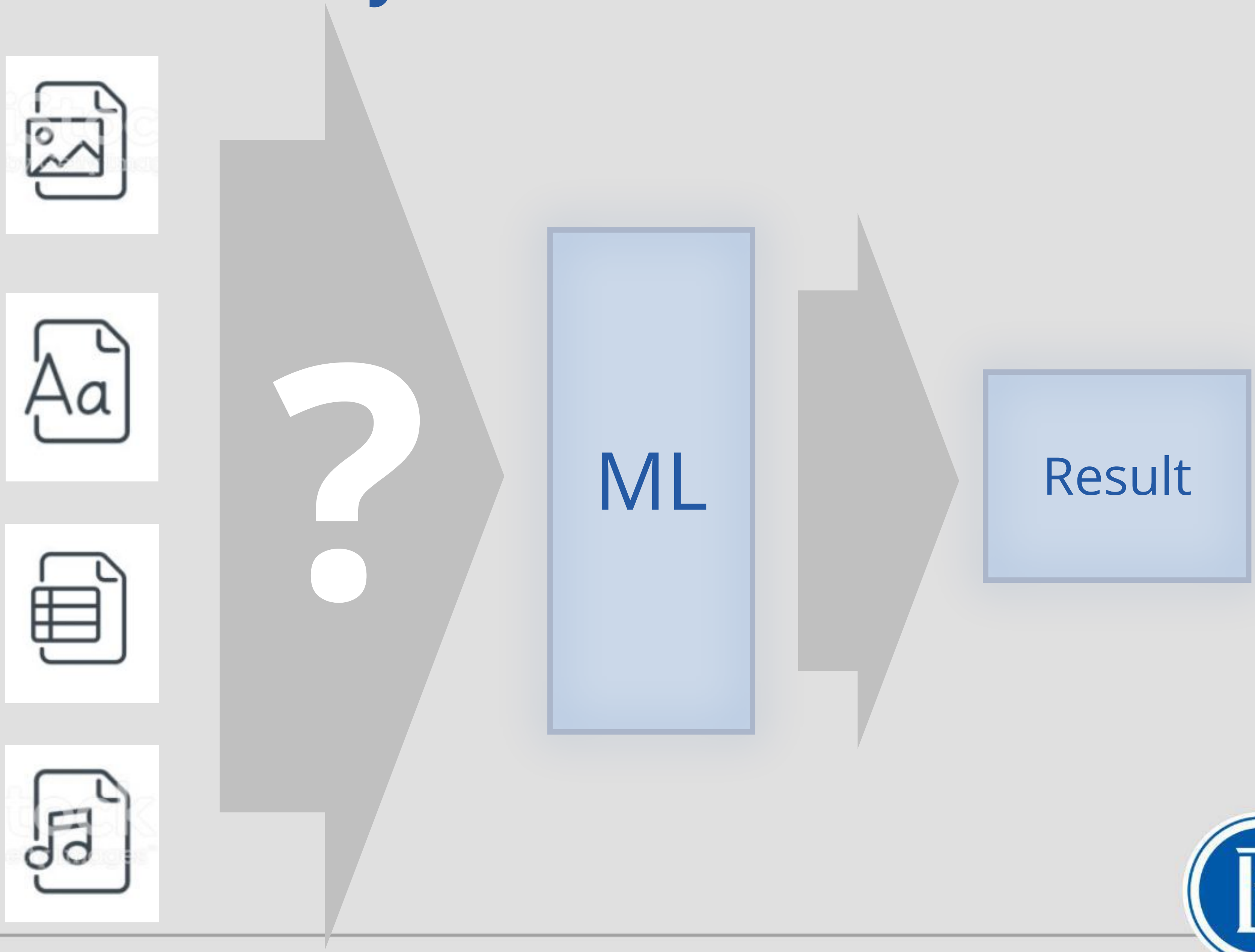
Signals

- Audio: music / speech
- Other signals

Input data may have different format



Input data may have different format



Word embeddings

text

“I love watching TV series”

“I” “love” “watching” “TV” “series”

word

$$\begin{bmatrix}123\\456\\12\\\dots\\89\end{bmatrix} \begin{bmatrix}23\\372\\8\\\dots\\83\end{bmatrix} \begin{bmatrix}16\\124\\76\\\dots\\29\end{bmatrix} \begin{bmatrix}2\\12\\299\\\dots\\65\end{bmatrix} \begin{bmatrix}177\\6\\504\\\dots\\304\end{bmatrix}$$

embeddings

What kind of embeddings do we want?

Different words ⟺ Different embeddings

Ecology
Love

Close words ⟺ Close vectors

Love
Adore

What does «close» mean?

Semantic closeness

«Usual» word closeness

Examples:

- computer
- laptop
- PC

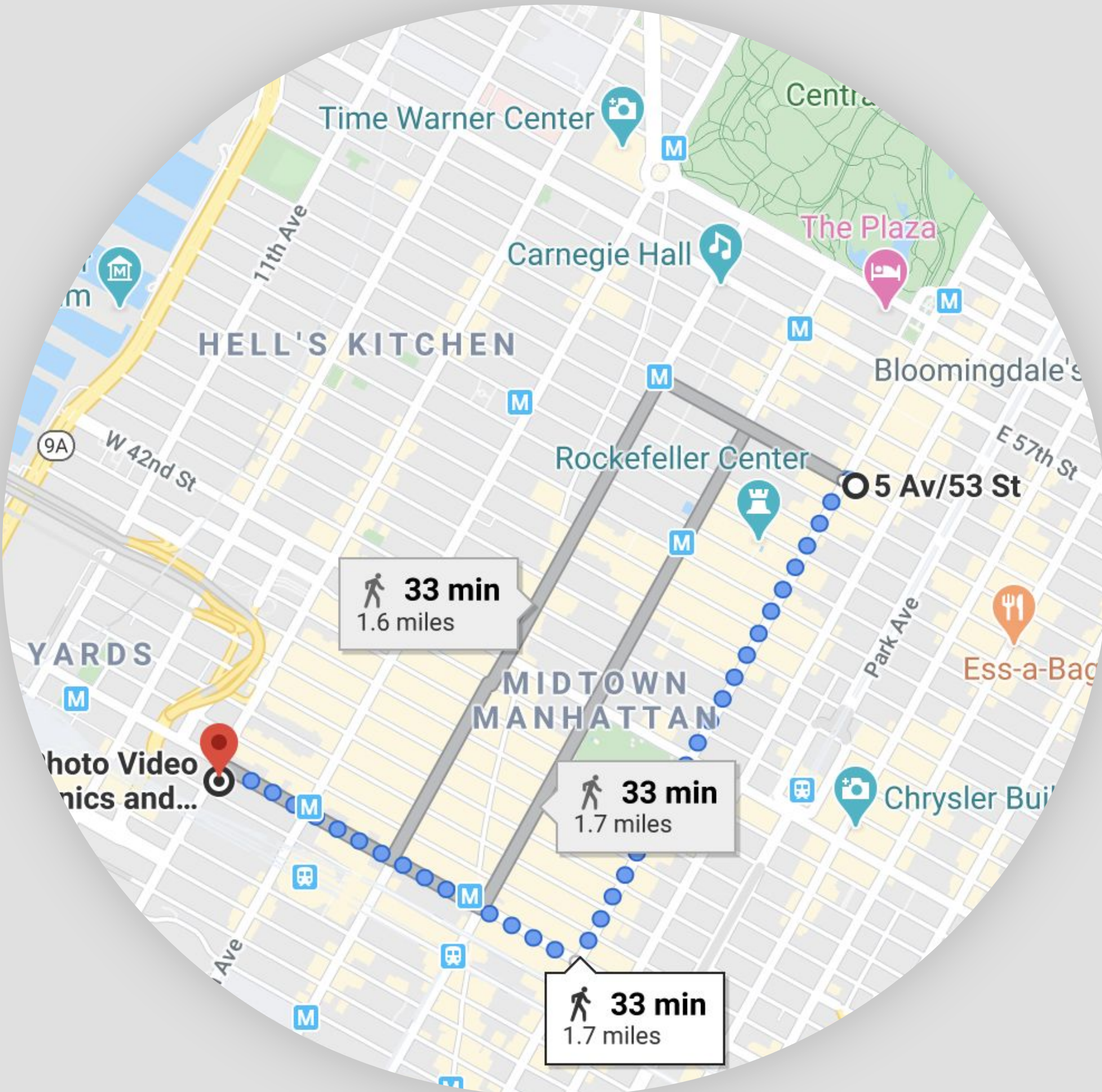
Closeness of embeddings

sim(w, v)

where sim() can be:

- Manhattan distance
- Euclidean distance
- Cosine distance

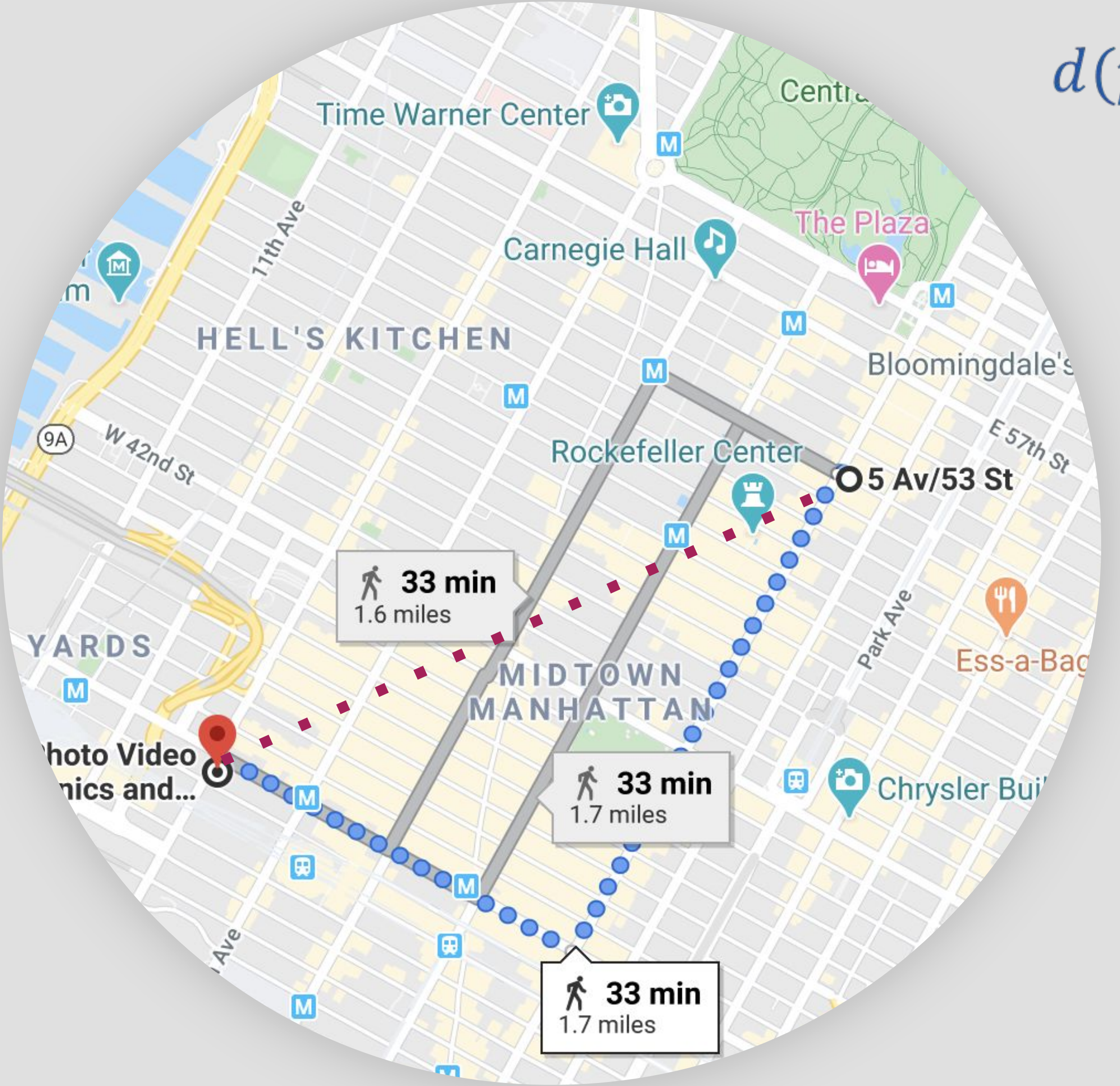
Manhattan distance



$$d(p,q) = \sum_{i=1}^{n} |p_i - q_i|$$

google.com/maps

Euclidean distance



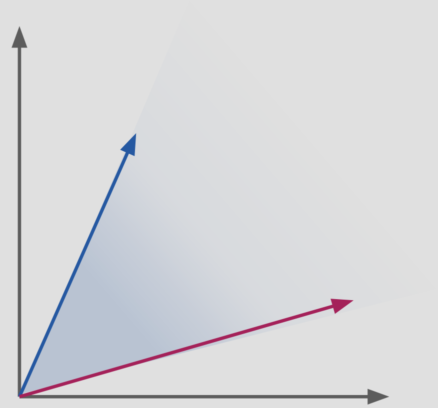
$$d(p,q) = \sqrt{\sum_{i=1}^{n} |p_i - q_i|^2}$$

google.com/maps

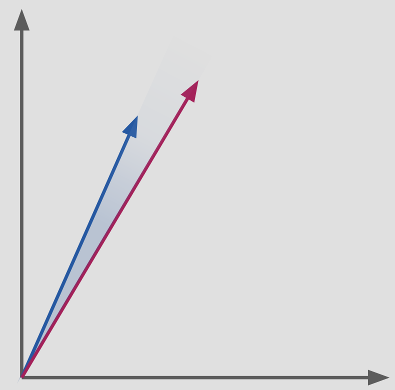
Cosine distance

$$p \cdot q = \| p \| \| q \| \cos(\theta)$$

- $$\cos(\theta) = \frac{p \cdot q}{\| p \| \| q \|} = \frac{\left(\sum_{i=1}^{n} p_i q_i\right)}{\sqrt{\sum_{i=1}^{n} p_i^2}\sqrt{\sum_{i=1}^{n} q_i^2}}$$



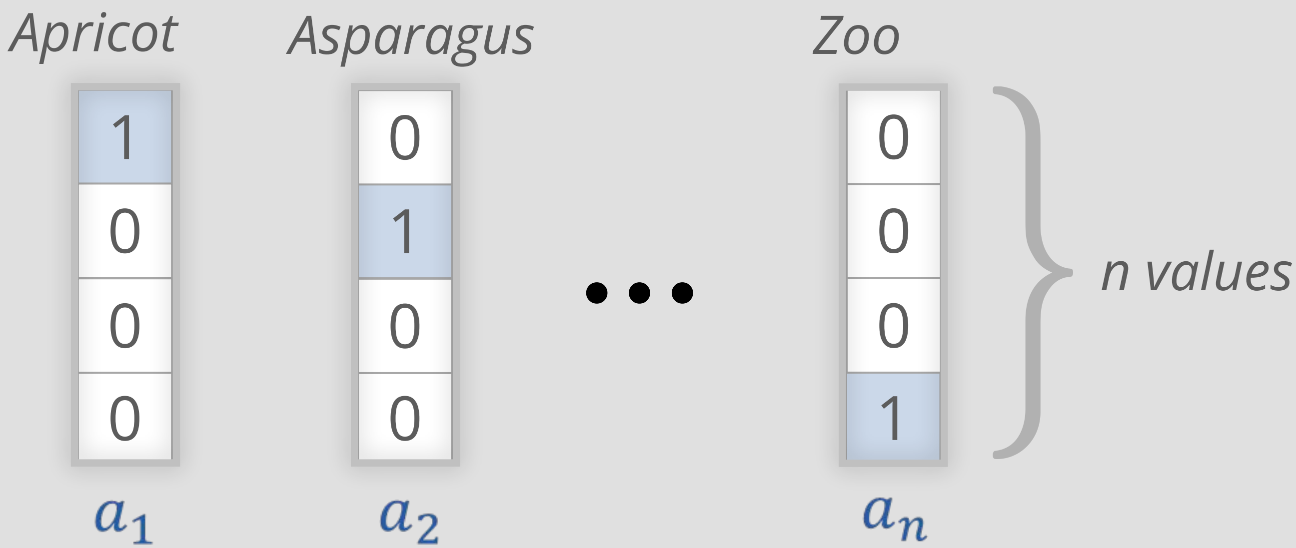
$\theta \approx 45°$
$\cos(\theta) \approx 0.7$



$\theta \approx 7°$
$\cos(\theta) \approx 0.99$

One-hot encoding

Suppose we have vocabulary V, |V| = n



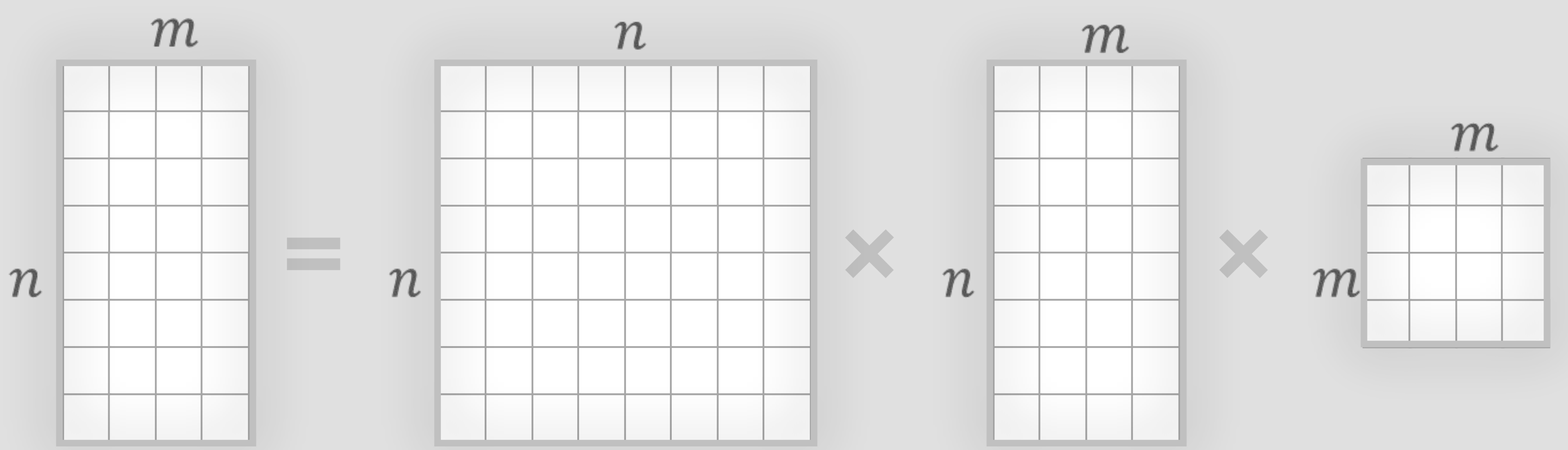
Advantages

- Simple way to obtain embeddings for a set of words

Disadvantages

- The documents will have huge unfixed length
- Embeddings are mutually orthogonal

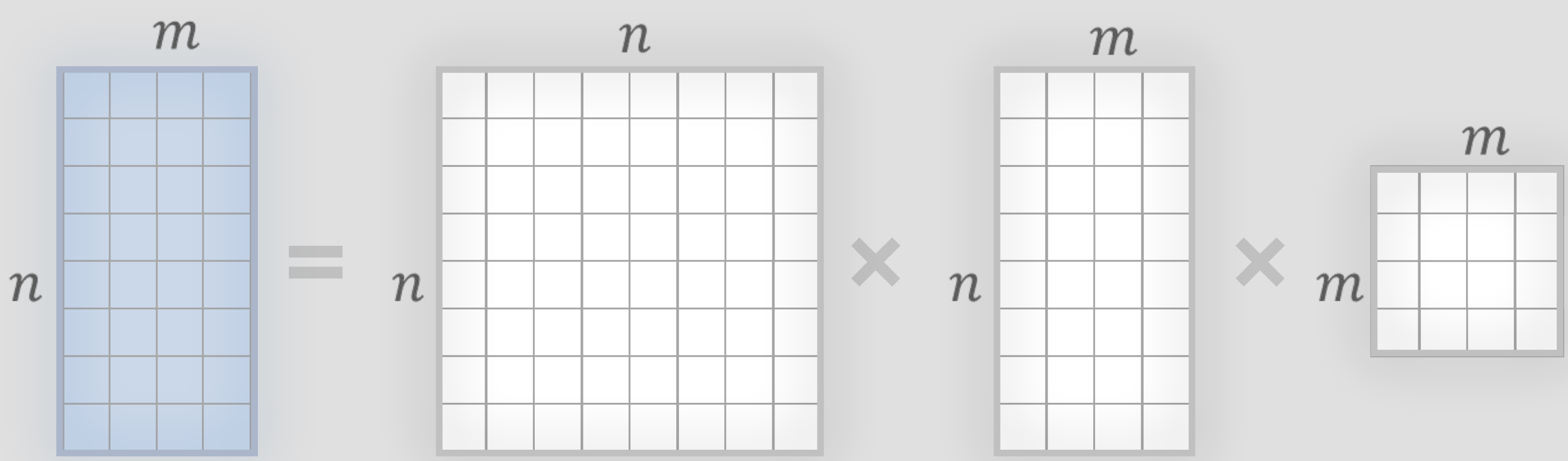
Singular Value Decomposition (SVD)



$\mathbf{M} \in \mathbb{R}^{n\times m}$ $\mathbf{U} \in \mathbb{R}^{n\times n}$ $\mathbf{\Sigma} \in \mathbb{R}^{n\times m}$ $\mathbf{V}^T \in \mathbb{R}^{m\times m}$

Singular Value Decomposition (SVD)

Suppose we have rectangular matrix $\mathbf{M} \in \mathbb{R}^{n\times m}$

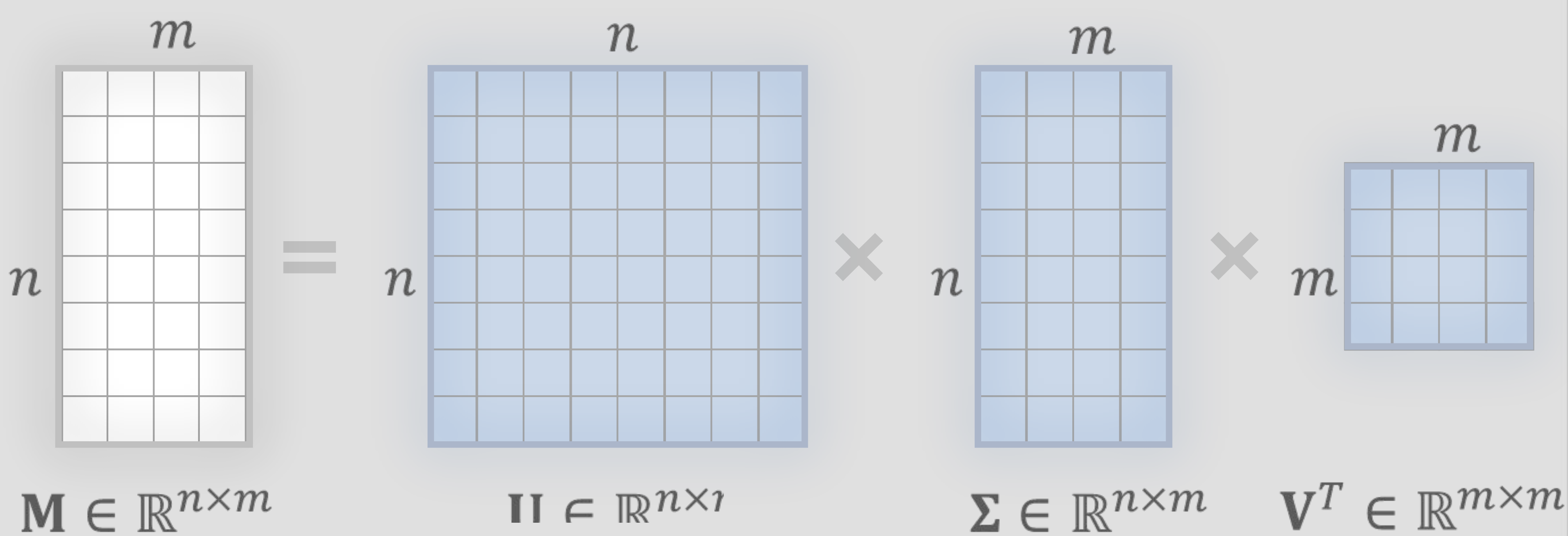


$\mathbf{M} \in \mathbb{R}^{n\times m}$ $\quad \mathbf{U} \in \mathbb{R}^{n\times n}$ $\quad \boldsymbol{\Sigma} \in \mathbb{R}^{n\times m}$ $\quad \mathbf{V}^T \in \mathbb{R}^{m\times m}$

Singular Value Decomposition (SVD)

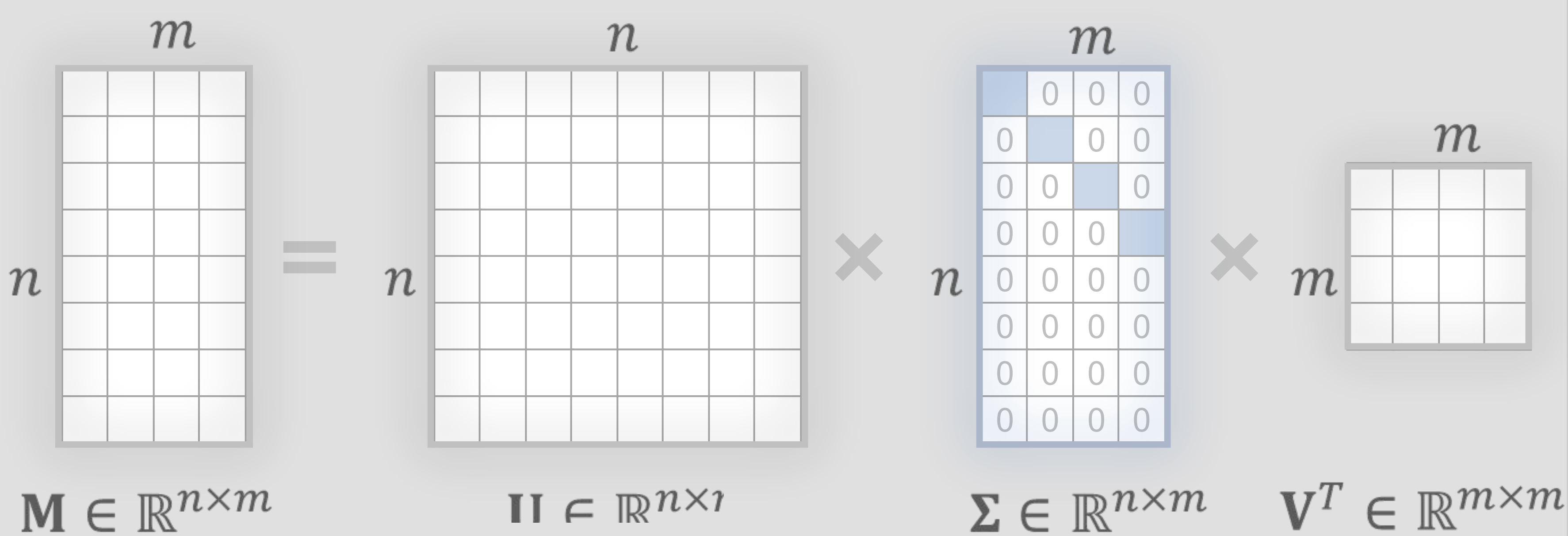
Suppose we have rectangular matrix $\mathbf{M} \in \mathbb{R}^{n \times m}$

The matrix can be represented as $\mathbf{M} = \mathbf{U\Sigma V}^T$



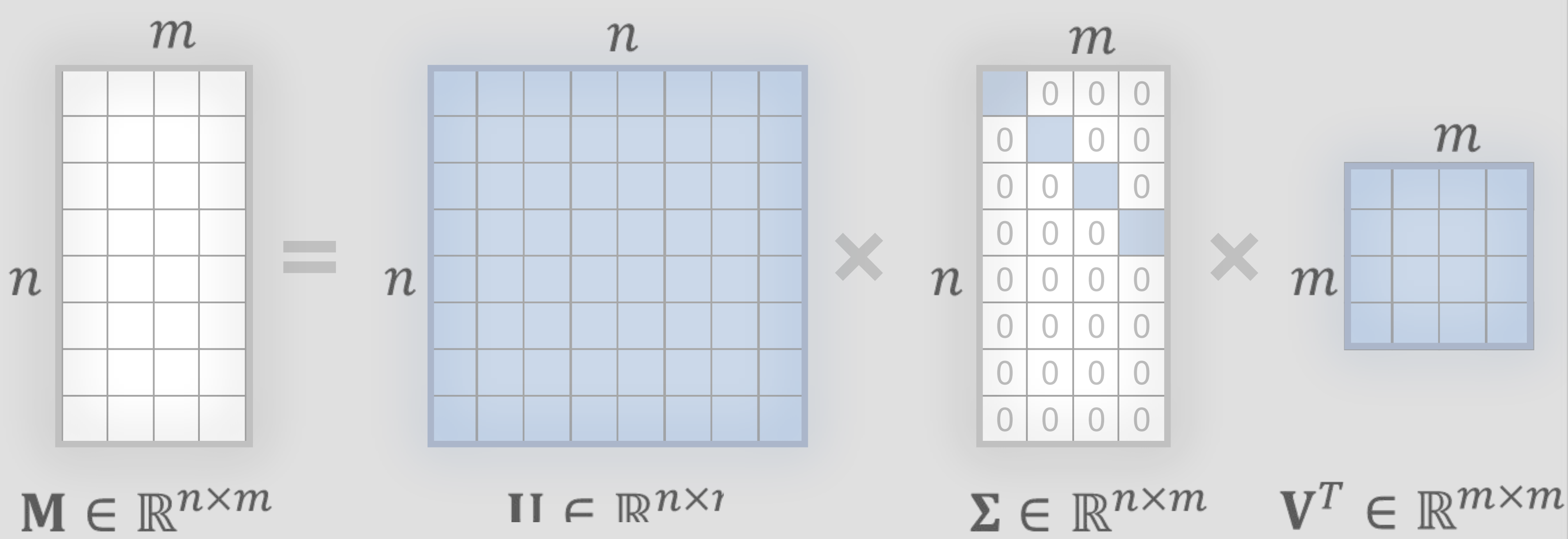
Singular Value Decomposition (SVD)

- $\boldsymbol{\Sigma} \in \mathbb{R}^{n\times m}$ is a diagonal matrix with non-negative values
- diagonal values of matrix $\boldsymbol{\Sigma}$ are singular values of matrix $\mathbf{M}$



Singular Value Decomposition (SVD)

Columns U and V are left and right singular vectors of matrix M

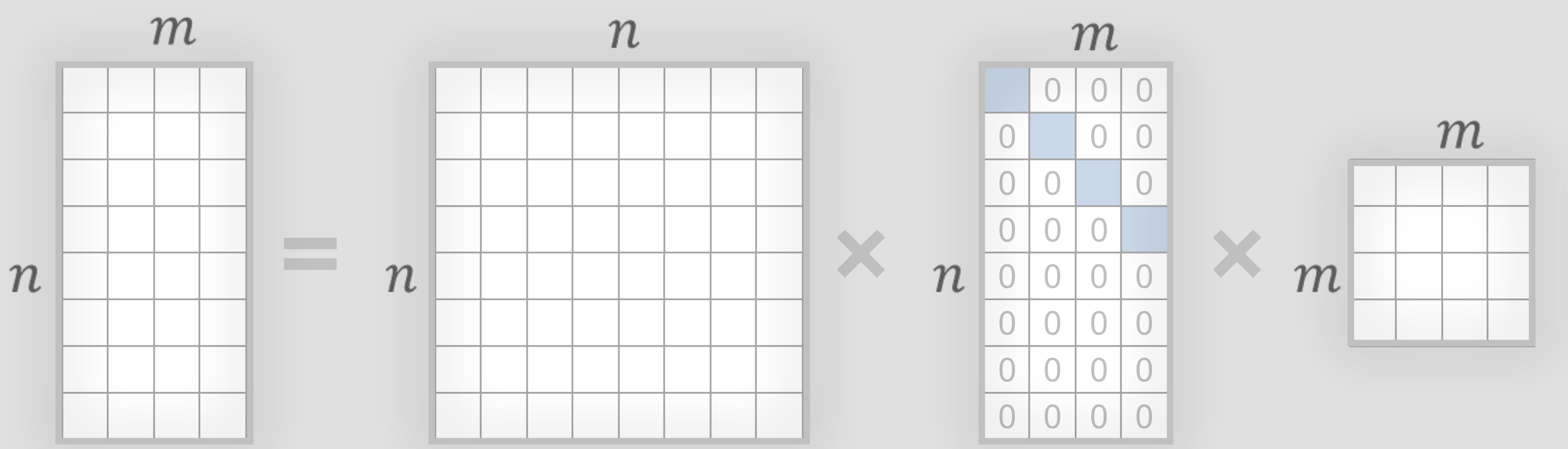


Singular Value Decomposition (SVD)

Suppose we a collection from m documents with n unique words

Then $\mathbf{M} \in \mathbb{R}^{n \times m}$ is a bag of words matrix

Let's apply SVD: $\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$

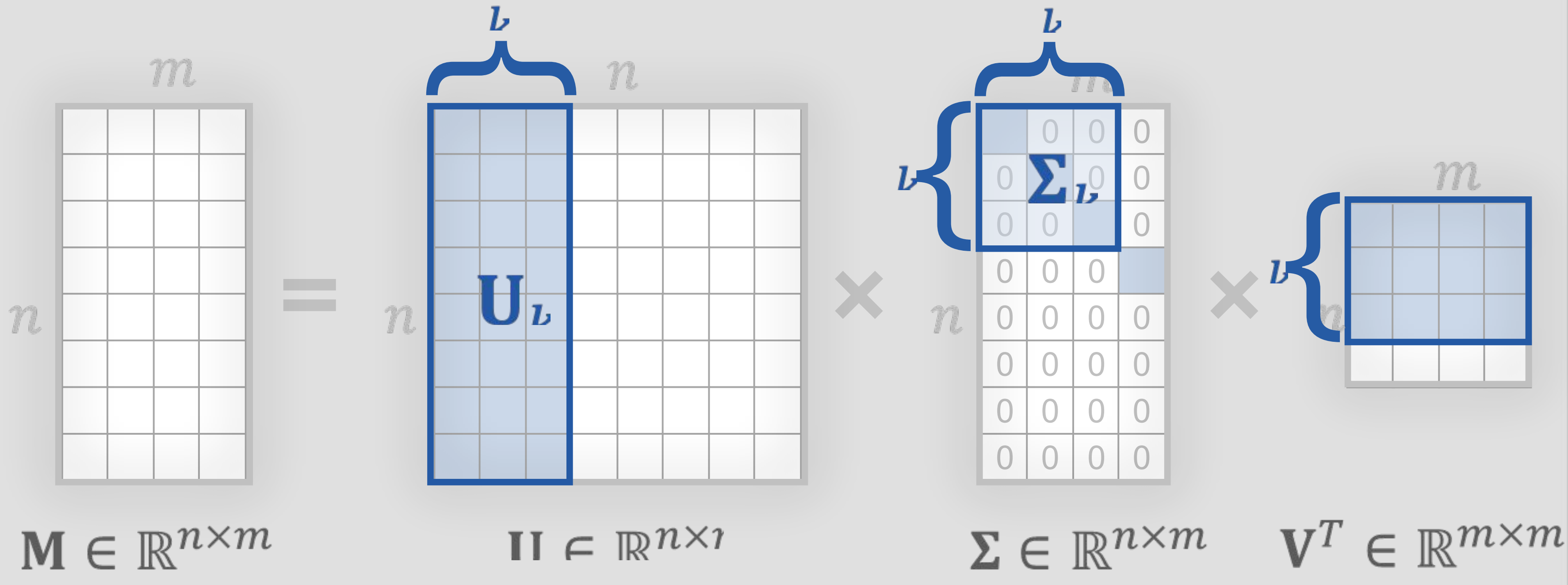


$\mathbf{M} \in \mathbb{R}^{n \times m}$ $\mathbf{U} \in \mathbb{R}^{n \times n}$ $\mathbf{\Sigma} \in \mathbb{R}^{n \times m}$ $\mathbf{V}^T \in \mathbb{R}^{m \times m}$

Singular Value Decomposition (reduced SVD)

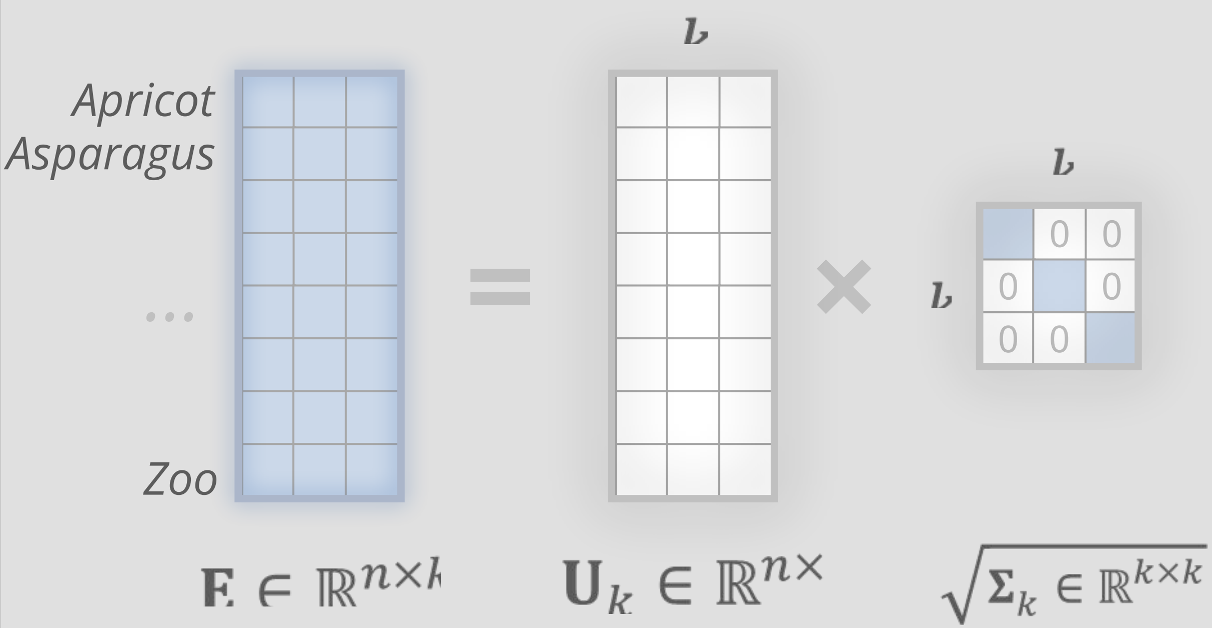
Pick top-k largest values of $\mathbf{\Sigma}$

Ignore all other values and columns of matrix U - we obtain matrices $\mathbf{U}_k$ and $\mathbf{\Sigma}_k$



Word embeddings with reduced SVD

We can use rows of matrix $\mathbf{U}_k\sqrt{\mathbf{\Sigma}_k}$ as word embeddings



By the way:

для получения векторов раскладывать можно не только матрицу «мешка слов»

Word embeddings with SVD

Improvements:

- Vectors have fixed size
- Vectors are no longer mutually orthogonal
- Semantic closeness is somehow taken into account

Nevertheless:

- Adding new words/documents requires new SVD calculation
- We need to operate with a huge BoW matrix
- Word embeddings are not that good

Main conclusions

- Word embeddings are used as features in NLP tasks
- Good word embeddings represent semantic closeness of words
- One-hot vectors can be useful but they are too sparse and mutually orthogonal
- SVD can produce word embeddings of fixed size that somehow represent semantics

Word2vec model

Distributive semantics hypothesis

- Words with similar meaning share similar context

«Today I ate tasty, juicy orange»

«This apple is so sweet and juicy»

«So sweet are the apricots, so tasty»

- Instead of frequency counters let's train a model to predict a word by its context (and vice versa)

- *Harris Zellig. Distributional structure // Word. — 1954. — Vol. 10, no. 23. — Pp. 146–162.*

Word2vec models: CBOW and skip-gram

- Continues Bag-of-Words

  predict central word by its context

I share **apricot** with friend

- Skip-gram

  predict context by central word

I share **apricot** with friend

- *Distributed Representations of Words and Phrases and their Compositionality. / Tomas Mikolov, Ilya Sutskever, Kai Chen et al. // NeurIPS — 2013. — Pp. 3111–3119.*

CBOW model

Suppose we have a collection with N unique words

To train a model we slide over text with a window of size 2C + 1

I	share	an	apricot	with	friend

step 1

CBOW model

I	share	an	apricot	with	friend

step 2

CBOW model

I	share	an	apricot	with	friend

step 3

CBOW model

I	share	an	apricot	with	friend

step
4

CBOW model

I	shape	an	apricot	with	friend

step 5

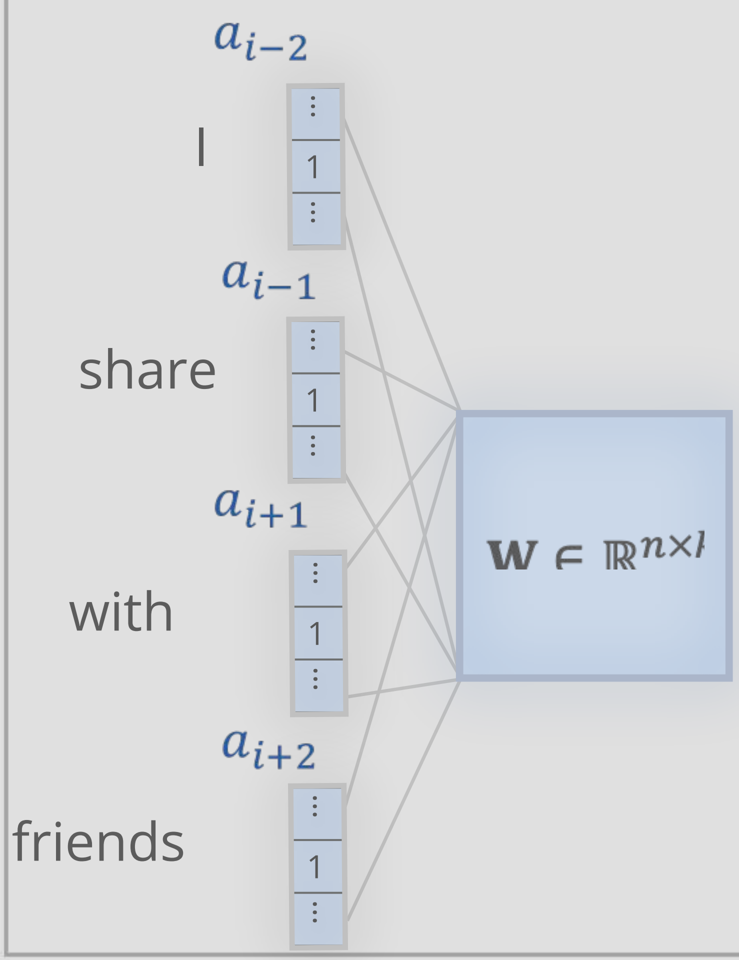
CBOW model

I	share	an	apricot	with	friend

step 6

CBOW as neural network

Model - two-layer neural network
Input - 2C one-hot context vectors of size n

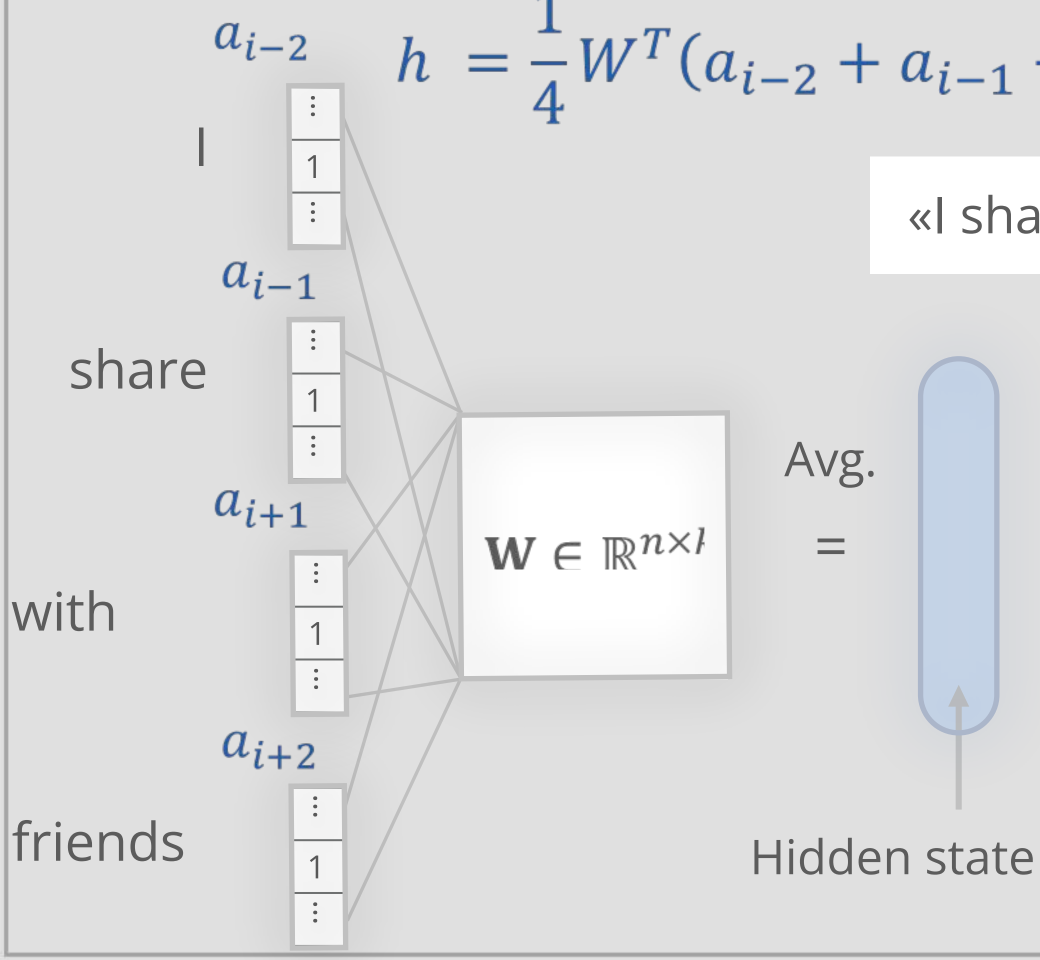


«I share **apricots** with friends»

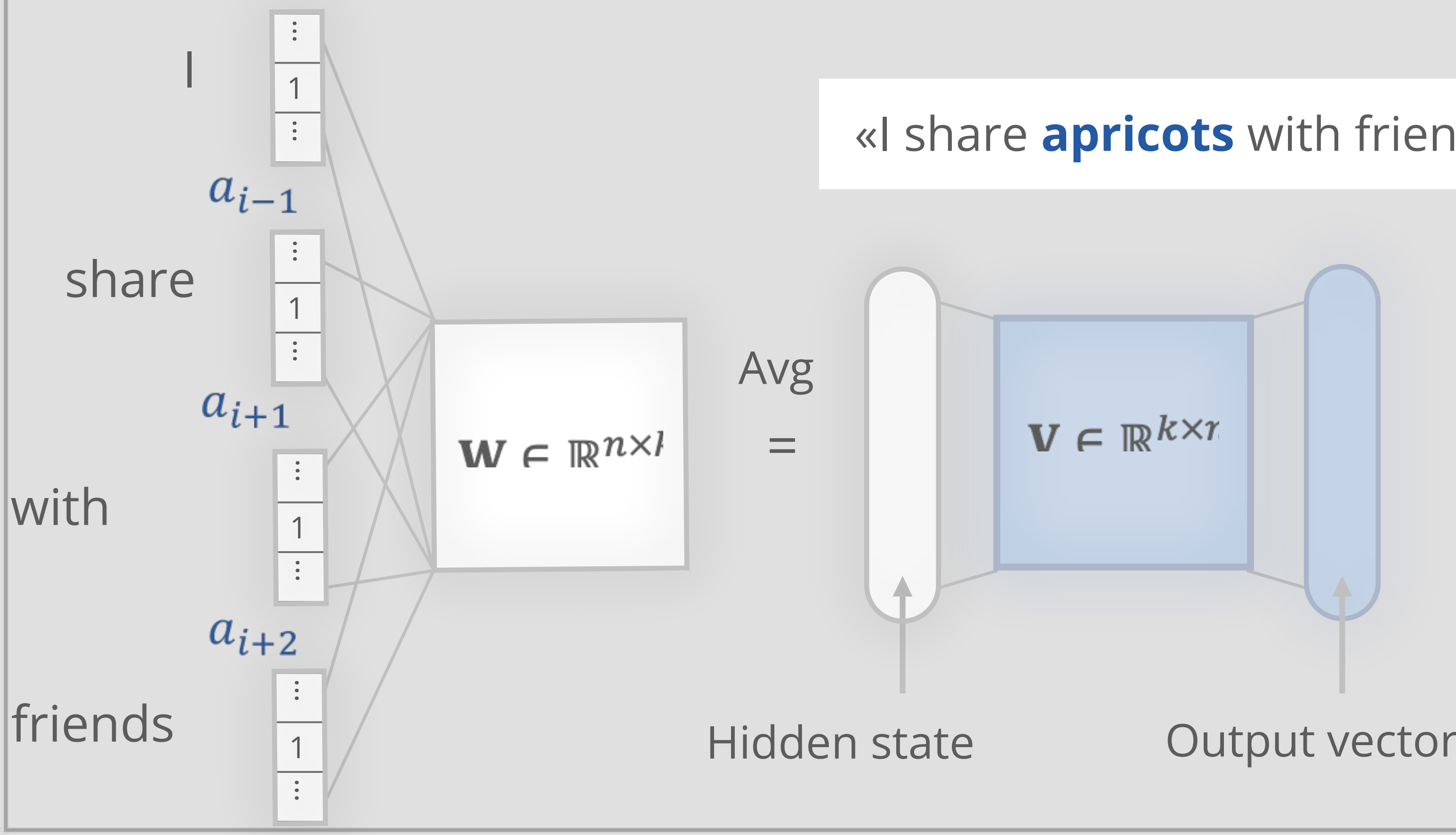
CBOW as neural network

$$h = \frac{1}{4} W^T (a_{i-2} + a_{i-1} + a_{i+1} + a_{i+2})$$

«I share **apricots** with friends»



CBOW as neural network



Loss function

- For window with index i predict word w_i by context c_i

$$\sum_{i=1}^{N} \log p(w_i|c_i) \rightarrow \max_{W,V}$$

Loss function

- For window with index i predict word w_i by context c_i

$$\sum_{i=1}^{N} \log p(w_i|c_i) \to \max_{W,V}$$

- Therefore model's output is a vector with n probabilities

Loss function

- For window with index i predict word by context

$$\sum_{i=1}^{N} \log p(w_i|c_i) \to \max_{W,V}$$

- Therefore model's output is a vector with n probabilities

- Therefore model's output is a vector with n probabilities

Softmax function:

$$softmax(b) = (z_1, \ldots, z_n), \qquad z_j = p(w_j|c_i) = \frac{e^{b_j}}{\sum_{k=1}^{n} e^{b_k}}$$

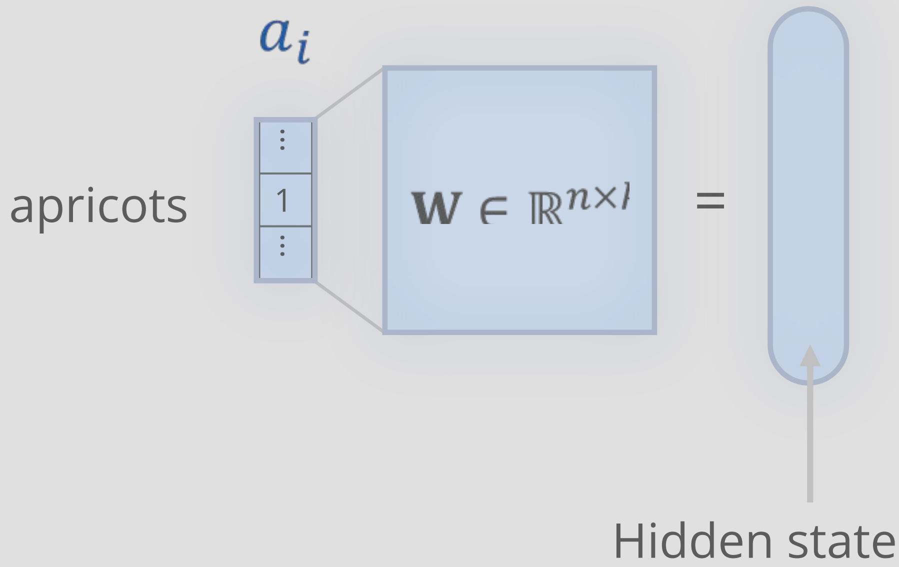
And the word embeddings?

- As a result of training we obtain two matrices: W and V
- Usually, rows of matrix W are used as word embeddings
- But both columns of V and the combination of two matrices can be used

Skip-gram as neural network

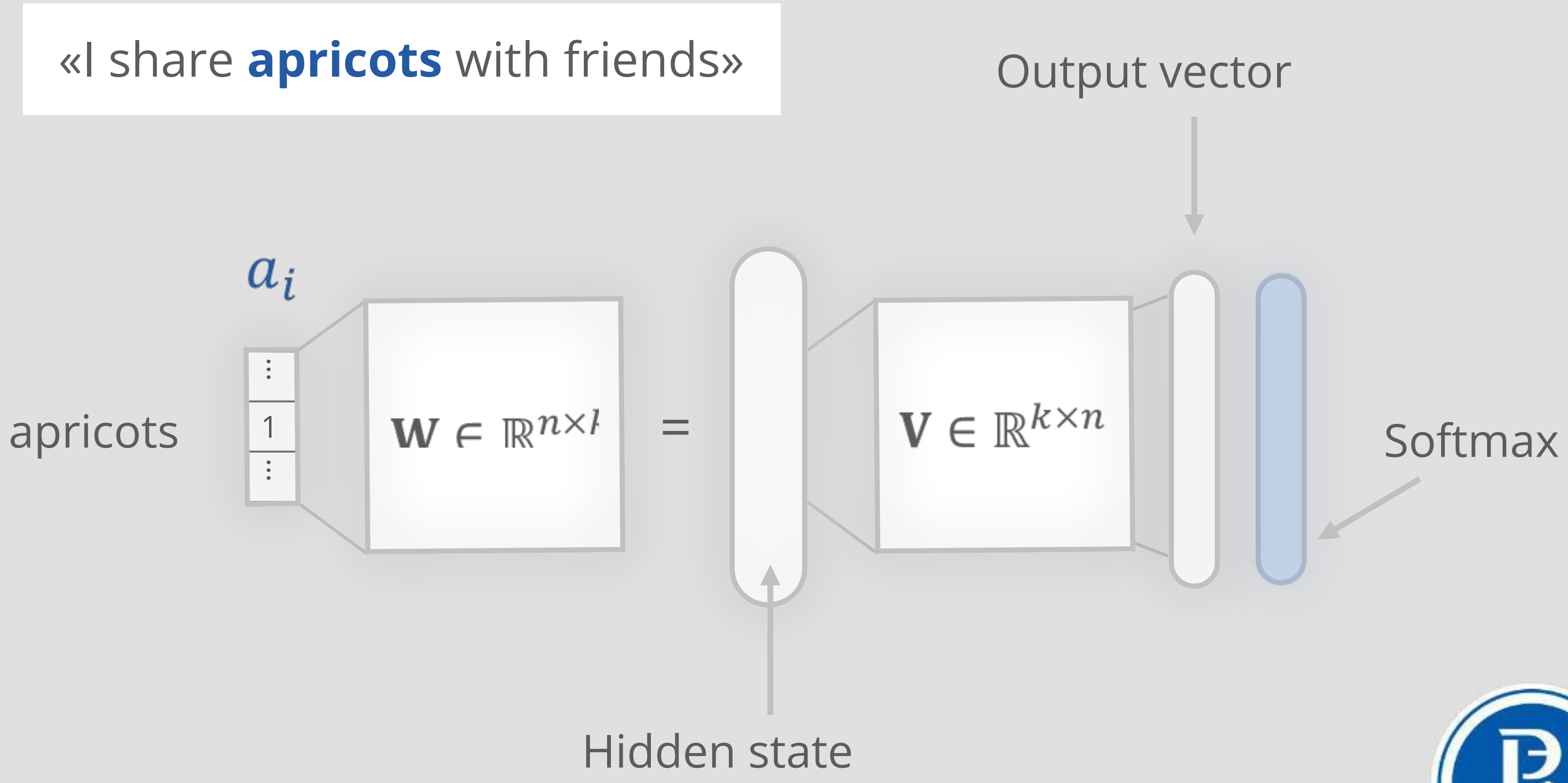
- Skip-gram model is arranged symmetrically

«I share **apricots** with friends»



Skip-gram as neural network

- Output — one probability distribution over words for the central word



Loss function

- Output — one probability distribution over words for the central word
- 2C words from actual context should have maximal values
- Loss function:

$$\sum_{i=1}^{N} \sum_{j=-C, j\neq 0}^{C} \log p(w_{i+j}|w_i) \to \max_{W,V}$$

Main conclusions

- Word2vec models train word representations based on predictions, not on statistics
- There are two basic models: CBOW and Skip-gram
- In canonical implementation word2vec is a two-layer neural network, and its weights are the resulting word embeddings
- The quality of word2vec embeddings is better then SVD embeddings, and we don’t need huge BoW matrices anymore

Word2vec training optimization: hierarchical softmax and SGNS

Word2vec neural approach disadvantages

- Training word2vec neural network is computationally difficult
- We need to calculate softmax *(O(n))* and update a lot of parameters

Word2vec neural approach disadvantages

- Training word2vec neural network is computationally difficult
- We need to calculate softmax *(O(n))* and update a lot of parameters
- In practice, word2vec is trained with optimization methods

Hierarchical softmax

- We still have a fully connected neural network
- The only thing that differs is softmax calculation

- *Mnih Andriy, Hinton Geoffrey E. A Scalable Hierarchical Distributed Language Model // NeurIPS. — 2008.*

Hierarchical softmax

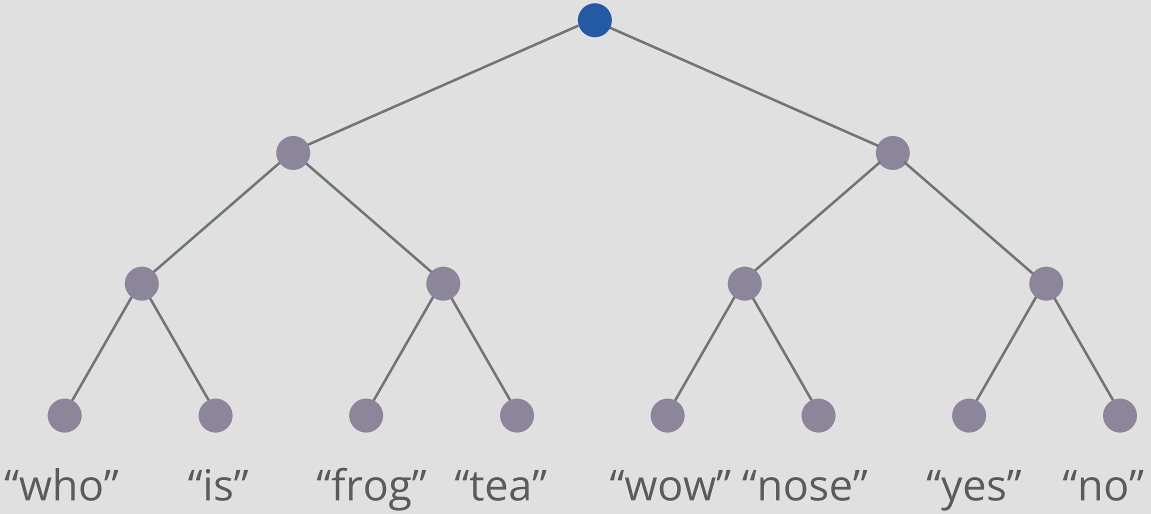
- We still have a fully connected neural network
- The only thing that differs is softmax calculation
- To calculate loss we don't need the whole vector of probabilities
- We only need the values in positions of words in the context window:

$$\sum_{i=1}^{N} \sum_{j=-C, j\neq 0}^{C} \log p(w_{i+j}|w_i) \rightarrow \max_{W,V}$$

Hierarchical softmax

- Let's consider skip-gram model
- Hidden state after first layer $h = W^T x$
- Let's change the second layer

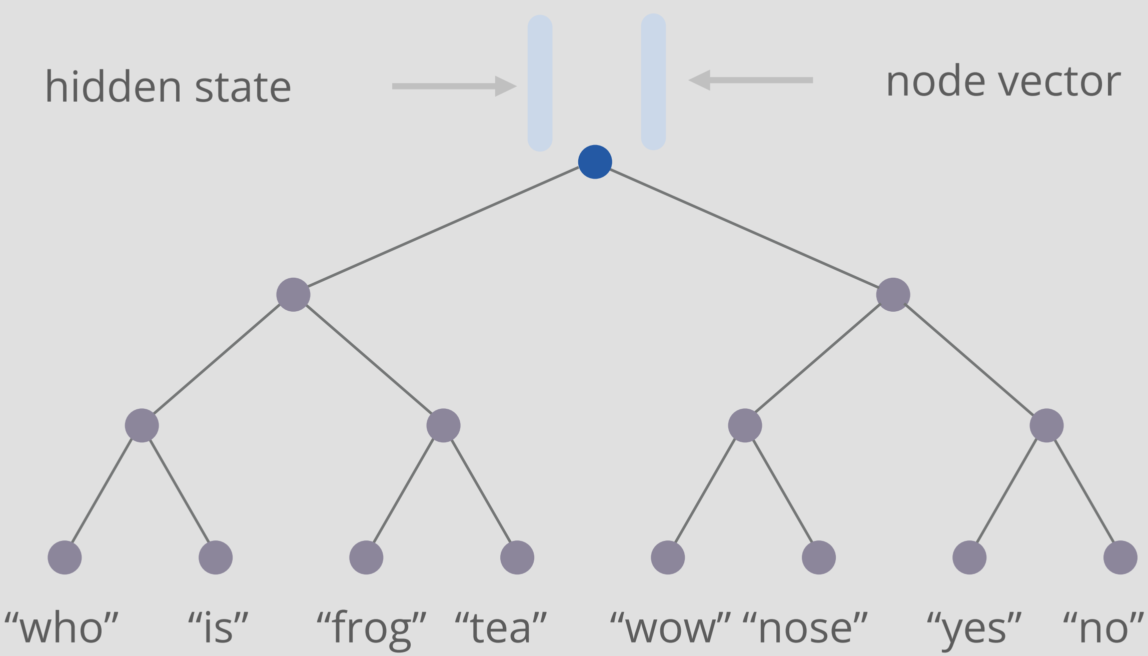
  into a binary tree (for example, Huffman tree)
- Assign every leaf one word from vocabulary
- Assign every internal node a vector of k weights



Hierarchical softmax

- Suppose we want to obtain a probability of a word w = “tea”
- Probability of paths (left and right) in current node:

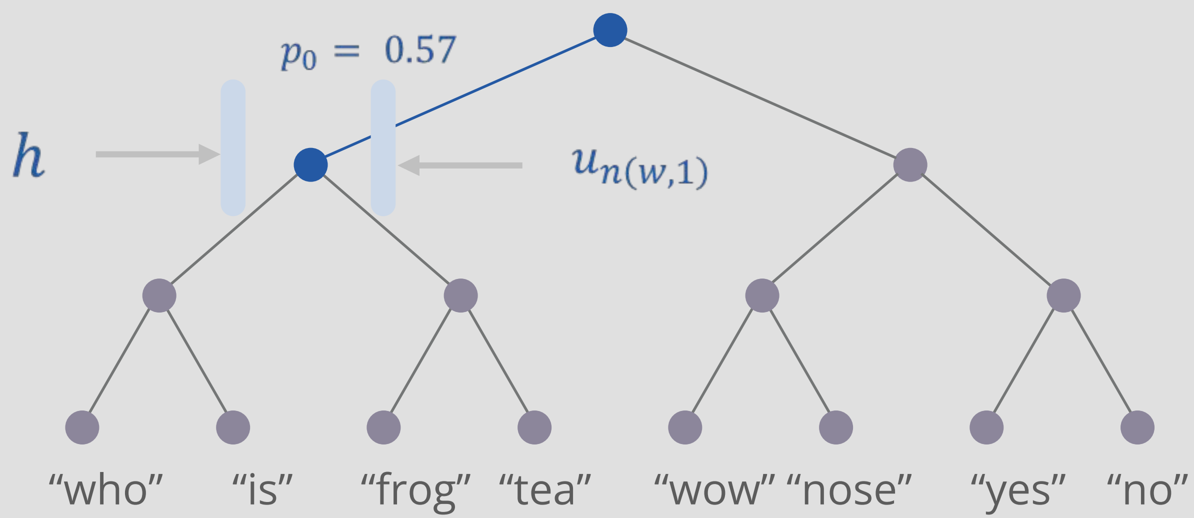
$$p_0 = \sigma\left(u_{n(w,0)}^T h\right) \qquad p_0 = \sigma\left(-u_{n(w,0)}^T h\right)$$



Hierarchical softmax

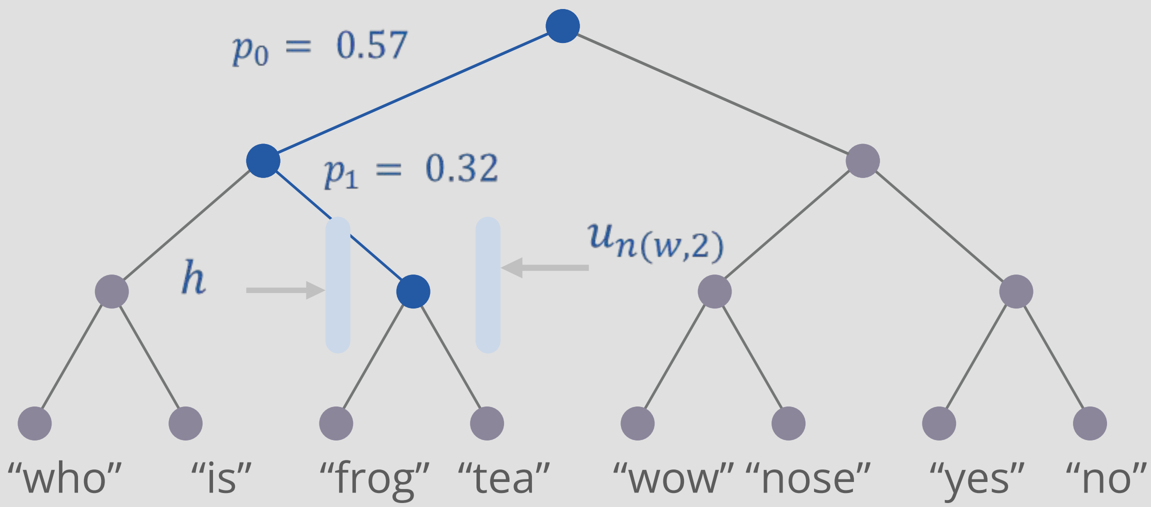
- Do the same in the next node:

$$p_1 = \sigma(-u_{n(w,1)}^T h)$$



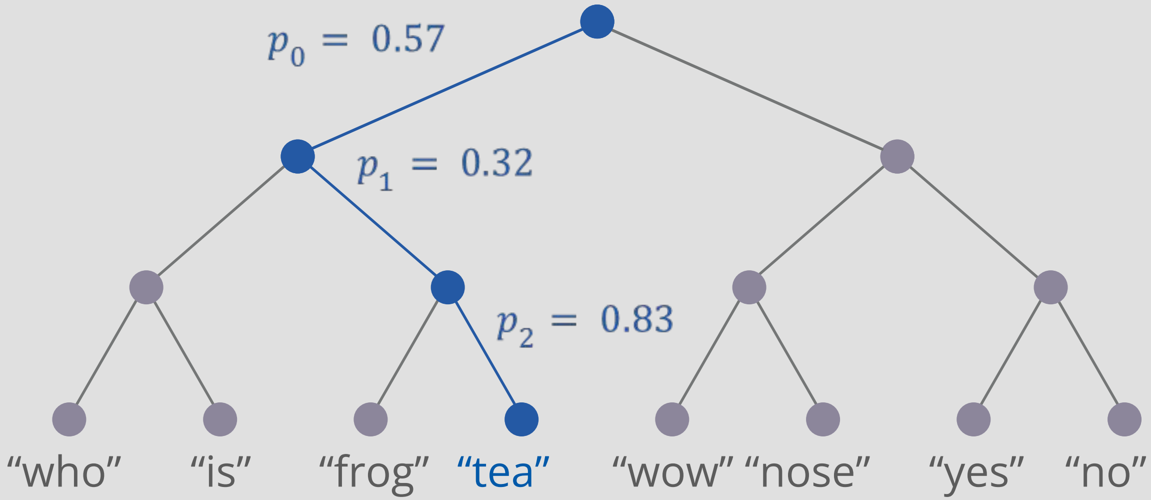
Hierarchical softmax

$$p_2 = \sigma(-u_{n(w,2)}^T h)$$



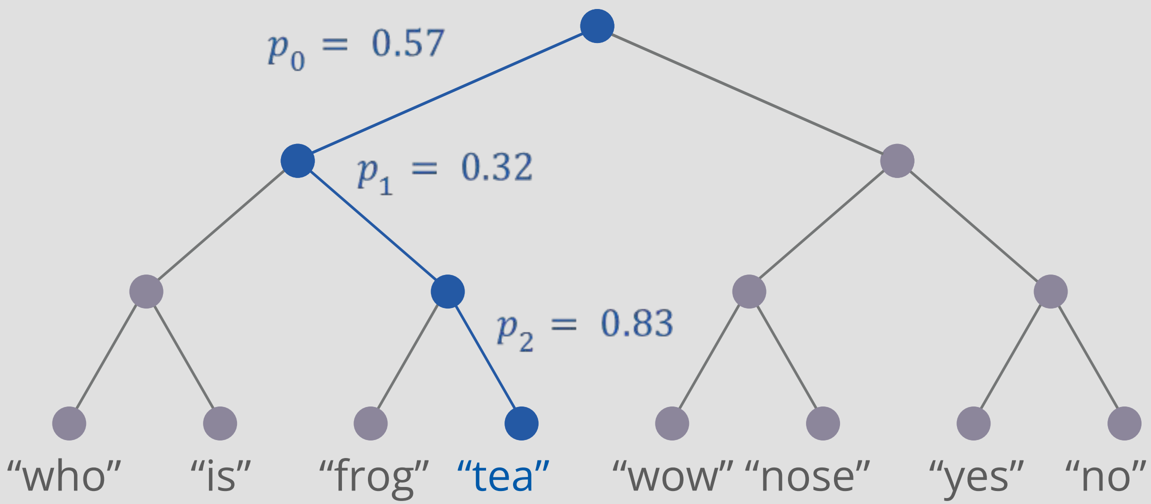
Hierarchical softmax

- In the end, we are at leaf with the word tea
- Every step i had a probability p_i



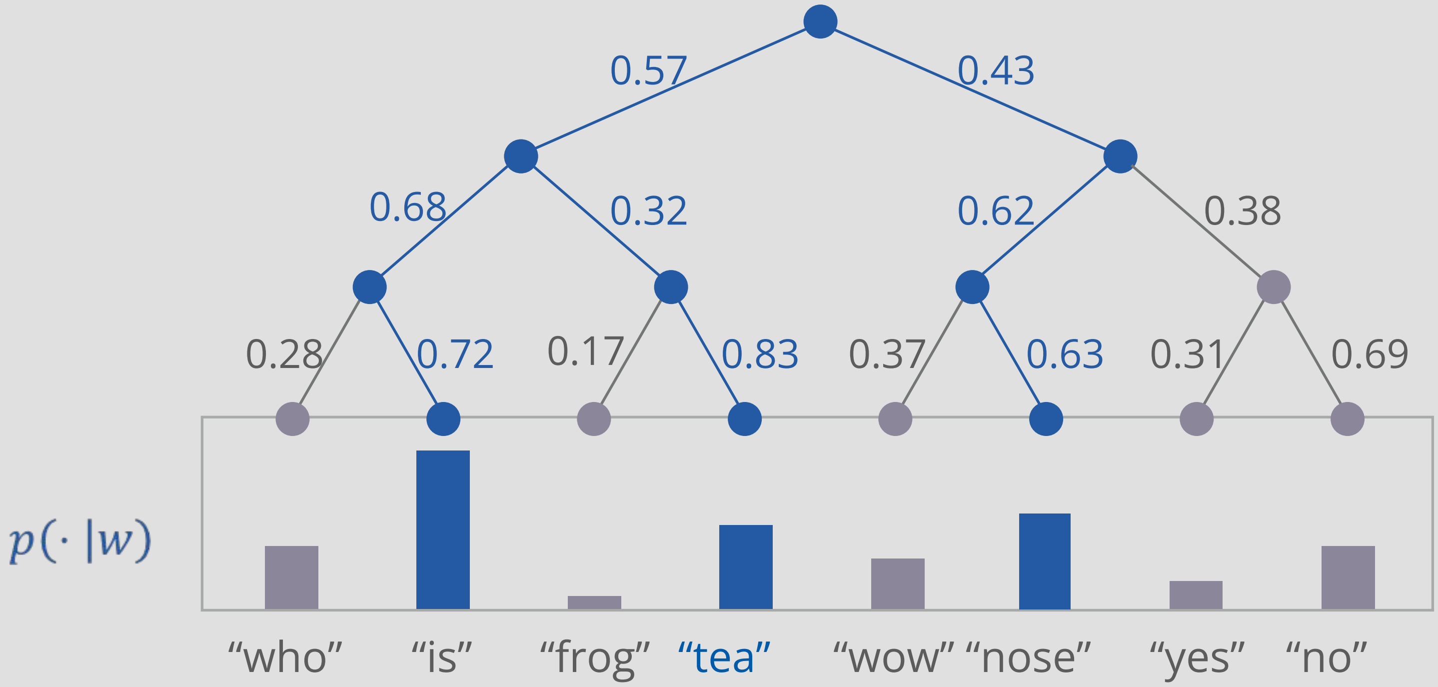
Hierarchical softmax

- In the end, we are at leaf with the word tea
- Every step i had a probability p_i
- The final probability of path is $\prod_i p_i$



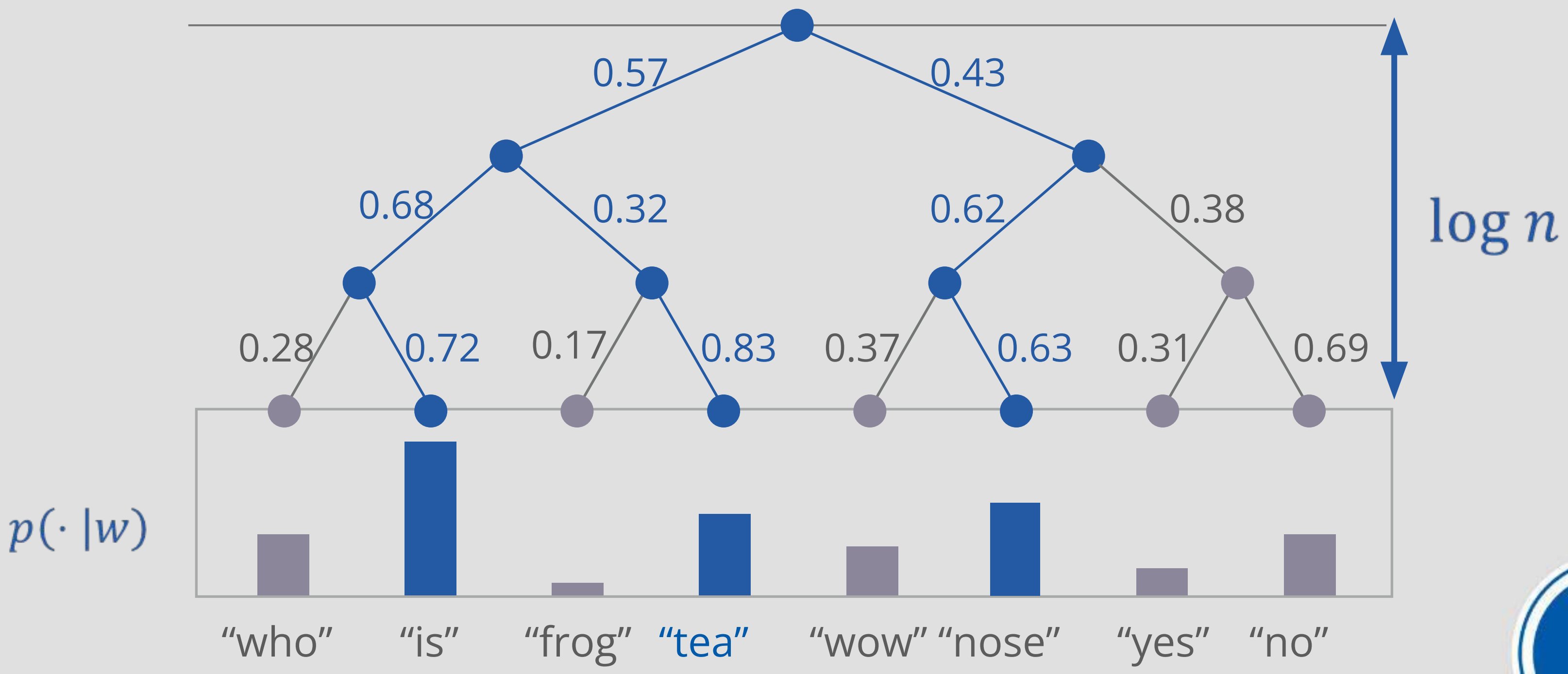
Hierarchical softmax

- If we do the procedure 2C times we obtain all probabilities required to calculate loss



Hierarchical softmax

- If we do the procedure 2C times we obtain all probabilities required to calculate loss
- The complexity of current calculation - O(log n)



All in all

Before:

- Obtain h from the first layer
- Multiply h with second layer matrix V
- Apply softmax
- Use only 2C probabilities from the softmax result

All in all

Before:

- Obtain h from the first layer
- Multiply h with second layer matrix V
- Apply softmax
- Use only 2C probabilities from the softmax result

Now:

- Obtain h from the first layer
- 2C times go through the tree
- Obtain only essential probabilities

Skip-gram negative sampling

- For skip-gram model there is another popular training optimization method

Skip-gram negative sampling

- For skip-gram model there is another popular training optimization method
- We change the formulation of the problem and loss function
- We will solve binary classification problem

Skip-gram negative sampling

- For skip-gram model there is another popular training optimization method
- We change the formulation of the problem and loss function
- We will solve binary classification problem

- Object - pair of words (w, s)
- Class 1: word s belongs to context w
- Class 2: word s doesn’t belong to context w
- For each word w we compare trainable vector v_w which will be the sought one

What are the advantages?

- Training the model for each input object requires an update of all weights of the input layer
- Softmax in classic approach leads to an update of all weights in all layers
- In new approach we only update the weights of the layers, that are involved in the current iteration of training

Skip-gram negative sampling

- Class probability is simulated with sigmoid:

$$p(1|(w,s)) = \frac{1}{1+\exp(-v_w^T v_s)} = \sigma(v_w^T v_s)$$

Skip-gram negative sampling

- Class probability is simulated with sigmoid:

$$p(1|(w,s)) = \frac{1}{1+\exp(-v_w^T v_s)} = \sigma(v_w^T v_s)$$

- Let D_1 be a subset of pairs (w, s) where s belongs to context w
- Let D_2 be a subset of all other possible pairs
- Then the likelihood function is:

$$L = \sum_{(w,s)\in D_1} \log(\sigma(v_w^T v_s)) + \sum_{(w,s)\in D_2} \log(\sigma(-v_w^T v_s))$$

Skip-gram negative sampling

If we optimize this function, we solve the task!

Skip-gram negative sampling

If we optimize this function, we solve the task!

But:

- We need examples of pairs
- D_1 can be obtained from data, while D_2 is not presented in data as is

Skip-gram negative sampling

If we optimize this function, we solve the task!

But:

- We need examples of pairs
- D_1 can be obtained from data, while D_2 is not presented in data as is

Solution: generate negative samples by sampling random pairs of words on each training step

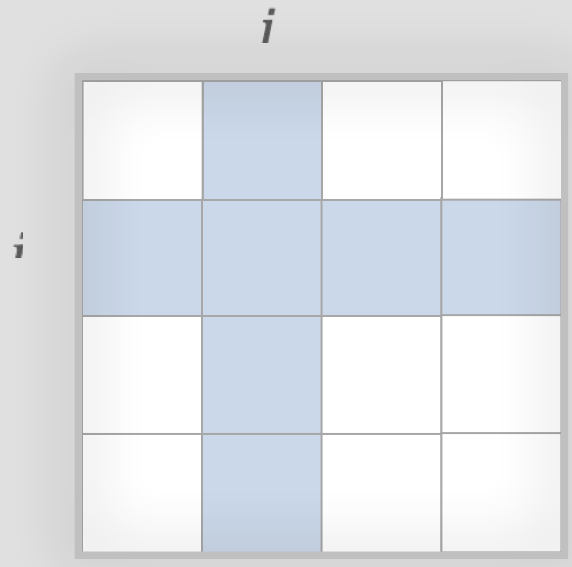
Main conclusions

- Canonical word2vec implementation is scales poorly on dictionary and corpus volume
- Main difficulty is the second layer and softmax calculation
- There are several training optimization methods, the main ones are hierarchical softmax and negative sampling

GloVe model

GloVe (Global Vectors)

- Construct a matrix $\mathbf{X} \in \mathbb{R}^{n\times n}$
- Every column and row correspond to a word from a dictionary

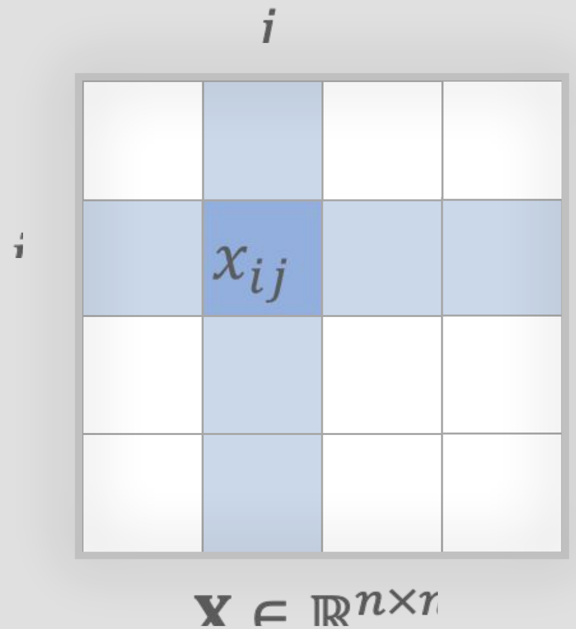


$\mathbf{X} \in \mathbb{R}^{n\times n}$

- *Pennington Jeffrey, Socher Richard, Manning Christopher D. Glove: Global Vectors for Word Representation. // EMNLP. — Vol. 14. — 2014. — Pp. 1532–1543.*

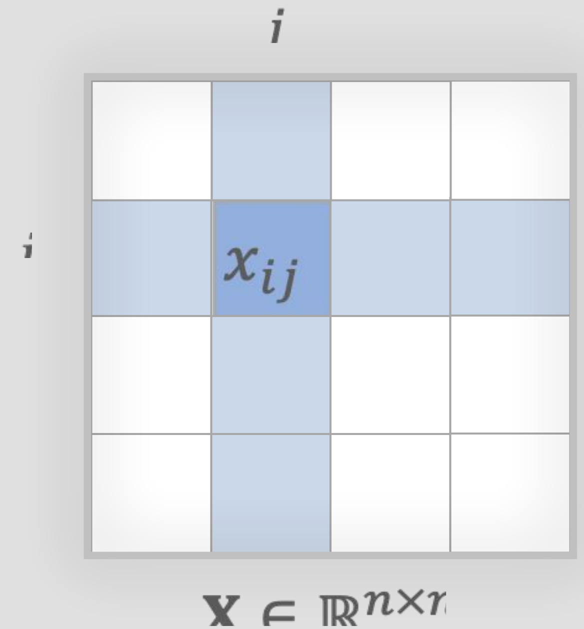
GloVe (Global Vectors)

- Construct a matrix $\mathbf{X} \in \mathbb{R}^{n \times n}$
- Every column and row correspond to a word from a dictionary
- x_{ij} - number of times word i occurs in context of word j



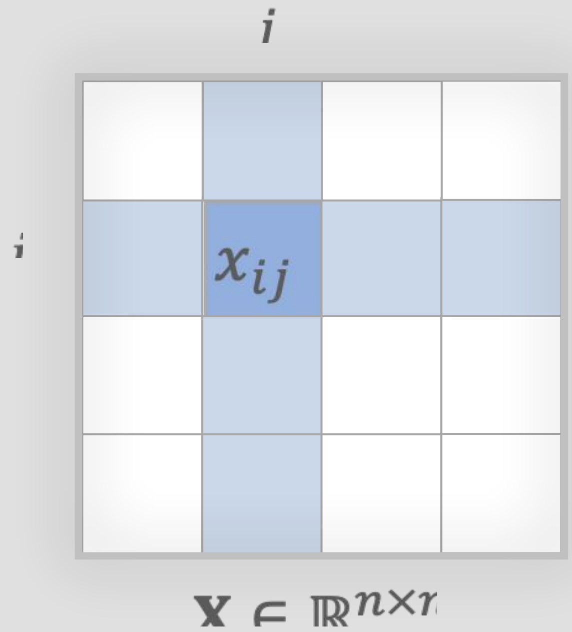
GloVe (Global Vectors)

- Construct a matrix $\mathbf{X} \in \mathbb{R}^{n \times n}$
- Every column and row correspond to a word from a dictionary
- x_{ij} - number of times word i occurs in context of word j
- $X_i = \sum_j x_{ij}$ - sum of elements of row i



GloVe (Global Vectors)

- Construct a matrix $\mathbf{X} \in \mathbb{R}^{n \times n}$
- Every column and row correspond to a word from a dictionary
- x_{ij} - number of times word i occurs in context of word j
- $X_i = \sum_j x_{ij}$ - sum of elements of row i
- $\mathbf{P}_{ij} = \frac{x_{ij}}{X_i}$ - probability of word j to occur in context of word i



GloVe (Global Vectors)

- Assume we know vector representation v_i for every word i
- Also assume that we know all x_{ij}

GloVe (Global Vectors)

- Assume we know vector representation v_i for every word i
- Also assume that we know all x_{ij}
- Define function $F\left(f(v_i, v_j, v_k)\right) = \frac{P_{ik}}{P_{jk}}$
- This function F shows which one of words i and j is more likely to occur in context of word k
- Function f is some function from input to real number

GloVe (Global Vectors)

- We need F to satisfy

$$F\left((v_i - v_j)^T v_k\right) = \frac{F(v_i^T v_k)}{F(v_j^T v_k)} = \frac{P_{ik}}{P_{ij}}$$

- F can be exp(x)

GloVe (Global Vectors)

- So

$$F(x) = \exp(x)$$

$$P_{ij} = \frac{x_{ij}}{X_i}$$

GloVe (Global Vectors)

- So

$$F(x) = \exp(x)$$
$$P_{ij} = \frac{x_{ij}}{X_i}$$

- Rewrite

$$F(v_i^T v_k) = P_{ik}$$
$$v_i^T v_k = \log(x_{ik}) - \log(X_i)$$

GloVe (Global Vectors)

- So

$$F(x) = \exp(x)$$
$$P_{ij} = \frac{x_{ij}}{X_i}$$

- Rewrite

$$F(v_i^T v_k) = P_{ik}$$
$$v_i^T v_k = \log(x_{ik}) - \log(X_i)$$

- Now we remember that we only have x_{ij}

GloVe (Global Vectors)

- we want

$$v_i^T v_k = \log(x_{ik}) - \log(X_i)$$

- Therefore we rewrite

$$\sum_i \sum_k F(x_{ik}) \left(v_i^T v_k + b_i + b_k - \log(x_{ik})\right)^2 \to \min_{v_i, b_i,\, i \in \{1,n\}},$$

$$\log(X_i) = b_i + b_k$$

and obtain word embeddings.

Main conclusions

- Another example on an approach based on frequencies
- In practice, it works quite similar to word2vec
- There are many pre-trained GloVe models

FastText model, Hashing Trick

Word2vec and GloVe problems:

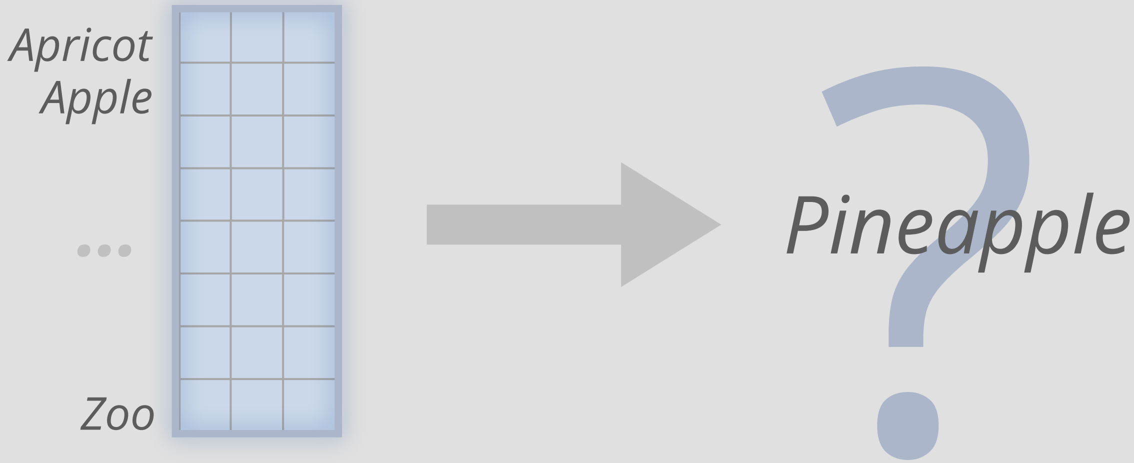
- Problem 1: Out-of-Vocabulary (OOV) words

Word2vec and GloVe problems:

- Problem 1: Out-of-Vocabulary (OOV) words
  We train our model on corpus V
  Then we try to obtain a vector for a new word w
  But there is no embedding for it

Word2vec and GloVe problems:

- Problem 1: Out-of-Vocabulary (OOV) words
  We train our model on corpus V
  Then we try to obtain a vector for a new word w
  But there is no embedding for it



Word2vec and GloVe problems:

- Problem 2: Lack of consideration of morphology
- Suppose we train a model for a language with rich morphology (for example, russian):

Word2vec and GloVe problems:

- Problem 2: Lack of consideration of morphology
- Suppose we train a model for a language with rich morphology (for example, russian):

- We get a new embedding for each word
- As a result:
  - Many similar embeddings (and redundant memory)
  - Less samples for each word in training data

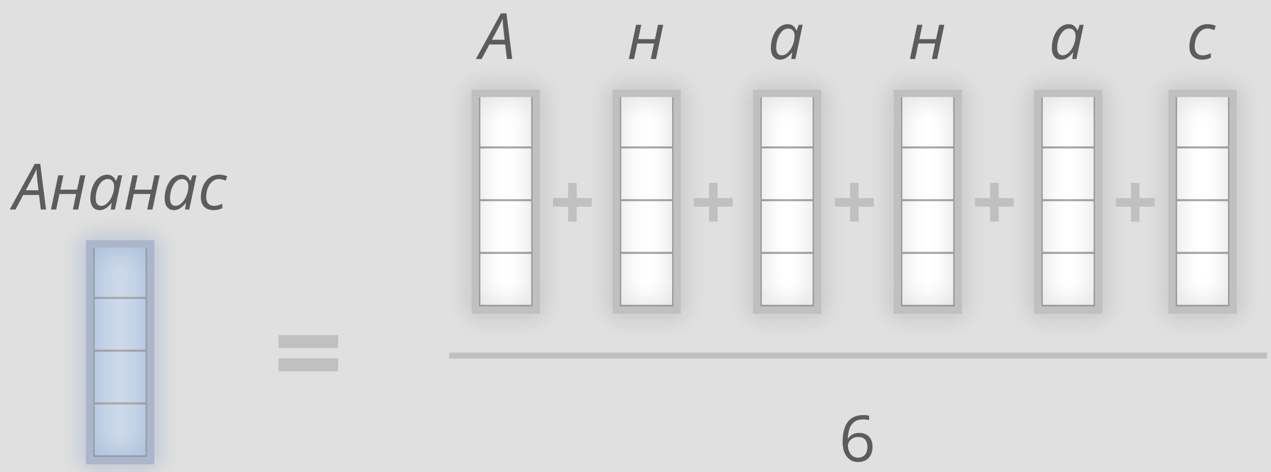
яблоко
яблока
яблоку
...
яблоками
яблоках

Character embeddings

- We can try to get embeddings for a smaller piece of language
- For example, for each character
- All training methods remain the same
- Word embedding can be obtained by averaging character embedding

Character embeddings

- We can try to get embeddings for a smaller piece of language
- For example, for each character
- All training methods remain the same
- Word embedding can be obtained by averaging character embedding



Character n-grams embeddings

- Instead of characters we can use character n-grams
- It improves the quality of the resulting word embeddings significantly

Character n-grams embeddings

- Instead of characters we can use character n-grams
- It improves the quality of the resulting word embeddings significantly

N-grams for a word «doctor»

N=3: ^do, doc, oct, cto, tor, or$

N=4: *^doc, doct, octo, octor, tor$*

N=5: *^doct, docto, octor, ctor$*

Character n-grams embeddings

- Instead of characters we can use character n-grams
- It improves the quality of the resulting word embeddings significantly

N-grams for a word «doctor»

N=3: ^do, doc, oct, cto, tor, or$

N=4: *^doc, doct, octo, octor, tor$*

N=5: *^doct, docto, octor, ctor$*

- Word embeddings are obtained by averaging

Word2vec and GloVe problems

Improvements:

- OOV problem solved
- Morphology problems also solved

But:

- There can be even more sequences of characters than there are different variants of words
- Tens of millions of vectors may simply not fit into RAM

Hash-functions and hash-tables

- String hash-function converts a string into a number

Hash-functions and hash-tables

- String hash-function converts a string into a number
- Function requirements:
  - The range of values is limited by an interval
  - Function values are distributed approximately uniformly over the interval

Hash-functions and hash-tables

- String hash-function converts a string into a number
- Function requirements:
  - The range of values is limited by an interval
  - Function values are distributed approximately uniformly over the interval

- Hash-table — an array of values + hash-function that transforms input string into array's indices

Hashing Trick

- We fix the maximum number of vectors that we want to train

Hashing Trick

- We fix the maximum number of vectors that we want to train
- Match them a hash-table

Hashing Trick

- We fix the maximum number of vectors that we want to train
- Match them a hash-table
- All n-gramm char-embeddings are distributed over the values of array

Hashing Trick

- We fix the maximum number of vectors that we want to train
- Match them a hash-table
- All n-gramm char-embeddings are distributed over the values of array
- Several n-grams use the same embedding

FastText

- Package that trains word embeddings
- Uses CBOW / skip-gram both for words and character n-grams
- Optimizes RAM consumption by hashing trick
- Parallels training process on CPU

- *Enriching Word Vectors with Subword Information / Piotr Bojanowski, Edouard Grave, Armand Joulin, Tomas Mikolov // Transactions of the Association for Computational Linguistics. — 2017. — Vol. 5. — Pp. 135–146.*

Main conclusions

- Classic word2vec models work poorly with OOV words and morphology
- Models that operate with word fragments can solve these problems
- FastText package allows to train such models efficiently on CPU

Word embeddings evaluation

Types of metrics

- Word embeddings’ quality can be measured by internal and external criteria

Types of metrics

- Word embeddings' quality can be measured by internal and external criteria
- Internal:
  - Quality of similar words search
  - Quality of analogies solving

Types of metrics

- Word embeddings' quality can be measured by internal and external criteria
- Internal:
    - Quality of similar words search
    - Quality of analogies solving
- External:
    - Quality of the final problem solution (the problem you use word embedding in)

Similar words search

- You have a word embedding model
- You have a corpus with human-evaluated semantic similarity between words

Similar words search

- You have a word embedding model
- You have a corpus with human-evaluated semantic similarity between words

- Calculate cosine similarity between word embeddings:

REMINDER

$$\cos(\theta) = \frac{p \cdot q}{\| p \| \| q \|} = \frac{\sqrt{\sum_{i=1}^{n} p_i q_i}}{\sqrt{\sum_{i=1}^{n} p_i^2} \sqrt{\sum_{i=1}^{n} q_i^2}}$$

Similar words search

- You have a word embedding model
- You have a corpus with human-evaluated semantic similarity between words
- Calculate cosine similarity between word embeddings:

REMINDER

$$\cos(\theta) = \frac{p \cdot q}{\parallel p \parallel \parallel q \parallel} = \frac{\sqrt{\sum_{i=1}^{n} p_i q_i}}{\sqrt{\sum_{i=1}^{n} p_i^2} \sqrt{\sum_{i=1}^{n} q_i^2}}$$

- Check that it correlates with human evaluation

cos(абрикос, персик) > cos(абрикос, маска)

Analogies solving

- You have triplets of words a, a^*, b
- Words a, a^* have some kind of relation

- a = читать, a^* = чтение
  b = петь

Analogies solving

- You have triplets of words a, a^*, b
- Words a, a^* have some kind of relation
- We want to find a word b^* that has the same kind of relation with the word b

- a = читать, a^* = чтение
  b = петь, b^* = пение

Analogies solving

- You have triplets of words a, a^*, b
- Words a, a^* have some kind of relation
- We want to find a word b^* that has the same kind of relation with the word b

$$a^* - a + b$$

-

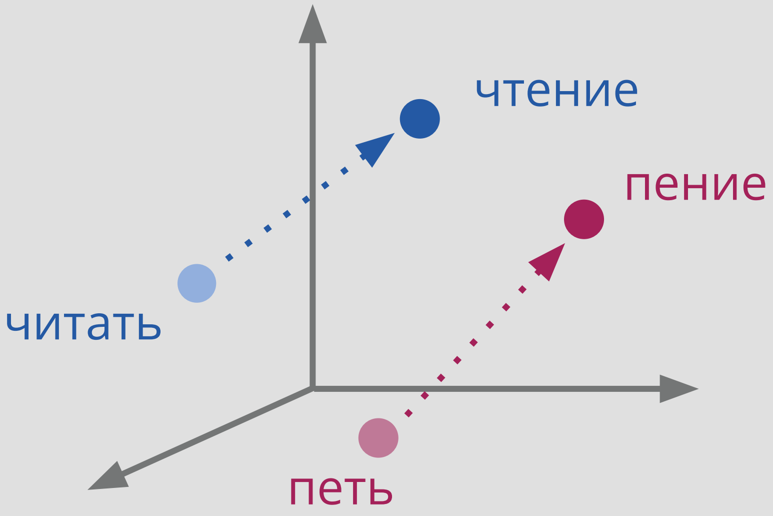
a = читать, a^* = чтение

b = петь, b^* = пение

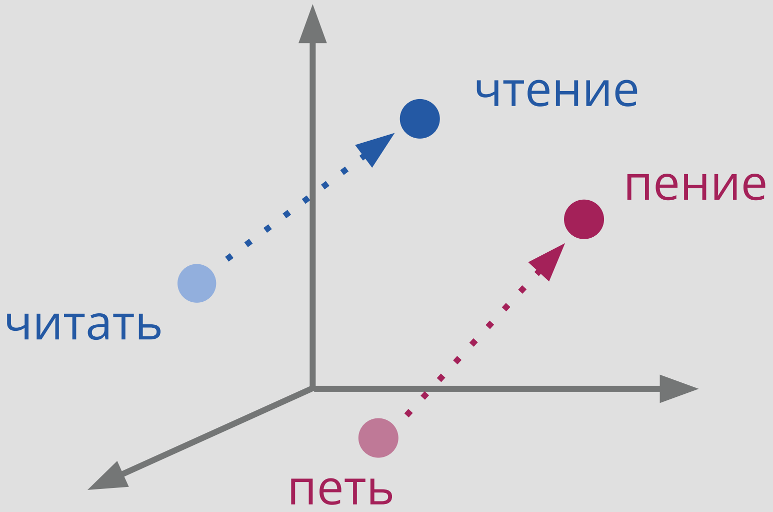
$$a^* - a + b \approx b^*$$

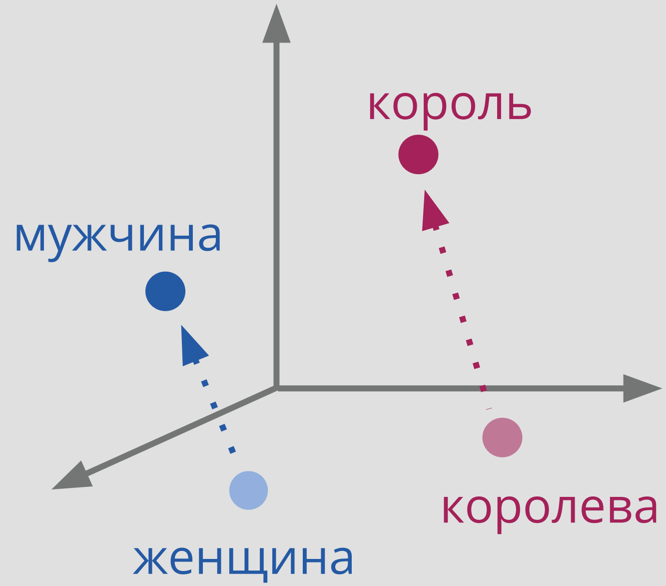
чтение − читать + петь ≈ пение

Analogies solving

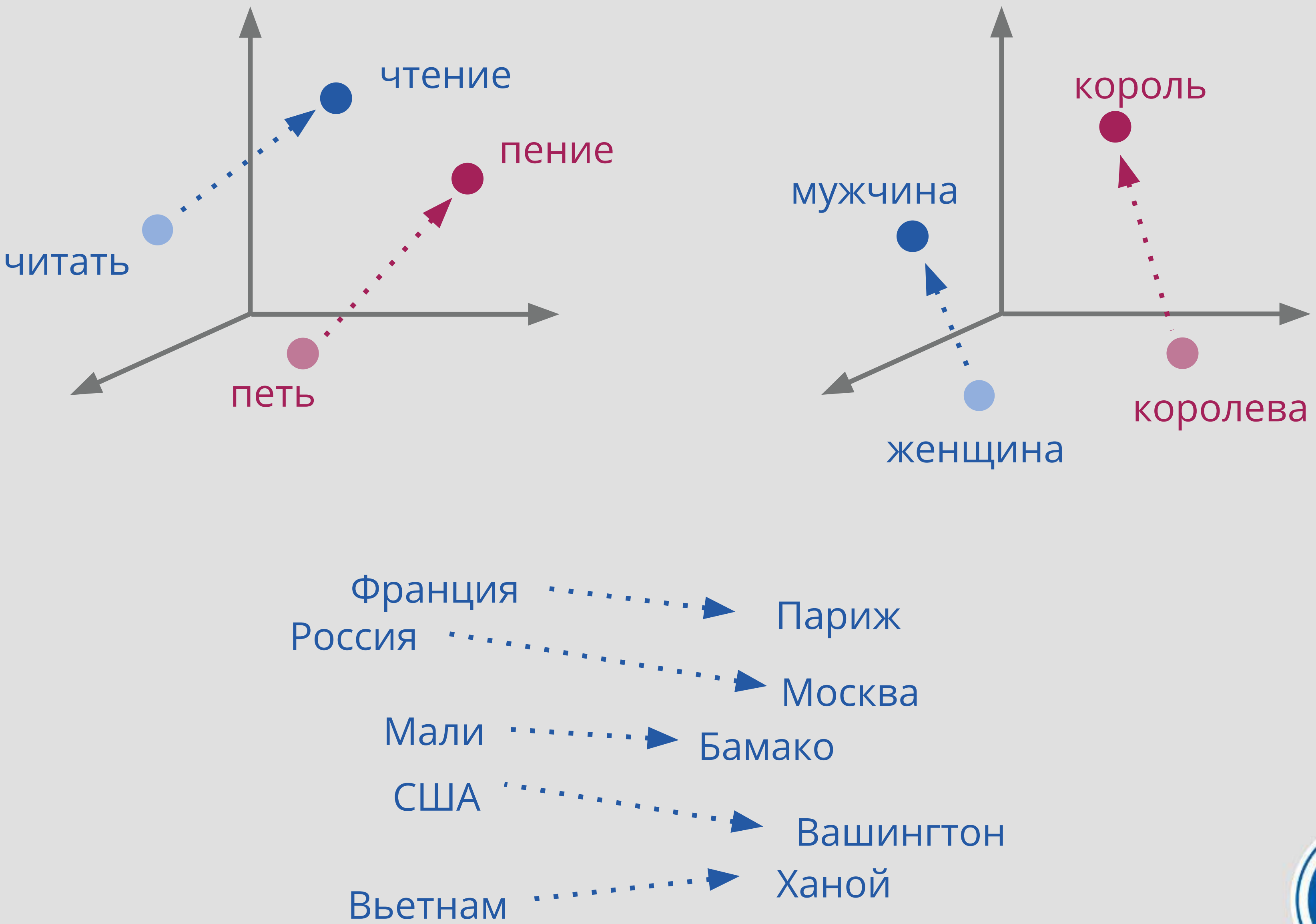


Analogies solving





Analogies solving



External metrics for word embeddings models

- We are not interested in evaluation of word embedding themselves

External metrics for word embeddings models

- We are not interested in evaluation of word embedding themselves
- We solve some task with word embedding models and track the metrics of the solution

External metrics for word embeddings models

- We are not interested in evaluation of word embedding themselves
- We solve some task with word embedding models and track the metrics of the solution
- For example, we can solve some classification problem with TF-IDF vectors of FastText

External metrics for word embeddings models

- We are not interested in evaluation of word embedding themselves
- We solve some task with word embedding models and track the metrics of the solution
- For example, we can solve some classification problem with TF-IDF vectors of FastText
- Mutual quality may differ for different tasks and datasets

Main conclusions

- The quality of word embeddings can be measured by different methods
- Internal criteria measure the models’ quality in terms of their internal properties
- External criteria are more abstract and focus on the final problem where the word embedding model is applied