

# Векторные представления слов

На основе лекций Апишева Мурата <https://openedu.ru/course/hse/TEXT/#>



- Идея векторных представлений.  
One-hot векторы. SVD
- Модель word2vec, методы её обучения
- Оптимизации обучения word2vec: иерархический softmax и SGNS
- Модель GloVe
- Модель FastText, приём Hashing Trick
- Оценка качества векторных представлений



# **Идея векторных представлений. One-hot векторы. SVD**



# Входные данные имеют разный формат



## Визуальный контент

- Изображения
- Видео

<https://www.istockphoto.com/ru/векторная/file-types-vector-icon-set-in-thin-line-style-gm848620942-139394503>



## Тексты

- Неструктурированные документы
- HTML / XML



## Структурированная информация

- Таблицы с выделенными признаками



## Сигналы

- Аудио: музыка / записи речи
- Иные сигналы произвольной природы



# Входные данные имеют разный формат



<https://www.istockphoto.com/ru/векторная/file-types-vector-icon-set-in-thin-line-style-gm848620942-139394503>

ML

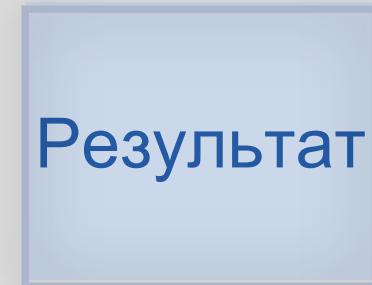
Результат



# Входные данные имеют разный формат



<https://www.istockphoto.com/ru/векторная/file-types-vector-icon-set-in-thin-line-style-gm848620942-139394503>



# Векторные представления слов

“Мой дядя самых честных правил”

“Мой” “самых” “правил”  
“дядя” “честных”

$$\begin{bmatrix} 123 \\ 456 \\ 12 \\ \dots \\ 89 \end{bmatrix}$$

$$\begin{bmatrix} 23 \\ 372 \\ 8 \\ \dots \\ 83 \end{bmatrix}$$

$$\begin{bmatrix} 16 \\ 124 \\ 76 \\ \dots \\ 29 \end{bmatrix}$$

$$\begin{bmatrix} 2 \\ 12 \\ 299 \\ \dots \\ 65 \end{bmatrix}$$

$$\begin{bmatrix} 177 \\ 6 \\ 504 \\ \dots \\ 304 \end{bmatrix}$$



# Какие векторы нам хочется иметь?

Разные  
слова



Разные  
векторы

Экология  
Любить

Близкие  
слова



Близкие  
векторы

Любить  
Обожать



# Что значит «близкие»?

## Семантическая близость слов

«Обычная» близость для слов

Например:

- компьютер
- ноутбук
- ПК
- лаптоп

## Близость $n$ -мерных векторов

$d(p, q)$ , где  $p$  и  $q$  — векторы:  
 $p = (p_1, p_2, \dots, p_n)$   
 $q = (q_1, q_2, \dots, q_n)$

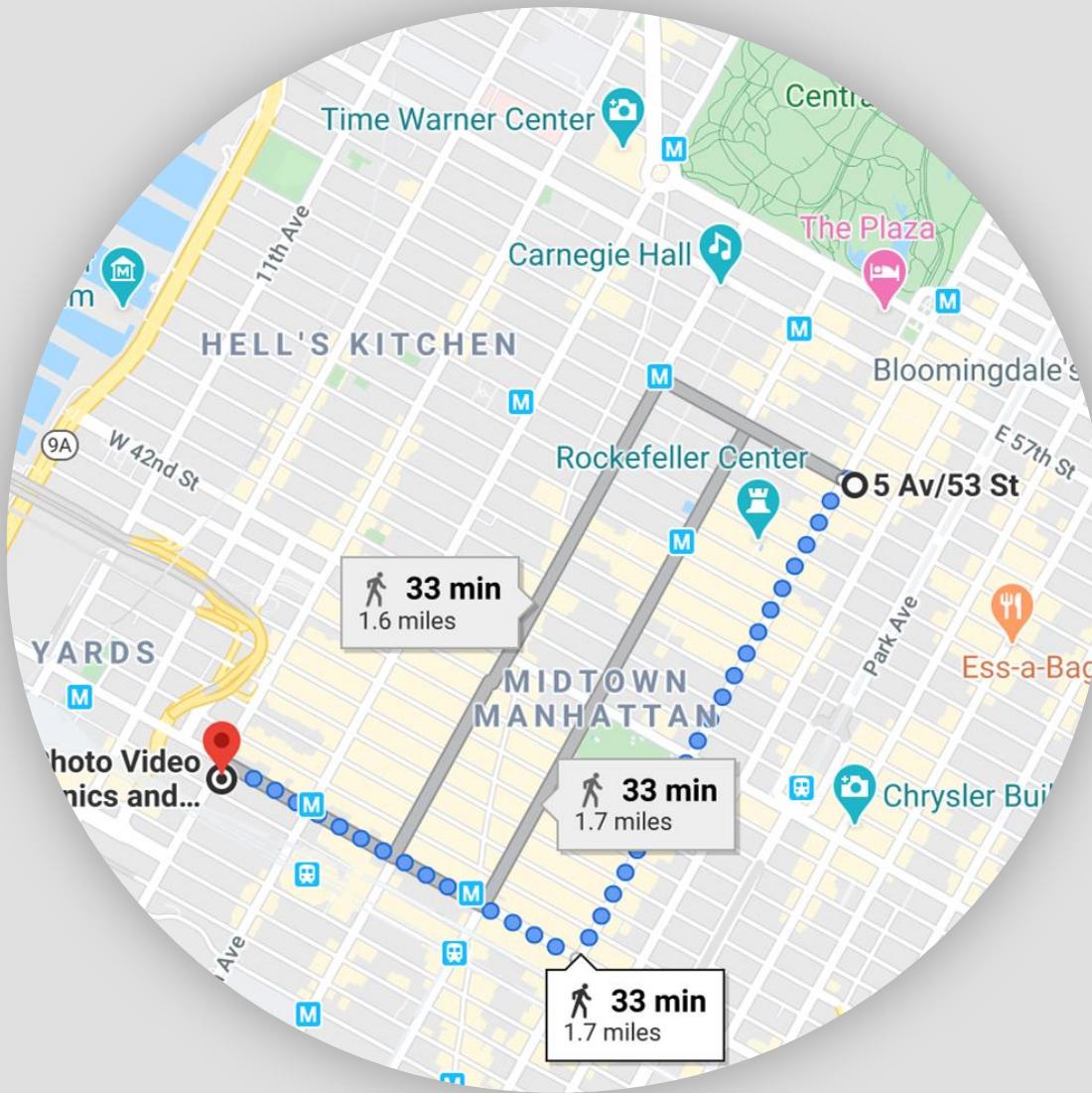
измеряется по-разному:

- Манхэттенское расстояние
- Евклидово расстояние
- Косинусное расстояние



# Манхэттенское расстояние

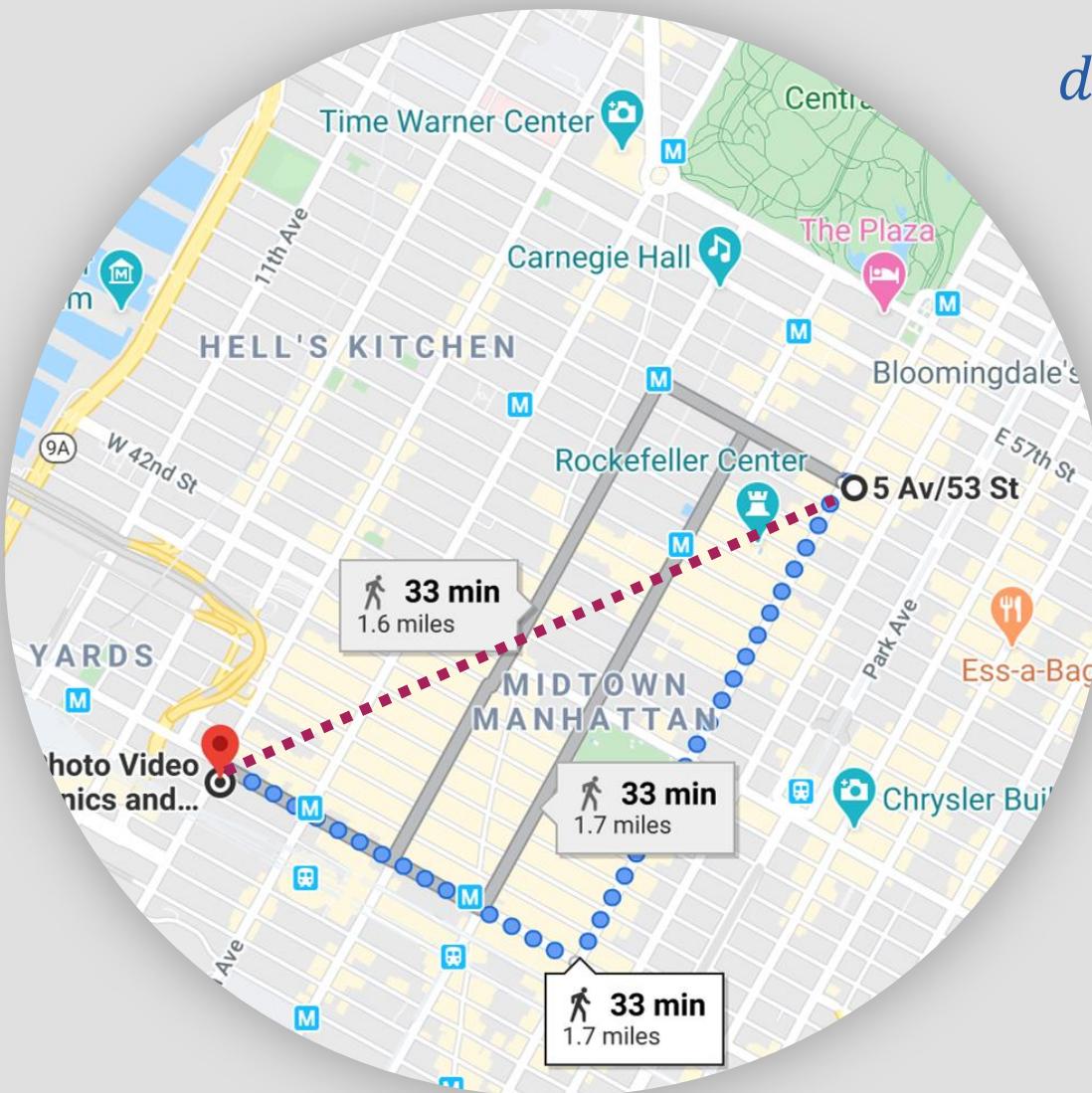
$$d(p, q) = \sum_{i=1}^n |p_i - q_i|$$



[google.com/maps](http://google.com/maps)



# Евклидово расстояние



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

[google.com/maps](http://google.com/maps)



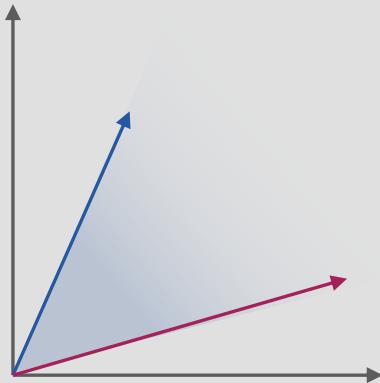
# Косинусное расстояние

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

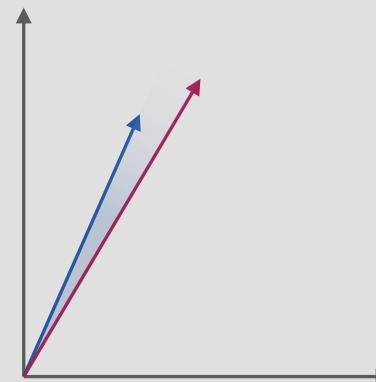
$$\cos(\theta) = \frac{p \cdot q}{\| p \| \| q \|} = \frac{(\sum_{i=1}^n p_i q_i)}{\sqrt{\sum_{i=1}^n p_i^2} \sqrt{\sum_{i=1}^n q_i^2}}$$

Косинусная близость —  $|\cos(\theta)|$

Косинусное расстояние —  $1 - |\cos(\theta)|$



$$\theta \approx 45^\circ$$
$$\cos(\theta) \approx 0.7$$



$$\theta \approx 7^\circ$$
$$\cos(\theta) \approx 0.99$$



# One-hot кодирование

Пусть у нас есть словарь из  $n$  уникальных слов

Абрикос    Апельсин

Яблоко

1
0
0
0

$a_1$

0
1
0
0

$a_2$

...

0
0
0
1

$a_n$



$n$  значений

## Достоинства

- Простой способ получить векторы для множества уникальных слов

## Недостатки

- Итоговые векторы имеют огромную нефиксированную длину
- Векторы одинаково далеки друг от друга



# Сингулярное разложение (SVD)

$$\begin{matrix} & m \\ \text{n} & \end{matrix} \begin{matrix} & n \\ = & \end{matrix} \begin{matrix} & n \\ \text{n} & \end{matrix} \times \begin{matrix} & m \\ \text{n} & \end{matrix} \times \begin{matrix} & m \\ \text{m} & \end{matrix}$$

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



# Сингулярное разложение (SVD)

- Дана прямоугольная матрица  $\mathbf{M} \in \mathbb{R}^{n \times m}$

$$\begin{matrix} & m \\ \text{---} & \text{---} \\ n & \end{matrix} = \begin{matrix} & n \\ \text{---} & \text{---} \\ n & \end{matrix} \times \begin{matrix} & m \\ \text{---} & \text{---} \\ n & \end{matrix} \times \begin{matrix} & m \\ \text{---} & \text{---} \\ m & \end{matrix}$$

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



# Сингулярное разложение (SVD)

- Дано прямоугольная матрица  $\mathbf{M} \in \mathbb{R}^{n \times m}$
- Она представима в виде  $\mathbf{M} = \mathbf{U}\Sigma\mathbf{V}^T$

$$\begin{matrix} & m \\ \begin{matrix} n \\ \text{---} \\ n \end{matrix} & \end{matrix} = \begin{matrix} & n \\ \begin{matrix} n \\ \text{---} \\ n \end{matrix} & \end{matrix} \times \begin{matrix} & m \\ \begin{matrix} n \\ \text{---} \\ n \end{matrix} & \end{matrix} \times \begin{matrix} & m \\ \begin{matrix} m \\ \text{---} \\ m \end{matrix} & \end{matrix}$$

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



# Сингулярное разложение (SVD)

- $\Sigma \in \mathbb{R}^{n \times m}$  — диагональная матрица с неотрицательными элементами
- Диагональные элементы  $\Sigma$  — сингулярные числа матрицы  $\mathbf{M}$

$$\begin{matrix} & m \\ \text{n} & \end{matrix} \begin{matrix} & n \\ = & \end{matrix} \begin{matrix} & n \\ \mathbf{U} \in \mathbb{R}^{n \times n} & \end{matrix} \begin{matrix} & n \\ \times & \end{matrix} \begin{matrix} & m \\ \Sigma \in \mathbb{R}^{n \times m} & \end{matrix} \begin{matrix} & m \\ \times & \end{matrix} \begin{matrix} & m \\ \mathbf{V}^T \in \mathbb{R}^{m \times m} & \end{matrix}$$



# Сингулярное разложение (SVD)

- Столбцы  $\mathbf{U}$  и  $\mathbf{V}$  — левые и правые сингулярные векторы матрицы  $\mathbf{M}$

$$\begin{matrix} m \\ n \end{matrix} \begin{matrix} n \\ n \end{matrix} \times \begin{matrix} m \\ n \end{matrix} = \begin{matrix} m \\ n \end{matrix} \times \begin{matrix} m \\ n \end{matrix} \times \begin{matrix} m \\ n \end{matrix} \times \begin{matrix} m \\ m \end{matrix}$$

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



# Сингулярное разложение (SVD)

- Пусть дана коллекция из  $m$  документов со словарём размера  $n$
- Пусть  $\mathbf{M} \in \mathbb{R}^{n \times m}$  — матрица «мешка слов»
- Применим SVD:  $\mathbf{M} = \mathbf{U}\Sigma\mathbf{V}^T$

$$\begin{matrix} & m \\ \mathbf{M} \in \mathbb{R}^{n \times m} & \end{matrix} = \begin{matrix} & n \\ \mathbf{U} \in \mathbb{R}^{n \times n} & \end{matrix} \times \begin{matrix} & m \\ \Sigma \in \mathbb{R}^{n \times m} & \end{matrix} \times \begin{matrix} & m \\ \mathbf{V}^T \in \mathbb{R}^{m \times m} & \end{matrix}$$



# Сингулярное разложение (усеченное SVD)

- Выберем  $k$  наибольших значения в  $\Sigma$
- Прочие значения и соответствующие столбцы  $\mathbf{U}$  игнорируем — получаем матрицы  $\mathbf{U}_k$  и  $\Sigma_k$

$$\begin{matrix} m \\ n \end{matrix} = \begin{matrix} k \\ n \end{matrix} \times \begin{matrix} k \\ n \end{matrix} \times \begin{matrix} m \\ n \end{matrix}$$

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



# Векторные представления с помощью усеченного SVD

- В качестве векторных представлений слов можно взять строки матрицы  $\mathbf{U}_k \sqrt{\Sigma_k}$

The diagram shows a matrix  $E \in \mathbb{R}^{n \times k}$  being factored into three components:  $E = U_k \sqrt{\Sigma_k} \mathbf{X}$ . The matrix  $E$  is a vertical column with rows labeled by words: Абрикос, Апельсин, Арбуз, ..., Черешня, Яблоко. It has  $n$  rows and  $k$  columns. The factor  $U_k \in \mathbb{R}^{n \times k}$  is shown as a tall column matrix with  $n$  rows and  $k$  columns. The factor  $\sqrt{\Sigma_k} \in \mathbb{R}^{k \times k}$  is shown as a square diagonal matrix with  $k$  rows and  $k$  columns, where only the top-left  $k \times k$  block contains non-zero values.

$$E \in \mathbb{R}^{n \times k} \quad U_k \in \mathbb{R}^{n \times k} \quad \sqrt{\Sigma_k} \in \mathbb{R}^{k \times k}$$

Кстати:

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



# Векторные представления с помощью SVD

Что стало лучше:

- Векторы имеют фиксированную длину
- Они не взаимно ортогональны
- Семантическая близость как-то учитывается

Но:

- Добавление новых слов/документов потребует построение нового разложения
- Мы в явном виде должны работать с большой разреженной матрицей «мешка слов»
- Качество векторов можно улучшить



# Основные выводы

- Векторные представления слов используются в качестве признаков при работе с текстами
- Хорошие векторные представления отражают свойства слов, например, семантическую близость
- One-hot векторы могут быть полезны, но они очень большие и взаимно ортогональные
- С помощью сингулярного разложения можно получить векторные представления фиксированного размера, учитывающие семантику



# Модель word2vec, методы её обучения



# Гипотеза дистрибутивной семантики

- Слова со схожим смыслом разделяют схожий контекст

«Сегодня я съел **вкусный, сочный апельсин**»

«Это яблоко очень сладкое и **сочное**»

«**Сладкие** у нас абрикосы, очень **вкусные**»

- Вместо того, чтобы собирать счётчики, обучим модель, которая предсказывает по слову его контекст (или наоборот)

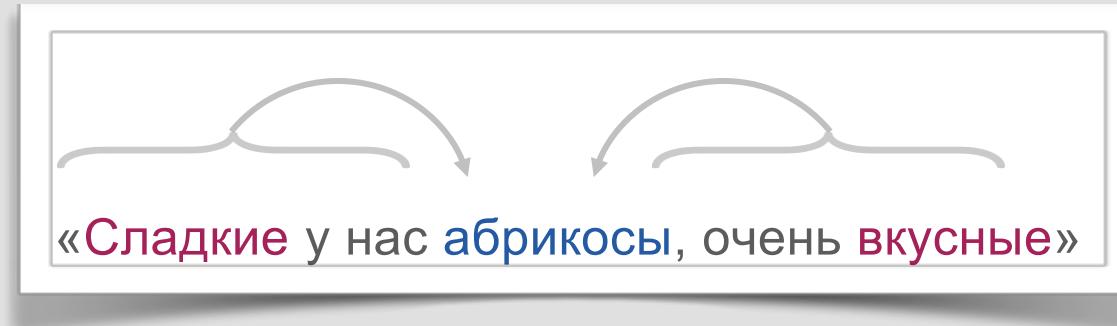
➤ Harris Zellig. *Distributional structure* // Word. — 1954. — Vol. 10, no. 23. — Pp. 146–162.



# Модели Word2vec: CBOW и skip-gram

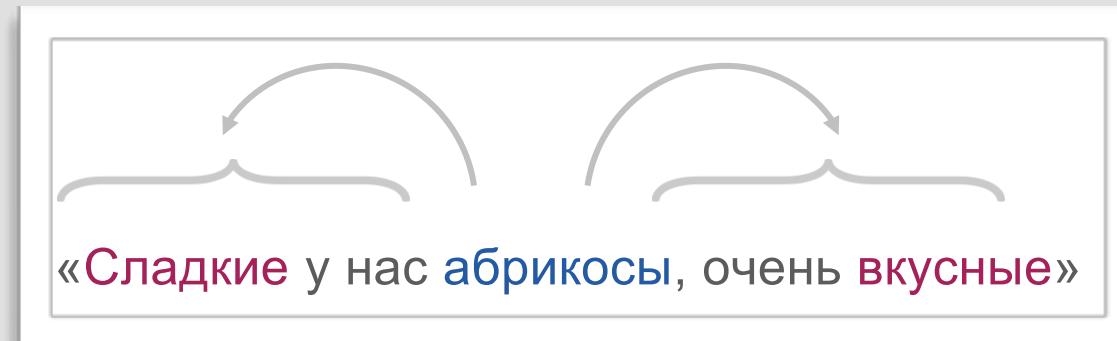
- Continues Bag-of-Words

предсказываем по контекстным словам центральное слово



- Skip-gram

предсказываем по центральному слову слова из его контекста



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



# Модель СВОУ

- Пусть  $N$  - общее число слов в коллекции
- При обучении идём окном размера  $2C + 1$  по всем словам

Сладкие

у

нас

абрикосы

очень

вкусные

шаг 1



# Модели Word2vec: CBOW и skip-gram

Сладкие

у

нас

абрикосы

очень

вкусные

шаг 2



# Модели Word2vec: CBOW и skip-gram

Сладкие

у

нас

абрикосы

очень

вкусные

шаг 3



# Модели Word2vec: CBOW и skip-gram

Сладкие

у

нас

абрикосы

очень

вкусные

шаг 4



# Модели Word2vec: CBOW и skip-gram

Сладкие

у

нас

абрикосы

очень

вкусные

шаг 5



# Модели Word2vec: CBOW и skip-gram

Сладкие

у

нас

абрикосы

очень

вкусные

шаг 6



# CBOW как нейронная сеть

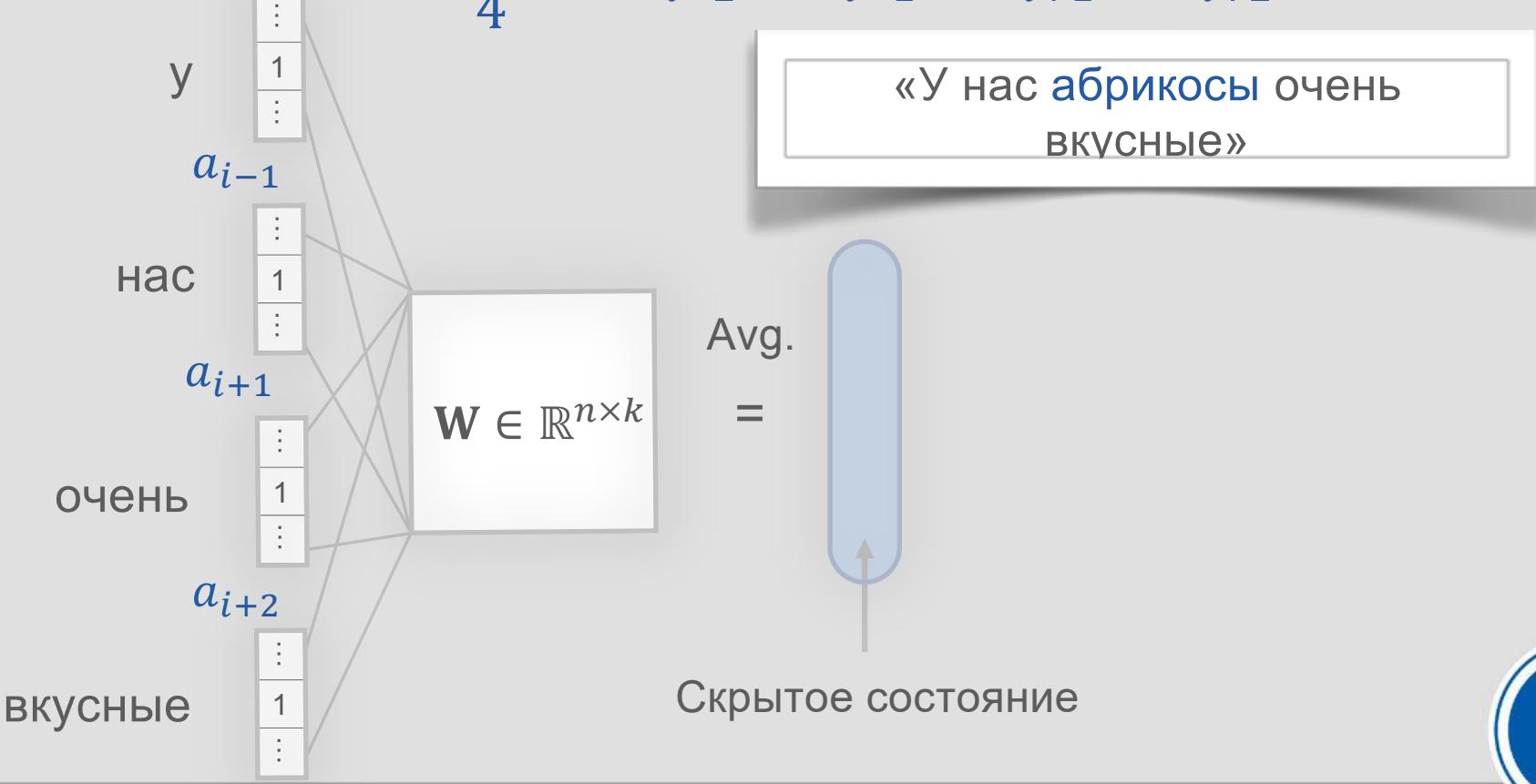
- Модель — двуслойная нейронная сеть
- На входе -  $2C$   $n$ -мерных one-hot векторов слов контекста



# CBOW как нейронная сеть

- Применим ко входу преобразование  $W$  и усредним результаты – получим  $k$ -мерный вектор скрытого состояния:

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



# CBOW как нейронная сеть

- Умножим скрытое состояние на вторую матрицу – получим  $n$ -мерный выходной вектор:

$$b = V^T h$$



# Оптимизируемый функционал

- Для  $i$  –го окна хотим по контексту  $c_i$  предсказывать  $w_i$ :

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



# Оптимизируемый функционал

- Для  $i$  –го окна хотим по контексту  $c_i$  предсказывать  $w_i$ :

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

- Для этого на выходе должен быть  $n$ -мерный вектор вероятностей



# Оптимизируемый функционал

- Для  $i$  –го окна хотим по контексту  $c_i$  предсказывать  $w_i$ :

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

- Для этого на выходе должен быть  $n$ -мерный вектор вероятностей
- Воспользуемся функцией Softmax:

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

- Если сеть не ошиблась, то значение вероятности в  $i$  –й позиции будет большим



# А где векторные представления?

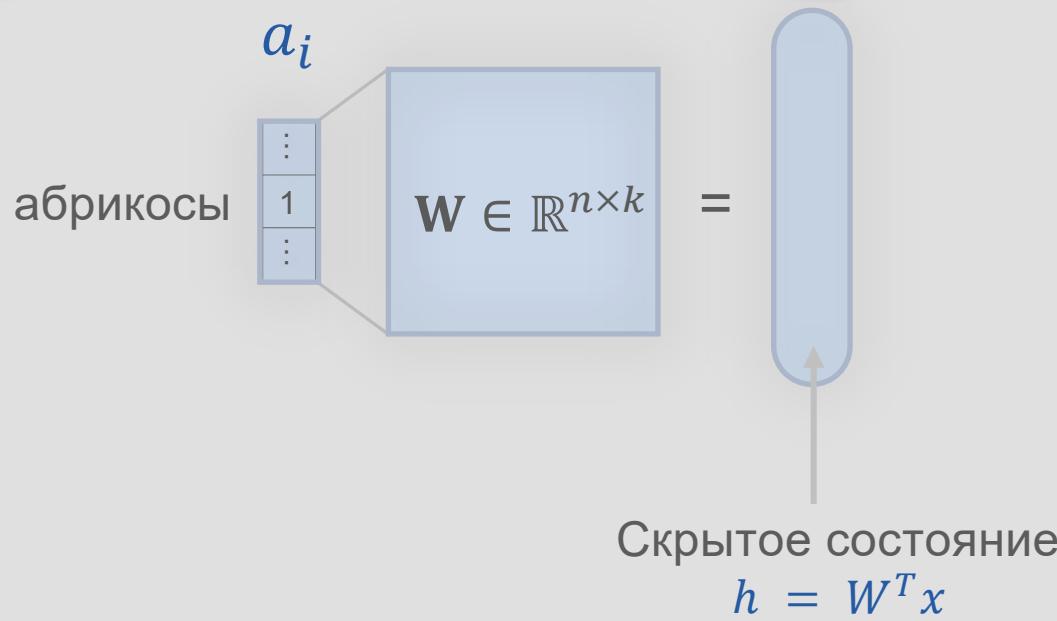
- В результате обучения мы получили две матрицы весов –  $W$  и  $V$
- Обычно в качестве векторов слов используются строки матрицы  $W$
- Но можно использовать и строки  $V^T$  или комбинацию обеих матриц



# Skip-gram как нейронная сеть

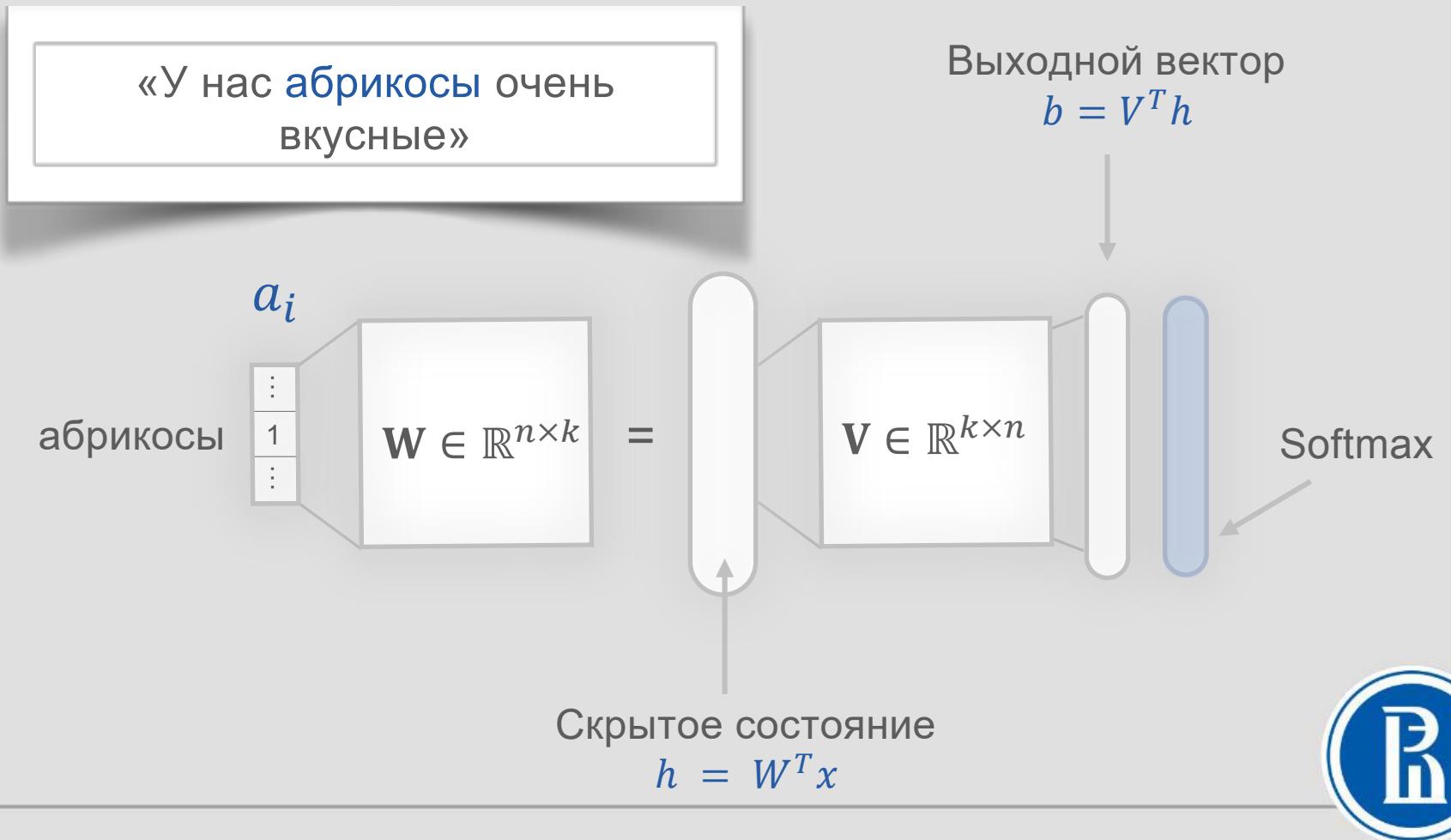
- Модель skip-gram устроена симметрично

«У нас **абрикосы** очень  
вкусные»



# Skip-gram как нейронная сеть

- На выходе — одно вероятностное распределение слов контекста при условии центрального слова



# Оптимизируемый функционал

- На выходе — одно вероятностное распределение слов контекста при условии центрального слова
- $2C$  слов настоящего контекста должны иметь в нём как можно большие значения
- Оптимизируемый функционал:

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

- На практике модель Skip-gram обычно работает лучше CBOW



# Основные выводы

- Модели word2vec строят представления на основе предсказаний, а не подсчёта статистики
- Есть две основные модели: CBOW и Skip-gram
- В каноническом варианте word2vec обучается двуслойная нейронная сеть, веса которой являются итоговыми векторными представлениями
- Качество векторных представлений word2vec выше, чем у SVD, не нужно строить огромные разреженные матрицы счётчиков



# Модель FastText, приём Hashing Trick



# Какие проблемы есть у word2vec и GloVe

- Проблема 1: Out-of-Vocabulary (OOV) слова



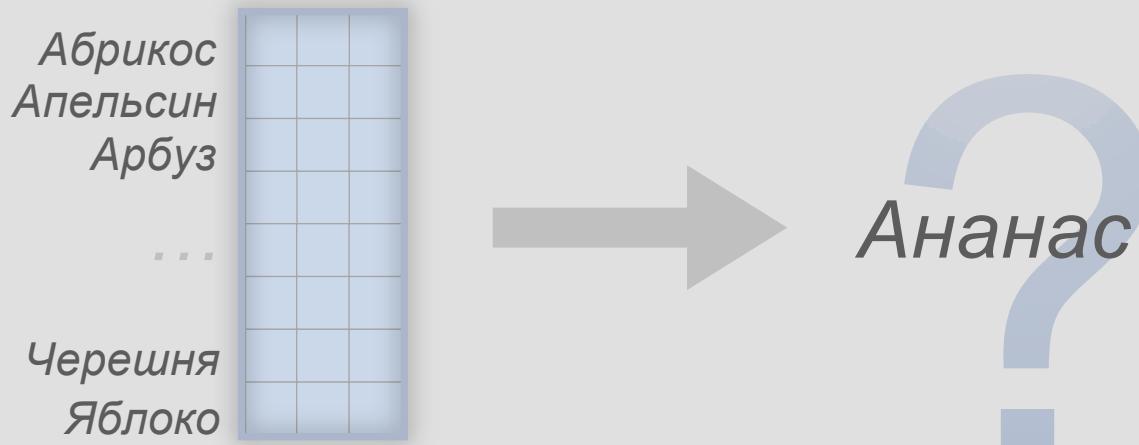
# Какие проблемы есть у word2vec и GloVe

- Проблема 1: Out-of-Vocabulary (OOV) слова
- Обучившись на корпусе со словарём  $V_1$ , мы получили векторы для всех слов из  $V_1$
- Попытаемся использовать эти векторы для документов со словарём  $V_2$
- Для слов из  $V_2$ , которых нет в  $V_1$ , векторов нет



# Какие проблемы есть у word2vec и GloVe

- Проблема 1: Out-of-Vocabulary (OOV) слова
- Обучившись на корпусе со словарём  $V_1$ , мы получили векторы для всех слов из  $V_1$
- Попытаемся использовать эти векторы для документов со словарём  $V_2$
- Для слов из  $V_2$ , которых нет в  $V_1$ , векторов нет



# Какие проблемы есть у word2vec и GloVe

- Проблема 2: Отсутствие учёта морфологии
- Допустим, мы обучаем векторы для языка с богатой морфологией (например, русского):



# Какие проблемы есть у word2vec и GloVe

- Проблема 2: Отсутствие учёта морфологии
- Допустим, мы обучаем векторы для языка с богатой морфологией (например, русского):
  - Для каждого варианта получаем свой вектор
  - В результате:
    - Много похожих векторов (лишнее потребление памяти)
    - Для каждого варианта меньше обучающих данных

яблоко  
яблока  
яблоку  
...  
яблоками  
яблоках



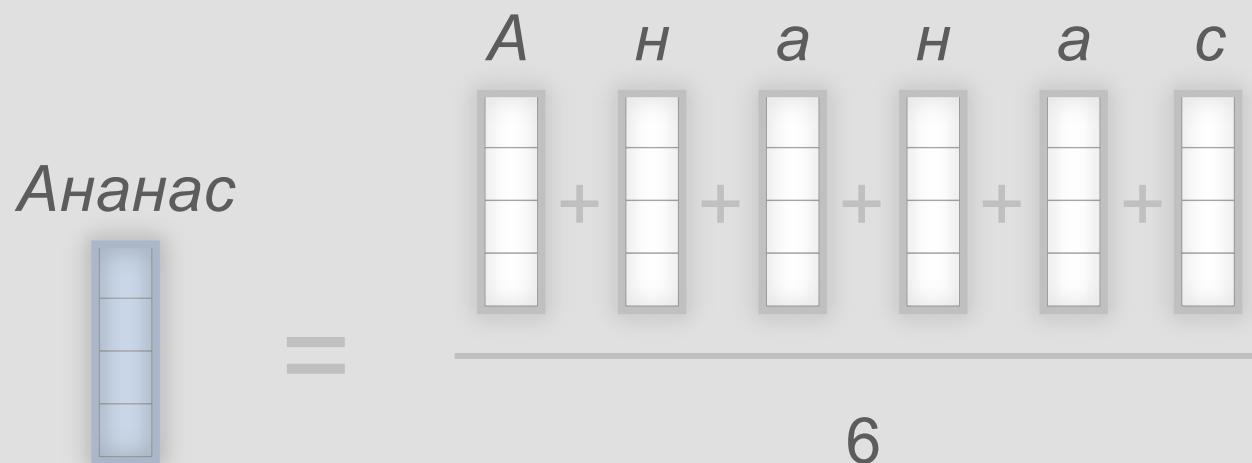
# Векторные представления символов

- Можно попробовать строить векторные представления для **меньшей единицы языка**
- Например, для каждого символа
- Все принципы обучения сохраняются
- Вектор для слова можно получать усреднением векторов символов, из которых оно состоит



# Векторные представления символов

- Можно попробовать строить векторные представления для **меньшей единицы языка**
- Например, для каждого символа
- Все принципы обучения сохраняются
- Вектор для слова можно получать усреднением векторов символов, из которых оно состоит



# Векторные представления символьных последовательностей (N-грамм)

- Вместо отдельных символов, можно дополнительно использовать **символьные последовательности**
- Это существенно улучшает качество итоговых векторов слов



# Векторные представления символьных последовательностей (N-грамм)

- Вместо отдельных символов, можно дополнительно использовать **символьные последовательности**
- Это существенно улучшает качество итоговых векторов слов

N-граммы слова «боцман»

N=3: ^бо, боц, оцм, цма, ман, ан\$

N=4: ^боц, боцм, оцма, цман, ман\$

N=5: ^боцм, боцма, оцман, цман\$



# Векторные представления символьных последовательностей (N-грамм)

- Вместо отдельных символов, можно дополнительно использовать **символьные последовательности**
- Это существенно улучшает качество итоговых векторов слов

N-граммы слова «боцман»

N=3: ^бо, боц, оцм, цма, ман, ан\$

N=4: ^боц, боцм, оцма, цман, ман\$

N=5: ^боцм, боцма, оцман, цман\$

- Векторы слов опять получаются усреднением



# Какие проблемы есть у word2vec и GloVe

Что стало лучше:

- Мы решили проблему OOV-слов
- С морфологией тоже справились

Но:

- Последовательностей символов может быть ещё больше, чем различных вариантов слов
- Десятки миллионов векторов могут просто не поместиться в оперативную память



# Хэш-функция и хэш-таблица

- Хэш-функция от строки — преобразование, которое строку переводит в целое число



# Хэш-функция и хэш-таблица

- Хэш-функция от строки — преобразование, которое строку переводит в целое число
- Требования к функции:
  - Область значений ограничена отрезком
  - На множестве всех строк значения распределяются примерно равномерно



# Хэш-функция и хэш-таблица

- Хэш-функция от строки — преобразование, которое строку переводит в целое число
- Требования к функции:
  - Область значений ограничена отрезком
  - На множестве всех строк значения распределяются примерно равномерно
- Хэш-таблица — массив + хэш-функция, которая переводит входные строки в индексы этого массива



# Hashing Trick

- Зафиксируем максимальное число векторов, которые мы хотим обучить



# Hashing Trick

- Зафиксируем максимальное число векторов, которые мы хотим обучить
- Сопоставим им хэш-таблицу



# Hashing Trick

- Зафиксируем максимальное число векторов, которые мы хотим обучить
- Сопоставим им хэш-таблицу
- Векторы для всех символьных N-грамм распределяются по ячейкам массива



# Hashing Trick

- Зафиксируем максимальное число векторов, которые мы хотим обучить
- Сопоставим им хэш-таблицу
- Векторы для всех символьных N-грамм распределяются по ячейкам массива
- Несколько N-грамм используют один и тот же вектор



# FastText

- Библиотека, предназначенная для построения векторных представлений слов
- Использует CBOW / skip-gram и для слов, и для символьных N-грамм
- Оптимизирует потребление памяти с помощью hashing trick
- Обучается параллельно на 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.*



# Основные выводы

- Чисто словарные векторные модели плохо справляются с новыми словами и морфологией
- Модели, учитывающие фрагменты слов, позволяют побороть эти сложности
- Открытая реализация FastText позволяет эффективно обучать такие модели на CPU



# **Оценка качества векторных представлений**



# Типы метрик

- Качество векторных представлений может измеряться с помощью **внутренних** и **внешних** критериев



# Типы метрик

- Качество векторных представлений может измеряться с помощью **внутренних** и **внешних** критериев
- Внутренние:
  - Качество поиска близких слов
  - Качество поиска аналогий



# Типы метрик

- Качество векторных представлений может измеряться с помощью **внутренних** и **внешних** критериев
- Внутренние:
  - Качество поиска близких слов
  - Качество поиска аналогий
- Внешние:
  - Качество решения любой задачи, использующей обученные векторы



# Поиск близких по смыслу слов

- Дан набор обученных векторов слов
- Даны оценки семантической близости между словами, проставленные людьми



# Поиск близких по смыслу слов

- Дан набор обученных векторов слов
- Даны оценки семантической близости между словами, проставленные людьми
- Измеряем косинусную близость между векторами пар слов

## REMINDER

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



# Поиск близких по смыслу слов

- Дан набор обученных векторов слов
- Даны оценки семантической близости между словами, проставленные людьми
- Измеряем косинусную близость между векторами пар слов

## REMINDER

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

- Проверяем, что она выше для близких слов и синонимов, и ниже – для прочих

$\cos(\text{абрикос, персик}) > \cos(\text{абрикос, маска})$



# Задача поиска аналогий

- Даны тройки слов  $a$ ,  $a^*$ ,  $b$
- Между словами  $a$  и  $a^*$  есть какая-то зависимость

$a$  = читать,  $a^*$  = чтение  
 $b$  = петь



# Задача поиска аналогий

- Даны тройки слов  $a$ ,  $a^*$ ,  $b$
- Между словами  $a$  и  $a^*$  есть какая-то зависимость
- Хотим найти слово  $b^*$ , такое, чтобы между ним и словом  $b$  была аналогичная зависимость

$a$  = читать,  $a^*$  = чтение

$b$  = петь,  $b^*$  = пение



# Задача поиска аналогий

- Даны тройки слов  $a$ ,  $a^*$ ,  $b$
- Между словами  $a$  и  $a^*$  есть какая-то зависимость
- Хотим найти слово  $b^*$ , такое, чтобы между ним и словом  $b$  была аналогичная зависимость
- Измеряем близость между истинным  $b^*$  и вектором  
$$a^* - a + b$$

$a$  = читать,  $a^*$  = чтение

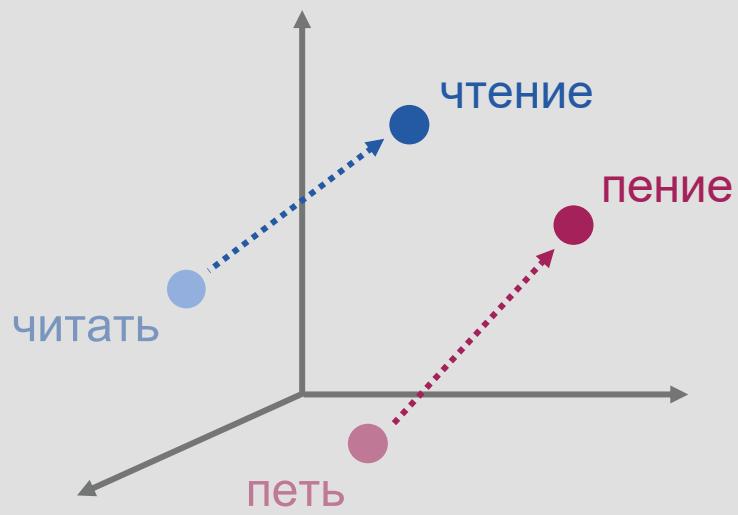
$b$  = петь,  $b^*$  = пение

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

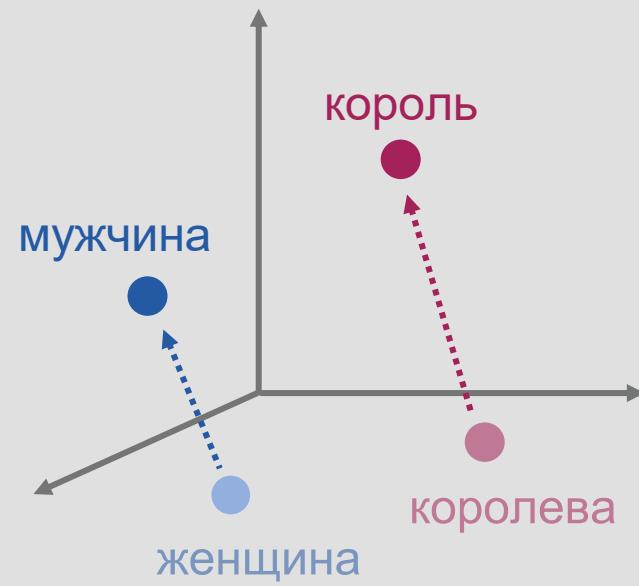
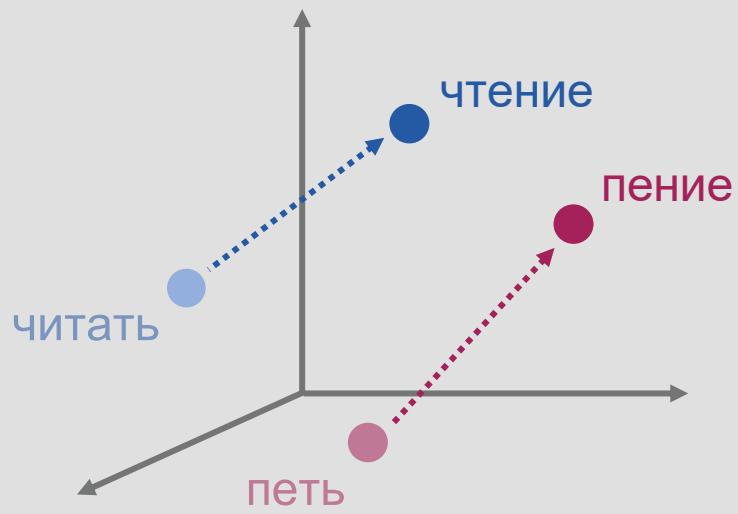
чтение – читать + петь  $\approx$  пение



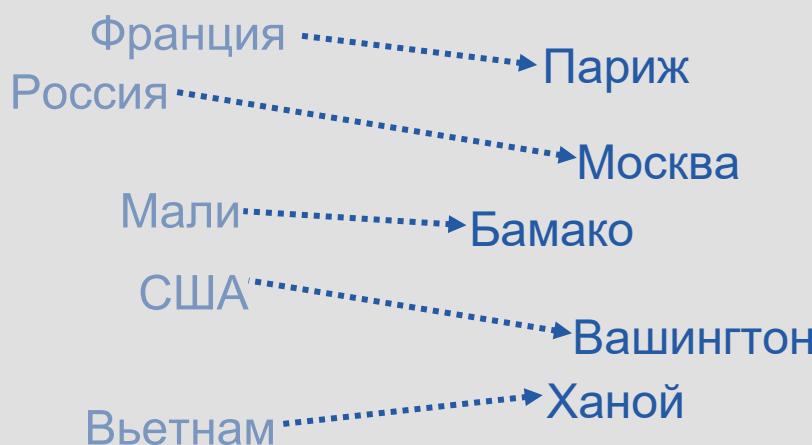
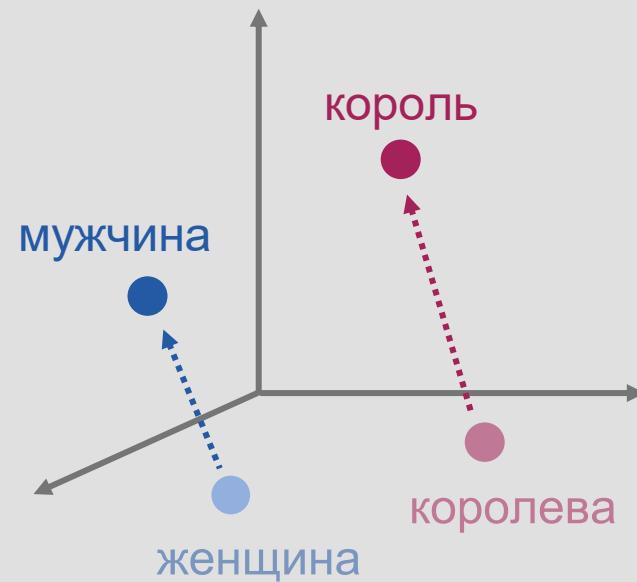
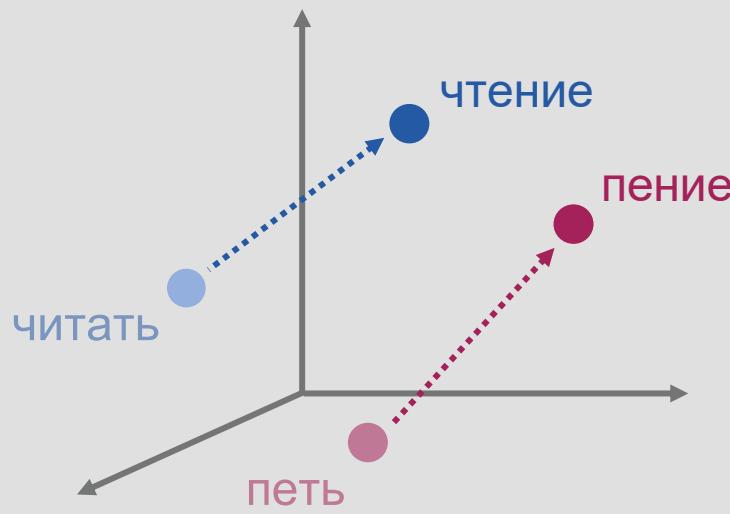
# Задача поиска аналогий



# Задача поиска аналогий



# Задача поиска аналогий



# Внешние оценки качества векторных представлений слов

- Свойства векторов сами по себе не интересуют



# Внешние оценки качества векторных представлений слов

- Свойства векторов сами по себе не интересуют
- Измеряется качество решения задачи



# Внешние оценки качества векторных представлений слов

- Свойства векторов сами по себе не интересуют
- Измеряется качество решения задачи
- Например, можно решать классификацию с помощью векторов TF-IDF, а можно с помощью FastText



# Внешние оценки качества векторных представлений слов

- Свойства векторов сами по себе не интересуют
- Измеряется качество решения задачи
- Например, можно решать классификацию с помощью векторов TF-IDF, а можно с помощью FastText
- Для различных задач и наборов данных взаимное качество векторов будет меняться



# Основные выводы

- Качество векторных представлений слов может определяться различными способами
- Внутренние критерии оценивают качество векторов с точки зрения их предполагаемых свойств
- Внешние абстрагируются от свойств векторов и оценивают их с точки зрения задачи, использующей их

