

# Введение в NLP. Дистрибутивная семантика

Парпулов Дмитрий  
программист в команде  
машиинного обучения почты



# План:

- Применения NLP
- Предобработка текста: токенизация, лемматизация, стемминг
- Векторное представление слов: классические методы
- Дистрибутивная семантика: word2vec
- Advanced vectorisation methods: Glove, Doc2Vec, FastText
- Bonus: BPE

# Введение в NLP



# NLP - это важно:

Люди выражают свои мысли, как правило, на естественном языке



*"А сегодня, в завтрашний  
день, не все могут смотреть.  
Вернее смотреть могут не  
только лишь все, мало кто  
может это делать."*

*Виталий Кличко*

# NLP - это важно:

Люди выражают свои мысли, как правило, на естественном языке

Генерят множество документов на естественном языке: блоги, твиты, отзывы, статьи, запросы, ...

Нужно уметь обрабатывать эти документы: определять тематики, эмоциональную окраску, выделять сущности, фильтровать спам, ...

# NLP - применение:

- NER (распознавание именованных сущностей)



# NER

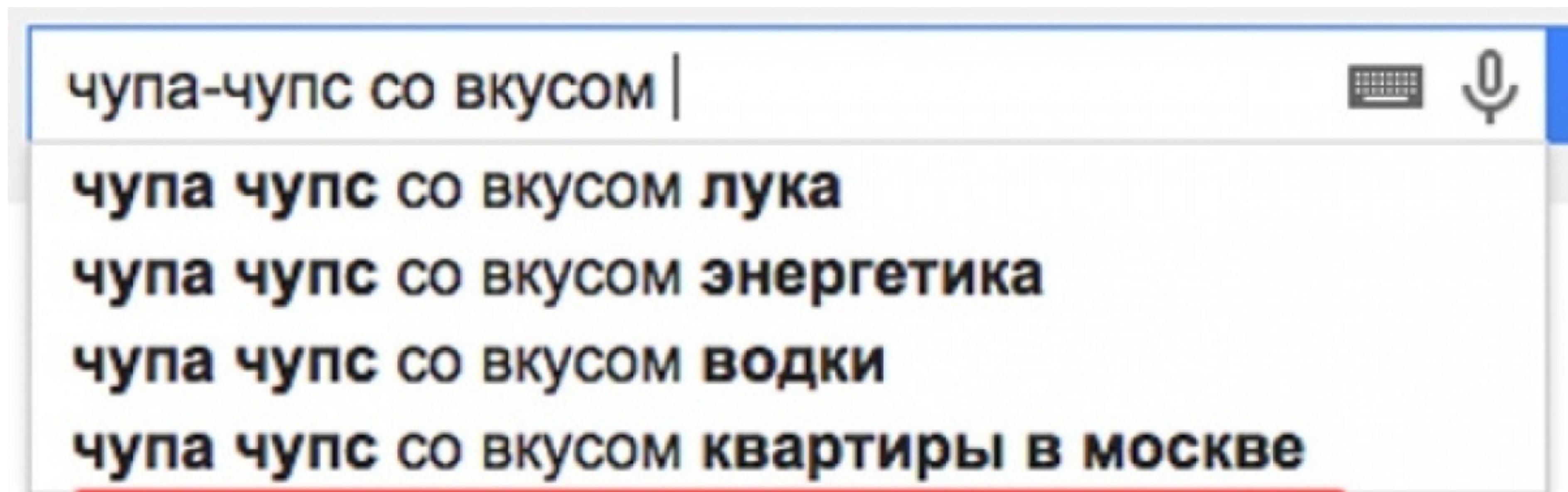


# NLP - применение:

- NER (распознавание именованных сущностей)
- Поиск

# Поиск

## Запросы на естественном языке



Для поиска нажмите "Ввод"

# NLP - применение:

- NER (распознавание именованных сущностей)
- Поиск
- Перевод

# Перевод

## Качество перевода улучшается с каждым годом

▼ ▼ ▼ 04.070 ПОДДЕРЖАТЬ ОТВЕТВИТЬ СОВРАТИТЬ

я женщин > Толстовки и кофты



Narajuku женщины **джемпер моей мамы говорит я довольно так пошел на хай**  
письма свободного покроя с капюшоном для леди битник осень-хай-стрит чернъ<sup>е</sup>  
белый

Ай Посмотреть название на английском

★★★★★ 4.8 (21 голоса(ов)) | 95 заказа(ов)

Цена: 1 197,68 руб. / шт.

Цена со скидкой: **1 005.78 руб.** / шт. 16% off | Осталось дней: 4

# NLP - применение:

- NER (распознавание именованных сущностей)
- Поиск
- Перевод
- Чат-боты

# Чат-боты

Мне ампутировали конечности  
после войны, не шути, пожалуйста,  
на эту тему.



Простите, но интернета, кажется,  
нет, а я без него как без рук.

Спросите что-нибудь



# NLP - применение:

- NER (распознавание именованных сущностей)
- Поиск
- Перевод
- Чат-боты
- Голосовые ассистенты

# Голосовые ассистенты

Распознавание голоса



# NLP - применение:

- NER (распознавание именованных сущностей)
- Поиск
- Перевод
- Чат-боты
- Голосовые ассистенты
- Анализ тональности

# Анализ тональности

## Мой отзыв:



Лучший в мире насос. Король насосов. Редко когда встретишь такой качественный и красивый насос. Насасывает так, что дух захватывает. Моя жена равнодушна к насосам, но даже она полюбила его как члена семьи. Нам пока нечего им надувать, мы взяли его исключительно ради его неотразимой внешности. Да и цена показалась доступной. Доставили реально быстро. Мы не успели даже соскучиться по нашему новому малышу-насосику. Продавец душка и лапочка! Всем насосы! Бог благословит качающих воздух!



# Сложности

Ключевая проблема обработки текстов - многозначность



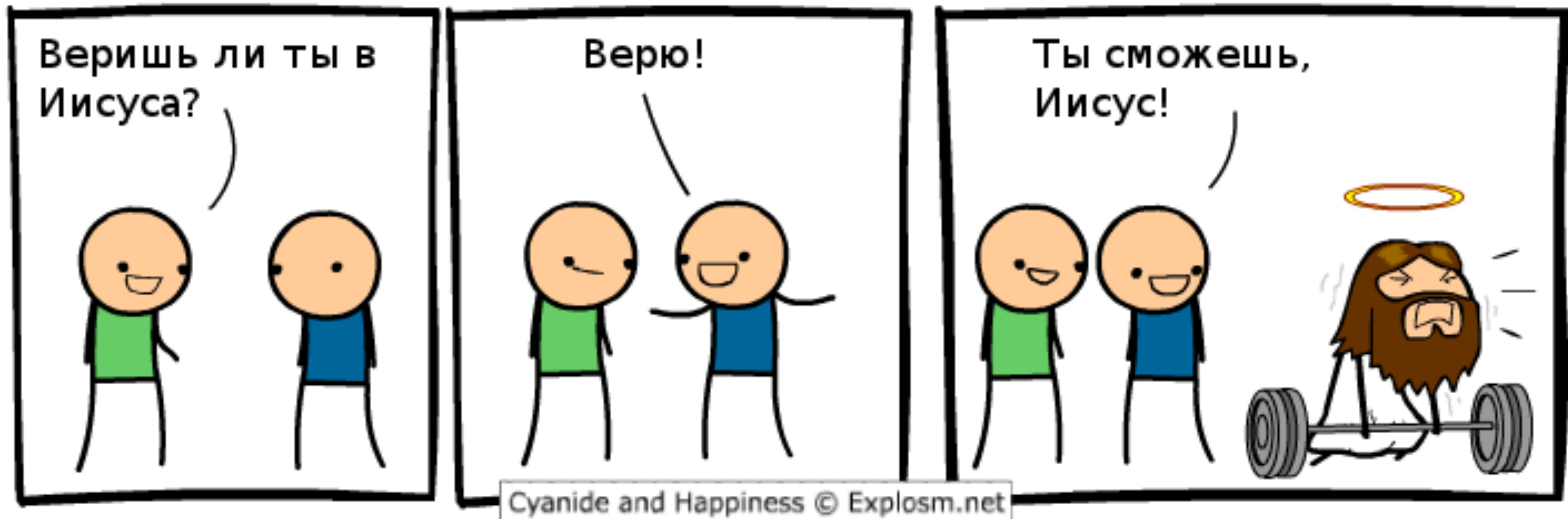
# Сложности

Ключевая проблема обработки текстов - многозначность



# Сложности

Ключевая проблема обработки текстов - многозначность



# Сложности

## Многозначность:

- Морфологическая: «мой», «три»
- Фонетическая: «скрип колеса», «скрипка-лиса»
- Лексическая: «рожа»
- Синтаксическая: «мужу изменять нельзя»

# Текст - уровни абстракции

- буквы
- слова
- нграммы
- предложения
- параграфы
- документы

Все зависит от решаемой задачи

# Токенизация

Токенизация - разбиение текста на токены (смысловые единицы), нужные для конкретной задачи.



# Токенизация

Токенизация - разбиение текста на токены (смысловые единицы), нужные для конкретной задачи.

То есть токенами могут быть слова, предложения, параграфы, ...

# Вначале было слово

Слово - *минимальный* фрагмент текста,  
имеющий смысловую значимость



# Вначале было слово



Слово - **минимальный** фрагмент текста,  
имеющий смысловую значимость

Нахождение границ слов - важная задача.

# Нахождение границ слов

«съешь ещё этих мягких французских булок да выпей чаю»



# Нахождение границ слов

«съешь ещё этих мягких французских булок да выпей чаю»

*words = line.split()*

# Нахождение границ слов

«съешь ещё этих мягких французских булок да выпей чаю»

*words = line.split()*



# Токенизация: особенности языка

«Rindfleischetikettierungsüberwachungsaufgabenübertragungsgesetz»

« Закон о передаче обязанностей контроля маркировки говядины»

# Токенизация: особенности языка

«Rindfleischetikettierungsüberwachungsaufgabenübertragungsgesetz»

« Закон о передаче обязанностей контроля маркировки говядины»

# Токенизация: особенности языка

«Rindfleischetikettierungsüberwachungsaufgabenübertragungsgesetz»

« Закон о передаче обязанностей контроля маркировки говядины»

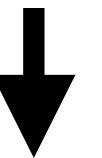


# Токенизация: особенности языка

莎拉波娃现在居住在美国东南部的佛罗里达。

# Токенизация: особенности языка

莎拉波娃现在居住在美国东南部的佛罗里达。



莎拉波娃 现在 居住 在 美国 东南部 的 佛罗里达

«Шарапова сейчас живет во Флориде, к юго-востоку от Соединенных Штатов.»



# Токенизация: пунктуация

- Finland's capital → Finland? Finlands? Finland's ?
- what're, I'm, isn't
- L'ensemble → L? L'? Le?
- 9 a.m, i.e.
- Hewlett-Packard, Немирович-Данченко
- San Francisco, Лос Анджелес, Нью-Васюки
- Анализ твитов - эмоджи ( ; ), :( )

# Токенизация

Нет универсального алгоритма токенизации.

По-хорошему, алгоритм токенизации составляется/  
выучивается *под конкретный корпус*.

Но и split по пробелам часто хватает =)

# Дальнейшая обработка

## Нормализация капитализации

- Us, us → us (местоимение)
- US → US (United States)

# Дальнейшая обработка

## Нормализация капитализации

- Us, us → us (местоимение)
- US → US (United States)

**Можно использовать эвристики:**

- приводить к нижнему регистру слова, стоящие в начале предложения;
- приводить к нижнему регистру слова в заголовках статей
- оставлять капитализацию в словах, стоящих в середине предложения

**Или использовать машинное обучение =)**

# Разбиение на предложения



# Разбиение на предложения

Ну ок, все ж просто: разбиваем по знакам препинания «.;?!»

# Разбиение на предложения

Ну ок, все ж просто: разбиваем по знакам препинания «.;?!»

*В связи с этим первый интервал пробегов был принят равным 350...700 тыс. км. (середина интервала - 525 тыс. км.), второй интервал -- 700...1050 тыс. км. (середина интервала - 875 тыс. км.) и третий интервал 1050...1400 тыс. км. (середина интервала -- 1225 тыс. км.).*



# Эвристики

- Предложение должно содержать буквы
- Предложение должно начинаться с заглавной буквы
- Сокращения из списка требуют «особого внимания» [‘г.’, ‘тыс.’, ‘млн.’, ‘ул.’, ‘км.’, ...]
- Отдельные большие буквы: А.Б. Иванов
- ...

# Эвристики

- Предложение должно содержать буквы
- Предложение должно начинаться с заглавной буквы
- Сокращения из списка требуют «особого внимания» [‘г.’, ‘тыс.’, ‘млн.’, ‘ул.’, ‘км.’, ...]
- Отдельные большие буквы: А.Б. Иванов
- ...

Или можно обучить ML-модель

# Нормализация



# Нормализация



Стемминг

# Нормализация



Стемминг

Лемматизация

# Стемминг

Превращают словоформу в **стем** (псевдооснову),  
отбрасывая псевдоокончание.

«больничных» —> «больничн»

# Стемминг

Превращают словоформу в **стем** (псевдооснову),  
отбрасывая псевдоокончание.

«больничных» —> «больничн»

Бывают эвристические и статистические алгоритмы.

Бессловарный подход

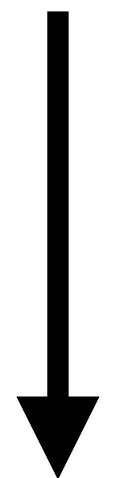
# Стемминг

ты - мой самый смешной друг, которого я знаю



# Стемминг

ты - мой самый смешной друг, которого я знаю



ты - мо сам смешн друг, котор я зна

# Стемминг

ты - мой самый смешной друг, которого я знаю



ты - мо сам смешн друг, котор я зна



я тя крч лю

# Стемминг: эвристики

Использует правила для конкретного языка.  
Примеры: Porter, Snowball stemmer

```
# Endings.

_re_perfective_gerund = re.compile(
    r"(((?P<ignore>[ая])(в|вши|вши́сь))|(и́в|и́вши|и́вши́сь|ы́в|ы́вши|ы́вши́сь))$"
)
_re_adjective = re.compile(
    r"(е́е|и́е|ы́е|о́е|и́ми|ы́ми|е́й|и́й|ы́й|о́й|е́м|и́м|ы́м|о́м|е́го|о́го|е́му|о́му|и́х|ы́х|"
    r"у́ю|ю́ю|а́я|я́я|о́ю|е́ю)$"
)
```

# Porter stemmer rules (1979)

## Step 1a

SSES → SS

IES → I

SS → SS

S →

caresses → caress

ponies → poni

ties → ti

caress → caress

cats → cat

- 5 шагов
- применялся в поиске

## Step 1b

(m>0) EED → EE

feed → feed

(\*v\*) ED →

agreed → agree

plastered → plaster

(\*v\*) ING →

bled → bled

motoring → motor

sing → sing

# Стемминг: вероятностный подход

словарями → словарь → словар-ями → ар-ями

топорами → топор → топор-ами → ор-ами

летящего → лететь → лет-ящего → ет-ящего

летящего → летящий → летяще-го → ящ-его

имя	ра	546
има	ро	154
оѓеющ	те	12
оѓе	ща	12

- Правила строятся на основе статистики по корпусу
- Нужны леммы слов

# В чем преимущество и недостатки стемминга?



# В чем преимущество и недостатки стемминга?

- снижение размерности пространства
- повышает полноту поиска
- снижает точность (одинаковые стемы разных слов)

Левый, левая, лев → лев

# Лемматизация

В больничном дворе стоит небольшой флигель,  
окруженный целым лесом репейника, крапивы и  
дикой конопли.



В больничный двор стоять небольшой флигель  
окружать целый лес репейник крапива и дикий  
конопля

# Лемматизация

**Я ТВОЙ ДОМ**



**ТРУБА ШАТАЛ**



# Лемматизация

**Я ТВОЙ ДОМ**



# Лемматизация

- Словарный подход
- Возможность генерации и склонения несловарных слов

«хрюкотали» —> «хрюкотать»

Популярны:

- mystem
- АОТ
- pymorphy

# Схожесть строковых данных

- коэффициент Жаккара (по словам/нграммам)
- алгоритм шинглов  
(применяется для больших документов)
- расстояние Левенштейна и аналоги

# Расстояние Левенштейна

Минимальное количество операций **вставки, удаления и замены** одного символа на другой, необходимых для превращения одной строки в другую (веса операций могут отличаться).

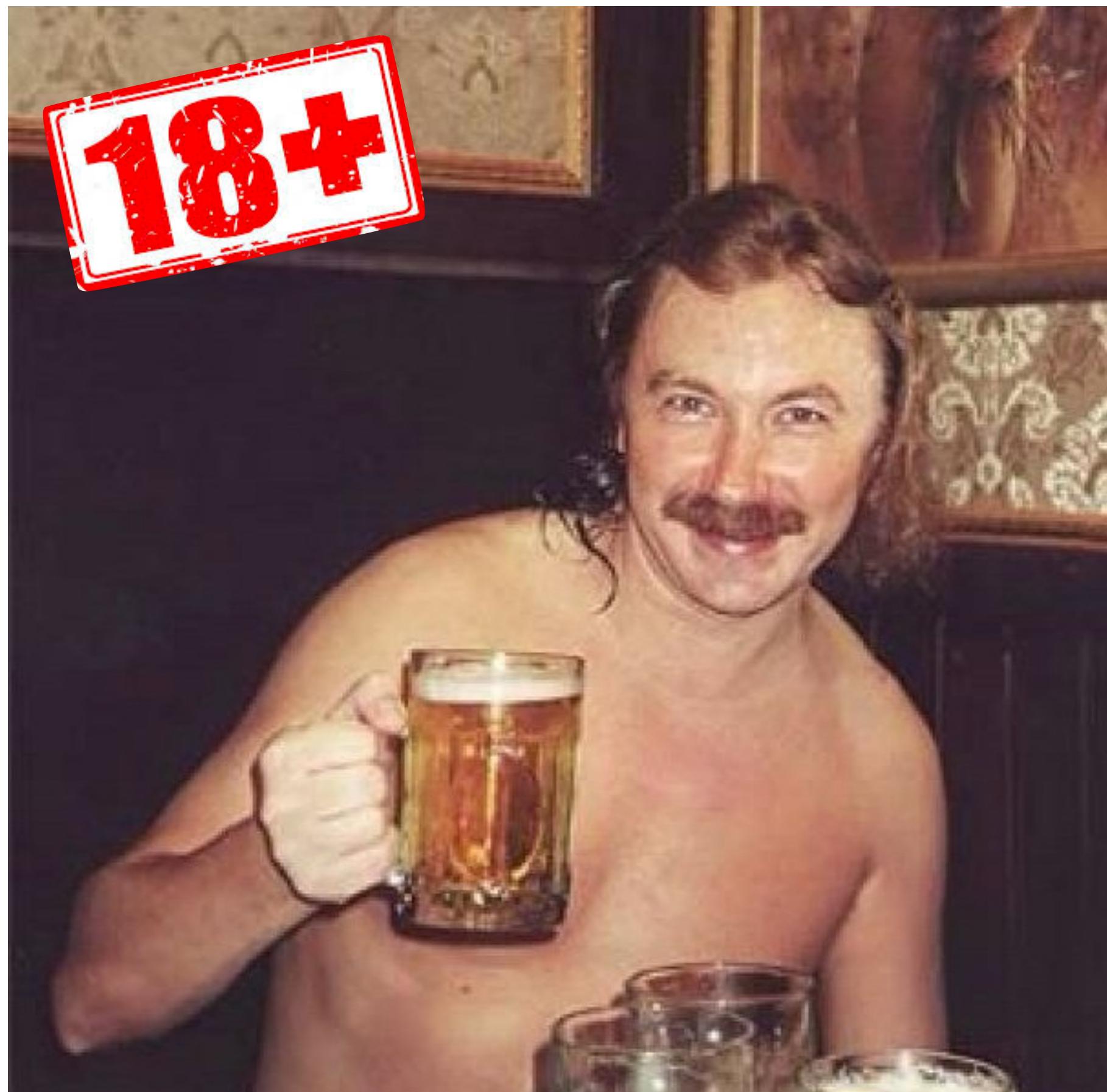
# Расстояние Левенштейна

«Выпьем пива» —> «Выпьем чайку»



# Расстояние Левенштейна

«Выпьем пива» —> «Выпьем чайку»



# Расстояние Левенштейна

Посчитаем, за какое минимальное число вставок, удалений и замены символов можно превратить пиво в чай



# Расстояние Левенштейна

0. Выпьем пива



# Расстояние Левенштейна

0. Выпьем пива
1. Выпьем пивка



# Расстояние Левенштейна

0. Выпьем пива
1. Выпьем пивка
2. Выпьем пивку



# Расстояние Левенштейна

0. Выпьем пива
1. Выпьем пивка
2. Выпьем пивку
3. Выпьем чивку



# Расстояние Левенштейна

0. Выпьем пива
1. Выпьем пивка
2. Выпьем пивку
3. Выпьем чивку
4. Выпьем чийку



# Расстояние Левенштейна

0. Выпьем пива
1. Выпьем пивка
2. Выпьем пивку
3. Выпьем чивку
4. Выпьем чийку
5. Выпьем чайку

$$diff(s_1, s_2) = 5$$



# Векторное представление слов

# Что хотим

Хотим получить такие вектора для слов, чтобы:

- они были плотные
- слова, схожие по смыслу, имели  
схожие векторы

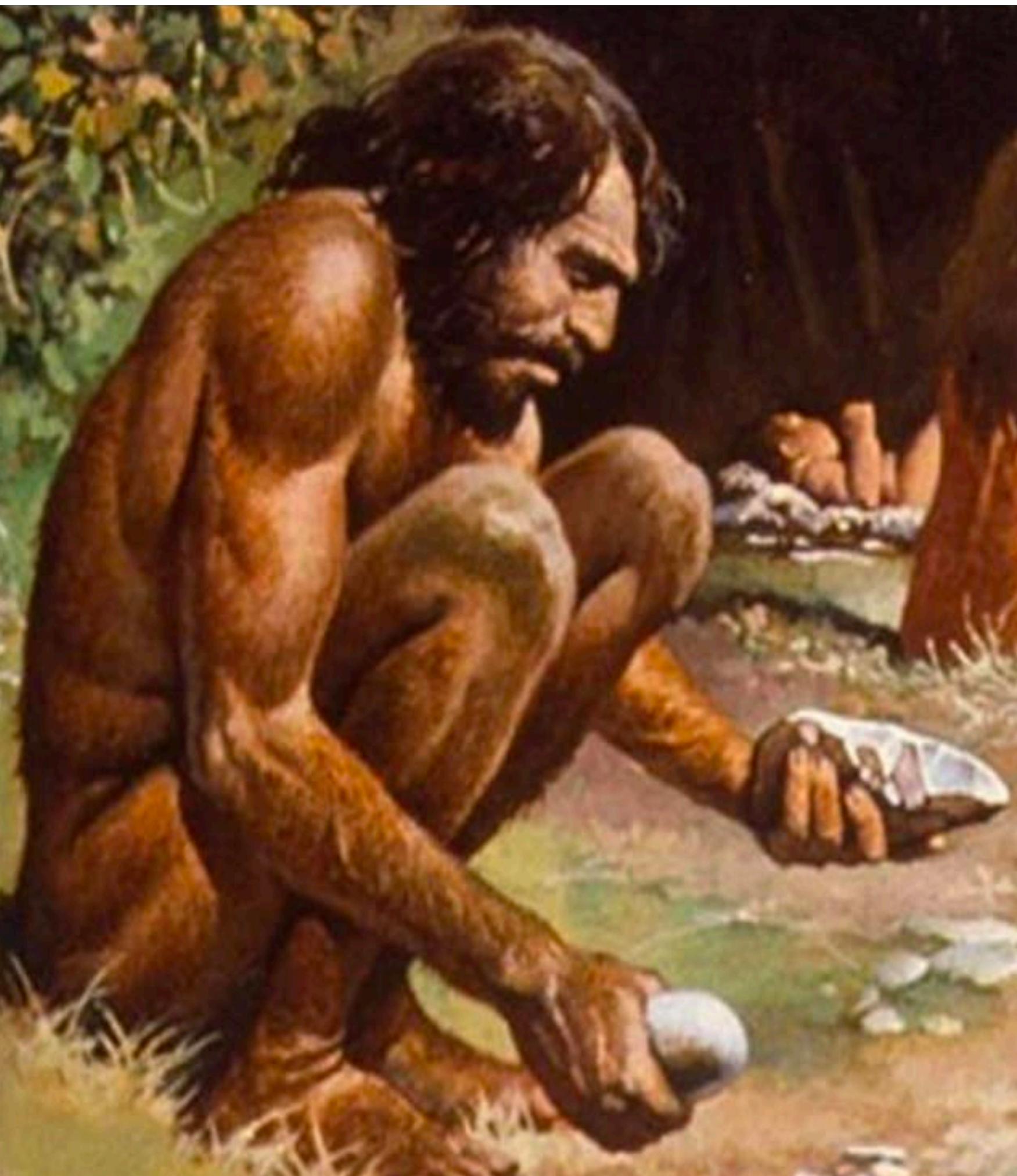
$\text{Vector}(\text{«красивый»}) \approx \text{Vector}(\text{«прекрасный»})$

# Классические подходы

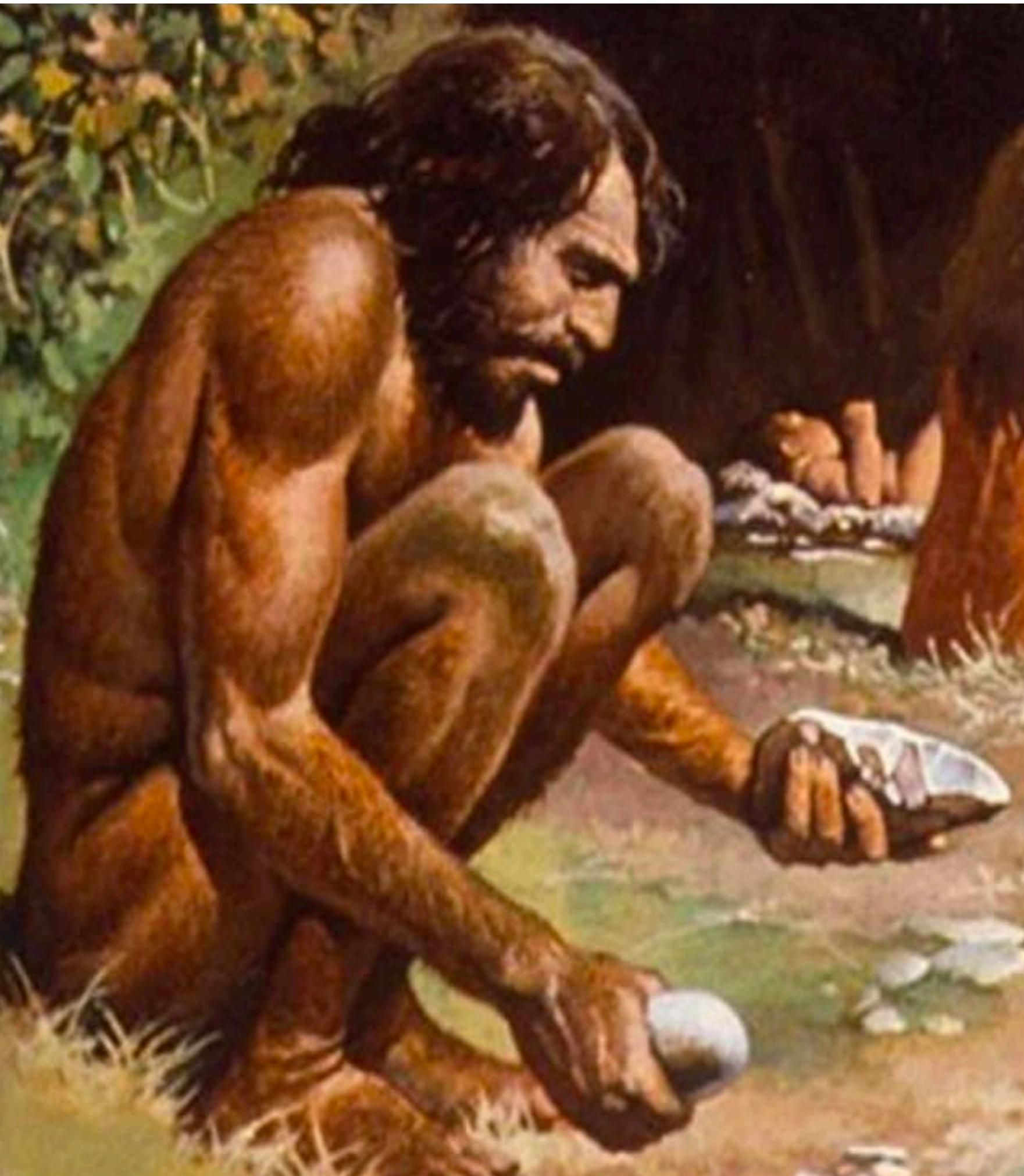


# Что делали раньше

- Понижение ранга матрицы  
«терм-документ»

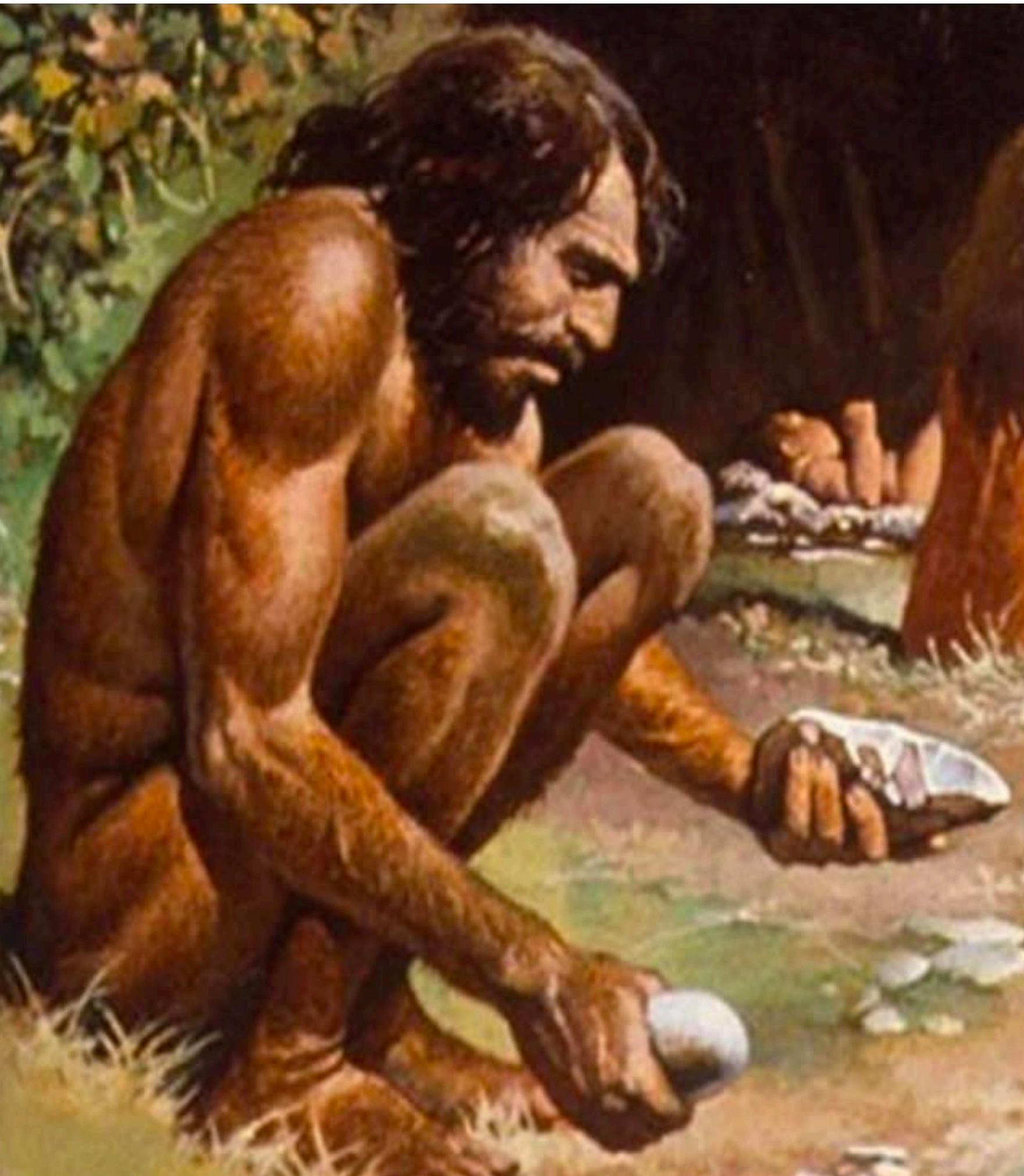


# Что делали раньше



- Понижение ранга матрицы «терм-документ»
- Понижение ранга матрицы совстречаемости

# Что делали раньше



- Понижение ранга матрицы «терм-документ»
- Понижение ранга матрицы совстречаемости
- Понижение ранга матрицы РМI

## Анализ матрицы «терм-документ»

	it	is	puppy	cat	a
it is a puppy	1	1	1	0	1
it is a kitten	1	1	0	0	1
it is a cat	1	1	0	1	1
it is a matrix	1	1	0	0	1

## Анализ матрицы «терм-документ»

	it	is	puppy	cat	a
it is a puppy	1	1	1	0	1
it is a kitten	1	1	0	0	1
it is a cat	1	1	0	1	1
it is a matrix	1	1	0	0	1

# Анализ матрицы «терм-документ»

## Минусы счетчиков:

- слишком большие значения для часто встречающихся слов

можно прикрутить *tf-idf*



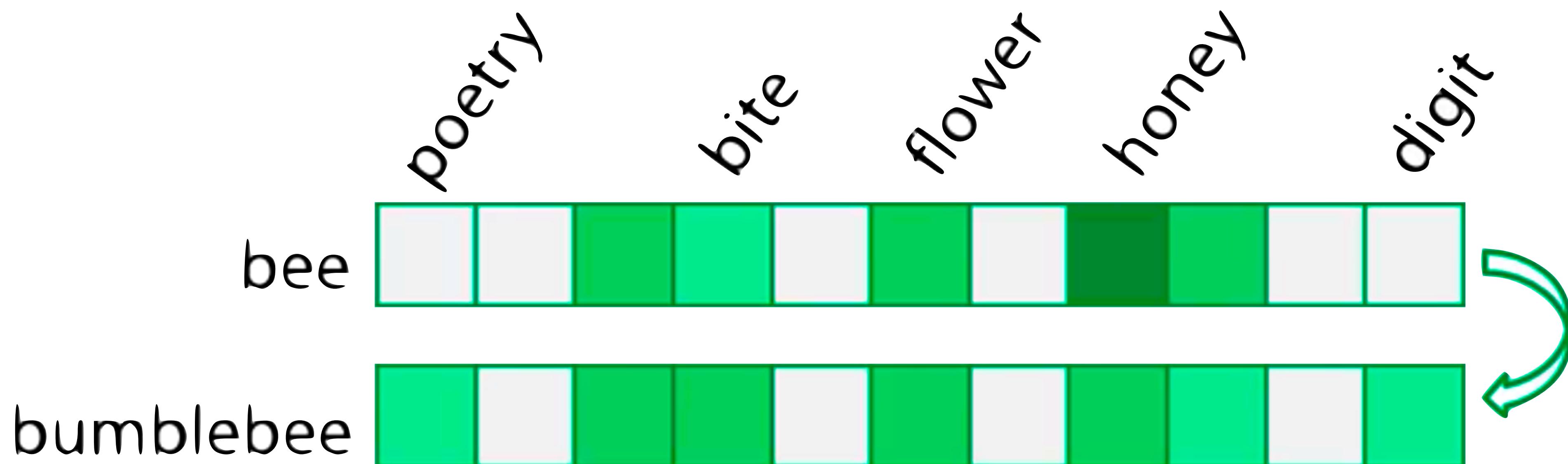
# Анализ матрицы «терм-документ»

## Минусы подхода:

- нужно много документов
- важна “тематическая однородность” документов
- размерность вектора сильно зависит от размера коллекции текстов

# Матрица со-встречаемости

сколько раз слово **оказалось в одном контексте** (например, в окне) с другим словом



# Матрица со-встречаемости

words



	наряд	город	полиция	здание
наряд	x	...	...	...
город	...	x	...	...
полиция	2	1	x	2
здание	...	...	...	...
..				
милиция	3	0	1	4

contexts



# Матрица со-встречаемости

words



	наряд	город	полиция	здание
наряд	x	...	...	...
город	...	x	...	...
полиция	2	1	x	2
здание	...	...	...	...
..				
милиция	3	0	1	4

contexts

У похожих  
слов вектора  
похожи

Недостатки  
тё же

# Матрица РМІ

$$PMI(w, c) = \frac{p(w, c)}{p(w) \cdot p(c)}$$

# Матрица РМІ

$$PMI(w, c) = \frac{p(w, c)}{p(w) \cdot p(c)}$$

А еще лучше:

$$PMI(w, c) = \log\left(\frac{p(w, c)}{p(w) \cdot p(c)}\right)$$

Чему равна РМІ в случае, если слова встречаются **независимо** друг от друга?

# Матрица РМІ

Вероятности можно оценить счетчиками

$$p(w) = \frac{N(w)}{N_{all\_words}}$$

$$p(w, c) = \frac{N(w, c)}{N_{all\_words}}$$

$N_{all\_words}$  - сумма всех элементов матрицы

# Матрица РМI

Вероятности можно оценить счетчиками

$$p(w) = \frac{N(w)}{N_{all\_words}}$$

$$p(w, c) = \frac{N(w, c)}{N_{all\_words}}$$

$$PMI(w, c) = \log\left(\frac{p(w, c)}{p(w) \cdot p(c)}\right)$$



$$PMI(w, c) = \log \frac{N(w, c) \cdot N_{all\_words}}{N(w) \cdot N(c)}$$

$N_{all\_words}$  - сумма всех элементов матрицы

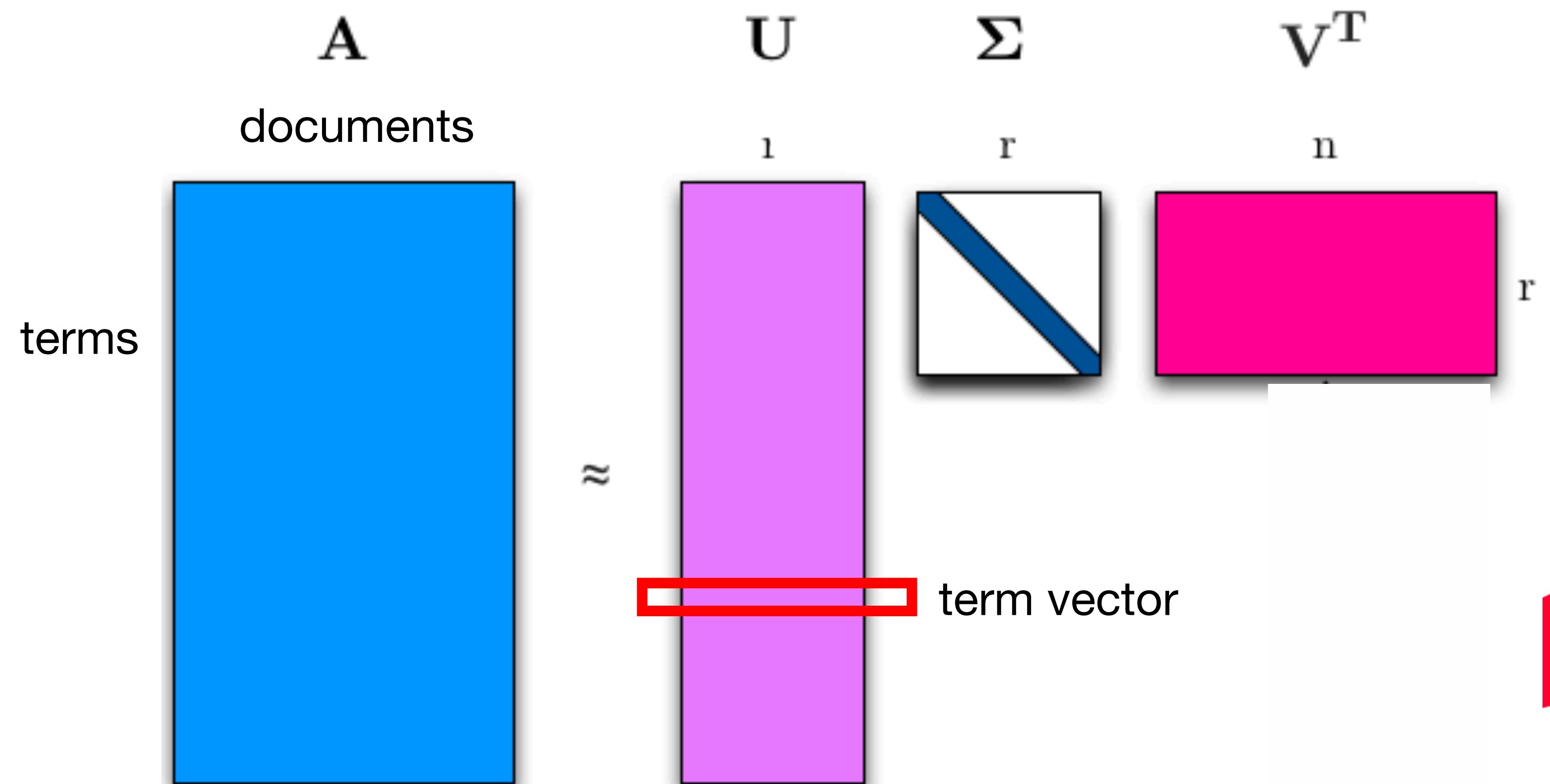
# Positive PMI

Редкие слова:  $\text{PMI} < 0$

$$PPMI(w, c) = \max(0, \log\left(\frac{p(w, c)}{p(w) \cdot p(c)}\right))$$

# Понижение ранга матрицы

- truncated SVD
- PCA

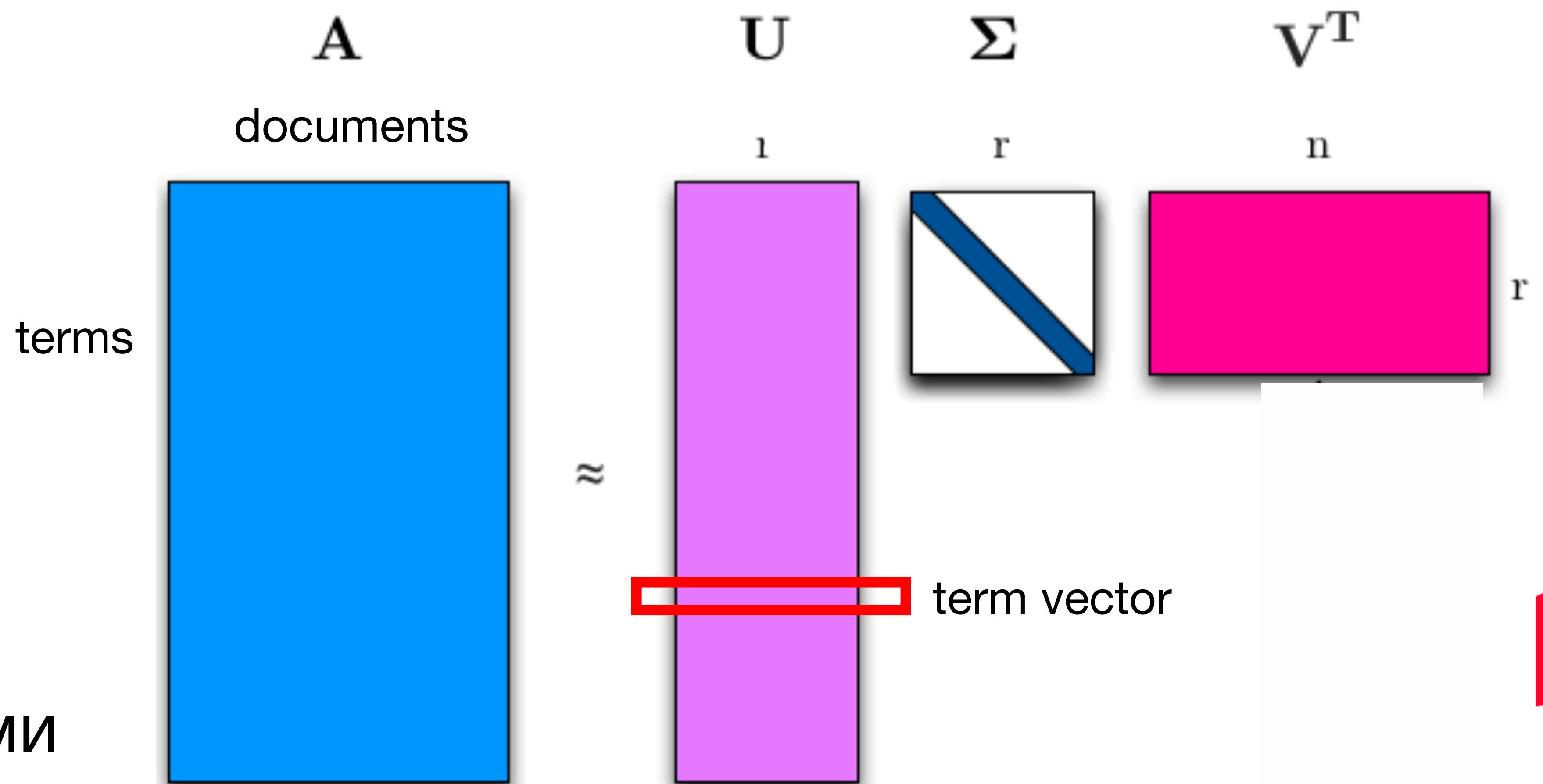


# Понижение ранга матрицы

- truncated SVD
- PCA

## Проблемы:

- вычислительно сложно
- непонятно, как работать с новыми документами



# Дистрибутивная семантика



# Дистрибутивная гипотеза

Смысл слова - распределение над его контекстами

**Джон Ферс:**

«You shall know a  
word by the  
**company it keeps»**



# Дистрибутивная гипотеза

Смысл слова - распределение над его контекстами

Джон Ферс:

«You shall know a word by the company it keeps»



# Дистрибутивная семантика

Я люблю тяжелую и ритмичную X.

Какую X вы предпочитаете слушать  
во время бега ?

Маликов пишет инструментальную X.



# Дистрибутивная семантика

Я люблю тяжелую и ритмичную X.

Какую X вы предпочитаете слушать во время бега ?

Маликов пишет инструментальную X.



# Дистрибутивная семантика

Я люблю тяжелую и ритмичную X.

Какую X вы предпочитаете слушать во время бега ?

Маликов пишет инструментальную X.



**1) Контекст определяет слово**

**2) Слова в похожих контекстах  
“близки по смыслу”**

# Word2vec



# Word2vec

Давайте предсказывать *вероятность слова по его контексту и наоборот*

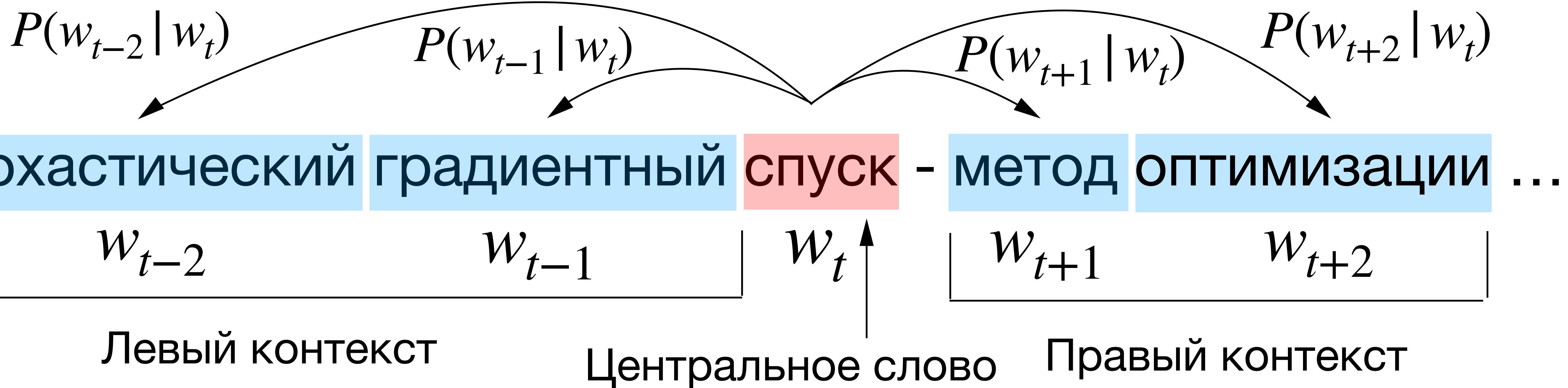
# Word2vec

Давайте предсказывать *вероятность слова по его контексту и наоборот*

Предсказываем слово по контексту -  
**Continuous Bag Of Words**

Предсказываем контекст по центральному слову -  
**Skip-Gram**

# Word2vec: skip-gram model



Для каждого центрального слова  $w_t$  **предсказать слова из контекста** в окне размера  $m$

# Word2vec: objective

Будем так настраивать вектора слов, чтобы максимизировать правдоподобие:

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

# Word2vec: objective

Будем так настраивать вектора слов, чтобы  
максимизировать правдоподобие:

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

# Word2vec: objective

Будем так настраивать вектора слов, чтобы  
максимизировать правдоподобие:

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

Параметры модели

Контекст

# В чем глубинный смысл этого произведения?

# Word2vec: objective

$$L(\theta) = \prod_{t=1}^T \left[ \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} P(w_{t+j} | w_t; \theta) \right]$$

$$P(c | w_t) = P(w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2} | w_t) = \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} P(w_{t+j} | w_t; \theta)$$

# Word2vec: objective

$$L(\theta) = \prod_{t=1}^T \left[ \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} P(w_{t+j} | w_t; \theta) \right]$$

$$P(c | w_t) = P(w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2} | w_t) = \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} P(w_{t+j} | w_t; \theta)$$

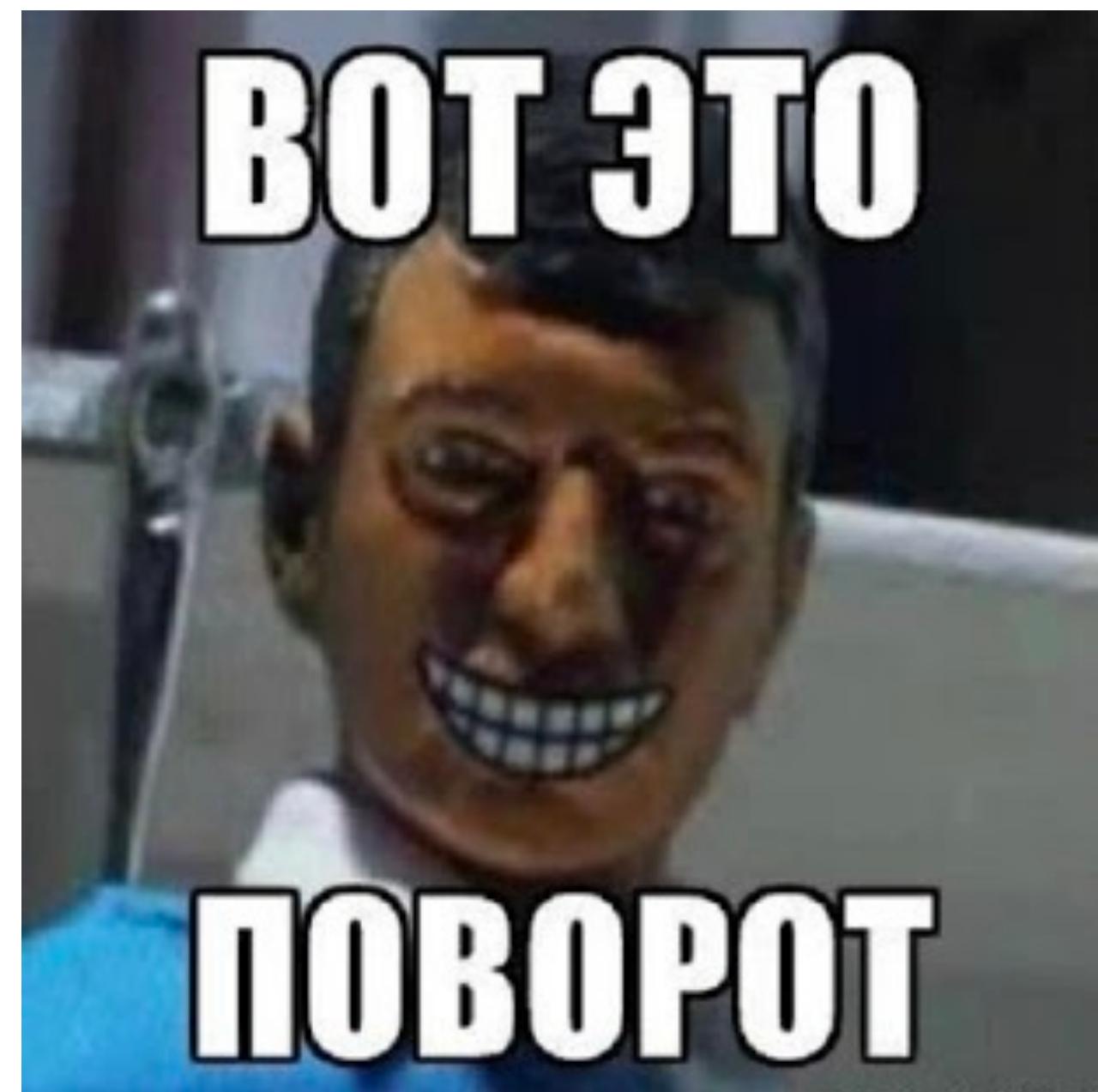
Контекст для каждого центрального слова - BOW !

# Word2vec: objective

$$L(\theta) = \prod_{t=1}^T \left[ \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} P(w_{t+j} | w_t; \theta) \right]$$

$$P(c | w_t) = P(w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2} | w_t) = \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} P(w_{t+j} | w_t; \theta)$$

Контекст для каждого центрального слова - BOW !



# Word2vec: objective

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

Привычнее все-таки минимизировать loss.

# Word2vec: objective

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

Привычнее все-таки минимизировать loss.

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

Как оценить  $P(w_{t+j} | w_t)$  ?

# Word2vec: objective

- Каждому слову соответствует векторное представление
- Каждое слово выступает в двух ипостасях: как центральное, и как контекстное
- Будем использовать для каждого слова два вектора:

$v_w$  - когда слово w - центральное

$u_w$  - когда слово w - контекстное

# Word2vec: objective

- Каждому слову соответствует векторное представление
- Каждое слово выступает в двух ипостасях: как центральное, и как контекстное
- Будем использовать для каждого слова два вектора:

$v_w$  - когда слово w - центральное

$u_w$  - когда слово w - контекстное



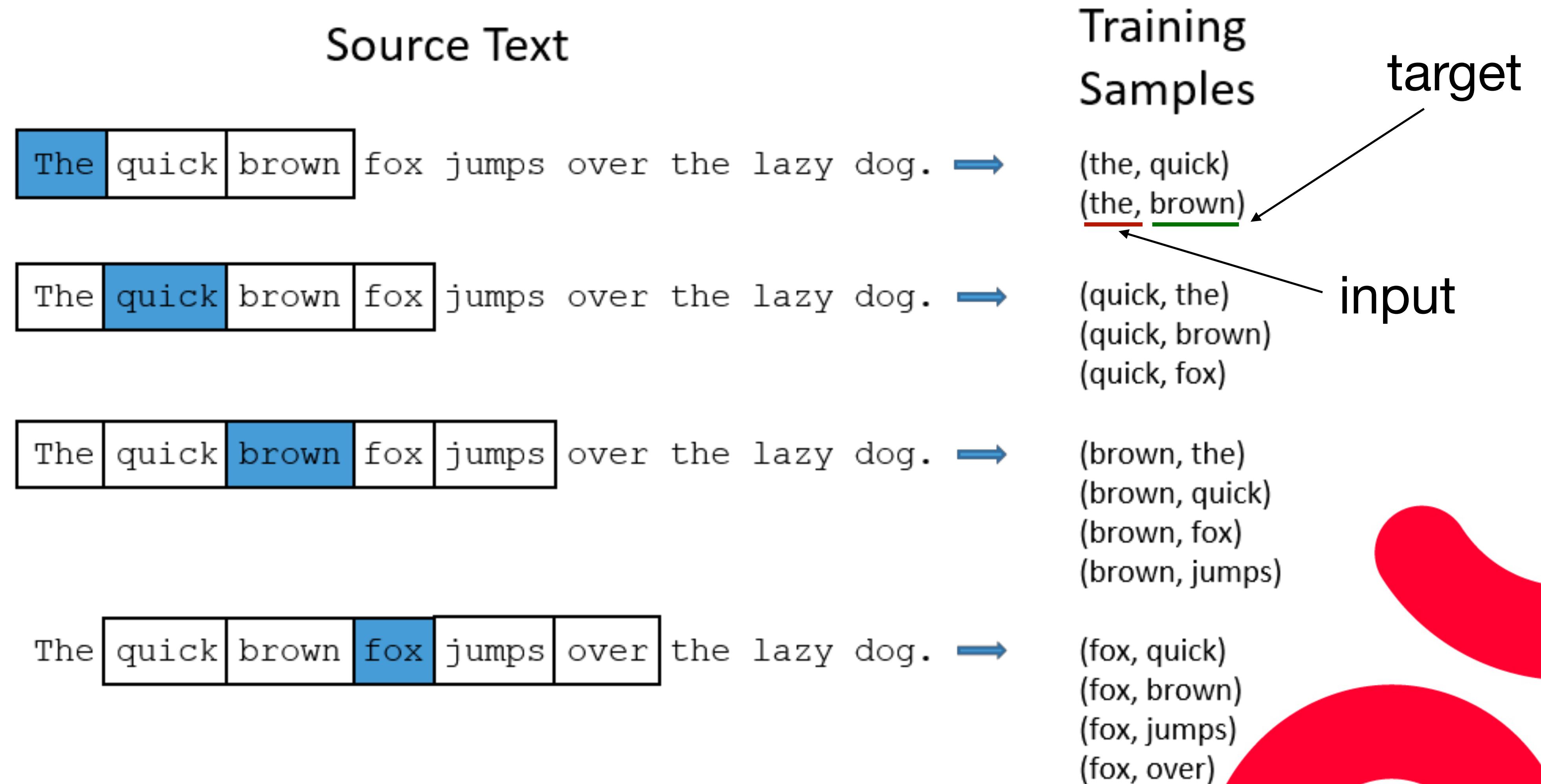
# Word2vec: learn vectors

**как учить вектора для слов?**



# Word2vec: learn vectors

## как учить вектора для слов?



# Word2vec: learn vectors

Все слова превращаем в one-hot вектора

# Word2vec: learn vectors

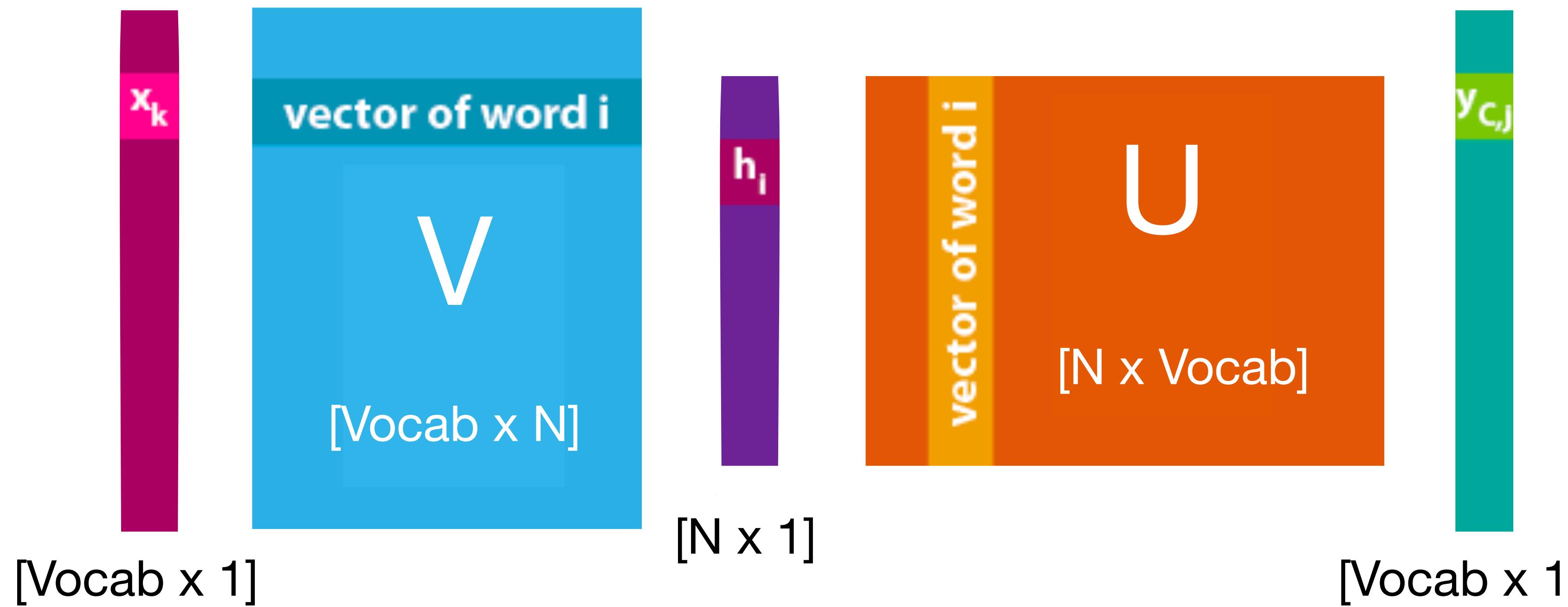
Все слова превращаем в one-hot вектора

А теперь следим за руками...



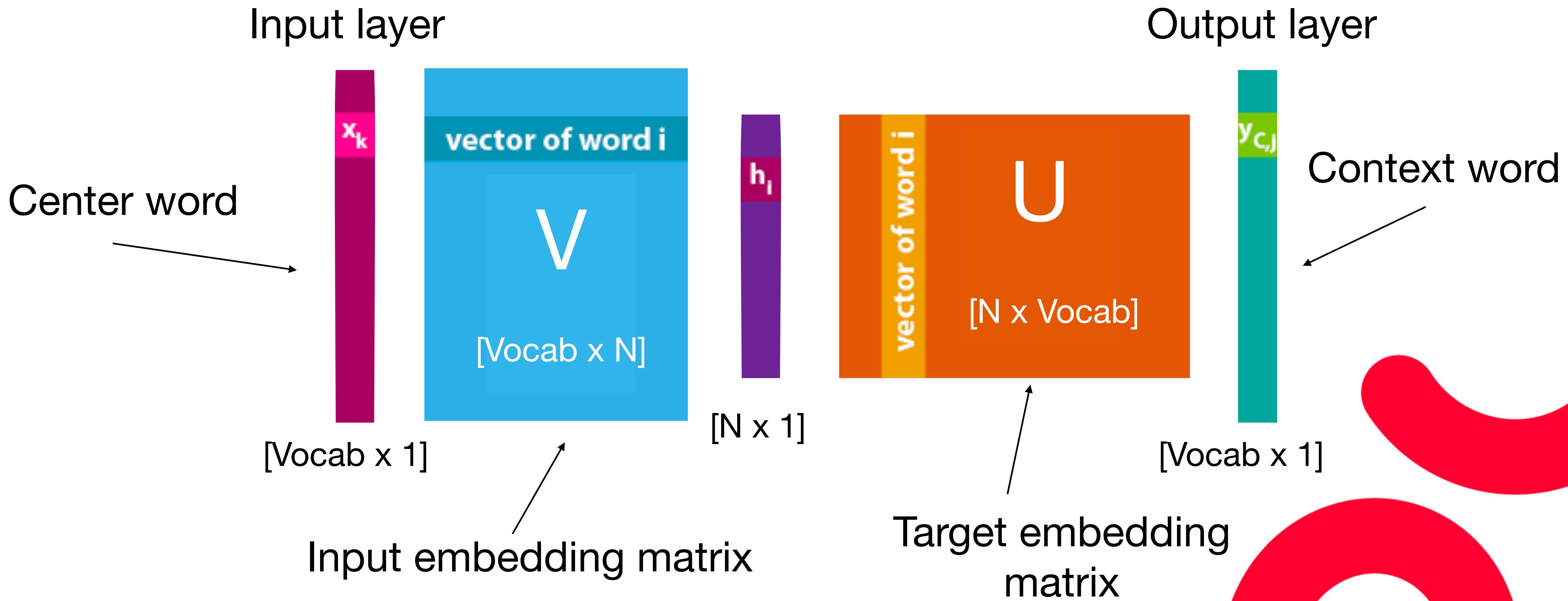
# Word2vec: learn vectors

Все слова превращаем в one-hot вектора



# Word2vec: learn vectors

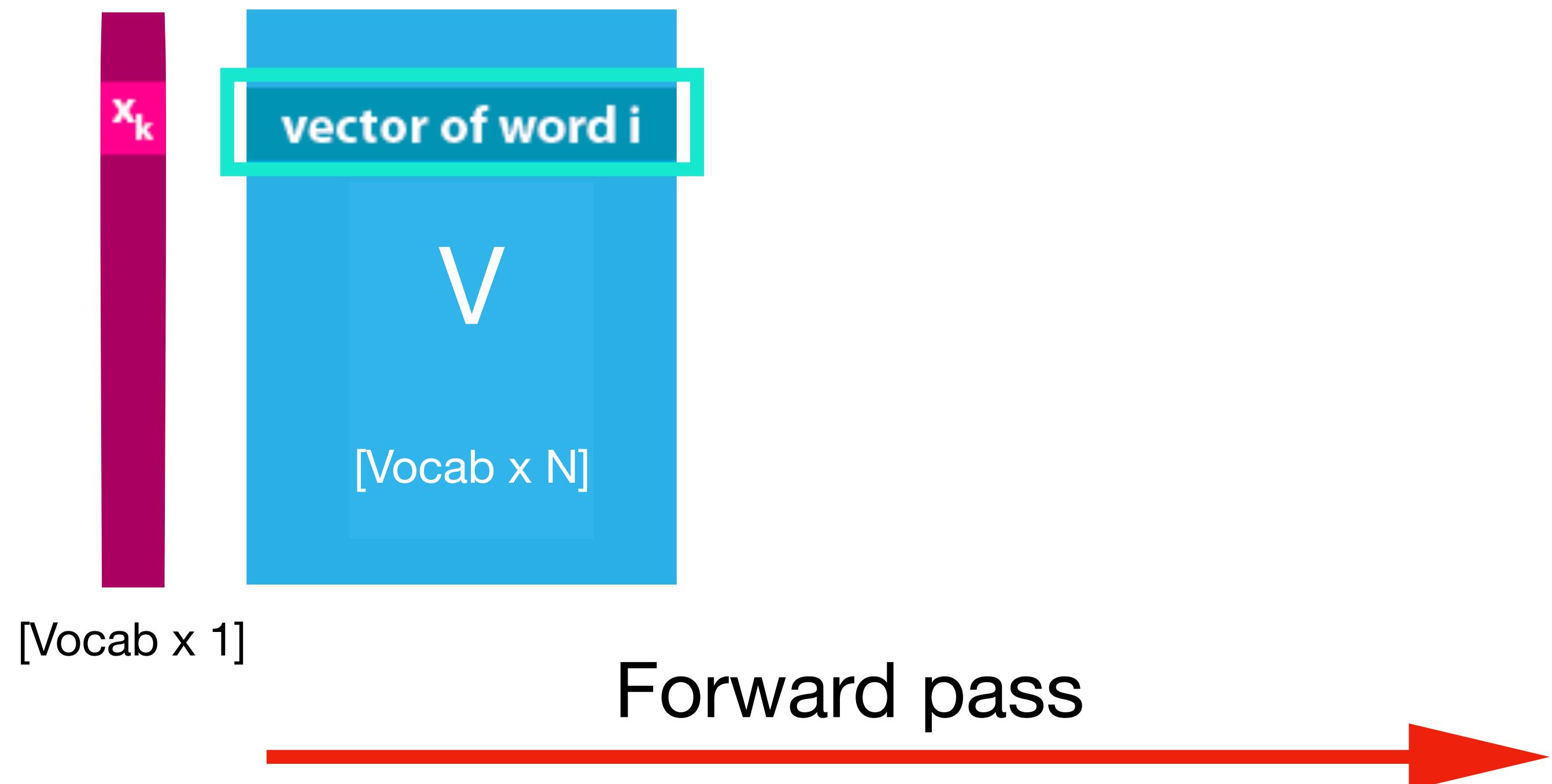
Все слова превращаем в one-hot вектора



# Word2vec: learn vectors

**Forward pass:**

$$h = V^T x$$

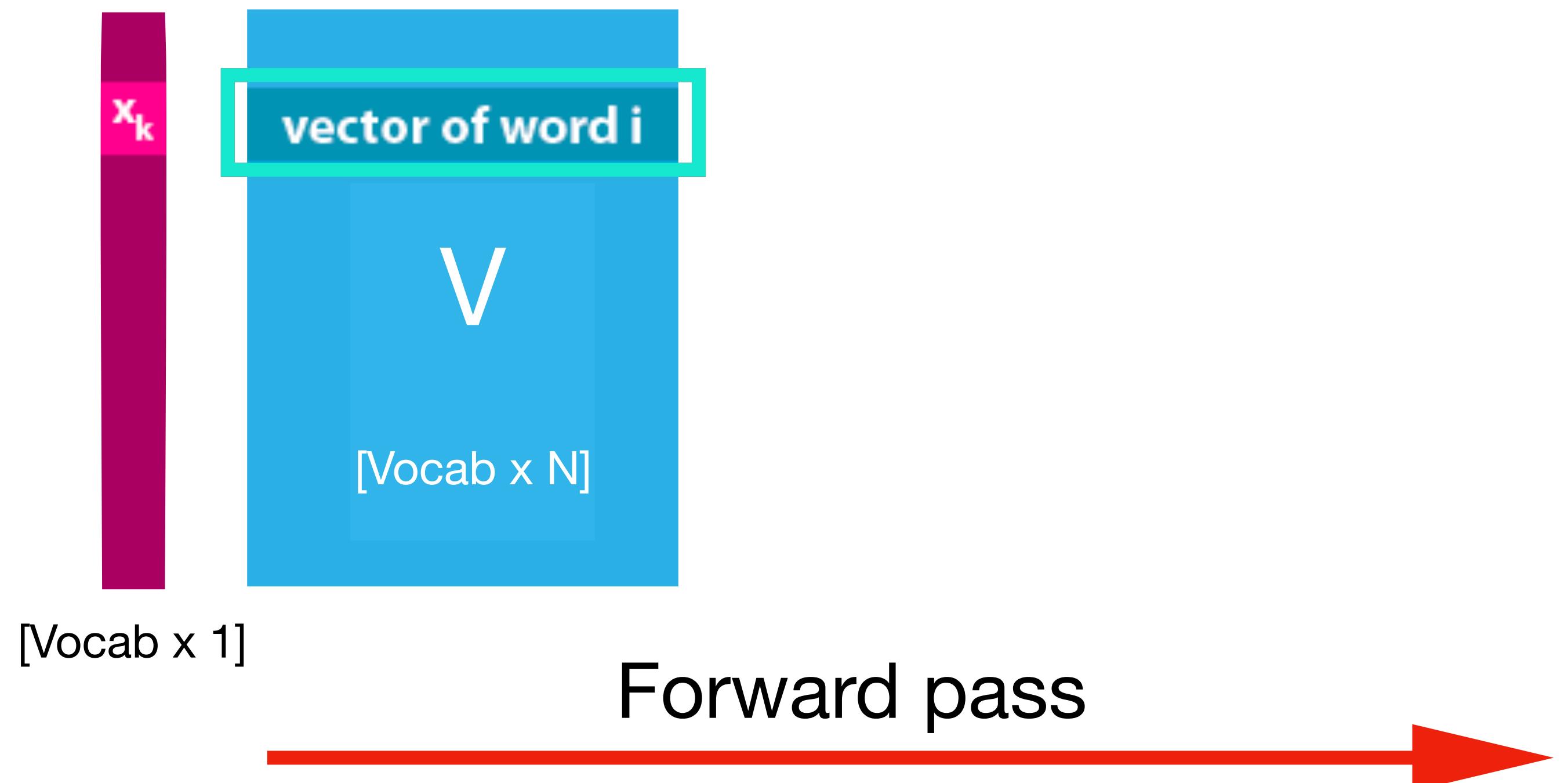


# Word2vec: learn vectors

**Forward pass:**

$$h = V^T x$$

$$h = V_{(k,:)}^T := v_{w_i}^T$$



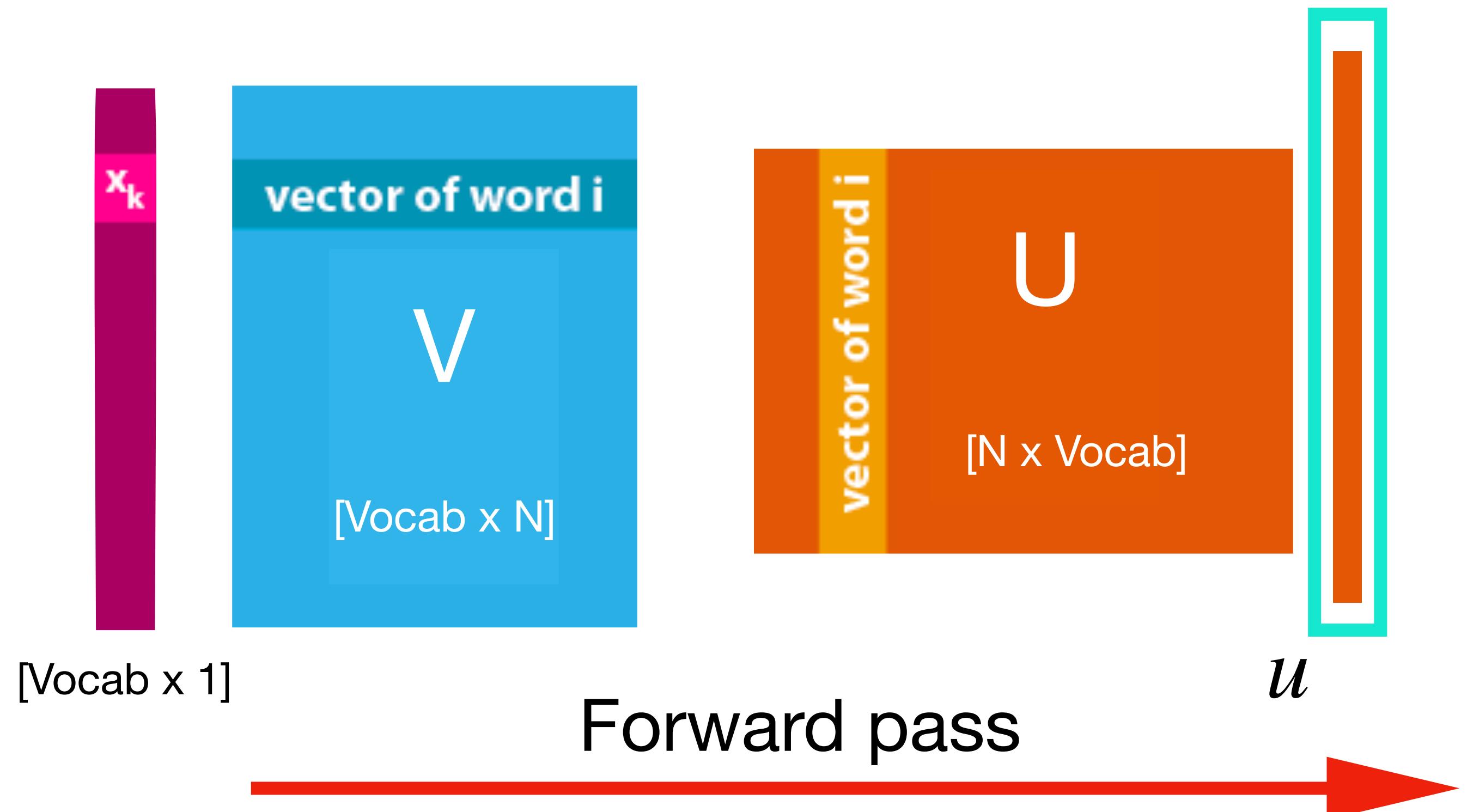
# Word2vec: learn vectors

**Forward pass:**

$$h = V^T x$$

$$h = V_{(k,:)}^T := v_{w_i}^T$$

$$u = U^T h = U^T V^T x$$



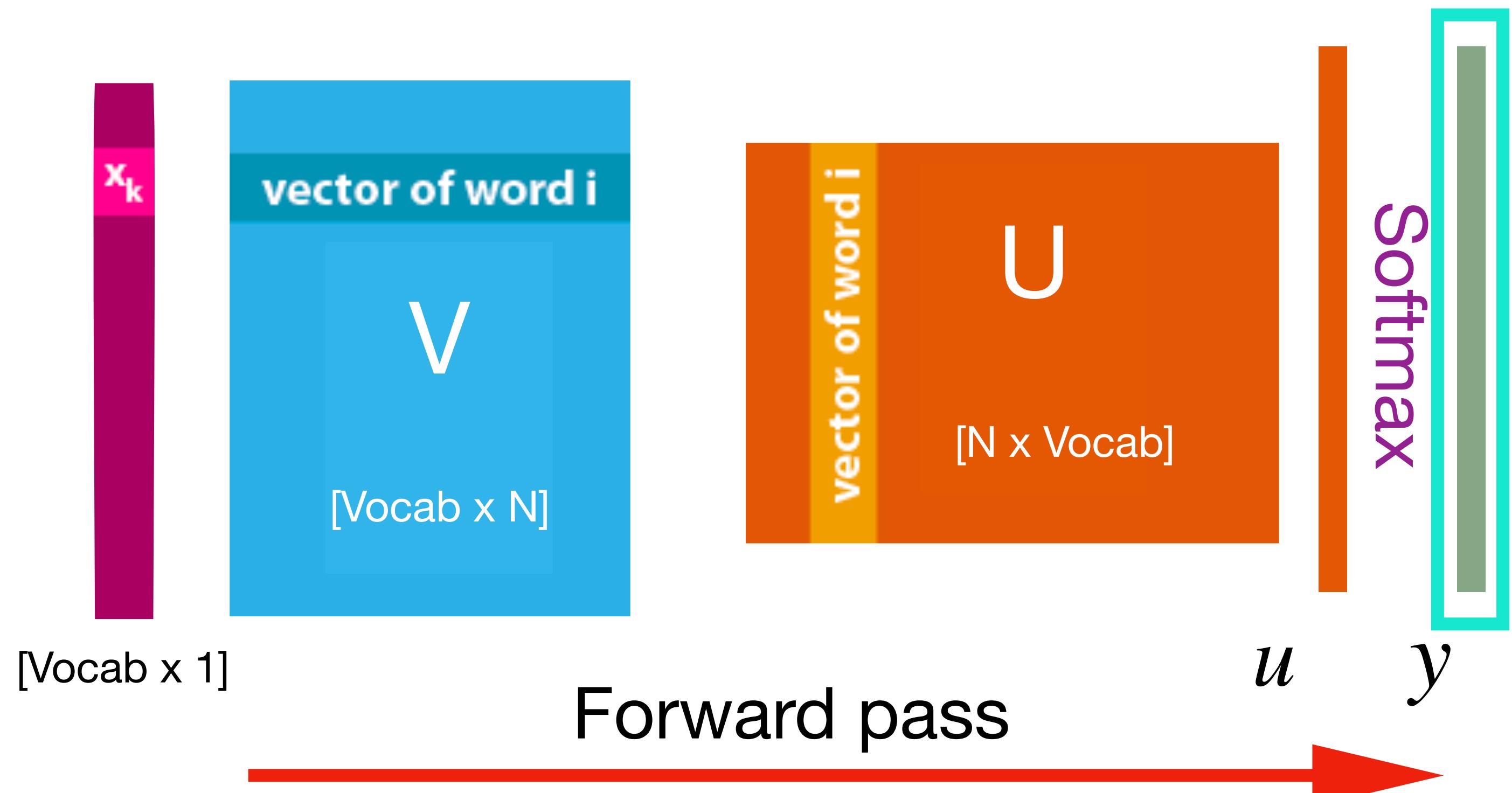
# Word2vec: learn vectors

**Forward pass:**

$$h = V^T x$$

$$h = V_{(k,:)}^T := v_{w_i}^T$$

$$u = U^T h = U^T V^T x$$



# Word2vec: learn vectors

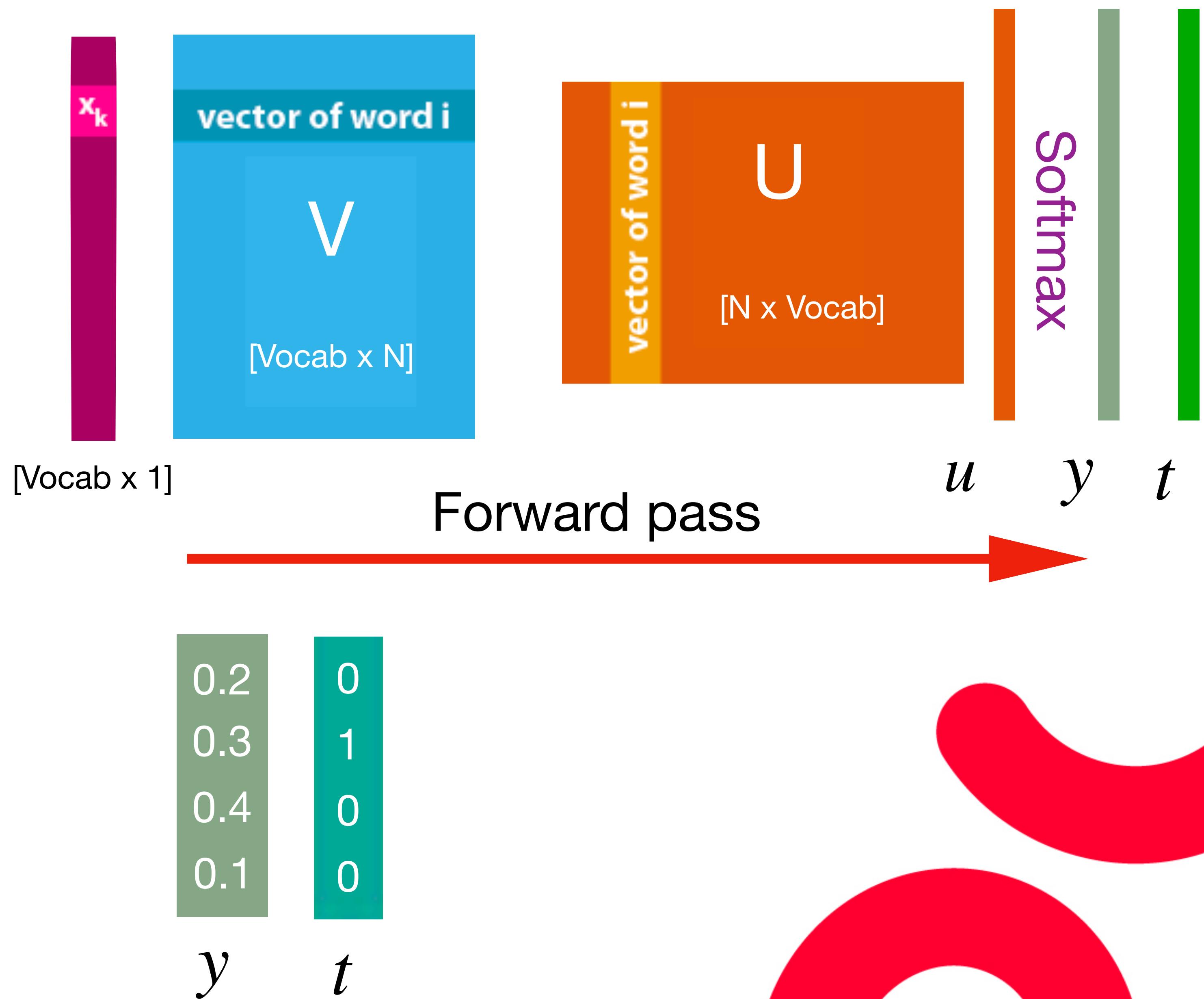
## Forward pass:

$$h = V^T x$$

$$h = V_{(k,:)}^T := v_{w_i}^T$$

$$u = U^T h = U^T V^T x$$

$$y = \text{Softmax}(u)$$



# Word2vec: learn vectors

## Forward pass:

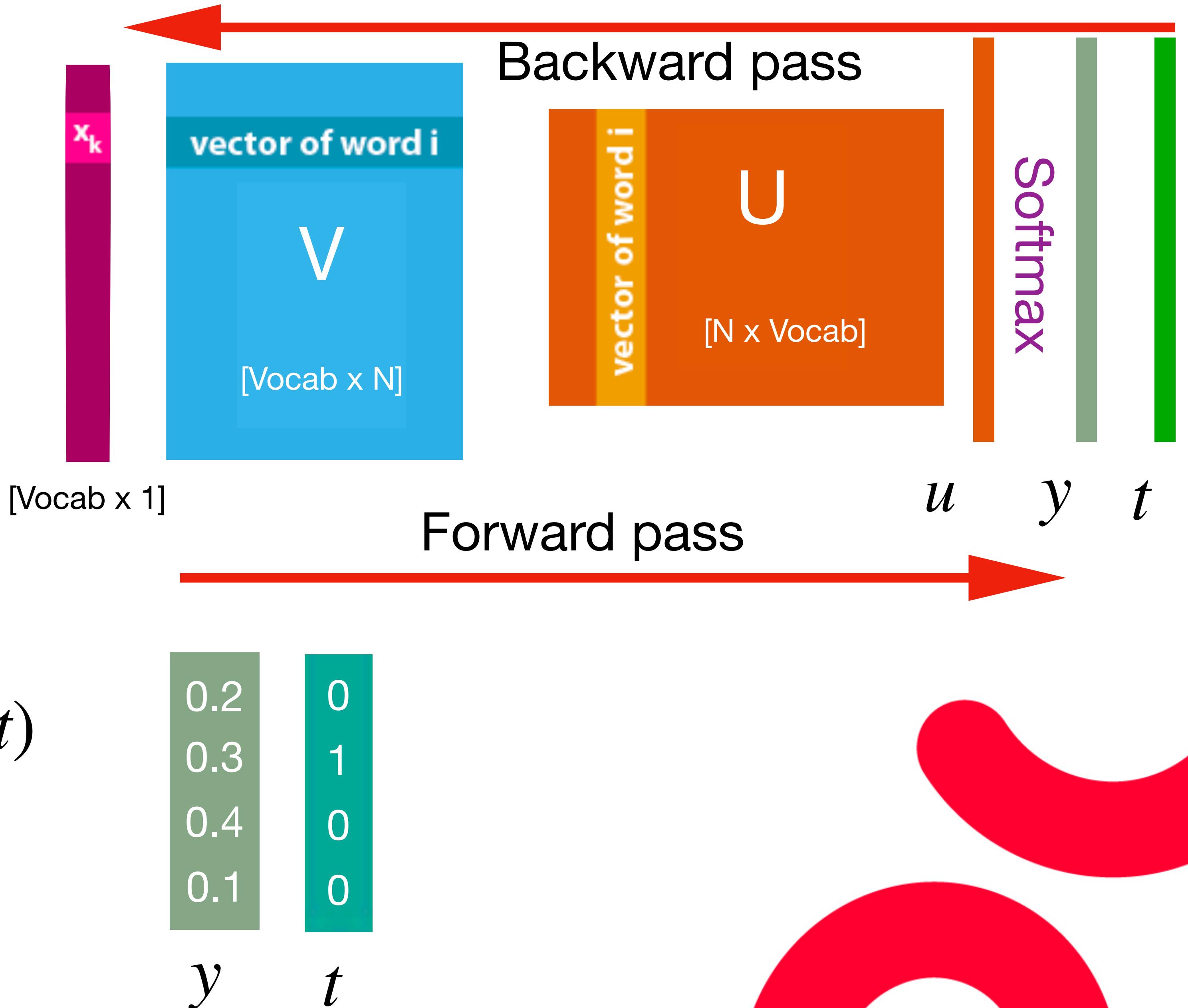
$$h = V^T x$$

$$h = V_{(k,:)}^T := v_{w_i}^T$$

$$u = U^T h = U^T V^T x$$

$$y = \text{Softmax}(u)$$

$$\text{loss} = \text{cross\_entropy}(y, t)$$



# Word2vec: learn vectors

Дальше необходимо посчитать частные производные от loss по каждому из параметров и сделать градиентный шаг

# Word2vec: learn vectors

Дальше необходимо посчитать частные производные от loss по каждому из параметров и сделать градиентный шаг

**Основная идея - chain rule:**

$$u = g(x)$$

$$y = f(u) = f(g(x))$$

$$\frac{dy}{dx} = \frac{dy}{du} \frac{du}{dx}$$

# Word2vec: learn vectors

## Алгоритм обучения:

- Делаем forward pass

# Word2vec: learn vectors

## Алгоритм обучения:

- Делаем forward pass
- Считаем loss между prediction и target

# Word2vec: learn vectors

## Алгоритм обучения:

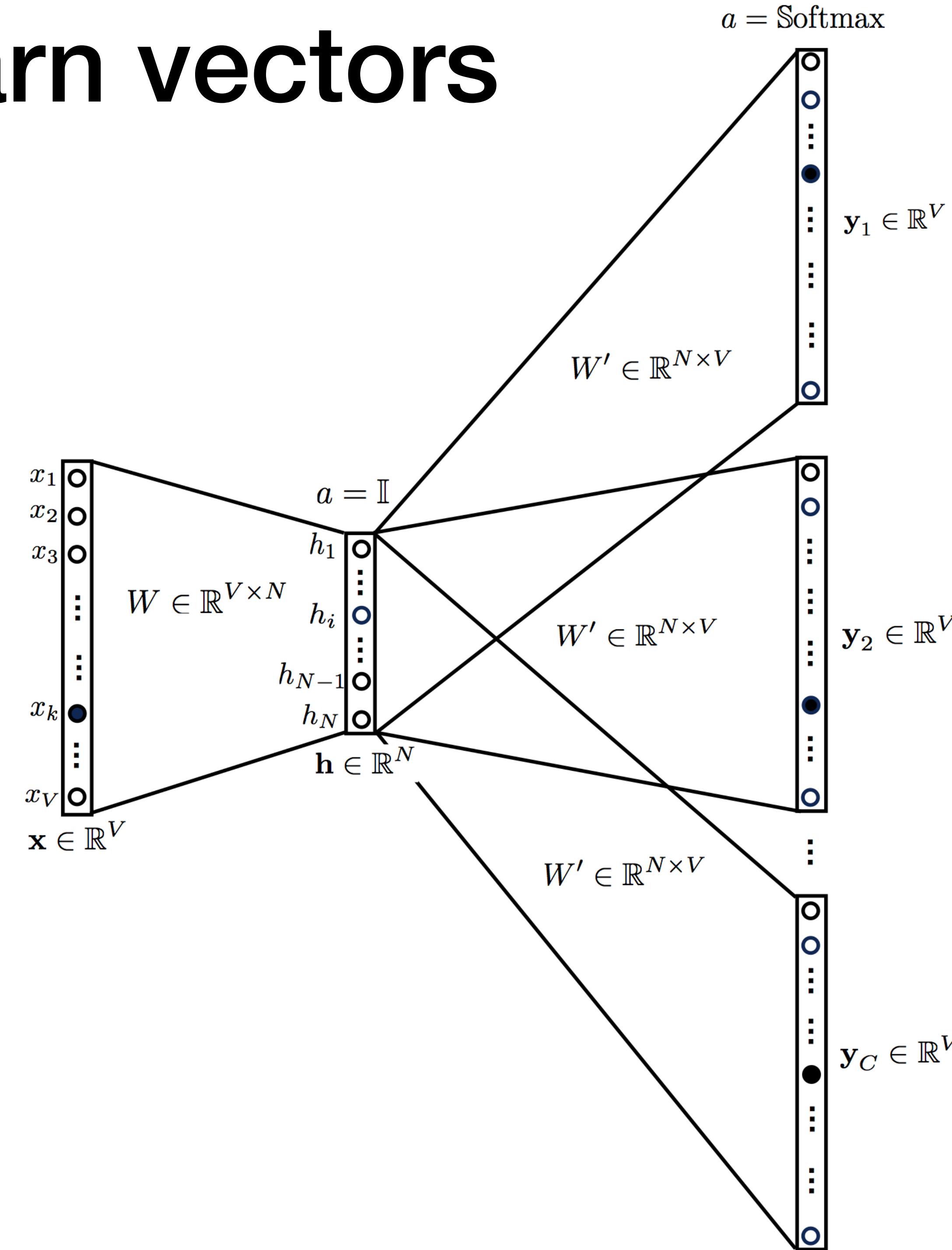
- Делаем forward pass
- Считаем loss между prediction и target
- Делаем backward pass, вычисляем градиенты и обновляем параметры

# Word2vec: learn vectors

$a = \text{Softmax}$

Можно модифицировать  
процедуру обучения сразу для  
всех контекстных слов.

И делать backpropagation,  
учитывая loss сразу по всем  
словам из контекста



# Word2vec: learn vectors

Но для каждого слова есть 2 вектора...

# Word2vec: learn vectors

Но для каждого слова есть 2 вектора...

Как правило, эти вектора складывают, получая итоговый результирующий вектор

$$v_{w_i}^{full} = v_{w_i} + u_{w_i}$$

# Word2vec: probability estimation

Имея вектора слов, можем их сравнивать.

Например, с помощью скалярного произведения:

$u_c^T \cdot v_t$  - чем больше, тем выше вероятность встретить  $w_c$  в контексте  $w_t$

# Word2vec: probability estimation

Имея вектора слов, можем их сравнивать.

Например, с помощью скалярного произведения:

$u_c^T \cdot v_t$  - чем больше, тем выше вероятность встретить  $w_c$  в контексте  $w_t$

Вспомним, как определена вероятность:

$$P(w_c | w_t) = \frac{\exp(u_c^T \cdot v_t)}{\sum_{w \in V} \exp(u_w^T \cdot v_t)}$$

# Word2vec: probability estimation

$$P(w_c | w_t) = \frac{\exp(u_c^T \cdot v_t)}{\sum_{w \in V} \exp(u_w^T \cdot v_t)}$$

# Word2vec: probability estimation

$$P(w_c | w_t) = \frac{\exp(u_c^T \cdot v_t)}{\sum_{w \in V} \exp(u_w^T \cdot v_t)}$$

Суммирование по всему словарю

# Word2vec: probability estimation



$$P(w_c | w_t) = \frac{\exp(u_c^T \cdot v_t)}{\sum_{w \in V} \exp(u_w^T \cdot v_t)}$$

Суммирование по всему словарю

# Word2vec: probability estimation



$$P(w_c | w_t) = \frac{\exp(u_c^T \cdot v_t)}{\sum_{w \in V} \exp(u_w^T \cdot v_t)}$$

Суммирование по всему словарю

**Есть несколько стратегий по эффективному вычислению/  
аппроксимации софтмакса**

Мы остановимся на Negative Sampling

# Word2vec: negative sampling

**Идея:**

Давайте вместо того, чтобы считать честные вероятности, решать задачу  
**бинарной классификации:**

# Word2vec: negative sampling

Идея:

Давайте вместо того, чтобы считать частные вероятности, решать задачу  
**бинарной классификации:**

Насэмплируем  $k$  рандомных слов **не из контекста  $\mathcal{W}_t$** , и будем учить  
модель **отличать контекстные слова от рандомно выбранных**

# Word2vec: negative sampling

Идея:

Давайте вместо того, чтобы считать честные вероятности, решать задачу  
**бинарной классификации:**

Насэмплируем  $k$  рандомных слов **не из контекста**  $\mathcal{W}_t$ , и будем учить  
модель **отличать контекстные слова от рандомно выбранных**

Будем максимизировать:

$$\log \sigma(u_c^T \cdot v_t) + \sum_{i=1}^k \mathbb{E}_{w_i \sim P_n(w)} [\log \sigma(-u_{w_i}^T \cdot v_t)]$$

# Word2vec: negative sampling

$$\log \sigma(u_c^T \cdot v_t) + \sum_{i=1}^k \mathbb{E}_{w_i \sim P_n(w)} [\log \sigma(-u_{w_i}^T \cdot v_t)]$$

$k = 5-20$  - для небольших датасетов;

$k = 2-5$  - для больших датасетов

# Word2vec: negative sampling

$$\log \sigma(u_c^T \cdot v_t) + \sum_{i=1}^k \mathbb{E}_{w_i \sim P_n(w)} [\log \sigma(-u_{w_i}^T \cdot v_t)]$$

$k = 5-20$  - для небольших датасетов;

$k = 2-5$  - для больших датасетов

$P_n(w)$  - распределение, из которого сэмплируем слова

$P_n(w) \approx U(w)^{3/4}$  - smoothed unigram distribution

# Word2vec: negative sampling

$$\log \sigma(u_c^T \cdot v_t) + \sum_{i=1}^k \mathbb{E}_{w_i \sim P_n(w)} [\log \sigma(-u_{w_i}^T \cdot v_t)]$$

**Что интересно:**

Можно показать, что skip-gram model with negative sampling делает разложение shifted PMI матрицы.

SVD-разложение для PMI-матрицы и SGNS дают похожие результаты при правильном подборе параметров.

# Word2vec: subsampling of frequent words

При сэмплировании есть **проблема частотных слов**: чаще попадают в контекст

# Word2vec: subsampling of frequent words

При сэмплировании есть **проблема частотных слов**: чаще попадают в контекст

$$P(w_i) = \sqrt{\frac{t}{f(w_i)}}$$

- вероятность **взять** слово (более агрессивно выбрасывает слова, частота которых больше чем t)

$f(w_i)$  - частота слова     $t$  - порог

Интуиция: вектора слишком частотных слов не должны сильно изменяться в процессе обучения

# Word2vec: learning phrases

$$v_{New\ York\ Times} \neq v_{New\ York} + v_{Times}$$

# Word2vec: learning phrases

$$v_{New\ York\ Times} \neq v_{New\ York} + v_{Times}$$

Найдем фразы с помощью:

$$\text{score}(w_i, w_j) = \frac{\text{count}(w_i w_j) - \delta}{\text{count}(w_i) \times \text{count}(w_j)}.$$

сount - соответствующие счетчики

$\delta$  - дисконтирующий коэффициент

# Word2vec: learning phrases

$$v_{New\ York\ Times} \neq v_{New\ York} + v_{Times}$$

Найдем фразы с помощью:

$$\text{score}(w_i, w_j) = \frac{\text{count}(w_i w_j) - \delta}{\text{count}(w_i) \times \text{count}(w_j)}.$$

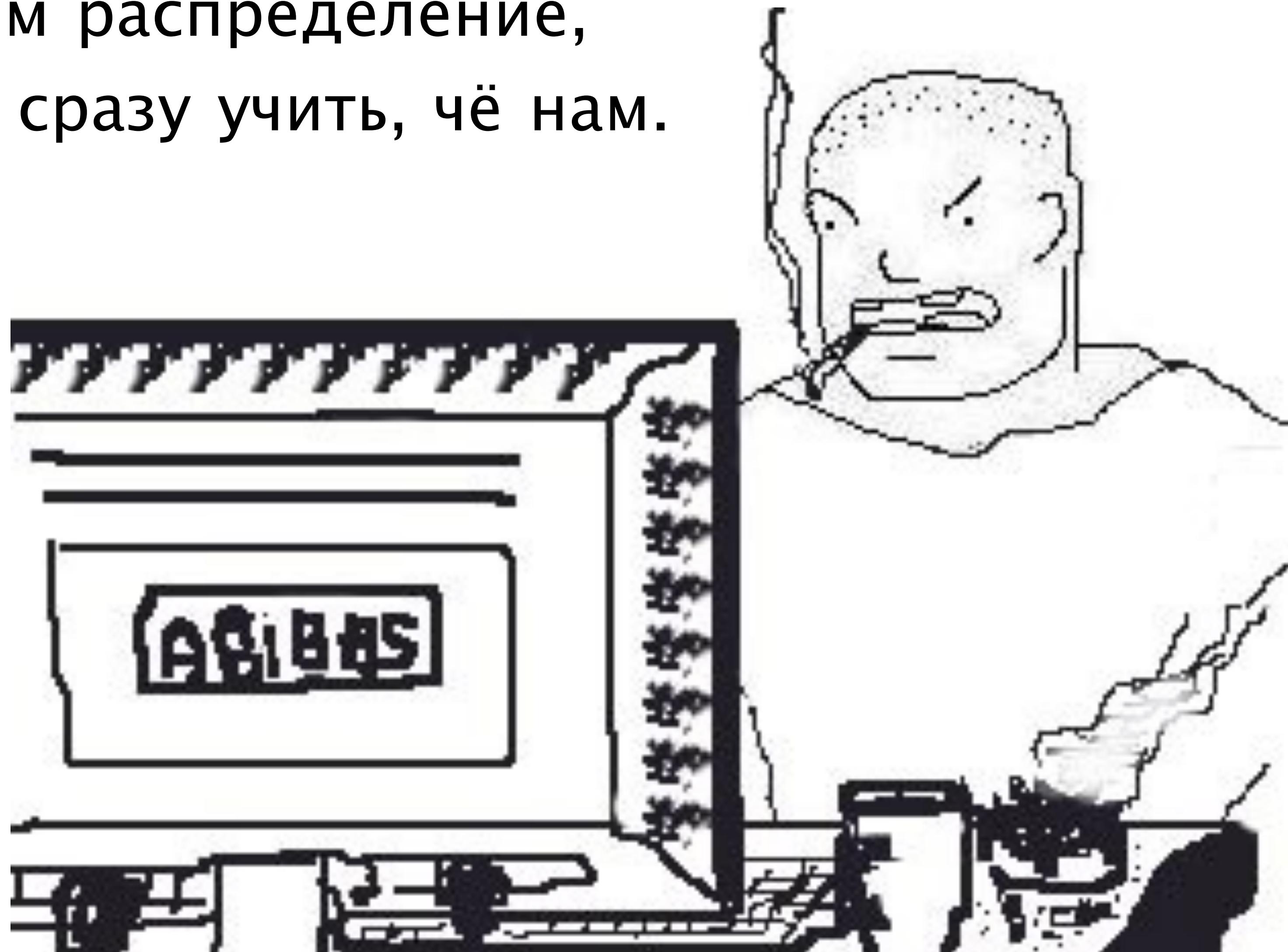
сount - соответствующие счетчики

$\delta$  - дисконтирующий коэффициент

И будем учить **эмбеддинги для фразы целиком**

# Word2vec - это просто!

Так, падажжи... два вектора для слова, negative sampling,  
туда-сюда... еще сгладим распределение,  
вектора для фраз будем сразу учить, чё нам.  
так падажжи



# Word2vec: text vectors

**Как получить вектора для текстов,  
если есть вектора для слов?**

- 1) Можно просто усреднить вектора слов в тексте. Это работает неплохо для сравнительно коротких текстов
- 2) Можно посчитать взвешенное среднее векторов слов с весами tf-idf

# Как оценить качество векторов?

- **Intrinsic evaluation** («внутренняя» оценка):
  - Оценка «в вакууме»
  - На какой-либо вспомогательной подзадаче
  - Стого говоря, не коррелирует с метриками на внешних задачах
  - Можно быстро проверить

# Как оценить качество векторов?

- **Intrinsic evaluation** («внутренняя» оценка):
  - Оценка «в вакууме»
  - На какой-либо вспомогательной подзадаче
  - Строго говоря, не коррелирует с метриками на внешних задачах
  - Можно быстро проверить
- **Extrinsic evaluation** («внешняя» оценка):
  - Оценка на реальной задаче
  - Может занимать продолжительно время
  - Если работает плохо, то непонятно, что именно

# Intrinsic evaluation: analogies

Оценка того, насколько вектора слов знают про **семантические и синтаксические аналогии** между словами



# Intrinsic evaluation: analogies

Оценка того, насколько вектора слов знают про **семантические и синтаксические аналогии** между словами

$$\boxed{a:b :: c:?"}$$

man:woman :: king:?



$$d = \arg \max_i \frac{(x_b - x_a + x_c)^T x_i}{\|x_b - x_a + x_c\|}$$

# Intrinsic evaluation: analogies

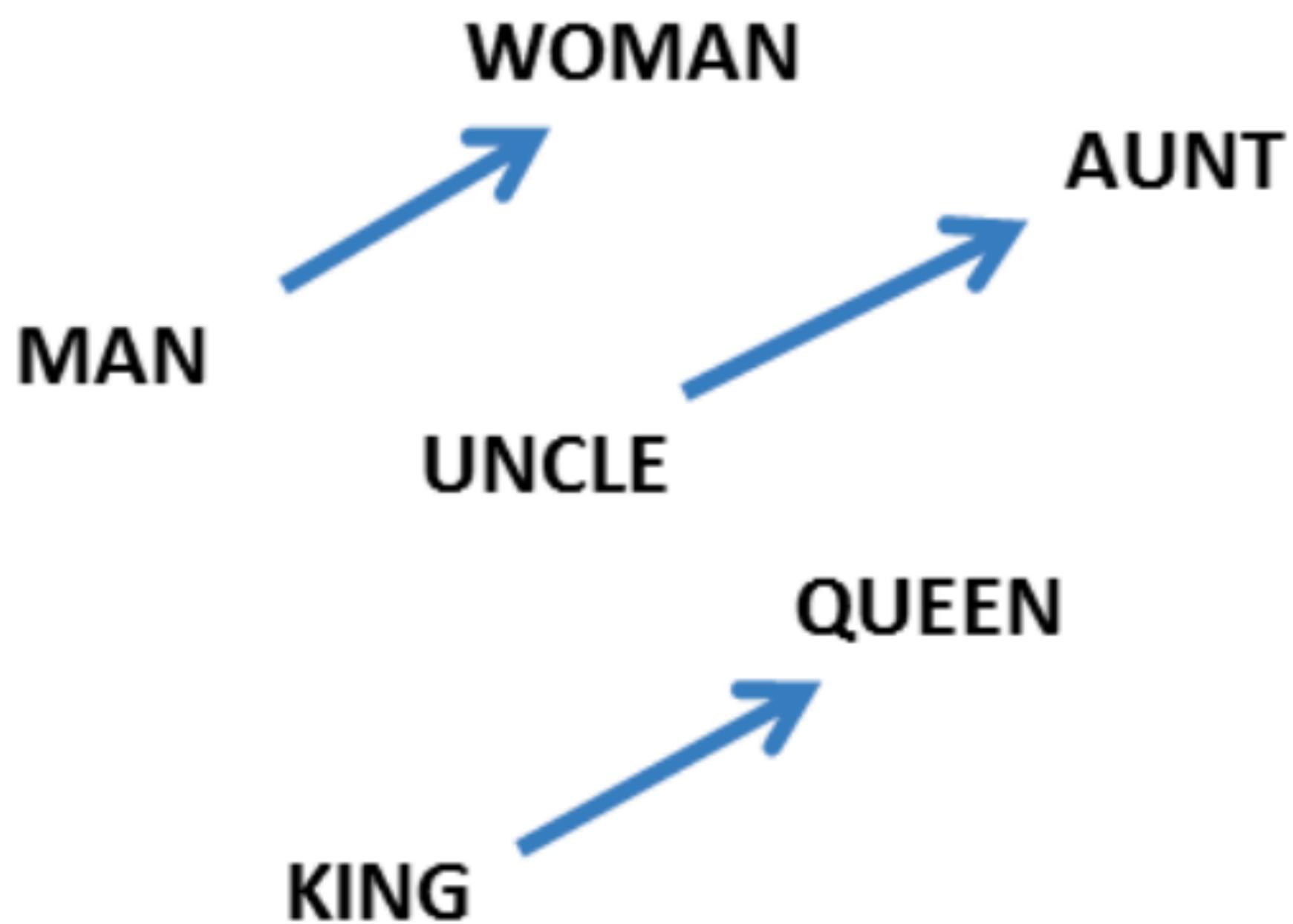
Оценка того, насколько вектора слов знают про **семантические и синтаксические аналогии** между словами

$$\boxed{a:b :: c: ?}$$

→

$$\boxed{d = \arg \max_i \frac{(x_b - x_a + x_c)^T x_i}{\|x_b - x_a + x_c\|}}$$

man:woman :: king:?



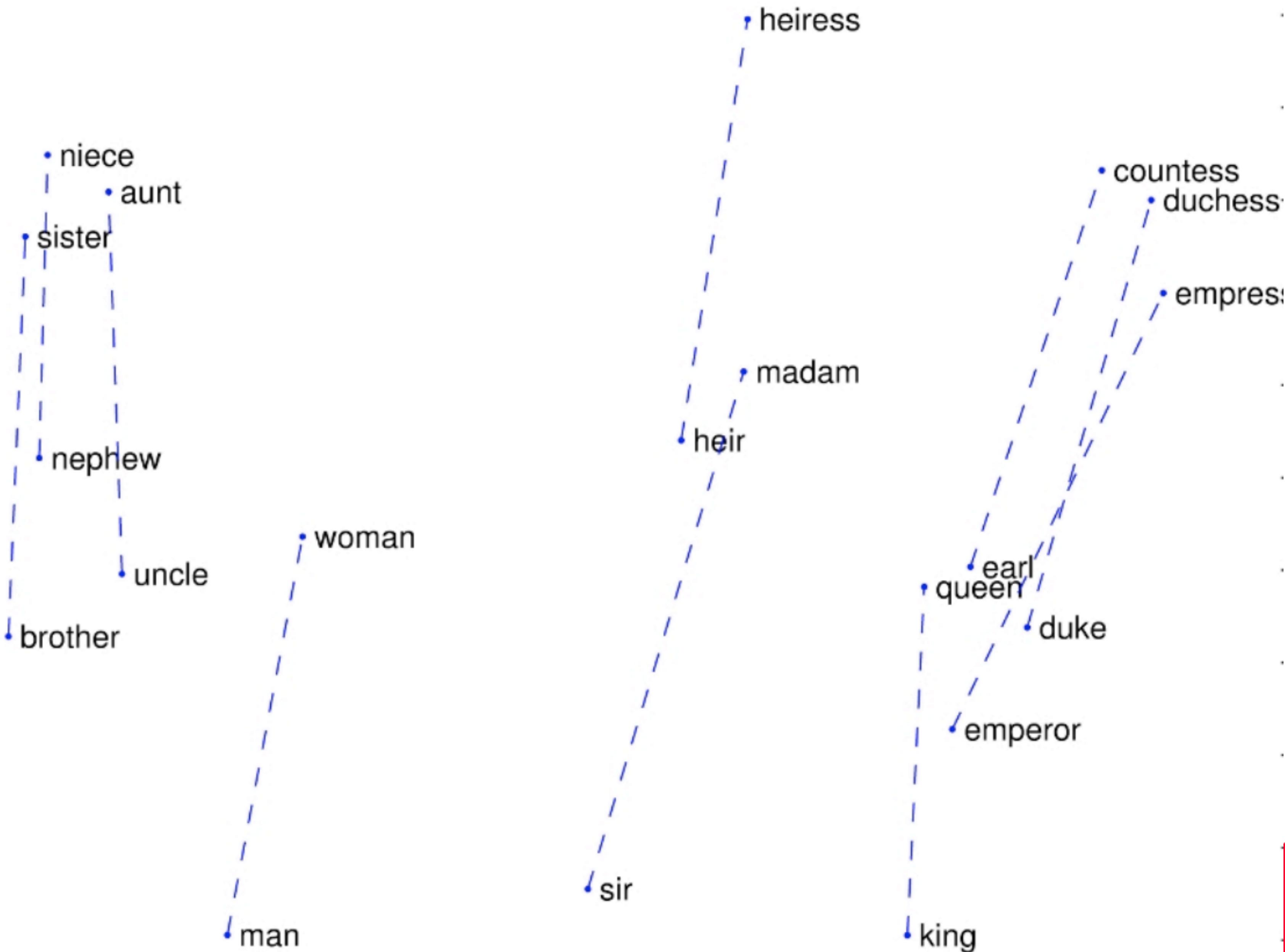
# Intrinsic evaluation: analogies

Компоненты векторов, к сожалению, неинтерпретируемы

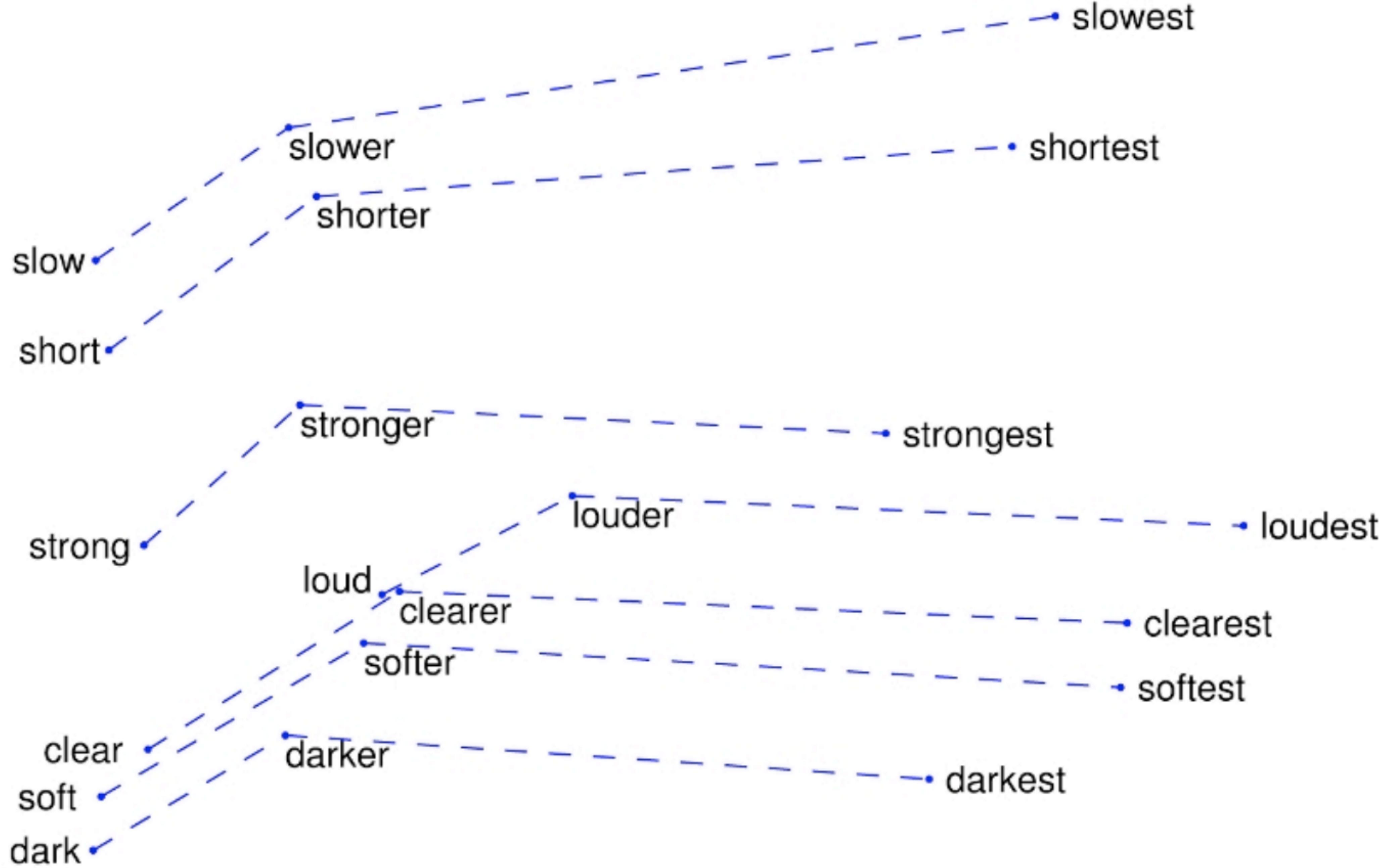
А было бы здорово...



# Intrinsic evaluation: analogies



# Intrinsic evaluation: analogies



# Intrinsic evaluation: analogies

<i>Expression</i>	<i>Nearest token</i>
Paris - France + Italy	Rome
bigger - big + cold	colder
sushi - Japan + Germany	bratwurst
Cu - copper + gold	Au
Windows - Microsoft + Google	Android
Montreal Canadiens - Montreal + Toronto	Toronto Maple Leafs

# Intrinsic evaluation: analogies

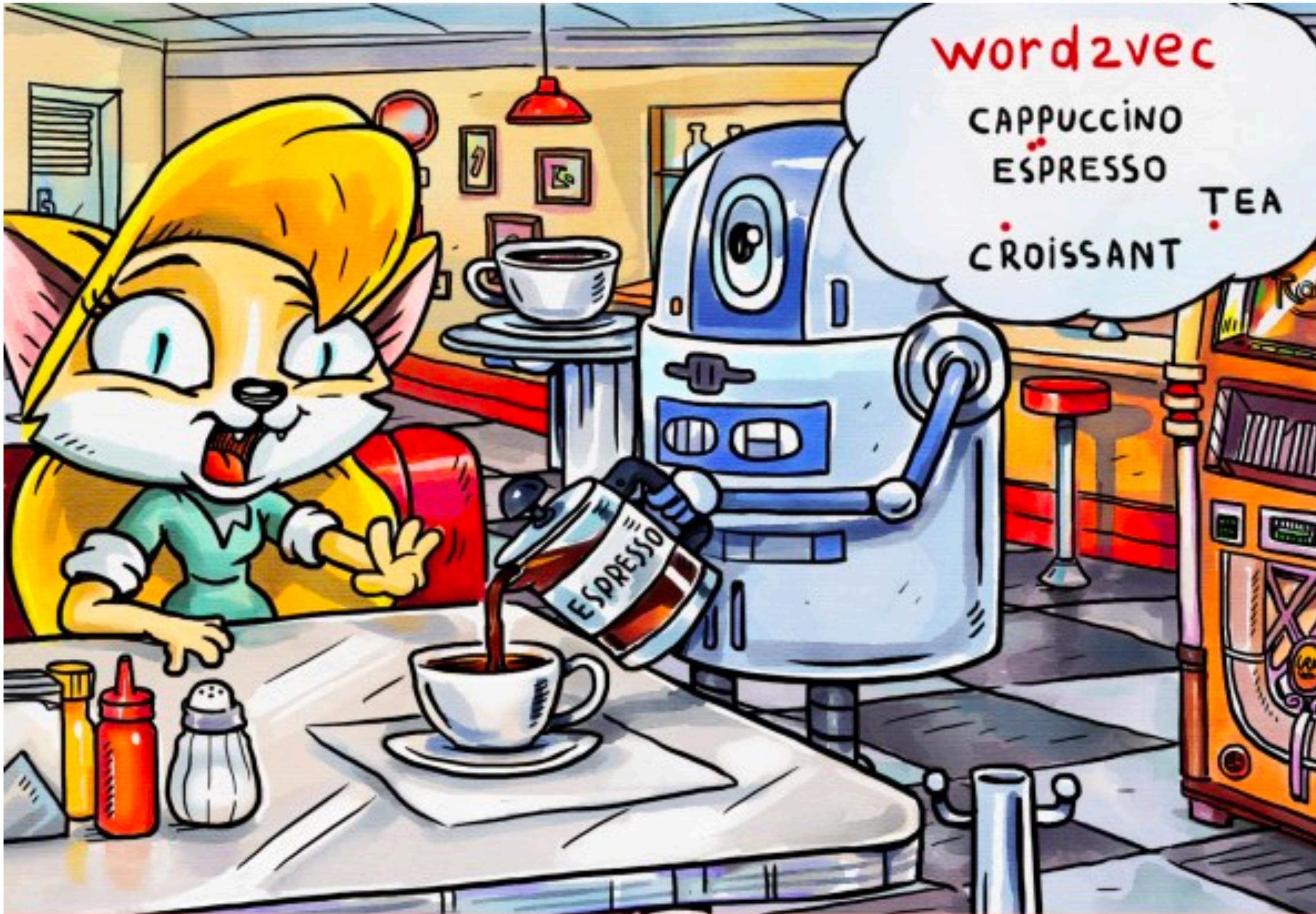
Word: преключение

- приключение 0.748698
- преключения 0.726111
- приключения 0.692828
- приключеия 0.670168
- прключение 0.666706
- приключеня 0.663286
- прключения 0.660438
- приключени 0.659609

Word: avito

- awito 0.693721
- авито 0.675299
- fvito 0.661414
- авита 0.659454
- irr 0.642429
- овито 0.606189
- avito 0.598056

# Intrinsic evaluation: analogies



- Espresso? But I ordered a cappuccino!
- Don't worry, the cosine distance between them is so small that they are almost the same thing.

# Word Vector Analogies

*Где зарыта собака:*  $\cos(b - a + a', x) \rightarrow \max$

# Word Vector Analogies

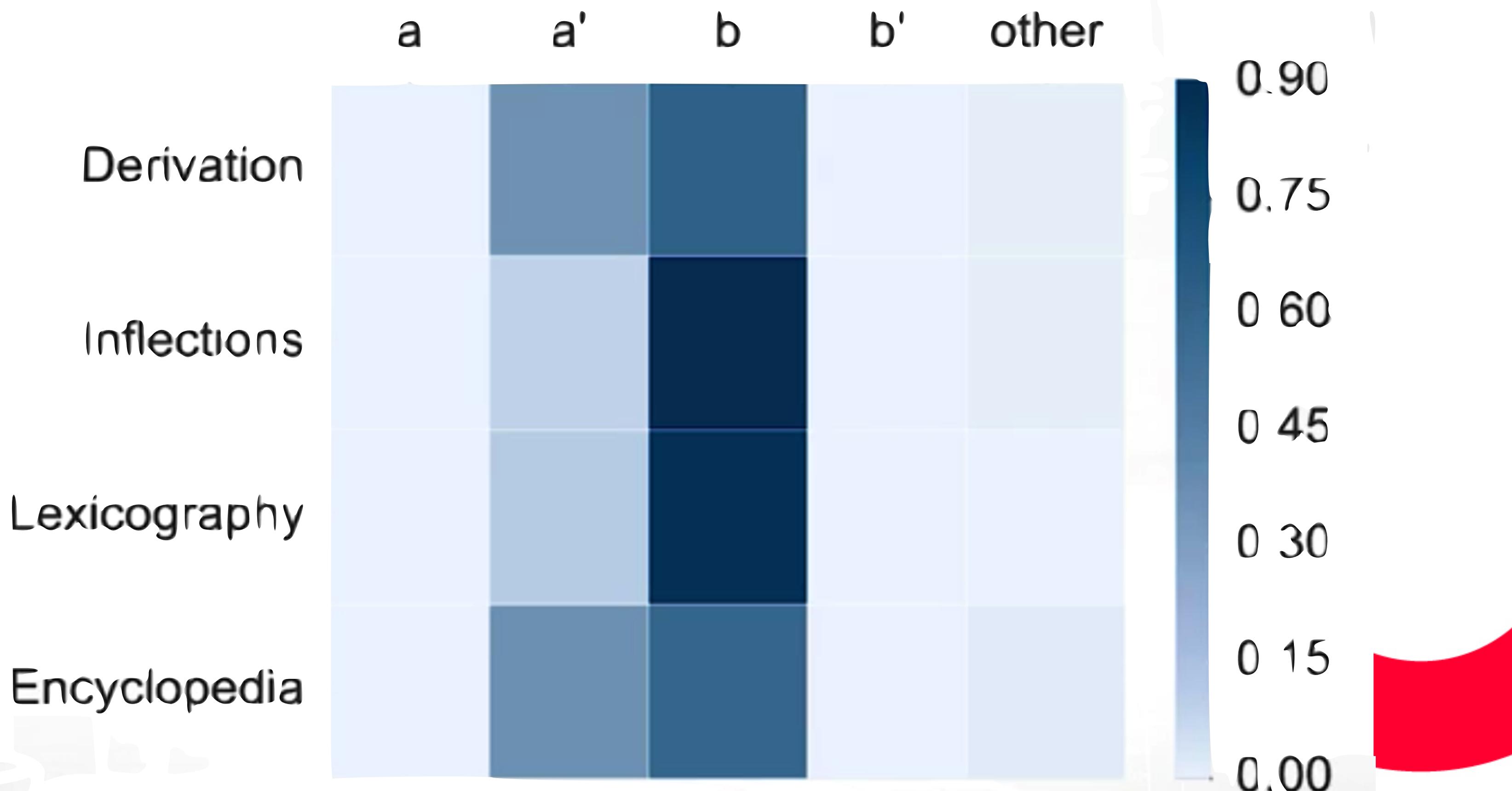


*Где зарыта собака:*  $\cos(b - a + a', x) \rightarrow \max_{x \notin \{a, a', b\}}$

# Word Vector Analogies



Где зарыта собака:  $\cos(b - a + a', x) \rightarrow \max_{x \notin \{a, a', b\}}$

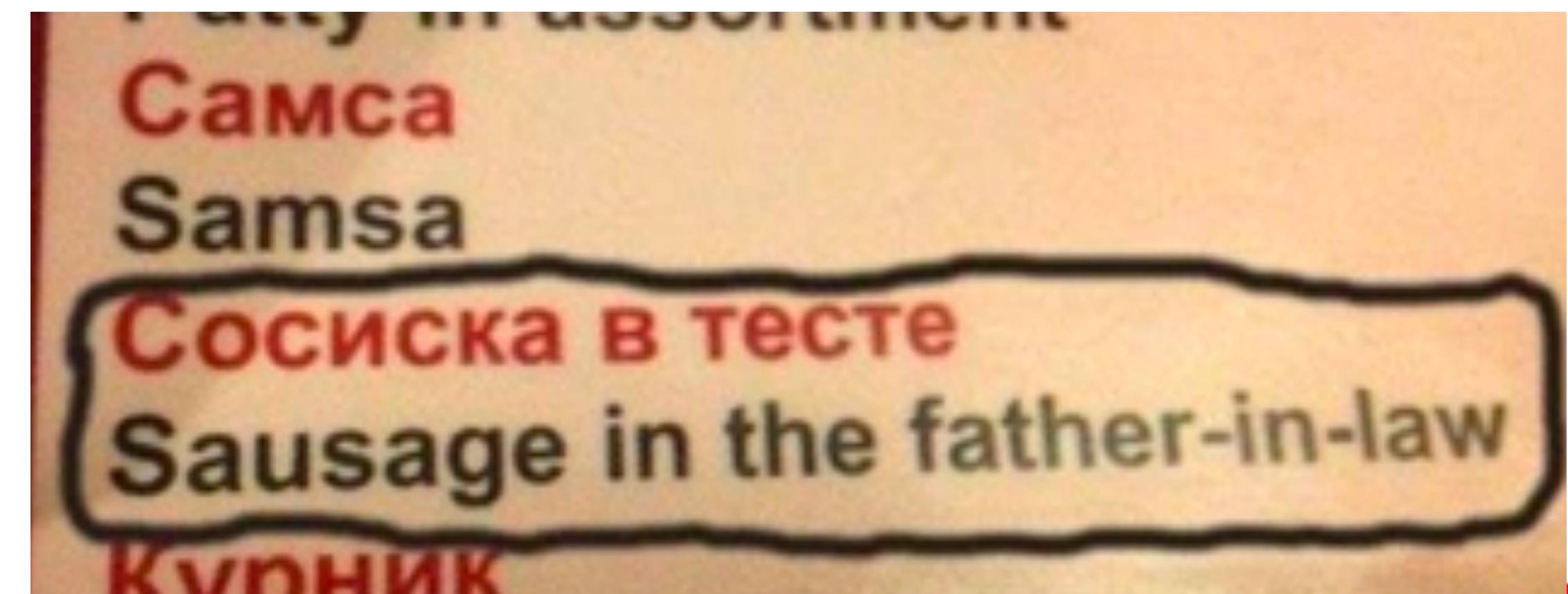


*king* – *man* + *woman*  $\approx$  *king*

# Extrinsic evaluation

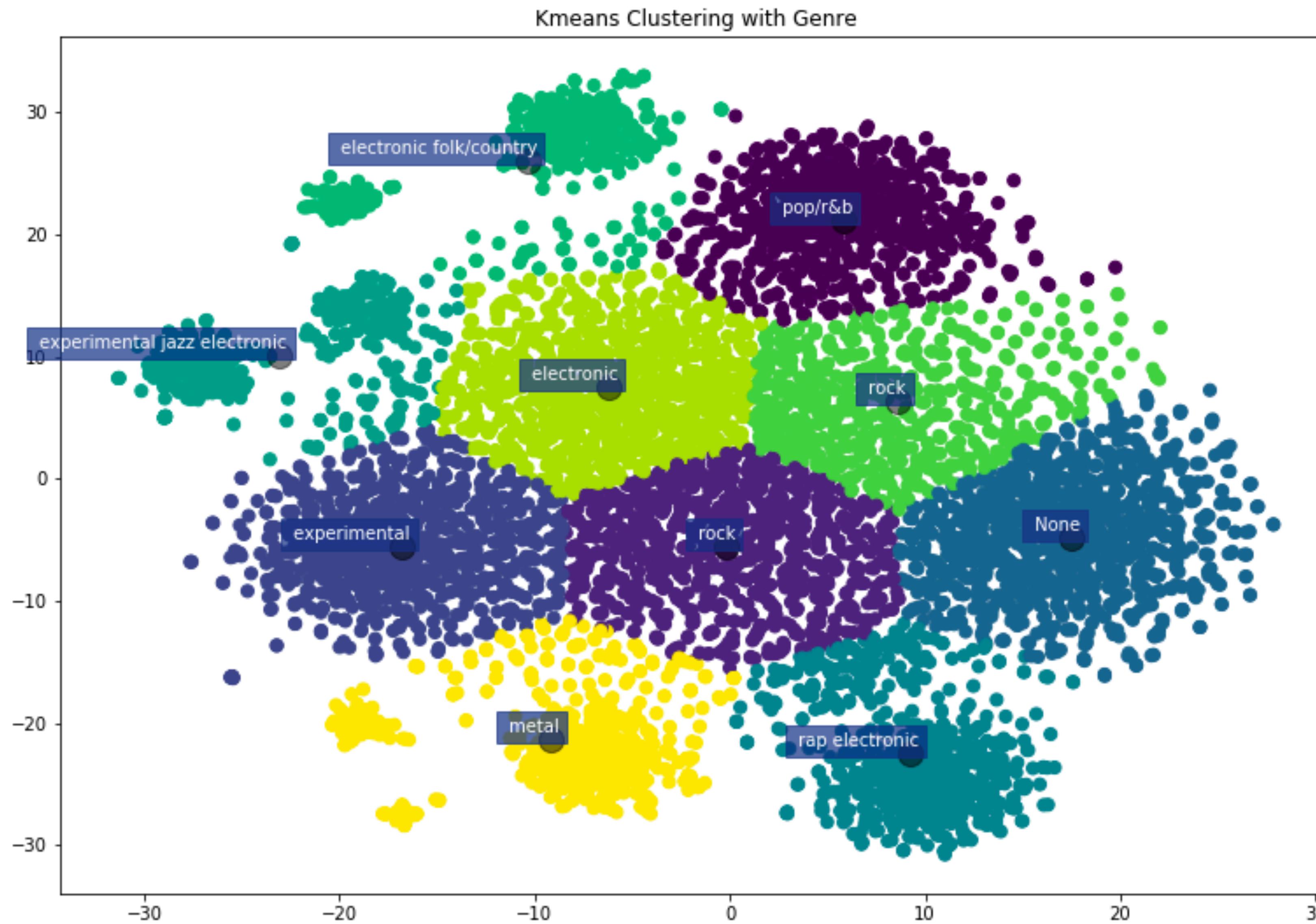
Оценка на конкретной внешней задаче:

- Машинный перевод
- NER
- Question answering
- etc.

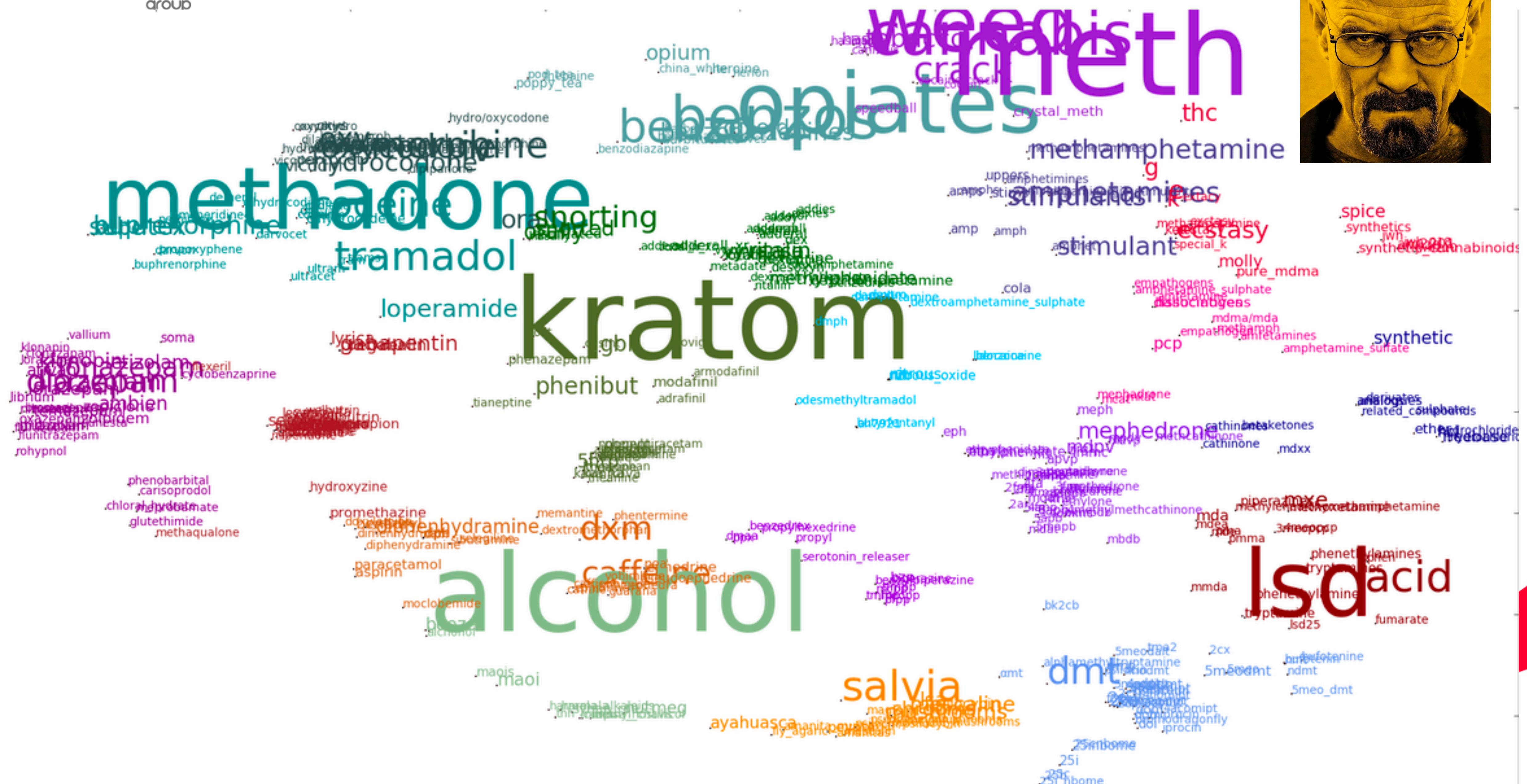


# Word Vector Visualisation

Можно разложить на плоскости (PCA, t-sne) и покластеризовать:



# Word Vector Visualisation



# Интересные применения Word2vec



## Music Recommendations: Spotify, Anghami

- learn song vectors
- create a “music taste” vector for a user by averaging together the vectors for songs that a user likes to listen to

Context  
Input  
Context

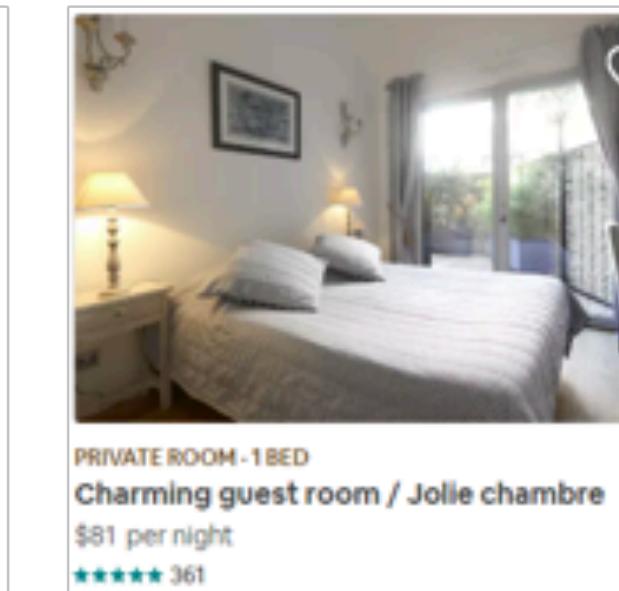
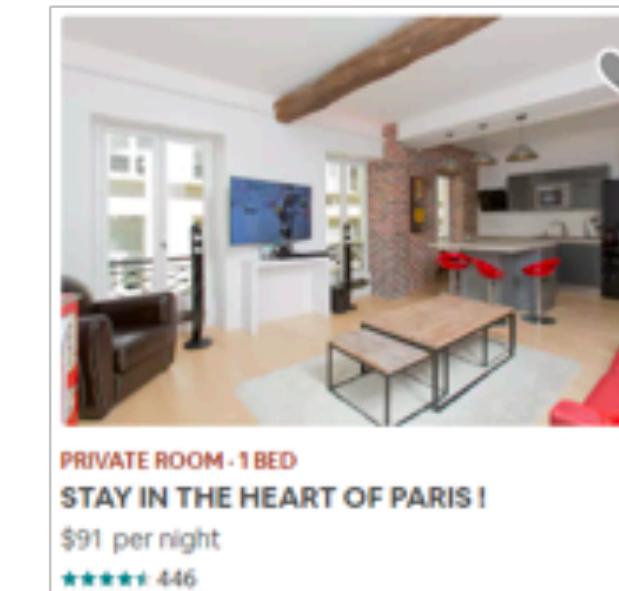
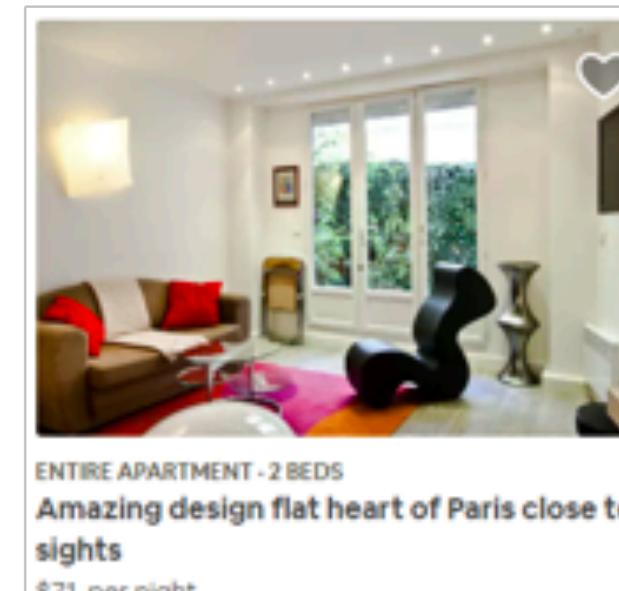
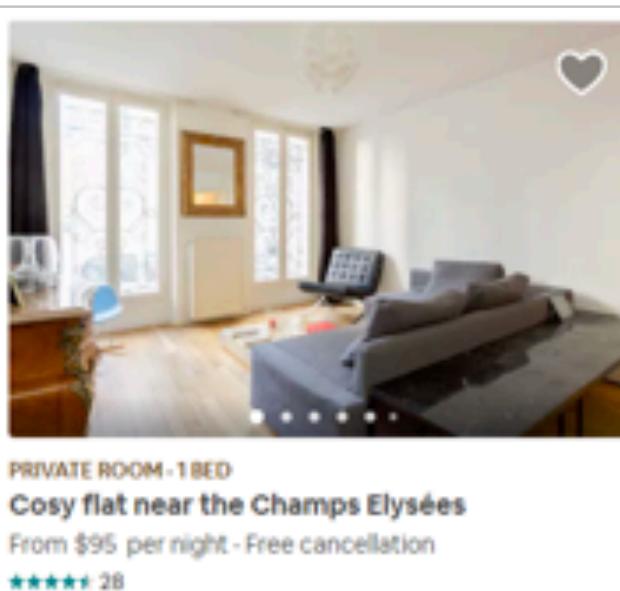
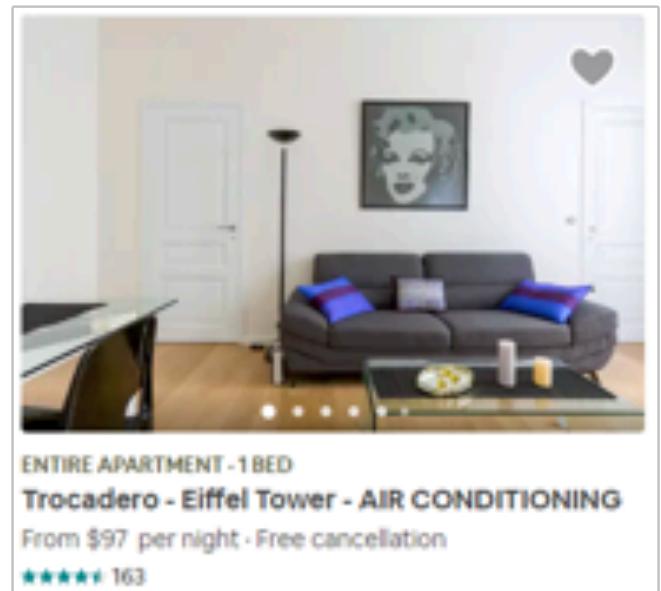
TITLE	ARTIST	ALBUM	
✓ Voyeur	Phantoms, Ni...	Broken Halo	4:40
✓ Somebodies Something	Tyne	Somebodies ...	3:44
✓ No Words - Kasbo Remix	Erik Hassle, C...	No Words (Re...	5:08
✓ I Will Wait	Aaron Krause	I Will Wait	3:54
✓ Lost	Ficci	Lost	4:07
✓ Falling Short (DarkO Remix)	Låpsley, DarkO	Falling Short (...	4:12
✓ Self Defined	Maya Payne	The Lucky On...	3:54
✓ Escape	Tongues.	Kitsuné Hot S...	3:07

# Интересные применения Word2vec

## Listing Recommendations: Airbnb



### VIEWING HISTORY



Context

Input

Context

- learn vector representations of user's listings
- apply for recommendations

<https://medium.com/airbnb-engineering/listing-embeddings-for-similar-listing-recommendations-and-real-time-personalization-in-search-601172f7603e>

# Интересные применения Word2vec

## Product Recommendations: Yahoo Mail



- learn embeddings for the products
- cluster products
- use this for effective product recommendations

Amazon.com	Inbox	Amazon Orders	Your Amazon.com	<a href="#">View order ↗</a>	Jun 12
Target	Inbox	Thanks for shopping Target! Here's your order #: 90		<a href="#">View order ↗</a>	Jun 10
Amazon.com (3)	Inbox	Amazon Orders	Your Amazon.com	<a href="#">View order ↗</a>	Jun 10
Amazon.com	Inbox	Your Amazon.com order of "Da		<a href="#">Track package ↗</a>	Jun 6
Amazon.com	Inbox	Amazon Orders	Your Amazon.com	<a href="#">View order ↗</a>	Jun 4
orderstatus@costco.com	Inbox	Your Costco.com Order Number i		<a href="#">View order ↗</a>	Jun 4
Amazon.com	Inbox	Your Amazon.com order of "Fr		<a href="#">Track package ↗</a>	Jun 3
orderstatus@costco.com	Inbox	Your Costco.com Order Number i		<a href="#">View order ↗</a>	Jun 2
Amazon.com	Inbox	Amazon Orders	Your Amazon.com	<a href="#">View order ↗</a>	Jun 2

# GloVe



# w2v drawbacks

Основной недостаток word2vec:

Моделирует **локальные зависимости**, не учитывая глобальной статистики по корпусу документов

# w2v drawbacks

Основной недостаток word2vec:

Моделирует **локальные зависимости**, не учитывая глобальной статистики по корпусу документов

Возникла идея совместить 2 подхода, взяв лучшее из каждого:

- построить осмысленные вектора
- учитывать не только локальные зависимости, но и статистику корпуса

# Global Vectors - GloVe

$$J(\theta) = \frac{1}{2} \sum_{i,j=1}^W f(P_{ij})(u_i^T v_j - \log P_{ij})^2$$

- два вектора для каждого слова
- скалярное произведение векторов должно быть близко к логарифму частоты со-встречаемости слов

# Global Vectors - GloVe

$$J(\theta) = \frac{1}{2} \sum_{i,j=1}^W f(P_{ij})(u_i^T v_j - \log P_{ij})^2$$

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

$$x_{\max} = 100$$

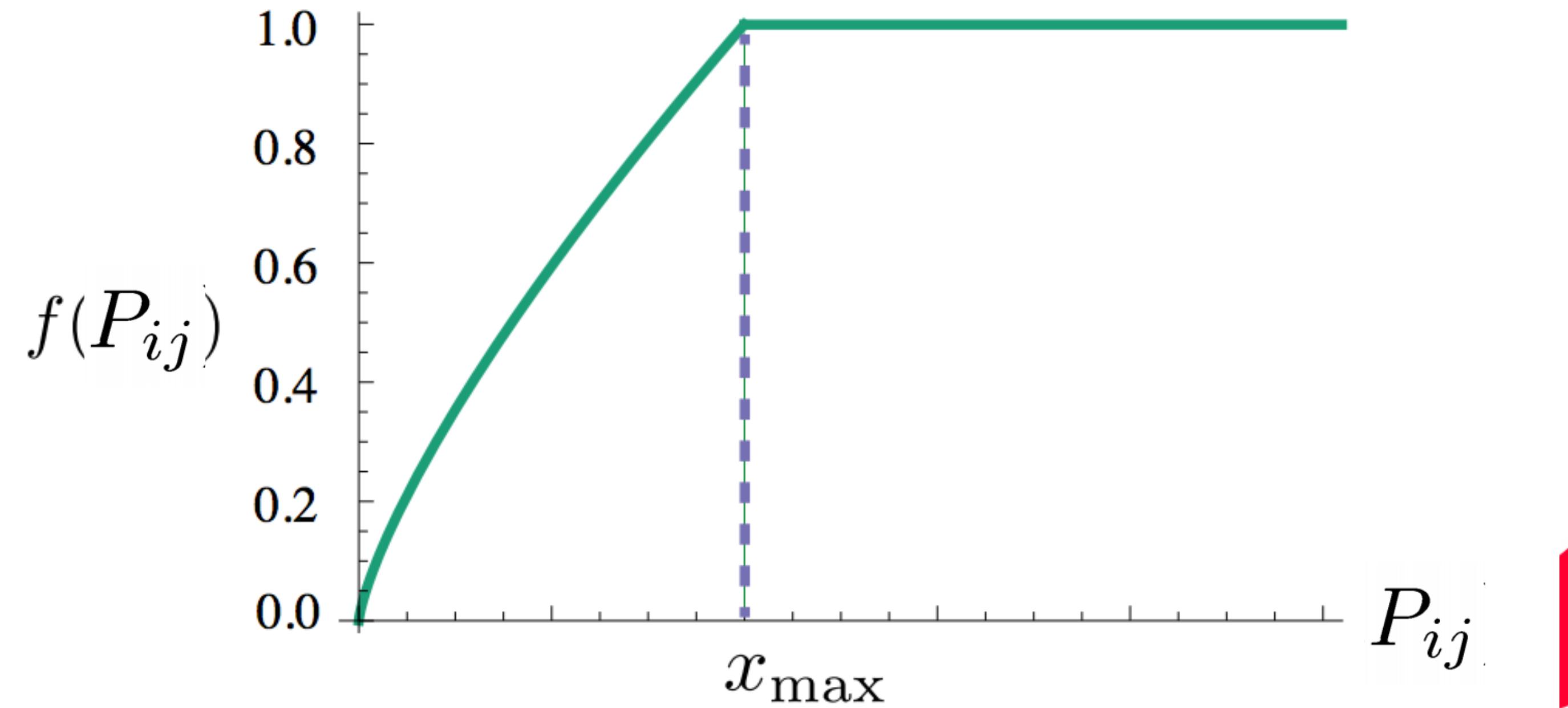


Figure 1: Weighting function  $f$  with  $\alpha = 3/4$

# Global Vectors - GloVe

Можно показать, что word2vec при обучении skip-gram with negative sampling (SGNS) неявно раскладывает смещенную матрицу PMI (shifted PMI).

В то же самое время, GloVe - явная факторизация: хотим получить такое разложение на 2 матрицы, произведение которых аппроксимирует матрицу со-встречаемости

# Global Vectors - GloVe

## Плюсы:

- быстро тренируется (уже знаем всю статистику по корпусу)
- масштабируется для больших датасетов
- получаются хорошие значимые вектора даже для небольших корпусов

# Doc2Vec



# Doc2Vec aka Paragraph2vec

Хотим учить сразу вектора для документов, а не конструировать их из векторов слов.

Эти вектора должны быть осмыслены, например, два похожих документа должны иметь похожие вектора.

Хотим также получать вектора для новых документов

Doc2Vec - gensim

Paragraph2Vec - original paper

# Paragraph2Vec

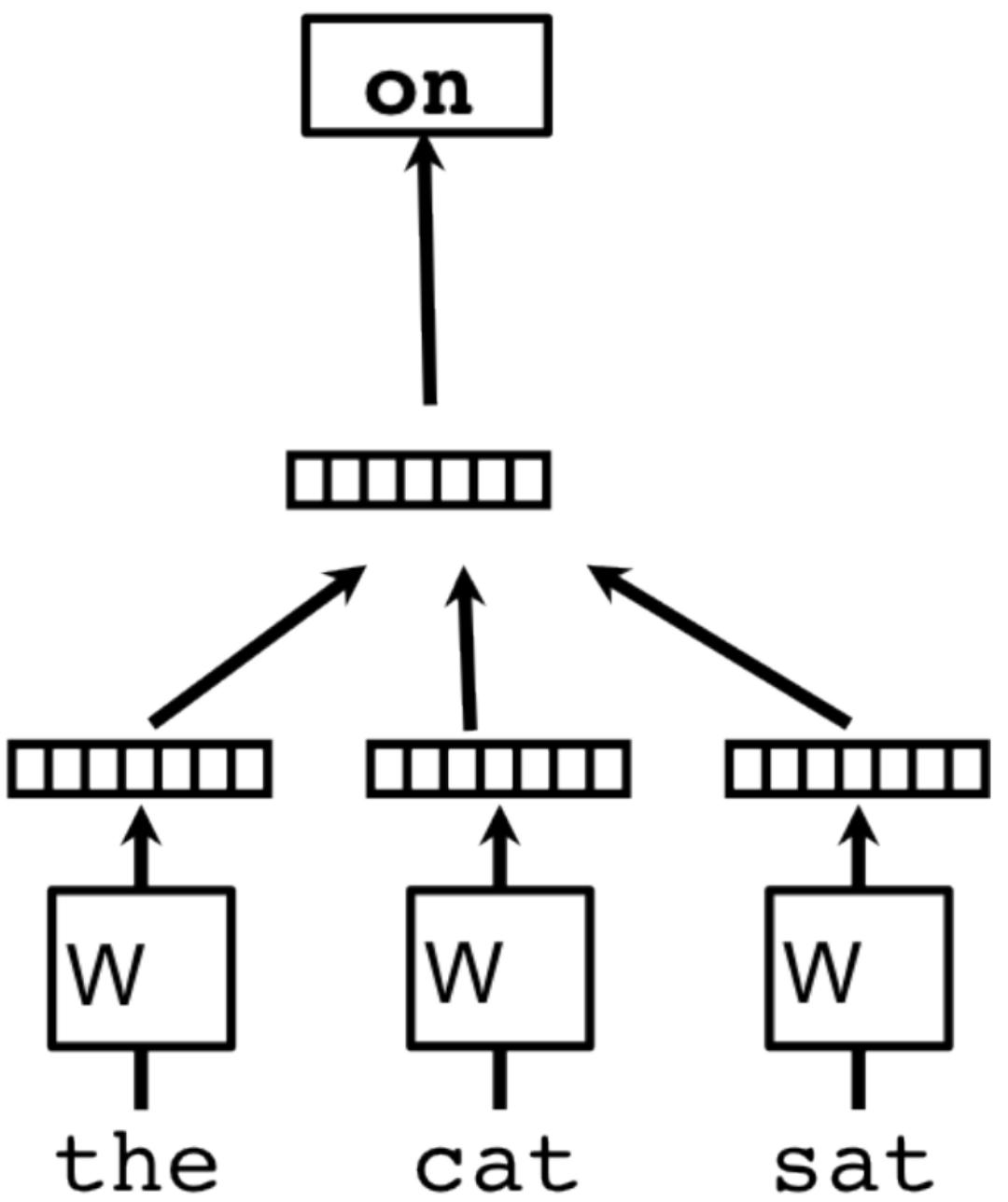
Будем учить вектора документов в процессе предсказания разных слов/контекстов этого документа.

# Paragraph2Vec

Будем учить вектора документов в процессе предсказания разных слов/контекстов этого документа.

## w2v - CBOW

Classifier



Average/Concatenate

Word Matrix

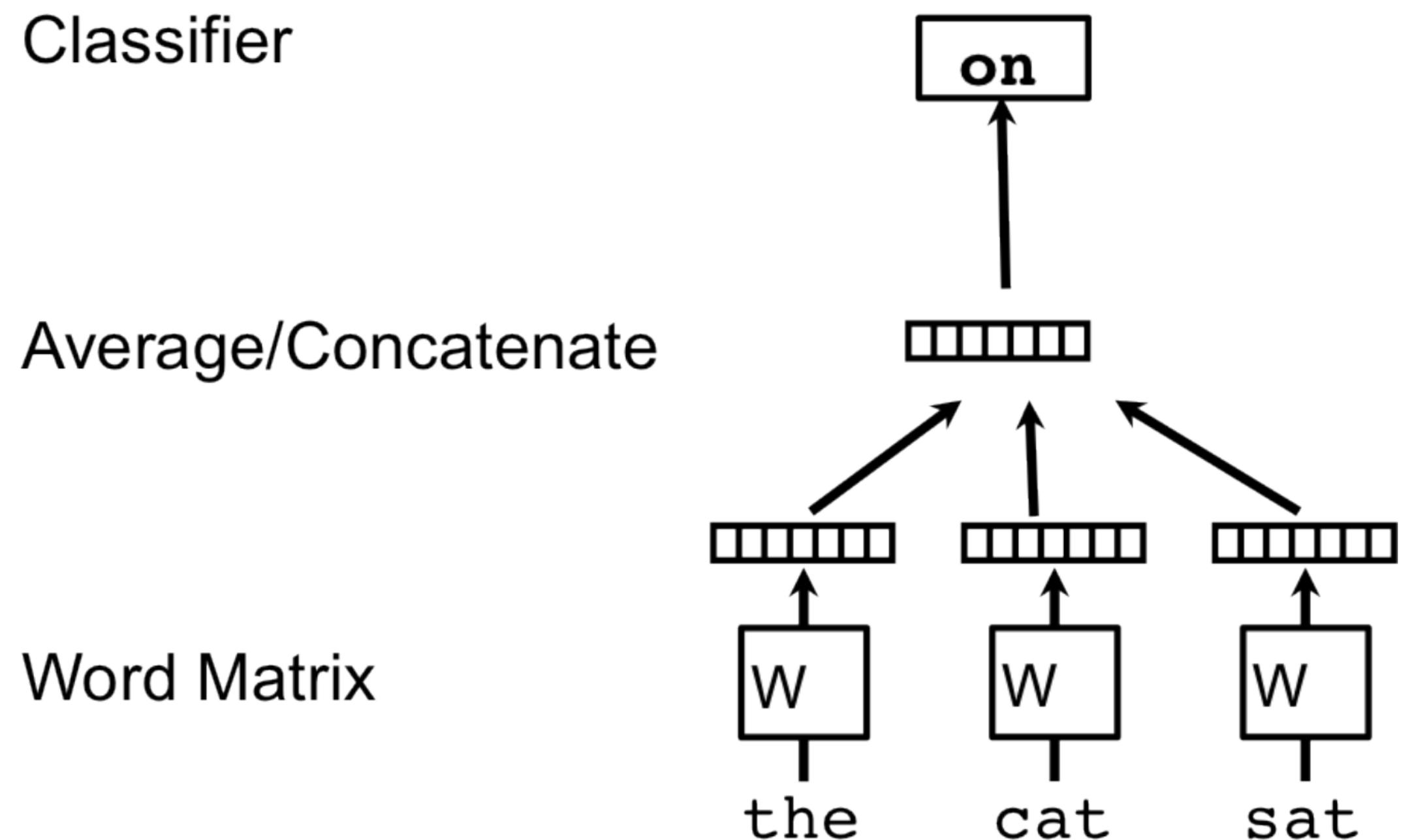
Учимся хорошо  
предсказывать слово по  
контексту

# Paragraph2Vec

Будем учить вектора документов в процессе предсказания разных слов/контекстов этого документа.

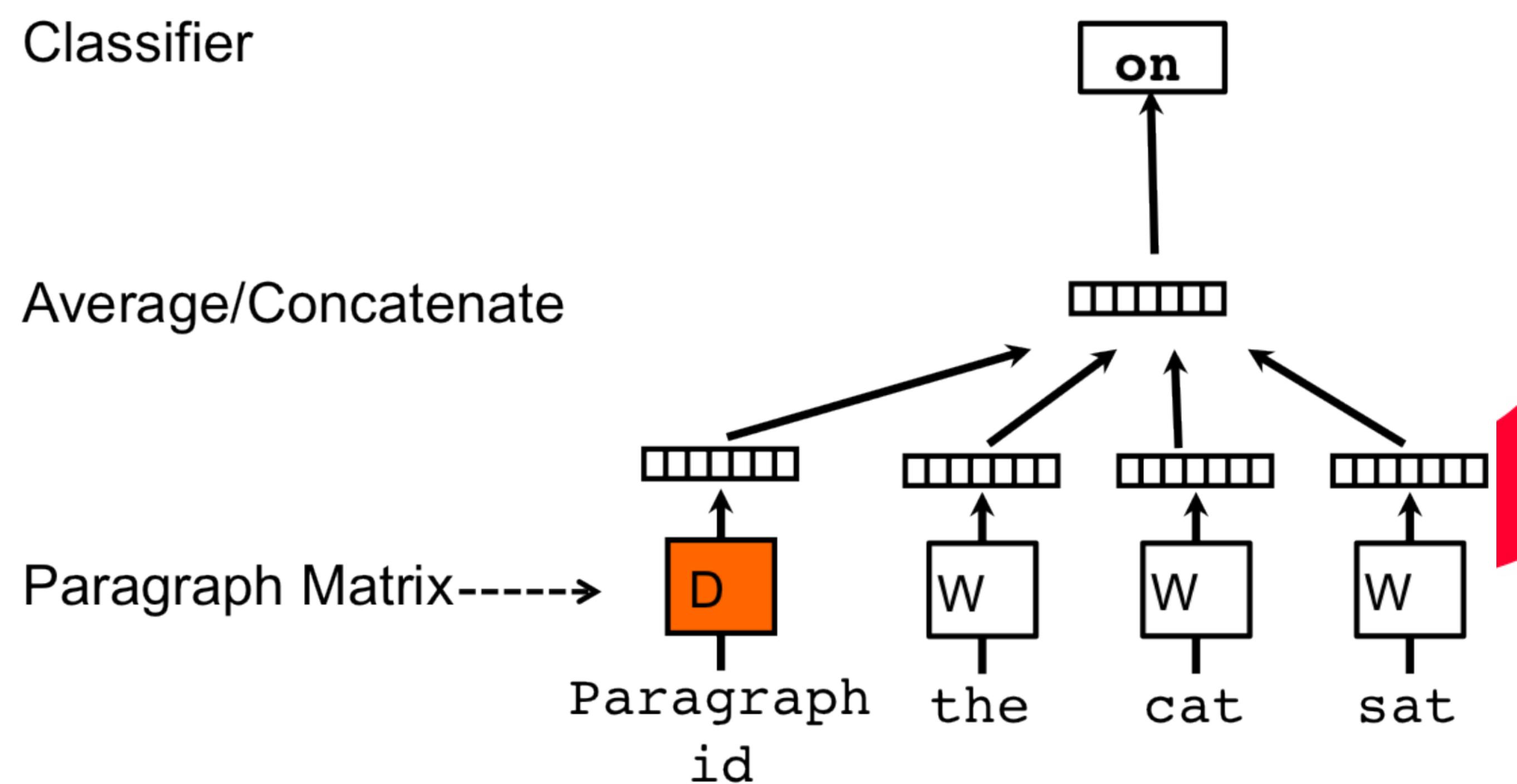
w2v - CBOW

Classifier



P2v - Distributed Memory

Classifier



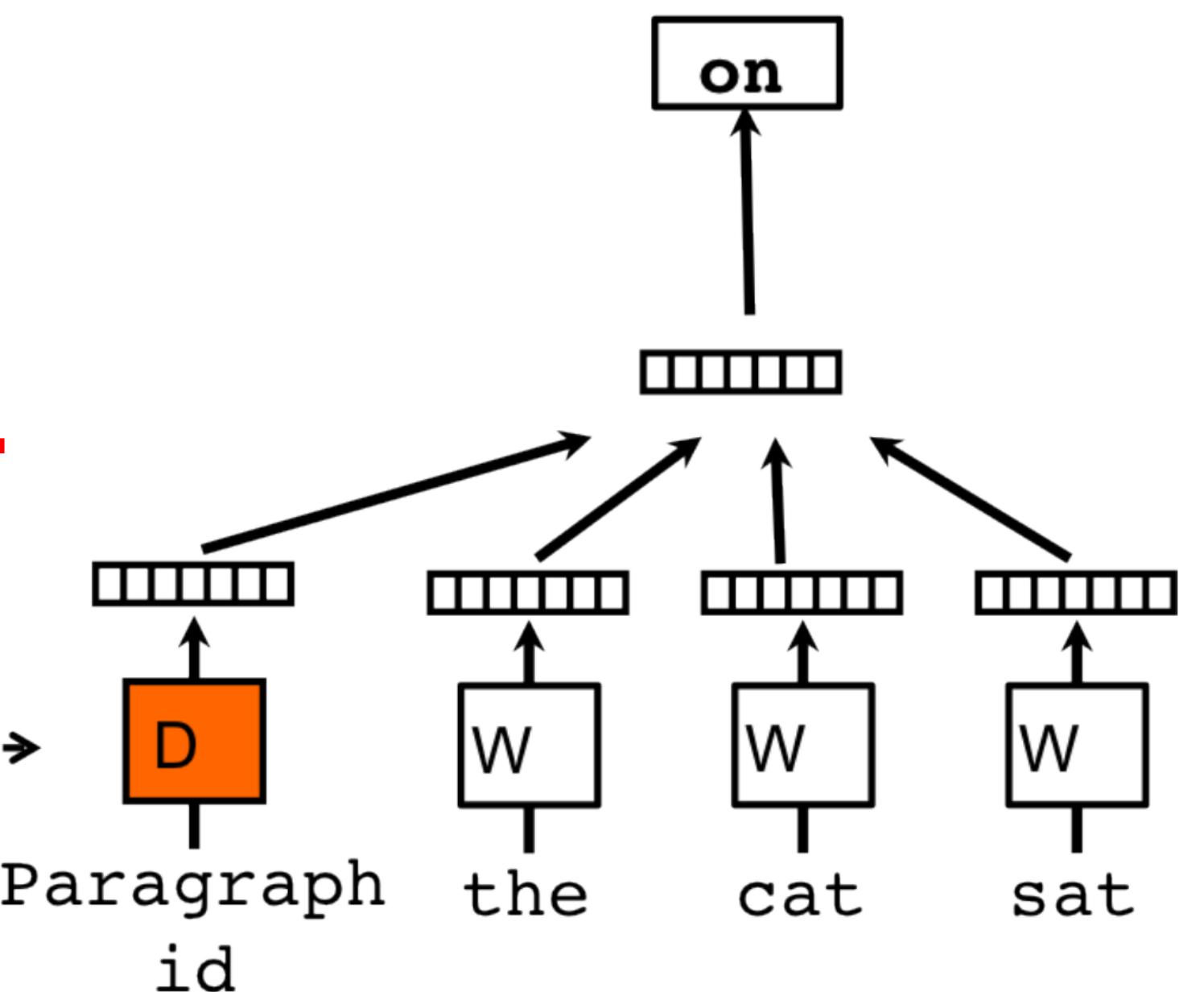
# Paragraph2Vec- DM

## Distributed Memory

Classifier

Average/Concatenate

Paragraph Matrix----->



# Paragraph2Vec- DM

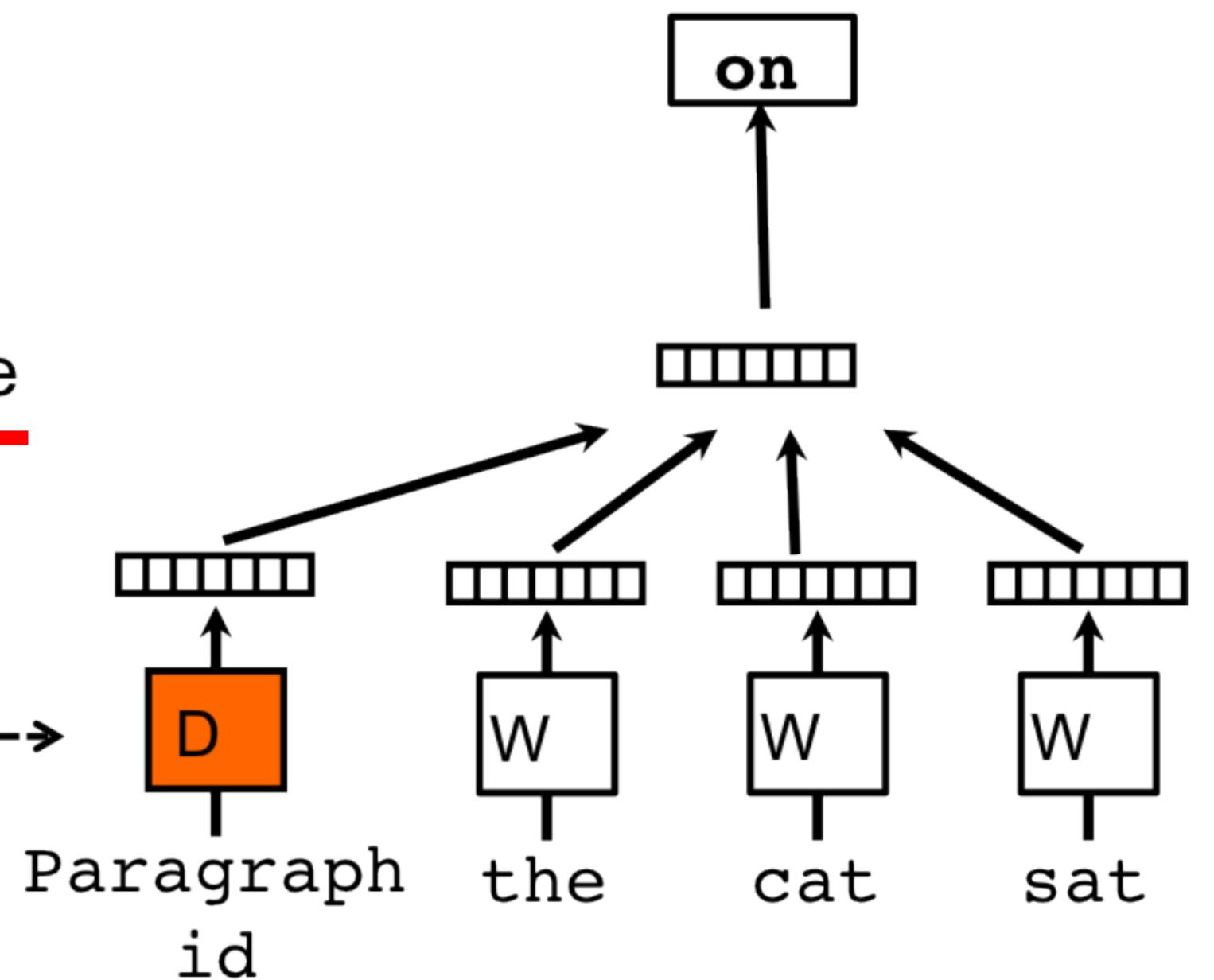
## Distributed Memory

Classifier

Дополнительно учим еще и вектор документа.

Average/Concatenate

Paragraph Matrix----->



$$\frac{1}{M} \sum_{i=1}^M \frac{1}{|D_i|} \sum_{t=k}^{|D_i|-k} \log(p(w_t^i | w_{t-k}^i, \dots, w_{t+k}^i, D_i))$$

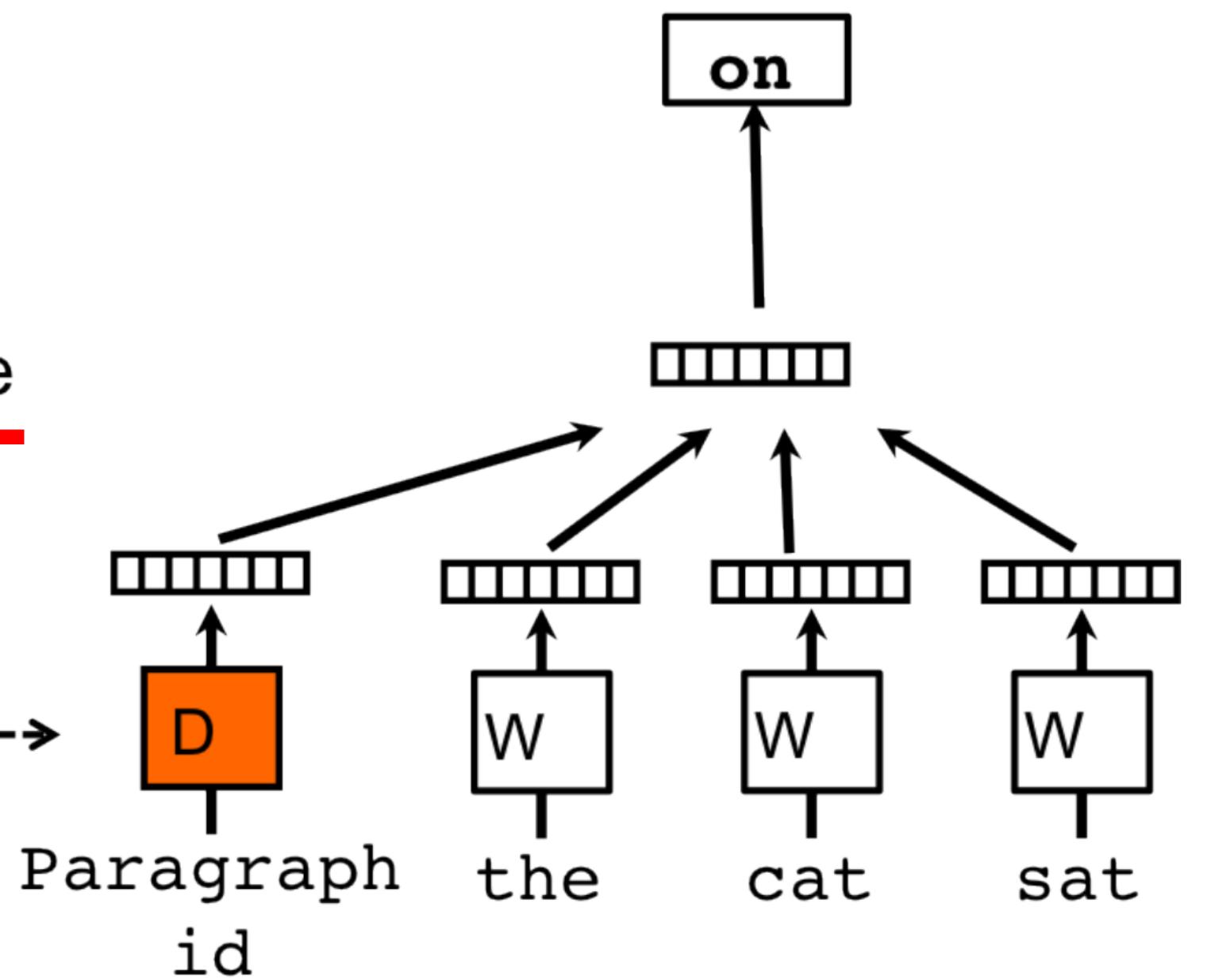
# Paragraph2Vec- DM

## Distributed Memory

Classifier

Average/Concatenate

Paragraph Matrix----->



Дополнительно учим еще и вектор документа.

**Интуиция:** вектор документа работает как распределенная память, которая хранит его тему и помогает «вспомнить» слова

$$\frac{1}{M} \sum_{i=1}^M \frac{1}{|D_i|} \sum_{t=k}^{|D_i|-k} \log(p(w_t^i | w_{t-k}^i, \dots, w_{t+k}^i, D_i))$$

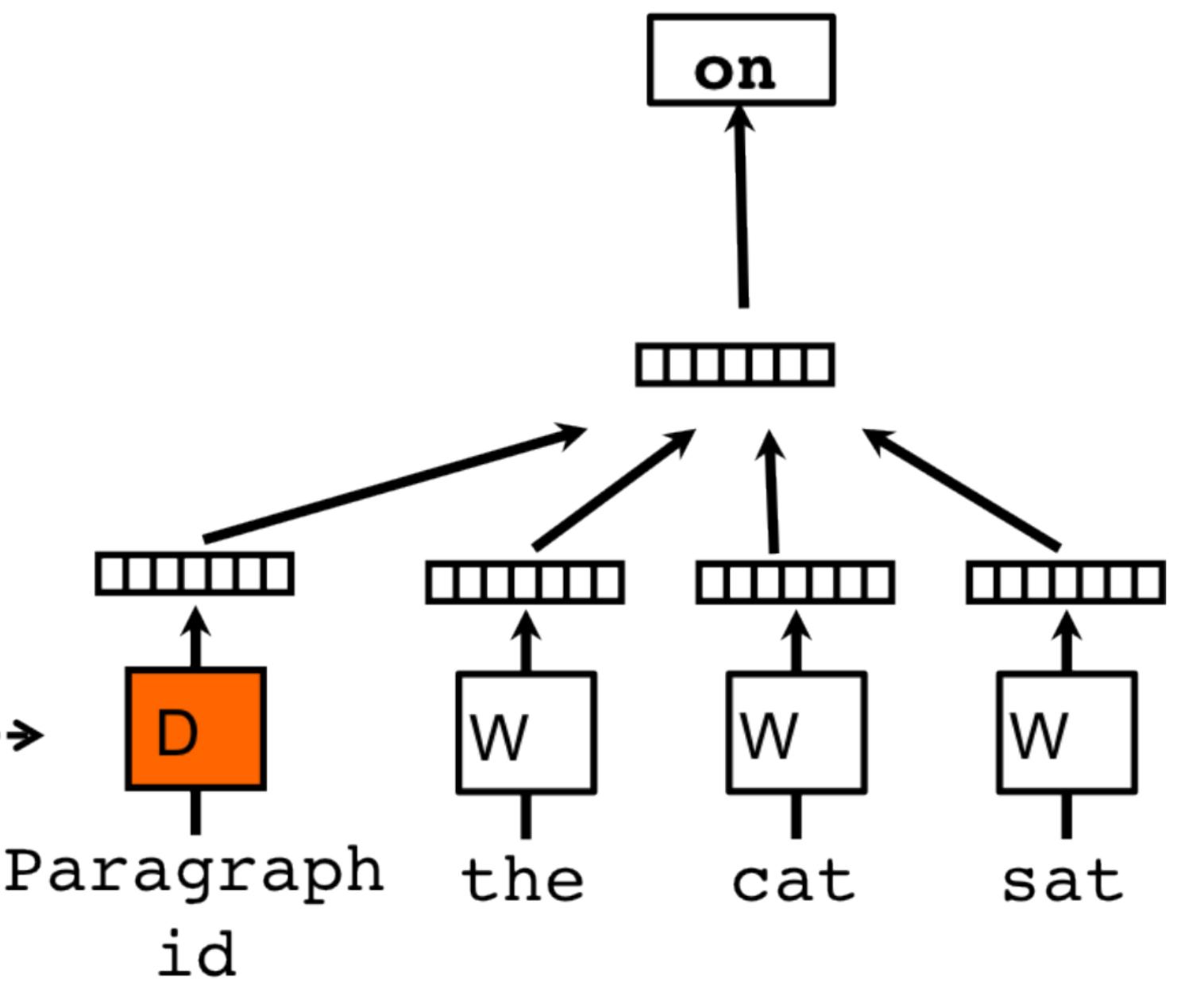
# Paragraph2Vec- DM

## Distributed Memory

Classifier

Average/Concatenate

Paragraph Matrix----->



**Контексты:** скользящее окно по словам

**Вектор документа:** общий при обучении для всех контекстов данного документа

**Вектора слов:** общие для всех текстов

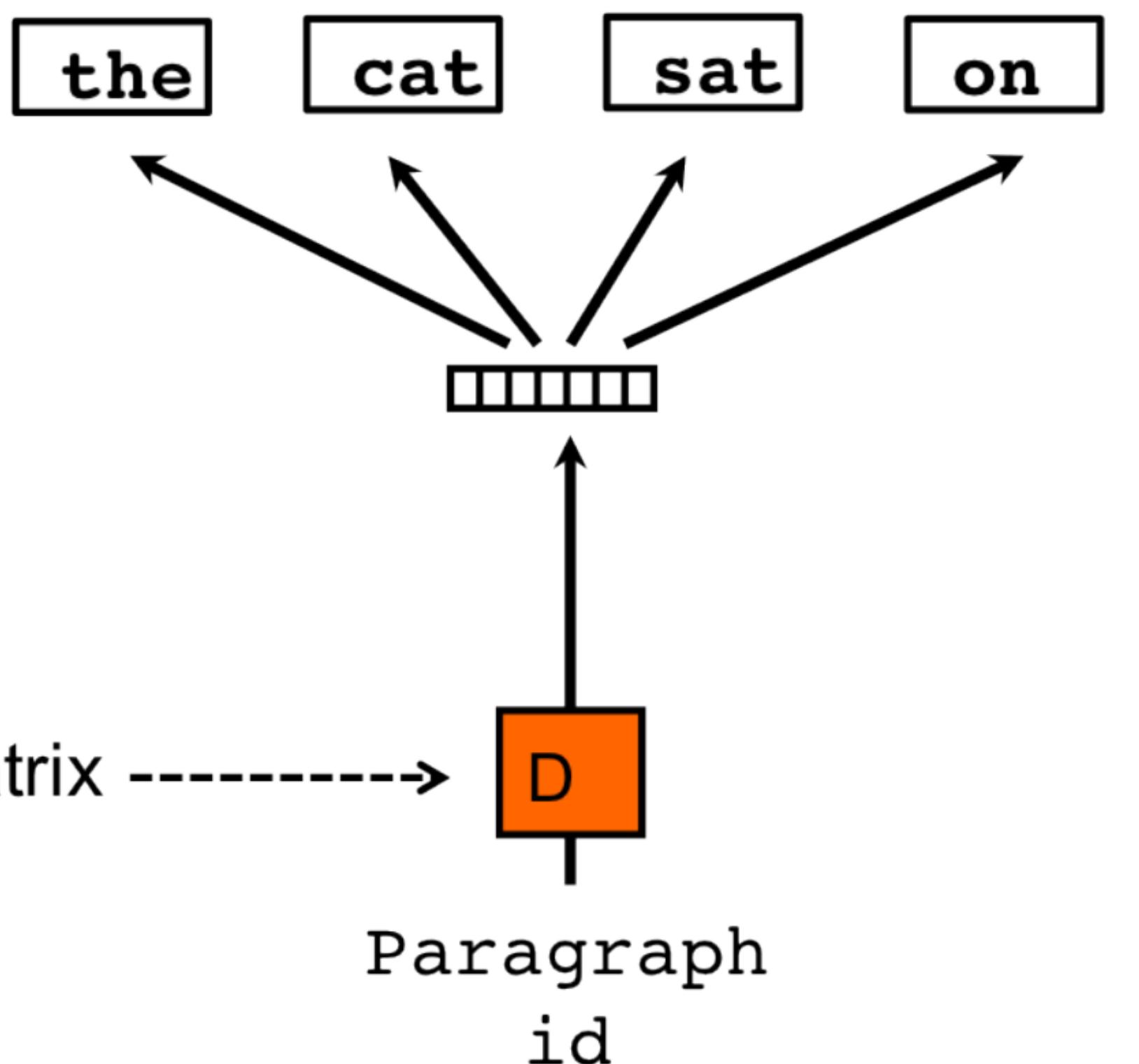
Учитывает порядок слов

$$\frac{1}{M} \sum_{i=1}^M \frac{1}{|D_i|} \sum_{t=k}^{|D_i|-k} \log(p(w_t^i | w_{t-k}^i, \dots, w_{t+k}^i, D_i))$$

# Paragraph2Vec - DBOW

## Distributed Bag Of Words Model

Classifier

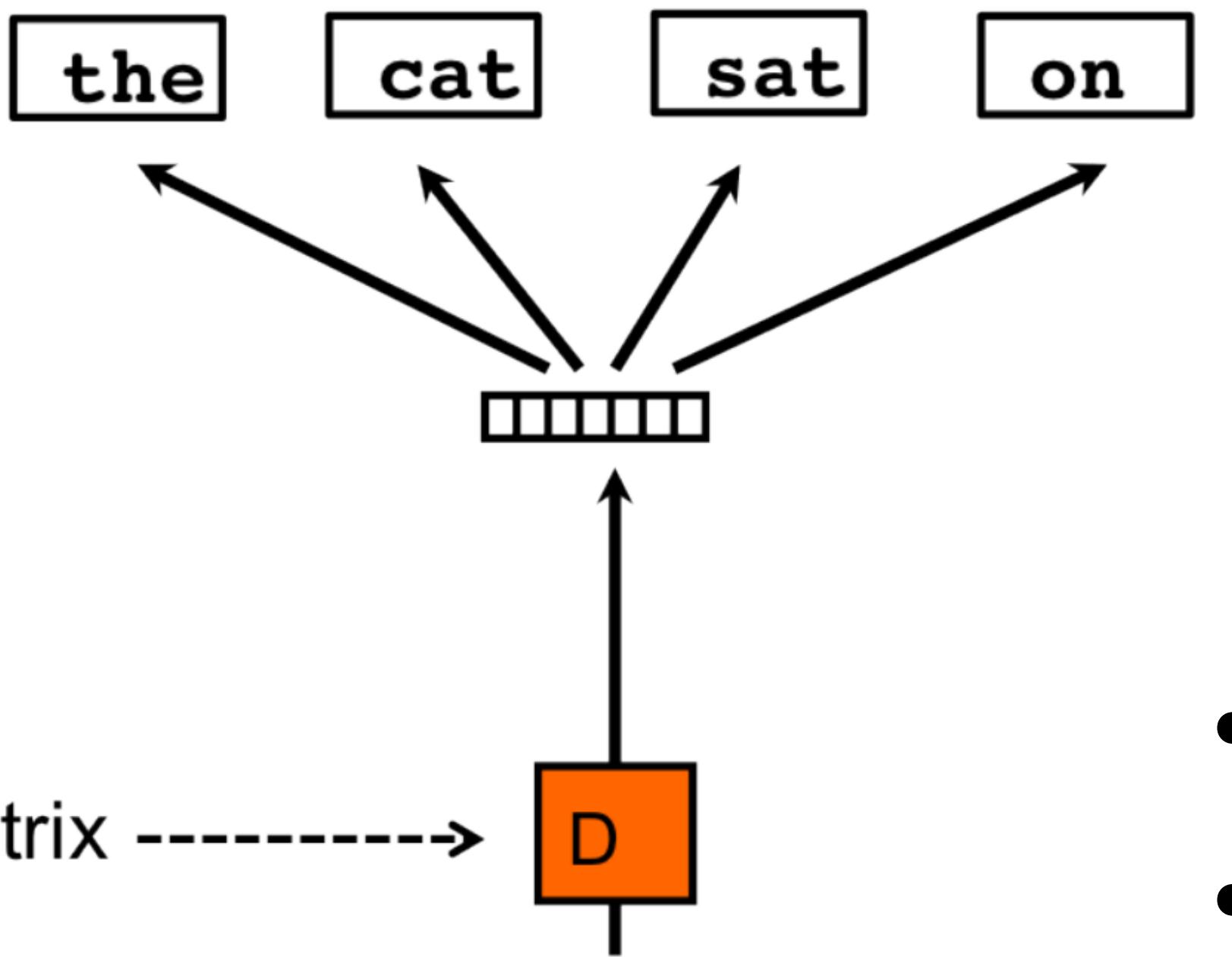


Будем требовать, чтобы модель корректно предсказывала randomный sample слов из текста **только по вектору документа**.

# Paragraph2Vec - DBOW

## Distributed Bag Of Words Model

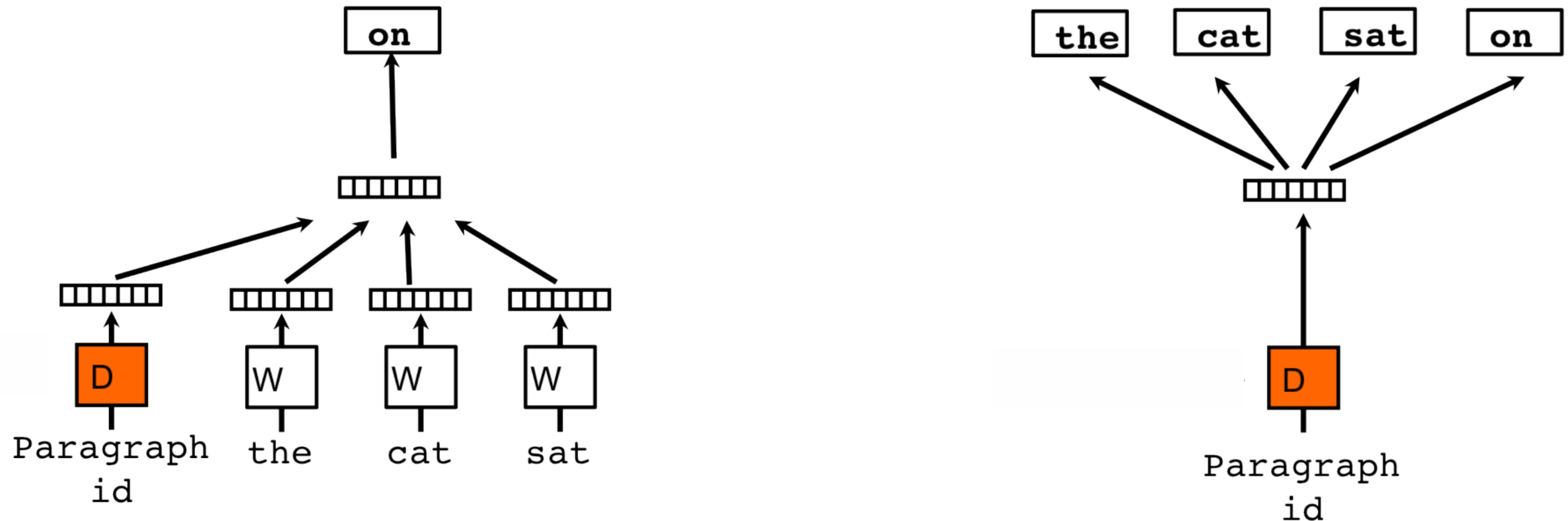
Classifier



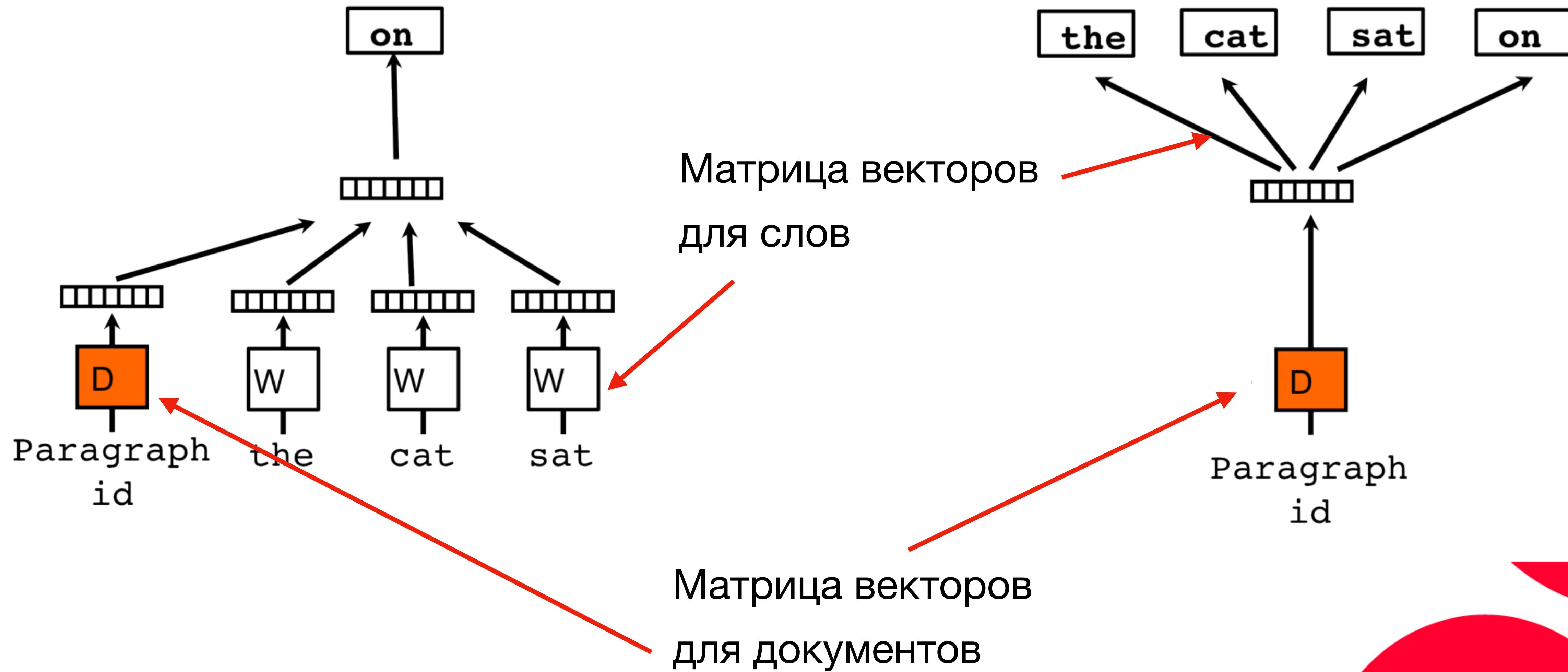
Будем требовать, чтобы модель корректно предсказывала рандомный сэмпл слов из текста **только** по вектору документа.

- берем рандомный сэмпл слов
- для каждого слова делаем предсказание (софтмакс на размер словаря)
- обновляем вектор (SGD)

# Вектора для новых текстов

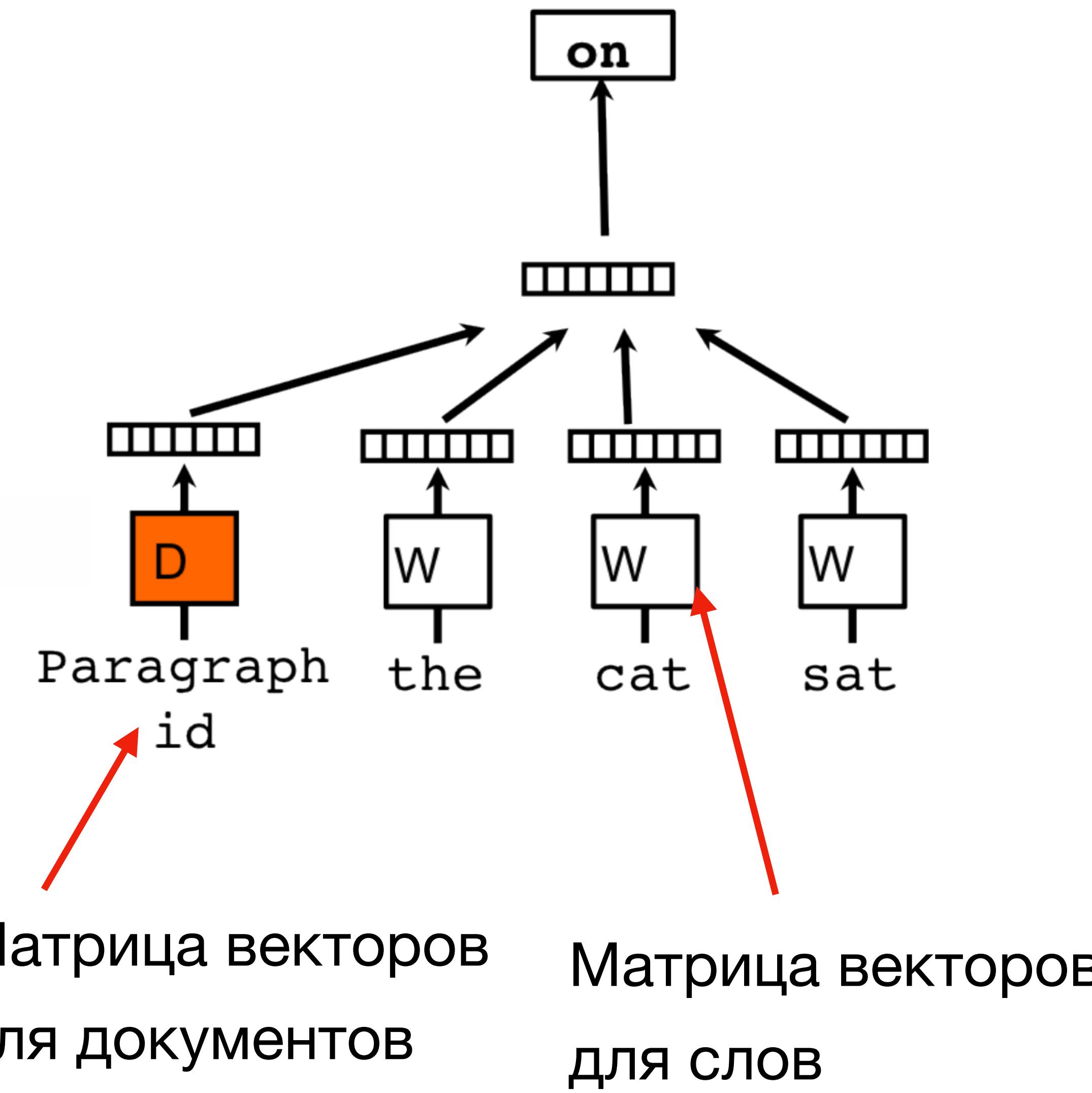


# Вектора для новых текстов

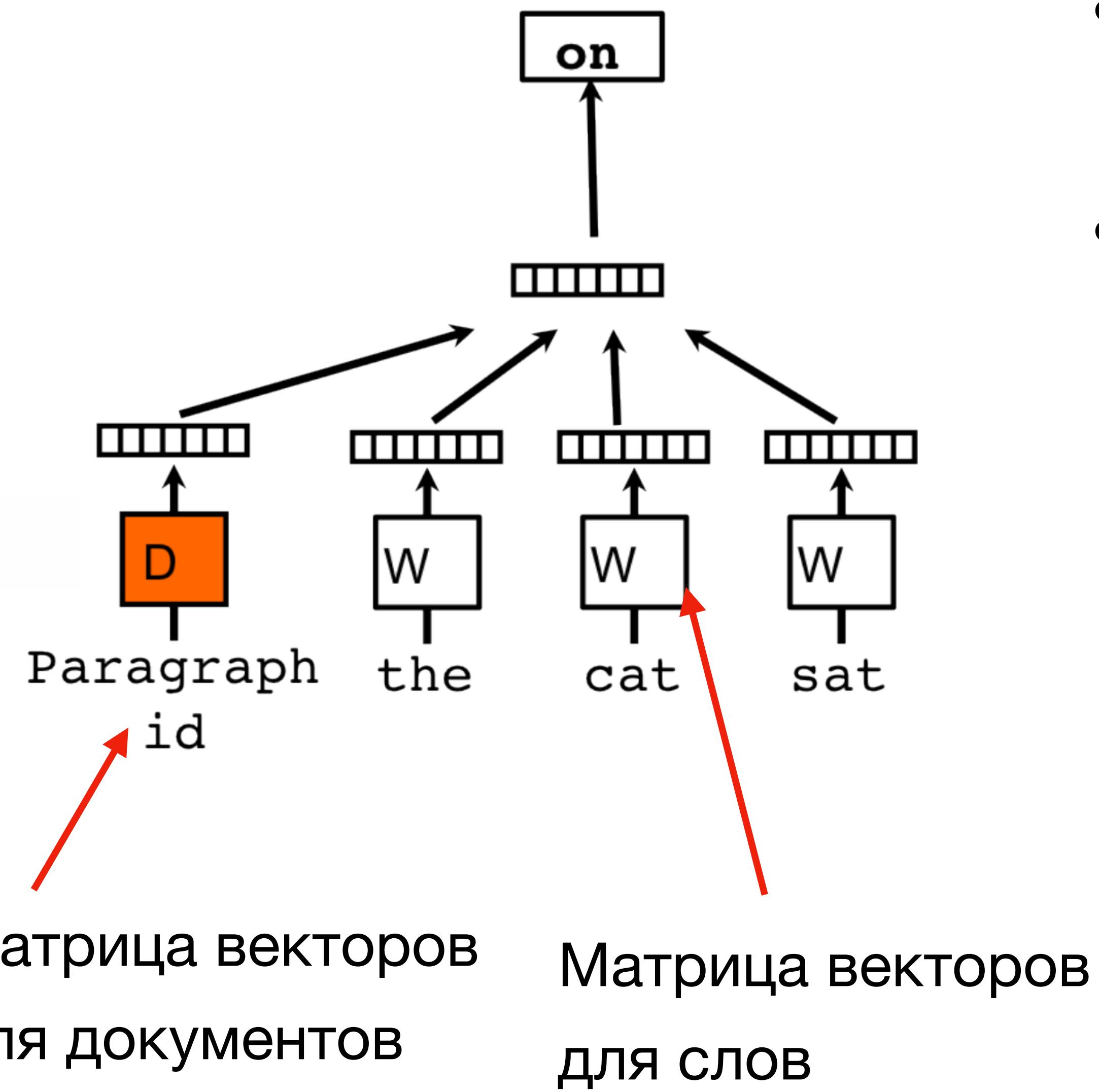


# Вектора для новых текстов

- Инициализируем новый вектор случайным образом

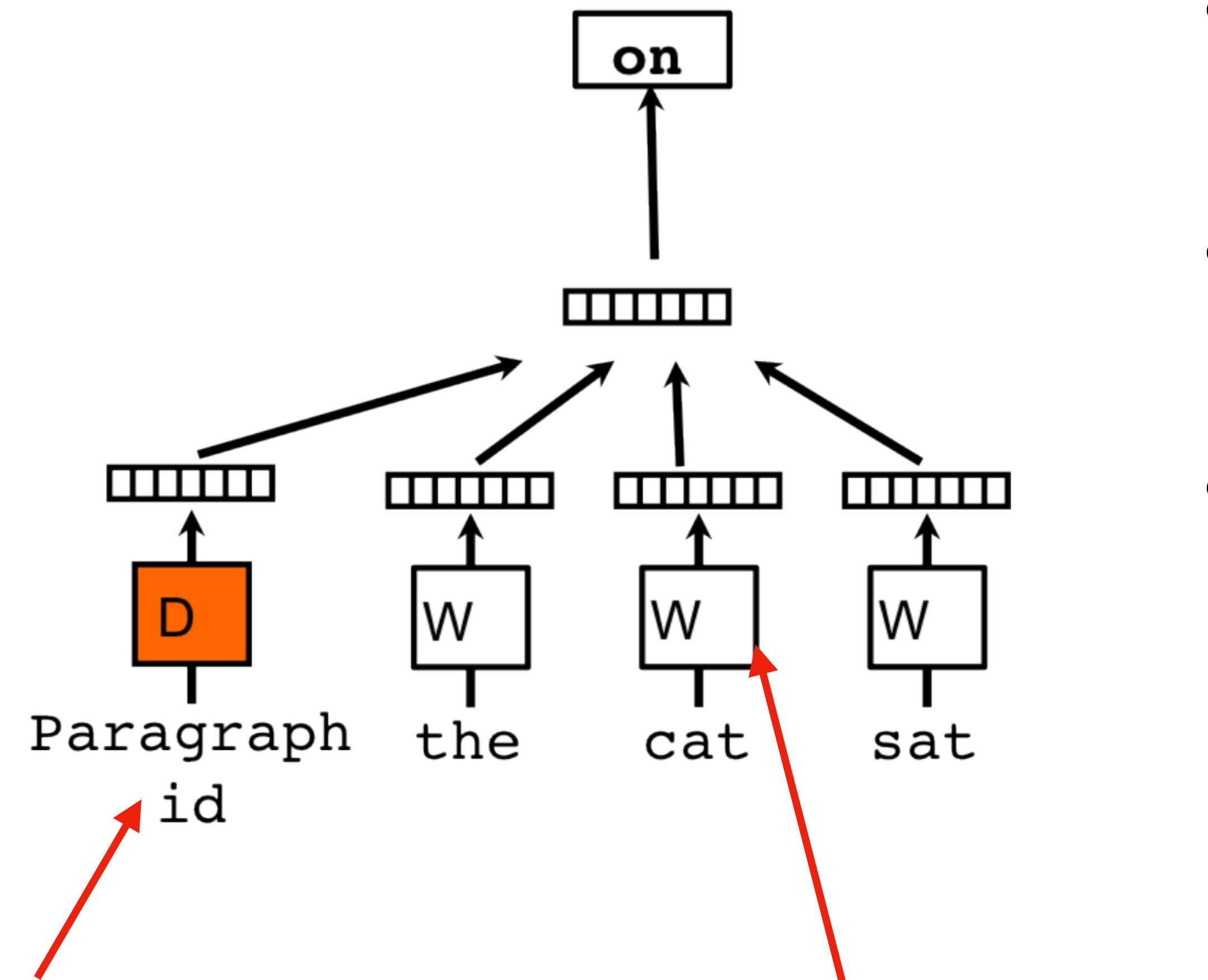


# Вектора для новых текстов



- Инициализируем новый вектор случайным образом
- Фиксируем матрицу для слов - **замораживаем веса**

# Вектора для новых текстов



Матрица векторов  
для документов

Матрица векторов  
для слов

- Инициализируем новый вектор случайным образом
- Фиксируем матрицу для слов - **замораживаем веса**
- Выполняем несколько итераций и доучиваем вектор нового документа

# Paragraph2Vec

## PV-DM

- Сложнее
- Нужно одновременно обучать больше параметров
- Качество получаемых векторов лучше: SOTA на тот момент на нескольких задачах

## PV-DBOW

- Концептуально проще
- Качество векторов похуже

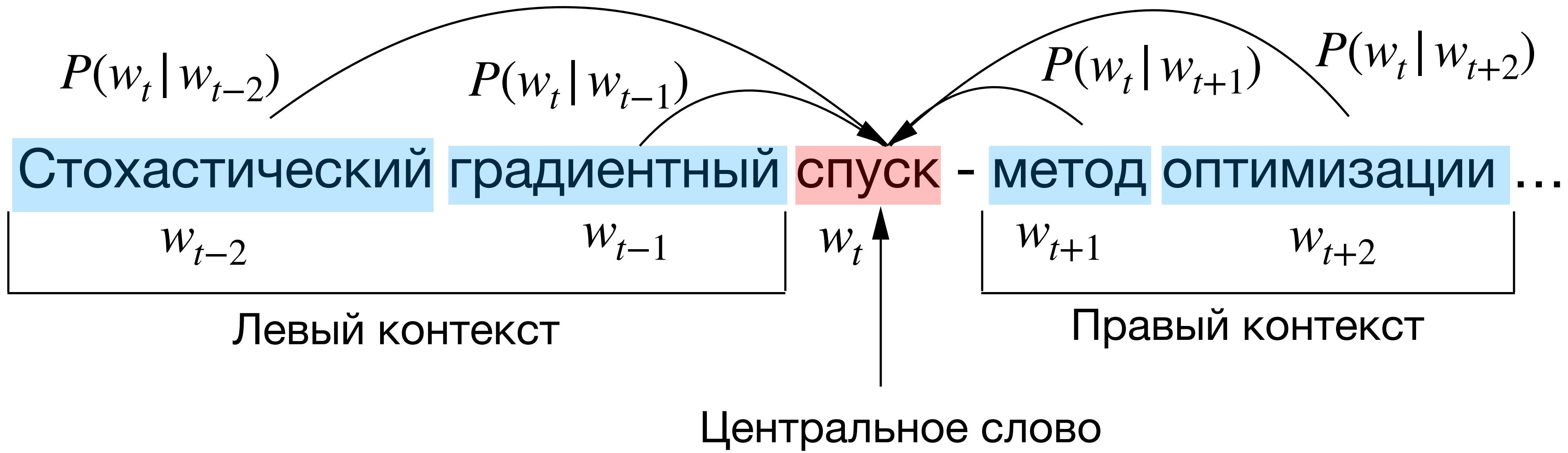
В статье указывают, что если обучить обоими способами и **сложить**, то это делает вектора более устойчивыми и работает еще лучше

# FastText



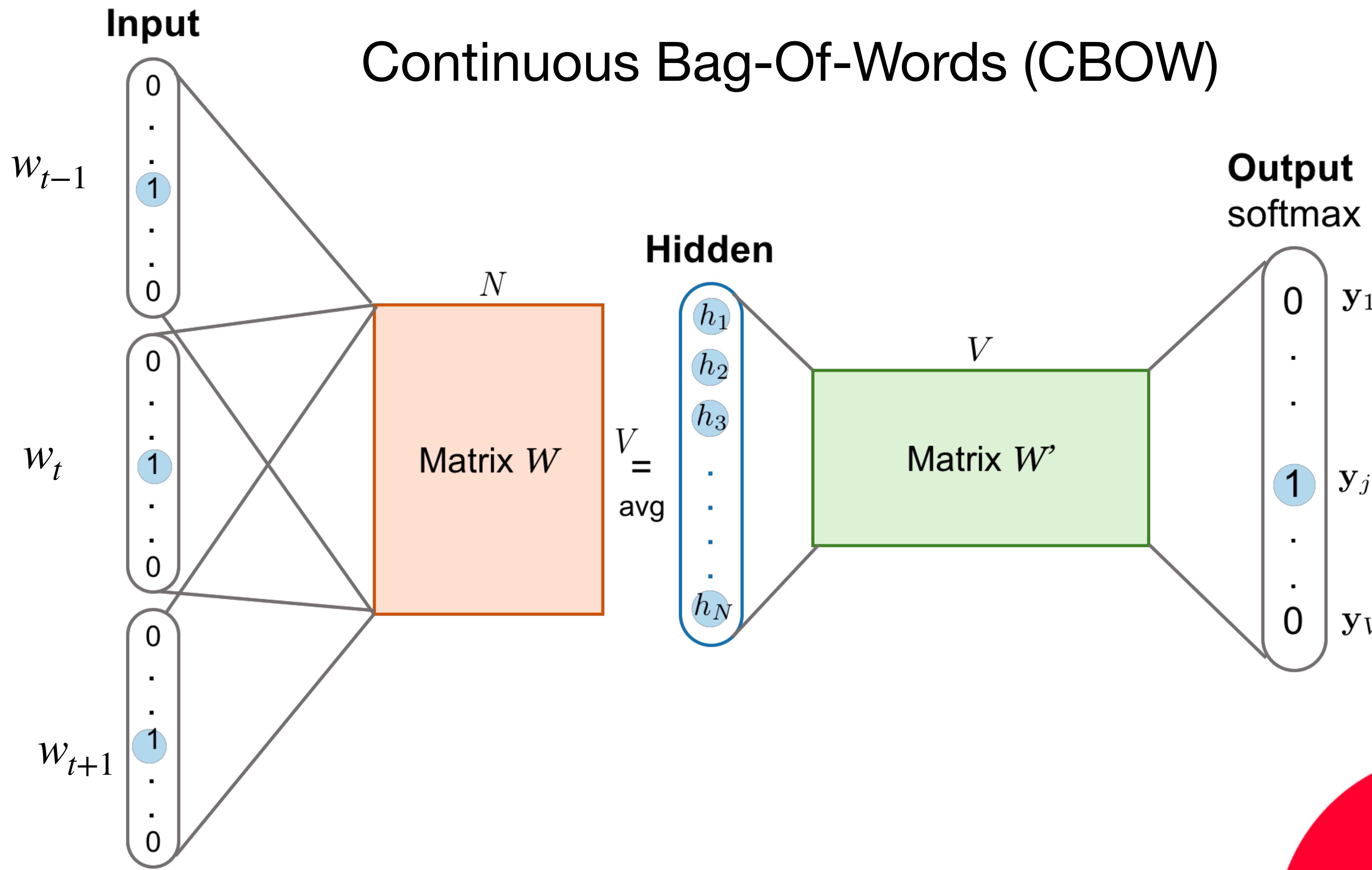
# Вспомним Word2vec

## Continuous Bag-Of-Words (CBOW)



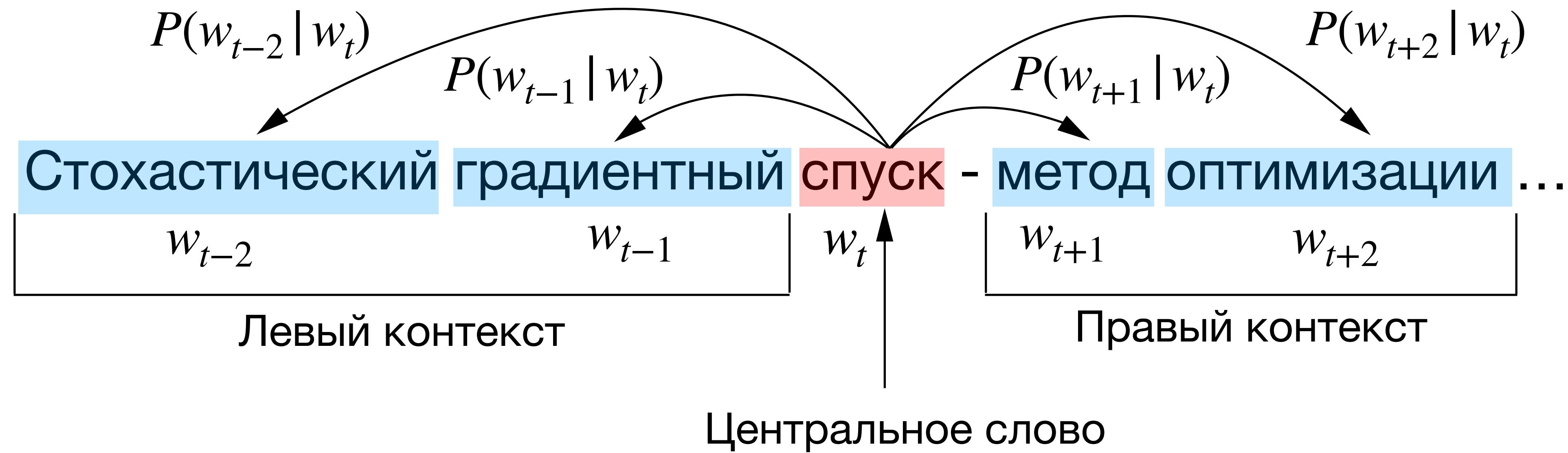
Для **контекстного окна** размера  $t$  предсказать центральное слово  $w_t$

# Вспомним Word2vec



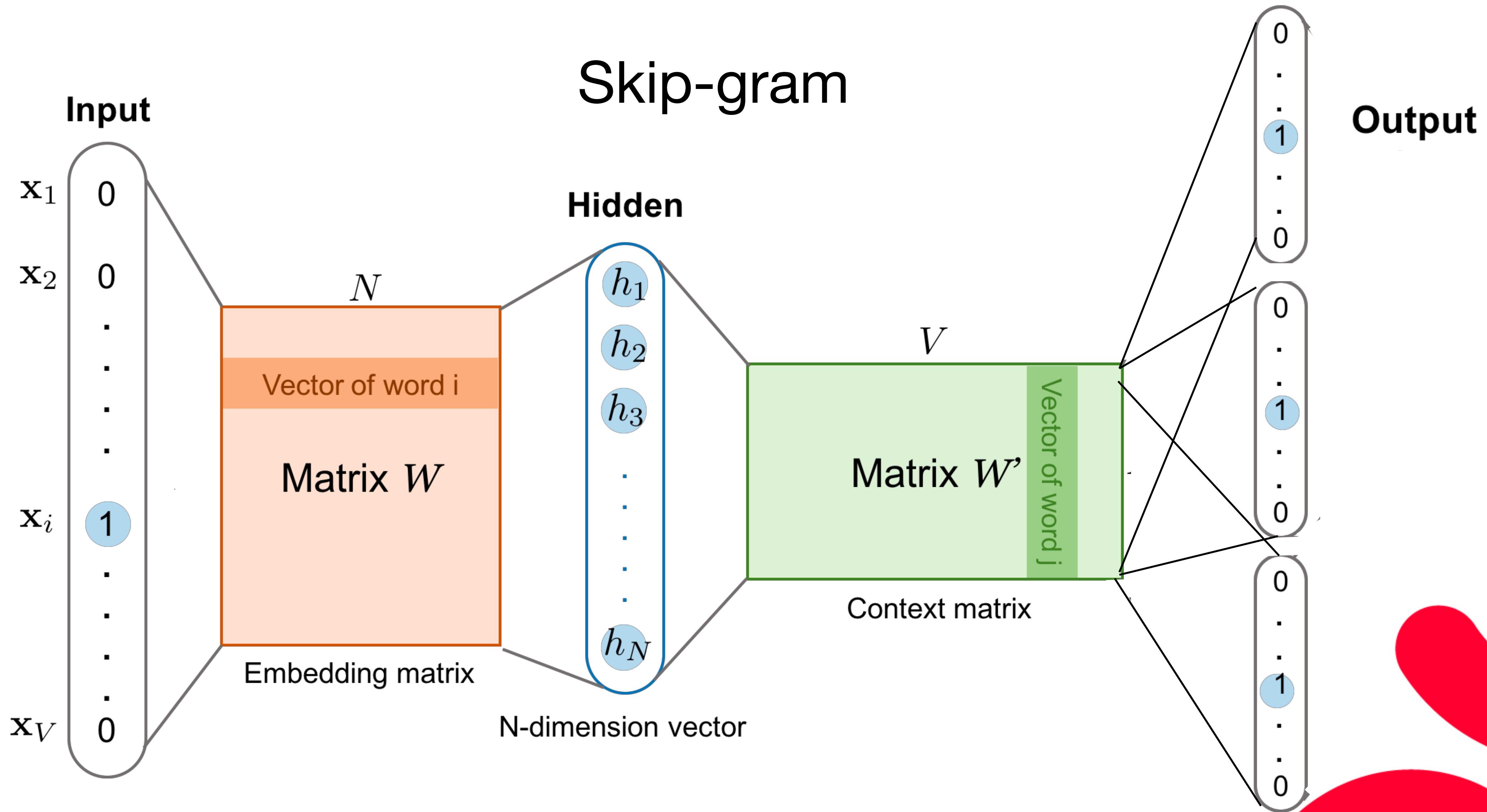
# Вспомним Word2vec

## Skip-Gram



Для каждого центрального слова  $w_t$  **предсказать слова из контекста** в окне размера  $m$

# Вспомним Word2vec



# FastText

Архитектура:

<https://arxiv.org/pdf/1607.01759.pdf>

<https://arxiv.org/pdf/1607.04606.pdf>

# FastText

Архитектура:

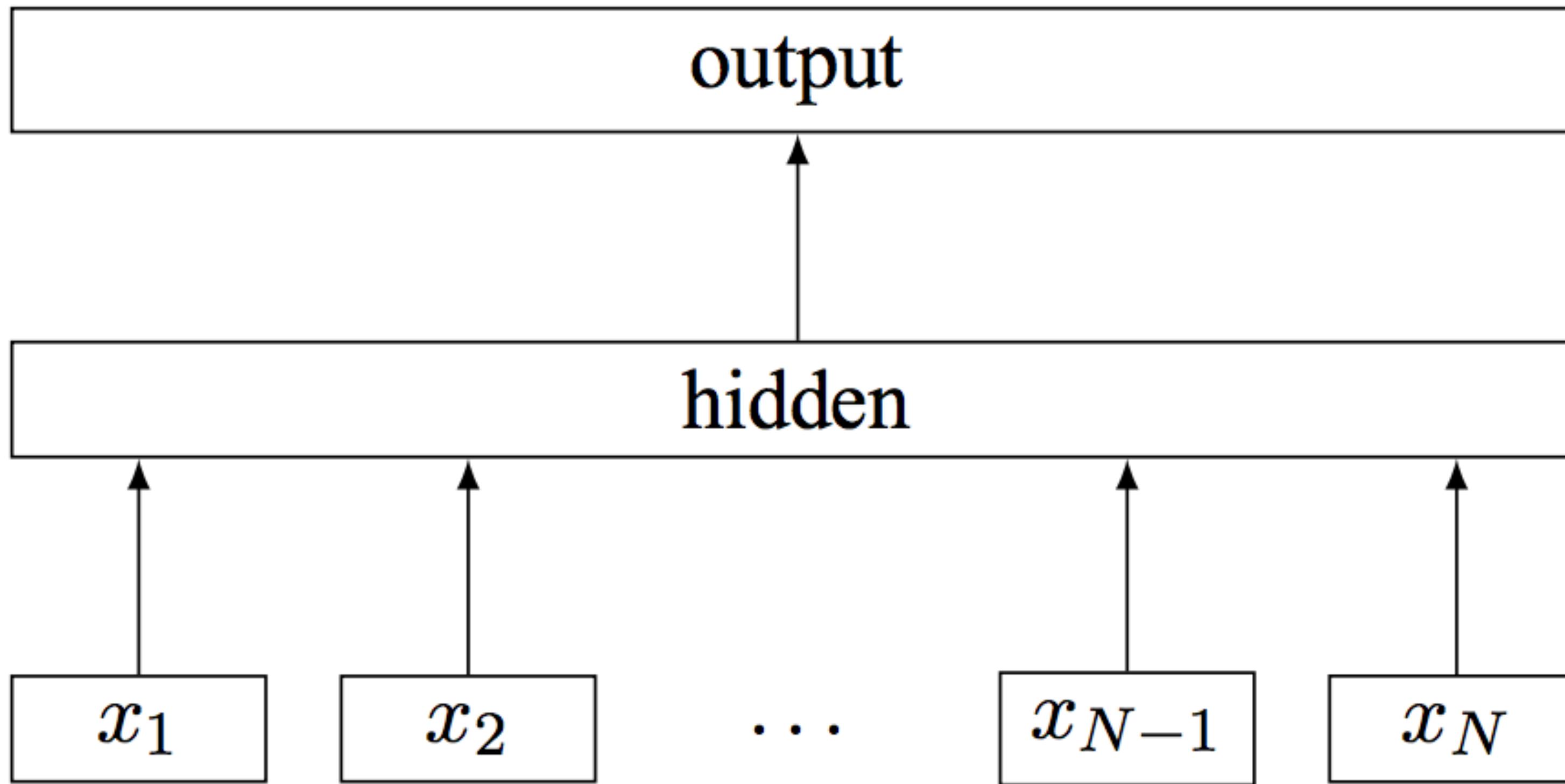


Рисунок взят из оригинальной статьи

# FastText



# FastText

## Идея фасттекста:

- Представляем слово в виде мешка символьных н-грамм разного порядка

# FastText

## Идея фасттекста:

- Представляем слово в виде мешка символьных н-грамм разного порядка
- Учим эмбеддинги для каждой н-граммы

# FastText

## Идея фасттекста:

- Представляем слово в виде мешка символьных н-грамм разного порядка
- Учим эмбеддинги для каждой н-граммы
- Суммируем эмбеддинги для всех н-грамм слова и получаем результирующий эмбеддинг слова

# FastText - n-gram representation

*where* →

# FastText - n-gram representation

*where* → <*where*> →  
 $n=3$

# FastText - n-gram representation

# FastText - n-gram representation

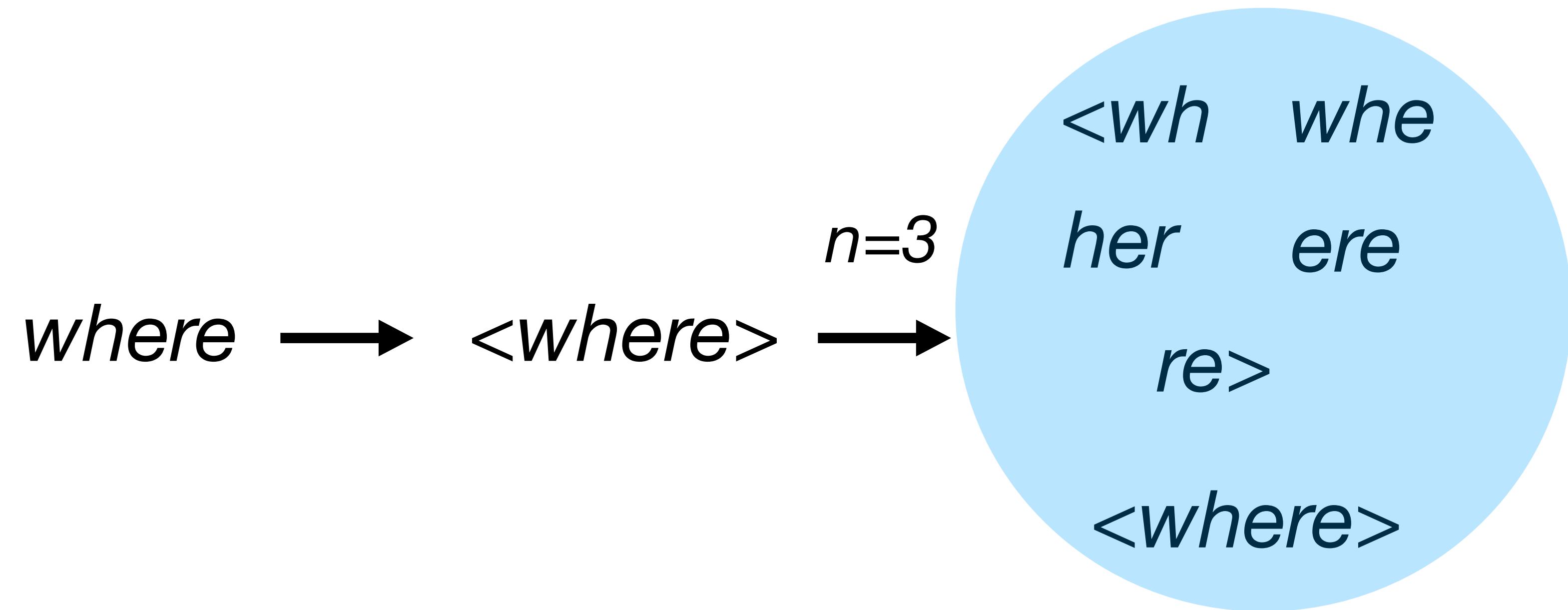
*where* → <*where*> →

$n=3$

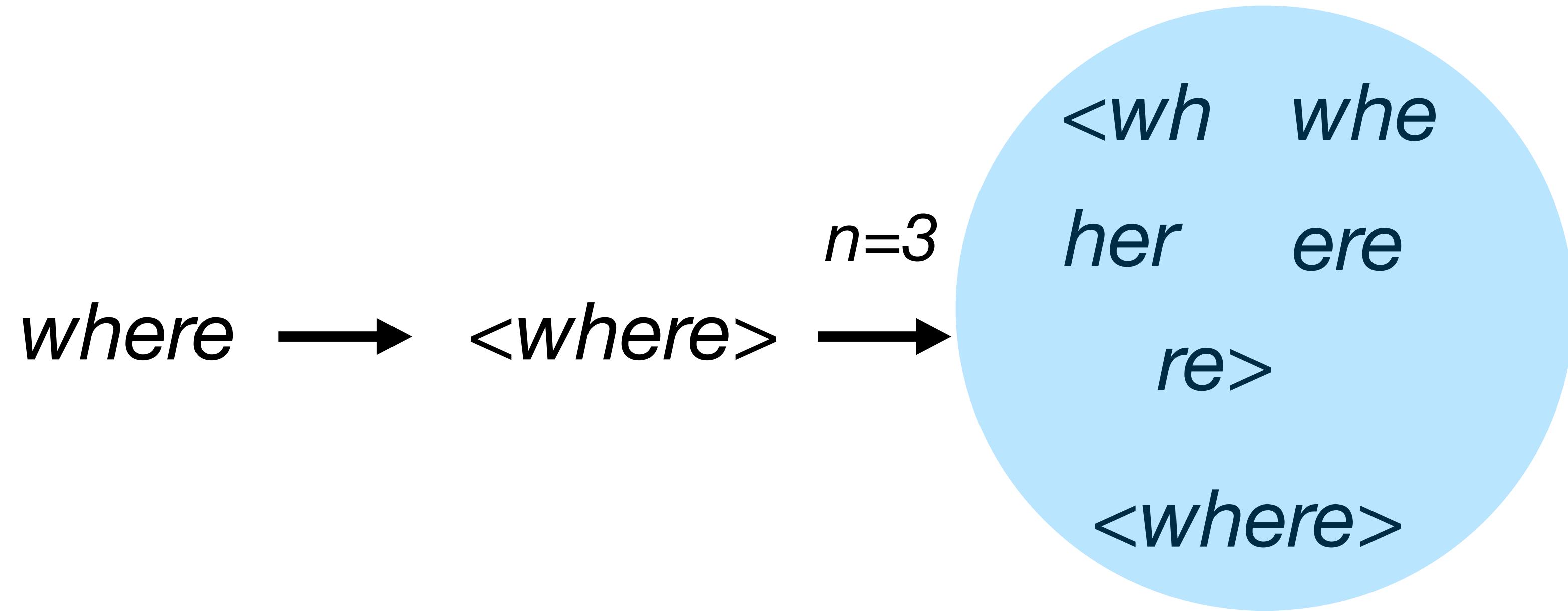
<*wh*    *whe*  
      *her*    *ere*  
          *re*>

<*where*>

# FastText - n-gram representation



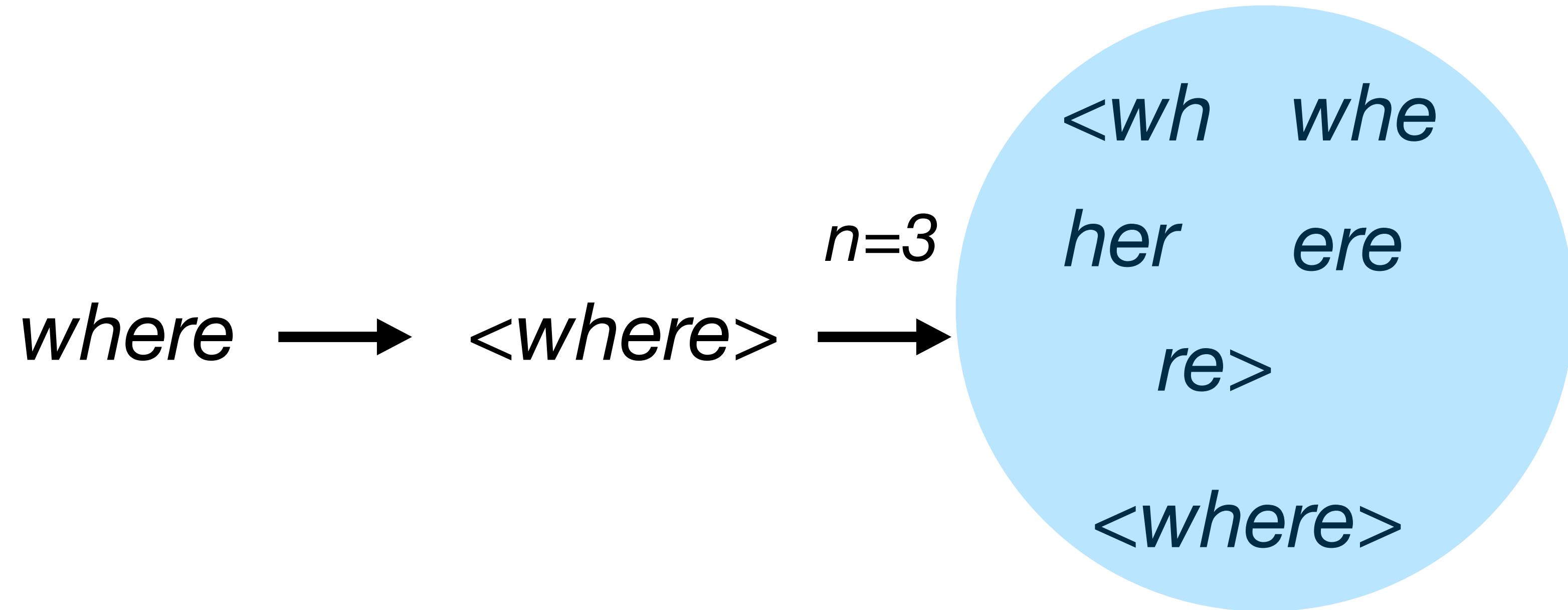
# FastText - n-gram representation



На практике извлекают нграммы в диапазоне [3;6]

Именно такой порядок нграмм подобран на основе экспериментов

# FastText - n-gram representation



На практике извлекают нграммы в диапазоне [3;6]

Именно такой порядок нграмм подобран на основе экспериментов

**Как его логически объясняют?**

# FastText - учим эмбеддинги

При обучении применяют тот же трюк с Negative Sampling

# FastText - учим эмбеддинги

При обучении применяют тот же трюк с Negative Sampling

То есть решаем задачу бинарной классификации, выбирая N  
случайных слов в качестве негативных примеров

$$loss = \log \left( 1 + e^{-s(w_t, w_c)} \right) + \sum_{n \in \mathcal{N}_{t,c}} \log \left( 1 + e^{s(w_t, n)} \right)$$

# FastText - учим эмбеддинги

При обучении применяют тот же трюк с Negative Sampling

То есть решаем задачу бинарной классификации, выбирая N случайных слов в качестве негативных примеров

$$loss = \log \left( 1 + e^{-s(w_t, w_c)} \right) + \sum_{n \in \mathcal{N}_{t,c}} \log \left( 1 + e^{s(w_t, n)} \right)$$

$s(w_t, w_c)$  - мера «похожести» слов

$$s(w_t, w_c) = \mathbf{u}_{w_t}^\top \mathbf{v}_{w_c}$$

# FastText - учим эмбеддинги

При обучении применяют тот же трюк с Negative Sampling

То есть решаем задачу бинарной классификации, выбирая N случайных слов в качестве негативных примеров

$$loss = \log \left( 1 + e^{-s(w_t, w_c)} \right) + \sum_{n \in \mathcal{N}_{t,c}} \log \left( 1 + e^{s(w_t, n)} \right)$$

$s(w_t, w_c)$  - мера «похожести» слов

$$s(w_t, w_c) = \mathbf{u}_{w_t}^\top \mathbf{v}_{w_c}$$

Интуитивно: хотим, чтобы  $s$  для слов из контекста был **больше, чем для случайных**

# FastText - учим эмбеддинги

## Negative Sampling

- 1) Сэмплируют негативные пары из  $\frac{\sqrt{U(w)}}{Z}, Z = \sum_w \sqrt{U(w)}$

# FastText - учим эмбеддинги

## Negative Sampling

- 1) Сэмплируют негативные пары из  $\frac{\sqrt{U(w)}}{Z}, Z = \sum_{w=1}^W \sqrt{U(w)}$
- 2) Выбрасывают слишком частотные слова

# FastText - учим эмбеддинги

## Negative Sampling

- 1) Сэмплируют негативные пары из  $\frac{\sqrt{U(w)}}{Z}, Z = \sum_w \sqrt{U(w)}$
- 2) Выбрасывают слишком частотные слова

Вероятность взять слово:

$$P(w) = \sqrt{\frac{t}{f(w)}} + \frac{t}{f(w)}$$

$$f(w) = \frac{\text{count}_w}{\text{total no of tokens}}$$
$$t = 10^{-4}$$

# FastText - учим эмбеддинги

## Negative Sampling

- 1) Сэмплируют негативные пары из  $\frac{\sqrt{U(w)}}{Z}, Z = \sum_w \sqrt{U(w)}$
- 2) Выбрасывают слишком частотные слова

Вероятность взять слово:

$$P(w) = \sqrt{\frac{t}{f(w)}} + \frac{t}{f(w)}$$

$$f(w) = \frac{\text{count}_w}{\text{total no of tokens}}$$

$t = 10^{-4}$

В экспериментах берут количество негативных примеров N=5.

Является гиперпараметром

# FastText - учим эмбеддинги

Учить эмбеддинги нграмм можно точно так же 2мя способами:

- СВОW - предсказываем центральное слово
- SkipGram - предсказываем контекст

Как правило, вектора skip-gram получаются более качественными.

# FastText - суммируем эмбеддинги

После обучения векторов получают суммарный вектор слова:

# FastText - суммируем эмбеддинги

После обучения векторов получают суммарный вектор слова:

$$s(w, c) = \sum_{g \in \mathcal{G}_w} \mathbf{z}_g^\top \mathbf{v}_c$$

Было:

$$s(w_t, w_c) = \mathbf{u}_{w_t}^\top \mathbf{v}_{w_c}$$

# FastText - суммируем эмбеддинги

После обучения векторов получают суммарный вектор слова:

$$s(w, c) = \sum_{g \in \mathcal{G}_w} \mathbf{z}_g^\top \mathbf{v}_c$$

Было:

$$s(w_t, w_c) = \mathbf{u}_{w_t}^\top \mathbf{v}_{w_c}$$

$\mathcal{G}_w \subset \{1, \dots, G\}$  - набор нграмм для данного слова w

$\mathbf{z}_g$  - эмбеддинг конкретной нграммы g

# FastText - supervised

Кроме обучения эмбеддингов unsupervised (на неразмеченных данных) FastText может учить вектора **под конкретную задачу** при наличии меток класса

Text_1	Label_1
Text_2	Label_2
...	
Text_n	Label_n

# FastText - supervised

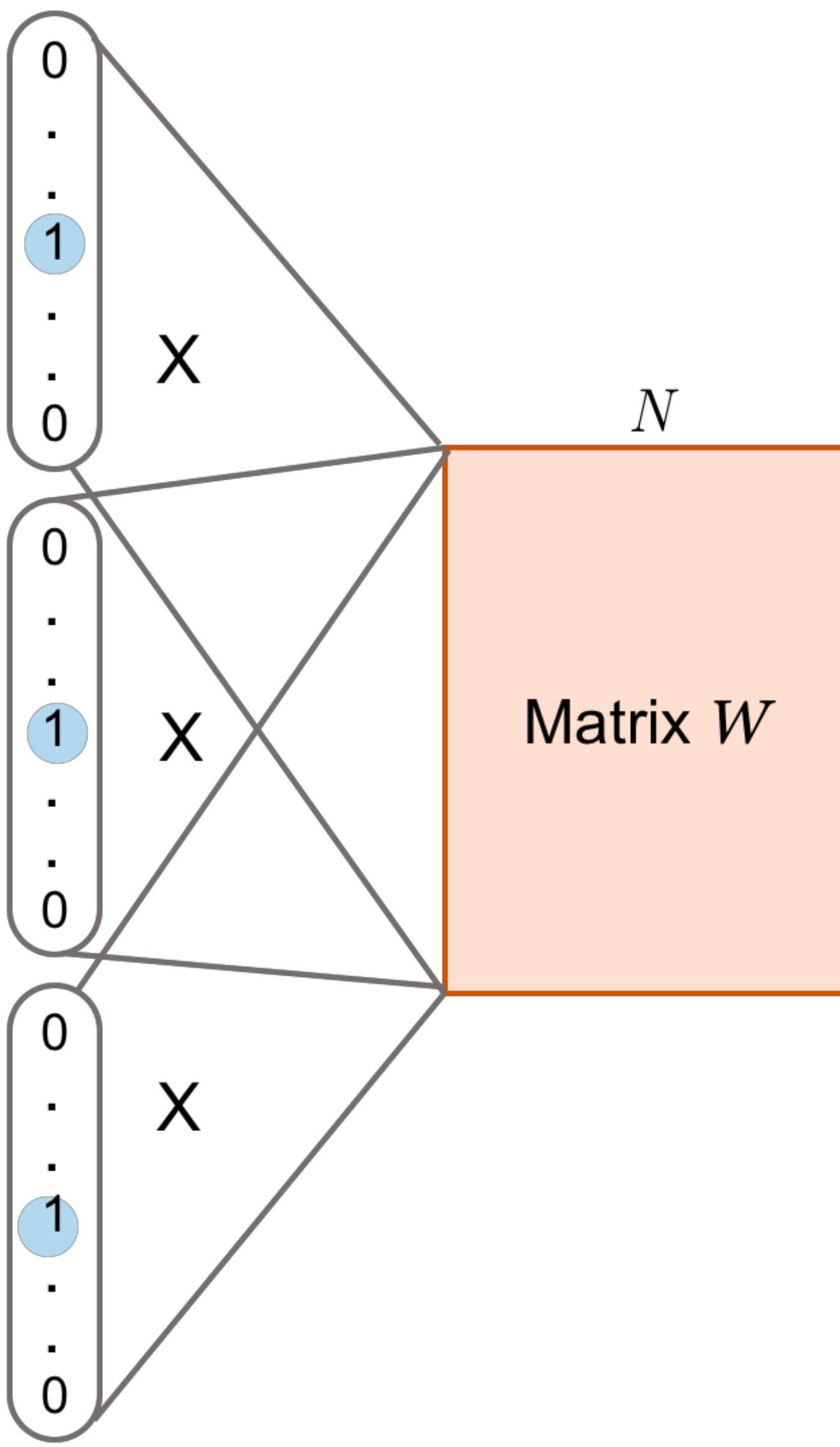
Кроме обучения эмбеддингов unsupervised (на неразмеченных данных) FastText может учить вектора **под конкретную задачу** при наличии меток класса

Text_1	Label_1
Text_2	Label_2
...	
Text_n	Label_n

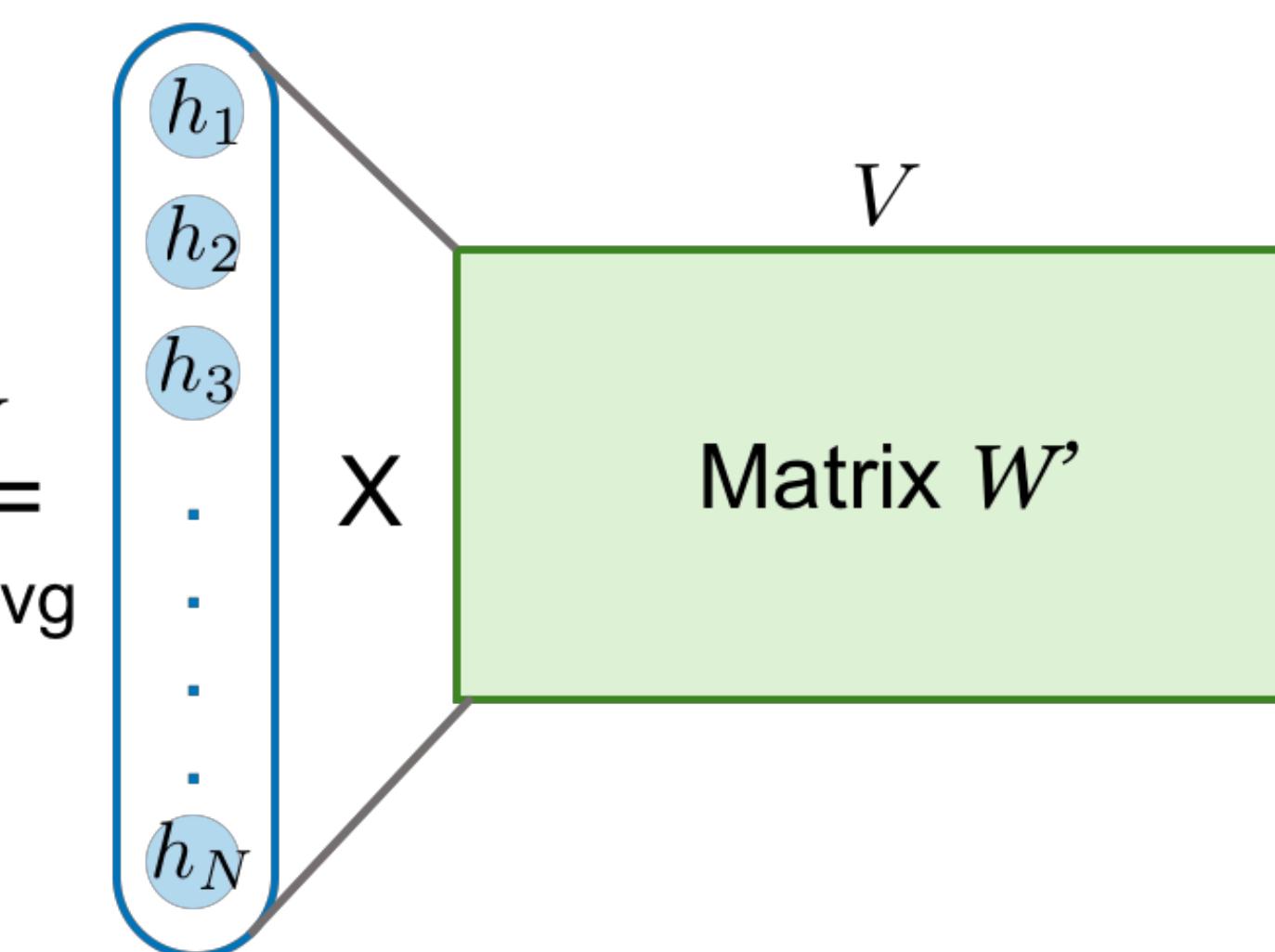
Как выглядит процесс обучения векторов ?

# FastText - supervised

**Input**



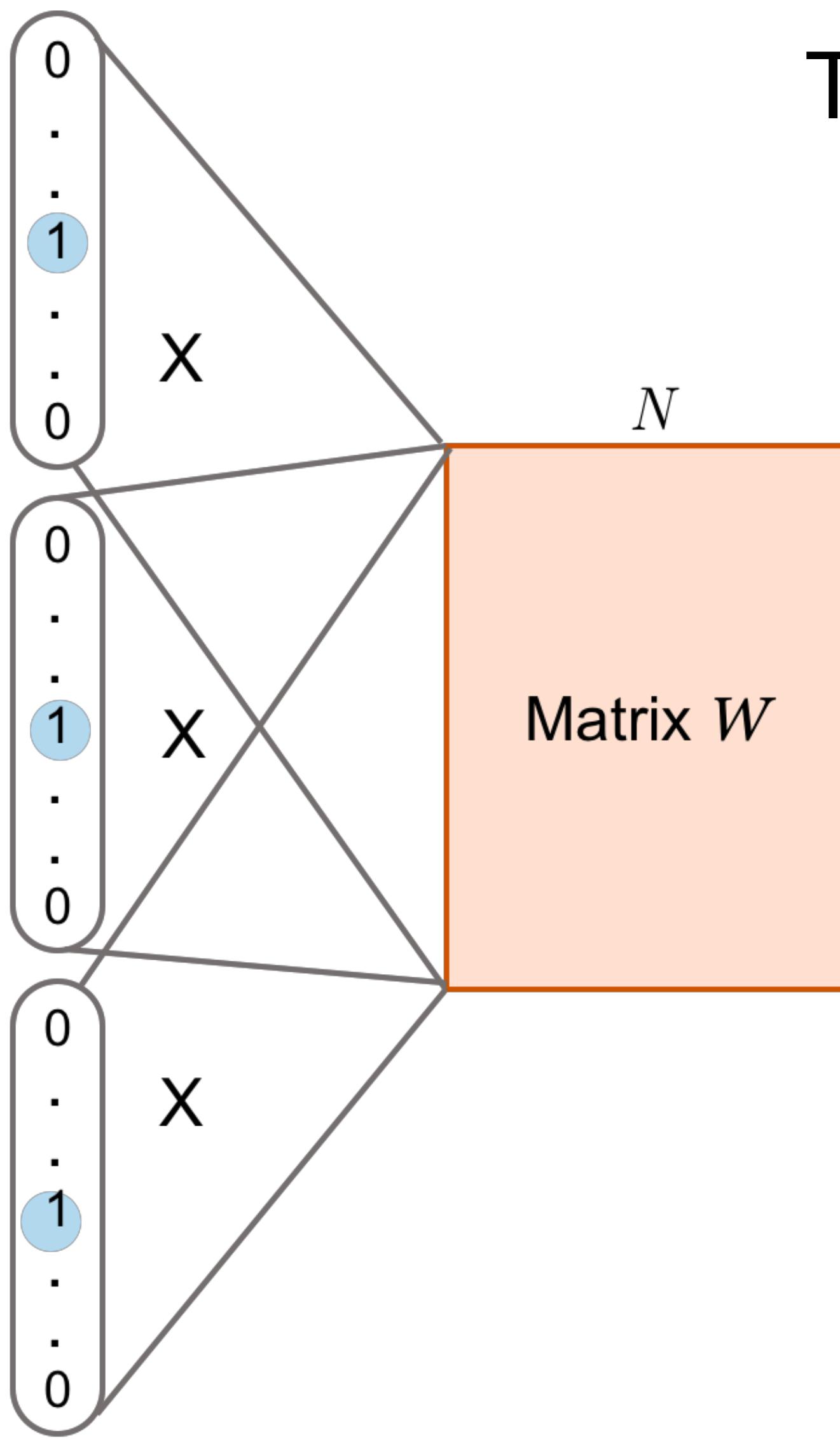
**Hidden**



?

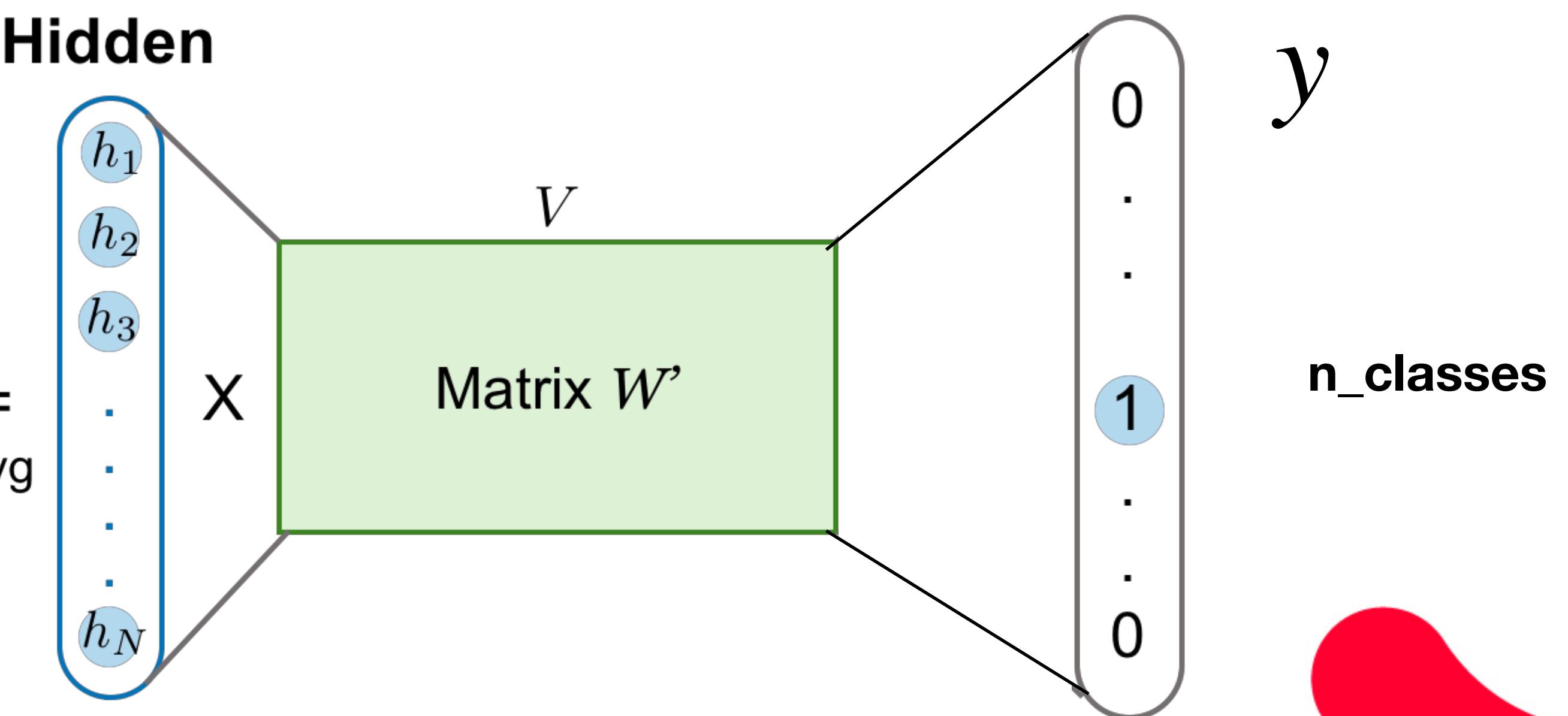
# FastText - supervised

**Input**



Target - one-hot вектор нужного класса

**Hidden**



# FastText

## Особенности реализации

- хеширует символьные нграммы (куда ж без этого)
- слова не хешируют
- можно задавать нграммы по словам (тоже хешируются)
- можно получить сжатую версию модели (весит на порядок меньше)
- soft-sliding window - рандомно сэмплируют каждый раз размер окна из  $U(1, \text{window\_size})$

# FastText

## Плюсы:

- может работать со словами вне словаря
- более качественные вектора для редких слов
- multi-core
- really fast!



# Bonus:

## Byte Pair Encoding



# Byte pair encoding (BPE)

# Byte pair encoding (BPE)

Алгоритм сжатия данных; применяется в задачах машинного перевода.



# Byte pair encoding (BPE)

Алгоритм сжатия данных; применяется в задачах машинного перевода.

Идея - рекурсивно замещать самую частую пару байтов новым байтом.

# Byte pair encoding (BPE)

Алгоритм сжатия данных; применяется в задачах машинного перевода.

Идея - рекурсивно замещать самую частую пару байтов новым байтом.



# Byte pair encoding (BPE)

карл у клары украл кораллы клара у карла украла кларнет

# Byte pair encoding (BPE)

klalpl u kllalrys yklpla l kloplalllys kllalpla u klapl lla yklpla lla kllalr hlelt

# Byte pair encoding (BPE)

klap<sub>l</sub> у к<sub>l</sub>lap<sub>ы</sub> ул<sub>к</sub>lpla<sub>л</sub> klopla<sub>л</sub>л<sub>ы</sub> к<sub>л</sub>lap<sub>a</sub> у kl<sub>а</sub>p<sub>l</sub>ла ул<sub>к</sub>lpla<sub>л</sub>ла к<sub>л</sub>lap<sub>и</sub>н<sub>е</sub>т

# Byte pair encoding (BPE)

klap<sub>l</sub> у klap<sub>ары</sub> yklpal<sub>л</sub> klop<sub>аллы</sub> klap<sub>ла</sub> у klap<sub>ла</sub> yklpal<sub>ла</sub> klap<sub>и</sub>elt

# Byte pair encoding (BPE)

klapIл у кIлIарIы уIкIрал kloIралIлIы кIлIарIа у klapIлIа уIкIралIа кIлIарIнleIt

# Byte pair encoding (BPE)

klapIл у клары укIрал клорапIллы клапла у кlapIлла укIралла кларIнлеit

# Byte pair encoding (BPE)

клары у краллы клапла у кралла клары

# Byte pair encoding (BPE)

klapл у клары ук|рал klo|ралл|лы клара у klap|ла ук|рала клар|н|еит

# Byte pair encoding (BPE)

карл у клары ук|рал klo|ралл|ы клара у кар|ла ук|рал|а клар|н|е|т

# Byte pair encoding (BPE)

карл у клары укірал kloралллы кларla у карлla укіралla кларінлеіт

# Byte pair encoding (BPE)

карл у клары украл kloраллы клара у карла украла кларынелт

# Byte pair encoding (BPE)

карл у клар ы украл к о рал л ы клар а у карл а украл а клар н е т

# Byte pair encoding (BPE)



# Byte pair encoding (BPE)

Как правило, задают размер словаря, который хотят получить, и запускают процедуру рекурсивного объединения.

BPE выучивает словарь частотных коллокаций.

И применяет его на тестовой коллекции данных.

<https://github.com/google/sentencepiece> - утилита для BPE

@mo

Внимание!!!



Спасибо за внимание