NLP до 2017

Елена Кантонистова

NLP ROADMAP

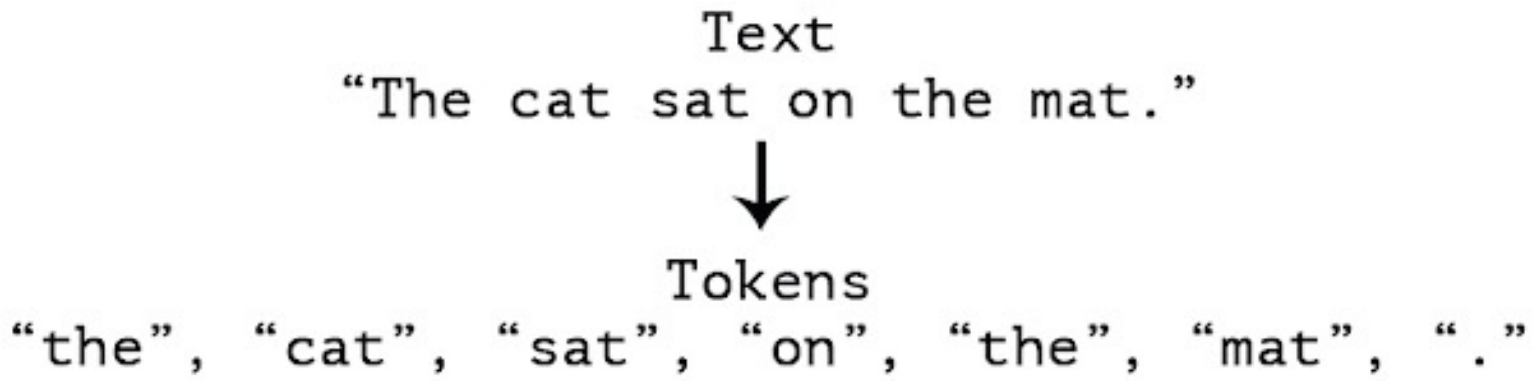

Re
Skip-Gram
Word2Vec
Stop Words
Bag of Words
CBOW
NLP
Lemmatization
Tokenization
LSTM
RNN
Attention Models
One Hot Representation

ТЕРМИНОЛОГИЯ

- документ = текст
- корпус – набор документов
- токен – формальное определение “слова”; токен может не иметь смыслового значения (например, “12fdh” или “авыдшл”), но обычно отделен от остальных токенов пробелами или знаками препинания

ТОКЕНИЗАЦИЯ ТЕКСТА

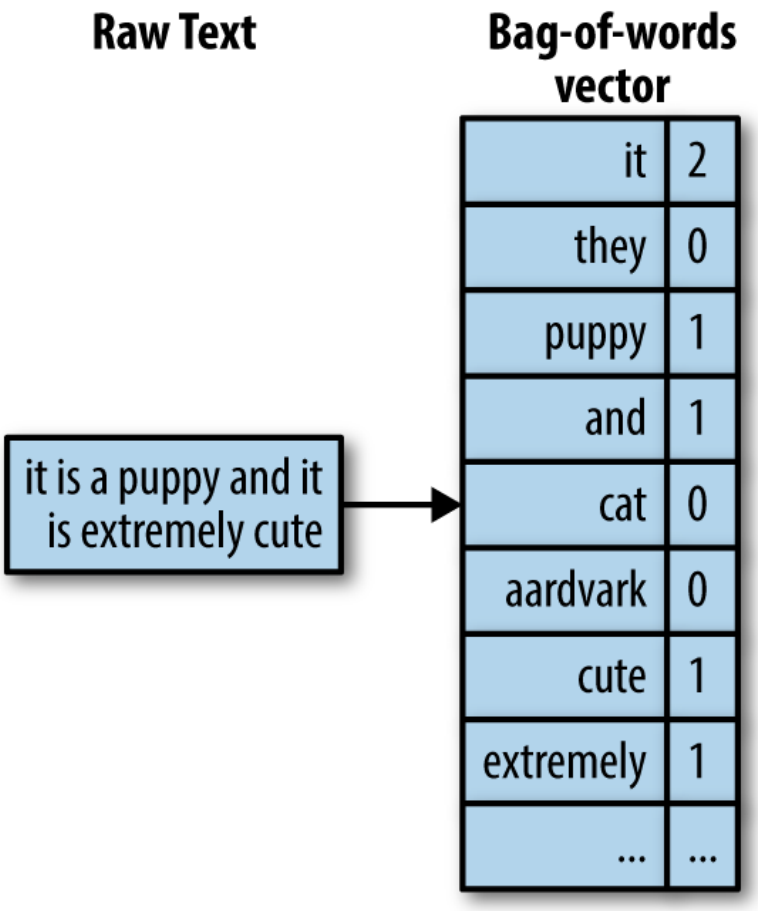
Чтобы работать с текстом, необходимо разбить его на токены. В простейшем случае токены – это слова (а также наборы букв, знаки препинания и т.д.).



МЕТОДЫ КОДИРОВАНИЯ ТЕКСТОВЫХ ДАННЫХ

BAG OF WORDS (МЕШОК СЛОВ)

- По корпусу создадим словарь из всех встречающихся в нем слов (можно убрать общеупотребительные часто встречающиеся слова и очень редкие слова).
- Каждое слово закодируем вектором, в котором стоит единица на месте, соответствующем месту этого слова в словаре, все остальные компоненты вектора – 0.
- Для кодирования документа сложим коды всех его слов.



BAG OF WORDS (ПРИМЕР)

Пусть корпус состоит из следующих документов:

- D1 - “I am feeling very happy today”
- D2 - “I am not well today”
- D3 - “I wish I could go to play”

Кодировка этих документов будет такой:

	I	am	feeling	very	happy	today	not	well	wish	could	go	to	play
D1	1	1	1	1	1	1	0	0	0	0	0	0	0
D2	1	1	0	0	0	1	1	1	0	0	0	0	0
D3	2	0	0	0	0	0	0	0	1	1	1	1	1

BAG OF WORDS

Используя bag of words (BOW), мы теряем информацию о порядке слов в документе.

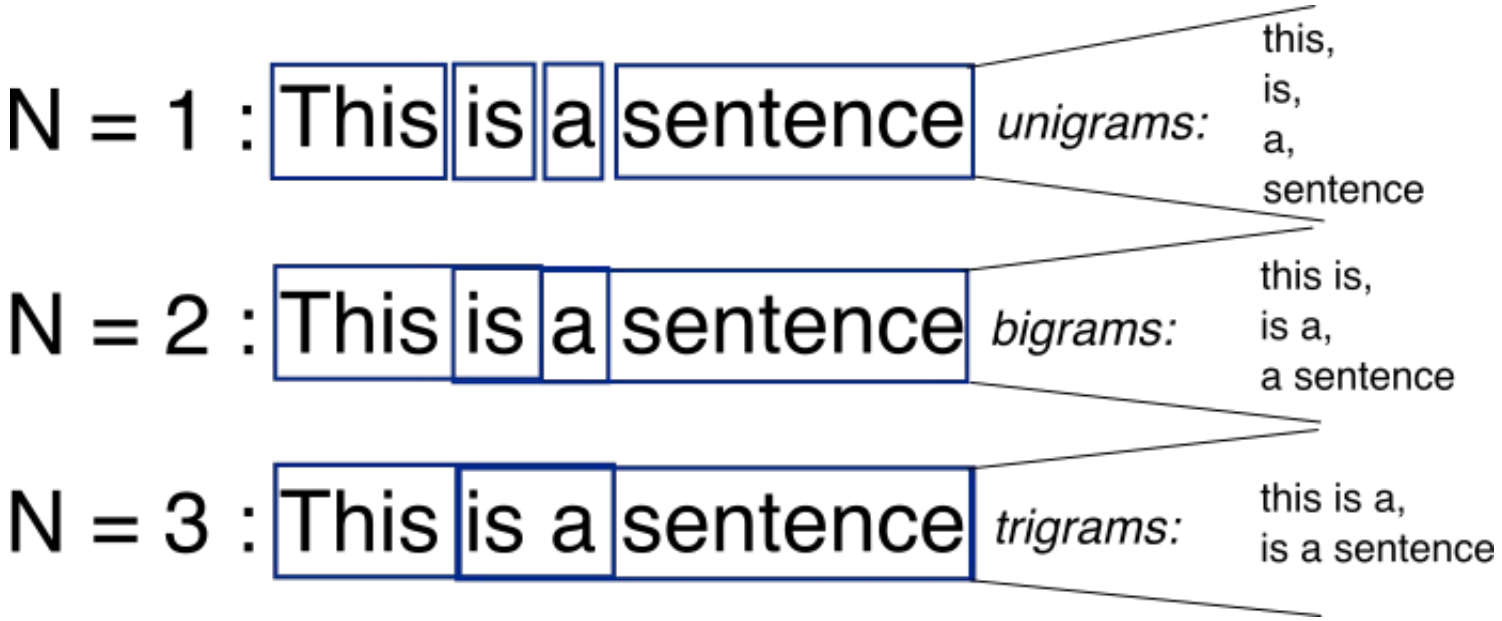
Пример: векторы документов “I have no cats” и “No, I have cats” будут идентичны.

N-GRAM BAG OF WORDS

В качестве слов в словаре можно использовать:

- N-граммы из букв (наборы букв длины N в слове)
- N-граммы из слов (наборы фраз длины N в документе)

Такой подход поможет учесть сходственные слова и опечатки.



TF-IDF

- слова, которые редко встречаются в корпусе, но присутствуют в документе, могут оказаться важными для характеристики документа.
- слова, которые встречаются во всех документах, наоборот, не важны.

TF-IDF

Tf-Idf (term frequency – inverse document frequency):

- $tf(t,d)$ - *частота вхождения слова* t *в документ* d:

$$\boldsymbol{tf(t,d)} = \frac{\boldsymbol{n_t}}{\sum_{\boldsymbol{k}} \boldsymbol{n_k}} = \frac{\text{число вхождений слова } \boldsymbol{t} \text{ в документ}}{\text{общее число слов в документе}}$$

$tf(t,d)$ показывает важность слова t в документе d.

TF-IDF

- $tf(t, d)$ - *частота вхождения слова* t *в документ* d:

$$\boldsymbol{tf(t, d)} = \frac{\boldsymbol{n_t}}{\sum_{\boldsymbol{k}} \boldsymbol{n_k}} = \frac{\text{число вхождений слова } \boldsymbol{t} \text{ в документ}}{\text{общее число слов в документе}}$$

$tf(t, d)$ *показывает важность слова* t *в документе* d.

- $idf(t, D)$ - *величина, обратная частоте, с которой слово* t *встречается в документах корпуса* D.

$$\boldsymbol{idf(t, D)} = \mathbf{log} \frac{|\boldsymbol{D}|}{|\{\boldsymbol{d_i} \in \boldsymbol{D} \mid \boldsymbol{t} \in \boldsymbol{d_i}\}|},$$

$|D|$ — число документов *в корпусе,*

$|\{d_i \in D \mid t \in d_i\}|$ - *число документов, в которых встречается слово* t

Учёт idf *уменьшает вес часто используемых в корпусе слов.*

WORD2VEC

Цель: для каждого слова из текста получить такой числовой вектор, чтобы векторы похожих по смыслу слов были “близки”.

WORD2VEC

Цель: для каждого слова из текста получить такой числовой вектор, чтобы векторы похожих по смыслу слов были “близки”.

Идея: слова, встречающиеся в похожих контекстах, имеют близкие значения.

- В 2013 году Томаш Миколов и его коллеги предложили word2vec – нейронную сеть, которую можно быстро обучить на огромном объеме текстов для получения векторов слов.

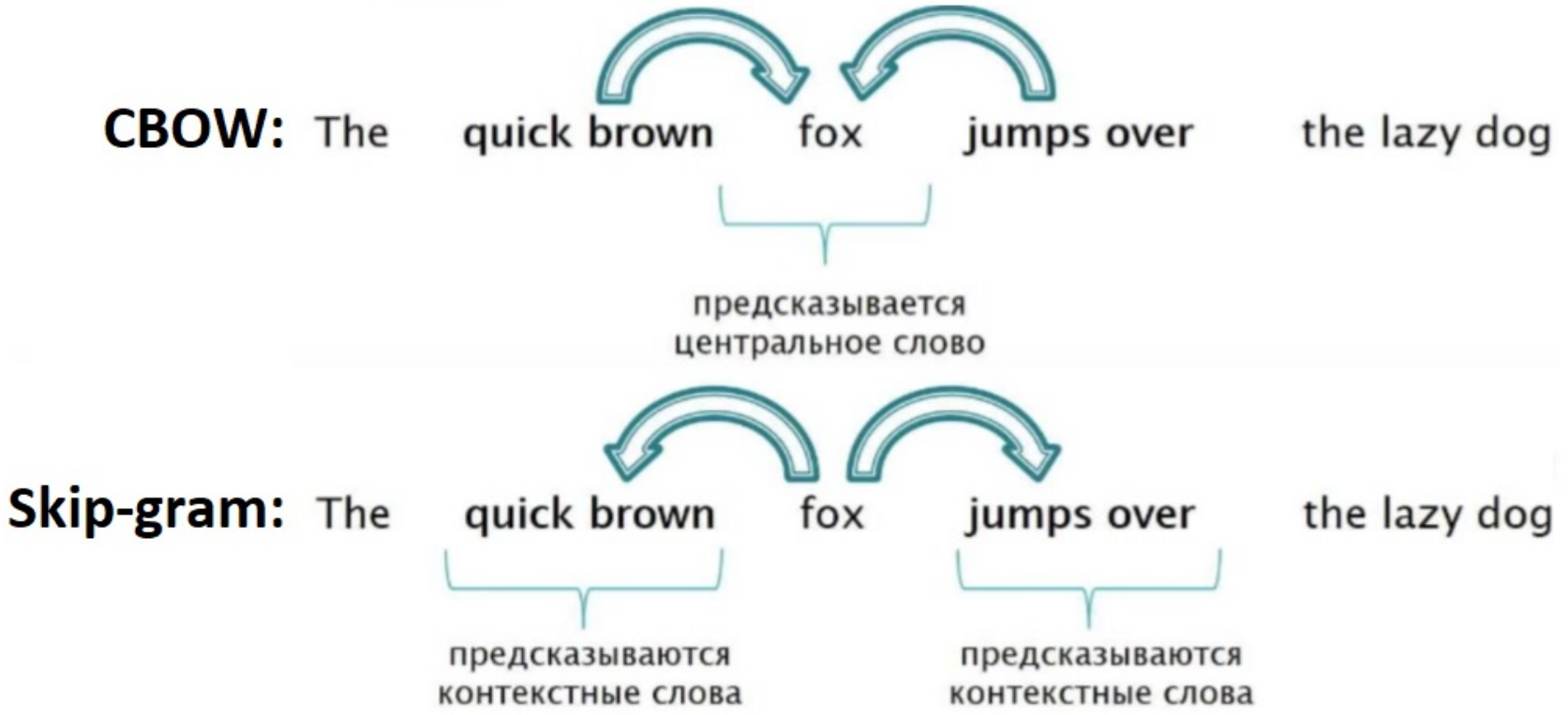
ВЕРОЯТНОСТНАЯ ПОСТАНОВКА ЗАДАЧИ

- будем предсказывать вероятность слова по его окружению (контексту)
- будем учить такие вектора слов, чтобы *вероятность, присваиваемая моделью слову была близка к вероятности встретить это слово в этом окружении в реальном тексте*.

WORD2VEC

Есть две разные модели word2vec – CBOW и Skip-gram.

- CBOW (Continuous Bag of Words) предсказывает вероятность слова по данному контексту
- Skip-gram (“словосочетание с пропуском”) предсказывает по данному слову вектор контекста



МОДЕЛЬ CBOW

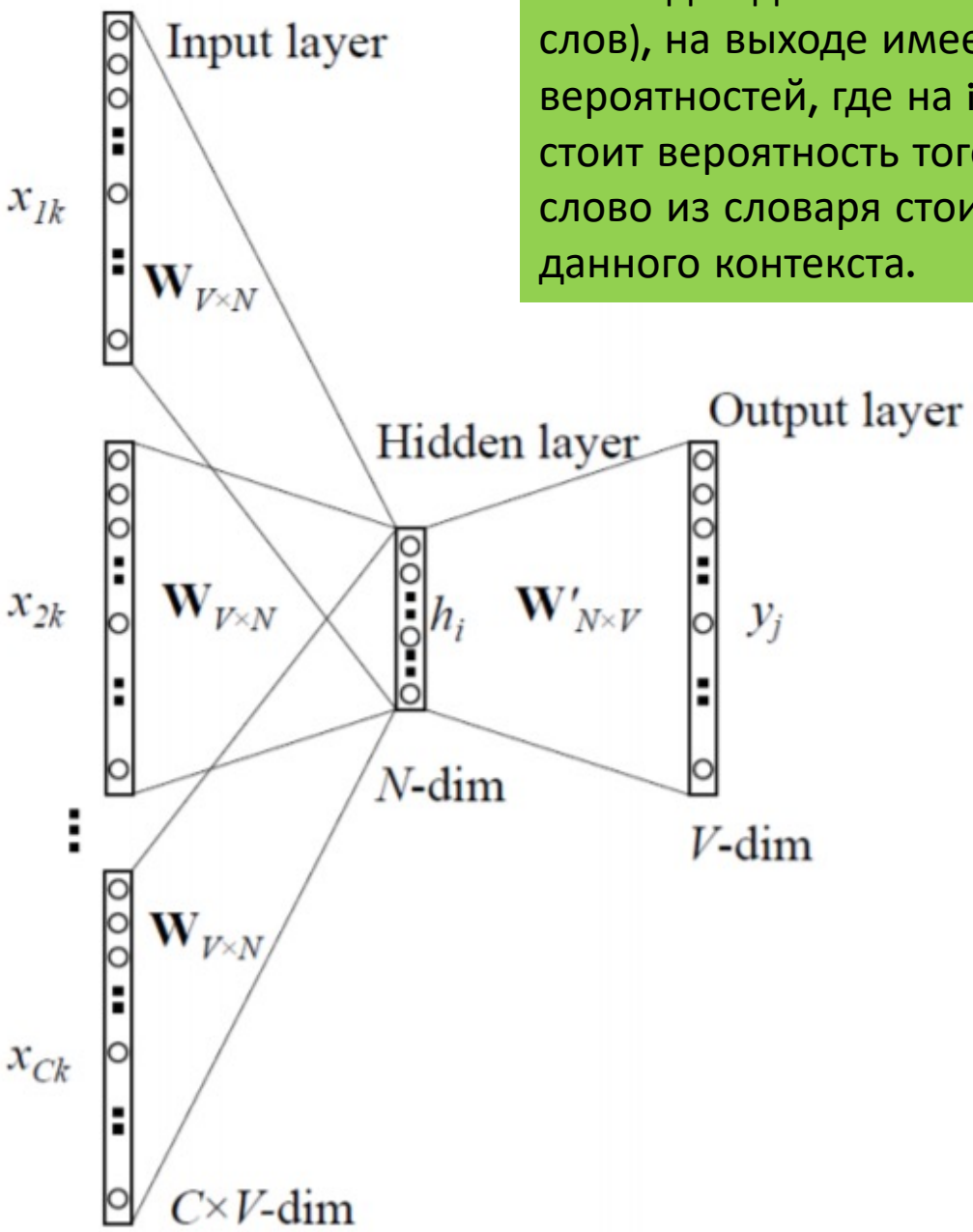
Обучаем модель с помощью *нейронной сети*.

- Скользящим окном ширины $2C + 1$ (слово и его контекст) проходим по всей коллекции
- **Вход (признаки):** one-hot представление слов контекста (векторы длины V)
- **Ответ (целевая переменная):** one-hot представление слова
- **Выход:** распределение вероятностей на словах коллекции (вектор длины V)
- **Оптимизируемый функционал –** минус логарифм правдоподобия:

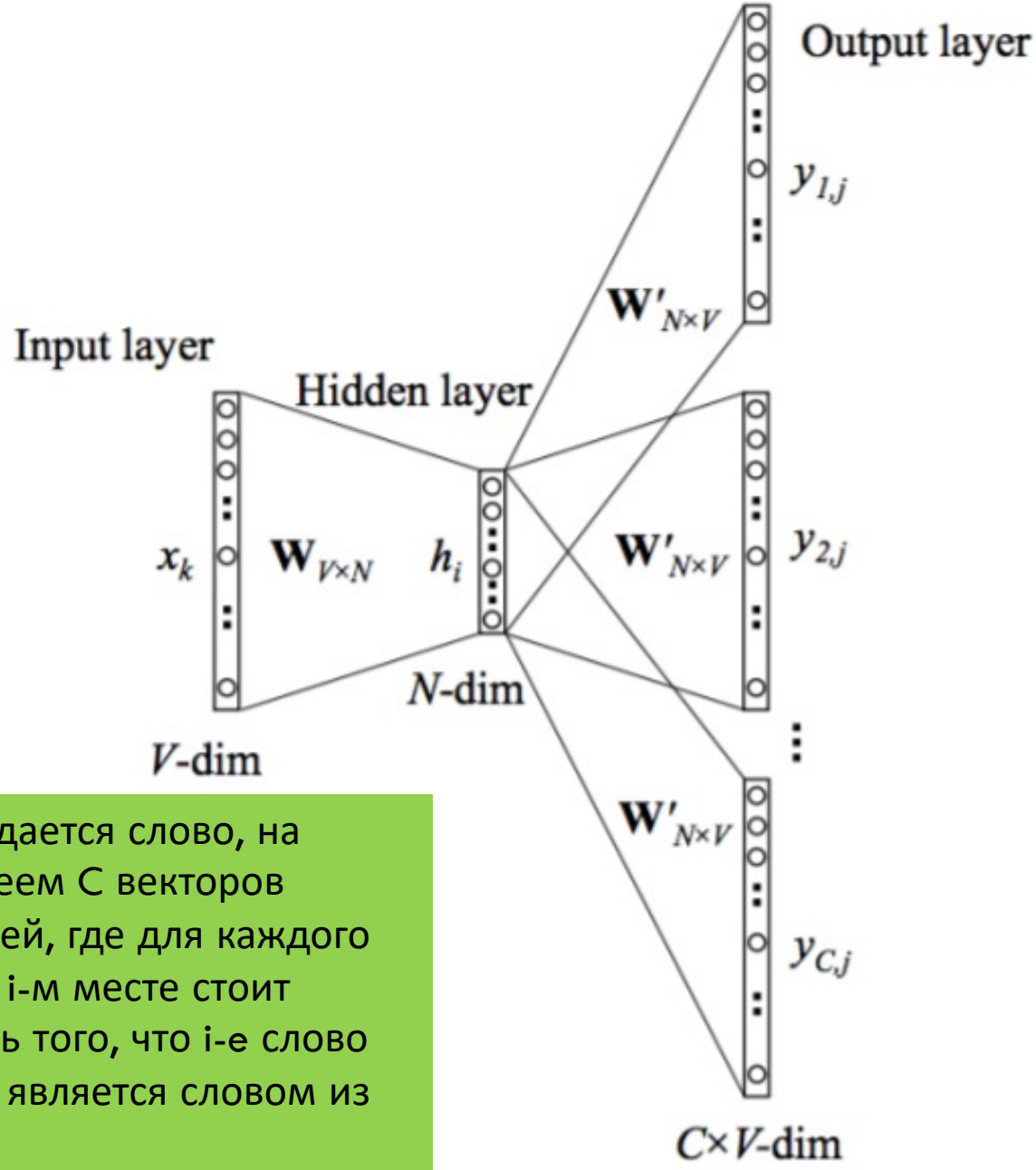
$$-\sum_{i=1}^{l}\sum_{j=1}^{n_i}\sum_{\substack{k=-\mathrm{C},\\ k\neq 0}}^{\mathrm{C}} \log p\left(w_{j+k}\middle|w_j\right) \to \min_{\{w\}}$$

МОДЕЛЬ CBOW

На вход подается контекст (C слов), на выходе имеем вектор вероятностей, где на i-м месте стоит вероятность того, что i-е слово из словаря стоит внутри данного контекста.



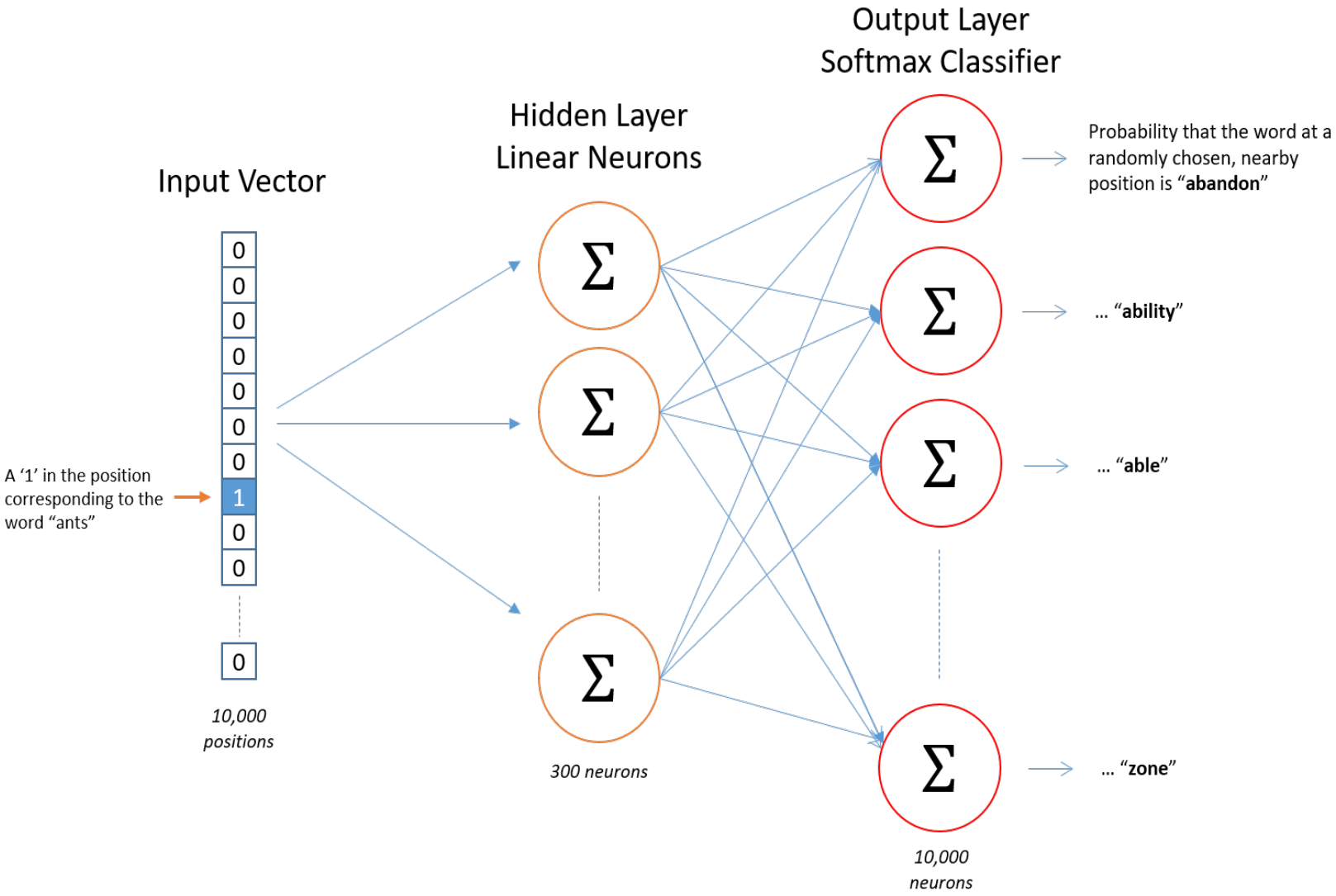
МОДЕЛЬ SKIP-GRAM



На вход подается слово, на выходе имеем C векторов вероятностей, где для каждого вектора на i-м месте стоит вероятность того, что i-е слово из словаря является словом из контекста.

МОДЕЛЬ SKIP-GRAM

На скрытом слое активации нет, на выходном слое – *softmax*.



http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

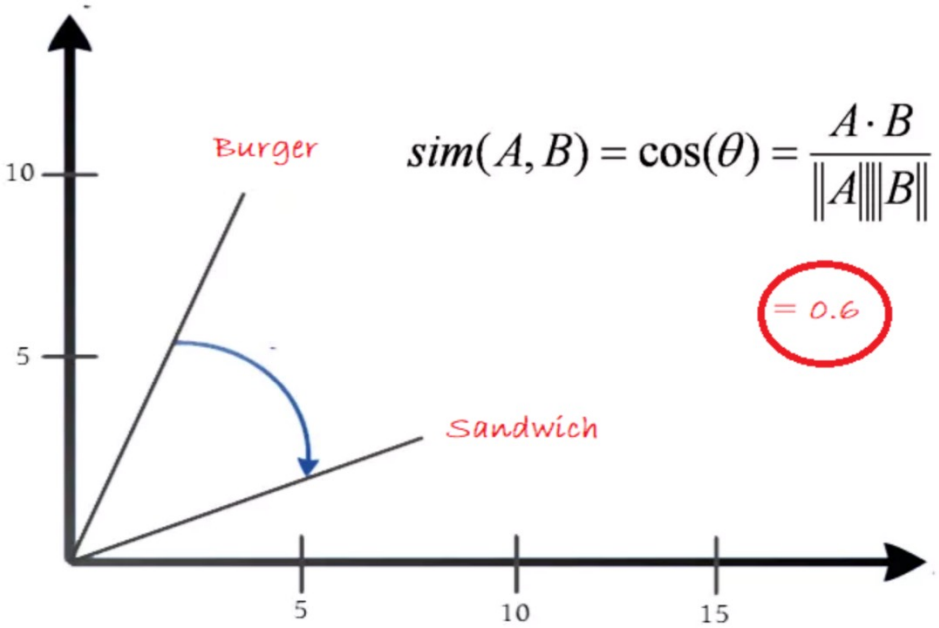
КОСИНУСНОЕ РАССТОЯНИЕ

- Скалярное произведение векторов x и y:

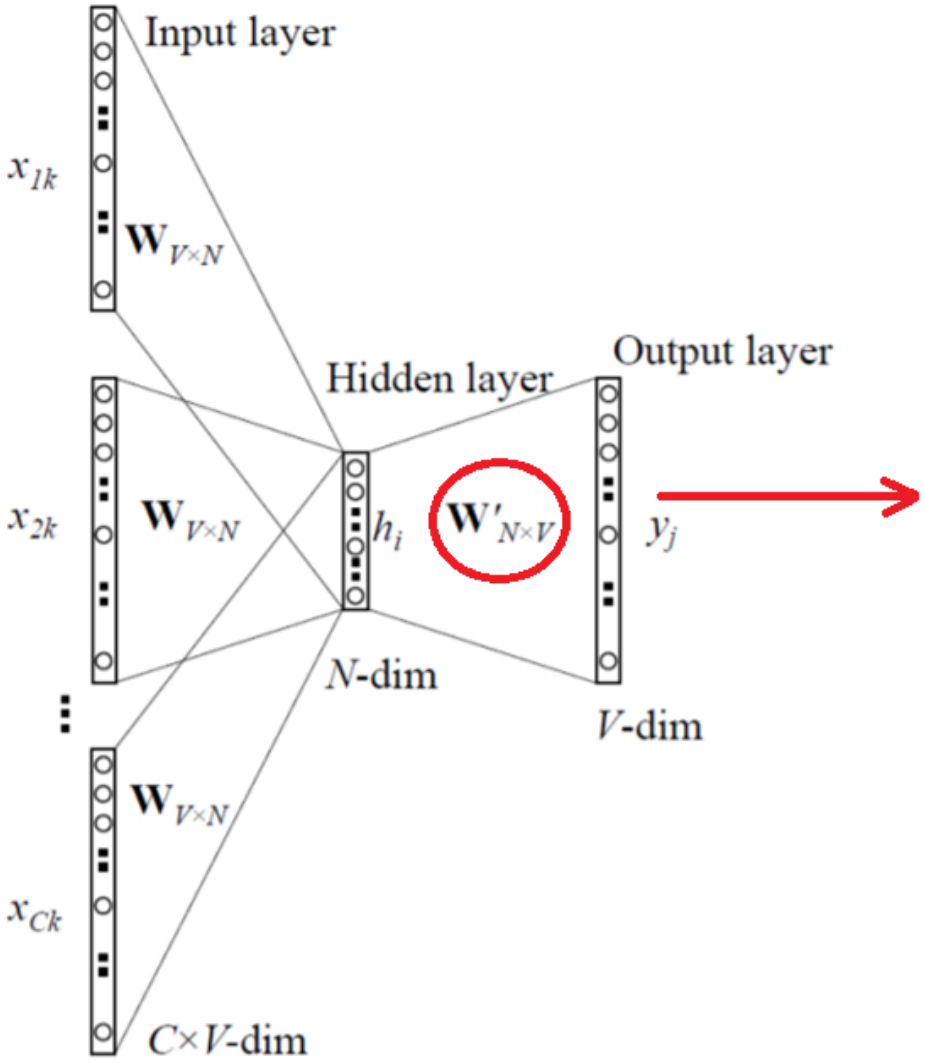
$$(x, y) = ||x|| \cdot ||y|| \cdot \cos(x, y)$$

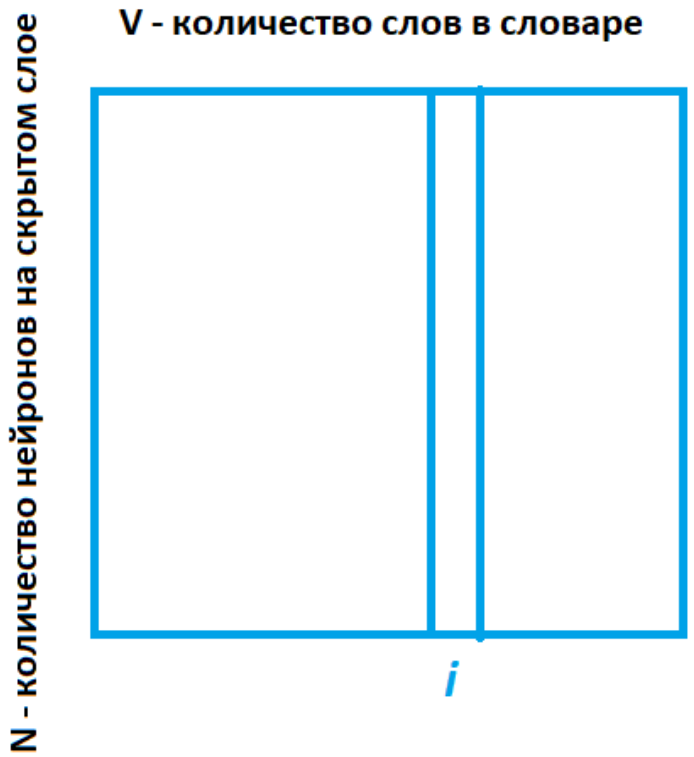
В качестве расстояния между словами используется косинусное расстояние:

$$\rho(w_i, w_j) = \frac{(w_i, w_j)}{||w_i|| \cdot ||w_j||}$$



ВЕКТОРЫ СЛОВ

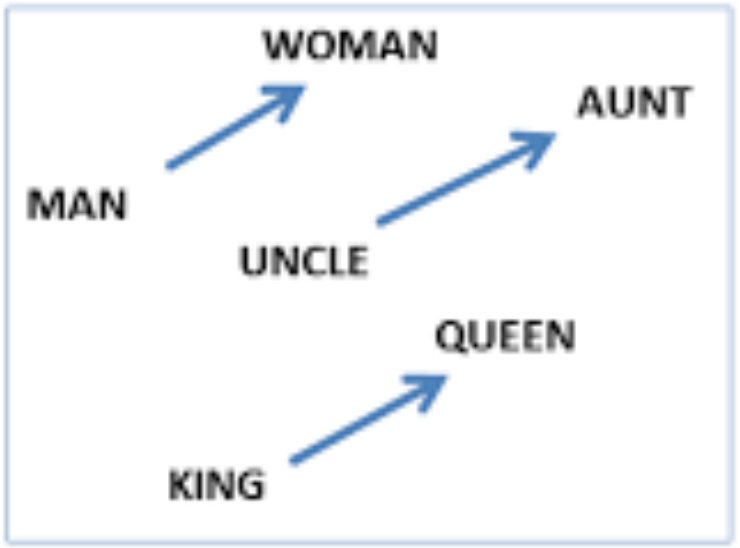


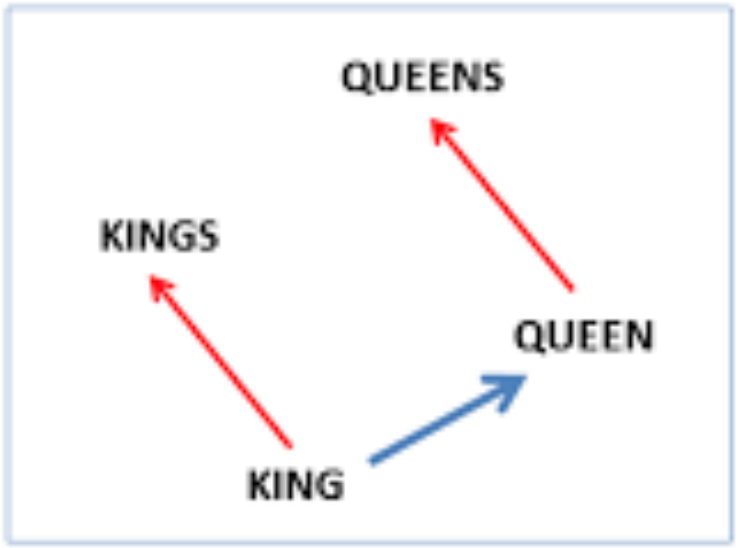


Числа, стоящие в ***i***-м столбце в полученной матрице весов – это word2vec-вектор длины *N*, представляющий *i*-е слово из словаря.

ВЕКТОРНЫЕ СООТНОШЕНИЯ МЕЖДУ СЛОВАМИ

За счёт использования косинусного расстояния между векторами слов, к векторам слов, полученных в результате применения word2vec, можно применять векторные операции сложения и вычитания, которые будут иметь смысл:





NEGATIVE SAMPLING

В архитектурах CBOW и SkipGram есть некоторая вычислительная трудность. Давайте рассматривать SkipGram, но в CBOW дела обстоят аналогично.

Когда мы для входного слова вычисляем вероятности слов быть в контексте данного слова, мы пользуемся формулой:

$$p(w_o|w_i) = \frac{exp^{(v_{w_o}, v_{w_i})}}{\sum_{w=1}^{W} exp^{(v_w, v_{w_i})}}.$$

Здесь

- w_i - входное слово, w_o - слово, для которого мы считаем вероятность быть в контексте слова w_i
- v_{w_o}, v_{w_i} - word2vec-векторы входного и выходного слов

В знаменателе суммирование идет по всем словам словаря в количестве W - их может быть очень много.

Затем после вычисления этой вероятности мы подставляем ее в функцию потерь и считаем по ней градиент.

В этом алгоритме кроется вычислительная сложность, так как в знаменателе стоит сумма по всем словам из словаря, которых может быть очень много.

NEGATIVE SAMPLING

Решением является ***negative sampling.***

Давайте попробуем перейти от задаче многоклассовой классификации (где число классов = числу слов в словаре) к своего рода задаче бинарной классификации!

Для каждого входного слова мы имеем:

- *положительные примеры*: это слова, которые действительно находятся в контексте вокруг этого слова
- достанем из нашего обучающего корпуса *отрицательные примеры*: это случайные контексты. В силу случайности можно сказать, что эти слова не находятся в контексте вокруг входного слова

Модифицируем функцию потерь - теперь она похожа на двухклассовый Log-Loss:

$$Loss = \sum_{(w,c_{true})} log(p(c_{true}|w)) + \sum_{(w,c_{false})} log(1 - p(c_{false}|w)).$$

NEGATIVE SAMPLING

Здесь

- (w, c_{true}) - все пары входное слово - реальный контекст этого слова. Для них мы считаем штраф при помощи первого слагаемого
- (w, c_{false}) - все пары входное слово - случайный (неверный) контекст этого слова. Для них мы считаем штраф при помощи второго слагаемого
- $p(c \mid w)$ - вероятность того, что слово c находится в контексте слова w

На практике мы можем взять $k \in [2, 20]$ случайных отрицательных контекстов. Тем самым мы сильно снизим вычислительную сложность алгоритма - ведь для каждого входного слова ***мы будем обновлять веса не у всех слов из словаря, а только у небольшого фиксированного количества слов***.

Эта процедура при этом сохраняет очень высокое итоговое качество модели.

NEGATIVE SAMPLING

Vanilla Skip-Gram

W_output (old)				Learning R.		grad_W_output				W_output (new)		
-0.560	0.340	0.160				0.064	0.071	-0.014		**-0.563**	**0.336**	**0.161**
-0.910	-0.440	1.560				0.098	0.015	0.063		**-0.915**	**-0.441**	**1.557**
-1.210	-0.130	-1.320				0.069	0.089	0.045		**-1.213**	**-0.134**	**-1.322**
1.670	-0.150	-1.030				0.014	0.085	0.079		**1.669**	**-0.154**	**-1.034**
1.720	-1.460	0.730				-0.021	0.067	0.071		**1.721**	**-1.463**	**0.726**
0.000	1.390	-0.120				-0.098	-0.088	0.091		**0.005**	**1.394**	**-0.125**
-0.060	1.520	-0.790				-0.072	-0.078	-0.089		**-0.056**	**1.524**	**-0.786**
0.800	1.850	-1.670				0.046	-0.079	-0.053		**0.798**	**1.854**	**-1.667**
-1.370	1.320	-0.480				-0.049	-0.087	0.025		**-1.368**	**1.324**	**-0.481**
0.670	1.990	-1.850				-0.060	0.092	0.042		**0.673**	**1.985**	**-1.852**
-1.520	-1.740	-1.860				0.074	0.050	0.070		**-1.524**	**-1.743**	**-1.864**
(11X3)						(11X3)				(11X3)		

Negative Sampling

W_output (old)				Learning R.		grad_W_output				W_output (new)		
-0.560	0.340	0.160				Not computed!				-0.560	0.340	0.160
-0.910	-0.440	1.560								-0.910	-0.440	1.560
-1.210	-0.130	-1.320								-1.210	-0.130	-1.320
1.670	-0.150	-1.030								1.670	-0.150	-1.030
1.720	-1.460	0.730								1.720	-1.460	0.730
0.000	1.390	-0.120								0.000	1.390	-0.120
-0.060	1.520	-0.790								-0.060	1.520	-0.790
0.800	1.850	-1.670								**0.798**	**1.849**	**-1.672**
-1.370	1.320	-0.480								**-1.366**	**1.318**	**-0.477**
0.670	1.990	-1.850								**0.667**	**1.985**	**-1.847**
-1.520	-1.740	-1.860								**-1.523**	**-1.744**	**-1.858**
(11X3)						(11X3)				(11X3)		

Positive sample, w_o
Negative sample, k=1
Negative sample, k=2
Negative sample, k=3

ДРУГИЕ ВАРИАНТЫ

- FastText https://amitness.com/2020/06/fasttext-embeddings/
- GLoVe https://nlp.stanford.edu/projects/glove/

3. litoria

4. leptodactylidae

5. rana

7. eleutherodactylus

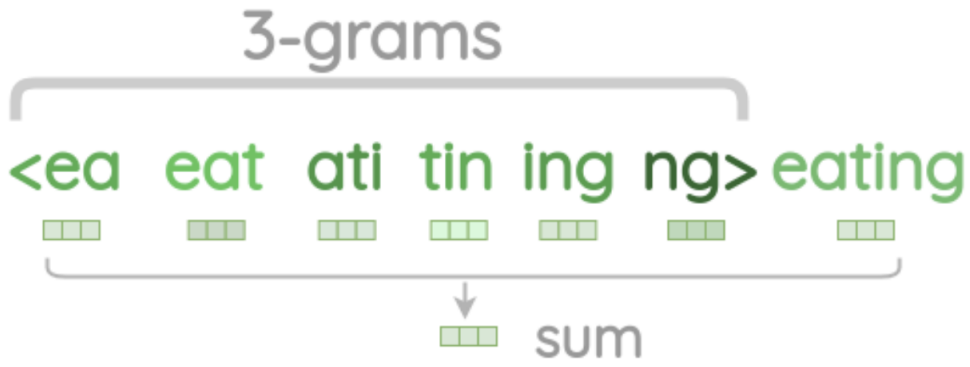
FASTTEXT

У алгоритма word2vec есть недостаток: если в новых текстах встречаются слова, отсутствующие в обучающей выборке, их не получится закодировать при помощи word2vec.

FastText - это модификация word2vec, которая решает эту проблему при помощи использования символьных N-грамм. То есть в качестве токенов используются не только слова, но и их кусочки, N-граммы.

- Например, 3-граммы слова кошка: *кош, ошк, шка.* В этом случае fasttext будет обучаться на следующих токенах: *ко, кош, ошк, шка, ка, кошка* (Токены ко и ка - на самом деле трехсимвольные: START_TOKEN + к + о и к + а + END_TOKEN, где START_TOKEN и END_TOKEN - специальные символы начала и конца слова).
- Чтобы посчитать вектор слова, мы суммируем векторы всех его N-грамм.

Поэтому, если встретится новое слово, то мы все равно сможем его векторизовать, так как оно состоит из N-грамм, которые с большой вероятностью присутствуют в обучении.



GLOVE

Еще один недостаток word2vec - учет только локального контекста, то есть модель берет в расчет только ближайших соседей слова для его векторизации, при этом игнорируя глобальную структуру корпуса текста. В результате модель может упустить некоторые важные семантические отношения между словами.

Модель GloVe решает эту проблему, объединяя как локальную, так и глобальную статистику появления слов в корпусе текста. Она использует матрицу совместной встречаемости слов, чтобы учесть глобальные закономерности в данных. Такой подход позволяет учитывать не только ближайшие слова, но и все слова в корпусе текста при вычислении векторных представлений.

Разберемся чуть подробнее.

- Первый шаг в GloVe - это создание матрицы совместной встречаемости X. Эта матрица содержит статистику того, как часто слова i и j появляются вместе в корпусе текста. Обычно для этого используется окно контекста - это количество слов слева и справа от данного слова, которое учитывается.
- Для каждой пары слов i и j значение X_{ij} определяется как частота, с которой слово j появляется в контексте слова i. Матрица совместной встречаемости может быть оценена с использованием эмпирических данных из корпуса текста.

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

GLOVE

- Далее мы вводим некоторую функцию $f(X_{ij})$, чтобы определить, насколько важна каждая пара слов для модели. Эта функция обычно определяется некоторыми эмпирическими формулами (далее опустим ее для простоты)
- Наконец, при обучении модели мы минимизируем следующую функцию:

$$J = \sum_{i,j=1}^{V} ((w_i, w_j) - log(X_{ij}))^2,$$

то есть обучаем модель таким векторам, чтобы скалярное произведение слов i и j было близко к логарифму совместной встречаемости слов (суммирование в формуле идет по всем парам слов из словаря размера V).