



NLP

Bag of Words, TF-IDF, Word2Vec

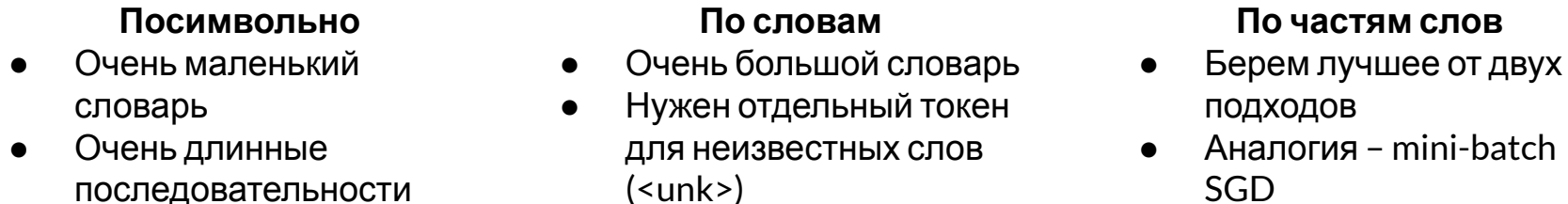


Мария Макарова
Senior Data Analyst, Playrix

Базовые понятия

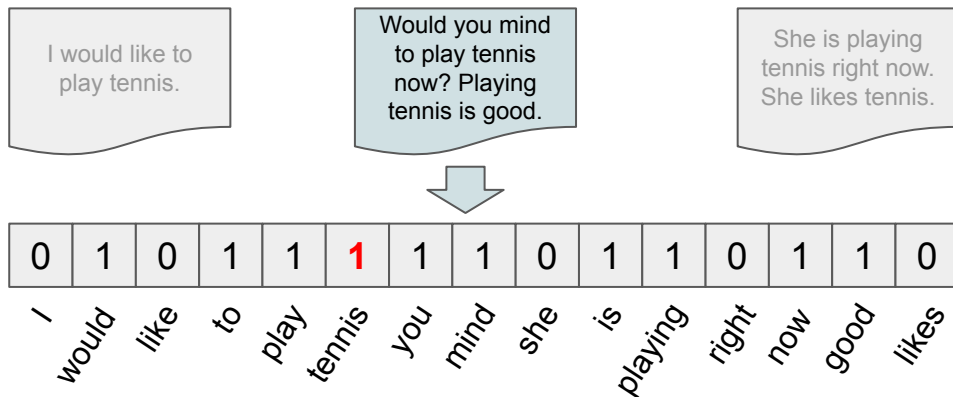
- Токен – последовательность символов (слово, слог, словосочетание и т д)
- Текст (документ) – последовательность токенов
- Корпус – набор текстов
- Токенизация – представление текста в виде последовательности

Токенизация



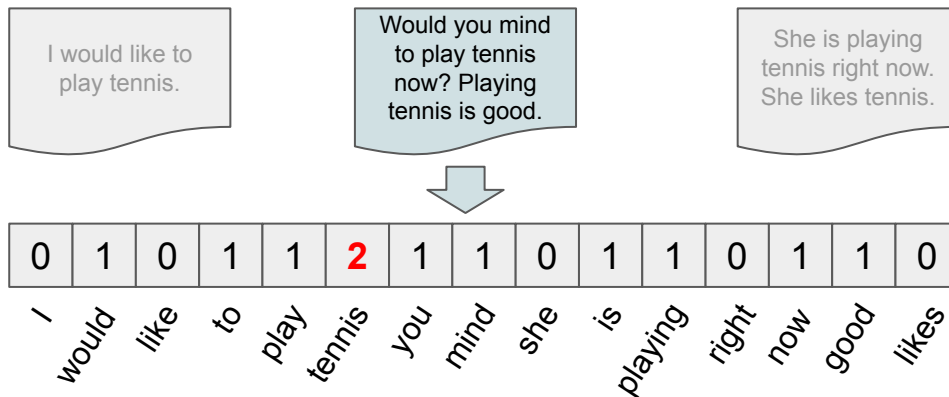
One-Hot Encoding

- Составляем словарь из всех слов в корпусе в том виде, в котором они присутствуют в текстах
- Получаем набор из N слов (размер словаря)
- Каждый текст = вектор длины N , где каждый элемент отвечает за **факт присутствия** (1 или 0) того или иного слова в тексте




Bag of Words

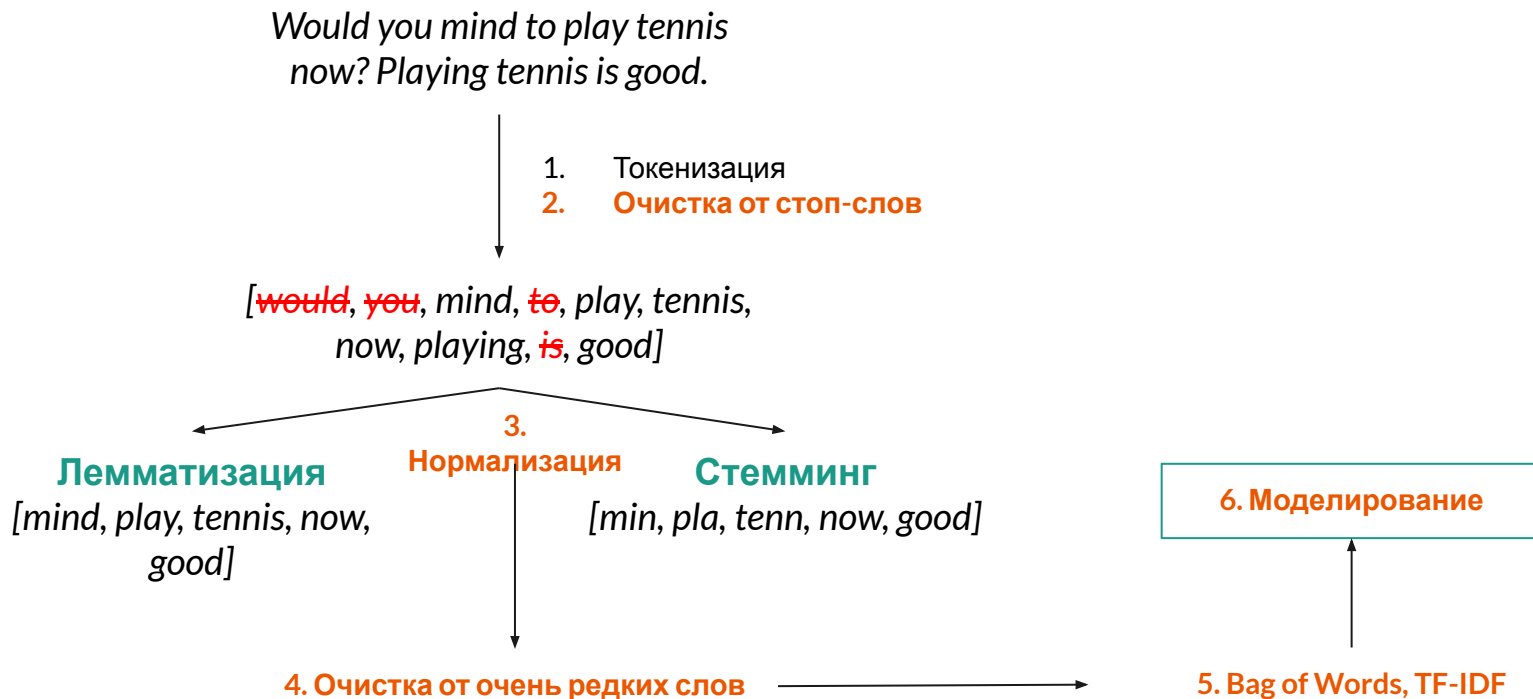
- Составляем словарь из всех слов в корпусе в том виде, в котором они присутствуют в текстах
- Получаем набор из N слов (размер словаря)
- Каждый текст = вектор длины N , где каждый элемент отвечает за **количество присутствий** того или иного слова в тексте



Недостатки ONE и Bag of Words

- 
- Очень большой словарь, а значит, очень много чисел для одного текста (представьте себе художественное произведение)
 - Не учитываются похожие по смыслу слова – “image” и “picture”
 - Очень похожие по смыслу тексты могут иметь сильно разное представление (одно и то же говорим разными словами)
 - Не учитываются порядок и форма слов
 - Слова с опечатками считаются разными словами

Предобработка текста



TF-IDF

- TF (Term Frequency) – как часто встречается слово *в рамках одного конкретного текста*.
- Если слово в *тексте* встречается часто (и оно не стоп-слово), то оно важное.
- Пример: набор документов, каждый из которых описывает правила одного вида спорта
 - В документе про теннис частыми будут слова “мяч”, “ракетка” и т д
 - В документе про беговые лыжи часто встретятся слова “лыжи”, “палки” и т д
- **Как будем считать частоту?**
 - Просто частота слова в тексте (сколько раз встретилось / количество слов в тексте)
 - Булева частота (по сути, One-Hot Encoding)
 - $\text{Log}(\text{частота})$
 - Любая другая характеристика частоты

TF-IDF

- **IDF (Inverse Document Frequency)** – как часто встречается слово в *других текстах*.
- Если слово встречается часто во *всех документах*, то оно не такое важное.
- Пример: набор документов, каждый из которых рассказывает про теннис в разных странах
 - Слово “теннис” будет встречаться в каждом документе – нам это ни о чем не говорит
 - Зато названия стран в каждом документе будут свои
- **Как будем считать частоту?**
 - $\text{Log}(N / n_t)$
 - N – общее количество текстов в корпусе
 - n_t – количество текстов, в которых встретилось конкретное слово t
 - $\text{Log}(1 + N / n_t)$
 - Любой другой способ

Итого: для каждого слова перемножаем $\text{tf} * \text{idf}$ -> получаем матрицу объекты-признаки

Дистрибутивная гипотеза -> Word2Vec

- ONE и TF-IDF дают очень длинные и разреженные векторы
- Хотим добиться более коротких и более емких с точки зрения смысла векторов

Идея: слова со схожим смыслом будут встречаться в схожих контекстах

- Давайте в векторное представление слова закладывать информацию о его контексте
- **Word2Vec**
 - Придумал Томаш Миколов в 2013 году
 - Хотим представлять каждое слово в виде вектора размера
 - Чтобы обучить модель, накладываем условия на вектора

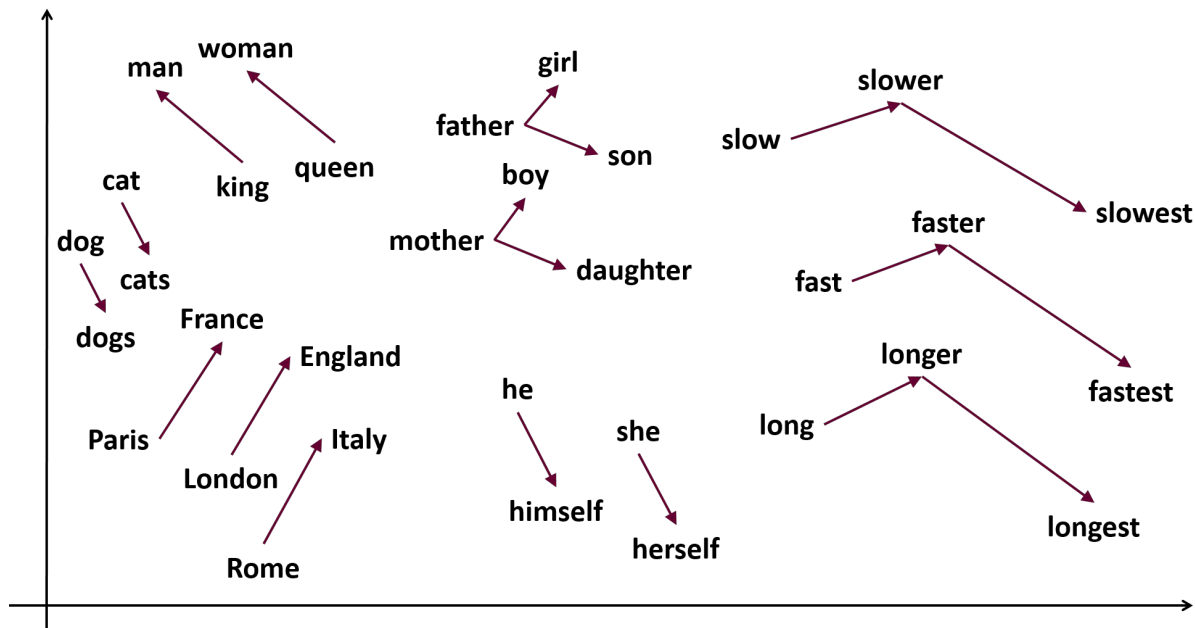


Статья: <https://arxiv.org/pdf/1301.3781.pdf>

Хороший курс: https://lena-voita.github.io/nlp_course/word_embeddings.html

Word2Vec – условие 1

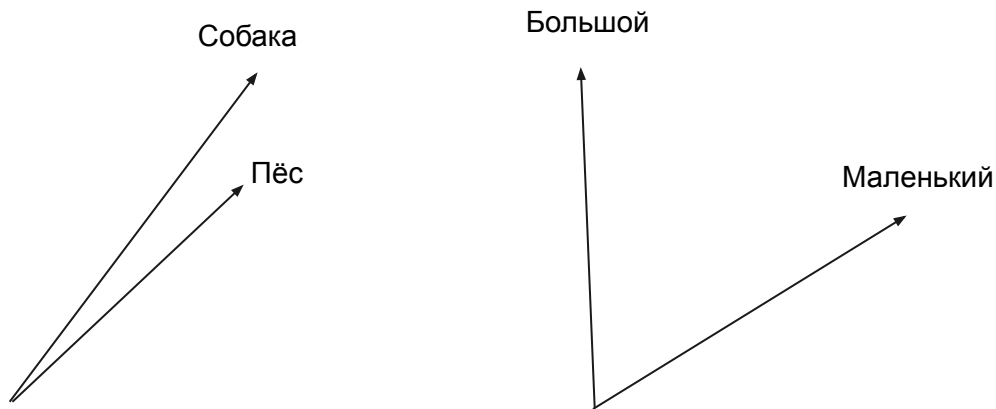
- Модель улавливает семантику слов



Word2Vec – условие 2



- Близкие по смыслу слова имеют близкие векторы (близость считаем по косинусному расстоянию = косинус угла = скалярное произведение / произведение норм векторов)



Word2Vec – условие 3

- Объяснимая арифметика векторов



Королева

-



Женщина

+



Мужчина

=



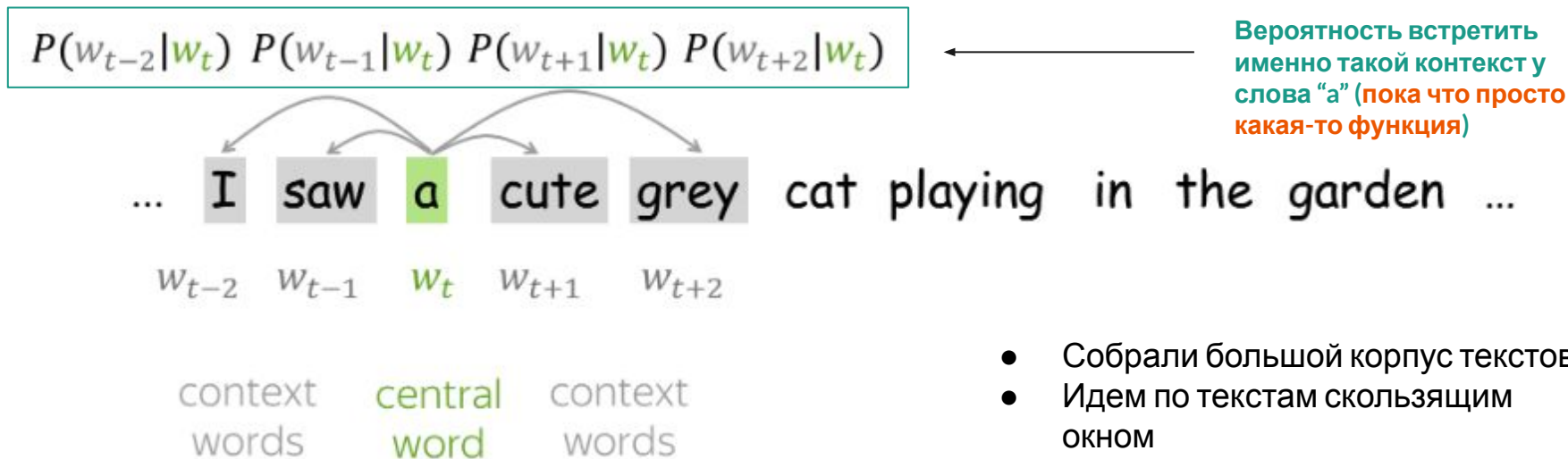
Король

Word2Vec. Основная идея



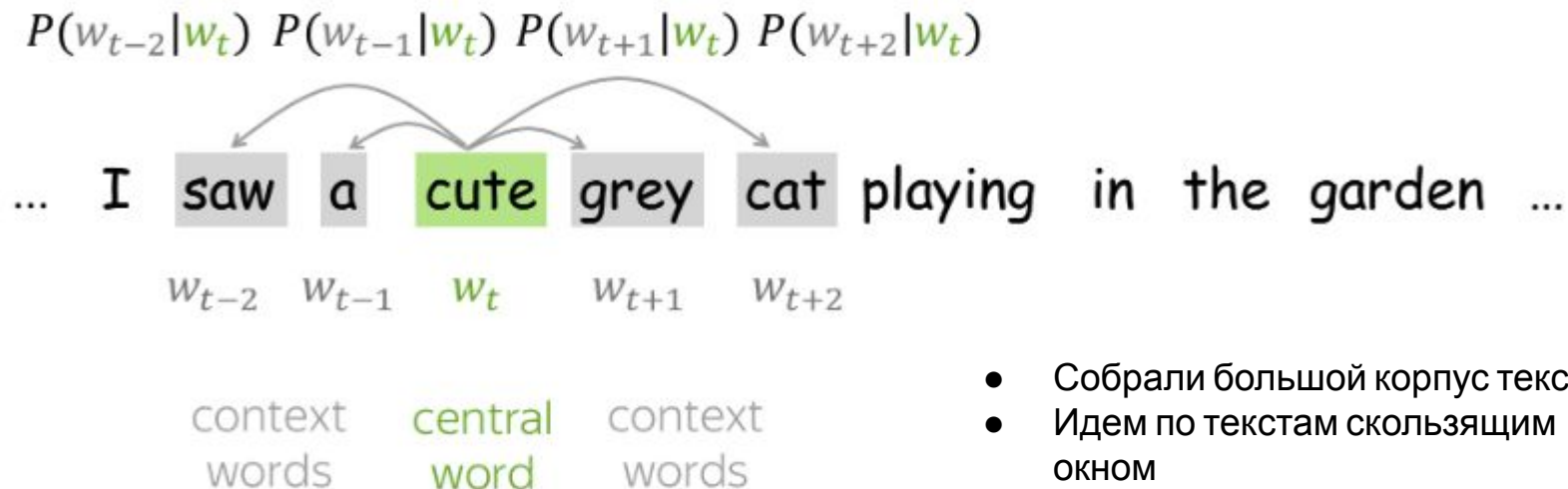
- **Основная идея:** представить информацию о контексте слов в виде векторов
- **Метод:** обучаем вектора, пытаюсь предсказать контекст, в котором встречается слово (skip-gram)
 - **CBOW** – по заданному контексту предсказываем слово
 - **Skip-gram** – по заданному слову предсказываем контекст

Word2Vec. Алгоритм



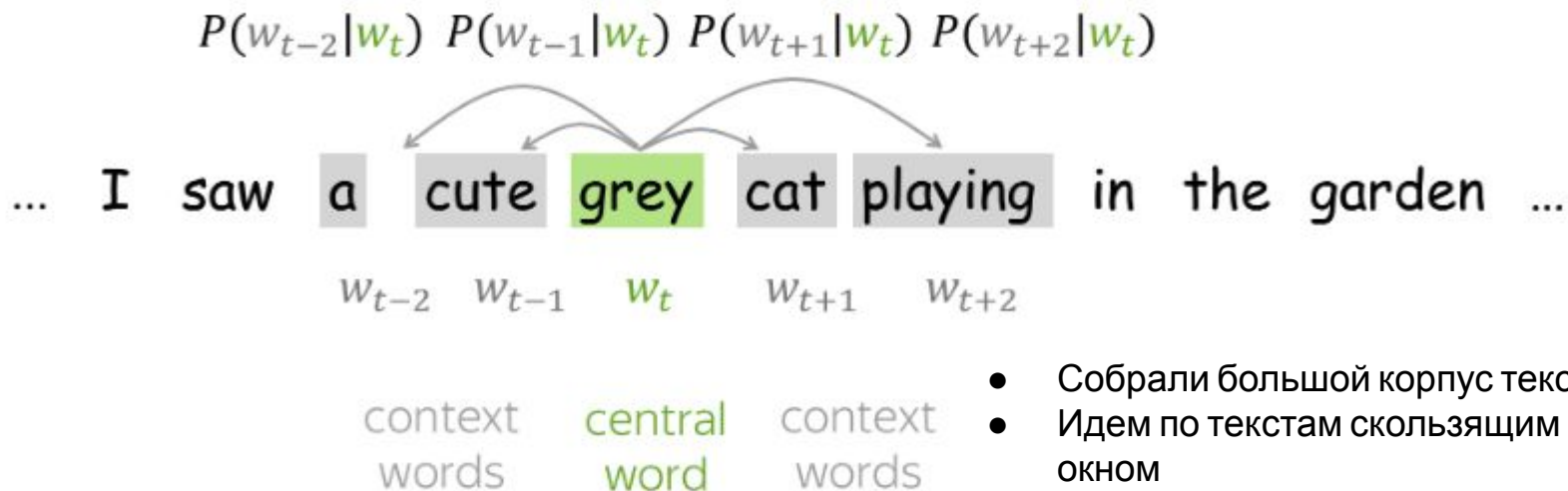
- Собрали большой корпус текстов
- Идем по текстам скользящим окном
- Считаем вероятность встретить контекст при фиксированном центральном слове
- Вектора слов w должны максимизировать вероятность

Word2Vec. Алгоритм



- Собрали большой корпус текстов
- Идем по текстам скользящим окном
- Считаем вероятность встретить контекст при фиксированном центральном слове
- Вектора слов w должны максимизировать вероятность

Word2Vec. Алгоритм



- Собрали большой корпус текстов
- Идем по текстам скользящим окном
- Считаем вероятность встретить контекст при фиксированном центральном слове
- Вектора слов w должны максимизировать вероятность

Функция потерь

- Выписываем **правдоподобие** (перемножаем все вероятности) и **максимизируем** его

Правдоподобие – вероятность получить именно ту ситуацию, которую мы имеем по факту

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

Пробегаемся по всем текстам

Пробегаемся по всем словам в каждом тексте

- Получаем функцию потерь – логарифмируем (чтобы получилась сумма) и умножаем на -1 и делим на количество текстов T

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

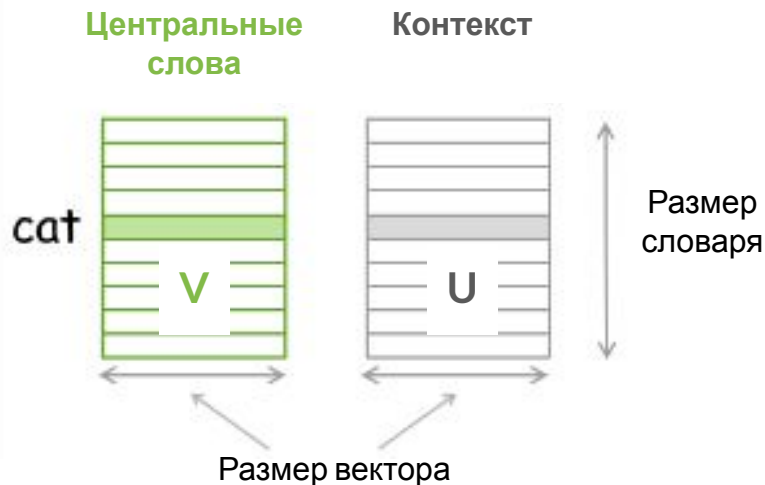
Параметры – сами обучаемые вектора

- Как считать вероятности? Как оптимизировать? Параметров много...

Давайте договоримся...

Как считать вероятности? Для каждого слова будем считать два вектора

- Когда слово является центральным, считаем вектор v
- Когда слово является контекстом, считаем вектор u
- После обучения обычно используют только вектора слов V



Количество столбцов в каждой матрице – гиперпараметр

Обучаем два вектора для каждого слова в корпусе текстов

Считаем вероятности

Как считать вероятности?

- Для центрального слова c и контекстного слова o

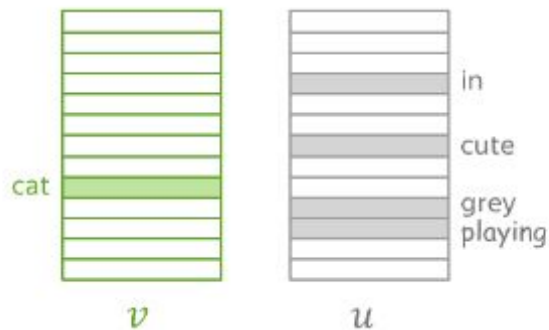
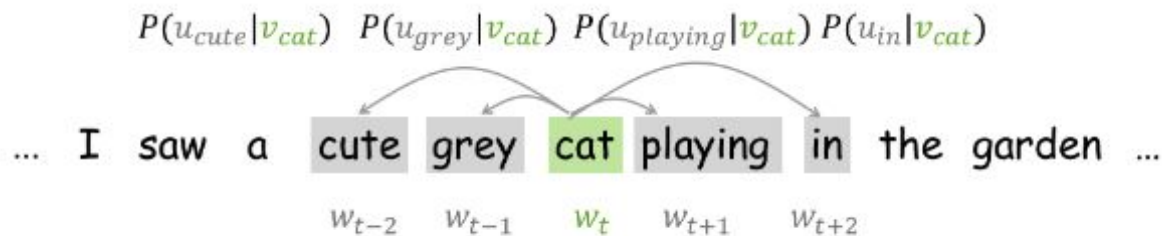
$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

Скалярное произведение: показывает близость векторов o и c
Больше скалярное произведение = больше вероятность

Нормируем по всему словарю, чтобы получить распределение вероятности

- Обычный softmax
- Хотим, чтобы вероятности для пар слов, встречающихся в одном контексте, были высокими (наоборот тоже хотим)

Скоро начнем обучать



Когда мы считаем вероятности, для центрального слова **cat** берем вектор из матрицы **V**, а для контекстных слов (cute, grey, playing, in) – вектора из матрицы **U**

Обучаем Word2Vec!

- Общая функция потерь – пытаемся максимизировать правдоподобие

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

... I saw a cute grey cat playing in the garden ...

- Значение функции потерь для одной пары слов (центральное **cat**, контекстное cute) в конкретном окне t

$$J_{t,j}(\theta) = -\log P(\text{cute} | \text{cat}) = -\log \frac{\exp u_{\text{cute}}^T v_{\text{cat}}}{\sum_{w \in \text{Voc}} \exp u_w^T v_{\text{cat}}} = -u_{\text{cute}}^T v_{\text{cat}} + \log \sum_{w \in \text{Voc}} \exp u_w^T v_{\text{cat}}$$

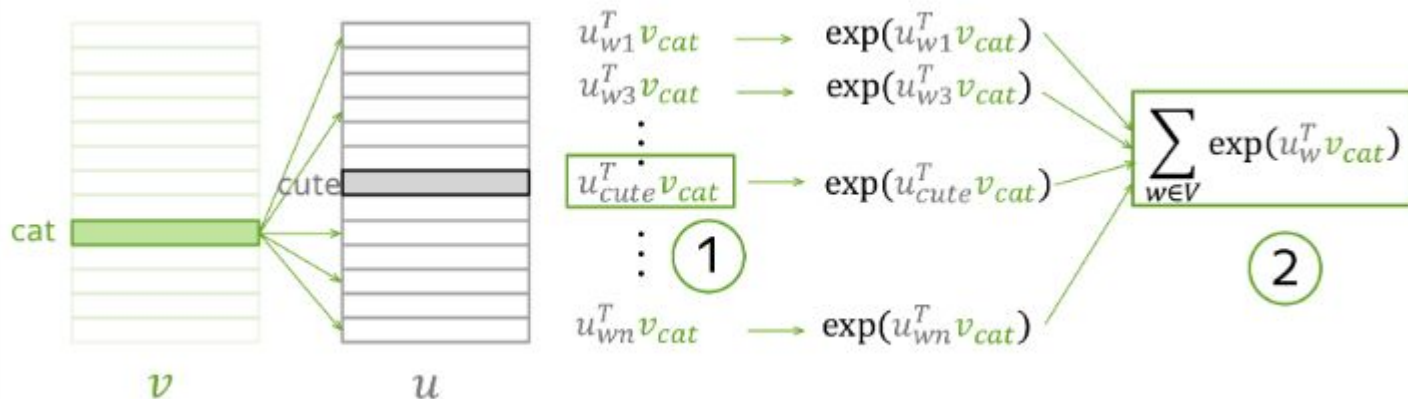
- Спойлер!** на каждом шаге мы считаем J только для пар слов из текущего окна t , но обновлять будем больше векторов, потому что во втором слагаемом присутствуют *контекстные вектора всех слов*

Обучаем Word2Vec!

1. Считаем скалярное произведение вектора \mathbf{V} центрального слова с контекстными векторами \mathbf{U} всех слов из словаря (то есть *по всем текстам в корпусе*)
2. От каждого скалярного произведения берем экспоненту
3. Считаем сумму (получили второе слагаемое в J)

Шпаргалка

$$J_{t,j}(\theta) = -u_{cute}^T v_{cat} + \log \sum_{w \in Voc} \exp u_w^T v_{cat}$$



Word2Vec

4. Считаем потери для данного окна (если окно включает 2 слова слева и 2 слова справа, то получаем 4

шага в рамках одного окна)

5. Делаем шаг градиентного спуска для вектора \mathbf{V} центрального слова и для векторов \mathbf{U} контекстных

вект 4

$$J_{t,j}(\theta) = -\underbrace{u_{cute}^T v_{cat}}_{\text{green box}} + \log \underbrace{\sum_{w \in V} \exp(u_w^T v_{cat})}_{\text{green box}}$$

5

$$v_{cat} := v_{cat} - \alpha \frac{\partial J_{t,j}(\theta)}{\partial v_{cat}}$$

$$u_w := u_w - \alpha \frac{\partial J_{t,j}(\theta)}{\partial u_w} \quad \forall w \in V$$

Обращаем внимание на знаки слагаемых

Мы минимизируем -> первое слагаемое максимизируем по модулю, второе слагаемое минимизируем по модулю.

Увеличиваем важность близкого контекста, уменьшаем важность дальнего контекста.

Word2Vec. Обобщаем

- Хотим представить каждое слово в виде вектора, например, размерности 300 (это гиперпараметр)
- Накладываем условия на эти вектора – арифметика, близость векторов слов со схожим контекстом
- Для удобства обучения для каждого слова обучаем дополнительный вектор (когда слово является контекстом другого слова)
- Основная задача – максимизировать правдоподобие (вероятность получить именно такой корпус текстов, который есть у нас по факту)
- Функция потерь = минус логарифм правдоподобия
- Чтобы посчитать вероятности для правдоподобия, используем softmax
- Берем весь корпус текстов и проходимся по нему скользящим окном (например, центральное слово, 2 слова слева и 2 слова справа)
- На каждом шаге (одно окно) делаем шаг градиентного спуска для основного вектора центрального слова и для контекстных векторов всех слов. По факту, для окна $2+1+2$ у нас будем 4 шага на окно.

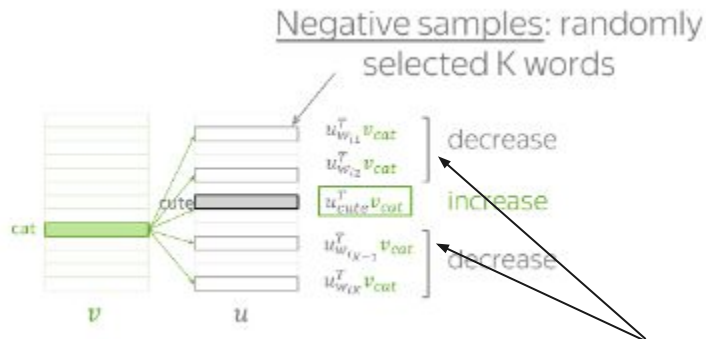
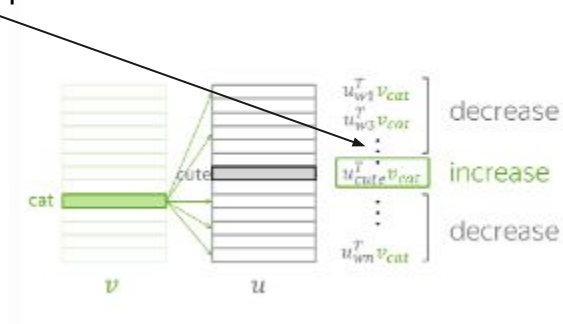
Word2Vec. Negative Sampling

- На каждом шаге (одно окно) обновляем контекстные вектора *всех слов* из словаря – много...
- Давайте на каждом шаге будем выбирать случайным образом несколько слов из дальнего контекста (аналогия – снова вспоминаем mini-batch SGD)
- Дальше делаем все то же самое, что и до этого

ДО

ПОСЛЕ

Точки – берем все слова



Обозримое количество
случайно выбранных слов

Word2Vec. CBOW

- Skip-gram
 - Мы опирались на вероятности появления контекстных слов возле какого-то центрального слова
 - $2 + 1 + 2$
- CBOW
 - Поступаем симметрично
 - Опираемся на вероятность появления центрального слова при заданных контекстных словах
 - $2 + 1 + 2$
 - Дальше логика такая же, как и была
- Skip-gram и CBOW одинаковы по эффективности
- Какое-то время считалось, что CBOW работал хуже, но оказалось, что это была ошибка в реализации в библиотеке gensim

Word2Vec. Библиотеки

- gensim – самая популярная и эффективная библиотека для тематического моделирования
 - Тематическое моделирование — способ построения модели коллекции текстовых документов, которая определяет, к каким темам относится каждый из документов.
- spacy – тоже есть Word2Vec
- nltk – библиотека для предобработки текста (исключение стоп-слов, лемматизация, стемминг и т д)
- re – библиотека для работы с регулярными выражениями



Спасибо за внимание!