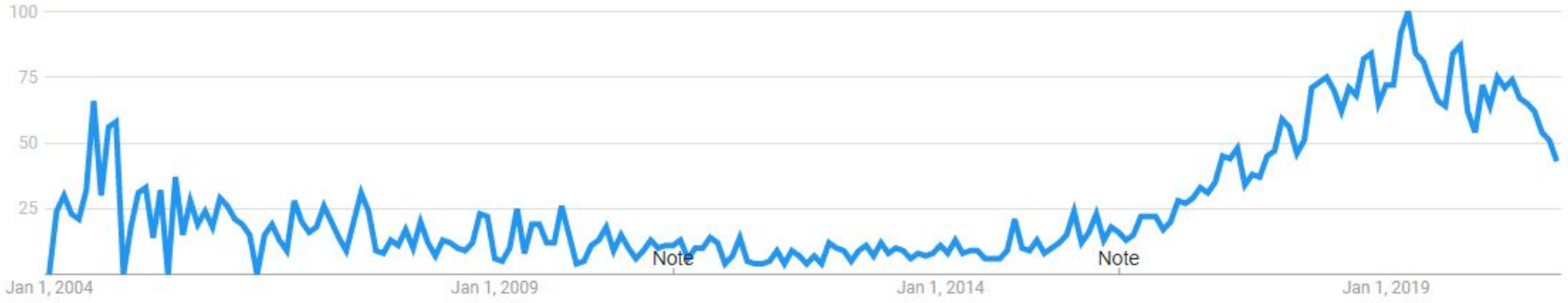
Эмбеддинги в NLP

Григорьев Петр, Карташев Николай

Мотивация

- Хотим хранить слова в памяти компьютера
- Хранить как строки не имеет смысла и затратно
- Хранить как номера слов в словаре очень неинформативно
- Вывод: хранить в качестве векторов
- Многие алгоритмы заточены под работу с векторами и матрицами



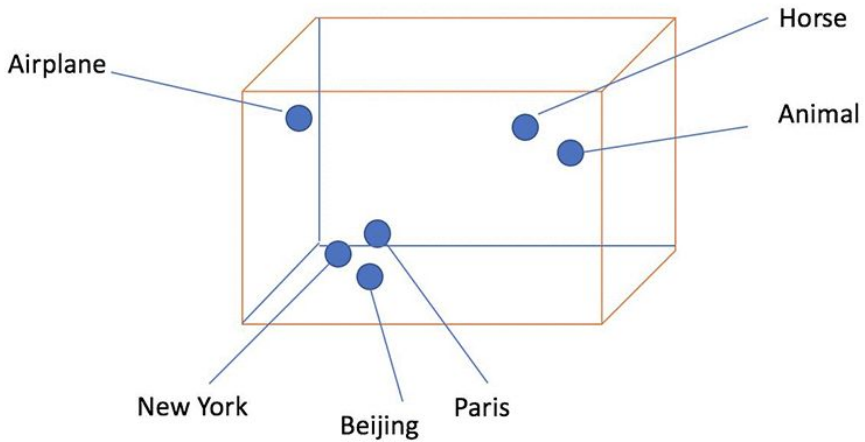
Векторные представления

Embedding или **векторное представление** - сопоставление объекту числового вектора в многомерном пространстве

Мы можем строить эмбеддинги для любых объектов:

- Слов
- Символов
- Документов
- И т.д.

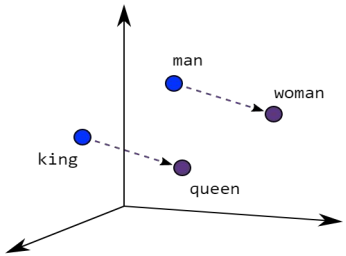
Сейчас нас интересуют **word embeddings**



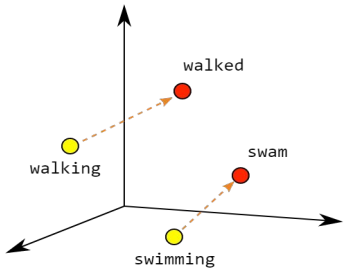
Требования к эмбеддингам

Перед тем как обсуждать способы построения векторного представления, озвучим интуитивные требования, который к ним возникают

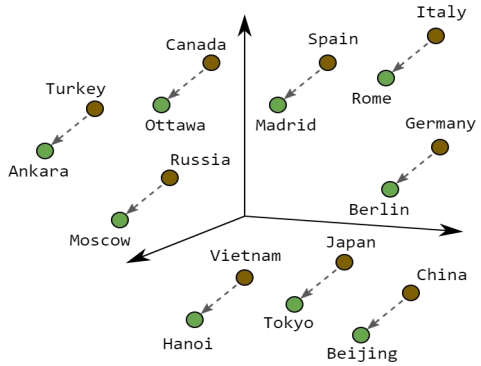
- Относительная компактность и удобство работы
- Близость векторов отражает близость исходных слов
- В идеале: выразить семантические отношения через векторную арифметику:



Male-Female



Verb Tense



Country-Capital

Почему не тезаурус?

1. Нет тезаурусов для всех языков
2. Семантика слов быстро меняется, тезаурус не сможет подстроиться
3. Плохо работает для глаголов, наречий

Эмбеддинг через One-hot

Самая простая идея: каждому слову сопоставить вектор с одной единицей, соответствующей позиции слова в словаре

Недостатки:

1. Огромный расход памяти
2. Не несет информации о словах
3. Все вектора образуют ортогональный базис пространства

	1	2	3	4	5	6	7	8	9
man	1	0	0	0	0	0	0	0	0
woman	0	1	0	0	0	0	0	0	0
boy	0	0	1	0	0	0	0	0	0
girl	0	0	0	1	0	0	0	0	0
prince	0	0	0	0	1	0	0	0	0
princess	0	0	0	0	0	1	0	0	0
queen	0	0	0	0	0	0	1	0	0
king	0	0	0	0	0	0	0	1	0
monarch	0	0	0	0	0	0	0	0	1

Эмбеддинг через SVD

Co-Occurrence Matrix - хранит совместную встречаемость для каждой пары слов. Она может вычисляться как:

1. Число попаданий пары слов в одно контекстное окно
2. Число документов в корпусе, где встречаются оба слова
3. И т.д.

Руководствуемся идеей, что *схожие слова часто встречаются в одних блоках с одними и теме же словами*

	I	love	IRS	ML	hate	Phy sics	.
I	0	2	0	0	1	0	2
love	2	0	1	1	0	0	0
IRS	0	1	0	0	0	0	1
ML	0	1	0	0	0	0	1
hate	1	0	0	0	0	1	0
Phy sics	0	0	0	0	1	0	1
.	2	0	1	1	0	1	0

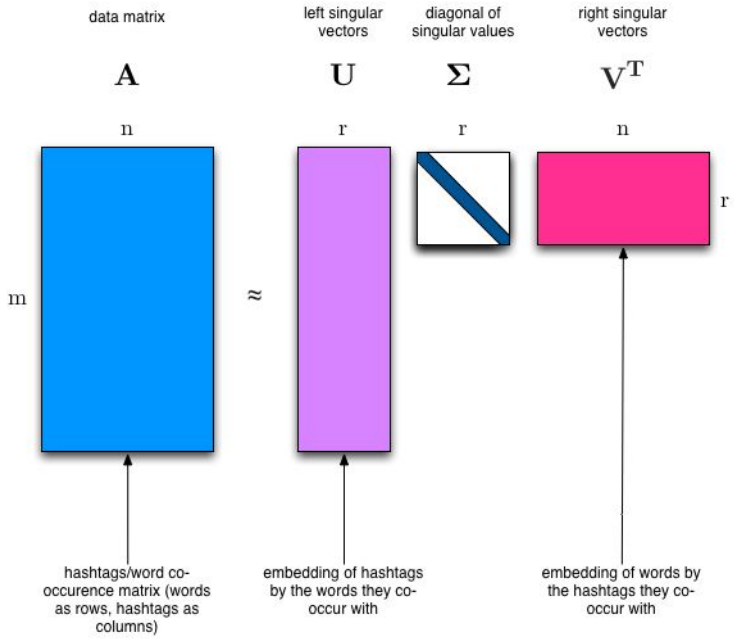
Эмбеддинг через SVD

Теперь можем сделать сингулярное разложение этой матрицы $X_{|L|\times|L|} = U\Sigma V^T$

Берем только первые **r** компонент, получаем r-мерное векторное представление

Умножив векторы (и поделив на произведение их длин), получаем значение, характеризующее как часто они встречаются вместе

Вектора близких слов получаются относительно близкими друг другу



Эмбеддинг через SVD

Преимущества метода:

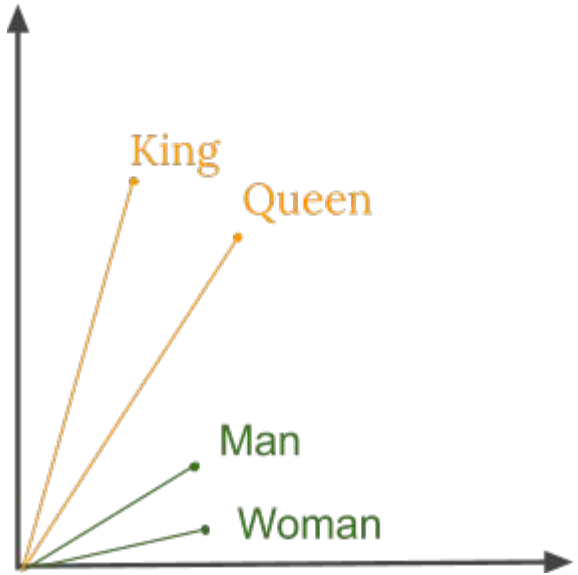
1. Сохраняются определенные семантические связи
2. SVD - хорошо изученная матричная операция, есть эффективные алгоритмы
3. Для улучшения результата можем использовать другие метрики сходства слов (напр. *Pointwise Mutual Information*)

Недостатки метода:

1. Матрица X имеет большую размерность и разреженная - трудно хранить
2. Простая частота совместной встречаемости не очень информативна (есть искажения из-за разной частоты слов)
3. Трудно добавить новое слово после сингулярного разложения

Word2Vec

- Будем руководствоваться новой идеей: схожие слова встречаются в схожих контекстах
- *“A word is characterized by the company it keeps”* J. R. Firth
- Использует один из двух алгоритмов обучения: CBOW или Skip-Gram
- В результате их работы мы получим веса на скрытых слоях, из которых можем выделить эмбеддинги

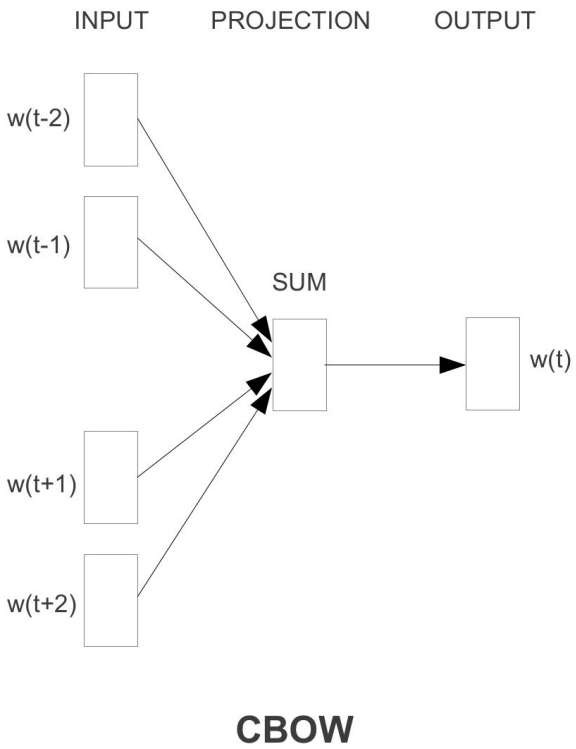


CBOW

Continuous bag of words - архитектура, предсказывающая слово по контексту. Простейшая аналогия - автодополнения в клавиатуре

Сканируем большой объем текстов и создаем датасет, где признаки - предыдущие и последующие K слов

$$P(w_t|c) = \frac{e^{s(w_t|c)}}{\sum_{w_i \in W} e^{s(w_i|c)}}$$

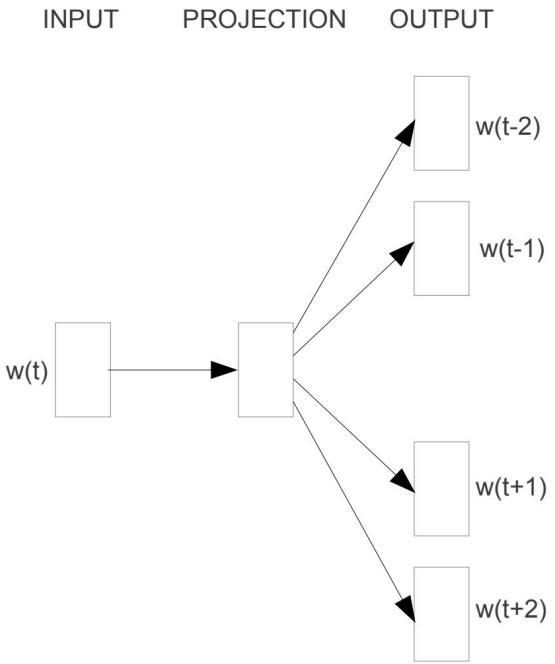


Skip-Gram

Пойдем обратным способом - будем пытаться угадать контекст по одному слову

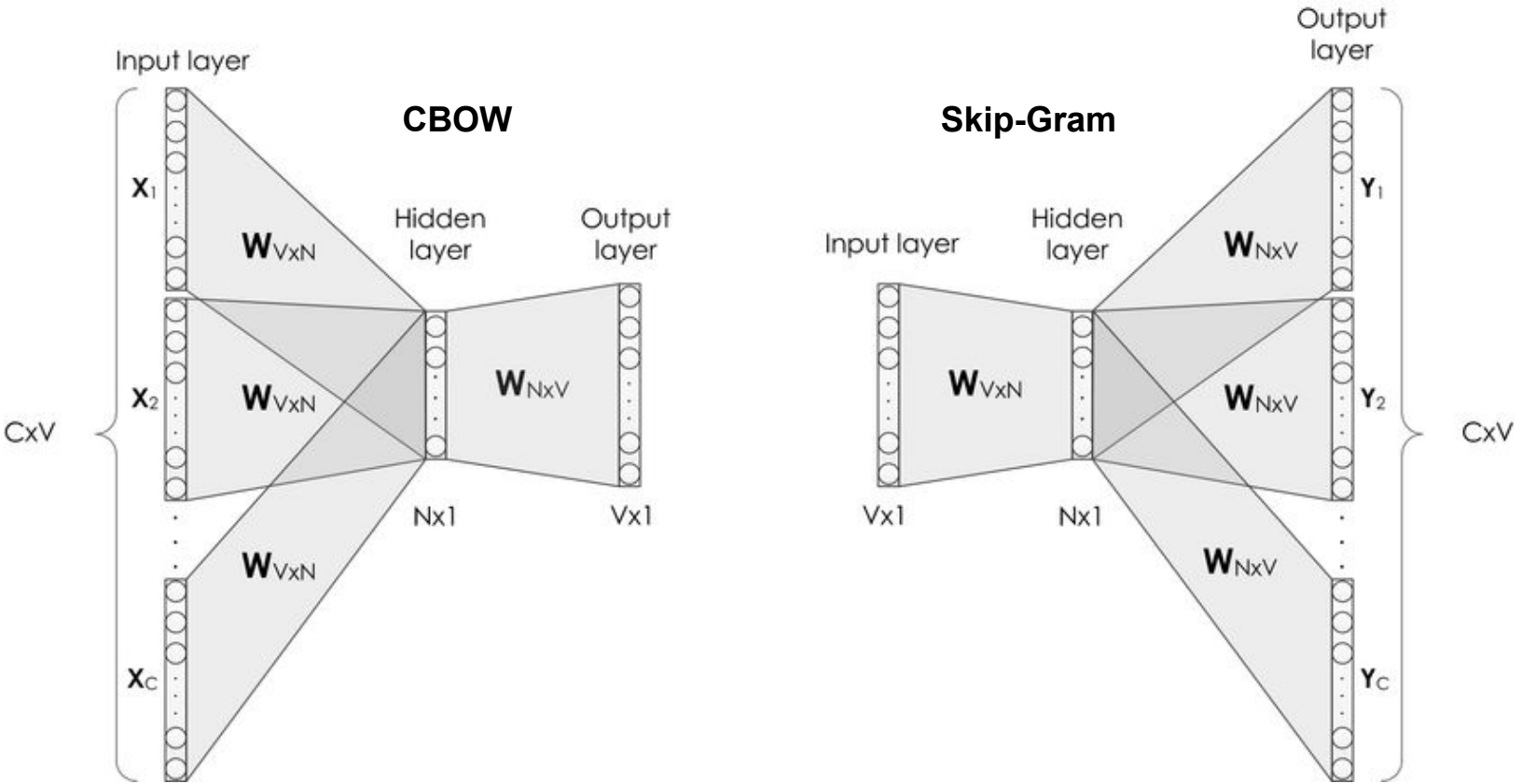
$$P(w_t|w_c) = \frac{e^{s(w_t|w_c)}}{\sum_{w_i \in W} e^{s(w_i|w_c)}}$$

$$s(w_t|w_c) = e^{v_t^T v_c}$$



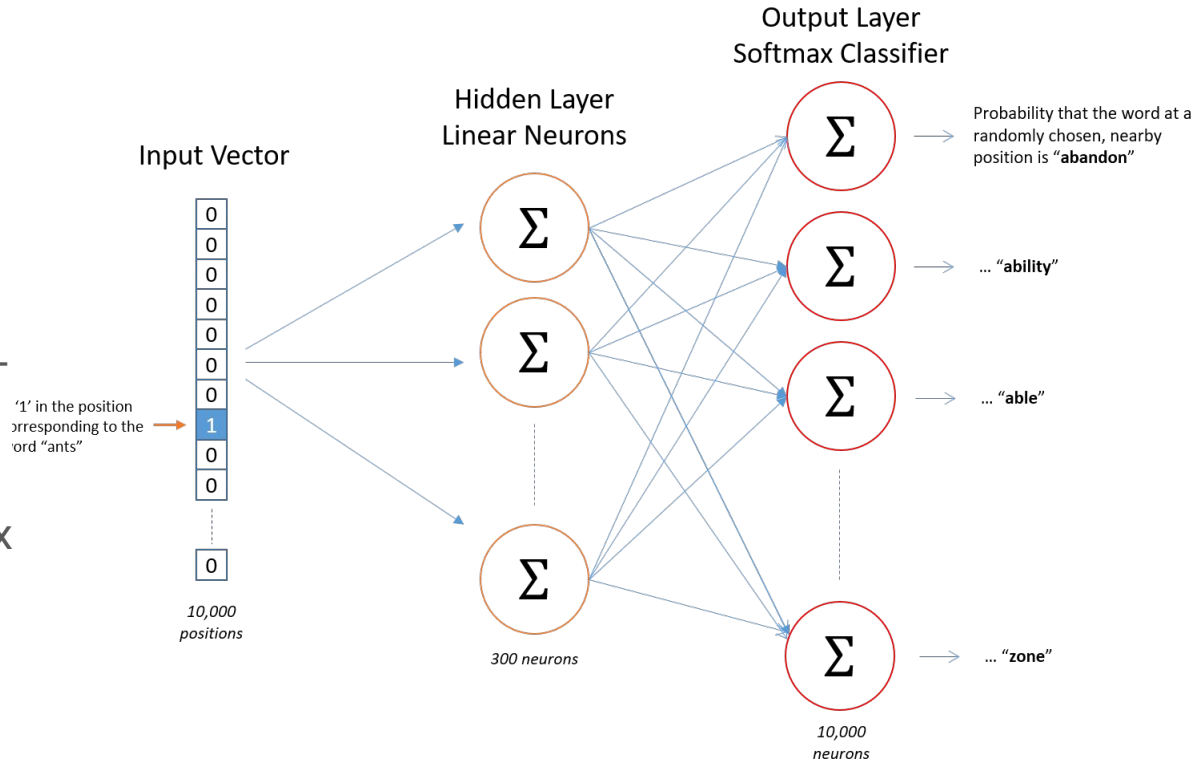
Skip-gram

Архитектура



Negative Sampling

- Однако возникает проблема: наша модель должна обучить $2 \times N \times V$ весов. Такая модель будет обучаться долго, хотим как-то оптимизировать
- Мы не можем вычислить ответ только для некоторых объектов, потому что для softmax необходимо знать вывод для всех



Negative Sampling

Хотя Negative Sampling - улучшение для Skip-Gram, он оптимизирует другую функцию:

$$\arg\max_{\theta} \prod_{w \in Text} \left[\prod_{c \in C(w)} p(c|w;\theta) \right] \longrightarrow \arg\max_{\theta} \prod_{(w,c) \in D} p(D=1|w,c;\theta)$$

Метод оптимизирует вероятность, что пара слово-контекст пришла из корпуса. У правого выражения есть тривиальное решение: всегда возвращать 1. Чтобы избежать этого, нужны негативные примеры (не из корпуса, плохих контекстов для слова)

Для этого можем случайным образом выделить слова из корпуса

$$\arg\max_{\theta} \sum_{(w,c) \in D} \log \frac{1}{1+e^{-v_c \cdot v_w}} + \sum_{(w,c) \in D'} \log\left(\frac{1}{1+e^{v_c \cdot v_w}}\right)$$

GloVe

- От Global Vectors, этот метод объединяет SVD и word2vec
- Строим co-occurrence матрицу X
- Хотим добиться следующего результата:

$$F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}} \longrightarrow F(w_i - w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}} \longrightarrow$$

$$F\left((w_i - w_j)^T \tilde{w}_k\right) = \frac{P_{ik}}{P_{jk}} \longrightarrow F\left((w_i - w_j)^T \tilde{w}_k\right) = \frac{F(w_i^T \tilde{w}_k)}{F(w_j^T \tilde{w}_k)}$$

$$F(w_i^T \tilde{w}_k) = P_{ik} = \frac{X_{ik}}{X_i}$$

GloVe

- Теперь хотим найти решение равенства на предыдущем слайде

$$w_i^T \tilde{w}_k = \log(P_{ik}) = \log(X_{ik}) - \log(X_i)$$

$$w_i^T \tilde{w}_k + b_i + \tilde{b}_k = \log(X_{ik})$$

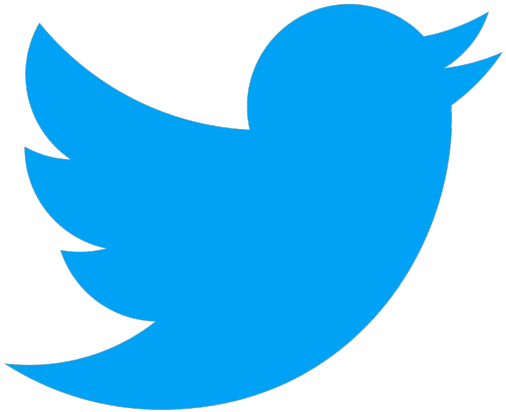
$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

- Здесь черта обозначает контекст
- *b* - bias, зависящий от слов в отдельности друг от друга
- *f(X)* - эвристика, балансирующая вклад пар. Она не убывает, растет медленно, чтобы очень частые пары не имели слишком большой вес, а f(0) = 0.

Символьные модели

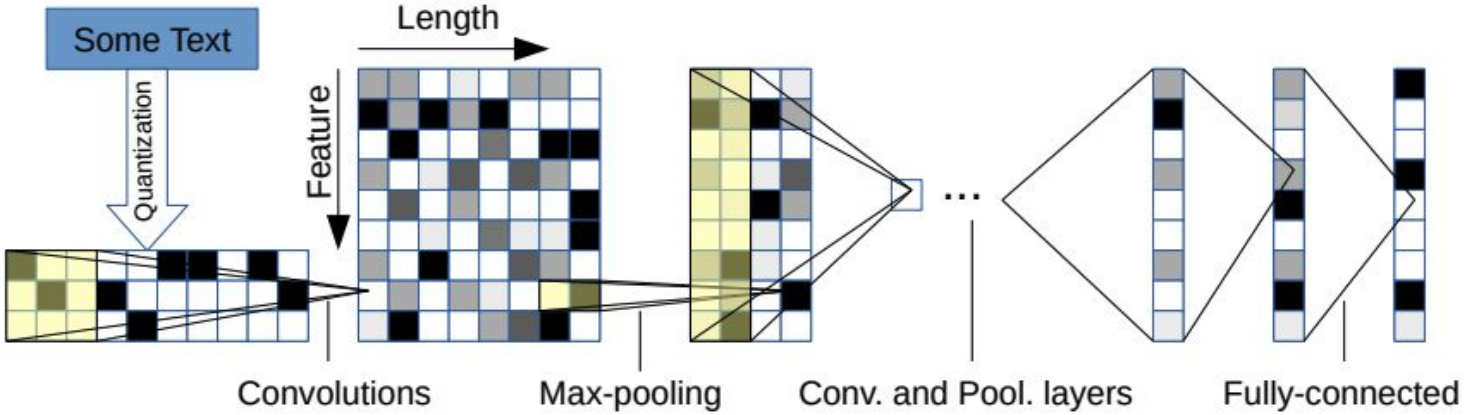
Почему мы иногда хотим работать с символами, а не словами?

1. Лучше для работы с OOV словами и очень редкими словами
2. Лучше для работы со словами с опечатками
3. Модель получается меньше, обучать проще
4. Другими словами, подходят для “шумных” текстовых данных



Символьные модели

- Определяем список символов
- Кодируем символы (one-hot)
- Обучаем языковую модель, например, предсказание следующего символа по текущему (1D-CNN)

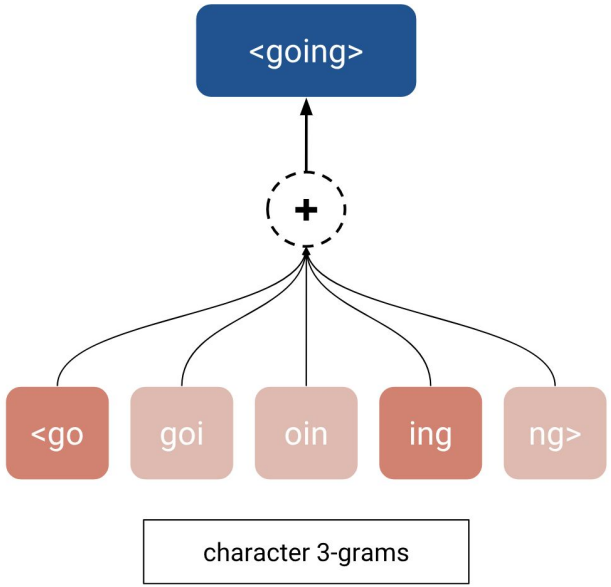


N-символьные эмбеддинги

- Строим векторные представления для токенов, состоящих из n символов, n-граммах
- Имеет многие преимущества посимвольного метода
- Одновременно имеет некоторую семантическую значимость (т.к. у морфем есть смысловая нагрузка)

fastText

- Основан на n-граммах
- Обучается аналогично word2vec
- Эмбеддинг слова вычисляется как усреднение эмбеддингов всех входящих в него n-грамм
- Намного лучше работает с OOV или очень редкими словами



Источники

- GloVe: Global Vectors for Word Representation,
  Jeffrey Pennington, Richard Socher, Christopher D. Manning
- word2vec Explained: Deriving Mikolov et al.’s Negative-Sampling Word-Embedding Method,
  Yoav Goldberg and Omer Levy
- Character-level Convolutional Networks for Text Classification*
  Xiang Zhang, Junbo Zhao, Yann LeCun
- Word2Vec (Skip-Gram model) Explained
- Co-occurrence matrix & Singular Value Decomposition(SVD)
- A Quick Overview of the Difference Between Word2vec and FastText
- FastText: Under the Hood