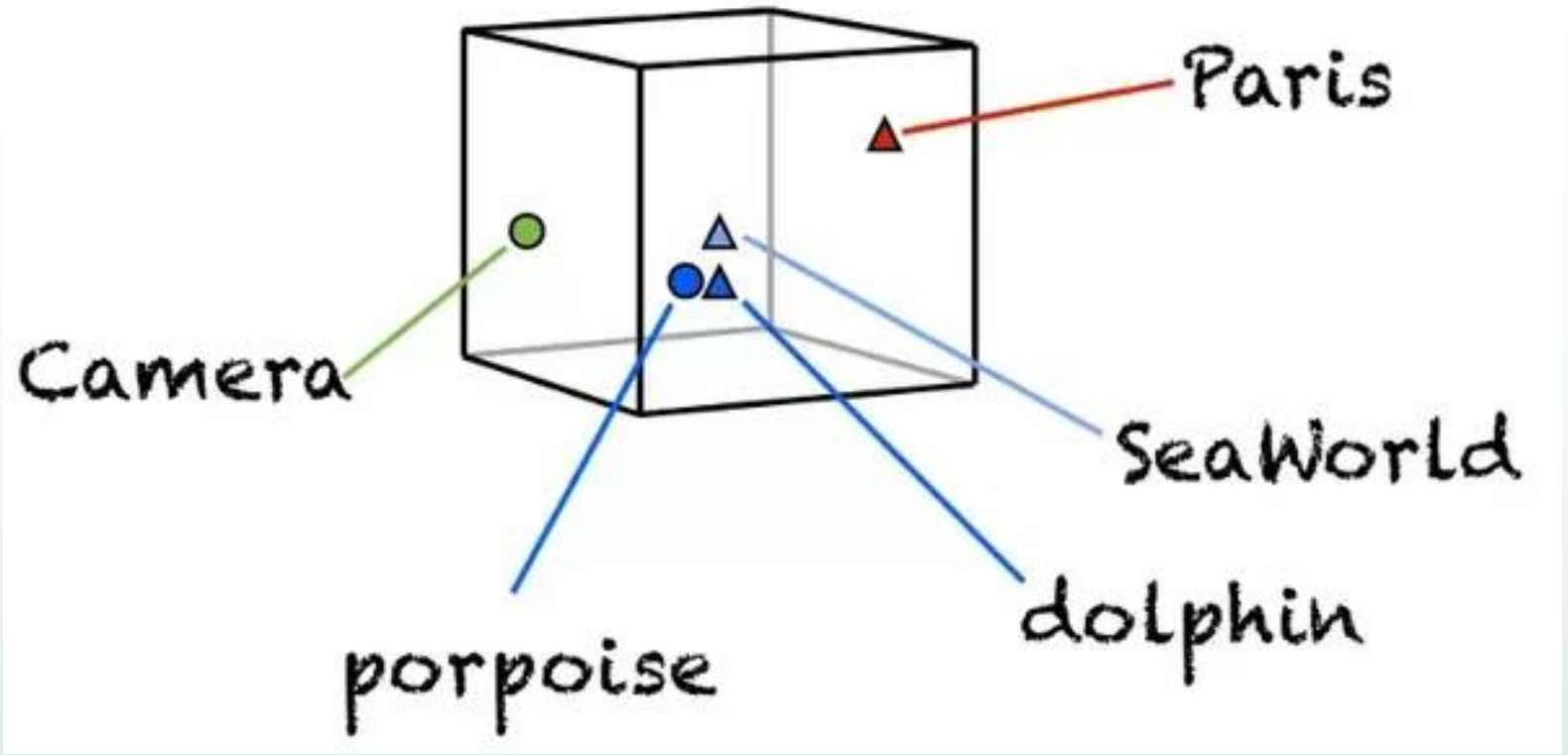
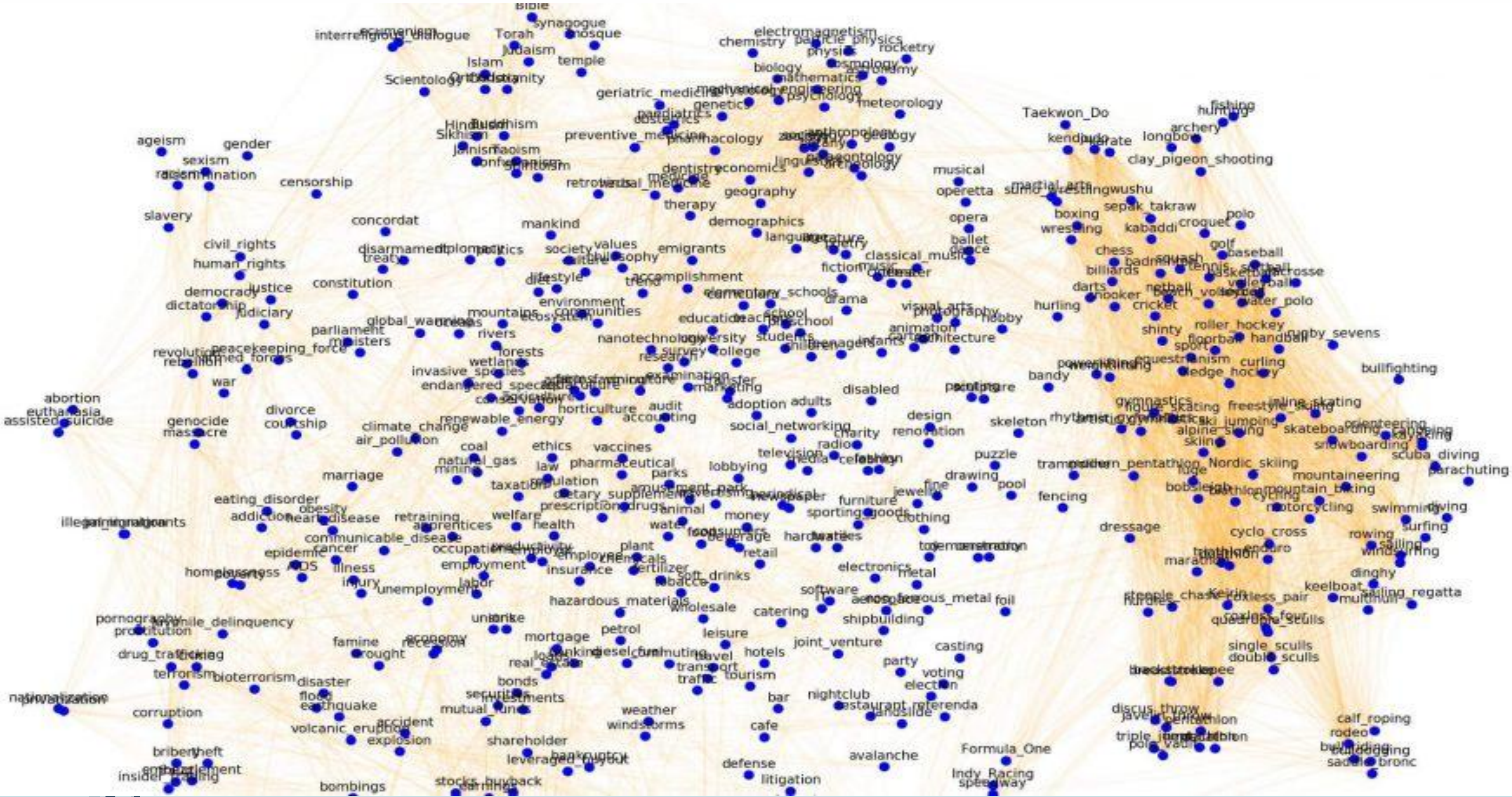


Word Embeddings. Word2vec

Векторные представления слов



word embeddings

Сжатые векторные представления слов

1. полезны сами по себе, например, для поиска синонимов или опечаток в поисковых запросах.

2. используются в качестве признаков для решениясамых различных задач:

- выявление именованных сущностей
- тэгирование частей речи
- машинный перевод
- кластеризация документов
- ранжирование документов
- анализ тональности текста

One-hot encoding

- Минусы

Признаковое пространство в one-hot векторах **имеет размерность, равную мощности словаря** коллекции, т.е. тысячи и десятки тысяч. Эта размерность растёт вместе с ростом словаря.

Никак не учитывает семантическую близость слов, **все векторы одинаково далеки друг от друга** в признаковом пространстве.

- Плюсы

Сжатые векторные представления для семантически близких слов близки как векторы (например, по косинусной мере). Это позволяет работать со словами, которых раньше не было в корпусе.

Сжатые векторные представления строятся в пространствах **фиксированной размерности** порядка десятков и сотен.

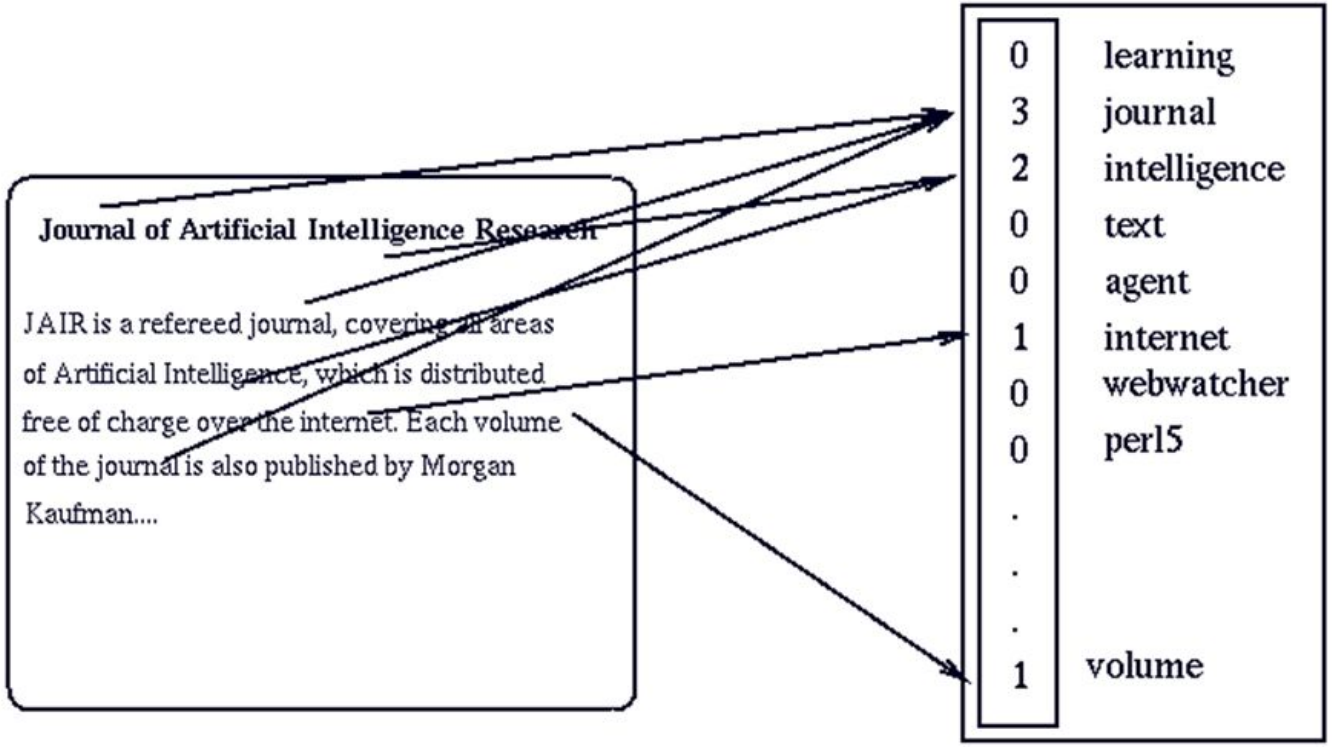
“мешок слов” (bag of words, BoW)

На выходе получим просто подсчет количества различных слов в тексте в одном векторе.

Мы теряем всю информацию о взаимном расположении слов внутри текста.

Terms	Documents								
	c1	c2	c3	c4	c5	m1	m2	m3	m4
computer	1	1	0	0	0	0	0	0	0
EPS	0	0	1	1	0	0	0	0	0
human	1	0	0	1	0	0	0	0	0
interface	1	0	1	0	0	0	0	0	0
response	0	1	0	0	1	0	0	0	0
system	0	1	1	2	0	0	0	0	0
time	0	1	0	0	1	0	0	0	0
user	0	1	1	0	1	0	0	0	0
graph	0	0	0	0	0	0	1	1	1
minors	0	0	0	0	0	0	0	1	1
survey	0	1	0	0	0	0	0	0	1
trees	0	0	0	0	0	1	1	1	0

Bag-of-words document representation



Алгоритм до word2vec

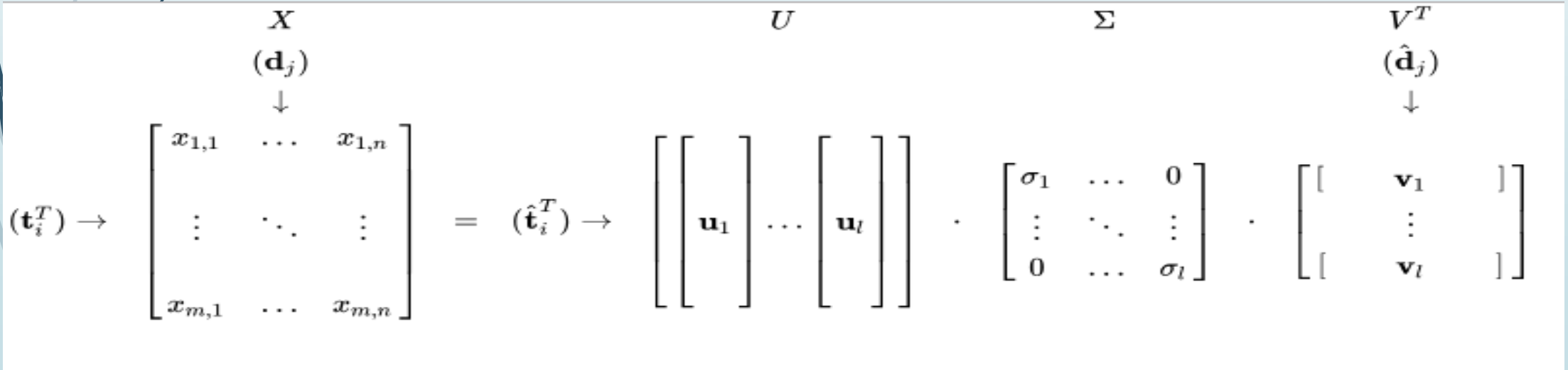
Представим наш корпус (набор текстов) в виде матрицы “слово-документ” (term-document).

1. По корпусу текстов D со словарём T строим матрицу со-встречаемостей X|T|×|T|

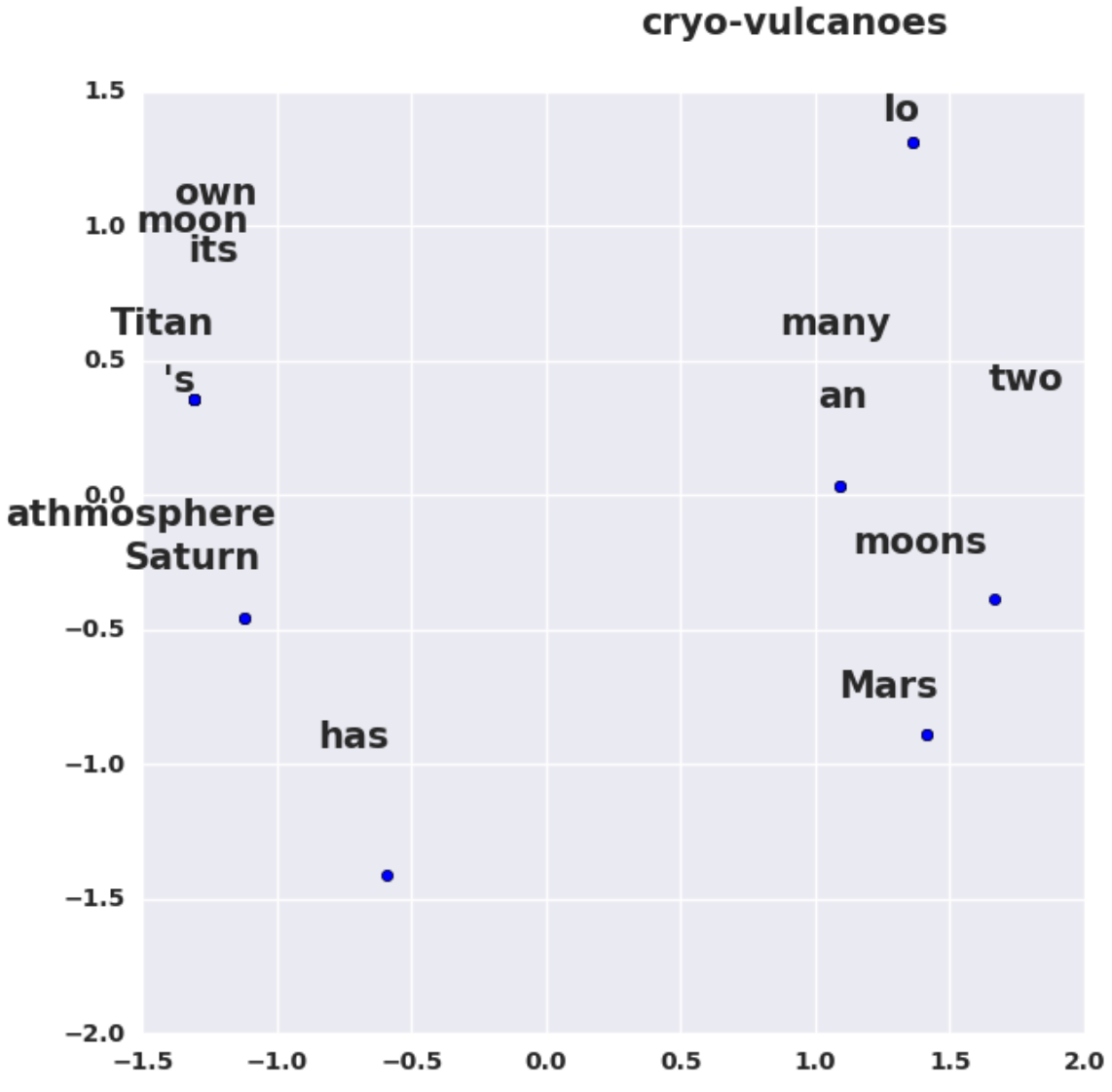
Можно понижать размерность:

2. SVD-разложение: X = USV T .

3. Из столбцов матрицы U выбираются первые K компонент



Пример



Используем следующий корпус слов:
s = ['Mars has an athmosphere', "Saturn 's moon Titan has its own athmosphere", 'Mars has two moons', 'Saturn has many moons', 'Io has cryo-vulcanoes']

Результаты, которые вы получите очень сильно зависят от корпуса, с которым вы работаете.

word2vec

В 2013 году мало кому известный чешский аспирант Томаш Миколов предложил свой подход к word embedding.

Слова, которые встречаются в одинаковых окружениях, имеют близкие значения

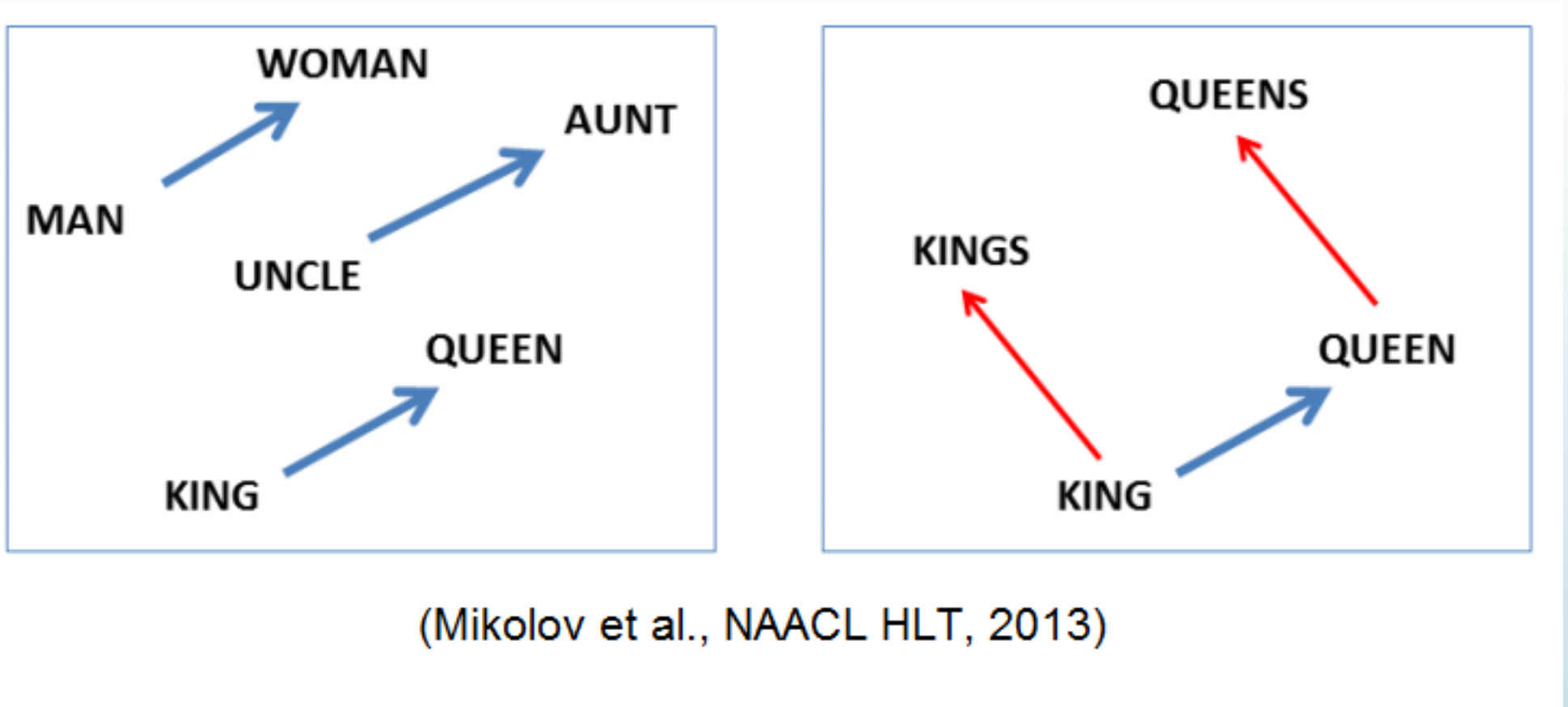
Учить такие вектора слов, чтобы вероятность, присваиваемая моделью слову была близка к вероятности встретить это слово в этом окружении в реальном тексте

softmax

$$P(w_o|w_c) = \frac{e^{s(w_o,w_c)}}{\sum_{w_i \in V} e^{s(w_i,w_c)}}$$

ω0— вектор целевого слова, ωc — это некоторый вектор контекста, вычисленный (например, путем усреднения) из векторов окружающих нужное слово других слов. А s(ω 1, ω 2) — это функция, которая двум векторам сопоставляет одно число.

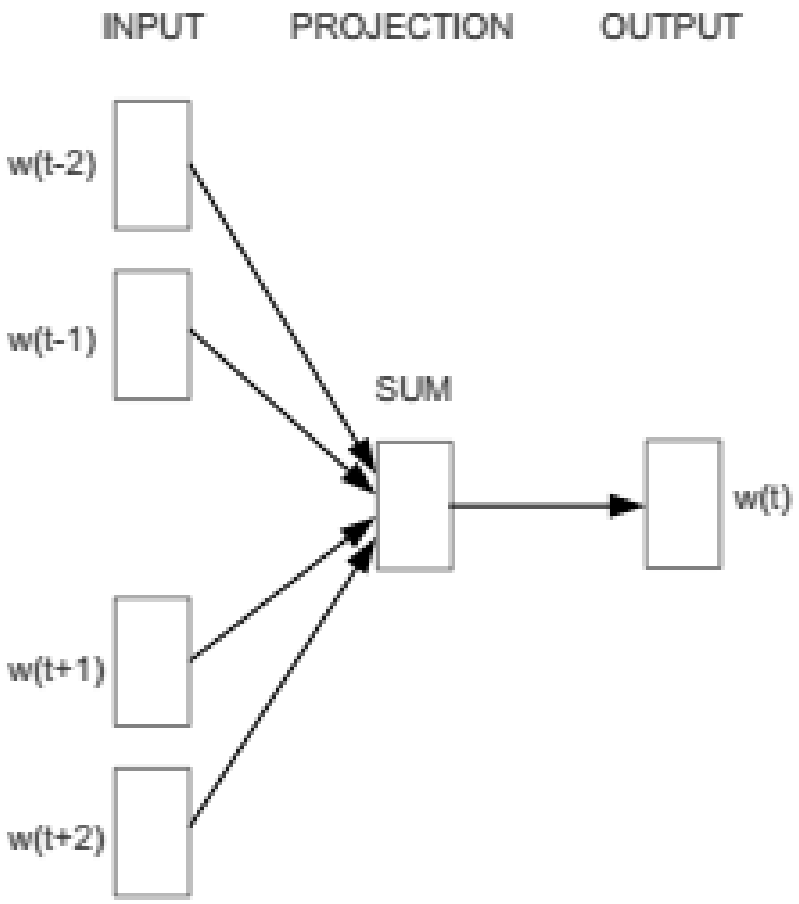
Классический пример



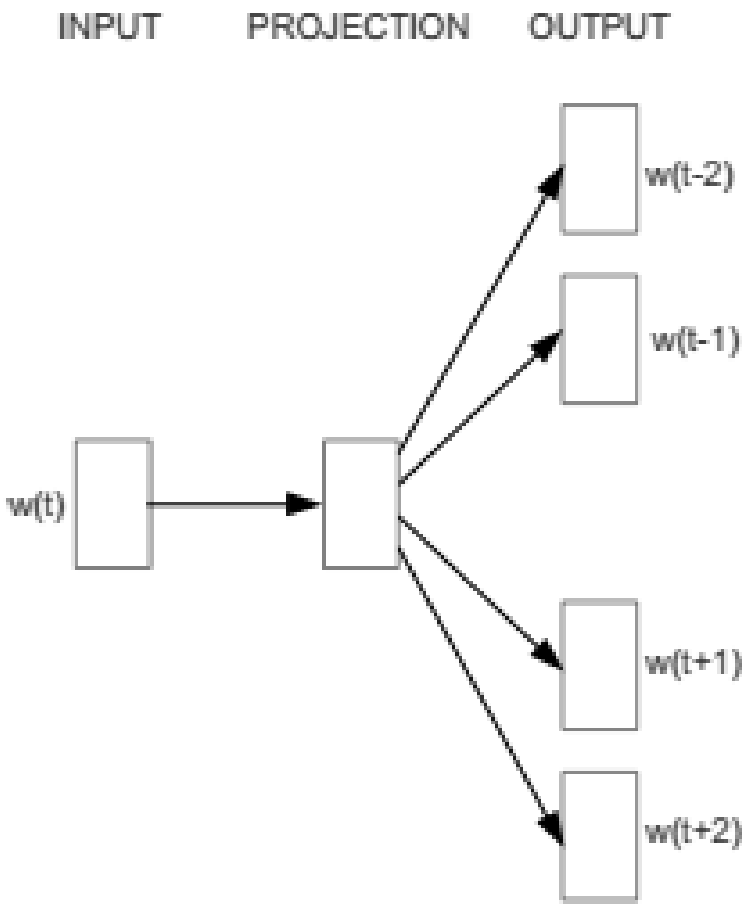
(Mikolov et al., NAACL HLT, 2013)

Хотя в модель не заложено явно никакой семантики, а только статистические свойства корпусов текстов, оказывается, что натренированная модель word2vec может улавливать некоторые семантические свойства слов.

Основные алгоритмы CBOW и Skip-gram



CBOW

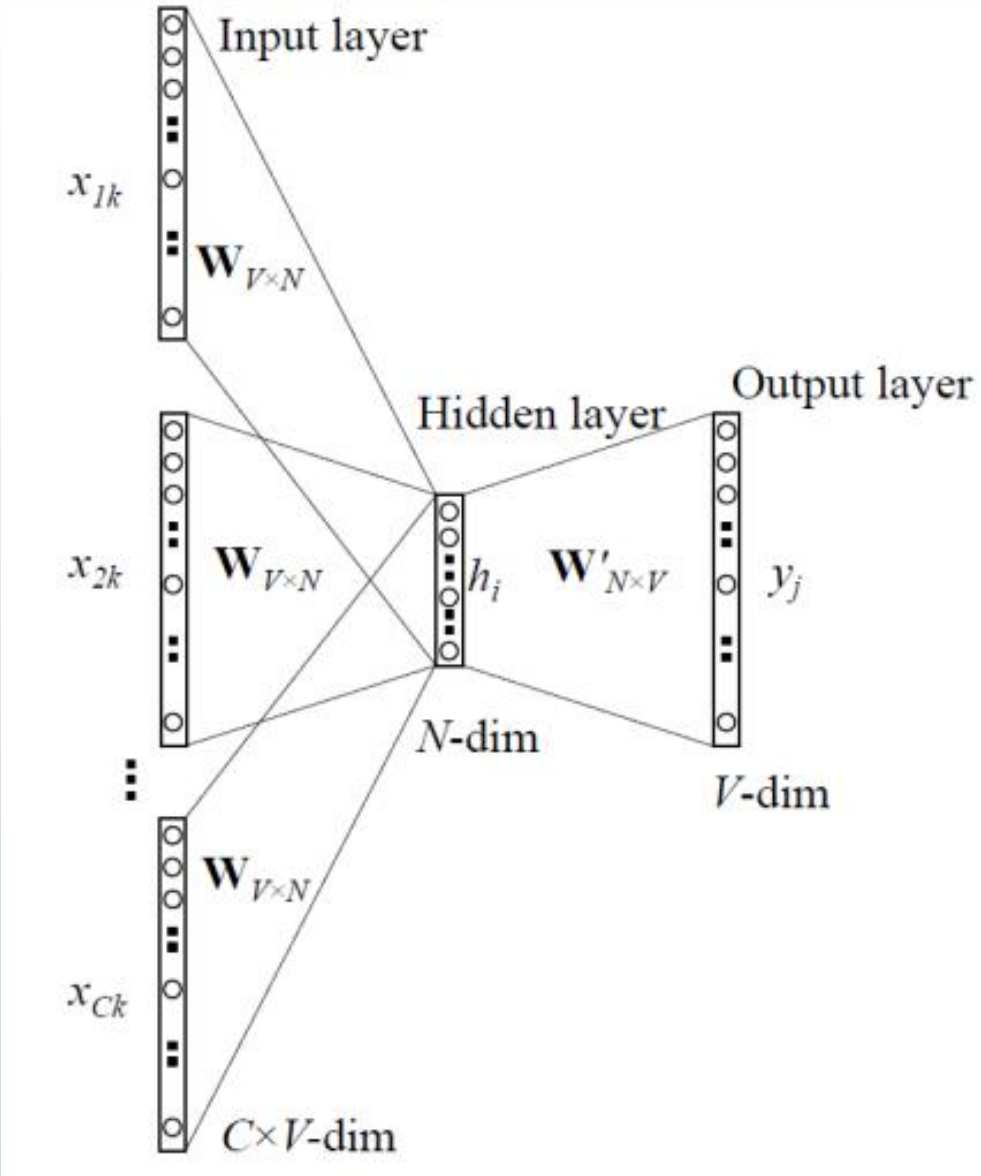


Skip-gram

Continuous Bag of Words

Процесс тренировки:

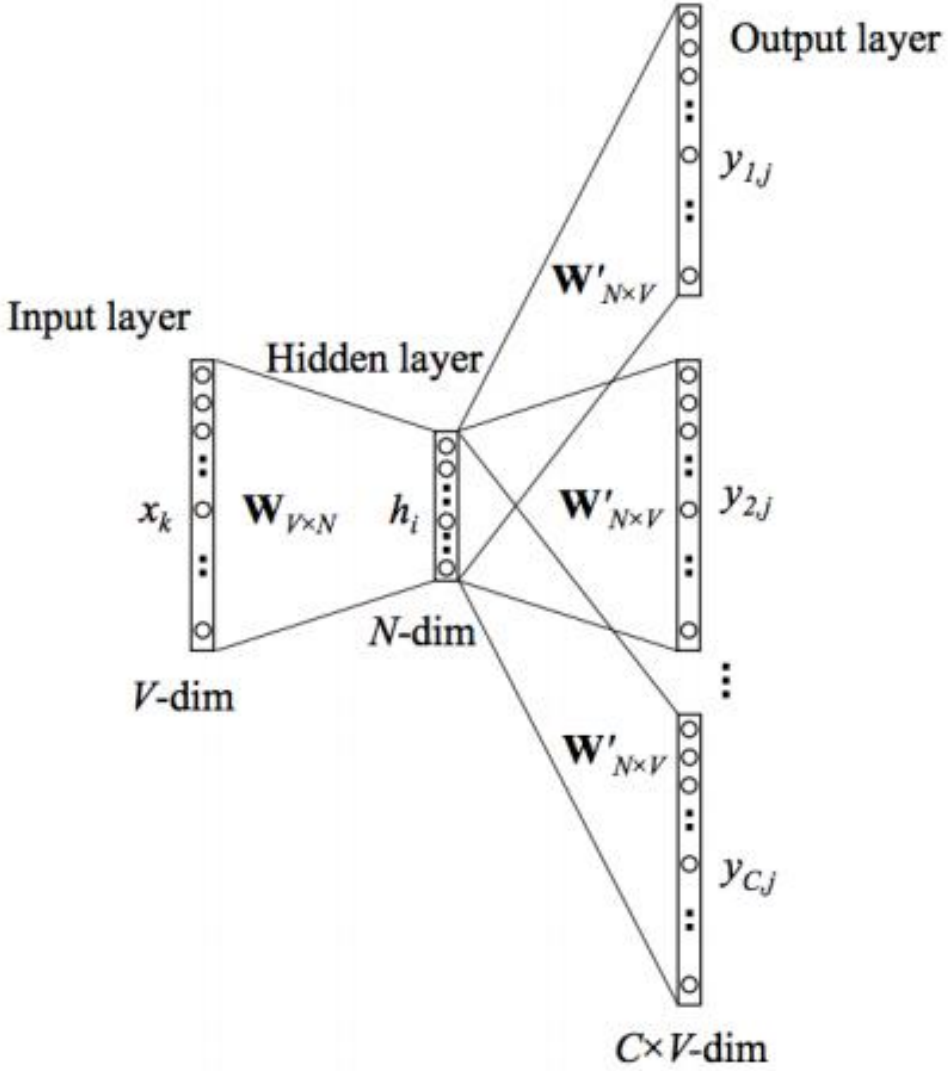
Берем последовательно (2k+1) слов, слово в центре является тем словом, которое должно быть предсказано. А окружающие слова являются контекстом длины по k с каждой стороны. Каждому слову в нашей модели сопоставлен уникальный вектор, который мы меняем в процессе обучения нашей модели.



Skip-gram

skip-gram - “словосочетание с пропуском”. Мы пытаемся из данного нам слова угадать его контекст (точнее вектор контекста)

k-skip-n-gram — это последовательность длиной n, где элементы находятся на расстоянии не более, чем k друг от друга.



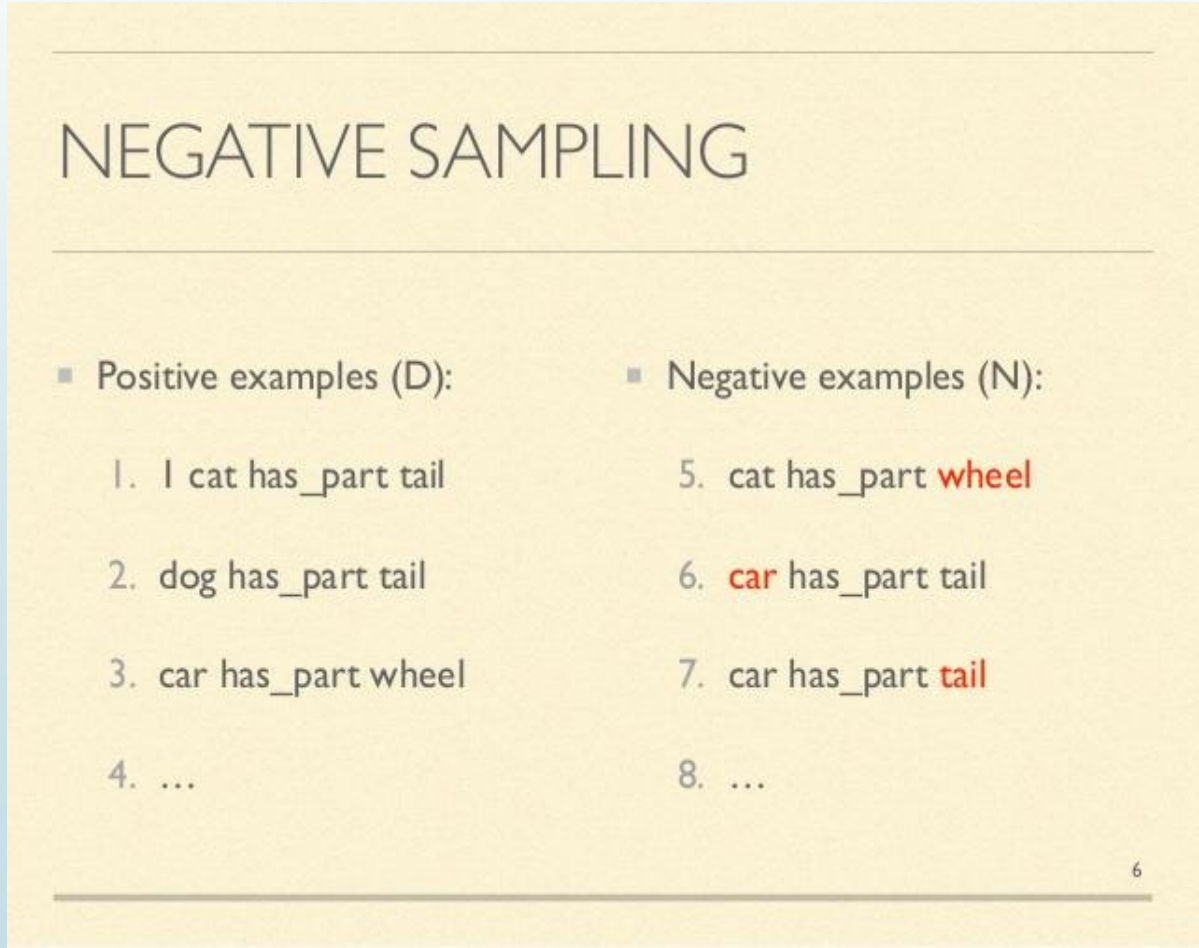
Negative Sampling

Суть этого подхода заключается в том, что мы максимизируем вероятность встречи для нужного слова в типичном контексте и одновременно минимизируем вероятность встречи в нетипичном контексте

$$NegS(w_o) = \sum_{i=1, x_i \sim D}^{i=k} -log(1 + e^{s(x_i, w_o)}) + \sum_{j=1, x_j \sim D'}^{j=l} -log(1 + e^{-s(x_j, w_o)})$$

Позитивная часть s(x, ω) отвечает за типичные контексты, и D — это распределение совместной встречаемости слова ω и остальных слов корпуса. Негативная часть --s(x, ω) — это, пожалуй, самое интересное — это набор слов, которые с нашим целевым словом встречаются редко. Этот набор порождается из распределения D', которое на практике берется как равномерное по всем словам словаря корпуса.

Negative sampling лучше работает с частотными словами и любит вектора слов небольшой размерности (50-100), работает быстрее, чем Hierarchical softmax.



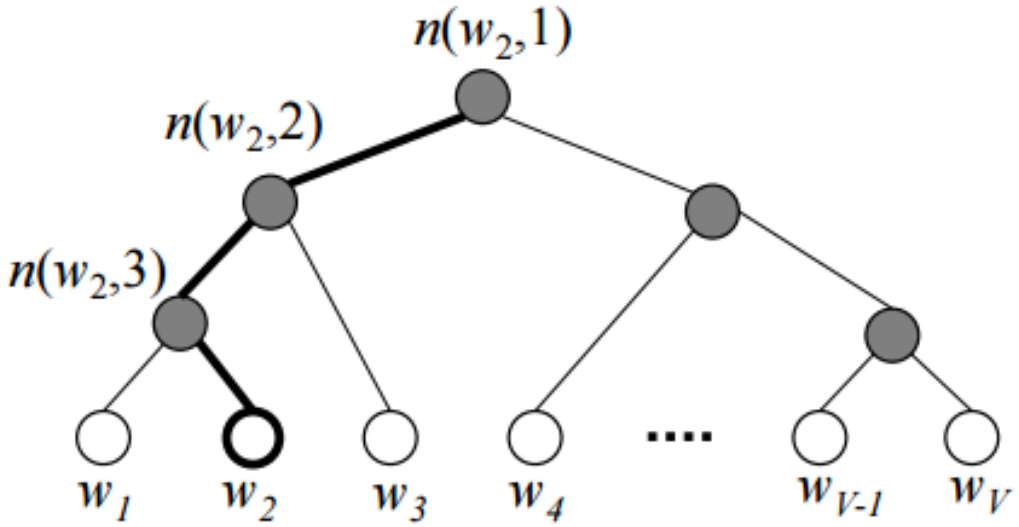
Hierarchical SoftMax

Можно не менять исходную формулу, а попробовать посчитать сам softmax более эффективно. Например, используя бинарное дерево (дерево Хаффмана). В полученном дереве V слов располагаются на листьях дерева.

Выделен путь от корня до слова w2. Длину пути обозначим L(w), а j-ую вершину на пути к слову w обозначим через n(w,j).

Вероятность того, что слово w будет выходным словом помощью иерархического softmax:

$$p(w = w_o) = \prod_{j=1}^{L(w)-1} \sigma([n(w, j+1) = lch(n(w, j))] v_{n(w,j)}^T u)$$



Hierarchical softmax лучше ведет себя при работе с не очень частотными словами, но работает медленнее, чем negative sampling.

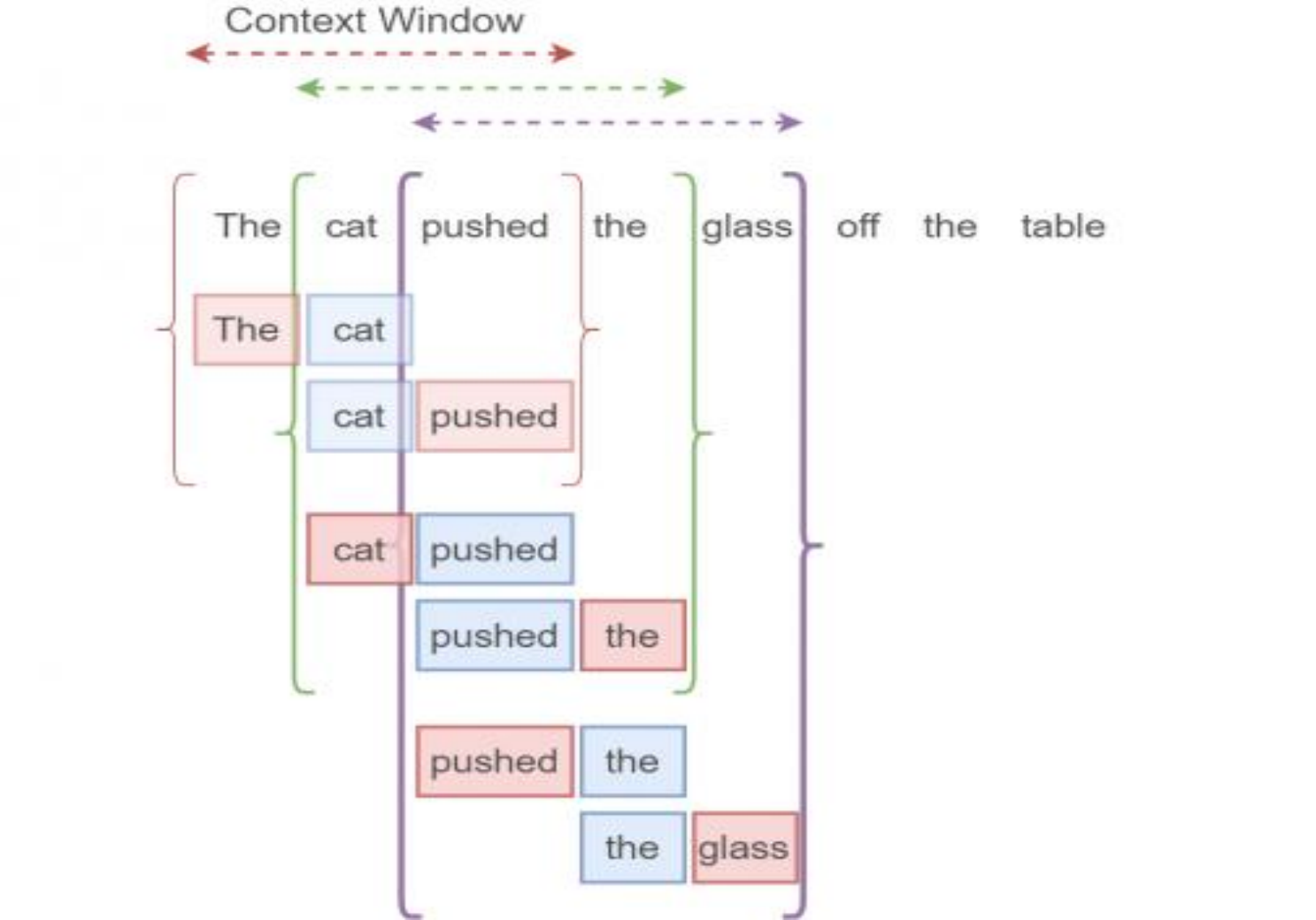
Алгоритм word2vec

- Читается корпус, и рассчитывается встречаемость каждого слова в корпусе (т.е. количество раз, когда слово встретилось в корпусе — и так для каждого слова)
- Массив слов сортируется по частоте (слова сохраняются в хэш-таблице), и удаляются редкие слова (их еще зовут гапаксами)
- Строится дерево Хаффмана. Дерево Хаффмана (Huffman Binary Tree) часто применяется для кодирования словаря — это значительно снижает вычислительную и временную сложность алгоритма.
- Из корпуса читается т.н. субпредложение (sub-sentence) и проводится субсэмплирование наиболее частотных слов (sub-sampling).
- По субпредложению проходим окном (размер окна задается алгоритму в качестве параметра). В данном случае под окном подразумевается максимальная дистанция между текущим и предсказываемым словом в предложении. Применяются алгоритмы на базе CBOW или Skip-gram.
- Применяется нейросеть прямого распространения (Feedforward Neural Network) с функцией активации иерархический софтмакс (Hierarchical Softmax) и/или негативное сэмплирование (Negative Sampling).

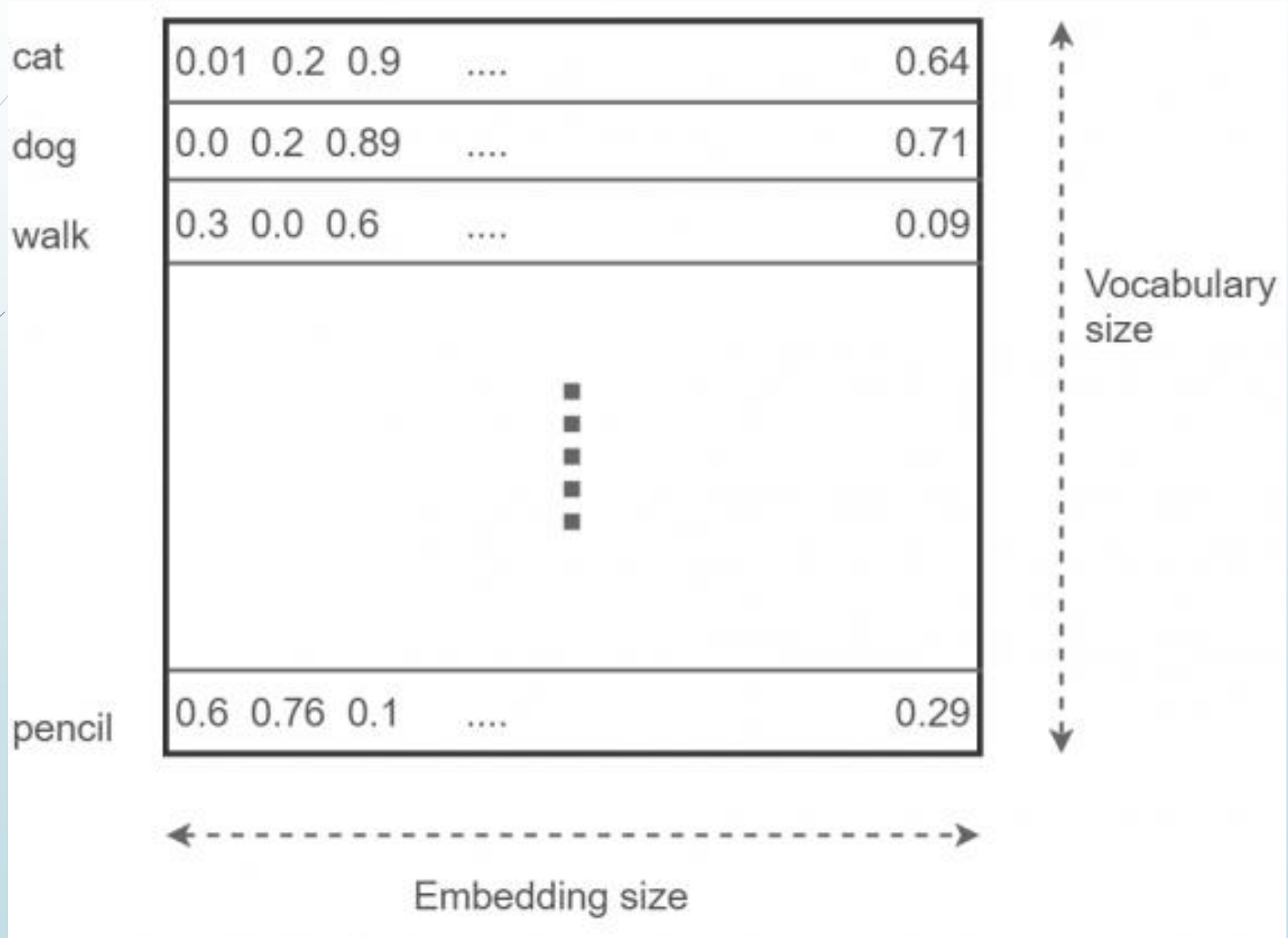
Backup

Окно

The cat pushed the glass off the table.



Embedding layer

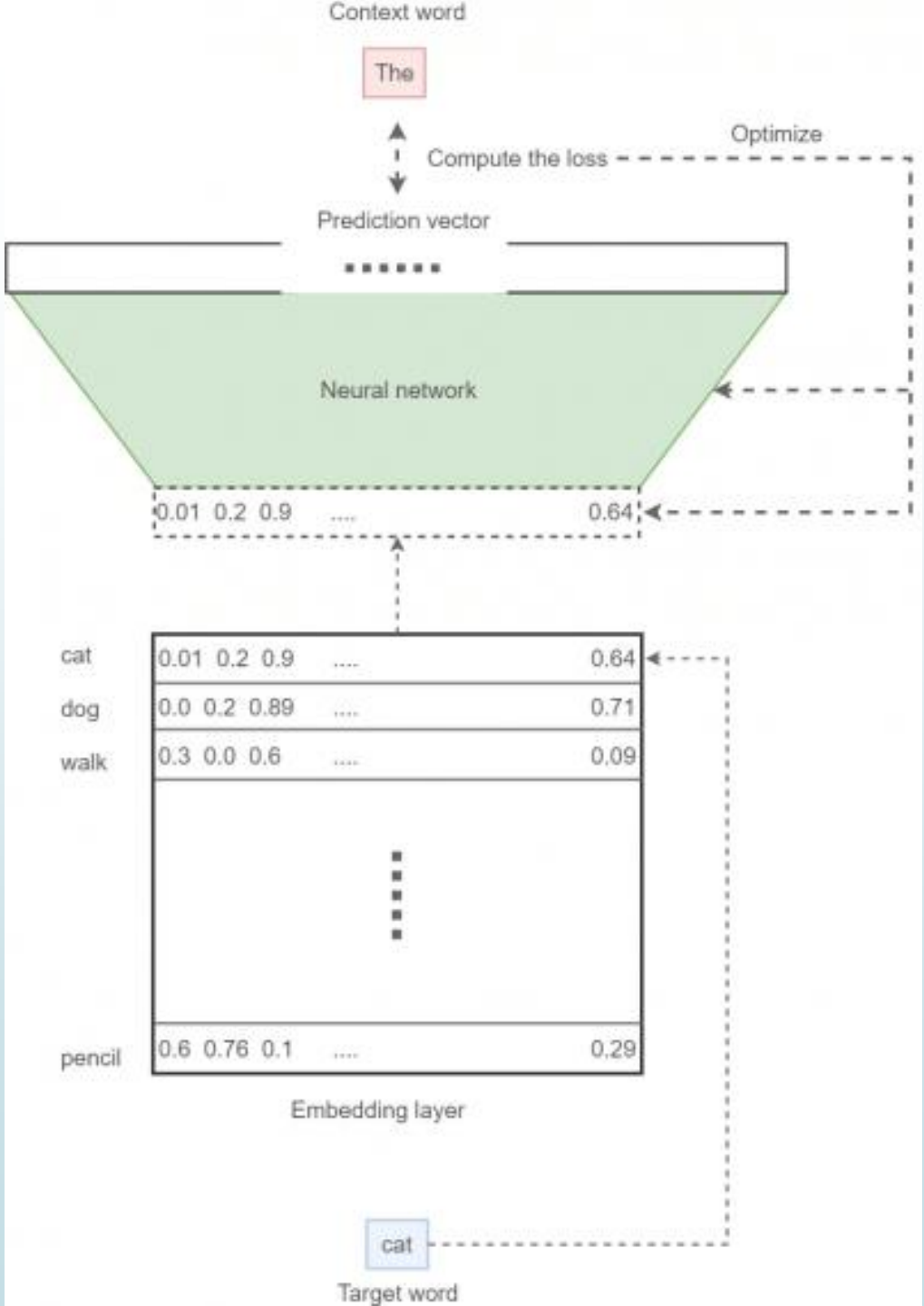


Обучение нейросети

- Для данного введенного слова (целевого слова) найдем соответствующий вектор из embedding layer;
- Скормим этот вектор нашей нейросети, затем попытаемся предсказать правильное выходное (контекстное) слово;
- Сравнив предсказанное слово и то слово, которое на самом деле находится в контекстном окне, вычислим функцию потерь;
- Используя функцию потерь вместе со стохастическим градиентным спуском, оптимизируем нейросеть и embedding layer.

Пример

Skip-gram + Negative Sampling



Функция потерь

$$\text{Loss} = \text{SigmoidCrossEntropy}(\text{Prediction}, \text{Correct Word}) + \sum_{1}^{K} E_{\text{Noise ID}} \text{SigmoidCrossEntropy}(\text{Prediction}, \text{Noise ID})$$

sampled softmax loss

SigmoidCrossEntropy это ошибка, которую мы можем определить на одном выходном узле независимо от остальных.