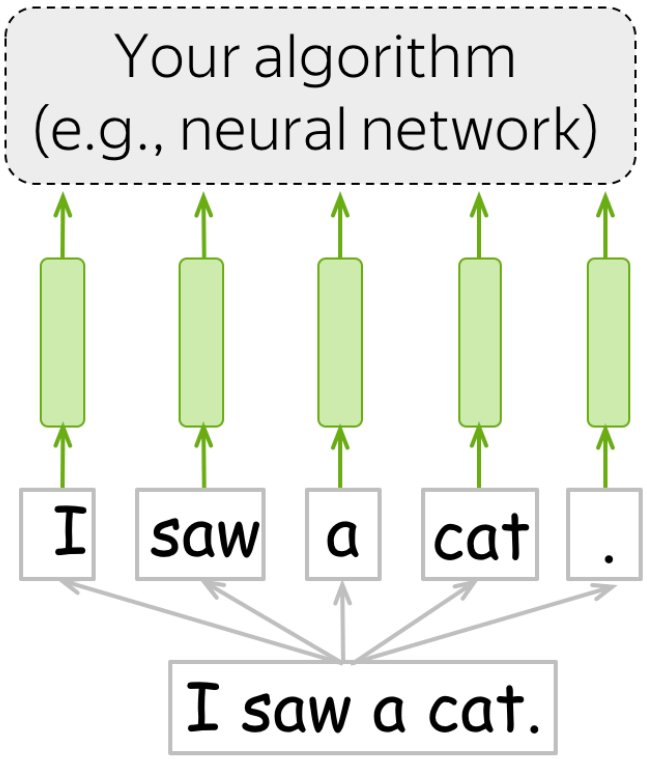
NLP Course

#NLP #embeddings

Введение

Что же такое word embeddings? Это векторна-числовая репрезентация токенов для модели.



Any algorithm for solving a task

Word representation - vector
(input for your model/algorithm)

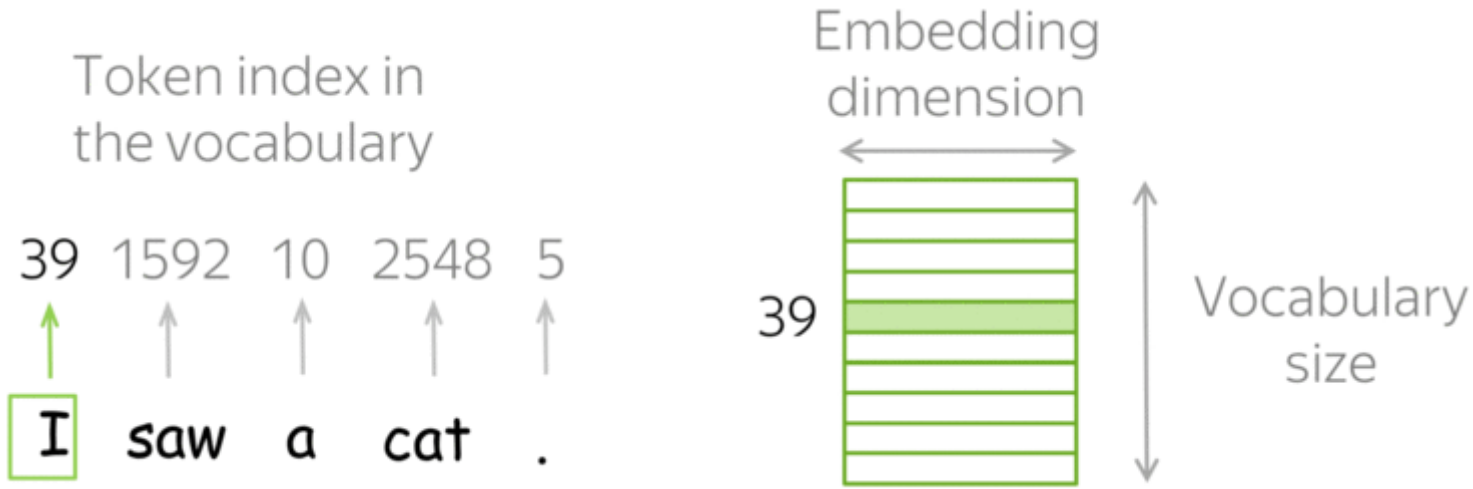
Sequence of tokens

Text (your input)

Vocabulary

На практике есть некоторый словарь доступных слов, которые вы выбираете заранее. В нём каждому слову из словаря соответсвует некоторый вектор - эмбединг, который находится по индексу слова в этом словаре.

Для слов которых нет в словаре, обычно, есть специальный токен UNK.



Как же получить эмбединг слова?

One-hot вектора

Презентация i-го слова в словаре, как вектора с нулями везде кроме i-го элемента, который равен единицы. Сам вектор имеет размерность словаря.

Недостатки:

- Не понимает значения слов (стол=собака=кот);
- Дорого для памяти, ну камон целый словарь хранить для каждого слова.

Distributional Semantics

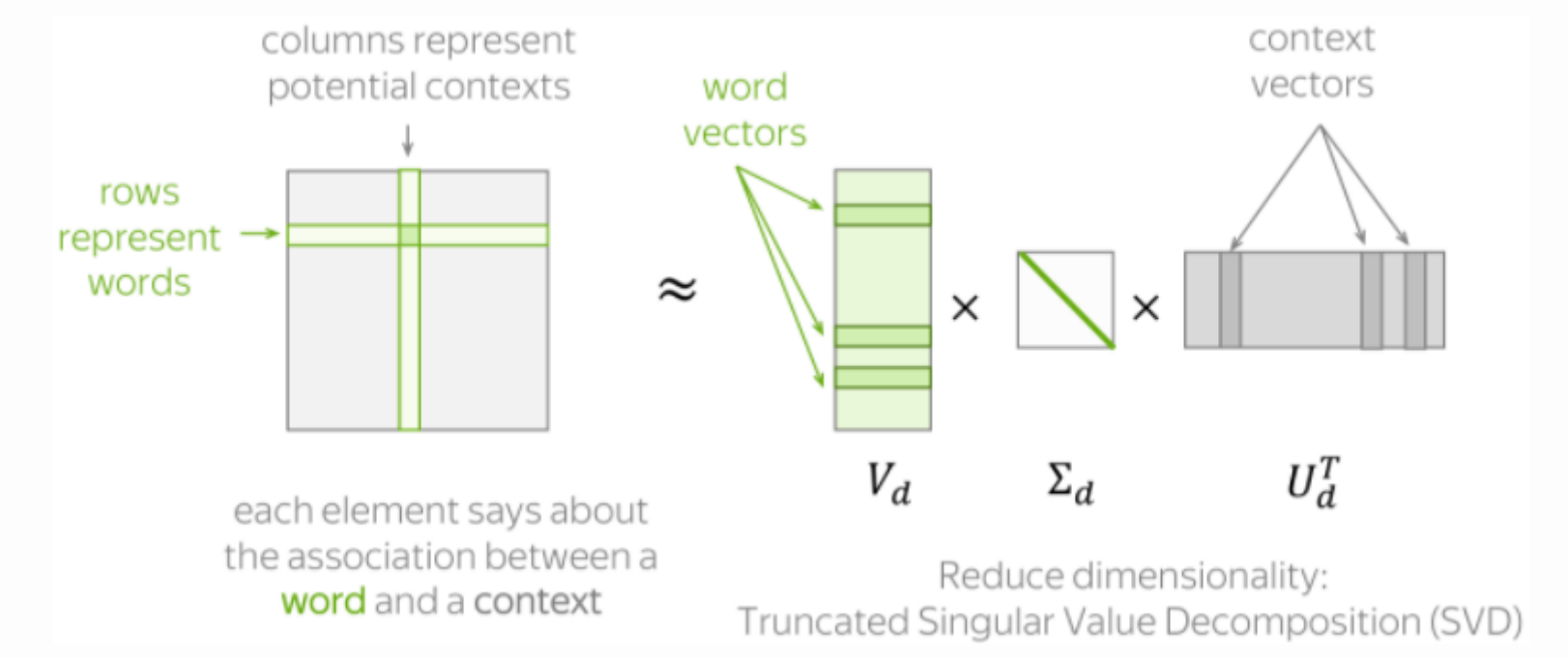
Как понять значения слов? Люди понимают значения слов по их контексту, поэтому слова которые часто встречаются в похожих контекстах -> имеют похожее значение.

> **Главная идея:** нам нужно засунуть информацию о контексте слова в векторное представление.

Count-based methods

Эти методы решают задачу прямоленейно.

> Count-based methods вносят информацию о контексте основываясь на global corpus statistics, то есть на глобальной статистике нашего корпуса текста.



Общий алгоритм этих методов:

1. Конструирование word-contex матрицы;
2. Уменьшаем её размерности.
    - для уменьшение занимаемой памяти;
    - для избавления от неинформативных контекстов, ведь слова появляются лишь в небольшом количестве контекстов.

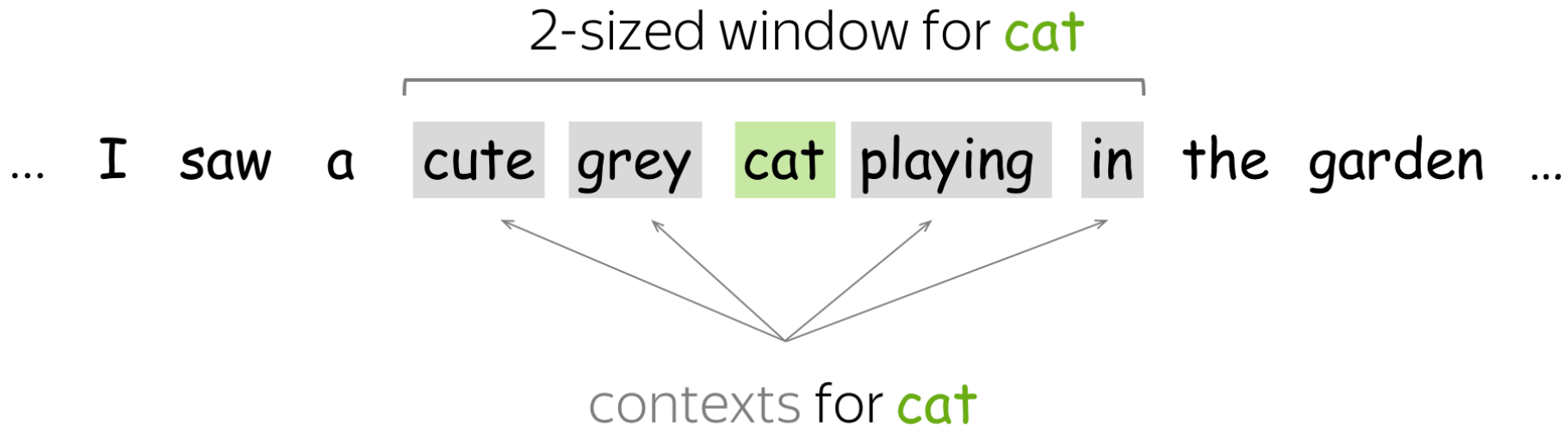
> dot product word/contex нормализированных векторов даёт близость/похожесть контекста к слову (или слова к контексту)

Для определния конкретного метода, требуется понять, что будет:

1. Possible contexts. Что вообще значит контекст?
2. The notion of association - формула для расчёта элементов нашей матрицы.

Simple: Co-Occurence Counts

Самый простой способ определить **контекст** это взять окно размера L оцентровать его по слову и посчитать кажое слово встречающиесе в окне с центральным словом (**элемент матрицы**). И пройтись этим способом по всему тексту.



Элемент матрицы (w, c) = $N(w, c)$ = это число раз, когда слово c появилось в контексте (окне) слова w.

Summarize

Контекст:

Окружающие слова в окне размера L

Элемент матрицы:

$N(w, c)$ - количество раз, когда слово c появилось в контексте (окне) слова w.

PPMI

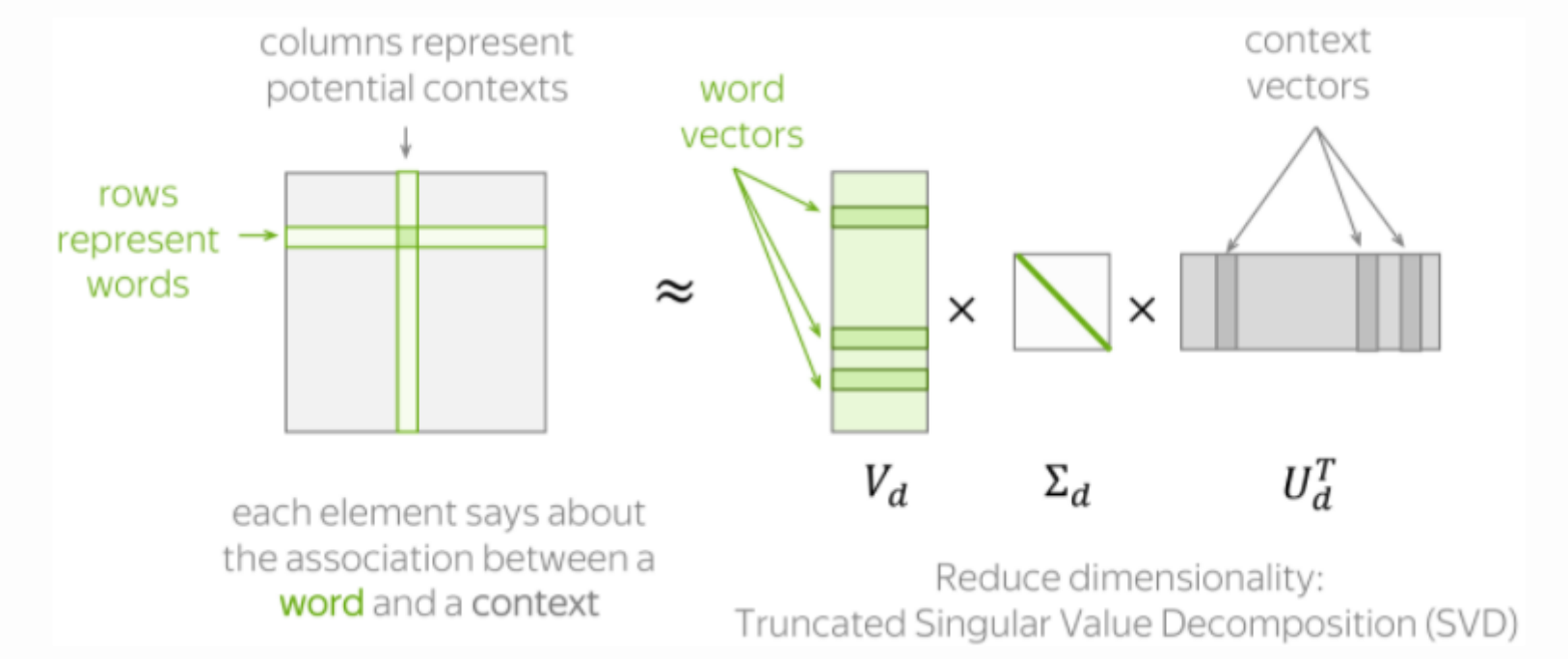
Исчтоник

#count-based-Method , #embeddings , #NLP

Count-based methods

Эти методы решают задачу прямоленейно.

> Count-based methods вносят информацию о контексте основываясь на global corpus statistics, то есть на глобальной статистике нашего корпуса текста.



Общий алгоритм этих методов:

1. Конструирование word-contex матрицы;
2. Уменьшаем её размерности.
    - для уменьшение занимаемой памяти;
    - для избавления от неинформативных контекстов, ведь слова появляются лишь в небольшом количестве контекстов.

> dot product word/contex нормализированных векторов даёт близость/похожесть контекста к слову (или слова к контексту)

Для определния конкретного метода, требуется понять, что будет:

1. Possible contexts. Что вообще значит контекст?
2. The notion of association - формула для расчёта элементов нашей матрицы.

Positive Pointwise Mutual Information (PPMI)

Контекст в этом методе определён также, но мера взаимосвязи слова и контекста более сложная: Суть PMI в получении числового значения правдоподобности того, что эти слова встретились неслучайно, при этом он учитывает вероятность того, что эти слова могут быть в одном контексте случайно.
Например. New York это не просто новый йорк - это город. Это словосочетание несёт в себе смысл и вероятность появление этих слов вместе высока. В то время как old garden в тексте про сады не несёт в себе той смысловой нагрузке, что New York это просто старый сад и эти слова скорее встретились случайно, чем это для них прям характерно garden может быть и nice, glow и blooming.

$$PMI(w,c) = \log \frac{P(w,c)}{P(w)P(c)} = \log \frac{P(w|c)}{P(w)} = \log \frac{P(c|w)}{P(c)} = \log \frac{N(w,c) * n}{N(w)N(c)}$$

> Заметка. На wiki используется $\log_2$

PMI значения.

- $P(w,c) = P(w)P(c)$ - слова встречаются вместе случайно - они независимы ($\log = 0$);
- $P(w,c) > P(w)P(c)$ - слова встречаются вместе чаще чем случайно ($\log > 0$);
- $P(w,c) < P(w)P(c)$ - слова встречаются вместе реже чем даже случайно ($\log < 0$).

$PPMI(w, c) = \max(0, PMI(w, c))$ и есть positive он отрубает отрицательные значения. Мотивация у него следующая:

- $PPMI$ появился из наблюдения, что отрицательные значения PMI ненадёжны (слова встречаются реже чем случайно -> корпус не полный), только если наш корпус текста не огромный;
- также он позволяет избежать $P(w, c) = 0$ (то есть слова не встречаются вместе). Этот случай плох тем, что $\log_2 0 = -\infty$

> На заметку. Было показано, что некоторые нейро-методы (например Word2Vec) неявно аппроксимируют факторизацию (#непонял) (сдвинутой) PMI матрицы.

Summarize

Контекст:
Окружающие слова в окне размера L

Элемент матрицы:

$$PPMI(w, c) = \max(0, PMI(w, c))$$

$$PMI(w, c) = \log \frac{P(w, c)}{P(w)P(c)} = \log \frac{N(w, c) * n}{N(w)N(c)}$$

LSA

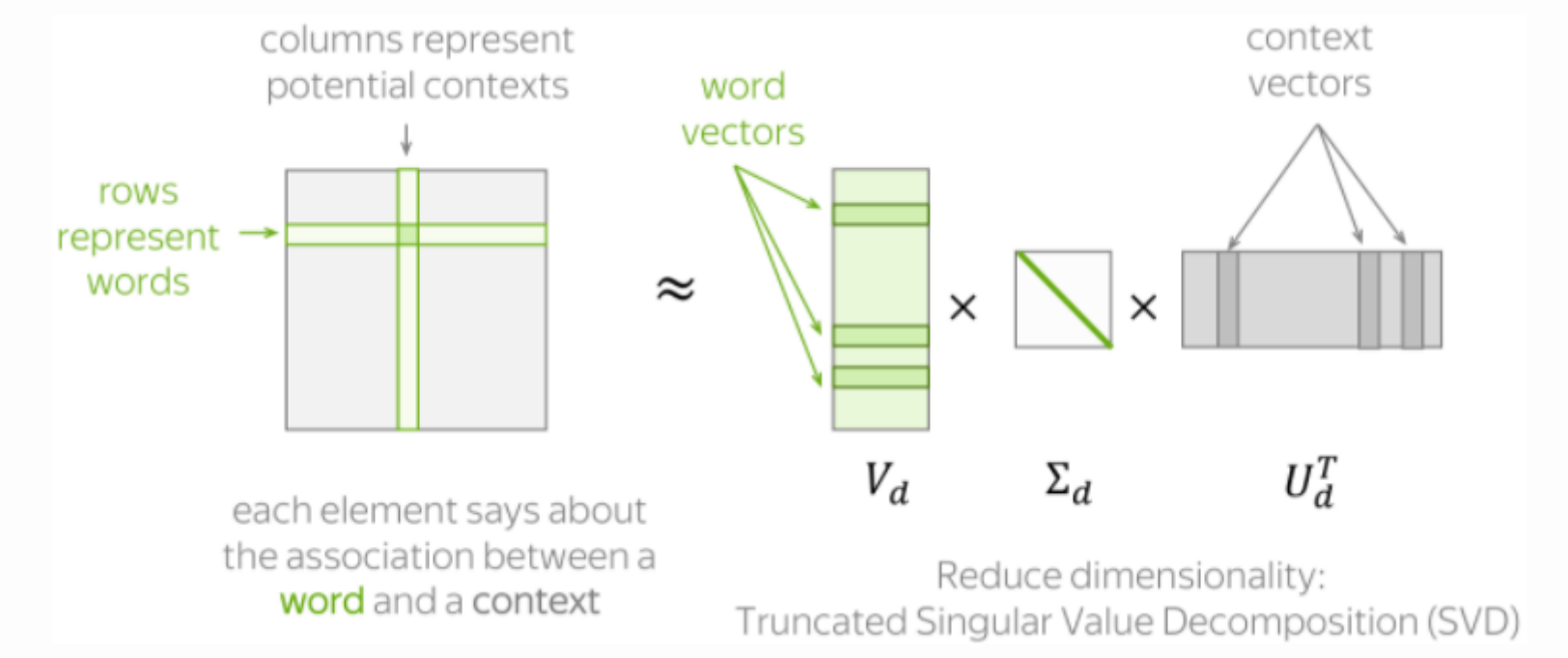
Источник

#count-based-method #embeddings #NLP

Count-based methods

Эти методы решают задачу прямоленейно.

> Count-based methods вносят информацию о контексте основываясь на global corpus statistics, то есть на глобальной статистике нашего корпуса текста.



Общий алгоритм этих методов:

1. Конструирование word-contex матрицы;
2. Уменьшаем её размерности.
    - для уменьшение занимаемой памяти;
    - для избавления от неинформативных контекстов, ведь слова появляются лишь в небольшом количестве контекстов.

> dot product word/contex нормализированных векторов даёт близость/похожесть контекста к слову (или слова к контексту)

Для определния конкретного метода, требуется понять, что будет:

1. Possible contexts. Что вообще значит контекст?
2. The notion of association - формула для расчёта элементов нашей матрицы.

Latent Semantic Analysis (LSA)

Это метод анализа коллекции документов, как следует в качестве контекста берётся весь документ, а не просто окно. То есть суть этого метода состоит в анализе связей между документами и терминами (словами) внутри них содержащихся.

> LSA предполагает, что слова близкие по значению встречаются в похожих текстах.

Так cosine similiarity между векторами документов, состоящими из числовых характерестик каждого слова, может использоваться для определения схожести документов.

LSA часто отсылыет к более общему подходу применения SVD апроксимации к term-document матрицы, элементы которой могут вычислятся по разному (e.g., simple co-occurrence, tf-idf, or some other weighting).

Пример такой матрицы можно увидеть на видео ниже. На нём показан процесс определния тем документов. Столбец отсылает к документу, а строка к слову, чем темнее цвет элемента тем больше его вес. Сначала сортируются документы по близости, после чего сортируются слова.

Word2Vec

Источник

#NLP , #embeddings , #prediction-based-method

Идея

В отличии от #count-based-method , которые реализуют идею того, что нам требуется "вставить информацию о конткесте в вектор слова (word vector)" довольно прямолинейно (буквально вставляют туда статистику копуса текста), word2vec решает эту проблему иначе, мы обучаем ML модель так, чтобы она научилась предсказывать контекст слова по контексту, то есть:

> **Узнаём** вектора слов (word vectors) обучая их **предсказывать контекст** (обычно соседние слова).

Word2Vec это модель чьи параметры это вектора слов. Эти параметры оптимизируются итеративно с использованием некоторой функции потерь. Эта функция потерь толкает вектора слов к тому, чтобы "знать" контекст в котором слово может появиться: вектора обучаются так, чтобы предсказывать возможный контекст соответсвующего слова.

> Если вектор слова "знает" контекст, то он "знает" и значение слова.

Основная идея word2vec следующая:

1. взять корпус текста
2. пройтись по нему "окном" двигаясь по одному слову (как в самом простом #count-based-method подходе)
3. Для центрального слова посчитать вероятность "контекста", то есть произведение вероятностей появления центрального слова с контекстными словами (словами окружающими центральное в окне)
4. Оптимизировать параметры модели, чтобы увеличить эту вероятность

Изображение одного шага, где cat - центральное слово; cute, grey, playing, in - контекстные слова.

Метод подробнее

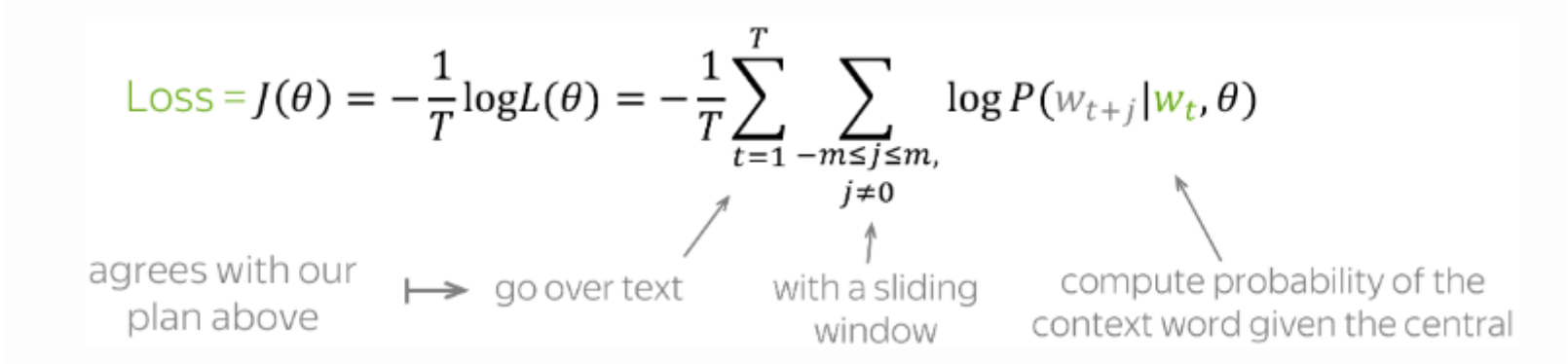
Функция потерь (Loss function)

Для каждой позиции $t = 1, \ldots, T$ в корпусе текста, word2vec предсказывает вероятность появления контекстных слов в m-sized окне, при условии центрального слова w_t:

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

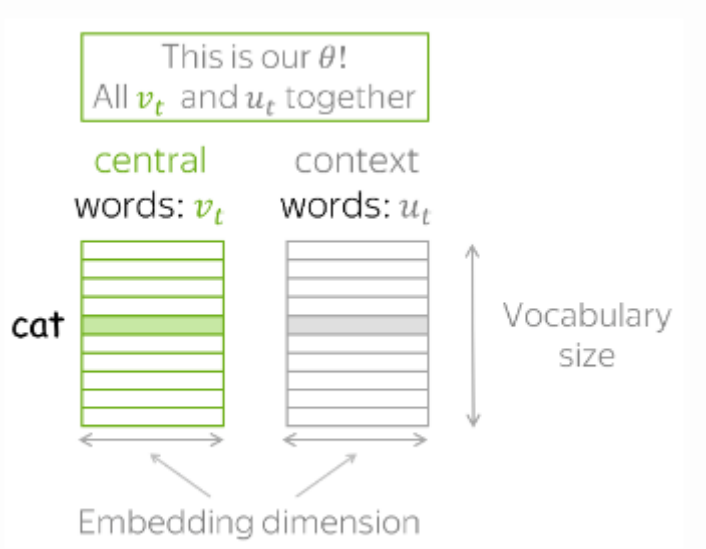
где θ это все параметры для оптимизации.

Тогда функция потерь (loss function) это average negative log-likelihood:



Для каждого слова w существует два вектора:

- Когда слово является центральным v_w
- Когда слово является контекстным u_w (Этот вектор обычно откидывется после обучения)



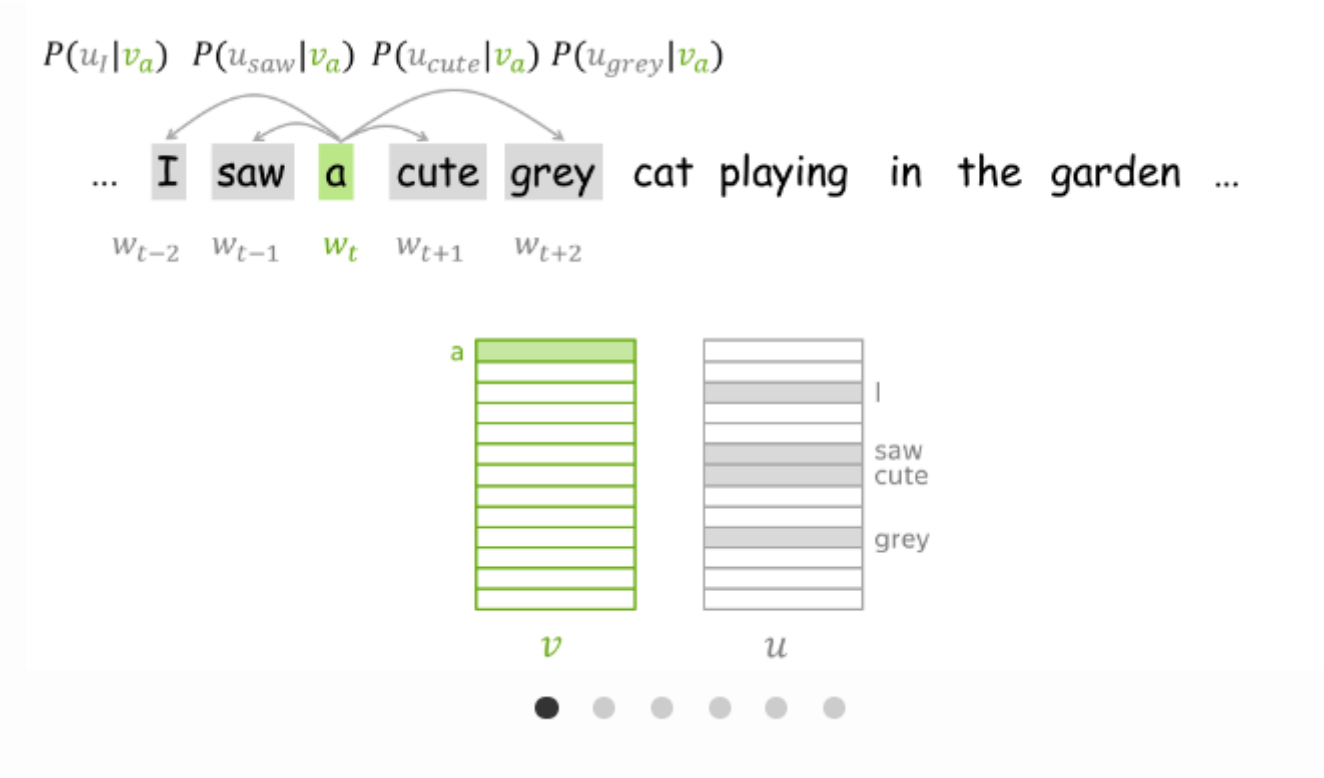
Тогда вероятность появления контестного слова o при условии центрального слова c (вероятность появления слова o в контексте слова c (контекст принадлежащий как бы слову c)) будет равняться:

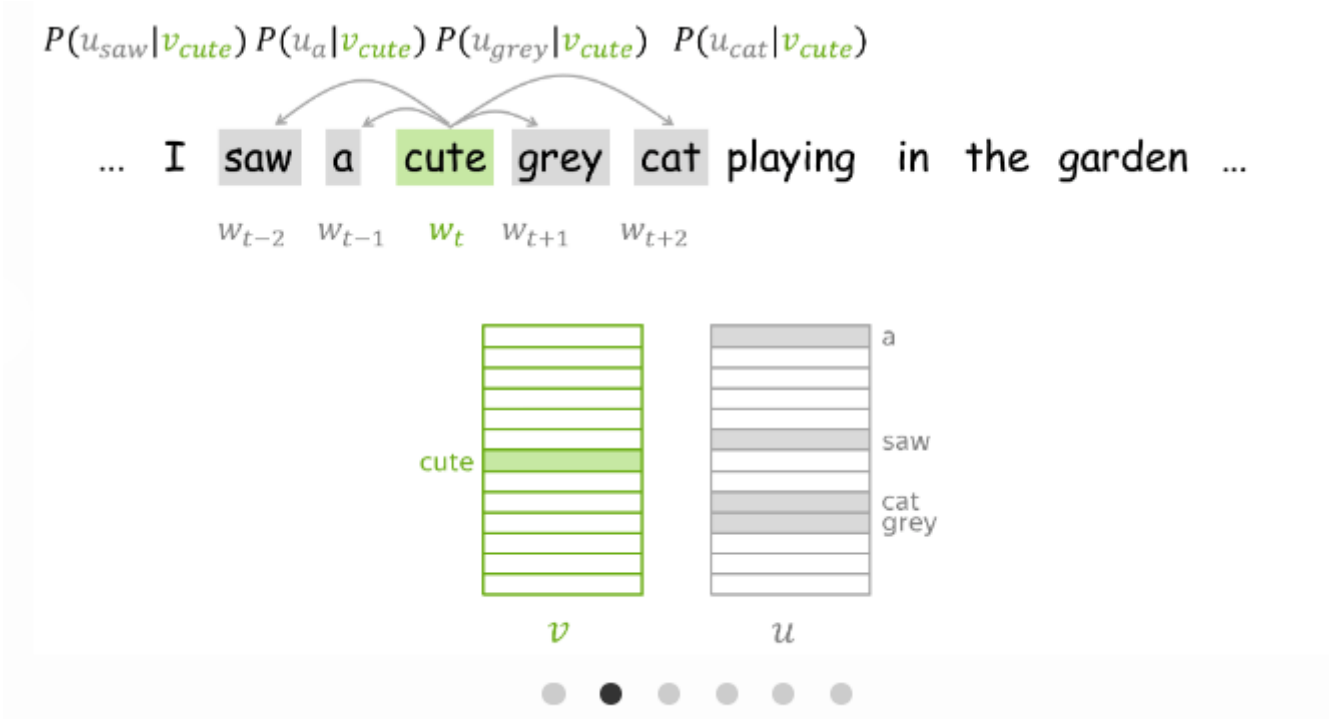
$$P(o|c) = \frac{exp(u_o^T v_c)}{\sum_{w \in V} exp(u_w^T v_c)}$$

(Да-да это Softmax)

Note. Идея окном по тексту стоит заметить что слово может быть как контекстом другого слово, так и центральным.

Для каждой пары слов центральное + контекст мы отдельно делаем апдейт.



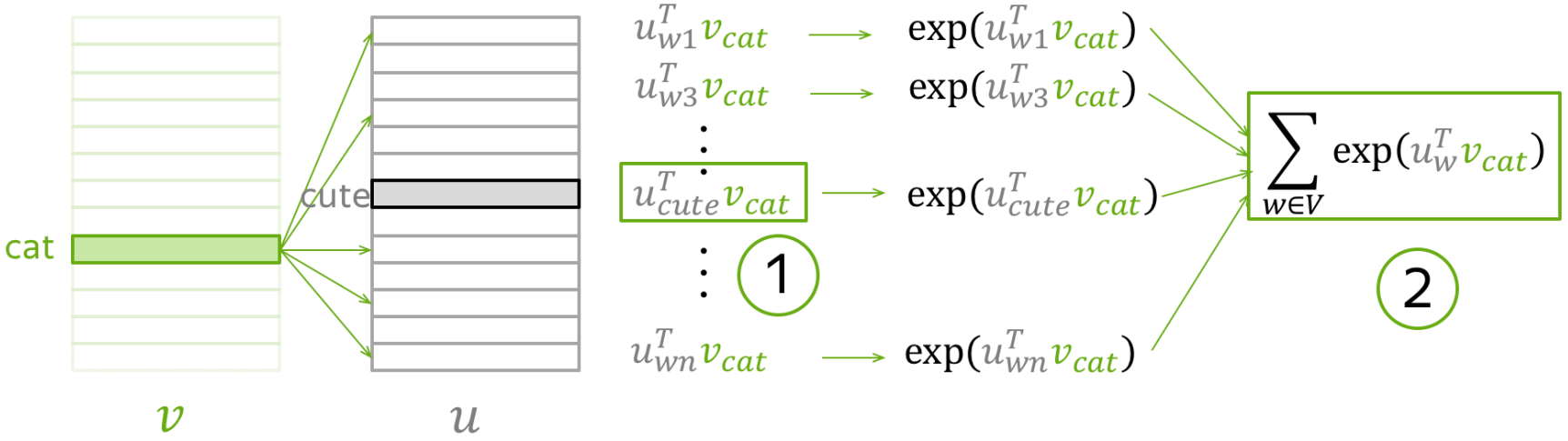

$P(u_{saw}|v_{cute})$ $P(u_a|v_{cute})$ $P(u_{grey}|v_{cute})$ $P(u_{cat}|v_{cute})$
... I saw a cute grey cat playing in the garden ...
w_{t-2} w_{t-1} w_t w_{t+1} w_{t+2}
cute
a
saw
cat
grey
v
u

One step

1. Take dot product of v_{cat} with **all** u

2. exp

3. sum all



4. get loss (for this one step)

$$J_{t,j}(\theta) = \underbrace{-u_{cute}^T v_{cat}}_{1} + \underbrace{\log \sum_{w \in V} \exp(u_w^T v_{cat})}_{2}$$

5. evaluate the gradient, make an update

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

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

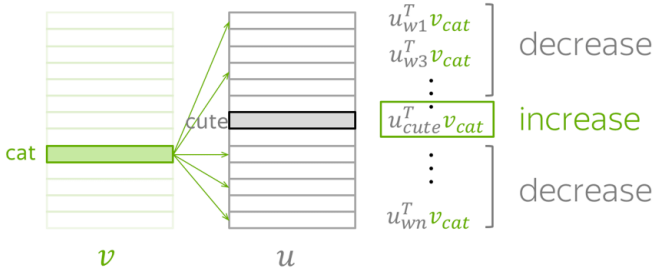
На кажом шаге (апдейте) мы увеличиваем вероятность токена контекста, а остальных слов (векторов контекста) уменьшаем.

Делая апдейты для всех контекстных слов мы получаем среднее над всеми апдейтами. То есть вероятность схожего контекста у синонимов или похожих по значению слов будет выше чем у других.

Fast training or Negative sampling

Изначально мы имеем $|V| + 1$ обучаемых параметров, что много с учётом того, что большинство слов словаря не подходят к центральному. Поэтому если мы будем обновлять лишь K случайных векторов (исключая контекстное слово) - мы получим в среднем тот же результат, но уменьшим обучаемые параметры до $K + 2$. Из-за большого кол-ва токенов при обучении каждое слово будет обновляться достаточное кол-во раз, чтобы выучить взаимосвязь между словами.

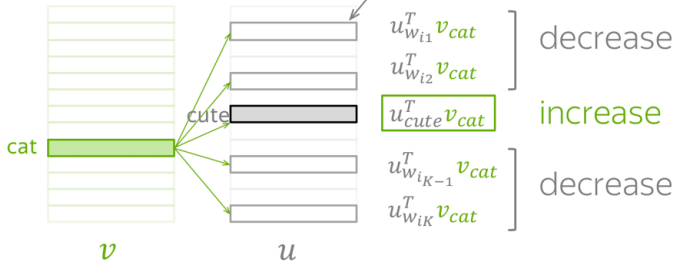
Dot product of v_{cat}:
- with u_{cute} - increase,
- with **all other** u - decrease



Parameters to be updated:
- v_{cat}
- u_w for all w in the vocabulary — |V| + 1 vectors

Dot product of v_{cat}:
- with u_{cute} - increase,
- with **a subset of other** u - decrease

Negative samples: randomly selected K words



Parameters to be updated:
- v_{cat}
- u_{cute} and u_w for w in K negative examples — K + 2 vectors

Обновленное лосс функция (теперь используется не софтмакс ведь при обновление вероятности обновляемых слов не складываются в единицу):

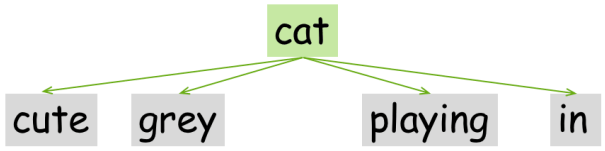
$$u^T_{cute}v_{cat} \qquad u^T_w v_{cat}$$

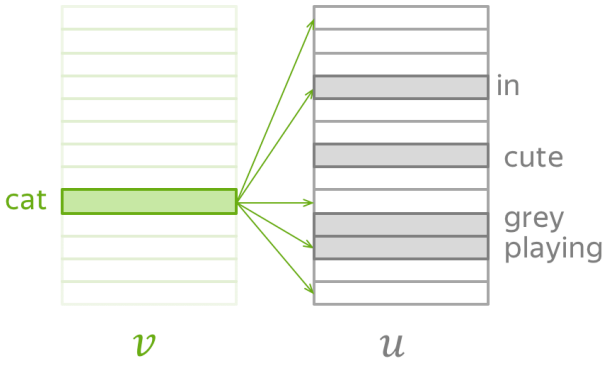
или

$$u^T_{cute}v_{cat} \qquad u^T_w v_{cat}$$

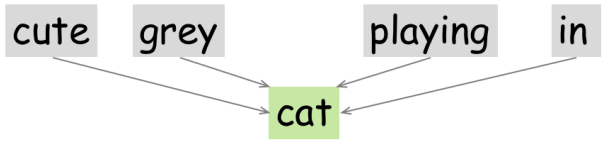
Word2Vec variants

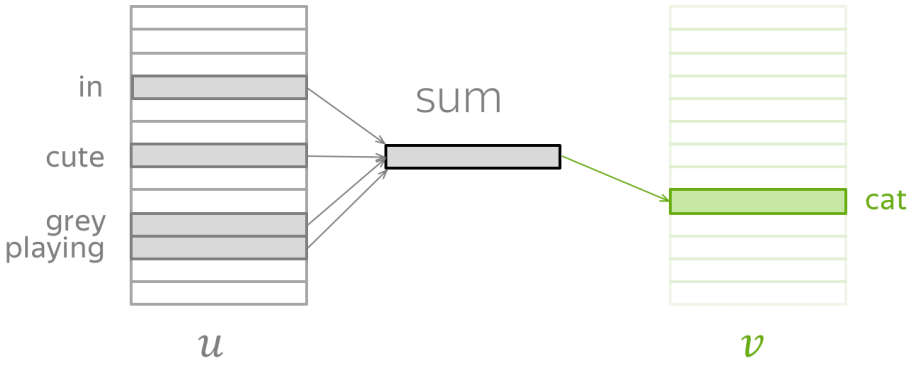
… I saw a cute grey cat playing in the garden …





Skip-Gram: from central predict context (one at a time)





CBOW: from sum of context predict central

Additional notes

- Visualization of Word2vec using different decomposition methods 🤩
- Word2Vec learn high-quality word representation very fast.

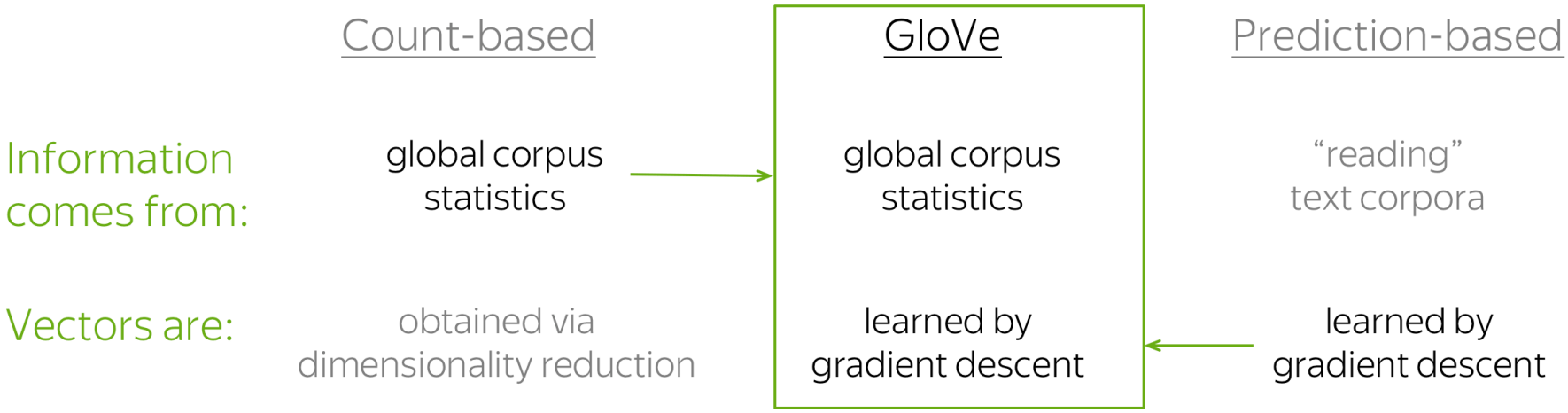
- Windows effect
  - Larger windows – more topical similarities

  - Smaller windows – more functional and syntactic similarities

GLoVe

#NLP #prediction-based-method #count-based-method #embeddings

Идея



Метод кратко

Схоже с Word2Vec мы имеет центральное слово и контекстные - наши параметры. Дополнительное мы добавляем scalar bias для кадого вектора.
Дополнительно метод контролирует влияние редко и часто встречающихся слов:

- rare events are penalized,
- very frequent events are not over-weighted.

context vector

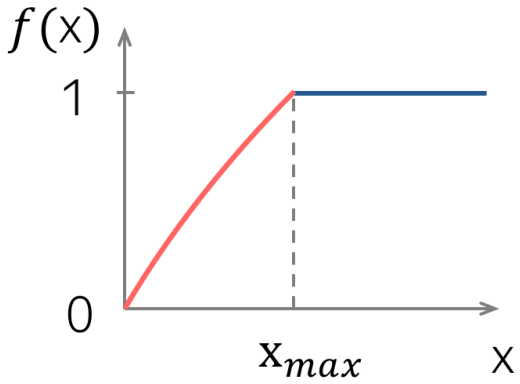
word vector

bias terms (also learned)

$$J(\theta) = \sum_{w,c \in V} f(\mathrm{N}(w, c)) \cdot (u_c^T v_w + b_c + \overline{b_w} - \log \mathrm{N}(w, c))^2$$

Weighting function to:

- penalize rare events
- not to over-weight frequent events



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

$\alpha = 0.75, x_{max} = 100$

Evaluation

Intrinsic:

- usually fast
- does not tell what is better in practice

Extrinsic:

- tells directly what is better in practice
- training several real-task models is expensive

Intrinsic Evaluation

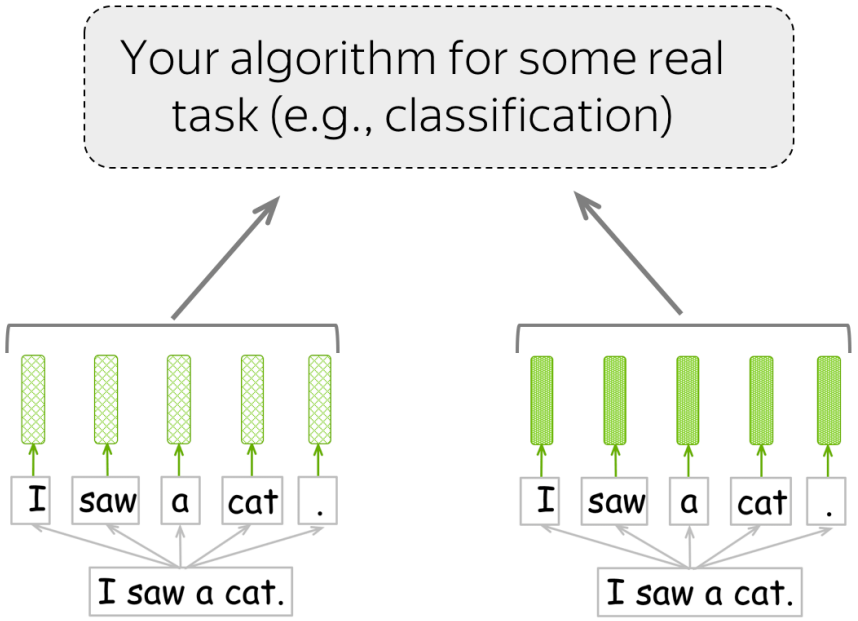
Основыается на внутрених св-ах.
Оценка эффективности эмбеденгов/модели в том насколько хорошо её внутрение свойства моделируют поведение изучаемых данных.

Например в случае Word Embeddings это может быть:

- Насколько близки по расстоянию вектора слов синонимов?
- Можно ли определить слово антоним?
- ...

Extrinsic Evaluation

Оценка на основание реальное задачи.



some embeddings

some other embeddings

Model with which embeddings performs better?

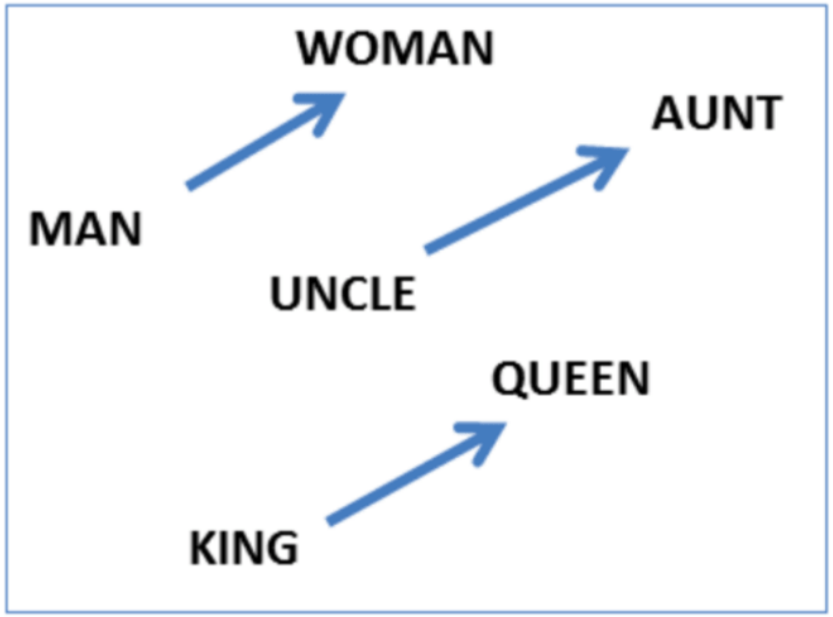
Train the same model several times: one model for each embedding set

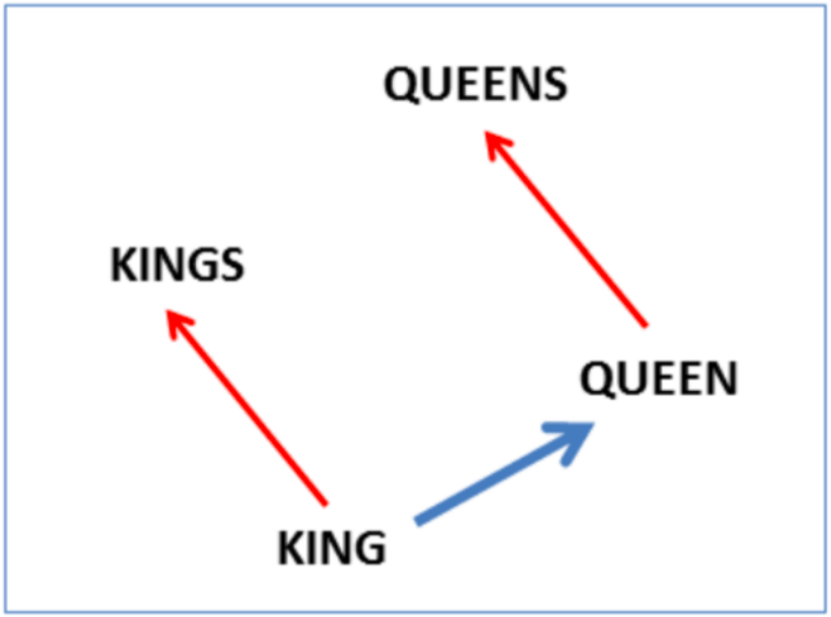
For the same dataset, you can get representations using different word embeddings

Interpretability

semantic: $v(\text{king}) - v(\text{man}) + v(\text{woman}) \approx v(\text{queen})$

syntactic: $v(\text{kings}) - v(\text{king}) + v(\text{queen}) \approx v(\text{queens})$





Similarity across languages

The recipe for building large dictionaries from small ones

Ingredients:

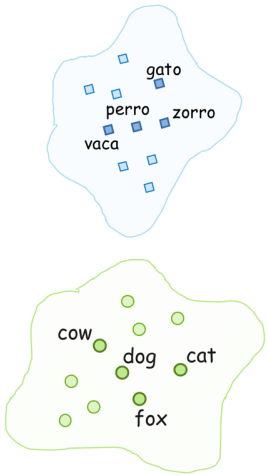
- corpus in one language (e.g., English)
- corpus in another language (e.g., Spanish)

- very small dictionary

  cat ↔ gato
  cow ↔ vaca
  dog ↔ perro
  fox ↔ zorro
  …

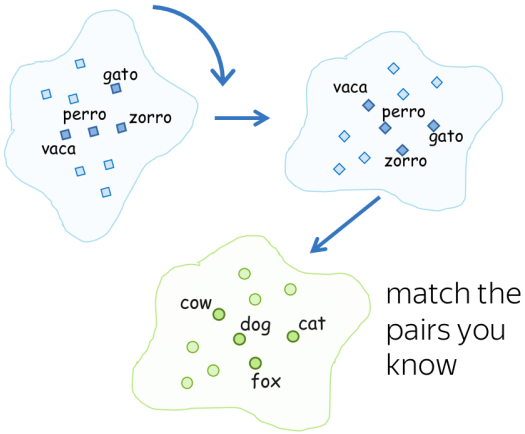
Step 1:

- train embeddings for each language



Step 2:

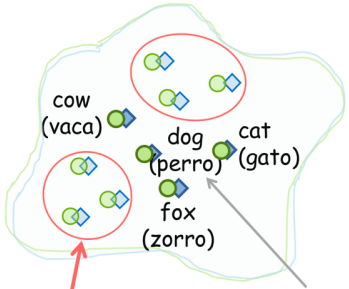
- linearly map one embeddings to the other to match words from the dictionary



match the pairs you know

Step 3:

- after matching the two spaces, get new pairs from the new matches



old translations – the ones you knew

new translations – you learned them!