Лекция 4: Эмбеддинги

Автор: Сергей Вячеславович Макрушин e-mail: SVMakrushin@fa.ru (mailto:SVMakrushin@fa.ru)

Финансовый университет, 2021 г.

При подготовке лекции использованы материалы:

- ...

v 0.3 01.04.21 v 0.5 06.04.21

In [1]:
```
# загружаем стиль для оформления презентации
from IPython.display import HTML
from urllib.request import urlopen
html = urlopen("file:./lec_v2.css")
HTML(html.read().decode('utf-8'))
```

Out[1]:

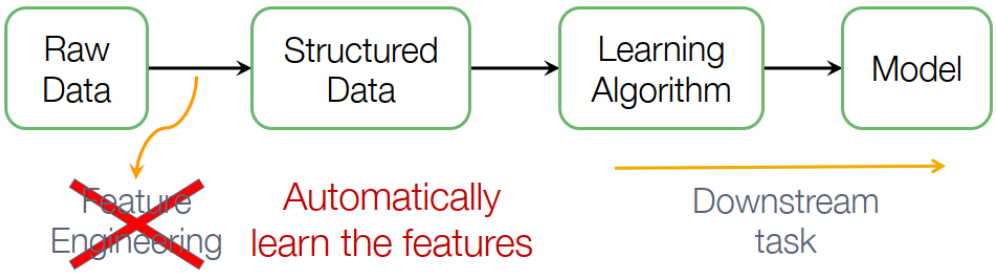
Введение

- к оглавлению

Мотивация для построения эмбеддинга

Эмбеддинг (embedding) ~ векторное представление

Проблема: Решение задачи машинного обучения (с учителем) требует конструирования признаков для каждой конкретной задачи!



Этапы решения задачи машинного обучения

- **upstream processing** - предварительная обработка (pre-processing)
- **downstream** - задачи решаемые после предварительной обработки
  - **Определение** (вариант 1): Downstream estimators - это общий термин, относящийся к алгоритмам оценки, используемым после этапа предварительной обработки данных.
  - **Определение** (вариант 2): Downstream tasks это то, что в данной области называют задачами обучения с учителем (supervised-learning), которые используют предобученные модели или компоненты.

Цель:

- Эффективное извлечение признаков (в задачи обучения без учителя) для задач машинного обучения на графах (сетях) без привязки к конкретной задаче.
- Для применения NN дискретные объекты (например слова) нужно представить в виде непрерывных, дифференциируемых обеъктов (чтобы потом можно было решать задачу

градиентного спуска при обучении).

Векторное представление и метрика

- к оглавлению

One-hot representation

Как можно представить значение объекта (слова и т.п.) для использования в компьютерных алгоритмах? Вариант решения: представить объекты (слова и т.п.) как отдельные one-hot vector.

One-hot представление - слово в словаре представляется в виде вектора, размер которого равен числу слов в словаре. При этом все элементы вектора, кроме одного, равны нулю, а элемент в позиции, соответствующей номеру слова в словаре, равен единице.

- В в векторе каждого слова единица появляется ровно по одному разу.
- На месте, где имеется единица у одного слова у всех других слов стоит 0.
- 2 слова из словаря в 15 слов в one-hot представлении:
  - motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
  - hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]
- One-hot представление может использоваться как для слов так и для других повторяющихся дискретных объектов в задачах машинного обучения, например, вершин графа.

Оценка one-hot представления:

- лучше чем использование последовательных номеров, т.к. **не несет очивидной семантики упорядоченности**
- **большая размерность**: длина векторов равна размеру словаря, в большинстве случаев (например, для слов) длина вкторов становится неприемлимо велика
- все вектора в one-hot представлении ортоганальны, т.е. кодирование предполагает, что **каждое слово одинаково (максимально) удалено ото всех других** (если использовать косинусную меру близости)

попытаться **закондировать похожесть между словами** в кодирующих их векторах и использовать одну из мер близости векторов для определения близости слов, которые они кодируют

Метрика

Метрика — функция на парах элементов множества, вводящая на нём **расстояние**, то есть, снабжающее его структурой метрического пространства.

Числовая функция $d : X \times X \to [0, \infty)$ является метрикой на множестве X, если

- $d(x, y) \geq 0$ - аксиома неотрицательности
- $d(x,\ y) = 0 \Leftrightarrow x = y$ - аксиома тождества
- $d(x,\ y) = d(y,\ x)$ - аксиома симметрии
- $d(x,\ z) \leqslant d(x,\ y) + d(y,\ z)$ - аксиома треугольника

Косинусная мера сходства

Мера сходства (similarity measure) - безразмерный показатель сходства сравниваемых объектов. Большинство коэффициентов нормированы и находятся в диапазоне от 0 (сходство отсутствует) до 1 (полное сходство)

Косинус угла θ_{AB} между двумя не нулевыми векторами $\mathbf{A}$ и $\mathbf{B}$ можно получить из скалярного призведения векторов: $\mathbf{A} \cdot \mathbf{B} = \|\mathbf{A}\| \, \|\mathbf{B}\| \cos \theta_{AB}$.

Тогда **косинусная мера сходства** (cosine similarity) между $\mathbf{A}$ и $\mathbf{B}$ равна:

$$cosine\_similarity(\mathbf{A}, \mathbf{B}) = \cos(\theta_{AB}) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

- косинусная мера сходства измеряет косинус угла между двумя ненулевыми векторами
- косинус 0 rad (0°) равен 1
  - $cosine\_similarity(\mathbf{A}, \mathbf{A}) = 1$
  - $cosine\_similarity(\mathbf{A}, \mathbf{B}) = 1 \nRightarrow \mathbf{A} = \mathbf{B}$
- косинус $\pi/2$ rad (90°) равен 0, косинусная мера сходства между ортоганальными векторами равна 0
- косинус π rad (180°) равен -1, косинусная мера сходства между противоположно направленными векторами равна 0
- если сравниваемые вектора $\mathbf{A}$ и $\mathbf{B}$ имеют только неотрицательные компоненеты то:
  - угол между ними $0 \leq \theta_{AB} \leq \pi/2$
  - мера сходства между ними $0 \leq cosine\_similarity(\mathbf{A}, \mathbf{B}) \leq 1$
  - в приложениях косинусная мера сходства часто применяется к векторам, имеющим только не отрицательные компоненты, поэтому принимается, что $0 \leq cosine\_similarity \leq 1$

Косинусная мера сходства не является метрикой

- Для косинусной меры сходства выполняется только аксиома симметрии ($d(x,\ y) = d(y,\ x)$)
- Если x вектора из неотрицательных компонент, то выполняется аксиома неотрицательности ($d(x, y) \geq 0$)
- Если рассматривать нормированные вектора из неотрицательных компонент то для функции $cosine\_distance = 1 - cosine\_similarity$ будут выполнятся все аксиомы, кроме аксиомы треугольника
- Для функции $angular\_distance = \frac{\cos^{-1}(cosine\_similarity)}{\pi}$ на множестве нормированных векторов (в т.ч. и с отрицательными компонентами) будут выполнятся все акисомы метрики
- **В многих приложениях нет необходимости заменять Косинусную меру сходства на метрику углового расстояния**

Представление слов

- к оглавлению

Дистрибутивная семантика

Дистрибутивная гипотеза - лингвистические единицы, встречающиеся в схожих контекстах, имеют близкие значения

- Контексты слова - это множестов слов которые встречаются рядом с рассматриваемом словом в тексте. Для получения котнтекстов можно использовать:
  - скользящие окна фиксированной ширины
  - рассматривать в качестве границ окон, определяющих контекст, границы предложений, абзацов

…government debt problems turning into ***banking*** *crises as happened in 2009…*
…saying that Europe needs unified ***banking*** *regulation to replace the hodgepodge…*
…India has just given its ***banking*** *system a shot in the arm…*

Пример: контекст слова banking

- Каждому слову (или понятию) из словаря присваивается свой контекстный вектор
- Множество векторов формирует векторное пространство слов
- Семантическое расстояние между словами (или понятиями) естественного языка, обычно вычисляется как косинусное расстояние между векторами вкторного пространства слов
- Для определения значения векторного представления слова (или понятия) используется множество его контекстов найденных в рассматриваемом текстовом корпусе

Матрица совместной встречаемости

Матрица совместной встречаемости(co-occurence matrix) - матрица в которой строки соответствуют определенным сущностям и столбцы соответствуют определенным сущностям (набор сущностей для строк и столбцов можтет отличаться, тогда матрица будет прямоугольной или набор сущностей может совпадать, тогда матрица будет квадратной) - элементы матрицы неотрицательные целые числа равные количеству раз когда сущность, которой соответствует строка, и сущность, котрой соответствует столбец, совместно встречались в общем контексте.

Для нашей задачи интересны два вида матриц совместной встречаемости:

- матрица *слова* x *документы* (прямоугольная матрица размерности: размер словаря на количество документов в корпусе)
- матрица *слова* x *слова* (квадратная матрица размерности: размер словаря на размер словаря)
  - контекстом является скользящее окно
  - собирается информация как о семантике, так и о синтаксисе (части речи и т.п.)

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

Пример матрицы совместной встречаемости слов в документе (term-document)

- Пример построен для симметричного (левый и правый контекст) окна ширины 1 (обычно используется ширина 5-10) для корпуса из трех предложений:
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.
- **Матрица совместной встречаемости является аналогом матрицы инцидентности (иногда и в явном виде рассматривается в этом смысле)**

Представление слов в виде векторов малой размерности

Проблемы с представлением слов в виде вектора матрицы совместной встречаемости:

- большой размер вектора
- квадратичный рост хранимого объема информации при увеличении словаря

- высокая разреженность матрицы совместной встречаемости
  - неэффективное хранение информации
  - низкая надежность использования меры сходства

Решение: построить представление слов малой размерности

- обчно вектора малой размерности для слов имеют размерность от 25 до 1000 компонент (словари могут содержать многие десятки тысяч слов)
- вектора плотные и хранят "самую ценную" информацию т.е. мера сходства для плотных векторов в среднем должна давать близкий результат к мере сходства для векторов совместной встречаемости
- векторное представление слов назвают: "word vectors", "word embeddings", "word representations"

Как уменьшить размерность векторов матрицы совместной встречаемости?

Существует несколько методов:

- **методы снижения размерности для матриц совместной встречаемости**
- **построения векторов малой размероности с помощью нейронных сетей**

Распредленное представление (Distributed representation)

Concept	Representation
Small Red Car	[1 0 0 0 0 0 0 0]
Large Blue SUV	[0 1 0 0 0 0 0 0]
Large Red SUV	[0 0 1 0 0 0 0 0]
Green Apple	[0 0 0 1 0 0 0 0]
Bumble Bee	[0 0 0 0 1 0 0 0]
Tall Building	[0 0 0 0 0 1 0 0]
Small Fish	[0 0 0 0 0 0 1 0]
Banana	[0 0 0 0 0 0 0 1]

Local Representation (One-hot representation)

* в Local Representation (в частности one-hot) – каждое измерение соответствует отдельному объекту

Concept	Representation
Small Red Car	[0.555 0.761 0.243 0.812]
Large Blue SUV	[0.773 0.309 0.289 0.835]
Large Red SUV	[0.766 0.780 0.294 0.834]
Green Apple	[0.153 0.022 0.654 0.513]
Bumble Bee	[0.045 0.219 0.488 0.647]
Tall Building	[0.955 0.085 0.900 0.773]
Small Fish	[0.118 0.192 0.432 0.618]
Banana	[0.184 0.232 0.671 0.589]

Distributed Representation

* в Distributed Representation – за измерениями не закреплен смысл, но расстояние между векторами несет смысловую нагрузку

Distributed representation describes the same data features across multiple scalable and interdependent layers. Each layer defines the information with the same level of accuracy, but adjusted for the level of scale. These layers are learned concurrently but in a non-linear fashion. This mimics human logic in a

neural network, since each concept can be accessed by more than one neuron firing and each neuron can represent more than one concept.

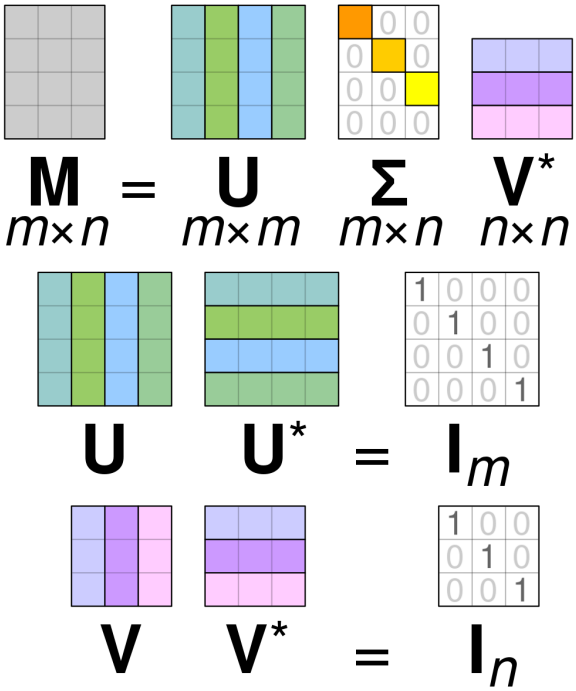
Распределенное представление описывает одни и те же функции данных на нескольких масштабируемых и взаимозависимых уровнях. Каждый слой определяет информацию с одинаковым уровнем точности, но с поправкой на уровень масштаба. Эти уровни изучаются одновременно, но нелинейным образом. Это имитирует человеческую логику в нейронной сети, поскольку к каждому понятию можно получить доступ с помощью более чем одного срабатывания нейрона, и каждый нейрон может представлять более одного понятия.

- Continuous values instead of discrete 1's and 0's.
- Each processing unit contributes to any and all concepts.
- The representations are dense (vs. localist representations which are sparse).
- Concepts are no longer localized in one unit (hence the "distributed" designation).
- We're able to represent a very large number of concepts with only 4 processing units (as opposed to being limited by n units to n concepts).
- We can learn new concepts without adding new units. All we need is a new configuration of values.
- **Most importantly, we are able to represent similarities better**: Large Red SUV [0.773 0.309 0.289 0.835] and Large Blue SUV [0.766 0.780 0.294 0.834] are much more similar to each other than they are to Small Fish [0.118 0.192 0.432 0.618].

Сингулярное разложение

Сингулярное разложение (Singular Value Decomposition, SVD) - разложение прямогуольной матрицы $\mathbf{M}$ вида: $\mathbf{M} = \mathbf{U\Sigma V}^T$.

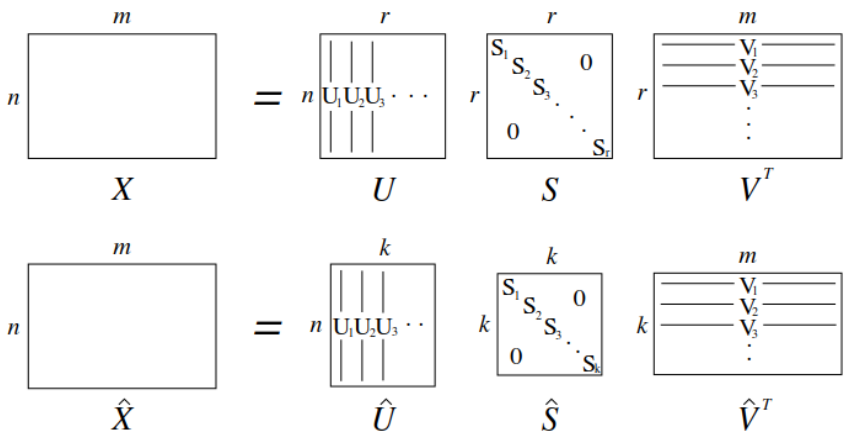
- Далее будем предполагать, что исходная матрица $\mathbf{M}$ является матрицей $m \times n$ над $\mathbb{R}$ (в общем случае сингулярное разложение описывается для матриц над $\mathbb{C}$, но для упрощения мы опустим этот случай). Тогда:
- $\mathbf{U}$ ортоганальная матрица ($\mathbf{UU}^T = \mathbf{U}^T\mathbf{U} = \mathbf{I}_m$) размера $m \times m$
- $\mathbf{V}$ ортоганальная матрица ($\mathbf{VV}^T = \mathbf{V}^T\mathbf{U} = \mathbf{I}_n$) размера $n \times n$
- $\mathbf{\Sigma}$ диагональная прямоугольная матрица размера $m \times n$ с неотрицательными действительными числами на главной диагонали
  - элементы σ_{ii}, лежащие на главной диагонали $\mathbf{\Sigma}$ - сингулярные числа
  - неотрицательное вещественное число σ называется **сингулярным числом** матрицы M тогда и только тогда, когда существуют два вектора единичной длины $u \in \mathbb{R}^m$ и $v \in \mathbb{R}^n$ такие, что:
    - $\mathbf{M}v = \sigma u$, и $\mathbf{M}^T u = \sigma v$
    - векторы u и v называются, соответственно, **левым сингулярным вектором** и **правым сингулярным вектором**, соответствующим сингулярному числу σ.
    - матрицы $\mathbf{U}$ и $\mathbf{V}$ состоят из левых и правых сингулярных векторов матрицы $\mathbf{M}$ соответственно



Визуализация умножений матриц в сингулярном разложении

Пусть r ($r \leq min(m, n)$) ранг (максимальное число линейно независимых строк (столбцов)) матрицы $\mathbf{M}$, тогда $\mathbf{\Sigma} = diag(\sigma_1, \dots, \sigma_r, 0, \dots, 0)$

- далее без ограничения общности будем считать что сингулярные значения в $\mathbf{\Sigma}$ представленны в порядке убывания, т.е. σ_1 - наибольшее сингулярное значение, а σ_r - наименьшее сингулярное значение.
- сингулярное разложение можно представить как произведение матриц измененных размерностей: $\mathbf{U}_r$ размерности $m \times r$, $\mathbf{\Sigma}_r = diag(\sigma_1, \dots, \sigma_r)$ размерности $r \times r$, $\mathbf{V}_r$ размерности $r \times n$: $\mathbf{M} = \mathbf{U}_r\mathbf{\Sigma}_r\mathbf{V}_r^T$



Пример матрицы совместной встречаемости слов в документе (term-document)

Задача: уменьшить размер матриц $\mathbf{U}$, $\mathbf{\Sigma}$, $\mathbf{V}$, снизив их ранг до $k < r$ (получим $\hat{\mathbf{U}}$, $\hat{\mathbf{\Sigma}}$, $\hat{\mathbf{V}}$ с размерностями $m \times k, k \times k, k \times n$ соответственно) и при этом получить произведение $\hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\hat{\mathbf{V}}^T = \hat{\mathbf{M}}$ максимально близкое к $\mathbf{M}$

- Качество приближения $\mathbf{M}$ с помощью $\hat{\mathbf{M}}$ определим с помощью нормы Фробениуса: $\|\mathbf{M} - \hat{\mathbf{M}}\|_\text{F} \to 0$
- **Норма Фробениуса** - $\|A\|_\text{F} = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2}$
- Теорема Эккарта — Янга показывает, что наилучшее приближение $\hat{\mathbf{M}}$ по норме Фробениуса получится с помощью $\hat{\mathbf{\Sigma}} = \mathbf{\Sigma}_k$ т.е. $\mathbf{\Sigma}_k = diag(\sigma_1, \dots, \sigma_k)$ диагональной матрицы из наибольших сингулярных значений.

Как уменьшить размерность векторов матрицы совместной встречаемости?

Использовать в качестве векторного представления слов сингулярные векторы матрицы заданного размера k

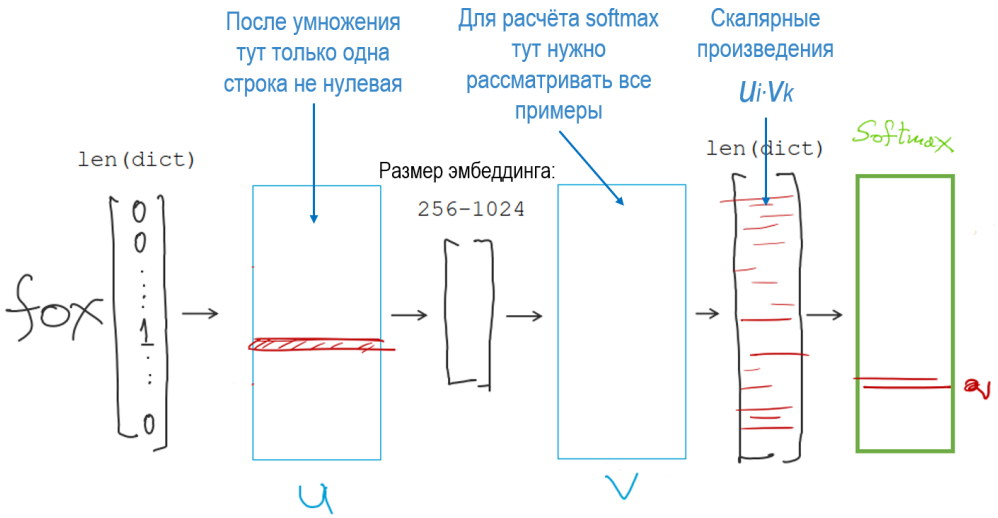
- Сложность выполнения сингулярного разложения $O(min(mn^2, m^2n))$ - это делает *невозможным использование сингулярного разложения* для получения представления слов в виде векторов малой размерности для больших матриц совместной встречаемости.
- существуют альтернативные **методы приближения матрицы** на основе ее **факторизации**, имеющие приемлемую сложность
- GloVe “Glove: Global Vectors for Word Representation” Pennington et al. (2014) and Levy and Goldberg (2014)

word2vec

- к оглавлению

Кейс word2vec

- Натренируем модель NN (что за модель? лучше, чтобы задача обучения без учителя!); и веса, которые соответствуют «1», будут тем самым вектором, который соответствует слову (векторным представлением слова)



Схематическое представление NN модели

Пример модели:

- Один скрытый слой (даже без нелинейности)!
- Обучающая выборка: корпус текстов + контексты слова
- Эмбеддинг: $w(fox) = u(fox) + v(fox)$ (альтернатива: конкатенация, но вектор будет в 2 раза длиннее)

- Базовый пример: максимизируем softmax (на самом деле не так)
- **Проблема:** знаменатель: для каждого примера нужно прогнать градиенты по всей матрице (правой)

$$p(o|\mathrm{i}) = \frac{e^{u_i \cdot v_o}}{\sum_k e^{u_i \cdot v_k}}$$

$$L = -\sum_j \ln p(o|i)$$

$$= -\sum_j \ln \frac{e^{u_i \cdot v_o}}{\sum_k e^{u_i \cdot v_k}}$$

word2vec

Идея word2vec состоит в том, чтобы напрямую обучать вектора слов малой размерности

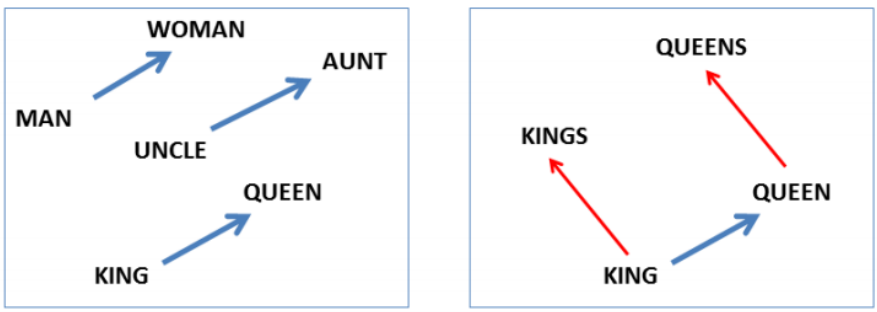
- A neural probabilistic language model (Bengio et al., 2003)
- word2vec (Mikolov et al. 2013) A recent, even simpler and faster model
- Faster and can easily incorporate a new sentence/document or add a word to the vocabulary

Для word2vec разработаны два основных алгоритма обучения:

- CBoW (англ. Continuous Bag of Words, «непрерывный мешок со словами», bag — мультимножество) предсказывает текущее слово, исходя из окружающего его контекста.
- Skip-gram - использует текущее слово, чтобы предугадывать окружающие его слова.

According to Mikolov:

- Skip-gram:
  - works well with small amount of the training data,
  - represents well even rare words or phrases.
- CBOW:
  - several times faster to train than the skip-gram,
  - slightly better accuracy for the frequent words



(Mikolov et al., NAACL HLT, 2013)

Векотры в word2vec

Эмбеднингги word2vec для русского языка: https://rusvectores.org/ru/ (https://rusvectores.org/ru/)

Модель CBOW

Модель CBOW учится как можно лучше по заданному контексту слова восстановить само слово.

- окно для предстказания может быть любым (не обязательно предсказывать следующее слово по предыдущим), обычно предсказывают центральное слово в окне по левому и правому контексту
- по сути модель CBOW это неглубокая нейронная сеть с одним скрытым слоем
- каждый вход сети - это вектор в one-hot представлении размерности $|V|$ (размер словаря)
- в CBOW $2m$ входов: $x^{(c-m)}, x^{(c-m+1)}, \dots, x^{(c-1)}, x^{(c+1)}, \dots, x^{(c+m-1)}, x^{(c+m)})$ (m - ширина окна) и 1 выход $y^{(c)}$
- $\mathcal{V} \in \mathbb{R}^{n \times |V|}$ матрица для преобразования входных слов во внутренний слой n - размерность внутреннего слоя
  - v_i: i-й столбец $\mathcal{V}$
- $\mathcal{U} \in \mathbb{R}^{|V| \times n}$ матрица для проеобразования значения внутреннего слоя в выходной слой n - размерность внутреннего слоя
  - u_i: i-я строка $\mathcal{V}$

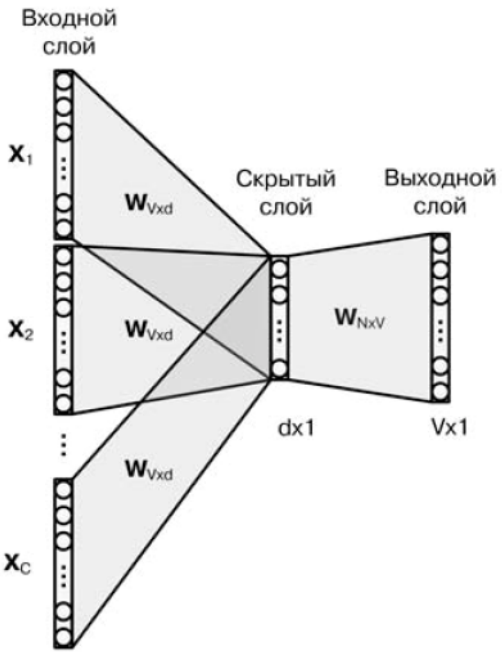


Схема модели CBOW

Алгоритм CBOW:

1. Из контекста слова $x^{(c)}$ получается $2m$ входных векторов в one-hot представлении: $x^{(c-m)}, x^{(c-m+1)}, \dots, x^{(c-1)}, x^{(c+1)}, \dots, x^{(c+m-1)}, x^{(c+m)})$ размерности $|V|$
2. Получаются $2m$ векторных представлений слов контекста: $v_{c-m} = \mathcal{V}x^{(c-m)}, v_{c-m+1} = \mathcal{V}x^{(c-m+1)}, \dots, v_{c+m} = \mathcal{V}x^{(c+m)})$ размерности n
3. Расчитывается среднее векторных представлений: $\hat{v} = (v_{c-m} + v_{c-m+1} + \dots + v_{c+m})/2m$ (значение для скрытого слоя)
4. Расчитвается вектор оценок выходного слоя $z = \mathcal{U}\hat{v}$ размерности $|V|$ (для всех слов словаря)
5. Вектор оценок преобразуется в вектор вероятностей для всех слов словаря: $\hat{y} = softmax(z)$
6. Центральное слово контекста $y^{(c)}$ one-hot вектор имеет 1 только в компоненте i_c, тогда модель предсказывает правильный ответ с вероятностью: $\hat{y}_{i_c}$

$$p(x^c|context(x^c);\theta) = \hat{y}_{i_c} = softmax(z)_{i_c} = \frac{\exp(u_{i_c}^T \hat{v})}{\sum_{j=1}^{|V|} \exp(u_j^T \hat{v})}$$

θ - все параметры модели: $\mathcal{V}$ и $\mathcal{U}$

$$\arg\max_{\boldsymbol{\theta}} \prod_{c \in Text} p(x^c|context(x^c);\theta) = \arg\max_{\boldsymbol{\theta}} \sum_{c \in Text} \ln p(x^c|context(x^c);\theta) = \arg\max_{\boldsymbol{\theta}} \sum_{c \in Text} \ln \frac{\exp}{\sum_{j=1}^{|V|}}$$

$$\max_{\boldsymbol{\theta}} \left(\sum_{c \in Text} u_{i_c}^T \hat{v} - \sum_{c \in Text} \ln \sum_{j=1}^{|V|} \exp(u_j^T \hat{v}) \right)$$

- на каждом шаге по $c \in Text$ используется градиентный спуск для улучшения параметров θ (векторных представлений $\mathcal{V}$ и $\mathcal{U}$)
- расчет на каждом шаге слагаемого $\ln \sum_{j=1}^{|V|} \exp(u_j^T \hat{v})$ чрезвыйчаной трудоемок: необходимо провести суммеирование по всему словарю!

Модель Skip-Gram

Модель Skip-Gram учится как можно лучше по заданному слову восстановить его контекст.

- окно для предстказания может быть любым (не обязательно предсказывать следующее слово по предыдущим), обычно предсказывают контекст по центральному слову в окне.
- вход сети - это один вектор $x^{(c)}$ в one-hot представлении размерности $|V|$ (размер словаря)
- в Skip-Gram $2m$ выходов: $y^{(c-m)}, y^{(c-m+1)}, \dots, y^{(c-1)}, y^{(c+1)}, \dots, y^{(c+m-1)}, y^{(c+m)})$ (m ширина окна)
- $\mathcal{V} \in \mathbb{R}^{n \times |V|}$ матрица для преобразования входного слова во внутренний слой n - размерность внутреннего слоя
  - v_i: i-й столбец $\mathcal{V}$
- $\mathcal{U} \in \mathbb{R}^{|V| \times n}$ матрица для проеобразования значения внутреннего слоя в выходной слой n - размерность внутреннего слоя
  - u_i: i-я строка $\mathcal{U}$

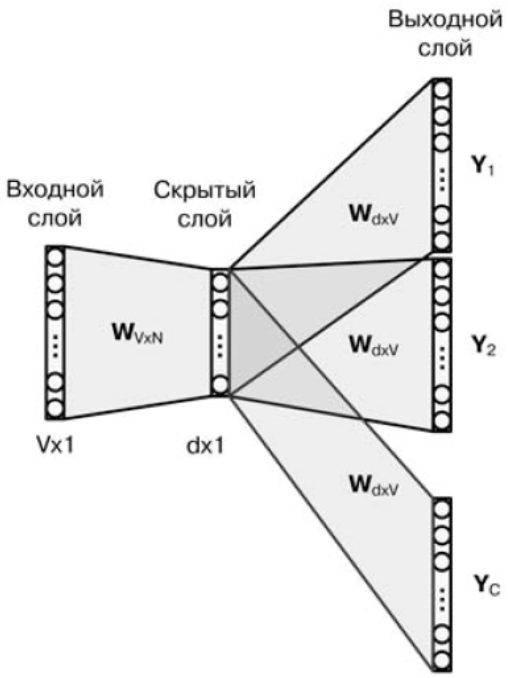


Схема модели Skip-Gram

Алгоритм Skip-Gram:

1. Имеется один входной вектор в one-hot представлении: $x^{(c)}$
2. Из него получается значение для скрытого слоя: $\hat{v} = v_c = \mathcal{V} x^{(c)}$
3. Расчитвается вектор оценок выходного слоя $z = \mathcal{U}\hat{v}$ размерности $|V|$ (для всех слов словаря)
4. Вектор оценок преобразуется в вектор вероятностей для всех слов словаря: $\hat{y} = softmax(z)$
5. Для всех выходов: $y^{(c-m)}, y^{(c-m+1)}, \dots, y^{(c+m)})$ для их компонент one-hot вектора $i_{c-m}, i_{c-m+1}, \dots, i_{c+m}$ извлекаются соответствующие компоненты вектора вероятностей: $\hat{y}_{i_{c-m}}, \hat{y}_{i_{c-m+1}}, \dots, \hat{y}_{i_{c+m}}$

$$p(context(x^c)|x^c;\theta) = \prod_{j=0, j\neq m}^{2m} \hat{y}_{i_{(c-m+j)}} = \prod_{j=0, j\neq m}^{2m} softmax(z)_{i_{(c-m+j)}}$$

$$\arg\max_{\boldsymbol{\theta}} p(context(x^c)|x^c;\theta) = \arg\max_{\boldsymbol{\theta}} \ln p(context(x^c)|x^c;\theta) = \sum_{j=0,j\neq m}^{2m} \ln softmax(z)_{i_{(c-m+j)}} = \arg m$$

$$\sum_{j=0,j\neq m}^{2m} \ln \frac{\exp(u_{i_{(c-m+j)}}^T v_c)}{\sum_{j=1}^{|V|} \exp(u_j^T v_c)} = \arg\max_{\boldsymbol{\theta}} \left(\sum_{j=0,j\neq m}^{2m} (u_{i_{(c-m+j)}}^T v_c) - 2m \ln \sum_{j=1}^{|V|} e \right.$$

- на каждом шаге по $c \in Text$ используется градиентный спуск для улучшения параметров θ (векторных представлений $\mathcal{V}$ и $\mathcal{U}$)
- расчет на каждом шаге слагаемого $\ln \sum_{j=1}^{|V|} \exp(u_j^T \hat{v})$ чрезвыйчаной трудоемок: необходимо провести суммирование по всему словарю!
- каждое слово имеет 2 векторых представления v_i и u_i упростить модель за счет использования одной матриц весов (т.е. $\mathcal{U} = \mathcal{V}^T$ не получится т.к.:
  - обычно слова редко оказываются в своем контексте (например, слово "футбол" редко встречается в ближайшей окрестности слова "футбол"
  - модель будет минимизировать вероятнсть $p(w|w)$
  - модель будет минимизировать произведение $u_i^T v_i$
  - если $v_i = u_i$ это будет приводит к минимазции нормы v_i для всех i, что очень нежелательно!

Негативное сэмплирование

Проблема: расчет на каждом шаге слагаемого $\sum_{j=1}^{|V|} \exp(u_j^T \hat{v})$ чрезвыйчаной трудоемок: необходимо провести суммеирование по всему словарю!

Идея: вместо того чтобы считать полную сумму $\sum_{j=1}^{|V|} \exp(u_j^T \hat{v})$ выберем несколько ее элементов случайным образом в качестве отрицательных примеров и обновим только их, то есть заменим сумму по всему словарю на гораздо более маленькую сумму $\sum_{j\in D} \exp(u_j^T \hat{v})$, где D выбранное небольшое подмножество отрицательных примеров.

- На практике для примеров берутся слова из распределения с вероятностью равной частоте в исходном словаре

Базовая реализация CBOW

- к оглавлению

In [1]:
```
# https://github.com/jojonki/word2vec-pytorch/blob/master/word2vec.ipynb
# see http://pytorch.org/tutorials/beginner/nlp/word_embeddings_tutorial.html
import torch
from torch.autograd import Variable
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

torch.manual_seed(1)
```

Out[1]:
```
<torch._C.Generator at 0x1e7f85305f0>
```

In [2]:
```
CONTEXT_SIZE = 2  # 2 words to the left, 2 to the right
text = """We are about to study the idea of a computational process.
Computational processes are abstract beings that inhabit computers.
As they evolve, processes manipulate other abstract things called data.
The evolution of a process is directed by a pattern of rules
called a program. People create programs to direct processes. In effect,
we conjure the spirits of the computer with our spells.""".split()

split_ind = (int)(len(text) * 0.8)

# By deriving a set from `raw_text`, we deduplicate the array
vocab = set(text)
vocab_size = len(vocab)
print('vocab_size:', vocab_size)

w2i = {w: i for i, w in enumerate(vocab)}
i2w = {i: w for i, w in enumerate(vocab)}
```

```
vocab_size: 49
```

In [4]:
```
# context window size is two
def create_cbow_dataset(text):
    data = []
    for i in range(2, len(text) - 2):
        context = [text[i - 2], text[i - 1],
                   text[i + 1], text[i + 2]]
        target = text[i]
        data.append((context, target))
    return data

# def create_skipgram_dataset(text):
#     import random
#     data = []
#     for i in range(2, len(text) - 2):
#         data.append((text[i], text[i-2], 1))
#         data.append((text[i], text[i-1], 1))
#         data.append((text[i], text[i+1], 1))
#         data.append((text[i], text[i+2], 1))
#         # negative sampling
#         for _ in range(4):
#             if random.random() < 0.5 or i >= len(text) - 3:
#                 rand_id = random.randint(0, i-1)
#             else:
#                 rand_id = random.randint(i+3, len(text)-1)
#             data.append((text[i], text[rand_id], 0))
#     return data

cbow_train = create_cbow_dataset(text)
# skipgram_train = create_skipgram_dataset(text)
print('cbow sample', cbow_train[0])
# print('skipgram sample', skipgram_train[0])
```

```
cbow sample (['We', 'are', 'to', 'study'], 'about')
```

In [5]:
```
class CBOW(nn.Module):
    def __init__(self, vocab_size, embd_size, context_size, hidden_size):
        super(CBOW, self).__init__()
        self.embeddings = nn.Embedding(vocab_size, embd_size)
        self.linear1 = nn.Linear(2*context_size*embd_size, hidden_size)
        self.linear2 = nn.Linear(hidden_size, vocab_size)

    def forward(self, inputs):
        embedded = self.embeddings(inputs).view((1, -1))
        hid = F.relu(self.linear1(embedded))
        out = self.linear2(hid)
        log_probs = F.log_softmax(out)
        return log_probs
```

`torch.nn.Embedding(num_embeddings, embedding_dim, padding_idx=None, max_norm=None, norm_type=2.0, scale_grad_by_freq=False, sparse=False, _weight=None)`

A simple lookup table that stores embeddings of a fixed dictionary and size.

This module is often used to store word embeddings and retrieve them using indices. The input to the module is a list of indices, and the output is the corresponding word embeddings.

Parameters

- num_embeddings (int) – size of the dictionary of embeddings
- embedding_dim (int) – the size of each embedding vector
- и т.д. (см.: https://pytorch.org/docs/stable/generated/torch.nn.Embedding.html (https://pytorch.org/docs/stable/generated/torch.nn.Embedding.html))

In [6]:
```
# class SkipGram(nn.Module):
#     def __init__(self, vocab_size, embd_size):
#         super(SkipGram, self).__init__()
#         self.embeddings = nn.Embedding(vocab_size, embd_size)

#     def forward(self, focus, context):
#         embed_focus = self.embeddings(focus).view((1, -1))
#         embed_ctx = self.embeddings(context).view((1, -1))
#         score = torch.mm(embed_focus, torch.t(embed_ctx))
#         log_probs = F.logsigmoid(score)

#         return log_probs
```

In [7]:
```
embd_size = 100
learning_rate = 0.001
n_epoch = 30

def train_cbow():
    hidden_size = 64
    losses = []
    loss_fn = nn.NLLLoss()
    model = CBOW(vocab_size, embd_size, CONTEXT_SIZE, hidden_size)
    print(model)
    optimizer = optim.SGD(model.parameters(), lr=learning_rate)

    for epoch in range(n_epoch):
        total_loss = .0
        for context, target in cbow_train:
            ctx_idxs = [w2i[w] for w in context]
            ctx_var = Variable(torch.LongTensor(ctx_idxs))

            model.zero_grad()
            log_probs = model(ctx_var)

            loss = loss_fn(log_probs, Variable(torch.LongTensor([w2i[target]])))

            loss.backward()
            optimizer.step()

#             print(f'!!! {loss.data}')
            total_loss += loss.data.item()
        losses.append(total_loss)
    return model, losses

cbow_model, cbow_losses = train_cbow()
```

```
CBOW(
  (embeddings): Embedding(49, 100)
  (linear1): Linear(in_features=400, out_features=64, bias=True)
  (linear2): Linear(in_features=64, out_features=49, bias=True)
)

C:\ProgramData\Anaconda3\envs\pyTorch_1_6\lib\site-packages\ipykernel_launcher.py:12:
UserWarning: Implicit dimension choice for log_softmax has been deprecated. Change the
call to include dim=X as an argument.
  if sys.path[0] == '':
```

In [53]:
```
# def train_skipgram():
#     losses = []
#     loss_fn = nn.MSELoss()
#     model = SkipGram(vocab_size, embd_size)
#     print(model)
#     optimizer = optim.SGD(model.parameters(), lr=learning_rate)

#     for epoch in range(n_epoch):
#         total_loss = .0
#         for in_w, out_w, target in skipgram_train:
#             in_w_var = Variable(torch.LongTensor([w2i[in_w]]))
#             out_w_var = Variable(torch.LongTensor([w2i[out_w]]))

#             model.zero_grad()
#             log_probs = model(in_w_var, out_w_var)
#             loss = loss_fn(log_probs[0], Variable(torch.Tensor([target])))

#             loss.backward()
#             optimizer.step()

#             total_loss += loss.data[0]
#         losses.append(total_loss)
#     return model, losses

# cbow_model, cbow_losses = train_cbow()
# sg_model, sg_losses = train_skipgram()
```

In [8]:
```
# test
# You have to use other dataset for test, but in this case I use training data because t
def test_cbow(test_data, model):
    print('====Test CBOW===')
    correct_ct = 0
    for ctx, target in test_data:
        ctx_idxs = [w2i[w] for w in ctx]
        ctx_var = Variable(torch.LongTensor(ctx_idxs))

        model.zero_grad()
        log_probs = model(ctx_var)
        _, predicted = torch.max(log_probs.data, 1)
        predicted_word = i2w[predicted[0]]
        print('predicted:', predicted_word)
        print('label    :', target)
        if predicted_word == target:
            correct_ct += 1

    print('Accuracy: {:.1f}% ({:d}/{:d})'.format(correct_ct/len(test_data)*100, correct_
```

In [9]:
```
# def test_skipgram(test_data, model):
#     print('====Test SkipGram===')
#     correct_ct = 0
#     for in_w, out_w, target in test_data:
#         in_w_var = Variable(torch.LongTensor([w2i[in_w]]))
#         out_w_var = Variable(torch.LongTensor([w2i[out_w]]))

#         model.zero_grad()
#         log_probs = model(in_w_var, out_w_var)
#         _, predicted = torch.max(log_probs.data, 1)
#         predicted = predicted[0]
#         if predicted == target:
#             correct_ct += 1

#     print('Accuracy: {:.1f}% ({:d}/{:d})'.format(correct_ct/len(test_data)*100, correc

# test_cbow(cbow_train, cbow_model)
# print('------')
# test_skipgram(skipgram_train, sg_model)
```

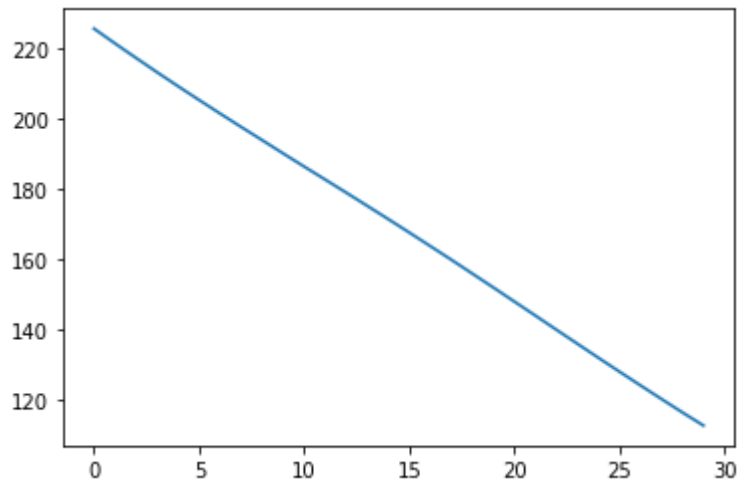
In [56]:
```
# test
# You have to use other dataset for test, but in this case I use training data because t
def test_cbow(test_data, model):
    print('====Test CBOW===')
    correct_ct = 0
    for ctx, target in test_data:
        ctx_idxs = [w2i[w] for w in ctx]
        ctx_var = Variable(torch.LongTensor(ctx_idxs))

        model.zero_grad()
        log_probs = model(ctx_var)
        _, predicted = torch.max(log_probs.data, 1)
        predicted_word = i2w[predicted[0]]
        print('predicted:', predicted_word)
        print('label    :', target)
        if predicted_word == target:
            correct_ct += 1

    print('Accuracy: {:.1f}% ({:d}/{:d})'.format(correct_ct/len(test_data)*100, correct_

cbow_model, cbow_losses = train_cbow()
```

```
CBOW(
  (embeddings): Embedding(49, 100)
  (linear1): Linear(in_features=400, out_features=64, bias=True)
  (linear2): Linear(in_features=64, out_features=49, bias=True)
)

C:\Users\alpha\.conda\envs\pyTorch_1_5v2\lib\site-packages\ipykernel_launcher.py:12: U
serWarning: Implicit dimension choice for log_softmax has been deprecated. Change the
call to include dim=X as an argument.
  if sys.path[0] == '':
```

In [57]:
```
# def test_skipgram(test_data, model):
#     print('====Test SkipGram===')
#     correct_ct = 0
#     for in_w, out_w, target in test_data:
#         in_w_var = Variable(torch.LongTensor([w2i[in_w]]))
#         out_w_var = Variable(torch.LongTensor([w2i[out_w]]))

#         model.zero_grad()
#         log_probs = model(in_w_var, out_w_var)
#         _, predicted = torch.max(log_probs.data, 1)
#         predicted = predicted[0]
#         if predicted == target:
#             correct_ct += 1

#     print('Accuracy: {:.1f}% ({:d}/{:d})'.format(correct_ct/len(test_data)*100, correc

# sg_model, sg_losses = train_skipgram()
```

In [10]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np

def showPlot(points, title):
    plt.figure()
    fig, ax = plt.subplots()
    plt.plot(points)

showPlot(cbow_losses, 'CBOW Losses')
# showPlot(sg_losses, 'SkipGram Losses')
```

```
<Figure size 432x288 with 0 Axes>
```



In [63]:
```
# processes manipulate other
target = 'manipulate'
w2i[target]
```

Out[63]:
```
41
```

In [64]:
```
ctx[10]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-64-4501a711378a> in <module>
----> 1 ctx[10]

NameError: name 'ctx' is not defined
```

In [ ]:
```
    for ctx, target in test_data:
        ctx_idxs = [w2i[w] for w in ctx]
        ctx_var = Variable(torch.LongTensor(ctx_idxs))

        model.zero_grad()
        log_probs = model(ctx_var)
        _, predicted = torch.max(log_probs.data, 1)
        predicted_word = i2w[predicted[0]]
        print('predicted:', predicted_word)
        print('label    :', target)
        if predicted_word == target:
```

In [12]:
```
ctx, target = cbow_train[14]
# print(ctx, target)
print(ctx, target)
ctx_idxs = [w2i[w] for w in ctx]
ctx_var = Variable(torch.LongTensor(ctx_idxs))
log_probs = cbow_model(ctx_var)

argsrt = torch.argsort(log_probs)
print(argsrt)

print(i2w[argsrt[0][-1].item()])
print(i2w[argsrt[0][-2].item()])
print(i2w[argsrt[0][-3].item()])
```

```
['abstract', 'beings', 'inhabit', 'computers.'] that

C:\ProgramData\Anaconda3\envs\pyTorch_1_6\lib\site-packages\ipykernel_launcher.py:12:
UserWarning: Implicit dimension choice for log_softmax has been deprecated. Change the
call to include dim=X as an argument.
  if sys.path[0] == '':

tensor([[22, 34,  2, 44, 47,  4,  5, 46, 43, 18, 45, 13, 21, 41, 38,  6, 37, 31,
         20, 40,  8,  1, 35, 33, 48, 12, 14, 30, 42, 17,  0, 16, 11, 26, 32, 29,
          7, 39, 28, 10, 25, 23,  9, 24, 19, 15,  3, 27, 36]])
computational
beings
that
```

In [72]:
```
i2w
```

Out[72]:
```
{0: 'they',
 1: 'create',
 2: 'abstract',
 3: 'idea',
 4: 'In',
 5: 'to',
 6: 'evolve,',
 7: 'we',
 8: 'are',
 9: 'study',
 10: 'data.',
 11: 'computer',
 12: 'of',
 13: 'effect,',
 14: 'is',
 15: 'directed',
 16: 'People',
 17: 'beings',
 18: 'We',
 19: 'the',
 20: 'process',
 21: 'processes.',
 22: 'called',
 23: 'pattern',
 24: 'things',
 25: 'with',
 26: 'As',
 27: 'that',
 28: 'about',
 29: 'other',
 30: 'The',
 31: 'programs',
 32: 'by',
 33: 'program.',
 34: 'direct',
 35: 'a',
 36: 'computers.',
 37: 'rules',
 38: 'conjure',
 39: 'spells.',
 40: 'processes',
 41: 'manipulate',
 42: 'spirits',
 43: 'process.',
 44: 'our',
 45: 'evolution',
 46: 'inhabit',
 47: 'computational',
 48: 'Computational'}
```

In [65]:
```
ctx, target = test_data[10]
ctx, target
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-65-21fbc5ffb2a0> in <module>
----> 1 ctx, target = test_data[10]
      2 ctx, target

NameError: name 'test_data' is not defined
```

In [ ]:

```
loss = loss_fn(log_probs, Variable(torch.LongTensor([w2i[target]])))
```

In [ ]:

In [ ]:

In [ ]:

Type *Markdown* and LaTeX: α^2

Type *Markdown* and LaTeX: α^2

In [ ]:

```
use gradient descent to update all relevant word vectors uc and
vj.



* скрытый слой сети - это фактически и есть матрица $W$ векторных представлений слов; n

the sentence represented by one-hot word vectors. The
input one hot vectors or context we will represent with an x(c). And
the output as y(c) and in the CBOW model, since we only have one
output,

• при вычислении выхода скрытого слоя мы берем просто среднее всех вход-
ных векторов; такая простота модели важна для того, чтобы в результате

предсказание
делается моделью, очень похожей на нейронную сеть; вообще, по сути word2vec –
это нейронная сеть, но неглубокая нейронная сеть, с одним скрытым уровнем


окна теперь мы
можем выбирать как захотим: нам не обязательно предсказывать следующее слово
по предыдущим, как в языковой модели, а можно, например, попытаться предска-
зать центральное слово в окне по левому и правому контексту
```

\Sigma — матрица размера m \times n с неотрицательными элементами, у которой элементы, лежащие на главной диагонали — это сингулярные числа (а все элементы, не лежащие на главной диагонали, являются нулевыми)

\mathbf{M} = \mathbf{U} \boldsymbol{\Sigma} \mathbf{V}^*

of cooccurrence matrix X.

Suppose {{math|'''M'''}} is a {{math|"m" × "n"}} [[matrix (mathematics)|matrix]] whose entries come from the [[field (mathematics)|field]] {{mvar|K}}, which is either the field of [[real number]]s or the field of [[complex number]]s. Then there exists a factorization, called a 'singular value decomposition' of {{math|'''M'''}}, of the form

: \mathbf{M} = \mathbf{U} \boldsymbol{\Sigma} \mathbf{V}^*

where

- {{math|"'U'"}} is an {{math|"m" × "m"}} [[unitary matrix]] over {{mvar|K}} (if {{math|"K" {{=}} }}\mathbb{R}, unitary matrices are [[orthogonal matrix|orthogonal matrices]]),
- {{math|"'Σ'"}} is a [[rectangular diagonal matrix|diagonal]] {{math|"m" × "n"}} matrix with non-negative real numbers on the diagonal,
- {{math|"'V*'"}} is an {{math|"n" × "n"}} [[unitary matrix]] over {{mvar|K}}, and {{math|"'V'"*}} is the [[conjugate transpose]] of {{math|"'V'"}}.

The diagonal entries {{mvar|σᵢ}} of {{math|"'Σ'"}} are known as the '''[[singular value]]s''' of {{math|"'M'"}}. A common convention is to list the singular values in descending order. In this case, the diagonal matrix, {{math|"'Σ'"}}, is uniquely determined by {{math|"'M'"}} (though not the matrices {{math|"'U'"}} and {{math|"'V'"}} if {{math|"'M'"}} is not square, see below).

In [ ]:

Косинусная мера сходства

Мера сходства (similarity measure) - безразмерный показатель сходства сравниваемых объектов.

Большинство коэффициентов нормированы и находятся в диапазоне от 0 (сходство отсутствует) до 1 (полное сходство)

Косинус угла θ_{AB} между двумя не нулевыми векторами $\mathbf{A}$ и $\mathbf{B}$ можно получить из скалярного призведения векторов: $\mathbf{A} \cdot \mathbf{B} = \|\mathbf{A}\| \, \|\mathbf{B}\| \cos \theta_{AB}$.

Тогда **косинусная мера сходства**(cosine similarity) между $\mathbf{A}$ и $\mathbf{B}$ равна:

$$cosine\_similarity(\mathbf{A}, \mathbf{B}) = \cos(\theta_{AB}) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

- косинусная мера сходства измеряет косинус угла между двумя ненулевыми векторами
- косинус 0 rad (0°) равен 1
  - $cosine\_similarity(\mathbf{A}, \mathbf{A}) = 1$
  - $cosine\_similarity(\mathbf{A}, \mathbf{B}) = 1 \nRightarrow \mathbf{A} = \mathbf{B}$
- косинус $\pi/2$ rad (90°) равен 0, косинусная мера сходства между ортоганальными векторами равна 0
- косинус π rad (180°) равен -1, косинусная мера сходства между противоположно направленными векторами равна 0
- если сравниваемые вектора $\mathbf{A}$ и $\mathbf{B}$ имеют только неотрицательные компоненеты то:
  - угол между ними $0 \leq \theta_{AB} \leq \pi/2$
  - мера сходства между ними $0 \leq cosine\_similarity(\mathbf{A}, \mathbf{B}) \leq 1$
  - в приложениях косинусная мера сходства часто применяется к векторам, имеющим только не отрицательные компоненты, поэтому принимается, что $0 \leq cosine\_similarity \leq 1$

Косинусная мера сходства применятся для:

- распознавании образов
- поисковых системах
- сравнительной лингвистике
- биоинформатике, хемоинформатике
- при сравнении строк
- в биологии для количественного определения степени сходства биологических объектов

Необходимо отметить, что косинусная мера сходства не является метрикой.

Метрическое пространство есть [[Пара (математика)#Упорядоченная пара|пара]] (X,;d), где X —

Числовая функция $d : X \times X \to [0, \infty)$ является метрикой на множестве X, если

- $d(x, y) \geq 0$ - аксиома не отрицательности
- $d(x,\ y) = 0 \Leftrightarrow x = y$ - аксиома тождества
- $d(x,\ y) = d(y,\ x)$ - аксиома симметрии
- $d(x,\ z) \leqslant d(x,\ y) + d(y,\ z)$ - аксиома треугольника
- Для косинусной меры сходства выполняется только аксиома симметрии.
- Если x вектора из неотрицательных компонент, то выполняется аксиома не отрицательности
- Если рассматривать вектора нормированные вектора из неотрицательных компонент то для функции $cosine\_distance = 1 - cosine\_similarity$ будут выполнятся все аксиомы, кроме аксиомы треугольника
- Для функции $angular\_distance = \frac{\cos^{-1}(cosine\_similarity)}{\pi}$ на множестве нормированных векторов (в т.ч. и с отрицательными компонентами) будут выполнятся все акисомы метрики
- **В многих приложениях нет необходимости заменять Косинусную меру сходства на метрику углового расстояния**

Векторное представление (word embedding) - общее название для различных подходов к моделированию языка и обучению представлений в обработке естественного языка, направленных на сопоставление словам (иногда и другим единицам текста) из некоторого словаря векторов из n мерного простарнства $\mathbb{R}^n$, где n значительно меньше количества слов в словаре.

Использование этого подхода базируется на **дистрибутивной гипотезе**: лингвистические единицы, встречающиеся в схожих контекстах, имеют близкие значения.

Существует несколько методов для построения такого сопоставления. Так, используют нейронные сети[1] , методы снижения размерности в применении к матрицам совместных упоминаний слов (word co-occurrence matrices)[2] и явные представления, обучающиеся на контекстах упоминаний слов (explicit representations)[3].

Продемонстрировано[кем?], что векторные представления слов и фраз способны значительно улучшить качество работы некоторых методов автоматической обработки естественного языка (например, синтаксический анализ[4] и анализ тональности[5]).

https://rusvectores.org/ru/# (https://rusvectores.org/ru/#)

Векторное представление — общее название для различных подходов к моделированию языка и обучению представлений в обработке естественного языка, направленных на сопоставление словам (и, возможно, фразам) из некоторого словаря векторов из {\displaystyle \mathbb {R} ^{n}} \R^{n} для {\displaystyle n} n, значительно меньшего количества слов в словаре. Теоретической базой для векторных представлений является дистрибутивная семантика.

Существует несколько методов для построения такого сопоставления. Так, используют нейронные сети[1] , методы снижения размерности в применении к матрицам совместных упоминаний слов (word co-occurrence matrices)[2] и явные представления, обучающиеся на контекстах упоминаний слов (explicit representations)[3].

Продемонстрировано[кем?], что векторные представления слов и фраз способны значительно улучшить качество работы некоторых методов автоматической обработки естественного языка (например, синтаксический анализ[4] и анализ тональности[5]).

екторного представления слов (word embeddings). В отличие от векторов, полученных прямым кодированием, — би- нарных, разреженных (почти полностью состоящих из нулей) и с большой раз- мерностью (их размерность совпадает с количеством слов в словаре) — векторные представления слов являются малоразмерными векторами вещественных чисел (то есть плотными векторами, в

противоположность разреженным), как показано на рис. 6.2. В отличие от векторов, полученных прямым кодированием, векторные представления слов конструируются из данных. При работе с огромными словаря- ми размерность векторов слов нередко может достигать 256,

In [ ]:

In [ ]:

next *qs line*
next an line
next an line
next df line
next ex line
next pl line
next mn line
next plmn line
next hn line

In [ ]: