

Эмбеддинги

Елена Кантоностова

Недостатки мешка слов

Из курса классического ML вам известны простые способы перевода текстов в векторы чисел (векторизация) - это Bag of words (мешок слов) и tf-idf.

У этих способов, не смотря на прекрасную простоту, есть пара недостатков:

- Большое число признаков в результате векторизации (а также разреженность матрицы признаков) - все это приводит к огромным временным затратам на обучение моделей, а также нередко и к переобучению
- Похожие слова кодируются совершенно по-разному, то есть эти кодировки не сохраняют семантический смысл слов - и это для большинства задач NLP критический недостаток.

Эти недостатки существенные, поэтому нужны другие, более продвинутые способы векторизации текстов, которые сохраняют семантический смысл слов, а также позволяют получить достаточно короткие плотные (не разреженные) числовые векторы.

Идея word2vec

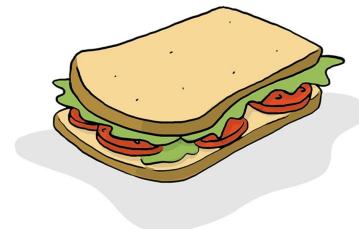
Алгоритм Word2Vec придумал Томаш Миколов в 2013 году, разработчик из Microsoft, Google и Facebook (в разные годы).

Идея алгоритма состоит в том, что мы будем обучать такие векторы слов, чтобы **слова, встречающиеся в похожих контекстах, имели близкие друг к другу векторы.**

Пример:

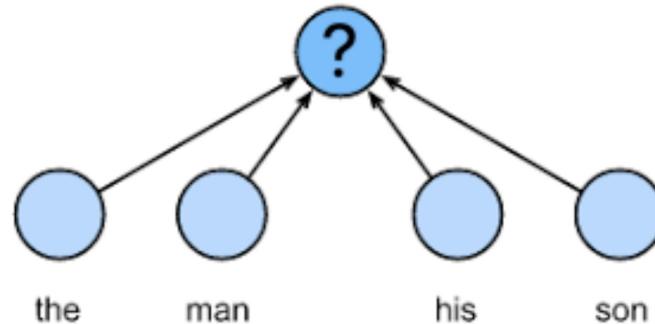
Фраза 1: "Сегодня утром я ел бутерброд с ветчиной и сыром."

Фраза 2: "Сегодня утром я ел сэндвич с ветчиной и сыром."



Как искать word2vec-векторы?

Будем решать вспомогательную задачу: **научим нейронную сеть по контексту предсказывать слово, стоящее внутри контекста.**



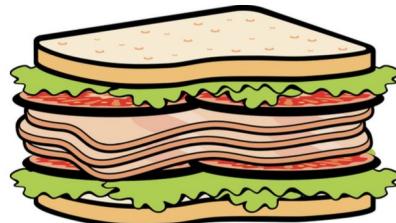
Ответ может быть, например, таким (с вероятностями):

- loves - 0.7
- needs - 0.25
- holds - 0.05

Почему именно такая задача?

Мы работаем в предположении, что **слова, встречающиеся в похожих контекстах, похожи!**

Например, слова бутерброд и сэндвич часто встречаются в одинаковых контекстах - значит, модель присвоит им похожие векторы.



$$s = (0.51, 0.7, 0.82, \dots)$$



$$b = (0.45, 0.72, 0.83, \dots)$$

Где взять данные для обучения?

С помощью скользящего окна движемся по тексту:

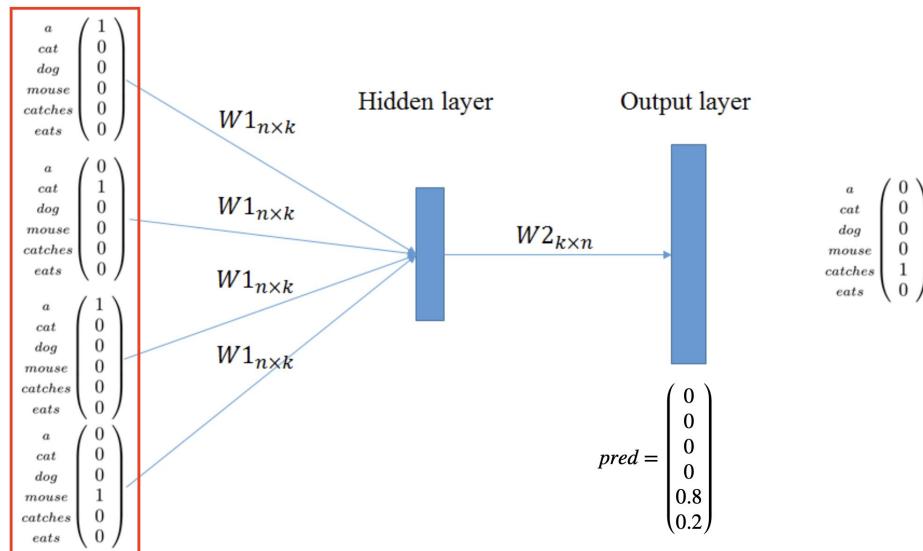
- объекты - контекст (окружение центрального слова в окне)
- ответы - центральное слово



Как устроен алгоритм?

Word2Vec - это полносвязная нейронная сеть с одним скрытым слоем.

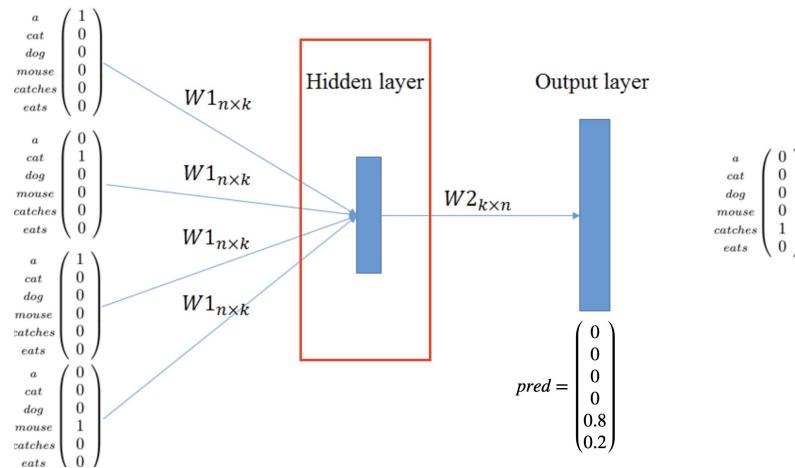
- На вход сети подаются слова контекста, закодированные при помощи OneHot-кодирования
- На выходе мы получаем вектор размерности количества слов в словаре, где на i -й позиции стоит вероятность того, что внутри данного контекста стоит i -е слово из словаря



Как устроен алгоритм?

Какие функции активации используются:

- На скрытом слое НЕТ функции активации!
- На выходном слое функция активации - softmax - классическая функция активации в задачах многоклассовой классификации, а у нас именно такая задача.



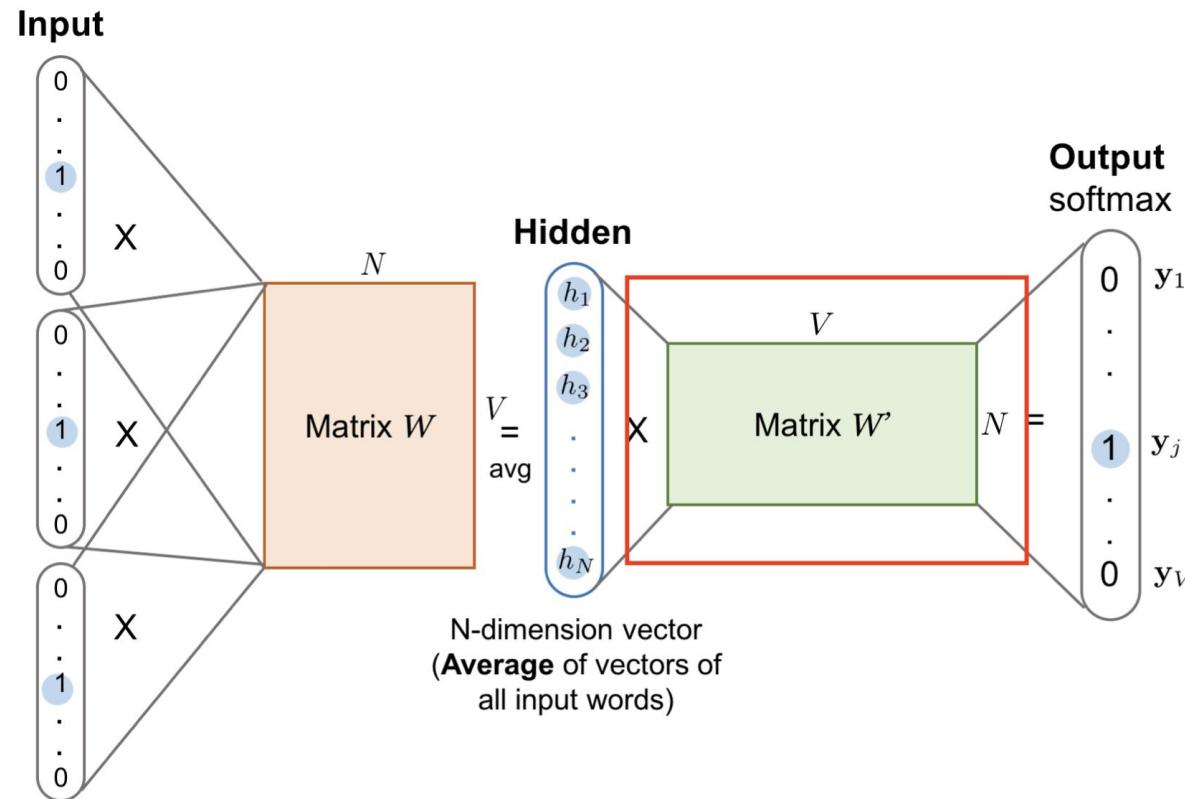
Как устроен алгоритм?

Какая функция потерь?

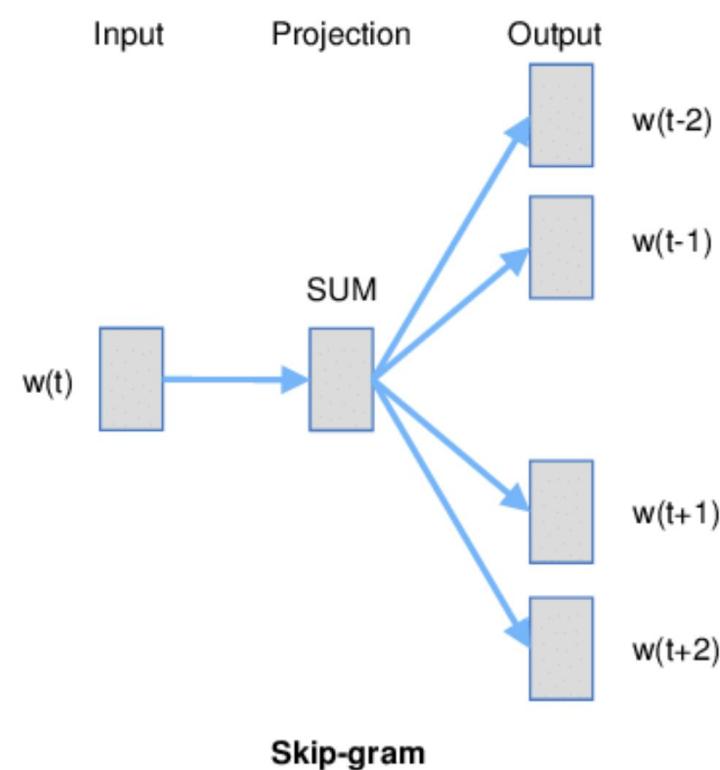
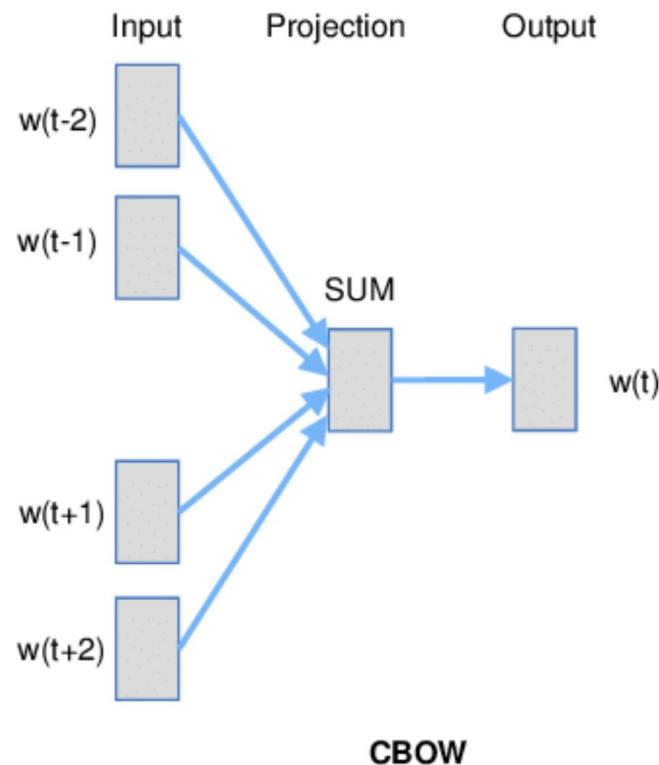
Тут тоже без неожиданностей: так как мы решаем задачу многоклассовой классификации с прогнозом вероятностей классов, то используем Cross-Entropy Loss (Log-Loss):

$$\text{Loss} = - \sum_{i=1}^{\text{output size}} y_i \cdot \log \hat{y}_i$$

А где же векторы слов?



CBOW и SkipGram



Вычислительные сложности

В архитектурах CBOW и SkipGram есть некоторая вычислительная трудность. Давайте рассматривать SkipGram, но в CBOW дела обстоят аналогично.

Когда мы для входного слова вычисляем вероятности слов быть в контексте данного слова, мы пользуемся формулой:

$$p(w_o|w_i) = \frac{\exp(v_{w_o}, v_{w_i})}{\sum_{w=1}^W \exp(v_w, v_{w_i})}.$$

Здесь

- w_i - входное слово, w_o - слово, для которого мы считаем вероятность быть в контексте слова w_i
- v_{w_o}, v_{w_i} - word2vec-векторы входного и выходного слов

В знаменателе суммирование идет по всем словам словаря в количестве W - их может быть очень много.

Затем после вычисления этой вероятности мы подставляем ее в функцию потерь и считаем по ней градиент.

В этом алгоритме кроется вычислительная сложность, так как в знаменателе стоит сумма по всем словам из словаря, которых может быть очень много.

Negative sampling

Давайте попробуем перейти от задаче многоклассовой классификации (где число классов = числу слов в словаре) к своего рода задаче бинарной классификации!

Для каждого входного слова мы имеем:

- *положительные примеры*: это слова, которые действительно находятся в контексте вокруг этого слова
- достанем из нашего обучающего корпуса *отрицательные примеры*: это случайные контексты. В силу случайности можно сказать, что эти слова не находятся в контексте вокруг входного слова

Negative sampling

Модифицируем функцию потерь - теперь она похожа на двухклассовый Log-Loss:

$$Loss = \sum_{(w, c_{true})} \log(p(c_{true}|w)) + \sum_{(w, c_{false})} \log(1 - p(c_{false}|w)).$$

Здесь

- (w, c_{true}) - все пары входное слово - реальный контекст этого слова. Для них мы считаем штраф при помощи первого слагаемого
- (w, c_{false}) - все пары входное слово - случайный (неверный) контекст этого слова. Для них мы считаем штраф при помощи второго слагаемого
- $p(c|w)$ - вероятность того, что слово c находится в контексте слова w

Negative sampling

На практике мы можем взять $k \in [2, 20]$ случайных отрицательных контекстов. Тем самым мы сильно снижим вычислительную сложность алгоритма - ведь для каждого входного слова **мы будем обновлять веса не у всех слов из словаря, а только у небольшого фиксированного количества слов.**

Эта процедура при этом сохраняет очень высокое итоговое качество модели.

Vanilla
Skip-Gram

$$\begin{array}{c} \text{W_output (old)} \\ \begin{matrix} -0.560 & 0.340 & 0.160 \\ -0.910 & -0.440 & 1.560 \\ -1.210 & -0.130 & -1.320 \\ 1.670 & -0.150 & -1.030 \\ 1.720 & -1.460 & 0.730 \\ 0.000 & 1.390 & -0.120 \\ -0.060 & 1.520 & -0.790 \\ 0.800 & 1.850 & -1.670 \\ -1.370 & 1.320 & -0.480 \\ 0.670 & 1.990 & -1.850 \\ -1.520 & -1.740 & -1.860 \end{matrix} \end{array} - \boxed{0.05} \times \begin{array}{c} \text{grad_W_output} \\ \begin{matrix} 0.064 & 0.071 & -0.014 \\ 0.098 & 0.015 & 0.063 \\ 0.069 & 0.089 & 0.045 \\ 0.014 & 0.085 & 0.079 \\ -0.021 & 0.067 & 0.071 \\ -0.098 & -0.088 & 0.091 \\ -0.072 & -0.078 & -0.089 \\ 0.046 & -0.079 & -0.053 \\ -0.049 & -0.087 & 0.025 \\ -0.060 & 0.092 & 0.042 \\ 0.074 & 0.050 & 0.070 \end{matrix} \end{array} = \begin{array}{c} \text{W_output (new)} \\ \begin{matrix} -0.563 & 0.336 & 0.161 \\ -0.915 & -0.441 & 1.557 \\ -1.213 & -0.134 & -1.322 \\ 1.669 & -0.154 & -1.034 \\ 1.721 & -1.463 & 0.726 \\ 0.005 & 1.394 & -0.125 \\ -0.056 & 1.524 & -0.786 \\ 0.798 & 1.854 & -1.667 \\ -1.368 & 1.324 & -0.481 \\ 0.673 & 1.985 & -1.852 \\ -1.524 & -1.743 & -1.864 \end{matrix} \end{array}$$

(11x3) (11x3) (11x3)

Negative
Sampling

$$\begin{array}{c} \text{W_output (old)} \\ \begin{matrix} -0.560 & 0.340 & 0.160 \\ -0.910 & -0.440 & 1.560 \\ -1.210 & -0.130 & -1.320 \\ 1.670 & -0.150 & -1.030 \\ 1.720 & -1.460 & 0.730 \\ 0.000 & 1.390 & -0.120 \\ -0.060 & 1.520 & -0.790 \\ 0.800 & 1.850 & -1.670 \\ -1.370 & 1.320 & -0.480 \\ 0.670 & 1.990 & -1.850 \\ -1.520 & -1.740 & -1.860 \end{matrix} \end{array} - \boxed{0.05} \times \begin{array}{c} \text{grad_W_output} \\ \text{Not computed!} \end{array} = \begin{array}{c} \text{W_output (new)} \\ \begin{matrix} -0.560 & 0.340 & 0.160 \\ -0.910 & -0.440 & 1.560 \\ -1.210 & -0.130 & -1.320 \\ 1.670 & -0.150 & -1.030 \\ 1.720 & -1.460 & 0.730 \\ 0.000 & 1.390 & -0.120 \\ -0.060 & 1.520 & -0.790 \\ \text{0.798} & \text{1.849} & \text{-1.672} \\ -1.366 & 1.318 & -0.477 \\ 0.667 & 1.985 & -1.847 \\ -1.523 & -1.744 & -1.858 \end{matrix} \end{array}$$

(11x3) (11x3) (11x3)

FastText

У алгоритма word2vec есть недостаток: если в новых текстах встречаются слова, отсутствующие в обучающей выборке, их не получится закодировать при помощи word2vec.

FastText - это модификация word2vec, которая решает эту проблему при помощи использования символьных N-грамм. То есть в качестве токенов используются не только слова, но и их кусочки, N-граммы.

- Например, 3-граммы слова кошка: кош, ошк, шка. В этом случае fasttext будет обучаться на следующих токенах: ко, кош, ошк, шка, ка, кошка (Токены ко и ка - на самом деле трехсимвольные: START_TOKEN + к + о и к + а + END_TOKEN, где START_TOKEN и END_TOKEN - специальные символы начала и конца слова).
- Само слово считается отдельной N-граммой.
- Чтобы посчитать вектор слова, мы суммируем векторы всех его N-грамм.

FastText

Поэтому, если встретится новое слово, то мы все равно сможем его векторизовать, так как оно состоит из N-грамм, которые с большой вероятностью присутствуют в обучении.



GloVe

Еще один недостаток word2vec - учет только локального контекста, то есть модель берет в расчет только ближайших соседей слова для его векторизации, при этом игнорируя глобальную структуру корпуса текста. В результате модель может упустить некоторые важные семантические отношения между словами.

Модель GloVe решает эту проблему, объединяя как локальную, так и глобальную статистику появления слов в корпусе текста. Она использует матрицу совместной встречаемости слов, чтобы учесть глобальные закономерности в данных. Такой подход позволяет учитывать не только ближайшие слова, но и все слова в корпусе текста при вычислении векторных представлений.

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k ice)/P(k steam)$	8.9	8.5×10^{-2}	1.36	0.96

Практика

[https://colab.research.google.com/drive/1ESTgWIEC8jj_kO-lx-U8r2D4z9Wja6KP?
usp=sharing](https://colab.research.google.com/drive/1ESTgWIEC8jj_kO-lx-U8r2D4z9Wja6KP?usp=sharing)