Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ФУНДАМЕНТАЛЬНЫЕ НАУКИ

КАФЕДРА МАТЕМАТИЧЕСКОЕ МОДЕЛИРОВАНИЕ

ОТЧЕТ ПО НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ:

Реализация алгоритма векторизации слов word2vec

Студент ФН12-11М _____ А. А. Мамаев
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель НИР _____ Е. С. Тверская
(Подпись, дата) (И.О.Фамилия)

Москва 2021 г.

Содержание

Введение

Векторное представление слов или “встраивание слов” (англ. word embedding) – термин, используемый в теории обработки естественного языка для разного рода представлений слов в форме действительного n-мерного вектора. Как правило, такое представление отображает семантическую связь между словами, например близость векторов в пространстве, соответствующих близким по значению словам.

В настоящий момент существует несколько алгоритмов векторизации слов, наиболее известные из которых Word2vec, FastText, GloVe и их модификации.

Векторное представление слов используется в задачах NLP, таких как семантический анализ и анализ тональности текста, а также во многих прикладных задачах, связанных с обработкой текстовой информации, например в рекомендательных системах и в интернет-рекламе для хранения и определения близких по смыслу поисковых запросов пользователей [1].

В рамках данной научно-исследовательской работы предлагается выполнить собственную реализацию алгоритма Word2vec на языке Python 3.

1. Общее описание Word2vec

Word2vec – класс алгоритмов, предназначенных для получения векторных представлений слов на естественном (в первую очередь – на английском) языке. Алгоритм был разработан и реализован в компании Google группой исследователей во главе с Томашом Миколовым в 2013 году.

Основная идея алгоритма заключается в задании векторного представления таким образом, что слова, имеющие близкий контекст в обучающих текстах, отображались в близкие векторы пространства [2].

Несмотря на простоту идеи, Word2vec оказался очень эффективным алгоритмом для передачи семантической связи между словами. Пусть имеется отображение

$$vec\colon W \to V \subseteq \Re^n \quad (1)$$

где W – множество слов входного языка, V– множество векторов, соответствующих входным словам. Обозначим также vec^{-1}отображение некоторого вектора $v \in \Re^n$ в слово, векторное представление которого наиболее близко к v, при этом в случае $v \notin V$ не гарантируется, что $vec(vec^{-1}(v)) = v$.

Тогда имеет место классический пример работы Word2vec, отражающий семантическую связь между словами:

$$vec^{-1}(vec("king") - vec("man") + vec("woman")) = "queen" \quad (2)$$

заключающийся, неформально говоря, в том, что *король* относительно *мужчины* есть то же самое, что относительно *женщины – королева*.

2. Принцип работы Word2vec

На начальном этапе всякому вектору $w \in W$сопоставляется случайный действительный вектор размерности n, причем n как правило велико: $100 \leq n \leq 1000$. Далее происходит итерационное улучшение модели за счет рассмотрения контекстов.

Рассмотрим число $r \ll n$ – окно рассмотрения контекста. Окном рассмотрения контекста слова w будем называть все слова, находящиеся в радиусе r с обеих сторон от w в исходном тексте. Пример окна рассмотрения контекста приведен на рисунке 1.

Рисунок 1 – Рассмотрение окна контекста размера $r = 3$ относительно слова “Andrew”

Для каждого слова в тексте (с учетом позиции в тексте) определим отображение

$$C\colon (W, \aleph) \rightarrow W^{2*r} \qquad (3)$$

данного слова в слова окна рассмотрения контекста.

На каждой итерации обучения модели происходит улучшение представлений векторов посредством изменения для каждого слова $w \in W$ векторов слов его контекста $C(w)$ таким образом, чтобы вероятность встретить любое из слов $c \in C(w)$в контексте w была максимальна. Такая модель, неформально говоря, предсказания нахождения слов c в контексте $C(w)$, называется Skip-gram [3]. Схема работы модели Skip-gram приведена на рисунке 2.

“Вероятность” при этом в сущности означает относительную частоту близости слов в тексте, определяемую формулой

$$P(w|c) = \frac{exp(vec(w)^T vec(c))}{\sum_{v \in W} exp(vec(v)^T vec(c))} \rightarrow max \quad (4)$$

являющейся так называемой softmax-функцией. Обучение модели в таком случае сводится к максимизации функции (4) для каждого слова.

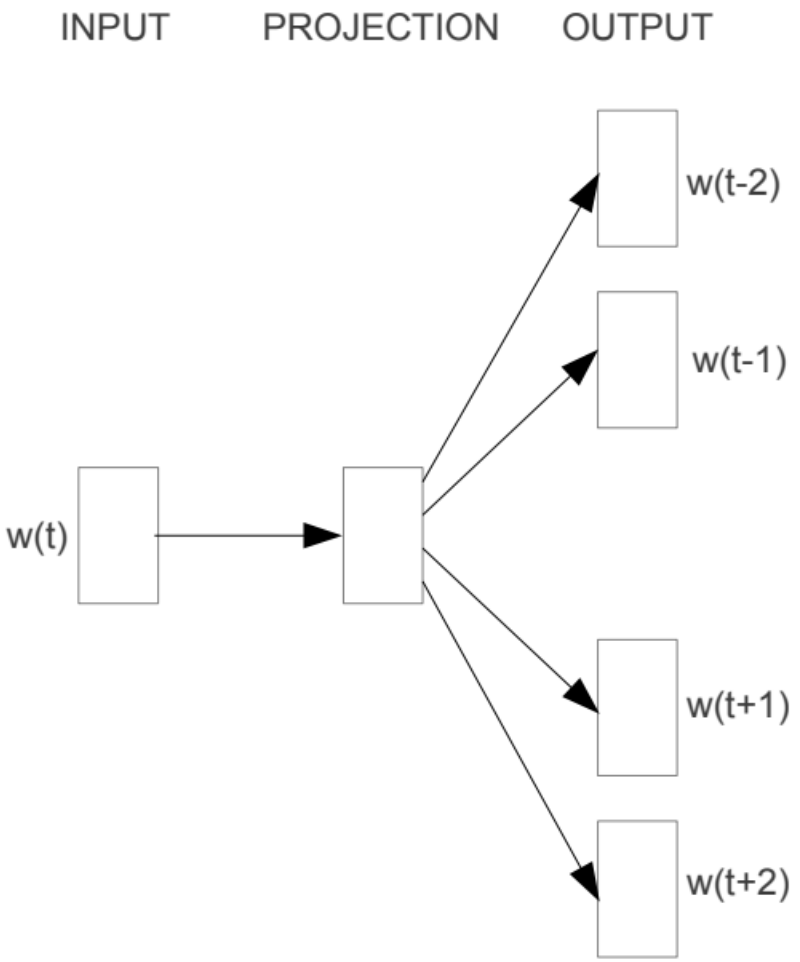

Рисунок 2 – Схема модели Skip-gram

Аналогичным образом может быть введена модель CBOW (Continuous bag of words), которой, в отличие от Skip-gram, присуще “предсказание” слова *w* по словам его контекста.

Другой моделью, используемой в Word2vec, является Negative Sampling. В данной модели обучение сводится к минимизации близости относительно каждого вектора *vec(w)* пространства набора векторов, которым соответствуют слова, не включаемые в контекст *w*.

Видно, что формула (4) трудоемка в вычислении ввиду наличия суммы экспонент по каждому из векторов пространства на каждом из шагов обучения. В настоящее время используются модели, которые не позволяют в явном виде предсказывать вероятности появления слов в контексте, процесс обучения которых, тем не менее, значительно быстрее [4]. Так, обучение модели может быть сведено к минимизации функции

$$\sum_{c \in C} exp(vec(w)^T \ vec(c)) + \sum_{c \notin C} exp(- vec(w)^T \ vec(c)) \quad (5)$$

где первая сумма соответствует близким по контексту словам, вторая – далеким (Negative Sampling), причем $c \notin C$подразумевает не все слова вне контекста, а какой-то набор фиксированного размера, например соизмеримый с окном контекста, или сигмоидальной функции

$$\sum_{c \in C} \frac{1}{1 + exp(-vec(w)^T vec(c))} - \sum_{c \notin C} \frac{1}{1+exp(-vec(w)^T vec(c))} \quad (6)$$

Существенно, что на каждом шаге оптимизации могут быть рассмотрены не все слова, а только некоторое их подмножество (*батч*) фиксированной длины.

3. Особенности реализации

Для реализации модели был выбран язык Python 3, за основу всех вычислений взята библиотека torch.

Так, для представления векторов, соответствующих словам, используется класс `torch.nn.Embedding`. Данный класс предоставляет доступ к векторам по некоторым их числовым идентификаторам (натуральные числа), позволяет проинициализировать векторы случайными значениями и, являясь параметром torch, оптимизируется в ходе обучения.

Для оптимизации модели использовался метод `torch.optim.Adam`. Хотя градиентный спуск `torch.optim.SGD` также может быть использован, его скорость работы оказалась существенно ниже.

Для определения ошибки модели в ходе рассмотрения близких по контексту слов использовалась функция сигмоида `torch.sigmoid`, в отличие от экспоненты ограниченная в интервале [0, 1] и функция кросс-энтропии `binary_cross_entropy`.

Полный исходный текст определения функции потерь в случае сравнения близких слов приведен на листинге 1, а в случае сравнения векторов с неконтекстными словами – на листинге 2.

Листинг 1 – определение ошибки сравнения слов в контексте

```
positive_embs = batch_embs.permute(0, 2, 1)
positive_sims = torch.bmm(batch_embs, positive_embs)
positive_probs = torch.sigmoid(torch.bmm(batch_embs, positive_embs))
positive_loss = torch.nn.functional.binary_cross_entropy(
                    positive_probs * self.positive_sim_mask,
                    self.positive_sim_mask.expand_as(positive_probs))
```

При этом `positive_sim_mask` – `2r`-диагональная матрица размера длины `2*2*r` с двумя полосами из единиц ширины r и нулями на диагонали.

Листинг 2 – определение ошибки сравнения слов вне контекста

```
negative_words = torch.randint(1, self.vocab_size,
 size=(batch_size, self.negative_samples_n))
negative_embs = self.embeddings(negative_words).permute(0, 2, 1)
negative_probs = torch.sigmoid(torch.bmm(batch_embs, negative_embs))
negative_loss = torch.nn.functional.
                binary_cross_entropy(negative_probs,
                      negative_probs.new_zeros(negative_probs.shape))
```

Входной текст проходит предварительную нормализацию сведением всех символов к строчному написанию, удалением небуквенных символов и разбиением текста на слова по пробельным символам:

```
i.lower() for i in
                re.sub('[^a-zA-Z]',' ', source).split(' ')
                if len(i) > min_token_size
```

4. Тестирование

В ходе тестирования на вход алгоритму подавался текст переведенного на английский язык романа Л. Н. Толстого “Война и мир”.

Аналогично примеру (2), в контексте романа можно выделить пары слов *князь Андрей* и *княжна Марья* Болконские (как правило, другим центральным персонажам не приписывается титул князя или княжны). Разумно ожидать, например, что

$$vec^{-1}(vec("prince") - vec("andrew") + vec("mary")) = "princess" \quad (7.1)$$

и

$$vec^{-1}(vec("andrew") - vec("prince") + vec("princess")) = "mary" \quad (7.2)$$

```
word2vec.most_similar_vector(
    word2vec.get_vector("prince")
    - word2vec.get_vector("andrew")
    + word2vec.get_vector("mary")
)
```

```
[('princess', 0.87626827),
 ('she', 0.8108855),
 ('that', 0.79813194),
```

(а)

```
word2vec.most_similar_vector(
    word2vec.get_vector("andrew")
    - word2vec.get_vector("prince")
    + word2vec.get_vector("princess")
)
```

```
[('mary', 0.8978914),
 ('she', 0.8663386),
 ('andrew', 0.83834875),
 ('after', 0.822762),
 ('her', 0.81443346),
 ('him', 0.8091848),
 ('was', 0.79340523),
 ('sha', 0.7916329),
 ('nat', 0.78282255),
```

(б)

Рисунок 3 – результаты выполнения тестов (7.1) и (7.2)

Результаты проведения данного теста приведены на рисунке 3. Видно также, что помимо имени княжны Марьи во втором тесте было подобрано и имя Наташи Ростовой, что также является правильным.

Что касается применения Word2vec в рекомендательных системах, можно проверить, например, что к слову “драгуны” близки слова “уланы” – иной род войск, “защищать” и “Фердинанд” – имя военначальника Фердинанда Карла Австрийского. Полный результат теста приведен на рисунке 4.

```
word2vec.most_similar("dragoons")
```

```
[('defend', 0.43215737),
 ('uhlans', 0.40895194),
 ('ferdinand', 0.3891443),
 ('noncommissioned', 0.35259026),
 ('seated', 0.34550554),
 ('stationed', 0.34464905),
 ('imperial', 0.34040946),
 ('died', 0.33877364),
```

Рисунок 4 – близкие в векторном пространстве слова к слову “драгуны”

Сравним время обучения реализованной модели и наиболее известной реализации Word2vec в библиотеке gensim [5]. Обучение реализованной модели происходит в среднем за 120 секунд, в то время как модели из библиотеки gensim – 22 секунды. Тем не менее, модели из gensim присущи как аппаратные, так и программные оптимизации, поэтому данный результат так же можно считать приемлемым для “наивной” реализации.

Заключение

В рамках настоящей научно-исследовательской работы была выполнена простейшая реализация алгоритма векторизации слов Word2vec.

В ходе работы были изучены основные принципы алгоритма и подходы к обучению модели, а также возможные применения.

На практическом примере рассмотрены основные направления использования векторного представления слов – поиск близких слов и сохранение семантической близости между операциями над векторами.

В результате тестирования установлено, что алгоритм работает корректно, а скорость обучения, являясь более низкой, чем у аналогов, остается допустимой для данной реализации.

Список использованных источников

1. *Л. Константиновский.* Практическое занятие по обработке текста в gensim с помощью алгоритма word2vec.
   URL: https://events.yandex.ru/events/science-seminars/26-oct-2016/
2. *T. Mikolov, I. Sutskever, K. Chen, G. Corrado, J. Dean*. Distributed Representations of Words and Phrases and their Compositionality, 2013
3. *T. Mikolov, I. Sutskever, K. Chen, G. Corrado, J. Dean*. Efficient Estimation of Word Representations in Vector Space, 2013
4. *V. Malykh.* Natural Language Processing course.
   URL: https://ods.ai/tracks/nlp-course
5. Gensim: Word2vec Embeddings
   URL: https://radimrehurek.com/gensim/models/word2vec.html