Московский Государственный Университет имени М.В.Ломоносова
Факультет Вычислительной Математики и Кибернетики
Кафедра Системного Программирования

Отчёт по заданию

Модель векторного представления слов на основе деревьев зависимостей предложений

Автор:
студент гр. 427
Архипенко Константин Владимирович

Москва, 2015

Содержание

1 Введение

В данной работе описывается модель `word2vecf` векторного представления слов на естественном языке. В отличие от популярной модели `word2vec`[1], она основана на использовании деревьев зависимостей предложений. Согласно [1], она лучше справляется с задачей поиска синонимов слов. Тем не менее, соответствующий программный инструмент[2] является плохо документированным и неудобным в использовании, а сама модель использовалась лишь в небольшом числе работ.

Целью работы является усовершенствование инструмента: переписывание вспомогательных программ для работы с моделью с языка программирования Python на язык C++, создание скрипта командной строки для запуска обучения модели, адаптация инструмента к формату входных данных, генерируемых как выходные данные синтаксическим анализатором (dependency parser) `RussianDependencyParser`[3] для русского языка.

[1] `https://code.google.com/p/word2vec/`

[2] `https://bitbucket.org/yoavgo/word2vecf/`

[3] `https://github.com/mathtexts/RussianDependencyParser/`

2 word2vec

Модель word2vec основана на использовании двухслойной нейронной сети прямого распространения. В соответствующем инструменте реализованы архитектуры *Continuous Bag-of-Words* и *Skip-gram* [2] для получения векторного представления слов фиксированной размерности d. Обучение при помощи word2vec является онлайновым: имеется скользящее по обучающему корпусу текстов симметричное контекстное окно переменного размера.

Будем рассматривать архитектуру *Skip-gram*, поскольку она является более качественной [2]. Предсказание текущего слова ω_i в этой архитектуре разбивается на отдельные предсказания этого слова на основе каждого из слов контекста:

$$\log P(\omega_i|\omega_1^{i-1}) = \log P(\omega_i|context(\omega_i)) = \sum_{j=-k}^{k} \log P(\omega_i|\omega_{i+j}).$$

Для ускорения обучения при вычислении $P(\omega_i|\omega_{i+j})$ вместо применения *softmax*-преобразования над всем словарём (как это делается, например, в ранней модели [3]) предлагаются подходы *hierarchical softmax* и *negative sampling*, второй из которых является более качественным [4].

Задачей *negative sampling* является отделение положительного примера (ω_i, ω_{i+j}) от s отрицательных примеров $(\omega_{neg_1}, \omega_{i+j}), ..., (\omega_{neg_s}, \omega_{i+j})$ на основе логистической регрессии, слова ω_{neg_l} выбираются случайным образом из некоторого распределения над словарём [5]:

$$\log P(\omega_i|\omega_{i+j}) = \log \sigma(v(\omega_i)v(\omega_{i+j})) + \sum_{l=1}^{s} \log \sigma(-v(\omega_{neg_l})v(\omega_{i+j})).$$

Здесь $v(\omega) \in \mathbb{R}^d$ – векторное представление слова ω, $\sigma(x) = \frac{1}{1+e^{-x}}$ – логистическая функция, а под произведением векторов понимается их скалярное произведение.

Особенностью векторных представлений word2vec является возможность решать на их основе задачи поиска синонимов и аналогий слов. Например, к вектору $v(king) - v(man) + v(woman)$ наиболее близким по косинусной мере среди векторов слов оказывается $v(queen)$ [6].

3 word2vecf

Использование линейного контекста является существенным недостатком `word2vec`. Модель `word2vecf` [1] вместо этого использует контекст на основе деревьев зависимостей предложений. При таком подходе появляется возможность в большей мере учитывать зависимости между словами в предложении и в меньшей мере учитывать близость в предложении двух слов, слабо связанных синтаксически.

Будем работать с деревьями зависимостей в формате конференции CoNLL-X [7]. Для каждой пары слов (`first`, `second`), в которой второе слово зависит от первого с типом `type`, составим две пары:

$$p_1 = (\texttt{first}, \texttt{type\_second});$$
$$p_2 = (\texttt{second}, \texttt{typeI\_first}).$$

Составим на основе корпуса зависимостей в формате CoNLL-X обучающий корпус пар, состоящий из пар вида p_1 и p_2 для всех имеющихся зависимостей. Составленный корпус и будет являться входными данными для инструмента `word2vecf`.

Инструмент использует два словаря – словарь слов (множество всех различных первых элементов пар в корпусе пар) и словарь контекстов (множество всех различных вторых элементов). Каждому из элементов в обоих словарях сопоставляется вещественнозначный вектор размерности d. Векторные представления слов и контекстов составляют веса первого и второго слоя нейронной сети соответственно.

В процессе обучения нейронная сеть предсказывает для текущей пары первый её элемент на основе второго. При этом из `word2vec` был заимствован подход *negative sampling* для ускорения обучения:

$$\log P(\omega_i|c_i) = \log \sigma(v(\omega_i)v(c_i)) + \sum_{l=1}^{s} \log \sigma(-v(\omega_{neg_l})v(c_i)).$$

Обучающий корпус пар может быть построен на основе корпуса зависимостей при помощи скрипта на языке Python 2, входящего в инструмент `word2vecf`.

4 Практическая часть

Исходный код используемых в данной работе инструментов находится в репозитории github[4].

4.1 Модификация `RussianDependencyParser`

В `RussianDependencyParser` добавлена возможность работы с файлом, содержащим большое число предложений.

Изменения касаются файлов `generateSentence.py` и `launch.sh`.

4.2 Реализация вспомогательных программ на языке C++

Инструмент `word2vecf` содержит два вспомогательных скрипта на языке Python 2:

- `scripts/vocab.py` – построение на основе корпуса зависимостей словаря, состоящего из всех различных слов, встретившихся в корпусе не менее *threshold* раз;
- `scripts/extract_deps.py` – построение на основе корпуса зависимостей и словаря, генерируемого предыдущим скриптом, обучающего корпуса пар.

Данные вспомогательные программы были переписаны на языке C++ с некоторыми улучшениями:

- возможность полноценной работы с Unicode: скрипт `scripts/extract_deps.py` не производит перевод русских букв в нижний регистр, во вспомогательных программах `cxx/vocab.cpp` и `cxx/extract_deps.cpp` перевод осуществляется;
- удаление пар с некоторыми нежелательными типами зависимостей: `undef`, `theme`, `ex`, `punct` и `lexmod` (и обратными к ним). Реализовано в `cxx/extract_deps.cpp`;
- удаление пар, где хотя бы одно из слов содержит символы, отличные от букв, цифр, знака подчёркивания и дефиса, реализовано в `cxx/extract_deps.cpp`.

[4] `https://github.com/arkhipenko-ispras/mathtexts/`

4.3 Адаптация `word2vecf` к `RussianDependencyParser`

Вспомогательные программы `cxx/vocab.cpp` и `cxx/extract_deps.cpp` работают с корпусом зависимостей, генерируемым `RussianDependencyParser`.

Формат этого корпуса отличается от CoNLL-X отсутствием последних двух столбцов.

4.4 Реализация скрипта для запуска `word2vecf`

Как часть инструмента `word2vecf` реализован скрипт командной строки `launch.sh` для запуска обучения на основе корпуса зависимостей. Автоматически при помощи вспомогательных программ на языке C++ генерируется обучающий корпус пар. Также можно настроить параметры обучения `word2vecf`, отредактировав скрипт.

Список литературы

[1] *Levy O., Goldberg Y.* Dependency-based word embeddings. — 2014.

[2] Efficient estimation of word representations in vector space / T. Mikolov, K. Chen, G. Corrado, J. Dean. — 2013.

[3] A neural probabilistic language model / Y. Bengio, R. Ducharme, P. Vincent, C. Jauvin. — 2003.

[4] Distributed representations of words and phrases and their compositionality / T. Mikolov, I. Sutskever, K. Chen et al. — 2013.

[5] *Levy O., Goldberg Y.* word2vec explained: Deriving Mikolov et al.'s negative-sampling word-embedding method. — 2014.

[6] *Mikolov T., tau Yih W., Zweig G.* Linguistic regularities in continuous space word representations. — 2013.

[7] *Buchholz S., Marsi E.* CoNLL-X shared task on multilingual dependency parsing. — 2006.