Аннотация

Данная работа включает в себя описание задачи построения языковых ассоциативных сетей, анализ существующих подходов к решению, а также изложение механизма работы предложенного метода вместе с полученными результатами. Было рассмотрено и протестировано несколько алгоритмов глубокого обучения, проведен сравнительный анализ их работы, выявлены возможные недостатки, а также приведены пути их решения и возможные направления для улучшения. В работе, кроме того, рассматриваются типичные для задачи регрессии проблемы, а также применение TripletLoss в задач обработки естественного языка.

В работе содержится 36 страниц, 12 иллюстраций, 2 таблицы. Использовано 9 источников литературы.

Abstract

This paper presents the task to construct and evaluate association networks from text corpora, as well as the detailed analysis of existing methods along with newly suggested approach. Several deep learning algorithms were considered, existing issues were revealed and probable solutions were proposed. The paper also outlines typical regression pitfalls and provide comprehensive way to employ all the perks of TripletLoss mechanism in the field of natural language processing.

The paper accounts for 36 pages, 12 illustrations, 2 tables and appeals for 9 outer resources.

Содержание

1 Введение

Языковые ассоциативные сети представляют собой ориентированные взвешенные графы, узлами являются слова, а ребра - наличие ассоциативной связи между ними. Вес каждого ребра - это численная интерпретация ассоциации между этими двумя словами.

Концептуально, ассоциативные сети представляют собой ментальный словарь языка. То есть число - вес ребра между двумя узлами в ориентированном графе - служит численной характеристикой того, насколько одно слово ассоциируется с другим в данном языке.

Конструирование таких сетей может производиться путем ручного сбора данных опросов или автоматически на основе текстового корпуса.

Ручное построение ассоциативных сетей трудоемко и требует много времени:

1. Фиксируется исходный словарь V конечной длины N существующих слов исследуемого языка;

2. Набирается группа респондентов, которые должны каждому слову $w \in V$ сопоставить k слов из исследуемого языка (они не обязательно из V), которые ассоциируются у них с w;

3. Для каждого $w \in V$ составляется статистика, которая показывает, какие слова респонденты указали, как ассоциирующиеся с w и сколько раз каждое из этих слов было указано, то есть $\forall w \in V$, $\exists! S_w = \{(\hat{w}_i, c_i)\}_{i=1}^{n_w}, n_w >= k, \hat{w}_i -$слово, которое хотя бы один респондент назвал ассоциацией на w, $c_i -$ сколько респондентов назвали $\hat{w}_i$ ассоциацией на w;

4. На основе данной статистики для каждого $w \in V$ и для каждого $\hat{w}_i \in S_w$ расчитывается оценка условной вероятности $p(\hat{w}_i|w)$ того, что $\hat{w}_i$ ассоциируется с w и мерой того, насколько $\hat{w}_i$ ассоциируется с w, как раз является оценка этой условной вероятности, то есть

$$\forall w \in V,\ \forall \hat{w}_i \in S_w\ p(\hat{w}_i|w) \approx \frac{c_i}{\sum\limits_{j=1}^{n_w} c_j} \tag{1}$$

;

5. Ассоциативную сеть можно представить в виде множества $G = \{(w, \hat{w}, p)_i\}_{i=1}^{Card}$ упорядоченных триплетов, где $Card-$мощность множества всех возможных триплетов, $Card >= N * k$;

Однако такой способ построения ассоциативных сетей позволяет получить наиболее точное приближение ментального словаря языка, поскольку задействует популяцию носителей языка.

Автоматические методы построения ассоциативных сетей полностью исключают опросы респондентов - самую затратную часть исследования. Вместо этого, ассоциативные сети строятся на основе корпуса печатных текстов, написанных на исследуемом языке следующим образом:

1. На основе имеющегося корпуса печатных текстов составляется словарь уникальных слов $\hat{V}$ конечной мощности $\hat{N}$;

2. Для каждого $w \in \hat{V}$ вычисляется его векторное представление, то есть устанавливается соответствие $E : \hat{V} -> \mathbf{R^d}, d-$размерность векторного представления слова;

3. Вводится функция расстояния (или псевдо-расстояния, когда требование симметричности не выполняется) $\rho : \hat{V} \times \hat{V} -> \mathbf{R}$, с помощью которой вычисляется оценка ассоциации между двумя словами;

4. Таким образом, снова получается ассоциативная сеть в виде множества упорядоченных триплетов;

Стоит упомянуть, что ассоциативные сети, полученные путем ручного сбора ответов респондентов, являются бенчмарком для алгоритмов автоматического конструирования сетей, то есть после построяния такого алгоритма, он тестируется на возможность повторить ассоциативную сеть, собранную в ручную. Таким образом, алгоритм автоматического построения ассоциативных сетей тем лучше, чем точнее он аппроксимирует условное распределение $P(\hat{w}|w), w \in V, \hat{w} \in \bigcup\limits_{w \in V} S_w$.

Почти аналогичную постановку задачи можно наблюдать в проблеме поиска семантических эмбедингов - векторных представлений для слов - [1, 2], однако при поиске семантических эмбедингов требовалось оценивать условную вероят-

ность $\hat{P}(\hat{w}|w)$ того, что $\hat{w}$ окажется в контексте w, что несложно сделать, введя формальное понятие контекста.

При построении же ассоциативных сетей, ввести формальное понятие ассоциации между двумя словами так, чтобы ее можно было оценить численно, не так просто, а существующий подход к понятию ассоциации (условная вероятность того, что $\hat{w}$ ассоциируется с w в ментальном словаре языка) требует существенных затрат. Поэтому задача при разработке алгоритмов автоматического построения ассоциативных сетей заключается в том, чтобы построить алгоритм, который наиболее точно приближает $P(\hat{w}|w), w \in V, \hat{w} \in \underset{w \in V}{\cup} S_w$.

В данной работе предложен алгоритм, который учится корректно моделировать распределение $P(\hat{w}|w)$, исходя из предположения, что $P(\hat{w}|w) = f(X, \theta) + \epsilon, \theta \in \Theta$, $X = \{(\hat{w}, w)_i\}_{i=1}^{\hat{N}}$ – множество всех упорядоченных пар слов из ассоциативной сети, составленной в ручную, мощности $\hat{N}$, $f(X, \theta)$ – семейство алгоритмов, параметризованных семейством параметров Θ, ϵ – случайная величина (ошибка), на которой выполняются предположения:

1. $E(\epsilon_i) = 0 \ \forall i$;
2. $Var(\epsilon_i) = \sigma^2 < \infty \ \forall i$;
3. $Cov(\epsilon_i, \epsilon_j) = 0, \ i \neq j$;

Семейство алгоритмов $f(X, \Theta)$ представляет собой семейство глубоких нейронных сетей, каждая из которых параметризована весами $\theta \in \Theta$, обучаемых с помощью процедуры градиентного спуска. Детальный обзор алгоритмов представлен в основной части работы.

2 Обзор литературы

Рассмотрим более подробно алгоритмы построения ассоциативных языковых сетей, упомянутые во Введении:

1. [3] представляет собой одну из первых работ, выполненных в этой области. Статья посвящена описанию и методу сбора данных для самой большой на то время языковой сети ассоциаций для английского языка. Группе из 150 респондентов были показаны 100-120 английских слов с тем, чтобы респонденты

написали напротив каждого данного слова первое слово, которое приходит им на ум. Далее была посчитана статистика и составлен датасет из 70 тысяч пар слов вместе с 33 характеристиками ассоциативной связи.

2. [4] описывает автоматический подход к построению ассоциативных сетей на основе корпуса текстов. В качестве векторного представления слов используется их $ppmi-$представление. Оно строится на основе PMI (Pointwise Mutual Information) - меры:

$$PMI(x, y) \equiv \log \frac{p(x, y)}{p(x) * p(y)} \tag{2}$$

Для того, чтобы построить $pmi-$представление слов, необходимо ввести понятие контекста - обычно, это фиксированное число слов до центрального слова и столько же после центрального слова, размер контекста является гиперпараметром алгоритма. После этого составляется множество пар слов $D = \{(c, ct)_i\}_{i=1}^{N}$,где $c-$это центральное слово, а $ct-$это слово, появившееся в контексте этого центрального слова. Для каждой пары слов считается оценка $pmi-$меры:

$$PMI(x, y) \approx \log \frac{\#(x, y)}{\#x * \#y} \tag{3}$$

$\#(x, y)-$сколько раз пара (x, y) встречается в D, $\#x-$сколько раз слово x встречается в корпусе.

Составляется матрица размера $C \times C, C-$количество уникальных слов в D, $ij-$ый элемент этой матрицы равен $\log \frac{\#(i, j)}{\#i * \#j}$; чтобы получить векторное представление $k-$ого слова необходимо взять $k-$ую строку этой матрицы.

PPMI (Positive Pointwise Mutual Information) отличается от 2 лишь тем, что элементы, меньшие нуля, зануляются. Это позволяет избежать больших отрицательных чисел, которые образуются, когда отношение в 2 под логарифмом близко к нулю.

После получения $ppmi-$представления слов, вводятся функции расстояния

$$f(x, y) = \frac{< x, y >}{||x|| * ||y||} \tag{4}$$

$$g(x, y) = \frac{< x, y >}{||y||} \tag{5}$$

Заметим, что 5 - псевдо-метрика, у которой не выполняется требование симметрии относительно агрументов, это позволяет получать разные оценки меры ассоциативности между двумя словами в зависимости от их порядка, что, вообще говоря, является желательным свойством, так как в ручную собранные ассоциативные сети им обладают.

Имея векторные представления слов и 4, 5, для каждой пары слов составляется упорядоченный триплет, после чего этот триплет добавляется в ассоциативную сеть. Алгоритм продолжается, пока не закончатся пары слов.

Такой подход, безусловно, имеет свои плюсы:

(a) Векторное представление слов можно быстро посчитать;

(b) Вычисление 4, 5 осуществляется за один проход по массиву пар и не требует больших вычислительных ресурсов;

(c) Можно получить несимметричные оценки ассоциативной меры между словами;

Однако, есть и недостатки:

(a) Векторное представление может не помещаться в память вычислительной машины, так как количество элементов - это квадрат от длины словаря уникальных слов (очевидным решением является хранение векторного представления в виде sparse matrix, ведь $ppmi-$представление позволяет это сделать);

(b) $ppmi-$представление использует только локальную статистику, эта проблема решается в [2];

(c) Итоговые оценки ассоциативной меры сильно зависят от выбора функции растояния, которые не могут быть подобраны в автоматическом режиме;

3 Постановка задачи

Предполагая, что есть в ручную собранная языковая ассоциативная сеть $HNet$, необходимо построить алгоритм $\mathbf{A}$ автоматического построения языковой ассоциативной сети, который наиболее точно аппроксимирует $HNet$.

4 Описание данных

Для исследования была взята ассоциативная сеть английского языка [5], которая собиралась с 2011 по 2018 год. Обозначим ее SW.

$SW = \{(cue, response, score)_i\}_{i=1}^{N}$, cue—это слово, которое давалось респондентам, $response$—это набор слов, который ассоциируется у респондента с cue, $score$—это оценка условной вероятности того, что $response$ ассоциируется с cue. В случае SW, $N = 1389599$.

cue представлены в виде нормализованных лексем. Нормализация заключалась в исправлении орфографических ошибок и приведению к нижнему регистру. В качестве cue также могли быть взяты словосочетания из двух слов - фразовые глаголы английского языка.

$response$ представляют собой последовательности слов, которые ассоциируются у респондентов с cue. Они были нормализованы так же, как и cue, однако длина $response$ может доходить до 15 слов.

Распределение длины ответов респондентов можно видеть на Рис. 1.

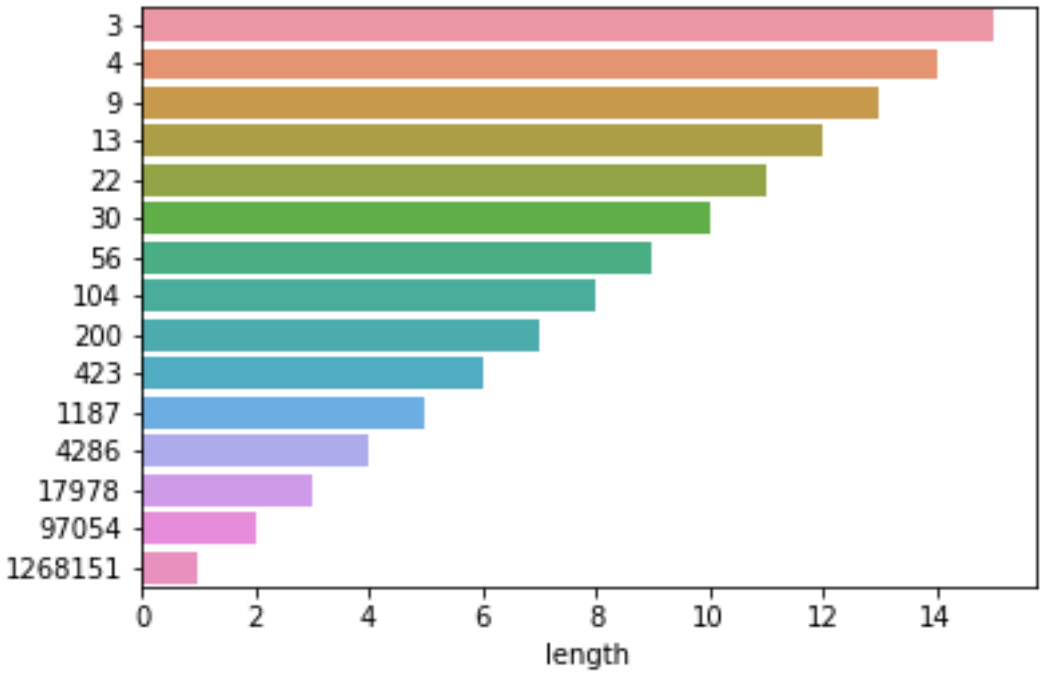

Рис. 1. Распределение длин ответов респондентов. На оси абсцисс отложены длины ответов, на оси ординат - количество ответов с такой длиной.

Пример отзыва из 15 лексем можно видеть на Рис. 2.

```
cue =  alien
response =  jodie foster but i know she wasn t in that movie  i m not st
score =  0.0034843205574912896
```

Рис. 2. Пример ответа респондента из 15 лексем

Пример части визуализированной *SW* можно видеть на Рис. 3.

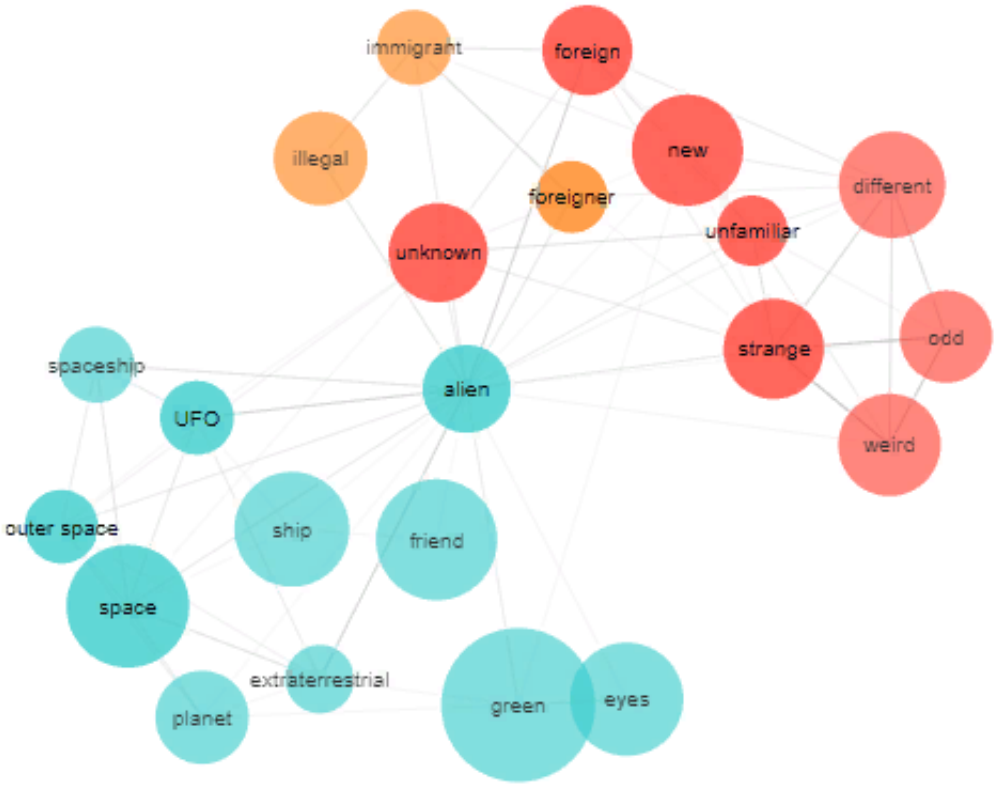

Рис. 3. Часть визуализированной сети ассоциаций SW

5 Подготовка данных

5.1 Разбиение

Подготовка данных для обучения алгоритма во многом обусловлена выбором алгоритма. Как указывалось во Введении, были выбраны глубокие нейронные сети, как потенциальные алгоритмы для автоматического построения ассоциативных сетей.

Исходя из этого выбора, *SD* необходимо было разбить на три части:

1. Датасет для обучения, обозначим его T;

2. Датасет для валидации модели и подбора гиперпараметров, обозначим его V;

3. Датасет для итогового тестирования модели, обозначим его Tst;

Датасет для обучения включает в себя 70% исходных данных, а V и Tst по 0.15%. Исходные данные отсортированы в лексикографическом порядке, поэтому перед разбиением они были перемешаны.

Важный нюанс касается временного промежутка, в течение которого собирались данные. Обычно, если данные зависят от времени, то разбиение на обучение, валидацию и тест должно это отражать, то есть на временной шкале эти три части не должны пересекаться. В случае данных ассоциативной сети SD, которые собирались в течение 7 лет, не ясно, оказало ли развитие языка существенное влияние на собираемую статистику, поэтому вопрос о влиянии временного промежутка на разбиение исходного датасета остается открытым, и, так как изначально данные SD размещались в открытом доступе без привязки к датам, было решено делать разбиение без учета влияния времени.

Стоит упомянуть о важности разделения данных для валидации и теста. Это разбиение в том числе необходимо, чтобы не возникло переобучения модели на данных для теста. Хотя модель не обучается на данных из датасета для валидации, подбор гиперпараметров и лучшей итерации осуществляется именно на них, поэтому если использовать датасет для теста в качестве валидационного, может возникнуть переобучение и обобщающая способность модели будет оценена неверно.

5.2 Cue, Response

Процедура подготовки последовательностей лексем для обучения алгоритма включает в себя несколько шагов:

1. Из cue и response были удалены все символы, которые не являются алфавитными, цифрами, пробелом или тире. Тире не было удалено, чтобы сохранить составные слова;

2. Обычно, для решения задач естественного языка проводится еще приведение

слова к его начальной форме, однако, в случае составления ассоциативных сетей, важно учитывать именно ту форму слова, в которой слово было названо респондентом, поэтому словоформы были оставлены без изменения;

3. Для уникальных словоформ из *cue* был составлен словарь D_c : $cue- > \mathbf{N}$, который содержал соответствие между уникальной словоформой и натуральным числом;

4. Аналогичный словарь D_r : $response- > \mathbf{N}$ был составлен для ответов респондентов;

5. Словоформы из cue и response были перенумерованы в соответствии с D_c и D_r;

Стоит упомянуть, что словари D_c и D_r составляются на основе словоформ из датасета для обучения, чтобы не было протекания информации из теста и валидации в обучение. Для словоформ, которые встречаются только в тесте или валидации, в каждом словаре предусматривается специальный символ, который обозначает словоформу, не встретившуюся в датасете для обучения.

Следует заметить, что векторное представление для символа, обозначающего словоформу, невстретившуюся в обучении, никак не будет задействовано во время обучения, поэтому во время валидации и теста будет использоваться значение, данное при инициализации. Возможным решением данного нюанса является исключение из словарей словоформ, которые встречаются во время обучения достаточно редко. Однако, в случае SD оба словаря получились достаточно большими, чтобы пренебрегать вероятностью встречи новой словоформы в валидации и тесте, не зависящим от времени.

5.3 Оценка условной вероятности и вывод функционала для обучения алгоритма

Объектом обучения алгоритма является корректное моделирование распределения $P(\hat{w}|w)$,указанного во Введении. Кроме предположений

1. $P(\hat{w}|w) = f(X, \theta) + \epsilon$

2. $E(\epsilon_i) = 0 \ \forall i$

3. $Var(\epsilon_i) = \sigma^2 < \infty \; \forall i$

4. $Cov(\epsilon_i, \epsilon_j) = 0, i \neq j$

необходимо сделать еще одно о распределении случайной величины ϵ_i. На основе этого предположения будет происходить выбор функции ошибки для обучения алгоритма. Для этого рассмотрим распределение целевой переменной на датасете для обучения (используем только его, чтобы не допустить протекания информации из валидации и теста в процесс обучения). Распределение можно видеть на Рис. 4.

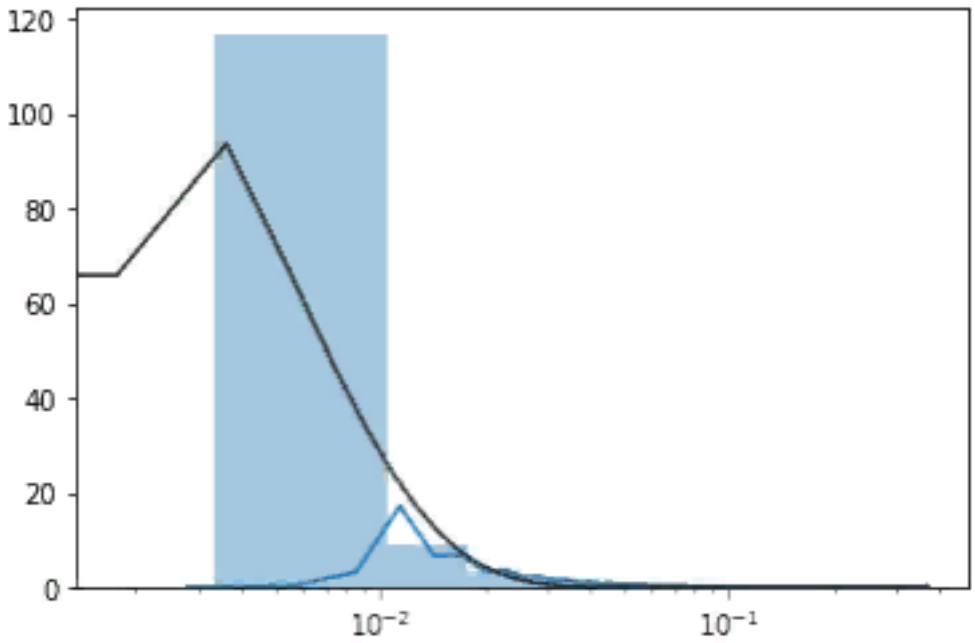

Рис. 4. Распределение целевой переменной на обучающей выборке. На оси абсцисс отложены значения целевой переменной, на оси ординат - количество объектов с таким значением целевой переменной. Черной линией выделена аппроксимация исходного распределения распределением Лапласа.

Распределение целевой переменной имеет сильное смещение влево, возможным решением (чтобы получить желаемое нормальное распределение или хотя бы сделать исходное распределение более симметричным) является удаление значений целевой переменной, которые больше некоторого квантиля. Однако это означает потерю части информации (большой части, так как удалять придется много, чтобы сделать распределение похожим на нормальное). Поэтому было решено отказаться от использования среднеквадратичной функции ошибки, выбор которой является автоматическим следствием предположения о нормальном распределении ошибки ϵ_i(и как следствие целевой переменной).

Принимая во внимание сильную смещенность распределения влево, было решено сделать предположение, что ошибка подчиняется распределению Лапласа:

$$\epsilon_i \sim Laplace(0, \sigma^2) \tag{6}$$

Как следствие, из предположения 1 о виде зависимости целевой переменной и свойства распределения Лапласа, получаем

$$P(\hat{w}|w) \sim Laplace(f(X, \theta), \sigma^2) \tag{7}$$

С целью демонстрации предположения 7, на Рис. 4 была показана аппроксимация распределения целевой переменной распределением Лапласа.

Принимая во внимание 7 и вид функции распределения $g(x|\mu, \sigma) = \frac{1}{2\sigma} \exp\left(-\frac{|x-\mu|}{\sigma}\right), x \sim Laplace(\mu, \sigma^2)$, можно получить вид функции ошибки для обучения алгоритма (для краткости $P(\hat{w}|w)$ заменяется на $P = \{p_i\}_{i=1}^N$):

$$\theta^* = \underset{\theta}{argmax}[L(\theta|P)] = \underset{\theta}{argmax}[g(P|\theta)] = \underset{\theta}{argmax}\left[\prod_{i=1}^{N} g(p_i|\theta)\right] =$$

$$\underset{\theta}{argmax}\left[\prod_{i=1}^{N} \frac{1}{2\sigma^2} \exp\left(-\frac{|p_i - f(x_i, \theta)|}{\sigma^2}\right)\right] = \underset{\theta}{argmax}\left[\prod_{i=1}^{N} \exp\left(-\frac{|p_i - f(x_i, \theta)|}{\sigma^2}\right)\right] =$$

$$\underset{\theta}{argmax}\left[\exp\left(-\frac{\sum_{i=1}^{N} |p_i - f(x_i, \theta)|}{\sigma^2}\right)\right] = \underset{\theta}{argmax}\left[\log\left[\exp\left(-\frac{\sum_{i=1}^{N} |p_i - f(x_i, \theta)|}{\sigma^2}\right)\right]\right] =$$

$$\underset{\theta}{argmax}\left[-\sum_{i=1}^{N} |p_i - f(x_i, \theta)|\right] = \underset{\theta}{argmin}\left[\sum_{i=1}^{N} |p_i - f(x_i, \theta)|\right] \tag{8}$$

, L - функция правдоподобия.

Таким образом, функционал для обучения алгоритма:

$$Loss = \frac{1}{N} \sum_{i=1}^{N} |p_i - f(x_i, \theta)| \tag{9}$$

Целевая переменная остается без изменений.

6 Выбор и описание модели

Каждая модель представлена отдельным проектом. Каждый проект состоит из нескольких файлов:

1. input.py - файл, в котором прописана структура подачи данных в модель для обучения, валидации и теста;
2. build.py - файл, в котором содержится архитектура модели;
3. train.py - файл, который содержит код для процедуры обучения модели;
4. eval.py - файл для валидации модели;
5. test.py - файл для итогового тестирования модели;

Алгоритмы реализованы с помощью фреймворка для глубокого обучения TensorFlow[1].

Все проекты можно найти в репозитории на гитхабе https://github.com/meduzick/Assoc Networks-for-English-Language.

6.1 Полносвязная сеть с двумя матрицами для векторных представлений и усреднением ответов

Идея состоит в том, чтобы использовать векторные представления cue и response. Это позволит получить вектор, который можно подать на вход полносвязной сети.

Алгоритм, таким образом, можно представить в виде двух частей:

1. Encoder;
2. Полносвязная сеть;

6.1.1 Encoder

Данная часть используется, чтобы получить векторное представление пары (cue, response). Для этого вводится две матрицы:

[1] https://www.tensorflow.org/

1. $E_{cue} : [|D_c|, 100]$;

2. $E_{resp} : [|D_r|, 100]$;

, где $|D_c|$ и $|D_r|$ обозначают мощность словарей для уникальных лексем из cue и response соответственно, 100 - выбранная в этой работе размерность векторного представления.

Используется две различные матрицы, так как cue и response порождают разные словари, при этом одинаковые слова в одной паре (cue, response) могут иметь совершенно разное значение, поскольку словоформа cue может использоваться в составе response и принимать другой оттенок значения. Исходя из этих предположений, было принято решение разграничить векторные представления для cue и response.

Как было описано в разделе 4, ответ респондента может состоять больше чем из одного слова. Составление словаря с учетом целых предложений нецелесообразно: это в разы увеличивает мощность словаря D_r, что также приводит к росту числа параметров модели. Было решено использовать простую стратегию: получать векторное представление для каждой лексемы в response, а затем брать линейную комбинацию этих векторов. Таким образом, получается векторное представление response, состоящего из нескольких лексем:

$$e_k = \sum_{i=1}^{n_k} \alpha_i * e_i^k \tag{10}$$

, где e_i^k—это векторное представление i-ой лексемы k-ого отзыва.

Реализация выбранной стратегии требовала некоторых дополнений к основному проекту. Как было сказано в начале раздела 6, реализация алгоритмов написана на TensorFlow, это подразумевает работу с объектами (тензорами) фиксированной формы. Для выполнения этого требования необходимо дополнить все последовательности response специальным символом <PAD> до определенной фиксированной длины FL. Из-за этого во время получения векторного представления для символов, обозначающих действительные лексемы response, к этому векторному представлению добавляется еще векторное представление для <PAD>, которое нельзя учитывать в 10, чтобы не получить неверных результатов.

Чтобы избавиться от векторного представления для <PAD>, на каждом шаге получения векторного представления E_r для response составлялся также трехмерный тензор T:

1. E_r имеет форму $[batch\_size, FL, 100]$ в виду того, что для каждого символа каждой последовательности длины FL из набора последовательностей размера $batch\_size$ берется векторное представление длины 100;

2. T имеет такую же форму как E_r и составляется по правилу

$$T_{ijk} = \begin{cases} 1, & j < l_i \\ 0, & \text{otherwise} \end{cases} \tag{11}$$

   , где l_i—тензор, содержащий действительные длины i-ой последовательности response;

3. Результирующий тензор получается как поэлементное произведение T и E_r и последующее усреднение вдоль оси, содержащей символы последовательностей (усреднение проводится с учетом реальных длин из l_i, а не FL);

В случае данного проекта, инициализация матриц E_{cue}, E_{resp} производится из нормального распределения.

6.1.2 Полносвязная сеть

Полносвязная сеть представляет собой последовательность перемножений матриц и нелинейностей, которую в общем случае можно отобразить как

$$a_{i+1} = s_{i+1}(W_{i+1} * a_i + b_{i+1}) \tag{12}$$

, a_i—активации, полученные на предыдущем слое сети, W_{i+1}—веса i+1-го слоя сети, s_{i+1}—нелинейность i+1-ого слоя (для каждого слоя может быть своя функция нелинейности), b_{i+1}—смещения.

Конфигурация полносвязной сети в данном проекте состоит из семи слоев. Размеры слоев в том порядке, в котором они идут в модели: $[1024, 512, 256, 128, 64, 32, 1]$.

В качестве нелинейности используется elu:

$$elu(x) = \begin{cases} x, & x > 0 \\ \alpha * (e^x - 1), & \text{otherwise} \end{cases} \tag{13}$$

elu считается дольше, чем аналоги Relu, LeakyRelu, ParametricRelu за счет появления экспоненты, однако имеет ряд положительных свойств, желательных при решении задачи:

1. У elu нет регионов насыщения, в которых производная равна 0 вследствие чего может возникнуть проблема мертвых нейронов;

2. Множество значений elu лежит как в положительных, так и в отрицательных числах, что благоприятно сказывается на траектории градиентного спуска (когда значения функций активации строго одного знака, траектория может получится ступенчатой);

3. Возможна такая ситуация, когда сети легче выучить смещения, чем веса, в таком случае, сеть зануляет первое слагаемое из 12, тем самым делая активации с предыдущего слоя очень близкими к нулю:

$$a_i = s_i(W_i * a_{i-1} + b_i), a_i -> 0 \tag{14}$$

$$elu(x) = 0 \Leftrightarrow x = 0 \tag{15}$$

Из 15, 14 следует, что в процессе зануления активаций i-го слоя, аргумент s_i также будет стремиться к нулю. Производная s_i участвует в образовании градиентов для нижележащих слоев, а значит, множитель $s_i'(0)$ будет входить в выражение для нижележащих градиентов, однако,

$$elu'(x) = \begin{cases} 1, & x > 0 \\ \alpha * e^x, & \text{otherwise} \end{cases} \tag{16}$$

, из чего следует, что производная 13 не обращается в 0 при аргументе равном нулю, а значит, нейрон не становится мертвым;

Инициализация была выбрана исходя из вида функции активации и заимствована из [6].

Стоит отметить выбор функции активации на последнем слое. Зачастую не рекомендуется применять никакие функции активации на последнем слое, кроме линейных. В этой работе, выбор сделан в пользу сигмоиды:

$$Sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{17}$$

по причине того, что целевая переменная не центрированная и лежит в интервале (0, 1), а сигмоида как раз позволяет избежать отрицательных значений и значений больше 1, которые могут получиться после линейного преобразования на последнем слое.

Однако, такой подход имеет и свои недостатки. Как видно из графика распределения целевой переменной на Рис. 4, большая часть значений целевой переменной меньше 0.05 и сосредоточена около нуля справа. Алгоритму достаточно выучить, что нужно линейным преобразованием на последнем слое получить большие по модулю отрицательные числа, вследствие чего значения сигмоиды будут очень близки к нулю и этого будет достаточно, чтобы минимизировать функционал ошибки, ведь 9 не так чуствительна к выбросам. Но из-за того, что производная сигмоиды имеет вид

$$Sigmoid'(x) = Sigmoid(x)(1 - Sigmoid(x)) \tag{18}$$

, она будет очень близка к нулю при больших по модулю отрицательных аргументах, что приводит к тому, что градиенты, в вычислении которых участвует производная сигмоиды, будут близки к нулю и произойдет отмирание нейронов сети.

Это, кроме всего прочего, негативно скажется на градиентах для матриц E_{cue}, E_{resp}, которые располагаются дальше всех от последнего слоя, выполняющих, однако, решающую роль в трансформировании пар (cue, response) в вид, пригодный для обучения полносвязной сети.

Возможным решением является использование skip-connections. Идея в том, чтобы использовать активации нижних слоев в вычислении активаций далеко остающих слоев. Формально, если a_i—активации i-го слоя, а k - достаточно большое

число, то

$$a_{i+k} = f_{i+k}(W_{i+k} * a_{i+k-1} + b_{i+k} + W_t * a_i) \tag{19}$$

, где W_t — матрица, позволяющая перевести активации a_i в размерность активаций a_{i+k}.

6.1.3 Обучение и результаты

Полный граф архитектуры можно видеть на Рис. 5.

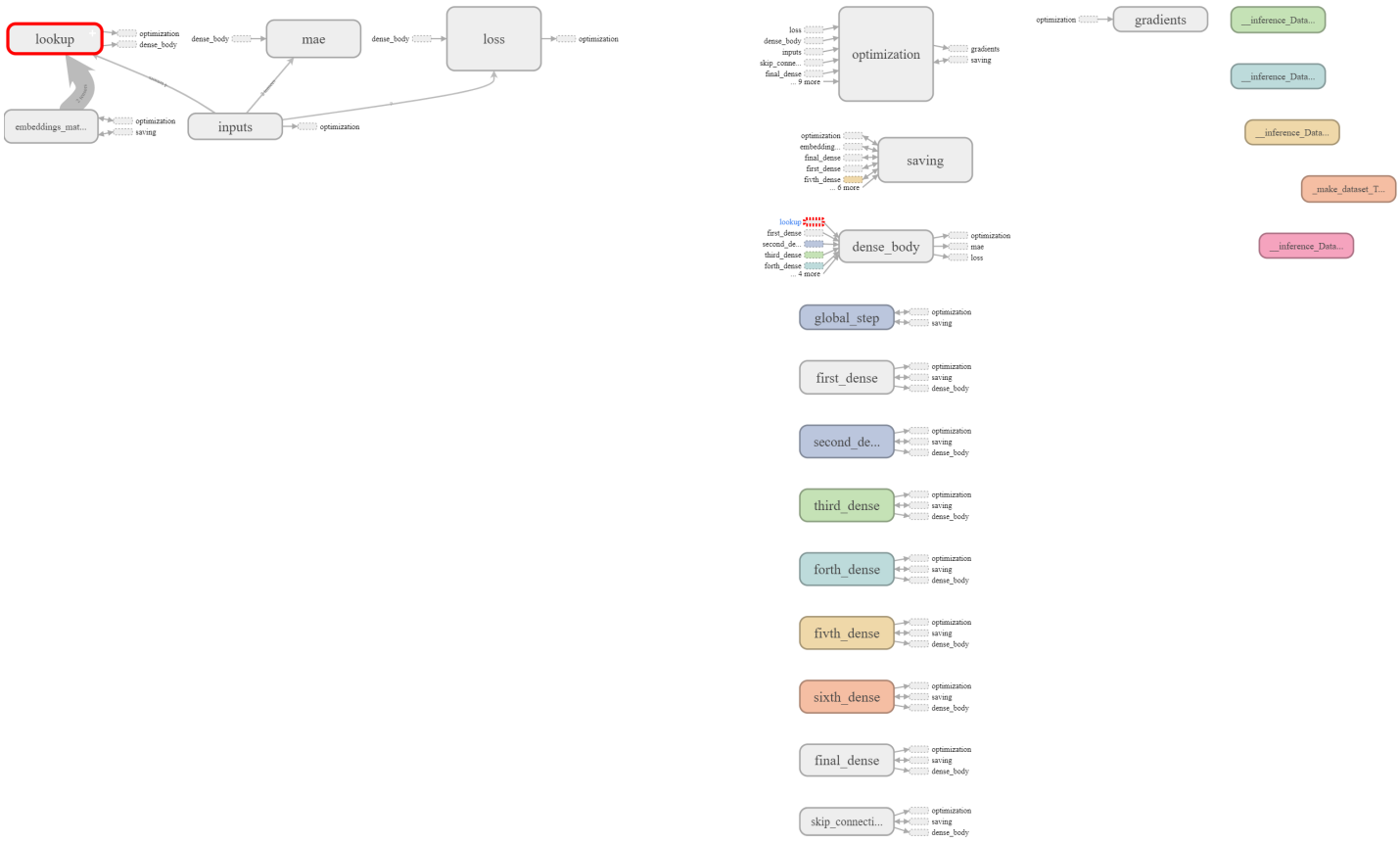

Рис. 5. Полный граф архитектуры

График значений функции ошибки во время обучения можно видеть на Рис.6.

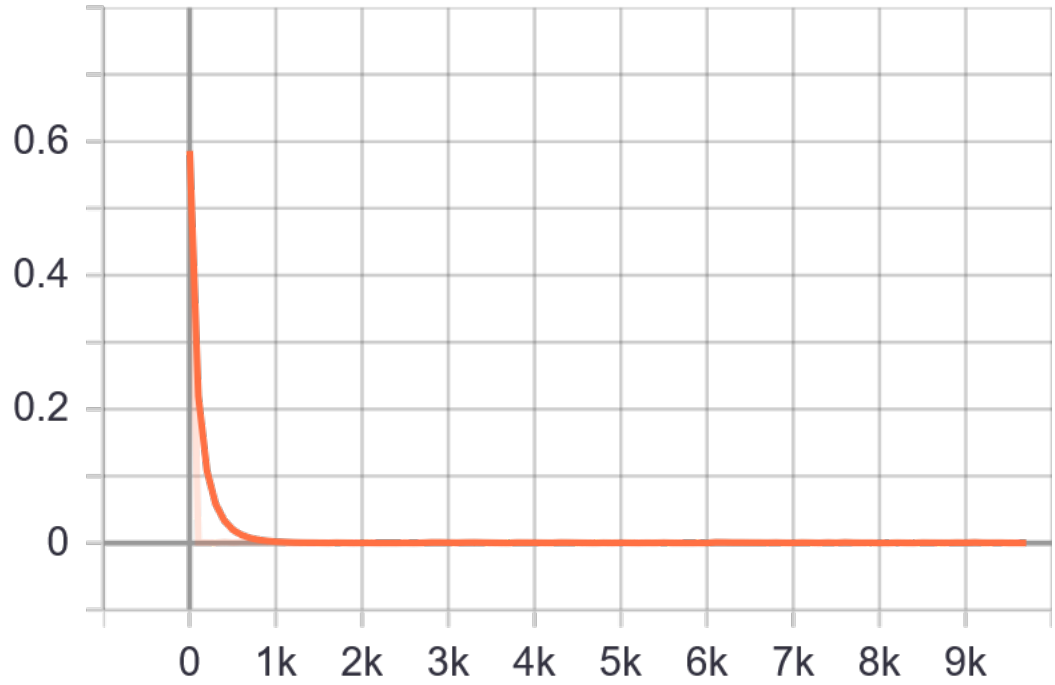

Рис. 6. График функции ошибки на обучении. На оси абсцисс расположены номера итераций процесса обучения, на оси ординат - значение функции ошибки.

График средней абсолютной ошибки на валидации можно видеть на Рис. 7.

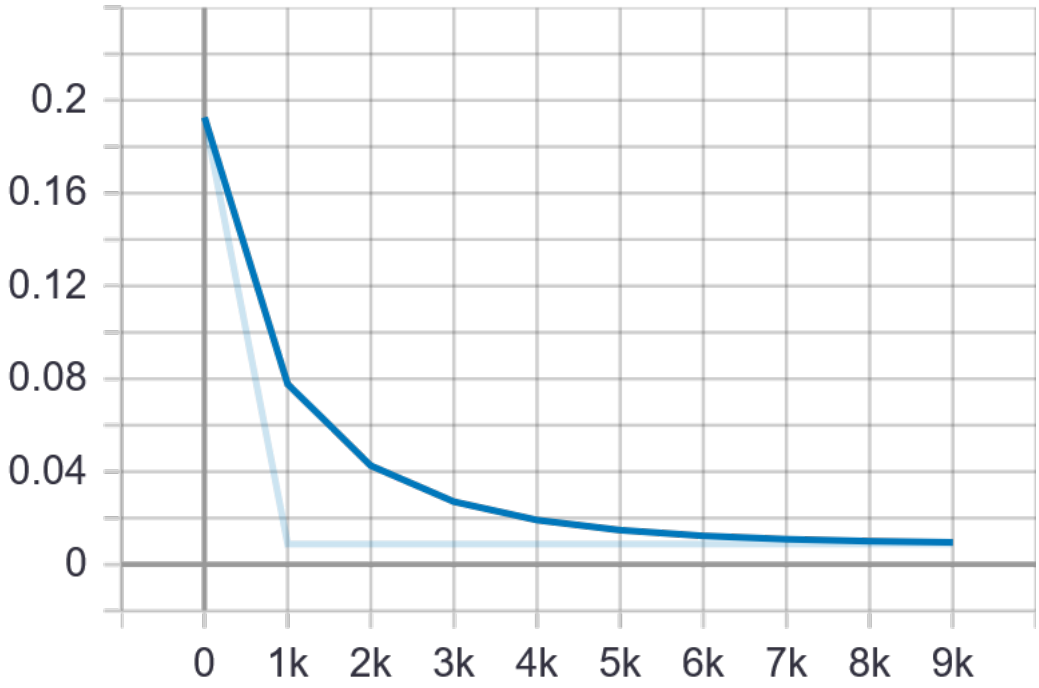

Рис. 7. Значения средней абсолютной ошибки на валидации. На оси абсцисс расположены номера итераций процесса валидации, на оси ординат - значение функции ошибки.

Конфигурация процедуры обучения:

1. Количество эпох = 10

2. Количество данных в модель за один шаг (batch size)= 100

3. Learning rate = 3e-04

Результаты обучения:

1. Средняя абсолютная ошибка на тесте $\approx 5e-03$

2. $R^2 = 0.6$

6.1.4 Объяснение результатов и недостатки

На графиках 6 и 7 видно, что период интенсивного обучения пришелся на первые тысячу шагов, это можно объяснить тем, что модель выучивала смещения на последнем слое. В дальнейшем, такого эффекта можно избежать, если проинициализировать смещения последнего слоя средним значением целевой переменной.

Однако после тысячной итерации, интенсивность обучения резко снижается. Если посмотреть на распределение градиентов на переменных - векторных представлениях пар (cue, response) в Приложении 1 (9, 10), то можно заметить, что все они сосредоточены около нуля (амплитуда колеблется не выше, чем 1e-05), что говорит о том, что модель не учит векторное представление слов. Это может быть связано с использованием сигмоиды на последнем слое и мертвыми нейронами, а также с большой мощностью обоих словарей D_c, Dr и относительно не высокой частотой лексем из этих словарей (особоенно из D_r).

В следующей секции приводятся способы решения данных проблем.

6.2 Переход к задаче классификации и использование векторных представлений из подзадачи

Основной идеей данного раздела является переход от исходной задачи регрессии к задаче классификации и использование результатов решения задачи классификации обратно в решении задачи регрессии.

6.2.1 Переход к задаче классификации

При более тщательном анализе распределения целевой переменной можно заметить, что уникальных значений относительно не много (это может быть объяснено

тем, что целевая переменная расчитывается как отношение натуральных чисел, каждое из которых взято из ограниченного интервала) и что большинство этих значений живут внутри непересекающихся отрезков на вещественной прямой. Было предложено разбиение исходного множества принимаемых целевой переменной значений на 10 полуинтервалов, распределение целевой переменной на некоторых из этих полуинтервалов можно видеть на графиках 11, 12, 13 в Приложении 1.

Таким образом, переход от задачи регрессии к задаче классификации осуществляется за счет присвоения паре $(cue, response)_i$ с исходной целевой переменной p_i новой целевой переменной

$$target_i = \sum_{j=1}^{10} \mathbb{1}_{p_i \in I_j} * j \tag{20}$$

, $\{I_i\}_{i=1}^{10}-$непересекающиеся упорядоченные интервалы, на которые было разбито множество значений исходной целевой переменной.

Из-за смещения распределения исходной целевой переменной влево, баланс классов также будет смещен в сторону классов с меньшими порядковыми номерами, так как интервалы выбирались по значению целевой переменной $P(\hat{w}|w)$,а не по количеству объектов внутри интервала.

Решение задачи регрессии в 6.1 показало, что возникают большие трудности с достаточно точной аппроксимацией смещенного влево распределения исходной целевой переменной, из-за чего модель не в состоянии адекватно присваивать значения из правого хвоста распределения объектам из тестовой выборки. Возможным решением является некоторое упрощение задачи, которое заключается в том, чтобы предсказать, в каком интервале будет находиться целевая переменная для данного объекта. Для того, чтобы решить эту упрощенную задачу, нужно решить задачу многоклассовой классификации со смещенным балансом классов и целевой переменной 20.

6.2.2 Построение модели классификации

Одной из возможных причин некачественного обучения векторного представления для слов, упомянутых в 6.1.4, является слишком большая мощность словарей. В особенности это касается словаря ответов респондентов, так как он включает

словоформы из живой речи носителей языка. Эти словоформы не были нормализованы для сохранения рисунка исходной ассоциативной сети. Однако из-за того, что почти все они употреблялись слишком редко, модель оказалась не в состоянии выучить для них векторное представление, так как просто не было достаточно данных, из-за чего использовалось почти не измененное значение, данное при инициализации - выборка из многомерного нормального распределения.

Решение этой проблемы, которое в дальнейшем используется в работе, заключается в использовании символов как неделимых токенов для построения векторного построения. То есть теперь векторное представление строится для символов, из которых состоят лексемы (cue, response), а затем это векторное представление используется для построения более сложного векторного представления лексем. При использовании такого подхода сохраняется рисунок исходной сети - не нужно проводить нормализацию слов, опасаясь большой мощности словаря, так как словарь, для которого будет строиться векторное представление в любом случае состоит из алфавита и некоторых других символов. Также автоматически решается проблема с недостаточным количеством данных для обучения векторного представления.

Предобработка данных для модели классификации почти никак не отличается от описанной в 6.1, за исключением того, что из лексем (cue, response) были удалены все символы не являющиеся алфавитными или тире. Это было сделано из предположения, что основной смысл все же переносится с помощью алфавитных символов. Однако это привело к тому, что в данных появились дубликаты (cue, response) с разными значениями целевой переменной. Это происходило по причине того, что ответами респондентов могли оказаться одни и те же слова, отличающиеся неалфавитным символом (etc и etc. в исходной сети являются разными узлами с разными весами). Во избежании попадания шума в обучающую и тестовую выборки, было решено оставлять из таких дубликатов объект с большим порядковым номером класса, чтобы еще больше не сместить распределение классов.

Как было сказано ранее, на основе векторного представления для символов строится более сложное представление для лексем. Для символов, составляющих cue и response, в модели конструируются две отдельные матрицы для векторного представления, далее векторное представление каждой из лексем (cue, response) пода-

ется на вход отдельной двунаправленной рекурентной сети, а результат работы рекурентной сети взвешивается с помощью attention (разный для cue и response), выходы из каждого attention соединяются и подаются на вход полносвязной сети.

Таким образом, построение векторного представления для лексем происходит отдельно для cue и response в симметричных ветках (инициализируются они, конечно, по-разному), строение которых более подробно можно описать так:

1. Так как словарь для cue и response одинаковый, то матрицы для векторных представлений символов cue и response имеют одинаковые размеры, в отличие от 6.1; инициализация из нормального распределения.

   Так как на каждой итерации обучения алгоритму подается определенное количество объектов, это число объектов в дальнейшем в работе будет обозначаться $batch\_size$ для простоты изложения.

   Теперь обучение алгоритма происходит в парадигме последовательностей символов, каждая последовательность должна иметь фиксированную длину (специфика реализации алгоритма), далее эта длина будет обозначаться max_len.

   Таким образом, на этапе формирования векторного представления для символов, алгоритму подаются данные формы $[batch\_size, max\_len]$;

2. Для каждого символа из каждой последовательности формируется векторное представление на основе соответствующей матрицы векторного представления: символу с порядковым номером k в словаре соответствует k-ая строка матрицы; таким образом, после получения векторного представления форма данных $[batch\_size, max\_len, emb\_dim], emb\_dim-$ размерность векторного представления;

3. Рекурентную сеть можно в общем случае охарактеризовать уравнением

$$h_{t+1} = f(h_t * W_{rec} + x_{t+1} * W_{cur} + b) \tag{21}$$

   В рамках решения данной задачи используется разновидность рекурентной сети lstm [7], позволяющая избегать проблемы затухающих градиентов.

   Lstm обрабатывает последовательность токенов длины L слева направо и на

каждом шаге производит вектор (lstm hidden state) длины $hidden\_units$—это гиперпараметр. В данной работе используется еще одна модификация рекурентных сетей - двунаправленные сети, это две рекурентные сети, обрабатывающие последовательности с разных концов, так что итоговый вектор на каждом шаге обработки последовательности складывается из двух векторов.

В результате работы двунаправленной lstm получается набор lstm hidden states формы $[batch\_size, max\_len, 2 * hidden\_units]$;

4. Чтобы собрать максимальное количество информации из результата работы двунаправленной lstm в этой работе применяется механизм attention [8]; если обозначить lstm hidden states двух lstm из двунаправленной сети как $\overrightarrow{h}$ и $\overleftarrow{h}$, то итоговый bilstm hidden state получается как конкатенация этих двух $h = [\overrightarrow{h}, \overleftarrow{h}]$, во время обработки последовательности длины L получается $\{h_i\}_{i=1}^{L}$ и итоговый вектор формируется как

$$h_f = \sum_{i=1}^{L} \alpha_i * h_i \tag{22}$$

, где α_i параметризуются весами переменной attention;

5. Размерность h_f равна $2 * hidden\_units$, а следовательно размерность совокупности объектов после применения attention $[batch\_size, 2 * hidden\_units$;

После этого результаты веток (векторное представление лексем) соединяются и подаются на вход полносвязной сети, описанной в 6.1 почти без изменений, за исключением того, что теперь используется кросс-энтропия в качестве функции потерь:

$$CE = -\sum_{i=1}^{N} p_i * log(\hat{p}_i) \tag{23}$$

Следует заметить, что архитектура и наивная реализация функции ошибки никак не учитывают смещенный баланс классов. Поскольку решается задача многоклассовой классификации, метрикой качества для была выбрана accuracy(количество правильно сделанных предсказаний). Однако дисбаланс классов делает эту метрику уязвимой для неправильного толкования. Чтобы побороть проблему смещенного баланса классов были рассмотрены следующие варианты:

1. Выборка из датасета с искусственным балансом классов;

2. Взвешенная функция ошибки;

3. Иерархическая классификация;

В данной работе, в качестве реализации первого предложения (искусственный баланс классов) была проведена реализация одновременно двух стратегий: случайная подвыборка меньшего размера из самых частотных классов и аугментация классов с наименьшим числом объектов.

Вторая стратегия заслуживает отдельного упоминания, так как представляет собой попытку расширения исходной ассоциативной сети. Расширение классов было сделано исходя из предположения, что если в паре из исходной ассоциативной сети (cue, response) участвует существительное (не собственное) в единственном числе, то пара, составленная из множественной формы этого существительного и партнера без изменений, если не будут иметь такое же значение исходной целевой переменной, то, по крайней мере, будет принадлежать одному классу.

Необходимо заметить, что осуществление 1 пункта проводится только на датасете для обучения, откуда следует, что датасеты для обучения и валидации имеют разное распределение классов, что тоже необходимо учитывать при анализе полученных метрик.

Реализация второго пункта (взвешенная функция ошибки) требует пересмотра вида функции ошибки и способа вычисления весов для классов. Были использованы следующие выражения:

$$CE = -\sum_{i=1}^{N} w_i * p_i * log(\hat{p}_i) \tag{24}$$

$$w_j = \frac{N}{\#j * 10} \tag{25}$$

, где N - размер датасета для обучения, $\#j-$количество объектов класса j в обучении, 10 - количество классов в задаче.

Кроме этого, был рассмотрен вариант с нормализацией весов.

Реализация последнего предложения требует построения иерархической системы классификаторов. В данной работе, была рассмотрена не сложная система из одного бинарного и одного многоклассового классификаторов:

1. Бинарный классификатор разделяет объекты, которые принадлежат самому большому классу и все остальные;
2. Многоклассовый классификатор разделяет на оставшиеся классы те объекты, которые бинарный классификатор не посчитал принадлежащими к самому большому классу;

Стоит заметить, как все три предложения были реализованы:

1. Для первого пункта лишь перестраивается обучающий датасет, архитектура используется та же, какая была описана выше;
2. Для реализации второго пункта нужно перестроить только функции ошибки. Архитектура и датасет остаются без изменений;
3. Для реализации третьего пункта, в данной работе использовались точно такие же классификаторы, как в первом и втором пункте, без изменения архитектуры;

Граф общей архитектуры можно видеть на Рис. 8.

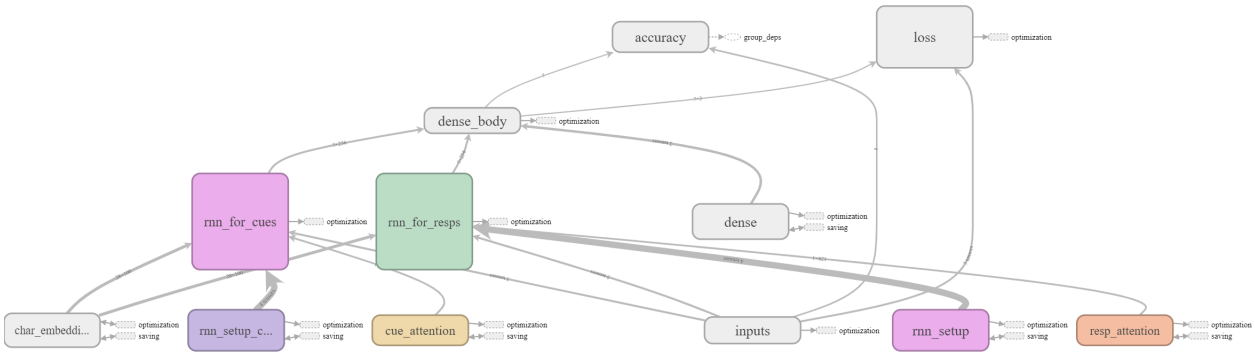

Рис. 8. Граф общей архитектуры многоклассового классификатора.

Результаты экспериментов для решения проблемы несбалансированности классов приведены в Табл. 1

Эксперимент	**Accuracy**	**Время**
Искуственный баланс классов	0.57	1
Взвешанная функция ошибки	**0.7**	**1**
Взвешанная функция ошибки (нормализация весов)	0.55	1
Иерархические классификаторы	0.66	2

Таблица 1. Результаты экспериментов для решения проблемы несбалансированности классов. Время включает в себя только время для обучения, расчитывается в часах.

6.2.3 Обогащение классификатора за счет использования TripletLoss

В разделе 6.1, как и в задаче классификации матрица для векторного представления слов (6.1) или символов (классификация) инициализировалась из нормального распределения. В этом разделе рассматривается попытка обогатить векторное представление символов в задаче классификации с помощью TripletLoss [9].

TripletLoss - это архитектура, которая позволяет учить богатое векторное представление различных объектов. Она состоит из энкодера и непосредственно функции ошибки TripletLoss, которая может быть записана как

$$TripletLoss = max(d(a, p) - d(a, n) + margin, 0) \tag{26}$$

, d - функция расстояния в пространстве векторных представлений объектов, a - anchor - объект из выборки, должен принадлежать определенному классу, p - positive - объект из выборки такого же класса, как и anchor, n - negative - объект из выборки, не должен принадлежать классу anchor.

Главная идея TripletLoss - увеличить расстояние в пространстве векторных представлений объектов для anchor и negative, и уменьшить - для anchor и positive.

В случае задачи многоклассовой классификации 6.2.2 в качестве энкодера для TripletLoss использовалась та же самая архитектура, описанная в 6.2.2, только без полносвязной сети, потому что ее вместе с 24 заменяет 26.

Формирование выборки происходит оффлайн (есть вариант формирования триплетов для TripletLoss онлайн): в каждом классе выбирается случайное подмножество и затем перебираются все возможные тройки (триплеты).

Стоит отметить, что в реализации TripletLoss для задачи классификации учиты-

ваются только hard - или semihard триплеты - это те, на которых значение 26 больше 0, то есть тройки, на которых и так выполняется задача TripletLoss, не влияют на процедуру обучения.

После окончания обучения из энкодера извлекаются обученные векторные представления для символов. Затем, они используются как инициализация в задаче многоклассовой классификации.

6.2.4 Использование результатов классификации

Переход от задачи регрессии к задаче классификации был обоснован тем, что нужно научиться решать более простую задачу. Научившись решать с определенным качеством задачу распределения пар (cue, response) по непересекающимся полуинтервалам, можно использовать векторные представления пар (это объединенные векторные представления лексем сразу перед полносвязной сетью в 6.2.2) для решения задачи регрессии.

В 6.1 было показано эмпирически, что хорошо аппроксимировать смещенное распределение минимизацией средней абсолютой ошибки - не просто с помощью использованной в 6.1 модели.

В этой секции в качестве алгоритма для решения задачи регрессии целевой переменной $P(\hat{w}|w)$ на векторных представлениях пар (cue, response), полученных классификатором, обученном на парах (cue, response) и целевой переменной 20, будет использоваться бустинг, которому свойственнено аппроксимировать любое распределение.

Процедура обучения и тестирования бустинга следующая:

1. Через обученный классификатор пропускается обучающий датасет пар (cue, response), чтобы получить векторные представления этих пар;

2. На этих представлениях и исходной целевой переменной $P(\hat{w}|w)$ обучается бустинг;

3. С помощью классификатора получаются векторные представления для тестовых пар (cue, response);

4. Бустинг тестируется с помощью полученных тестовых представлений и ис-

ходной целевой переменной;

Результаты экспериментов можно видеть в Табл. 2.

Эксперимент	**MAE**	$\mathbf{R^2}$
LGBM	0.0046	0.67
LGBM + TripletLoss	**0.0043**	**0.68**

Таблица 2. Результаты экспериментов по решению задачи регрессии на исходной целевой переменной. LGBM - реализация бустинга от Facebook.

7 Дальнейшее исследование

Анализируя способность алгоритмов 6.1, 6.2.4 восстанавливать собранные вручную ассоциативные сети, можно сказать, что такой подход обладает несколькими ключевыми особенностями:

1. Несимметричность относительно порядка слова в парах (cue, response): важное свойство, которое выделяет построение ассоциативных языковых сетей из ряда задач обработки естественного языка. В 6.1 несимметричность возникает из-за строго порядка конкатенации векторного представления cue и response и матричного умножения, в 6.2.4 несимметричность обусловлена тем, что векторное представление для cue и response конструируется в параллельных ветках;

2. Непосредственная ориентированность на восстановление вручную собранных ассоциативных сетей: оба алгоритма пытаются корректно восстановить ассоциативные меры между словами, анализируя совместное распределение информации, которую можно достать из слов, и ассоциативных оценок;

3. Устойчивость оценок: алгоритмы не зависят от печатных источников, так как не опираются на семантические векторные представления, в которых можно наблюдать влияние субъективных оценок (текст одного автора) или влияние времени на взаимоотношения слов (данные для ручных сетей собираются в течение нескольких лет и за этот промежуток времени влияние времени не сильно заметно на языке, хотя этот вопрос все еще открыт);

4. Возможность восстанавливать ассоциативные оценки между любыми двумя

лексемами: это в первую очередь об алгоритме, описанном в 6.2.4, так как он использует векторное представление символов, чтобы строить векторное представление лексем для пар (cue, response) для оценки ассоциативной меры, и не зависит от фиксированного словаря лексем;

5. Возможность улучшать обобщающую способность: при наличии новых данных оба алгоритма можно дообучить на них и вложить в них новую информацию;

Однако существуют и недостатки:

1. Относительно не высокие метрики качества (это можно частично объяснить тем, что задача напрямую связана с ментальным лексиконом человека, который сам по себе устроен очень сложно даже для человеческого сознания);

2. Сложно интерпретировать результат: модели нелинейны относительно входных данных и чтобы обосновать решение модели нужно проводить глубокий анализ работы механизмов модели;

8 Заключение

В данной работе была рассмотрена задача автоматического построения языковых ассоциативных сетей, а также предложено два алгоритма, моделирующих распределение ассоциативных оценок на множестве пар слов (cue, response), использующих методы глубокого и машинного обучения. С помощью данных алгоритмов можно конструировать сколь угодно масштабные языковые ассоциативные сети сравнимые с теми, которые собираются вручную.

9 Приложение 1

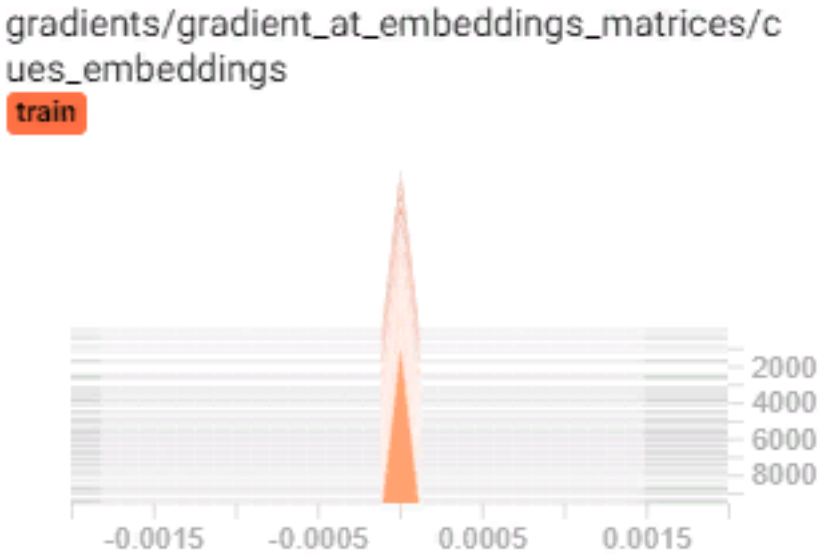

Рис. 9. Распределение градиентов во времени на матрице векторного представления cue. На оси справа отложены номера итераций, на оси спереди - значения градиента.

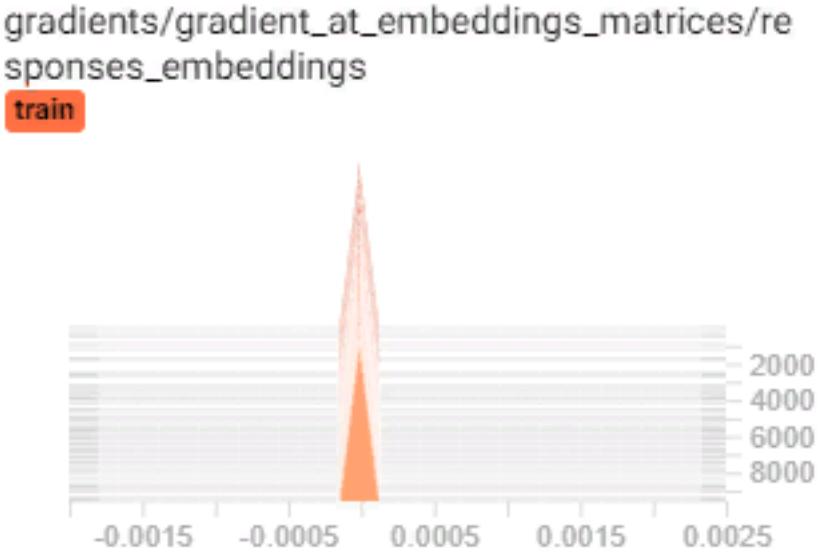

Рис. 10. Распределение градиентов во времени на матрице векторного представления response. На оси справа отложены номера итераций, на оси спереди - значения градиента.

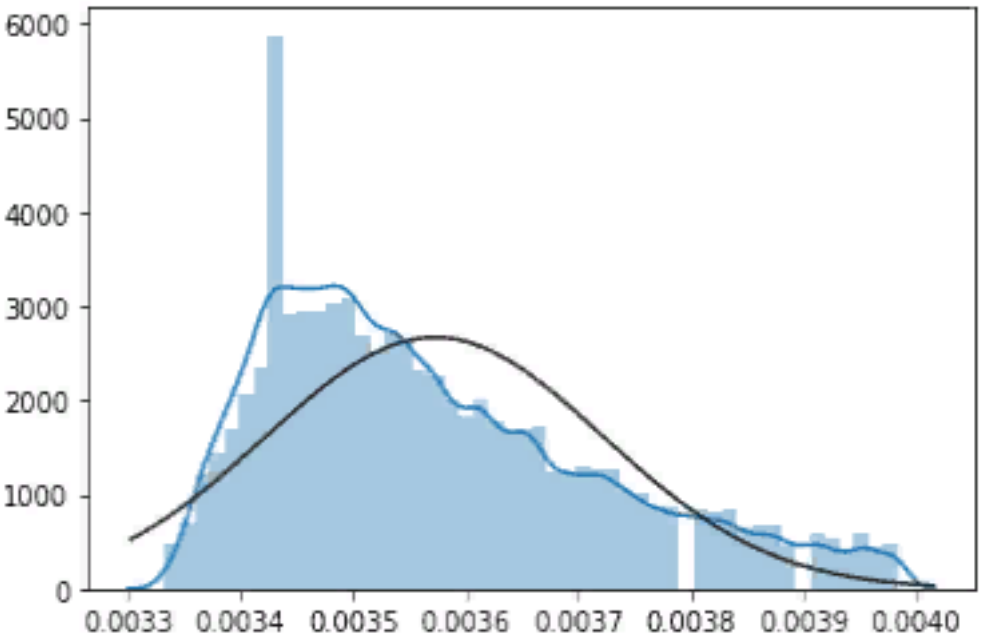

Рис. 11. Распределение целевой переменной внутри полуинтервала [0.0033, 0.004). На оси абсцисс расположены значения целевой переменной, на оси ординат - количество объектов с таким значением. Черной линией показана аппроксимация нормальным распределением

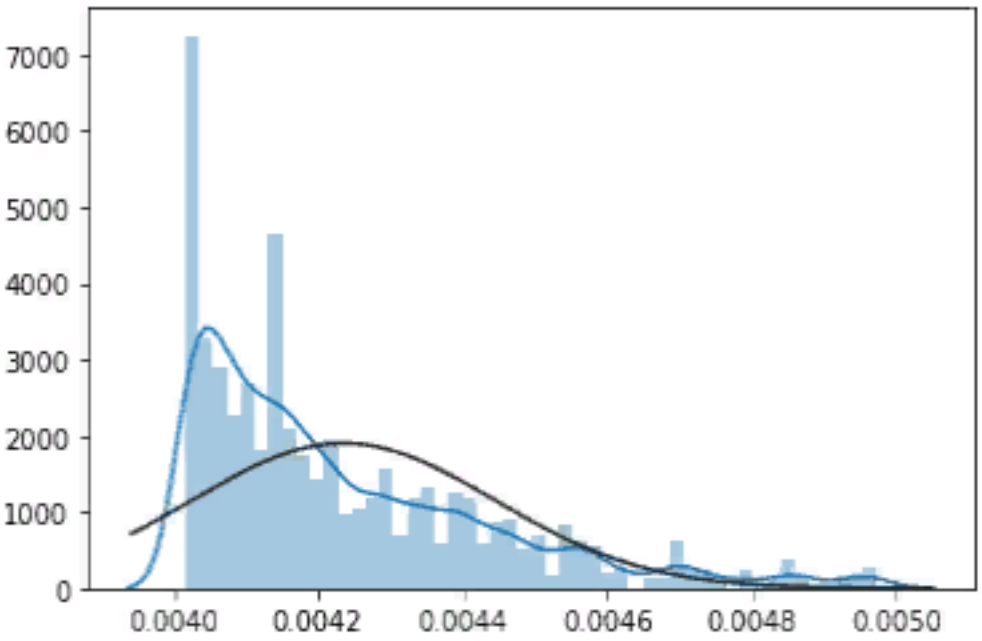

Рис. 12. Распределение целевой переменной внутри полуинтервала [0.004, 0.005). На оси абсцисс расположены значения целевой переменной, на оси ординат - количество объектов с таким значением. Черной линией показана аппроксимация нормальным распределением

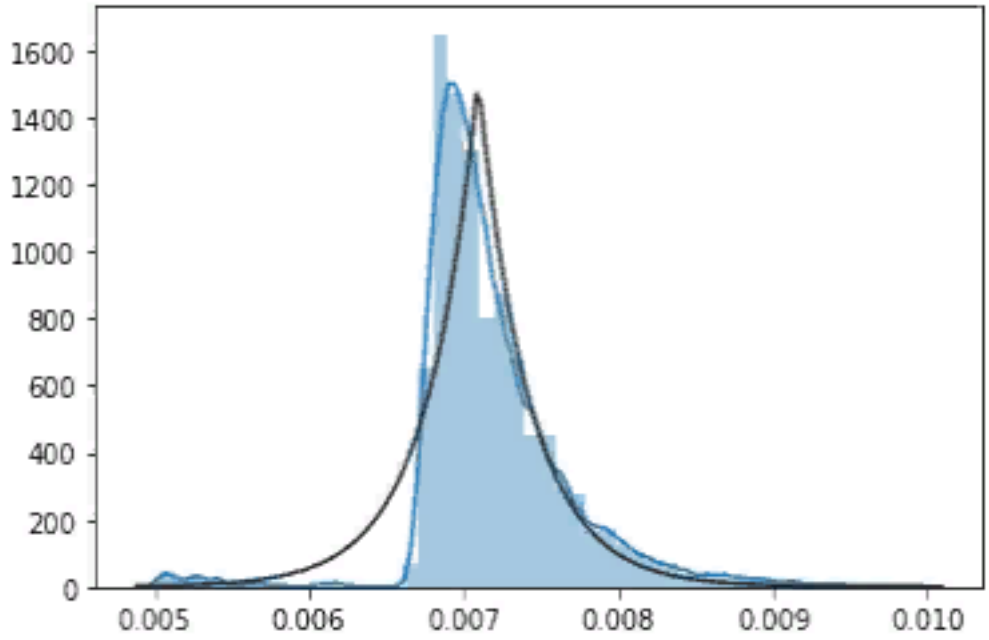

Рис. 13. Распределение целевой переменной внутри полуинтервала [0.005, 0.01). На оси абсцисс расположены значения целевой переменной, на оси ординат - количество объектов с таким значением. Черной линией показана аппроксимация распределением Лапласа.

Список литературы

[1] Mikolov T., Chen K., Corrado G., Dean J. Efficient Estimation of Word Representations in Vector Space. URL: https://arxiv.org/abs/1301.3781.

[2] Pennington J., Socher R., Manning C.D. GloVe: Global Vectors for Word Representation. URL: https://nlp.stanford.edu/pubs/glove.pdf.

[3] Douglas L.N., Mcevoy C.L., Schreiber T.A. The University of South Florida free association, rhyme, and word fragment normsBehavior Research Methods, Instruments, Computers, 36 (3), 402–407, 2004.

[4] Galea D., Bruza P. Deriving Word Association Networks from Text Corpora. URL: http://ceur-ws.org/Vol-1419/paper0038.pdf.

[5] Deyne S.D., Navarro D.J., Perfors A., Brysbaert M., Storms G. The "Small World of Words" English word association norms for over 12,000 cue words. Behavior Research Methods. DOI 10.3758/s13428-018-1115-7, (2018)

[6] He K., Zhang X., Ren S., Sun J. Delving Deep into Rectifiers:

Surpassing Human-Level Performance on ImageNet Classification. URL: https://arxiv.org/abs/1502.01852.

[7] Hochreiter S., Schmidhuber J. Long short-term memory. Neural computation, MIT Press, 1997.

[8] Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A.N., Kaiser L., Polosukhin I. Attention Is All You Need31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA. 2017

[9] Schroff F., Kalenichenko D., Philbin J. FaceNet: A Unified Embedding for Face Recognition and ClusteringProceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. 2015