Аннотация

Современные методы обработки естественного языка основаны на моделях векторного представления слов. Одна из особенностей этих моделей состоит в том, чтобы решать задачу пропорциональной аналогии. Данная задача неоднократно исследовалась для неконтекстуализированных моделей. В этой работе исследуются аффинные преобразования параллельного переноса для контекстуализированных моделей. Проводились численные эксперименты с векторным представлениями моделей ELMo и BERT, которые показали, что параллельный перенос не всегда дает возможность получить верное решение.

Abstract

Modern methods of natural language processing (NLP) are based on models of vector representation of words. One of the features of these models is to solve the problem of proportional analogy. This problem has been repeatedly investigated for non-contextualized models. In this paper, we investigate affine transformations of parallel transport for contextualized models. Numerical experiments were carried out with vector representations of the ELMo and BERT models, which showed that parallel transfer does not always make it possible to obtain the correct solution.

Содержание

Введение

В последние годы технологии машинного обучения стали неотъемлемой частью нашей жизни. Они представлены голосовыми помощниками, рекомендательными системами, умными домами, умными автомобилями и другими системами. Важной частью этих систем являются модули, которые помогают сделать понятным для компьютера то, что от него требуется. Для систем по обработке текста это модули обработки естественного языка или Natural Language Processing (NLP).

Компьютер без дополнительной помощи не способен обрабатывать естественный текст, зато компьютер хорошо работает с числами. Поэтому для того, чтобы «подружить» вычислительную машину с текстом, нужно представить текст в виде чисел или в виде многомерных векторов. Эти вектора также называют эмбеддингами.

Модели, использующие данный принцип, называются моделями векторного представления слов. В основе большей части данных моделей лежит гипотеза дистрибутивности [1]. Эта гипотеза заключается в том, что слова со схожим смыслом встречаются в похожих контекстах.

Прорывной и наиболее известной моделью векторного представления слов является выпущенная в 2013 году модель word2vec [2]. Было представлено две архитектуры модели нейронной сети word2vec: Continuous Bag-of-Words (CBOW) и Skip-gram. Continuous Bag-of-Words дословно переводится, как «непрерывный мешок слов». Работает архитектура похожим образом, предсказывается вероятность появления слова по его контексту в виде окна фиксированного размера. Архитектура Skip-gram, наоборот, предсказывает вероятность появления контекста у заданного слова. Порядок слов в контексте не влияет на результат ни в одном из этих алгоритмов. В процессе обучения модель корректирует веса между входным и скрытым слоем, которые в дальнейшем станут эмбеддингами слов.

Оказалось, что полученные векторные представления слов скрывают в себе семантические отношения между словами. Это хорошо заметно на примере задачи по построению пропорциональной аналогии. Эту задачу можно сформулировать так: «какое слово *d* относится к слову *c* так, как слово *b* относится к слову *a*». В модели word2vec это отношение можно выразить в виде разницы векторов. Для слов *a* и *b* с соответствующими им векторами v_a и v_b вектор разности $v_a - v_b$ будет характеризовать семантическую связь между словами. Тогда решением задачи

пропорциональной аналогии будет выражение $v_d - v_c = v_b - v_a$, где v_d и v_c эмбеддинги слов *d* и *c* соответственно.

Из полученного выражения получаем: $v_d = v_c + v_b - v_a$. Но вероятность того, что полученный вектор v_d совпадает с вектором какого-либо слова крайне мала, поэтому в качестве ответа берется слово с вектором наиболее близким к v_d, формула (1).

$$v_d = \underset{v'}{argmax} \cos(v', v_c + v_b - v_a) \quad (1)$$

Данная задача для модели word2vec исследовалась в работе [3]. Где автор пришел к выводу, что эта модель не всегда дает правильный ответ для задачи пропорциональной аналогии.

В настоящее время появляется все больше моделей для обработки естественного языка.

В 2017 году в исследованиях [4] и [5], а позже и в [6] была предложена модель контекстуализированной модели обработки естественного языка ELMo (Embeddings from Language Models). Если в модели word2vec векторное представление слов было одним и тем же независимо от контекста, то в ELMo решается эта проблема. Для каждого контекста будет свой эмбеддинг. В основе архитектуры ELMo лежат блоки долгой краткосрочной памяти (LSTM - Long Short-Term Memory). Данные блоки расположены в прямом и обратном направлениях для того, чтобы при создании эмбеддинга учитывался контекст до и после слова.

Вскоре после выхода ELMo вышла модель BERT [7] или Bidirectional Encoder Representations from Transformers. BERT – это модель, побившая несколько рекордов по успешности решения ряда NLP-задач, например, BERT показала лучшее качество на тесте SQuAD 1.1 [8]. BERT также контекстуализированная модель. Архитектура модели BERT представляет из себя последовательность двунаправленных кодировщиков из Transformer [9]. В основе обучения модели лежат две идеи:

1. Первая состоит в том, чтобы заменить 15% текста масками и заставить модель предсказывать пропущенные слова.
2. Вторая идея заключается в том, чтобы научить модель оценивать насколько одно предложение является логичным продолжением второго.

Успех модели, помимо хорошего качества, можно объяснить тем, что код модели был выложен в открытый доступ, а также были выложены различные

модели, предобученные на больших объемах данных. Это дало возможность всем разработчикам встроить модель BERT в свои модели машинного обучения для обработки естественного языка.

ELMo и BERT - контекстуализированные модели, это значит, что векторное представление одного и того же слова будет отличаться в зависимости от его контекста. Отсюда возникает вопрос, возможно ли провести аффинные преобразования в семантическом пространстве модели BERT?

Целью данной работы является исследование аффинных преобразований для модели BERT и определение точности этих преобразований.

Для достижения поставленной цели потребовалось решить следующие **задачи**:

1. Исследование моделей векторного представления слов;
2. Исследование методов оценки аффинных преобразований;
3. Разработка метода оценки точности параллельного переноса для контекстуализированных моделей;
4. Подготовка экспериментальных данных;
5. Проведение экспериментов;
6. Оценка полученных результатов.

1 Обзор литературы

1.1 Появления моделей векторных представлений слов

Векторное представление слов является одним из ключевых инструментов в обработке естественного языка. Основная идея заключается в том, чтобы сопоставить каждому слову вектор определенной величины.

Самой простой реализацией модели векторного представления слов является one-hot encoding. Идея этой модели заключается в том, что в наборе из *K* слов каждому слову k_i сопоставить вектор v_i длиной *K* со всеми нулями и одной единицей в позиции *i*, где *i* - это номер слова во всем наборе (2).

$$v_i^j = \begin{cases} 1, i = j \\ 0, i \neq j \end{cases}, j = 1, ..., K \tag{2}$$

Недостатком этого метода является то, что по данным векторным представлениям нельзя судить о семантической схожести слов. Также при обработке реальных текстов размер словаря будет очень большим, а значит и длина векторных представлений будет очень большой, данные вектора неэффективно хранить в памяти. Также у многих алгоритмов машинного обучения могут возникнуть сложности с обработкой разреженных векторов.

Позднее появились более продвинутые модели векторного представления. В этих реализациях уже наблюдаются семантические отношения между эмбеддингами слов. Модели векторного представления делятся на две группы в зависимости от используемых методов.

Первая группа - статистические модели векторного представления. При работе этих моделей строится матрица совместной встречаемости слов, далее эта матрица подвергается сингулярному разложению. Одна из полученных после разложения матриц содержит вектора слов.

Вторая группа - предиктивные модели. Данные модели для создания эмбеддинга используют контекст, используются слова, попадающие в окно определенного размера вокруг интересующего нас слова. Для работы используются нейронные сети.

В данный момент большей популярностью пользуются предиктивные модели. Большая популярность пришла к этим моделям с появлением word2vec.

1.2 Word2vec

Идея создания векторов в word2vec [2] основана на предположении о контекстной близости, а именно на том, что слова встречающиеся в одинаковых контекстах скорее всего имеют схожее значение.

Модель word2vec является простейшей нейронной сетью с обратным распространением ошибки. У модели один скрытый слой. В основе обучения модели лежит идея, что тренировать можно не только на контексте из предыдущих слов, но также использовать слова, после целевого слова. При этом порядок слов в контексте не учитывается.

Существует 2 метода обучения word2vec: CBOW (Continuous Bag of Words) и Skip-gram. Основная цель архитектуры Continuous Bag-of-Word состоит в том, чтобы предсказать пропущенное слово по его контексту. В Skip-gram предсказывается контекст по целевому слову.

На рисунке 1 представлены изображения архитектур CBOW и Skip-gram. Где V – мощность словаря, x – слова, подающиеся на вход нейронной сети, закодированные методом one-hot encoding (2) для словаря мощностью V, N – количество нейронов в скрытом слое и y – результат работы функции активации softmax.

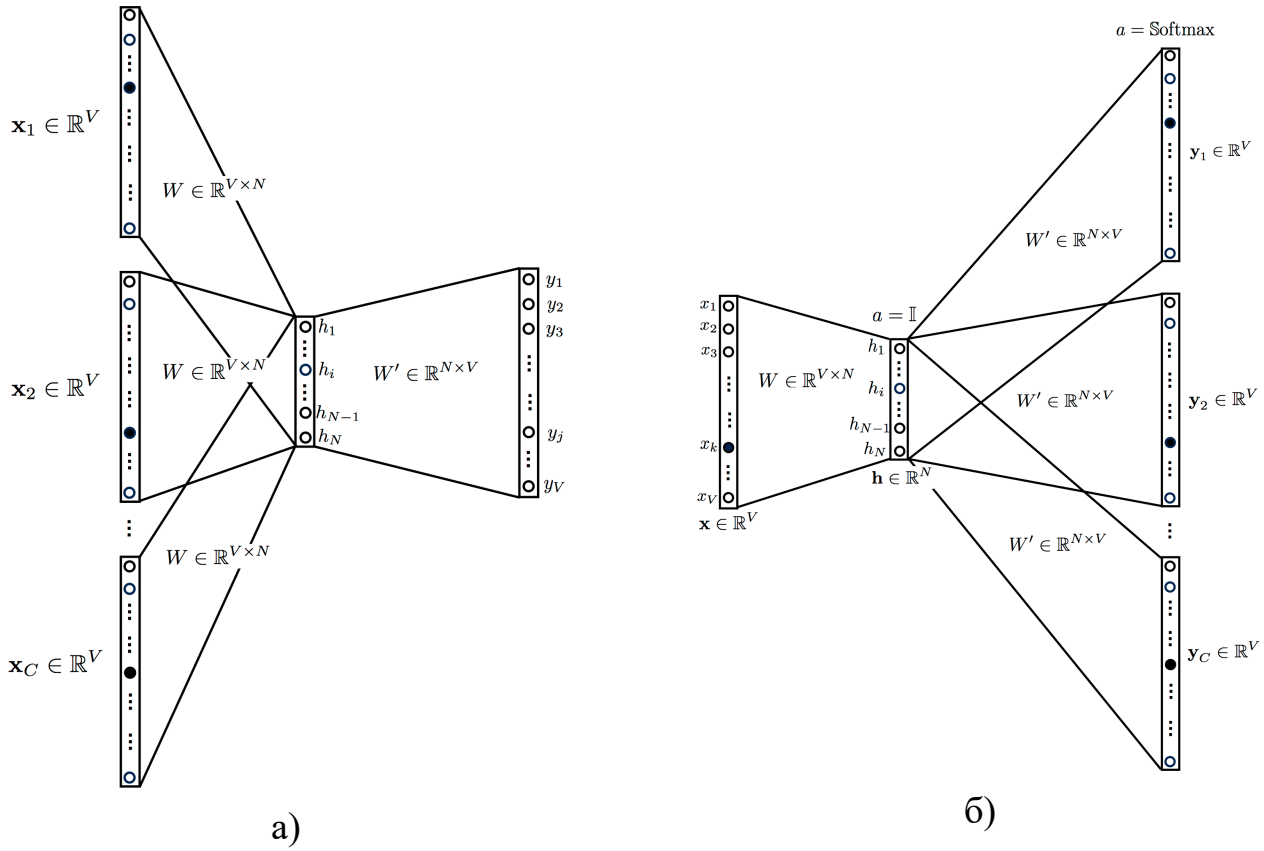

Рис. 1. Схемы архитектур CBOW (а), Skip-gram (б)

Архитектуры представляют из себя полносвязные нейронные сети с обратным распространением ошибки. Для больших словарей метод активации softmax накладывает серьезные вычислительные нагрузки. При работе softmax каждое слово представляет из себя отдельный класс, авторы word2vec предложили использовать бинарную классификацию вместо многоклассовой. Предлагается научить модель отличать пары слов, которые встречаются в одном контексте, от тех, которые никогда не стоят в одном контексте.

Появление модели word2vec послужило мощным толчком для развития моделей обработки естественного языка.

1.3 FastText

FastText [10] является продолжением развития модели word2vec. FastText имеет архитектуру Skip-gram. При этом в fastText отличается от word2vec тем, что у новой модели используются N-граммы символов. Например, для слова молоко 3-граммами являются <мо, мол, оло, лок, око, ко>, где символы « < » и « > » кодируют начало и конец слова соответственно. Векторные представления строятся именно для N-грамм, векторные представления слов - это сумма векторных представлений всех его N-грамм. При этом решается проблема того, что словарь модели word2vec был ограничен и не все формы слов вошли в словарь. Также, использование N-грамм позволяет получать векторные представления для редких слов.

В русском языке существуют слова омонимы и омографы, слова, которые совпадают в написании, но имеют разный смысл. Предыдущие методы обработки естественного языка никак не решали эту проблему. Одной из первых эту проблему постаралась решить модель ELMo.

1.4 ELMo

ELMo (Embeddings from Language Models) – модель обработки естественного языка, которая представляет собой двунаправленную рекуррентную нейронную сеть с LSTM. Модель была предложена в работах [4–6]. Модель учитывает семантическую неоднозначность слов в предложениях, и эмбеддинги, присваиваемые словам, зависят не только от самого слова, но и от контекста. Основная идея получения эмбеддингов – использование скрытых состояний LSTM. Разберемся поподробнее с LSTM блоками.

1.4.1 LSTM

LSTM или Long short-term memory были предложены в статье [11]. Название блоков дословно переводится, как долгая краткосрочная память. Данные блоки являются разновидностью архитектуры рекуррентных нейронных сетях и предназначены для того, чтобы хранить информацию на длинные и короткие промежутки времени.

Данные блоки имеют одну особенность, в них нет функции активации. За счет этого хранимая информация не размывается по времени и во время обучения при использовании метода обратного распространения ошибки вычисляемый градиент не исчезает.

В LSTM модуле есть 2 основных компонента: состояние ячейки и различные фильтры. Состояние ячейки – это память сети, которая передается по всей цепочке.

Во время обучения состояние ячейки постоянно меняется. Происходит добавление и удаление информации. Все это контролируют фильтры. Фильтры состоят из сигмоидальной нейронной сети и операции поточечного умножения. Сигмоидальный модуль возвращает числа в диапазоне [0;1], которые обозначают долю блока информации, которую следует пропустить дальше по сети. Фильтры бывают трех типов:

1. Забывания;
2. Входные;
3. Выходные.

На рисунке 2.а изображен фильтр забывания. На данном этапе решается какую информацию можно забыть или оставить. h_{t-1} – значения выхода из предыдущего блока, x_t – вход данного блока. Данные значения проходят обработку в сигмоидальном блоке. Результаты находятся в диапазоне [0;1]. То, что ближе к 0 будет забыто, что ближе к 1 оставлено (3).

$$f_t = \sigma(W_f \cdot [h_{t-1},x_t] + b_f) \tag{3}$$

На рисунке 2.б решается какая информация будет храниться в состоянии ячейки. Сигмоидальный блок решает какую информацию необходимо обновить (4), tanh-слой строит вектор со значениями, которые могут быть добавлены в состояние ячейки (5).

$$i_t = \sigma(W_i \cdot [h_{t-1},x_t] + b_i) \tag{4}$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1},x_t] + b_C) \tag{5}$$

На рисунке 2.в проиллюстрирован процесс изменения состояния ячейки. Ненужная информация в C_{t-1} забывается после умножения на f_t. Затем к состоянию ячейки добавляются текущие изменения $i_t * \tilde{C}_t$. Все вместе можно записать формулой (6).

$$\tilde{C}_t = f - t * C_{t-1} + i_t * \tilde{C}_t \tag{6}$$

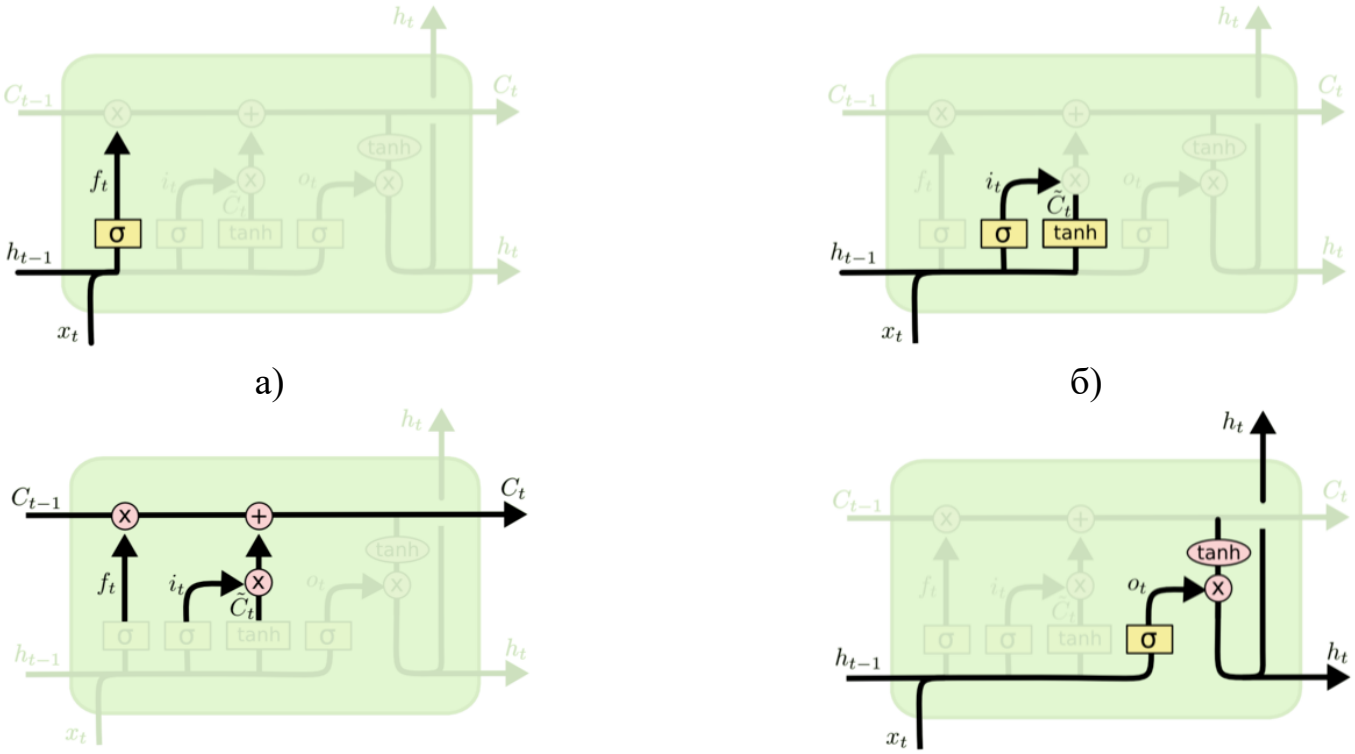

Рис. 2. Виды фильтров: a) слой фильтра забывания, b) слой входного фильтра, c) применение текущих изменений, d) слой выходного фильтра.

На рисунке 2.г изображено формирование выходов. Сначала в сигмоидальном слое поступает информация предыдущего выхода h_{t-1} и текущего входа x_t, где определяется какая информация из состояния ячейки будет отправлена на выход (7). Далее значения из состояния ячейки обрабатываются tanh-слоем и перемножаются со значениями с сигмоидального слоя (8).

$$o_t = \sigma(W_o \cdot [h_{t-1},x_t]) \tag{7}$$

$$\tilde{h}_t = o_t * \tanh(C_t) \tag{8}$$

После преобразований h_t и C_t передаются на следующий блок по цепочке.

1.4.2 Структура ELMo

Архитектура ELMo представлена на рисунке 3.а. ELMo состит из двух двухслойных разнонаправленных рекуррентных LSTM сетей. Было замечено, что верхние слои LSTM сети отвечают за семантический смысл слова, а нижние – за синтаксис и грамматику.

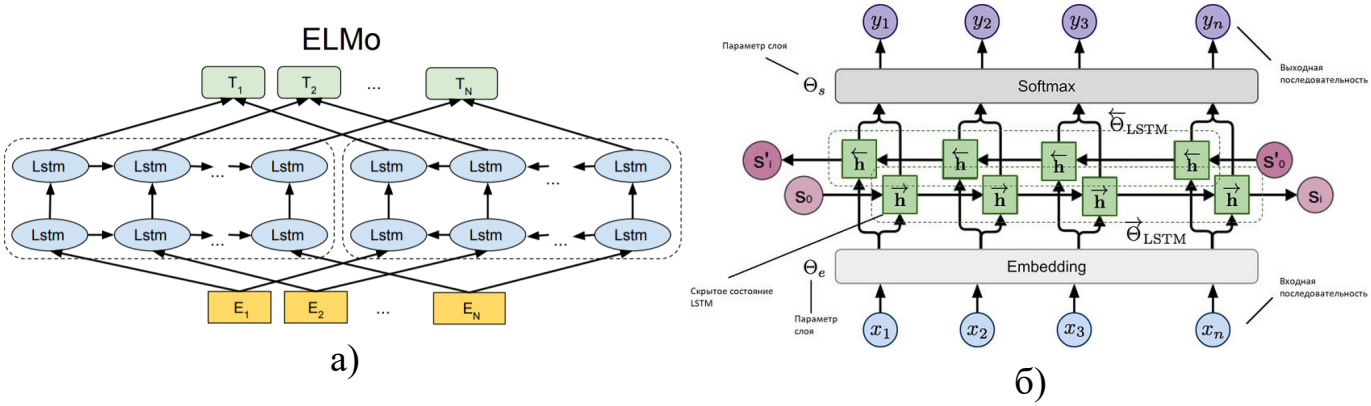

Рис. 3. Архитектура модели ELMo (а), принцип работы модели ELMo (б)

На рисунке 3.б схематично изображен принцип работы модели ELMo. Пусть на вход подаются токены $t_1, ..., t_N$, на которые разделено предложение. Будем считать логарифм правдоподобия метки слова в обоих направлениях, учитывая контекст справа и слева от слова. Так модель предсказывает вероятность следующего токена с учетом истории.

Пусть есть L слоев сети, входные и выходные данные представляются в виде векторных представлений слов. Каждый результирующий вектор будет считаться на основании множества (9).

$$\left\{ x_k^{LM}, \overrightarrow{h_{k,j}^{LM}}, \overleftarrow{h_{k,j}^{LM}} | j = 1, ..., L \right\} = \left\{ h_{k,j}^{LM} | j = 1, ..., L \right\} \tag{9}$$

где x_k^{LM} – входящий токен, $\overrightarrow{h_{k,j}^{LM}}, \overleftarrow{h_{k,j}^{LM}}$ – скрытые слои в обоих направлениях.

Результатом работы ELMo будет выражение (10).

$$ELMO_k^{task} = \gamma^{task} \sum_{j=0}^{L} s_i^{task} h_{k,j}^{LM} \tag{10}$$

где γ^{task} – масштабирующий коэффициент, который регулирует то, как могут отличаться по норме ембеддинги слов, s_i^{task} – обучаемые параметры, нормализованные функцией softmax.

Векторным представлением слова будет являться взвешенная сумма значений на всех скрытых слоях ELMo.

Вскоре после выхода ELMo появилась еще одна мощная контекстуализированная модель, называется она BERT.

1.5 BERT

BERT [7] или Bidirectional Encoder Representations from Transformers – модель обработки естественного языка выпущенная в 2018 году командой Google AI. В основе работы модели лежит ряд недавних разработок в сфере обработки естественного языка. Можно выделить разработки Semi-supervised Sequence learning [12], уже упомянутый ELMo [4–6], ULMFiT [13], OpenAI Transformer [14] и Transformer [9].

Принцип работы модели BERT основан на применении кодировщиков (encoders) из Transformer. Рассмотрим подробнее принцип работы кодировщиков.

1.5.1 Encoder from Transformer

Кодировщик – это составная часть модели трансформера, подробно он описан в статье [9]. Принцип работы модели трансформера основан на механизме внутреннего внимания. Работа модели состоит в том, что модель принимает на вход предложение на одном языке и выводит на другом. Другими словами, осуществляется перевод предложения с одного языка на другой.

Важным преимуществом модели трансформера является то, что все вычисления могут выполняться параллельно, в том числе и при использовании TPU.

Модель состоит из двух компонентов: кодирующего и декодирующего. Предложение сначала попадает в кодирующий компонент, далее передается в декодирующий компонент, на выходе из декодирующего компонента получается переведенное предложение.

Каждый компонент состоит из более мелких модулей (рисунок 4). Кодирующий компонент состоит из кодировщиков (encoders), декодирующий – из декодировщиков (decoders). По структуре все модули в каждом компоненте

идентичны, разницу составляют только веса, получаемые при обучении модели. Количество модулей в каждом компоненте одинаковое.

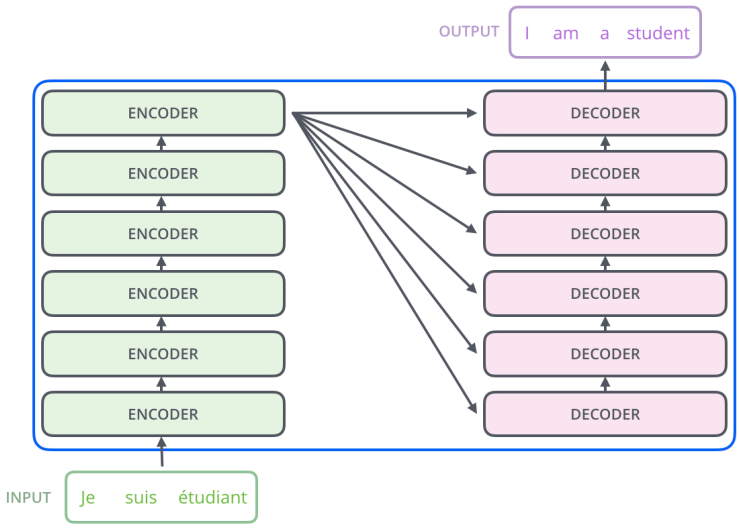

Рис. 4. Структура модели Transformer

Кодировщик состоит из двух подслоев (рисунок 5). Первый слой – слой внутреннего внимания (self-attention), данный блок помогает модели работать с контекстом во время кодирования слова, подробнее механизм работы разберем позже. Второй слой – нейронная сеть прямого распределения (feed-forward neural network). По сути, второй слой представляет из себя полносвязную нейронную сеть, применяемую к каждому слову в отдельности.

Декодировщик состоит из тех же слоев (рисунок 5), но с одним исключением. Между слоем внутреннего внимания и слоем прямого распределения стоит слой внимания (Encoder-Decoder Attention), который помогает декодеру сосредотачиваться на более значимых частях предложения.

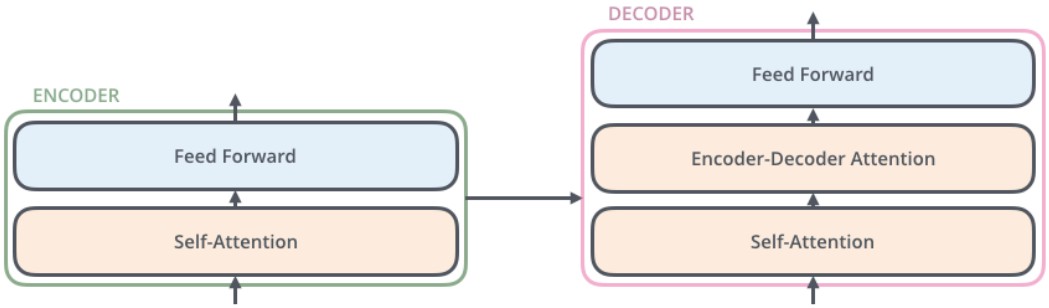

Рис. 5. Структура кодировщика и декодировщика

Рассмотрим алгоритм работы блока кодировщика. На вход блоку подается информация с выхода предыдущего блока, в случае первого блока, на вход подаются

неконтекстуализированные эмбеддинги слов. Эмбеддинги слов попадают в слой внутреннего внимания.

Векторные представления слов подаются в виде матрицы X размером $L * N$, где L – длина векторного представления, N - количество слов в предложении.

Первый этап – это создание трех матриц, матриц запроса (11), ключа (12) и значения (13).

$$Q = X * W_Q \tag{11}$$

$$K = X * W_K \tag{12}$$

$$V = X * W_V \tag{13}$$

где X – матрица эмбеддингов слов, Q, K, V – матрицы запроса ключа и значения соответственно, W_Q, W_K, W_V – матрицы весов, получаемые в процессе обучения моделей.

Следующим шагом необходимо получить для каждого слова в предложении коэффициент схожести S со всеми словами в этом же предложении (14). Далее полученные коэффициенты делятся на корень из размерности векторного представления $\sqrt{L}$ для получения более стабильного градиента. После этого полученные значения пропускают через функцию softmax для нормирования весов (15). Наибольшие коэффициенты будут у слов самим с собой, это нормально. На данном этапе необходимо определить какие из слов в предложении несут похожую смысловую нагрузку. Чем больше значение коэффициента, тем больше слова похожи друг на друга. Влияние нерелевантных слов необходимо свести к минимуму, подобрав им минимальные веса. Далее полученные веса M умножаются на матрицу значений V (16) и складываются. Полученная взвешенная сумма Z для каждого слова будет результатом работы слоя внутреннего внимания.

$$S = Q * K^T \tag{14}$$

$$M = softmax(\frac{S}{\sqrt{L}}) \tag{15}$$

$$Z = M * V \tag{16}$$

Если объединить все выражения (14) – (16), то получится следующее (17).

$$Z = softmax(\frac{Q * K^T}{\sqrt{L}}) * V \tag{17}$$

Данный метод внутреннего внимания совершенствуется добавлением механизма под названием: множественное внимание (multi-head attention). Этот прием позволяет улучшить производительность внутреннего слоя за счет повышения способности модели фокусироваться на разных позициях. При этом создается несколько подпространств со своими случайно сгенерированными матрицами запроса, ключа и значения. В модели трансформера используется 8 подпространств.

Результатом работы этих 8 подпространств внутреннего внимания будут 8 матриц весов Z. Для того, чтобы не увеличивать размерность данных предлагается конкатенировать все матрицы Z и умножить их на еще одну матрицу весов W_O.

Важная часть обработки естественного языка – обработка порядка слов во входящем предложении. В модели трансформер эта проблема решается путем добавления вектора позиционного кодирования. Эти вектора имеют определенный шаблон и добавляются к каждому входящему эмбеддингу. Позиционное кодирование позволяет определить позицию каждого слова или расстояние между словами в предложении. Идея метода в том, что добавление векторов к эмбеддингам создает осмысленное расстояние между векторными представлениями в процессе проецирования в $Q/K/V$ векторы и скалярного произведения при вычислении внимания.

В приложении А представлена визуализация одного из методов позиционного кодирования. Каждая строка – это один вектор, первому слову соответствует первая строка, второму – вторая и так далее. По центру есть разрыв, значения слева и справа сгенерированы разными функциями, слева используется функция с синусом, справа с косинусом.

Декодеры работают по-похожему принципу с небольшими изменениями. В слое внутреннего внимания скрываются все эмбеддниги, следующие за целевым векторным представлением. Упомянутый ранее промежуточный слой внимания Encoder-Decoder attention, помимо взвешенных сумм со слоя внутреннего внимания, принимает на вход матрицы ключей и значений (K,V) с выхода последнего кодировщика.

Во время обучения данные проходят по всему маршруту и результат сравнивается с эталонным. Все матрицы весов изначально задаются случайными

значениям и подбираются методом обратного распространения ошибки.

1.5.2 Структура BERT

В модели BERT для векторного представления слов используются только кодировщики. Структура модели представлена на рисунке 6. Модель состоит из 12 кодировщиков.

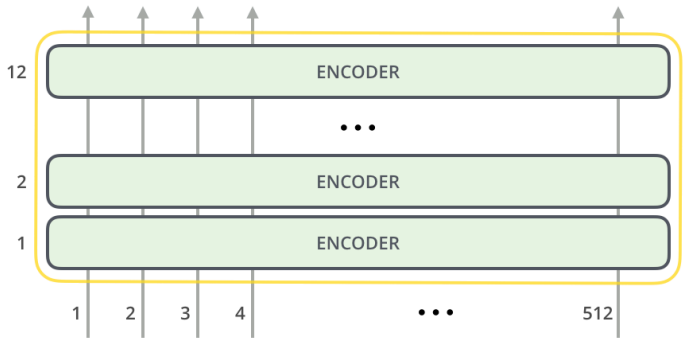

Рис. 6. Структура модели BERT

В процессе работы модель BERT принимает на вход по аналогии с трансформером набор слов, при этом в начало текста помещается специальный токен *[CLS]*, в конец каждого предложения помещается токен *[SEP]* для разграничения предложений.

При этом выход модели существенно отличается от трансформера. Трансформер выдает готовое переведенное предложение. BERT имеет на выходе только эмбеддинги слов, которые в последствии можно использовать в различный моделях машинного обучения, например, для задачи классификации текстов.

Модель BERT обучена для решения двух задач. Первая задача состоит в том, чтобы предсказывать пропущенные слова в предложении. Модель обучалась на текстах, в которых было скрыто 15% слов. Вторая задача заключается в том, чтобы определить насколько одно предложение является логичным продолжением второго.

Важным преимуществом модели BERT стало то, что при обучении не требуется размеченный датасет. Модель способна сама подготовить себе данные для обучения. Это позволило разработчикам обучить модель на больших объемах данных.

1.6 Исследование методов оценки аффинных преобразований

В основе большинства методов оценки аффинных преобразований лежит описанное в формуле (18) косинусное сходство. Для удобства применяется также косинусное расстояние (19).

$$similarity(A,B) = cos(A,B) = cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} \tag{18}$$

$$distance(A,B) = 1 - similarity(A,B) \tag{19}$$

Аффинные преобразования над векторными представлениями слов предполагают задачу пропорциональной аналогии. Задача пропорциональной аналогии заключается в том, чтобы установить для группы пар слов определенную логическую связь в парах. Рассмотрим группу пар слов $(a : a^*), (b : b^*)$. Известно, что слова в первой группе относятся друг к другу также, как и слова во второй группе. Представить это можно следующим образом (20).

$$(a : a^*) :: (b : b^*) \tag{20}$$

Если в парах действует одна и та же связь, то можно выразить одно из слов через 3 других слова (21).

$$y = v_b - v_a + v_{a^*} \tag{21}$$

где y - искомый вектор, v_x - эмбеддинг для слова x.

Но вектору y может не соответствовать ни одного слова, поэтому в качестве ответа берется ближайшее слово к y. Для корректно работающей модели ответ совпадет с b^*. Результат можно выразить формулой (22).

$$y^* = \underset{v \notin \{v_b, v_a, v_{a^*}\}}{argmax} \; cos(v, v_b - v_a + v_{a^*}) \tag{22}$$

Интуитивно понятным способом оценить качество будет сопоставить все пары слов друг с другом, посчитать одно из слов по формуле (22) [15] и найти среднее значение косинусной близости после трансформаций. Данный способ получил название **3CosAdd**.

Существует пердположение, что 23 эквивалентно (22). Этот метод получил

название **3CosMul** [15].

$$y^* = \underset{v \notin \{v_b, v_a, v_{a^*}\}}{argmax} \; cos(v, v_{a^*}) - cos(v, v_a) + cos(v, v_b) \tag{23}$$

Если в группе у пар одна и та же связь, то можно посчитать среднюю разницу между векторами в парах и задача аналогии будет решаться через посчитанную разницу. Эта идея предлагается в метрике **3CosAvg** (24) [16].

$$y^* = \underset{v \notin \{v_b, v_a, v_{a^*}\}}{argmax} \; cos(v, v_b + \frac{\sum_{i=1}^{n} v_{a_i^*}}{n} - \frac{\sum_{i=1}^{n} v_{a_i}}{n}) \tag{24}$$

Существуют и другие методы оценки аффинных преобразований. Метод **Only-b** [17] основан на предположении, что вектора v_b и v_{b^*} находятся максимально близко друг к другу (25).

$$y^* = \underset{v \notin \{v_b, v_a, v_{a^*}\}}{argmax} \; cos(v, v_b) \tag{25}$$

В методе **Ignore-a** [17] предполагается, что результат ближе всего к сумме векторов v_b и v_{a^*} (26).

$$y^* = \underset{v \notin \{v_b, v_a, v_{a^*}\}}{argmax} \; cos(v, v_b + v_{a^*}) \tag{26}$$

В методе **Add-opposite** [17] результатом является вектор, ближайший к выражению $-(v_{a^*} - v_a) + v_b$ (27).

$$y^* = \underset{v \notin \{v_b, v_a, v_{a^*}\}}{argmax} \; cos(v, -(v_{a^*} - v_a) + v_b) \tag{27}$$

Метод **MULTIPLY** [17] имеет следующий вид (28).

$$y^* = \underset{v \notin \{v_b, v_a, v_{a^*}\}}{argmax} \; \frac{cos(v, v_{a^*}) \; cos(v, v_b)}{cos(v, v_a)} \tag{28}$$

2 Теоретическая часть

2.1 Постановка задачи

Необходимо исследовать возможность и оценить качество аффинных преобразований при решении задачи пропорциональной аналогии в семантическом пространстве контекстуализированной модели векторного представления слов.

Пусть имеется множество слов W мощностью L. Каждому слову $w \in W$ соответствует множество векторных представлений $V_w = \{v_{1w}, v_{2w}, ..., v_{nw}\}|V_w \subset V$, где v_{iw} – векторное представление слова w в i-ом контексте, V множество всех векторных представлений слов. При этом существует функция η, которая преобразует множество эмбеддингов слова в векторное представление $v'_w = \eta(V_w)|v'_w \in V'$.

Выделим пары слов $(w_{11},w_{12}), (w_{21},w_{22}), ..., (w_{k1},w_{k2})$ из множества слов W, где w_{ij} – j-ое слово в паре под номером i. Все пары подобраны таким образом, что в них действует одно и то же семантическое отношение $R : w_{i1} \rightarrow w_{i2}$. Это означает, что можно можно составить пропорциональные аналогии слов: $(w_{i1}, w_{i2}) :: (w_{j1}, w_{j2}), i \neq j$. Каждой паре слов можно сопоставить пары векторов $(v'_{11},v'_{12}), (v'_{21},v'_{22}), ..., (v'_{k1},v'_{k2})$, где v'_{ij} – j-ый вектор в паре под номером i. . В итоге получаются следующие выражения:

$$
\begin{aligned}
&w \in W, |W| = L \\
&V_w = \{v_{1w}, v_{2w}, ..., v_{nw}\}|V_w \subset V \\
&v'_w = \eta(V_w) \\
&v'_w \in V', |V'| = L \\
&f : W \rightarrow V' \\
&P = \{(w_{i1}, w_{i2})|w_{i1}, w_{i2} \in W, i = \overline{1,k}\} \\
&R : w_{i1} \rightarrow w_{i2} \\
&V'_P = \{(v'_{i1}, v'_{i2})|v'_{i1}, v'_{i2} \in V', i = \overline{1,k}\} \\
&g = P \rightarrow V'_P
\end{aligned}
$$

Аффинное преобразование – это отображение пространства в себя, при котором параллельные прямые переходят в параллельные прямые, пересекающиеся – в

пересекающиеся, скрещивающиеся – в скрещивающиеся.

Математически аффинное преобразование $\varphi : \mathbb{R}^n \to \mathbb{R}^n$ выглядит выглядит как в выражении (29).

$$\varphi(x) = M \cdot x + v \tag{29}$$

где M – обратимая матрица, $v \in \mathbb{R}^n$.

Аффинные преобразования бывают разных видов: движения, растяжения и подобия. Задача пропорциональной аналогии относится к движению, типу аффинного преобразования, при котором сохраняются расстояния между любыми двумя точками. Математическая запись пропорциональной аналогии выглядит следующим образом:

$$\varphi(x) = x + v$$

Сформированная пара векторов $(v'_{i1}, v'_{i2}), i = \overline{1,k}$ является парой радиус-векторами для точек A_{i1}, A_{i2}. Необходимо установить существование аффинного преобразования, при котором все точки A_{i1} будут соответствовать точкам A_{i2}.

$$v'_{i2} = v'_{i1} + d$$

Это означает, что расстояние между любыми двумя точками $A_{i1}, A_{j1}, i \neq j$ будет точно таким же, как и между любыми двумя точками $A_{i2}, A_{j2}, i \neq j$. Но вероятность того, что при параллельном переносе попадется точка, для которой существует радиус вектор $v'_r \in V'$, крайне мала.

Однако есть решение, можно взять вектор максимально близкий к искомому, то есть для вектора $v''_r \notin V'$ сопоставить вектор $v'_r \in V'$:

$$v''_{i2} = v'_{i1} + d$$

С ограничением:

$$v'_{i2} = \underset{v \in V'}{argmax}\ cos(v, v''_{i2})$$

2.2 Метод исследования

2.2.1 Подготовка экспериментальных данных

Для получения эмбеддингов слов были взяты тексты из электронной библиотеки КиберЛенинка. Тексты двух жанров: литература и политика. Объем каждого жанра 1 миллион предложений.

Для оценки качества аффинных преобразований используется датасет примеров аффинных переносов *Google_analogy_test_set*. Данный датасет был переведен на русский язык с сохранением семантических отношений между словами. В датасет вошло 6 групп, среди которых 2 группы с семантическими связями и 4 группы с грамматическими.

2.2.2 Разработка метода оценки точности параллельного переноса для контекстуализированных моделей

Для контекстуализированных моделей описанные ранее методы оценки аффинных преобразований в чистом виде не применимы. Для одних и тех же слов в разных контекстах будут получаться разные эмбеддинги. Получить единственный эмбеддинг для токена можно после усреднения всех полученных эмбеддингов для этого токена. В результате каждому токену будет соответствовать только одно векторное представление.

После получения усредненных эмбеддингов для всех токенов можно применять описанные ранее методы оценки аффинных преобразований.

Для оценки качества аффинных преобразований будут использоваться 4 метода: 3CosAdd (22), 3CosAdd_3, 3CosMul (23) и 3CosAvg (24).

Метод 3CosAdd_3 аналогичен 3CosAdd с одним отличием, в методе 3CosAdd_3 верный ответ для задачи пропорциональной аналогии будет искаться среди топ 3 слов с максимальным косинусным сходством с целевым словом.

Качество методов 3CosAdd, 3CosAdd_3 и 3CosMul будем оценивать по метрике accuracy (30), помимо этого, посчитаем среднее косинусное расстояние (19) между целевым эмбеддингом и полученным.

Для метода 3CosAvg посчитаем косинусное расстояние(19) между двумя полученными векторами в группах.

$$accuracy = \frac{N_{correct}}{N_{all}} \tag{30}$$

где $N_{correct}$ – количество веррных ответов, N_{all} – общее количество уникальных комбинаций для задачи пропорциональной аналогии.

2.2.3 Векторное представление слов

BERT. Для обработки текстов были взяты две модели BERT:

Первая модель *bert-base-multilingual-cased* – модель BERT, включает в себя 104 языка, в том числе русский язык. Для обучения модели применялись тексты с Википедии. 12 слоев, размер эмбеддинга равен 768. Во время экспериментов, в названии моделей, в которых используется данная модель, будет строка *bert-base-multilingual-cased*.

Вторая модель *DeepPavlov/rubert-base-cased-sentence* – модель BERT [18], обученная специально для русского языка. 12 слоев, размер эмбеддинга равен 768. Во время экспериментов, в названии моделей, в которых используется данная модель, будет строка *DeepPavlov*.

Для проведения экспериментов необходимо подготовить данные для их обработки в модели BERT. Сначала весь текст необходимо разбить на отдельные предложения, далее провести токенизацию и индексацию предложений. На этом этапе обработанные предложения по-одному отправляются в модель BERT.

Полученные после обработки объекты представляют из себя четырёхразмерные тензоры, где оси отражают следующую информацию (в скобках представлено количество элементов):

1) Номер слоя (13 слоев);
2) Номер батча (1 предложение);
3) Количество слов/токенов в предложении (количество токенов в предложении);
4) Векторное представление (768 свойств).

По оси слоев, первый слой - это эмбеддинг, поступающий на вход модели, остальные 12 слоев отображают выходы 12 энкодеров. Номер батча в нашем случае не важен, так как используется только одно предложение. Следующая ось отображает токены в предложении с сохранением порядка. Последняя ось отвечает за векторное представление каждого токена.

Получить итоговое векторное представление для токена можно несколькими способами (рисунок 7). В нашем случае используется способ с суммированием последних четырех слоев, данный способ показывает хорошее качество. Способ с конкатенацией последних четырех не используется, так как он требует в 4 раза больше ресурсов.

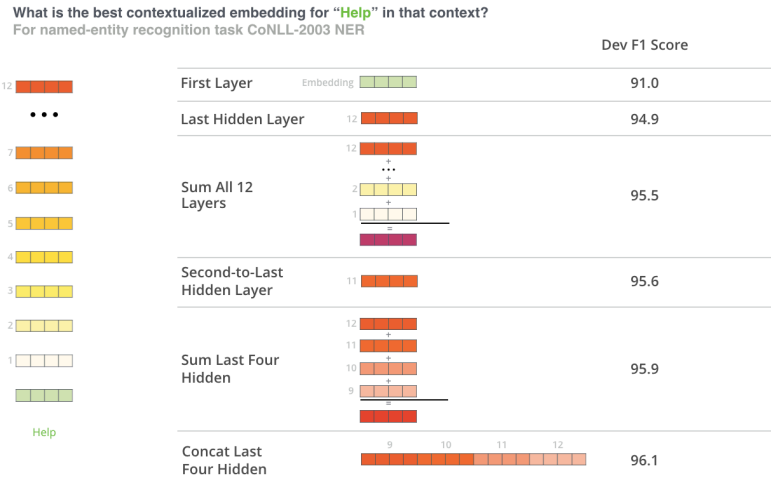

Рис. 7. Возможные варианты получения векторного представления

Описанным ранее методом обрабатываются все подготовленные предложения. В словаре модели BERT, представлены не все слова. Слова, которых нет в словаре, модель разбивает на N-граммы символов. Эмбеддинги получаются именно для N-грамм, некоторые слова делятся на N-граммы по 2 символа. Данные векторные представления уже невозможно использовать для задачи пропорциональной аналогии, так как на этапе тестирования уже нельзя будет определить каким словам соответствуют эти N-граммы.

Есть 3 способа решить эту проблему. Первый способ – получить эмбеддинги для всех токенов, но на этапе тестирования брать только те пары слов, для который есть векторные представления. В названиях моделей, использующих данный способ, не будет индексов.

Второй способ заключается в том, чтобы объединить N-граммы в первоначальное слово, а в качестве векторного представления посчитать величину среднего эмбеддинга для всех использованных токенов. В названиях моделей, использующих данный способ, будет индекс *2*.

Третий способ похож на второй, но с одним исключением. Будет считаться не средняя величина эмбеддингов, а их сумма. Данных подход похож на то, как с векторными представлениями N-граммов работают в модели FastText. В названиях

моделей, использующих данный способ, будет индекс *3*.

Ожидается, что лучшее качество покажет первый способ, однако его тестовый датасет будет меньше, чем у второго и третьего способов.

После того, как получены векторные представления для всего текста, считаются средние эмбеддинги для каждого токена во всех контекстах. Данные эмбеддинги уже готовы к тестированию.

ELMo. Использовалась одна реализация модели ELMo, а именно модель в реализации от DeepPavlov. Модель обучена на большом объеме данных из русской Википедии. Порядок обработки текста аналогичен порядку обработки текста модели BERT, но с одним исключением. В модели BERT был свой токенизатор, который обрабатывал входящие предложения. Для ELMo такого инструмента нет, поэтому входящий текст необходимо токенизирвоать. Из этого вытекает то, что слова не будут разбиваться на N-граммы и обработку N-грамм для ELMo можно будет пропустить.

ELMo создает векторные представления размерностью 1024, модели ELMo будут с припиской elmo.

В сумме получилось 14 моделей векторного представления слов. Используется две модели BERT с тремя способами работы с N-граммами и одна модель ELMo. При этом каждая модель будет работать на текстах двух жанров.

2.2.4 Визуализация данных

Для наглядного представления эмбеддингов слов спроецируем их на плоскость. Сделаем это при помощи метода понижения размерности, а именно метода главных компонент (PCA, principal component analysis). Данный метод позволяет сохранить глобальную структуру данных, а также дает возможность проанализировать взаимное расположение слов.

2.2.5 Оценка результатов

Результаты будут получены для каждой модели, каждого жанра текста, для групп с разными связями между словами. Будут проведены эксперименты с различными приемами работы с N-граммами. Будет выяснено возможно ли аффинные преобразования в семантических пространствах контекстуализированных моделей и оценено их качество.

2.2.6 Программная часть

Для проведения экспериментов выбран язык программирования python. Выбор языка программирования основан на том, что именно на python реализованы и обучены модели векторного представления слов.

В качестве IDE будет использоваться Jupyter Notebok от Google Colab. Выбор Jupyter Notebok основан на том, что это удобный инструмент для аналитической работы, который позволяет интерактивно взаимодействовать с кодом. Google Colab был выбран в качестве платформы для работы с Jupyter Notebok по следующим причинам:

1. Предоставление мощных устройств с 12ГБ оперативной памяти с возможностью подключения GPU и TPU;
2. Возможность запустить параллельно несколько Jupyter Notebok для увеличения скорости работы;
3. Доступ к Google диску для чтения исходных данных и записи результатов.

В качестве хранилища данных используется корпоративный Google диск так, как у него большая вместимость и есть возможность работать с Google Colab. Для увеличения скорости обработки будем параллельно запускать несколько Jupyter Notebok в Google Colab. В процессе обработки данных над одной моделью могут одновременно работать несколько устройств, но одно устройство работает только над одной моделью.

При обработке предложений их порядок не меняется и все предложения поделены на пронумерованные пачки по 1000 штук. Во время обработки предложений устройство проверяет с какими номерами не хватает на диске пачек, если такие есть, то создается пустой файл с номером пачки, чтобы обозначить, что пачка с данным номером уже обрабатывается и другие устройства не начали работать над этими же данными. после окончания обработки пачки результат также сохраняется с номером пачки. По окончанию процесса все пустые файлы удаляются.

Алгоритм агрегации слов построен похожим образом. Но перед началом необходимо составить словарь используемых слов и файлов, в которых эти слова хранятся. Словарь также делится на пачки определенной длины и обрабатывается несколькими устройствами. Важно перед началом агрегации скопировать все эмбеддинги на устройство, чтобы постоянно не гонять много данных через Google диск. В противном случае можно получить бан от Google диска.

Далее остаются менее ресурсозатратные операции, которые в состоянии выполнить одно устройство.

В качестве фреймворков для работы с моделями векторного представления слов используются библиотеки transformer и tensorflow для BERT и ELMo соответственно. Выбор данных фреймворков основан на том, что у них есть доступ реализациям необходимых моделей векторного представления слов.

Ожидается, что объем эмбеддингов будет большим и оперативной памяти может не хватить, поэтому все данные будем обрабатывать пачками и сохранять на Google диске. Хранить данные будем в бинарном формате pickle, данный формат позволяет хранить структуры данных python на диске без дополнительной обработки.

Для отрисовки графиков используется библиотека matplotlib. Данная библиотека является мощным и удобным инструментом для работы с графиками. Для подсчета косинусного расстояния используется функция cosine из библиотеки scipy.

Ссылки на репозиторий с кодом и на примеры получения и тестирования моделей находятся в приложении Б.

3 Практическая часть

3.1 Техническая часть

Для работы модели BERT используется библиотека transformers, которая имеет реализации многих моделей машинного обучения, основанных на модели Transformer. При обработке текста предложения подаются в модель по-одному, на выходе получается 4 мерный тензор, описанный ранее, типа torch.Tensor. Полная обработка миллиона предложения для 1 модели на 1 устройстве занимает около 30 часов. Для увеличения скорости обработки данных один и тот же текст обрабатывало сразу несколько устройств, при этом все данные отправлялись на один Google диск в формате pickle. В зависимости от модели объем полученных эмбеддингов достигает 40 ГБ.

Для работы модели ELMO был использован tensorflow_hub для скачивания модели и tensorflow версии 1.15, так как только для этой версии tensorflow есть русская версия ELMo. На этапе тестирования была выявлена проблема того, что модель ELMo не оптимизирована под получение эмбеддингов слов в отдельных предложениях. Из-за этого оперативной памяти устройства хватало для обработки 22-25 предложений и приходилось вручную перезапускать среду выполнения. Это резко ограничило объем текста, обрабатываемого моделью ELMo. Для модели ELMO обработано только по две тысячи предложений каждого жанра. Полученные эмбеддинги также сохранялись в формате pickle на Google диск.

Далее обработка моделей BERT и ELMO совпадает. Процесс агрегации эмбеддингов может достигать 20 часов при выполнении на 1 устройстве в зависимости от количества уникальных токенов. Данный процесс также выполняется на нескольких устройствах с сохранением результатов на Google диск.

3.2 Визуализация векторных представлений

Визуализируем векторные представления при помощи метода главных компонент.

На рисунках 8, 9 и 10 изображены векторные представления для тестовых пар из группы «прилагательное – сравнительная степень».

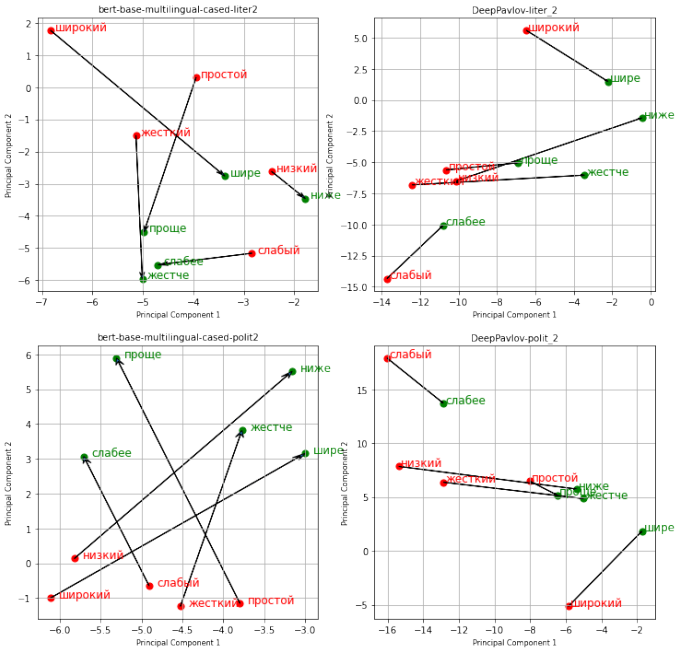

Рис. 8. Визуализация эмбеддингов из группы «прилагательное – сравнительная степень» для модели BERT с обработкой N-грамм через вычисление среднего эмбеддинга; Слева модель bert-base-multilingual-cased, справа от DeepPavlov; Верхние модели обучены на жанре литературы, нижние на политике

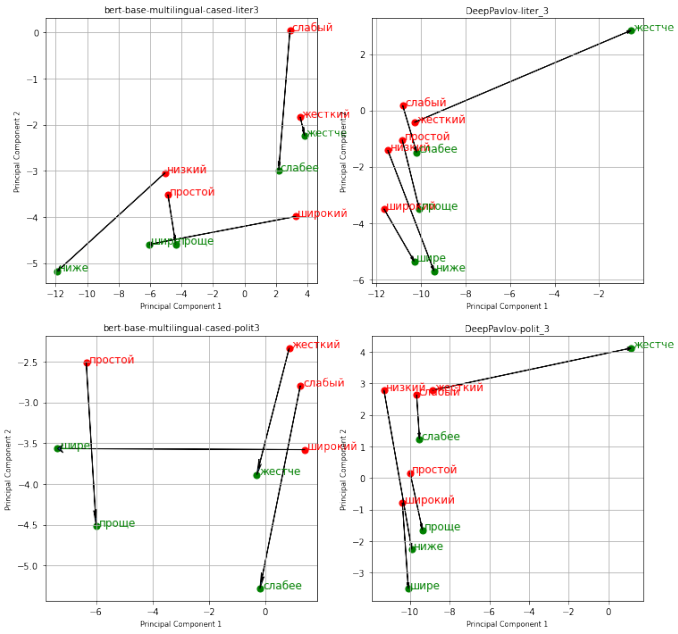

Рис. 9. Визуализация эмбеддингов из группы «прилагательное – сравнительная степень» для модели BERT с обработкой N-грамм через вычисление суммы эмбеддингов; Слева модель bert-base-multilingual-cased, справа от DeepPavlov; Верхние модели обучены на жанре литературы, нижние на политике

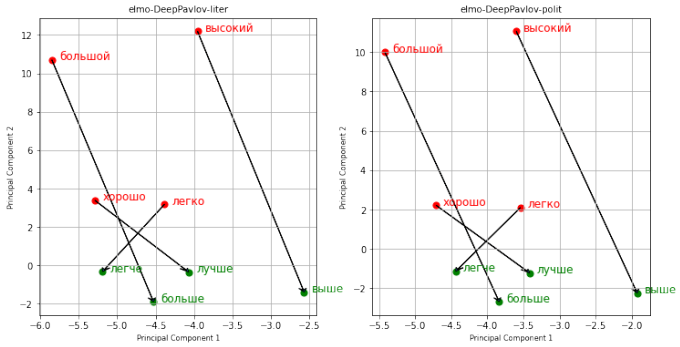

Рис. 10. Визуализация эмбеддингов из группы «прилагательное – сравнительная степень» для модели ELMo; Слева модель обучена на жанре литературы, справа на политике

Первое слово в паре отображается красным цветов, второе – зеленым. Стрелки направлены от первого слова ко второму. В приложении В находятся рисунки 8, 9 и 10 в лучшем качестве, а также визуализация группы «мужской пол – женский пол».

На первый взгляд, все вектора расположено хаотично, но в некоторых примерах прослеживается тенденция того, что вектора направлены в одну сторону. В идеальном случае задачи пропорционального переноса вектора должны быть сонаправлены и быть равной длины. По данным визуализациям можно предположить, что модели не смогут достичь идеального качества.

В визуализации векторных представлений модели ELMo (рисунок 10) можно заметить, что вектора разности при изменении жанра почти не изменяются. Из этого можно сделать вывод, что в модели BERT контекст сильнее влияет на векторные представления, чем в модели ELMo.

Отобразим на плоскости векторные представления во всех контекстах (красные) и их среднее значение (зеленое) на плоскости. На рисунке 11 векторные представления слова «Москва».

Все точки можно условно поделить на 2 группы, справа и слева от центра, опытным путем было установлено, что группа справа отвечает за слова на кириллице, в группу слева входят слова, написанные на латинице.

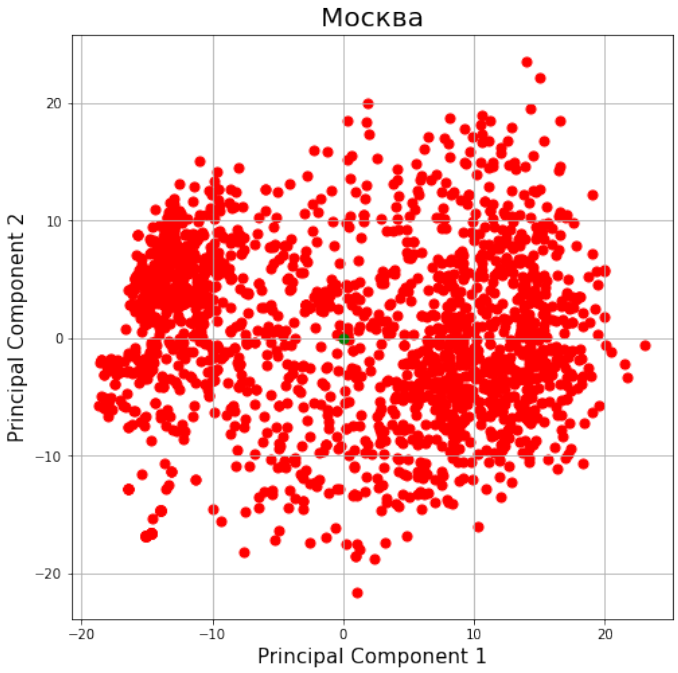

Рис. 11. Визуализация всех векторных представлений модели BERT для слова «Москва»

3.3 Получение результатов тестирования

На рисунках 12, 13, 14 и 15 представлены результаты нескольких моделей. Результаты тестирования всех моделей отображены в приложении Г.

Первая колонка описывает вид связи в группе. Строка *metric* указывает на использованный метод оценки. Колонка *accuracy* отражает долю верных ответов. Колонка *cosine* отображает среднее косинусное расстояние между посчитанным эмбеддингом и правильным эмбеддингом. Последняя колонка описывает количество тестов в группе.

Не во всех таблицах с результатами присутствуют все 6 строк с тестируемыми группами. Это произошло из-за ограниченности словарей моделей. В случае с ELMo сильно сказался объем обрабатываемых текстов.

	accuracy			cosine				count
metric	3CosAdd	3CosAdd_3	3CosMul	3CosAdd	3CosAdd_3	3CosMul	3CosAvg	
Антонимы	0.000000	0.000000	0.000000	0.793130	0.819003	0.918517	1.000000	20
Мужской пол - женский пол	0.659091	0.757576	0.643939	0.917592	0.917783	0.937557	1.000000	132
Национальность - прилагательное	0.333333	0.333333	0.333333	0.899240	0.911733	0.968291	1.000000	6
Прилагательное - наречие	0.222222	0.460784	0.222222	0.857969	0.888816	0.930546	1.000000	306
Прилагательное - сравнительная степенье	0.206522	0.351449	0.215580	0.845331	0.862445	0.907858	1.000000	552
Столицы - страны	0.666667	1.000000	0.666667	0.832437	0.893409	0.832437	0.921754	3

Рис. 12. Результаты тестирования модели BERT от DeepPavlov без обработки N-грамм, обученной на жанре литературы

	accuracy			cosine				count
metric	3CosAdd	3CosAdd_3	3CosMul	3CosAdd	3CosAdd_3	3CosMul	3CosAvg	
Антонимы	0.000000	0.017857	0.000000	0.812737	0.837997	0.923055	1.000000	56
Мужской пол - женский пол	0.631868	0.730769	0.626374	0.915560	0.916735	0.940726	1.000000	182
Национальность - прилагательное	0.401961	0.500000	0.441176	0.860990	0.869891	0.974966	0.999993	408
Прилагательное - наречие	0.115942	0.259058	0.117754	0.862808	0.897032	0.944166	1.000000	552
Прилагательное - сравнительная степенье	0.198276	0.305419	0.197044	0.844081	0.860322	0.910856	1.000000	812
Столицы - страны	0.029851	0.034826	0.044776	0.797999	0.842147	0.888389	0.998932	201

Рис. 13. Результаты тестирования модели BERT от DeepPavlov с обработкой N-грамм через вычисление среднего эмбеддинга, обученной на жанре литературы

	accuracy			cosine				count
metric	3CosAdd	3CosAdd_3	3CosMul	3CosAdd	3CosAdd_3	3CosMul	3CosAvg	
Антонимы	0.000000	0.000000	0.000000	0.813344	0.834249	0.922889	1.000000	56
Мужской пол - женский пол	0.576923	0.686813	0.626374	0.838709	0.840142	0.940715	1.000000	182
Национальность - прилагательное	0.350490	0.397059	0.441176	0.905001	0.921356	0.974966	0.999995	408
Прилагательное - наречие	0.103261	0.230072	0.119565	0.874341	0.905416	0.944115	1.000000	552
Прилагательное - сравнительная степенье	0.162698	0.265873	0.201058	0.758347	0.779599	0.910763	1.000000	756
Столицы - страны	0.038043	0.043478	0.043478	0.739623	0.781124	0.889273	0.999069	184

Рис. 14. Результаты тестирования модели BERT от DeepPavlov с обработкой N-грамм через вычисление суммы эмбеддингов, обученной на жанре литературы

	accuracy			cosine				count
metric	3CosAdd	3CosAdd_3	3CosMul	3CosAdd	3CosAdd_3	3CosMul	3CosAvg	
Антонимы	0.000000	0.000000	0.000000	0.529524	0.711460	0.795986	1.000000	2
Национальность - прилагательное	1.000000	1.000000	1.000000	0.777929	0.777929	0.911426	0.997948	11
Прилагательное - наречие	0.800000	0.933333	0.800000	0.695183	0.710281	0.834578	1.000000	30
Прилагательное - сравнительная степенье	0.214286	0.410714	0.214286	0.667024	0.728323	0.755446	1.000000	56

Рис. 15. Результаты тестирования модели ELMo, обученной на жанре литературы

Данные результаты показывают, что аффинные преобразования в модели BERT возможны, но их качество плохое. Это может быть связано с тем, что в эмбеддингах помимо семантического смысла скрывается другая информация о словах, например, вектор позиционного кодирования. При этом, качество аффинных преобразований модели ELMo на некоторых тестовых группах довольно высокое.

Результаты метода 3CosAvg показывают, что аффинные преобразования работают почти идеально. Это можно объяснить тем, что модели неплохо распознают общий семантический смысл слов и в методе 3CosAvg от моделей не требуется четкий ответ в виде слова, им достаточно дать векторное представление. Следует понимать, что в таблице округленные, хоть и до 6 знака после запятой.

Результаты у методов 3CosAdd, 3CosAdd_3 и 3CosMul получились невысокими из-за того, что рядом с целевым вектором находится много других векторных представлений, которые по смыслу близки к целевому слову.

Проведенные эксперименты показали, что задачу пропорциональной аналогии не всегда можно решить простым параллельным переносом. Семантическое пространство имеет сложную структуру и нельзя с помощью простого параллельного переноса точно определить значение слова. Несомненно, определенные закономерности в параллельном переносе присутствуют и можно приблизительно узнать значение полученного слова. При помощи метода главных компонент было выявлено, что на векторное представление влияет множество факторов: архитектура модели, данные используемые для обучения модели векторного представления, данные используемые для получения эмббеддингов, методы работы с N-граммами и многие другие факторы. Подытоживая, можно сказать, что для решения задачи пропорциональной аналогии параллельного переноса недостаточно, для ее решения необходимы более продвинутые методы преобразования векторов.

4 Заключение

Модели векторного представления слов являются важной частью в автоматической обработке естественного языка. В процессе обучения модели векторного представления слов ориентируются на контекст слова, при этом каждому слову сопоставляется вектор определенной размерности. Данные векторы образуют семантическое векторное пространство. В задаче пропорциональной аналогии предлагается выразить семантическую связь между парой слов в виде вектора разности, и утверждается, что для других пар слов с такой же семантической связью вектор разности векторных представлений будет совпадать. Данный вопрос широко исследовался для различных неконтекстуализированных моделей таких, как word2vec или FastText, где эмбеддинги одного и того же слова в разных контекстах будут одинаковыми. Но он не исследовался для контекстуализированных моделей таких, как BERT или ELMo. В этих моделях векторное представление одного и того же слова в разных контекстах будет отличаться. Это помогает модели лучше передать смысл слова в данном контексте. Особенно это заметно на примерах омонимов.

В данной работе исследуются аффинные преобразования в семантических пространствах контекстуализированных моделей. Был разработан метод для оценки параллельного переноса в этих моделях. Были получены эмбеддинги слов на основании текстов двух жанров и трех моделей векторного представления. Также был подготовлен датасет с раличными типами аналогий. Были проведены численные эксперименты с использованием 4 методов оценки аффинных преобразований: 3CosAdd, 3CosAdd_3, 3CosMul и 3CosAvg. В ходе работы некоторые семантические связи были спроецированы на плоскость с использованием метода понижения размерности.

Численные эксперименты показали, что при решении задачи пропорциональной аналогии параллельный перенос не всегда дает верные ответы. Семантическое пространство имеет сложную структуру и нельзя с помощью простого параллельного переноса точно определить значение слова. При этом общий семантический смысл модель находит довольно неплохо. Было установлено, что на векторное представление влияет множество факторов: архитектура модели, данные используемые для обучения модели векторного представления, данные используемые для получения эмббеддингов, методы работы с N-граммами и многие

другие факторы. Для решения задачи пропорциональной аналогии применение одного параллельного переноса мало, для решения этой задачи требуются более продвинутые методы работы с векторами.

Данная работа может быть основой для последующих исследований контекстуализированных моделей векторного представления. В качестве одного из направлений можно выделить кластеризацию эмбеддингов слов с целью поиска омонимов и определения их семантического значения. Также можно выделить исследование семантической значимости одной или группы позиций в векторных представлениях слов.

Список использованных источников

1. Harris Z. S. Distributional structure //Word. – 1954. – Т. 10. – №. 2-3. – С. 146-162.
2. Mikolov T. et al. Distributed representations of words and phrases and their compositionality //arXiv preprint arXiv:1310.4546. – 2013.
3. Корогодина О. В. Исследование аффинных преобразований в семантическом пространстве Word2Vec // вкр. – Московский институт электроники и математики им. А.Н. Тихонова, 2020.
4. Peters M. E. et al. Semi-supervised sequence tagging with bidirectional language models //arXiv preprint arXiv:1705.00108. – 2017.
5. McCann B. et al. Learned in translation: Contextualized word vectors //arXiv preprint arXiv:1708.00107. – 2017.
6. Peters M. E. et al. Deep contextualized word representations //arXiv preprint arXiv:1802.05365. – 2018.
7. Devlin J. et al. Bert: Pre-training of deep bidirectional transformers for language understanding //arXiv preprint arXiv:1810.04805. – 2018.
8. Rajpurkar P. et al. Squad: 100,000+ questions for machine comprehension of text //arXiv preprint arXiv:1606.05250. – 2016.
9. Vaswani A. et al. Attention is all you need //arXiv preprint arXiv:1706.03762. – 2017.
10. Joulin A. et al. Bag of tricks for efficient text classification //arXiv preprint arXiv:1607.01759. – 2016.
11. Hochreiter S., Schmidhuber J. Long short-term memory //Neural computation. – 1997. – Т. 9. – №. 8. – С. 1735-1780.
12. Dai A. M., Le Q. V. Semi-supervised sequence learning //arXiv preprint arXiv:1511.01432. – 2015.
13. Howard J., Ruder S. Universal language model fine-tuning for text classification //arXiv preprint arXiv:1801.06146. – 2018.

14. Radford A. et al. Improving language understanding by generative pre-training. – 2018.

15. Levy O., Goldberg Y. Linguistic Regularities in Sparse and Explicit Word Representations // Proceedings of the Eighteenth Conference on Computational Natural Language Learning, 2014. P.171-180

16. Drozd A., Gladkova A., Matsuoka S. Word Embeddings, Analogies, and Machine Learning: Beyond King-Man+Woman=Queen // Conference: Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers, 2016. P.3519–3530.

17. Linzen T. Issues in evaluating semantic spaces using word analogies // Proceedings of the 1st Workshop on Evaluating Vector-Space Representations for NLP, 2016. P. 13–18.

18. Kuratov Y., Arkhipov M. Adaptation of deep bidirectional multilingual transformers for russian language //arXiv preprint arXiv:1905.07213. – 2019.