

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ  
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ  
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

**Московский институт электроники и математики  
им. А.Н.Тихонова НИУ ВШЭ**

**Департамент компьютерной инженерии**

Курс «Системное проектирование цифровых устройств»

**ОТЧЕТ  
ПО ЛАБОРАТОРНОЙ РАБОТЕ №1**

Тема работы: «Разработка и программирование Soft-процессорных ядер с  
архитектурой однокластерный MIPS. Часть 1»

**Выполнили:**

Студенты группы БИВ174  
Бригада №5

Подчерзцев Алексей Евгеньевич  
Солодянкин Андрей Александрович

**Принял:**

асс. МИЭМ НИУ ВШЭ  
Американов А.А.

Москва 2021 г.

т31

т32

## **Аннотация**

A Software Tool for Automatic Generation of a Graph of Conversation Using a Drama  
Corpus

## **Abstract**

# Содержание

<b>Введение</b>	<b>6</b>
<b>1 Обзор литературы</b>	<b>9</b>
1.1 Появления моделей векторных представлений слов	9
1.2 Word2vec	10
1.3 FastText	11
1.4 ELMo	11
1.4.1 LSTM	12
1.4.2 Структура ELMo	14
1.5 BERT	15
1.5.1 Encoder from Transformer	15
1.5.2 Структура BERT	19
1.6 Исследование методов оценки аффинных преобразований	20
<b>2 Теоретическая часть</b>	<b>22</b>
2.1 Постановка задачи	22
2.2 Метод исследования	24
2.2.1 Подготовка экспериментальных данных	24
2.2.2 Разработка метода оценки точности параллельного переноса для контекстуализированных моделей	24
2.2.3 Векторное представление слов	24
2.2.4 Визуализация данных	26
2.2.5 Оценка результатов	26
<b>3 Практическая часть</b>	<b>28</b>
3.1 Выводы	28
<b>4 Заключение</b>	<b>29</b>
<b>Список использованных источников</b>	<b>30</b>

# 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 эмбедингами.

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 [18]. 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 эмбедингами 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 для решением

задачи пропорциональной аналогии будет выражение  $v_d - v_c = v_b - v_a$ , где  $v_d$  и  $v_c$  эмбединги 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).

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

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 эмбединг. 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 эмбединга 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 практики 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).

$$v_i^j = \begin{cases} 1, i = j \\ 0, i \neq j \end{cases}, j = 1, \dots, K \quad (2)$$

Недостатком этого метода является то, что по данным векторным представлениям нельзя судить о семантической схожести слов. Также для при обработке реальных текстов размер словаря будет очень большим, а значит длина векторных представлений будет очень большой, данные вектора неэффективно хранить в памяти. Также к многим 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 эмбедингами 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 - предиктивными модели. Данные модели для создания эмбединга 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 [18] 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.

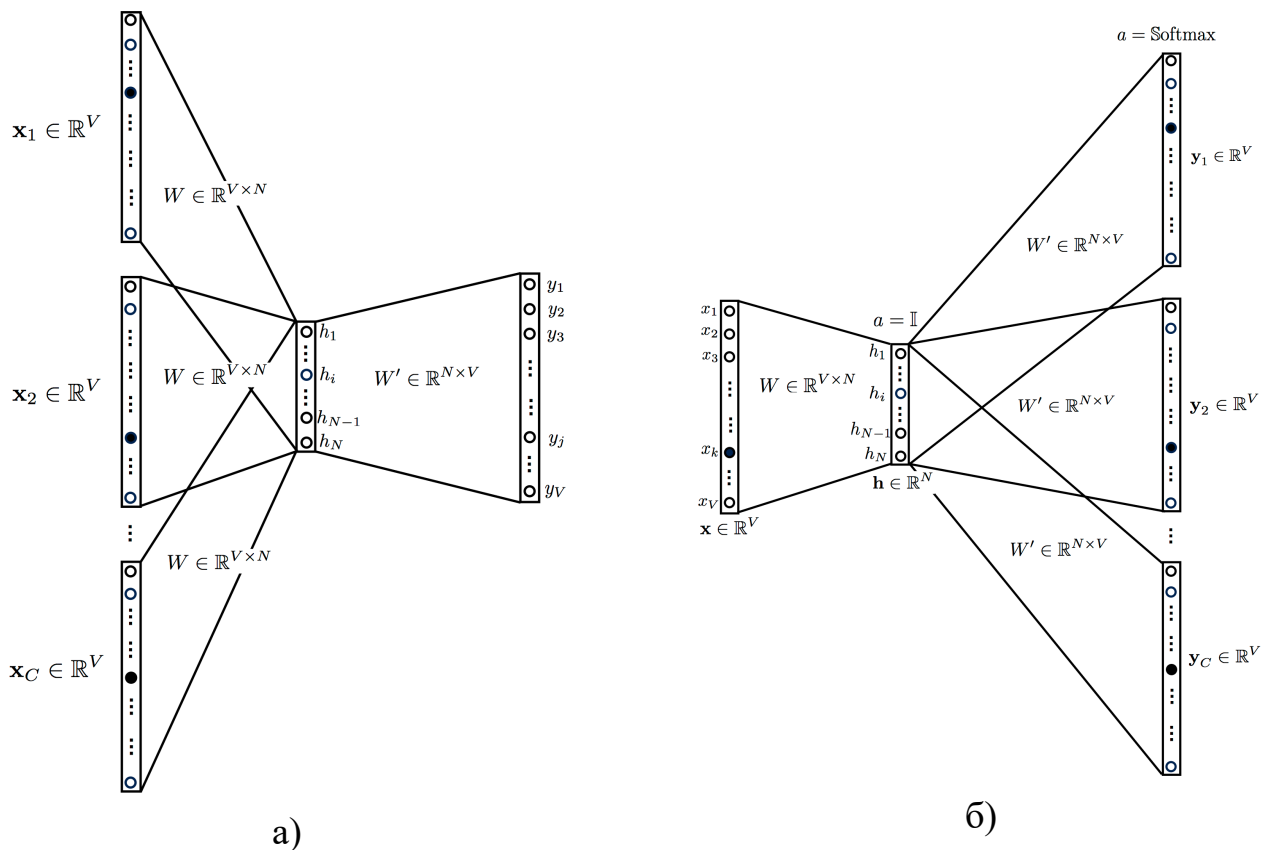


Рис. 1. Схемы архитектур CBOW (а), Skip-gram (б)

Архитектуры представляют из себя полносвязные нейронные сети с обратным распространением ошибки. Для больших словарей метод активации softmax накладывает серьезные вычислительные нагрузки. Поэтому для оптимизации было предложено использовать метод негативного семплирования (negative sampling).

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 эмбединги, присваиваемые словам, зависят не только от самого слова, но и от контекста. Основная идея получения эмбедингов – использование скрытых состояний LSTM.

Разберемся по подробнее с LSTM блоками.

### 1.4.1 LSTM

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

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

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).

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

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

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).

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

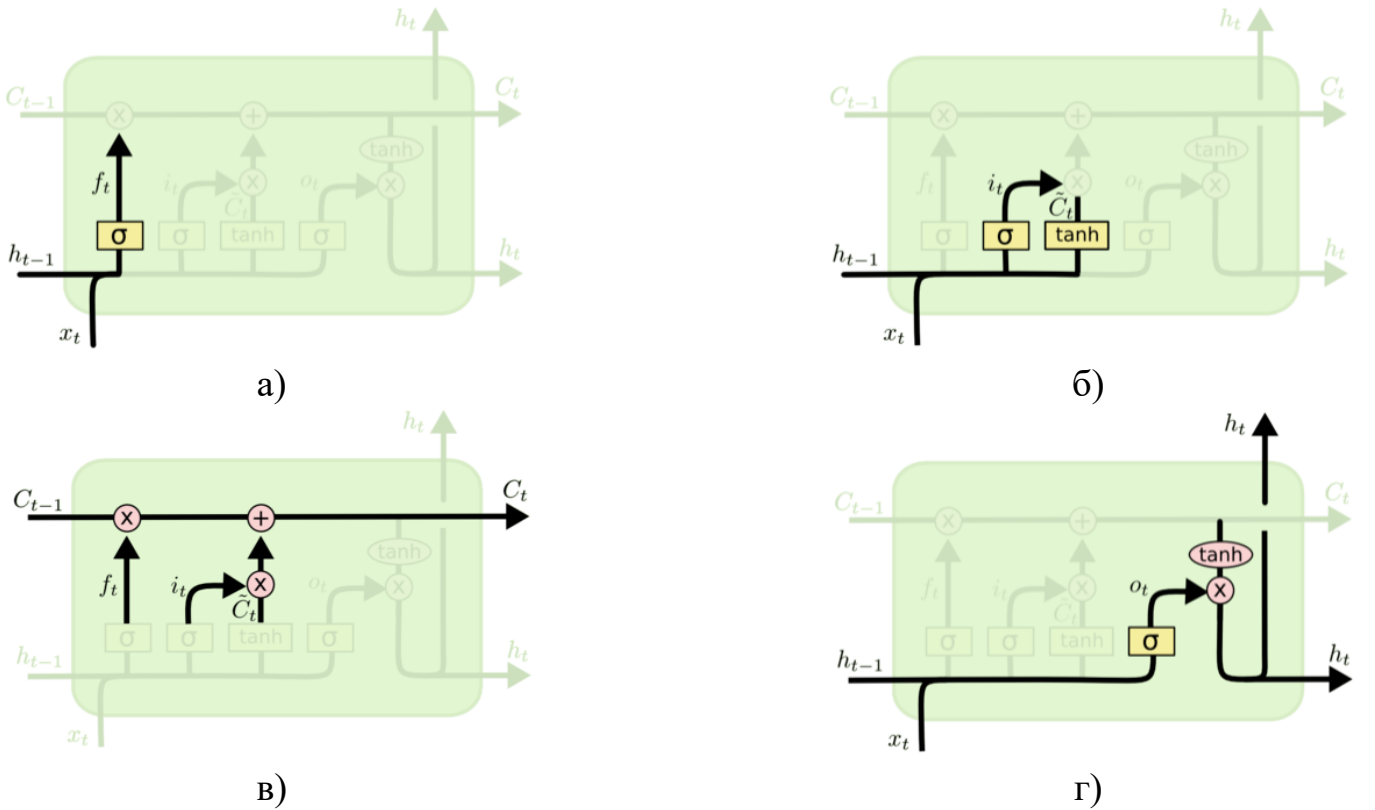


Рис. 2. Виды фильтров: а) слой фильтра забывания, б) слой входного фильтра, в) применение текущих изменений, г) слой выходного фильтра.

На рисунке 2.г изображен финальный формирования выходов. Сначала в сигмоидальном слое поступает информация предыдущего выхода  $h_{t-1}$  и текущего входа  $x_t$ , где определятся 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).

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

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

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 состоит 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.

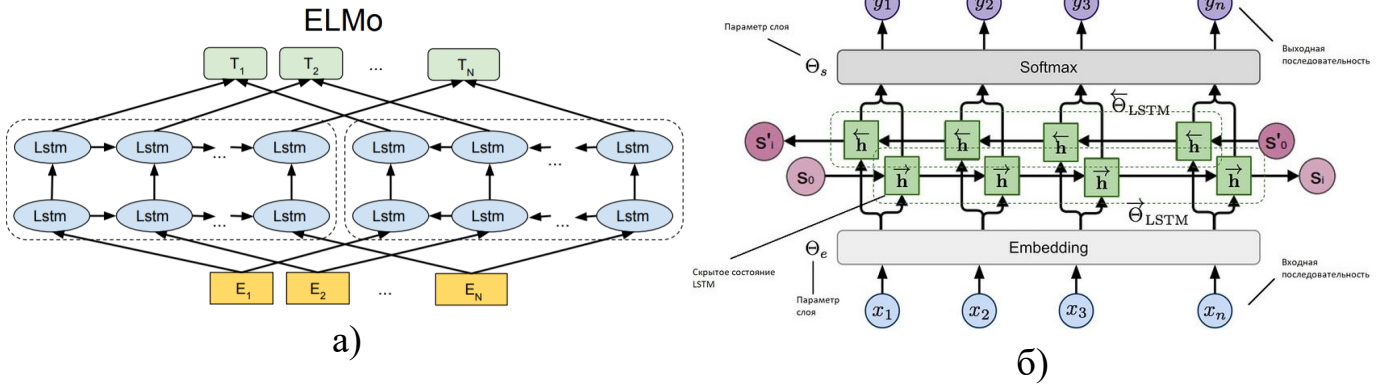


Рис. 3. Архитектура модели ELMo (а), принцип работы модели ELMo (б)

На рисунке 3.б схематично изображен принцип работы модели ELMo. Пусть на вход подаются токены  $t_1, \dots, t_N$ , 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).

$$\left\{ x_k^{LM}, \overrightarrow{h_{k,j}^{LM}}, \overleftarrow{h_{k,j}^{LM}} | j = 1, \dots, L \right\} = \{ h_{k,j}^{LM} | j = 1, \dots, L \} \quad (9)$$

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).

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

где  $\gamma^{task}$  – масштабирующий коэффициент, который регулирует то, как могут отличаться по норме эмбединги 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 обработке 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 части 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.

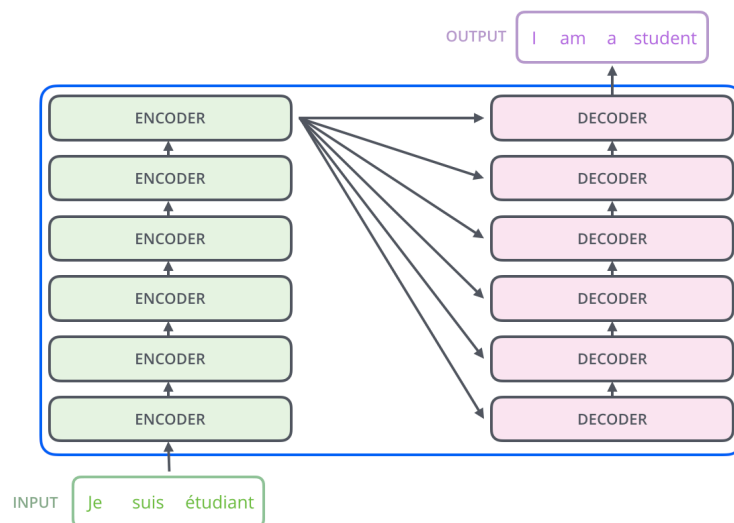


Рис. 4. Структура модели Transformer

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.

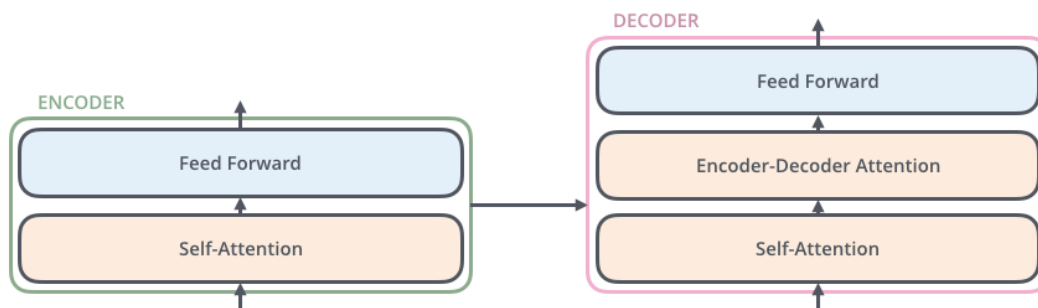


Рис. 5. Структура кодировщика и декодировщика

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



не контекстуализированные эмбединги слов. Эмбединги 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).

$$Q = X * W_Q \quad (11)$$

$$K = X * W_K \quad (12)$$

$$V = X * W_V \quad (13)$$

где  $X$  – матрица эмбедингов 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  $\text{softmax}$  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.

$$S = Q * K^T \quad (14)$$

$$M = \text{softmax}\left(\frac{S}{\sqrt{L}}\right) \quad (15)$$

$$Z = M * V \quad (16)$$

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

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

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 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 эмбедингу. Позиционное кодирование позволяет определить позицию каждого слова или расстояние между словами в предложении. Идея метода в том, что добавление векторов к эмбедингам создает осмысленное расстояние между векторными представлениями в процессе проецирования в  $Q/K/V$  векторы и скалярного произведения при вычислении внимания.

На рисунке [в примечаниях] 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 эмбединги, 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.

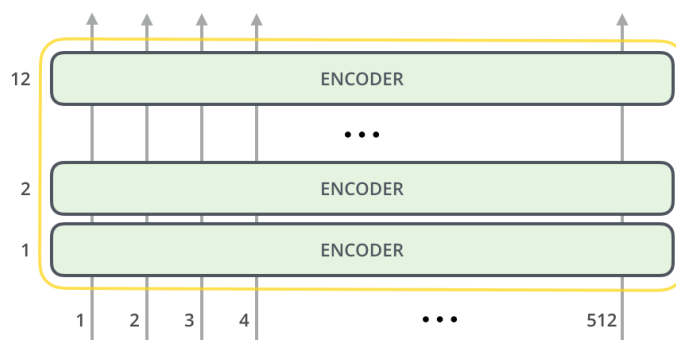


Рис. 6. Структура модели BERT

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 эмбединги слов, которые в последствии можно использовать в различных 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 (??) косинусное сходство. Для удобства применяется также косинусное расстояние (18).

$$distance(A,B) = 1 - similarity(A,B) \quad (18)$$

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 (19).

$$(a : a^*) :: (b : b^*) \quad (19)$$

Если в парах действует одна и та же связь, то можно выразить одно из слов через 3 других слова (20).

$$y = v_b - v_a + v_{a^*} \quad (20)$$

где

$y$  - искомый вектор,

$v_x$  - эмбединг 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 (21).

$$y^* = \underset{v \notin \{v_b, v_a, v_{a^*}\}}{argmax} \cos(v, v_b - v_a + v_{a^*}) \quad (21)$$

Интуитивно понятным способом оценить качество будет сопоставить все пары слов друг с другом, посчитать одно из слов по формуле (21) [15] и найти среднее значение косинусной близости после трансформаций. Данный способ получил название **3CosAdd**.

Существует предположение, что 22 эквивалентно (21). Этот метод получил название **3CosMul** [15].

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

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** (23) [16].

$$y^* = \underset{v \notin \{v_b, v_a, v_{a^*}\}}{\operatorname{argmax}} \cos\left(v, v_b + \frac{\sum_{i=1}^n v_{a_i^*}}{n} - \frac{\sum_{i=1}^n v_{a_i}}{n}\right) \quad (23)$$

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 (24).

$$y^* = \underset{v \notin \{v_b, v_a, v_{a^*}\}}{\operatorname{argmax}} \cos(v, v_b) \quad (24)$$

В методе **Ignore-a** [17] предполагается, что результат ближе всего к сумме векторов  $v_b$  и  $v_{a^*}$  (25).

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

В методе **Add-opposite** [17] результатом является вектор, ближайший к выражению  $-(v_{a^*} - v_a) + v_b$  (26).

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

Метод **MULTIPLY** [17] имеет следующий вид (27).

$$y^* = \underset{v \notin \{v_b, v_a, v_{a^*}\}}{\operatorname{argmax}} \frac{\cos(v, v_{a^*}) \cos(v, v_b)}{\cos(v, v_a)} \quad (27)$$

## 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  $V_w = \{v_{1w}, v_{2w}, \dots, v_{nw}\} | V_w \subset V$ , где  $v_{iw}$  – векторное представление слова  $w$  в  $i$ -ом контексте,  $V$  множество всех векторных представлений слов. При этом существует функция  $\eta$ , которая преобразует множество эмбедингов слова в векторное представление  $v'_w = \eta(V_w) | v'_w \in V'$ .

Выделим пары слов  $(w_{11}, w_{12}), (w_{21}, w_{22}), \dots, (w_{k1}, w_{k2})$  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 составить пропорциональные аналогии слов:  $(w_{i1}, w_{i2}) :: (w_{j1}, w_{j2}), i \neq j$ . Каждой паре слов можно сопоставить пары векторов  $(v'_{11}, v'_{12}), (v'_{21}, v'_{22}), \dots, (v'_{k1}, v'_{k2})$ , где  $v'_{ij}$  –  $j$ -ый вектор в паре под номером  $i$ . В итоге получаются следующие выражения:

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

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

пересекающиеся, скрещивающиеся – в скрещивающиеся.

Математически аффинное преобразование  $\varphi : \mathbb{R}^n \rightarrow \mathbb{R}^n$  выглядит так в выражении (28).

$$\varphi(x) = M \cdot x + v \quad (28)$$

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 попадетсся 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 эмбедингов слов были взяты тексты из электронной библиотеки КиберЛенинка. Тексты двух жанров: литература и политика.

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 эмбединги. Получить единственный эмбединг для токена можно после усреднения всех полученных эмбедингов для этого токена. В результате каждому токену будет соответствовать только одно векторное представление.

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

Для оценки качества аффинных преобразований будут использоваться 3 метода: 3CosAdd, 3CosMul и 3CosAvg.

### 2.2.3 Векторное представление слов

*ELMo*

*BERT*. Для обработки текстов были взяты 2 модели BERT:

Первая модель *bert-base-ultilingual-cased* – модель BERT, включает в себя 104 языка, в том числе русский язык. Для обучения модели применялись тексты с Википедии. 12 слоев, размер эмбединга равен 768.

Вторая модель *DeepPavlov/rubert-base-cased-sentence* – модель BERT [], обученная специально для русского языка. 12 слоев, размер эмбединга равен 768.



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.

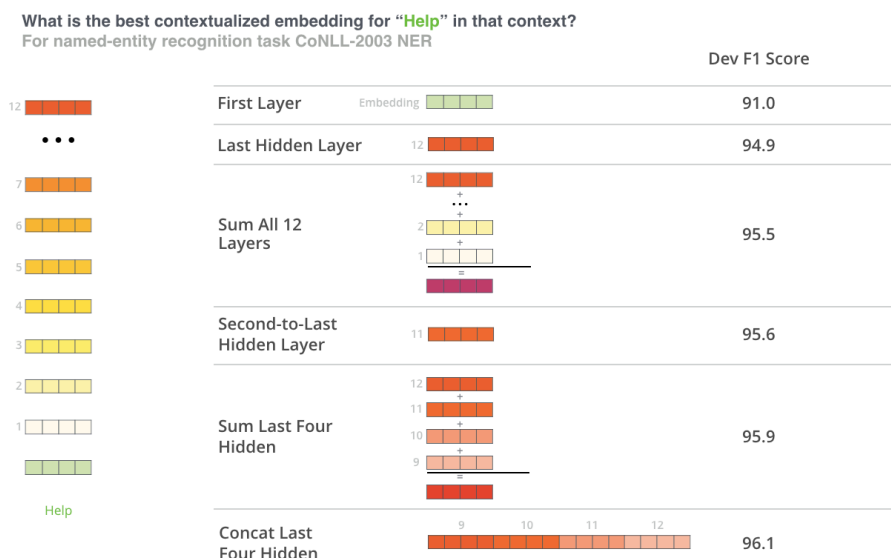


Рис. 7. Возможные варианты получения векторного представления

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. Эмбединги 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 эмбединги для всех токенов, но на этапе тестирования брать только те пары слов, для которых есть векторные представления.

Второй способ заключается в том, чтобы объединить N-граммы в первоначальное слово, а в качестве векторного представления посчитать величину среднего эмбединга для всех использованных токенов.

Третий способ похож на второй, но с одним исключением. Будет считаться не средняя величина эмбедингов, а их сумма. Данный подход похож на то, как с векторными представлениями N-граммов работают в модели fastText.

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 эмбединги для каждого токена во всех контекстах.

#### **2.2.4 Визуализация данных**

Для наглядного представления эмбедингов 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.

## **3 Практическая часть**

### **3.1 Выводы**

- 1.

## **4 Заключение**

### 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.