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 однотактный MIPS. Часть 1»

**Выполнили:**

Студенты группы БИВ174
Бригада №5

Подчезерцев Алексей Евгеньевич
Солодянкин Андрей Александрович

**Принял:**
асс. МИЭМ НИУ ВШЭ
Американов А.А.

Москва 2021 г.

## Аннотация

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

## Abstract

# 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 [9]. 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 [6]. 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 для 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 []. Где пришли 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 исследовании [] 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 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 [5]. BERT также контекстуализированная модель. Архитектура модели BERT представляет из себя последовательность двунаправленных кодировщиков из Transformer. 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).

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

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 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 - предиктивными 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 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.

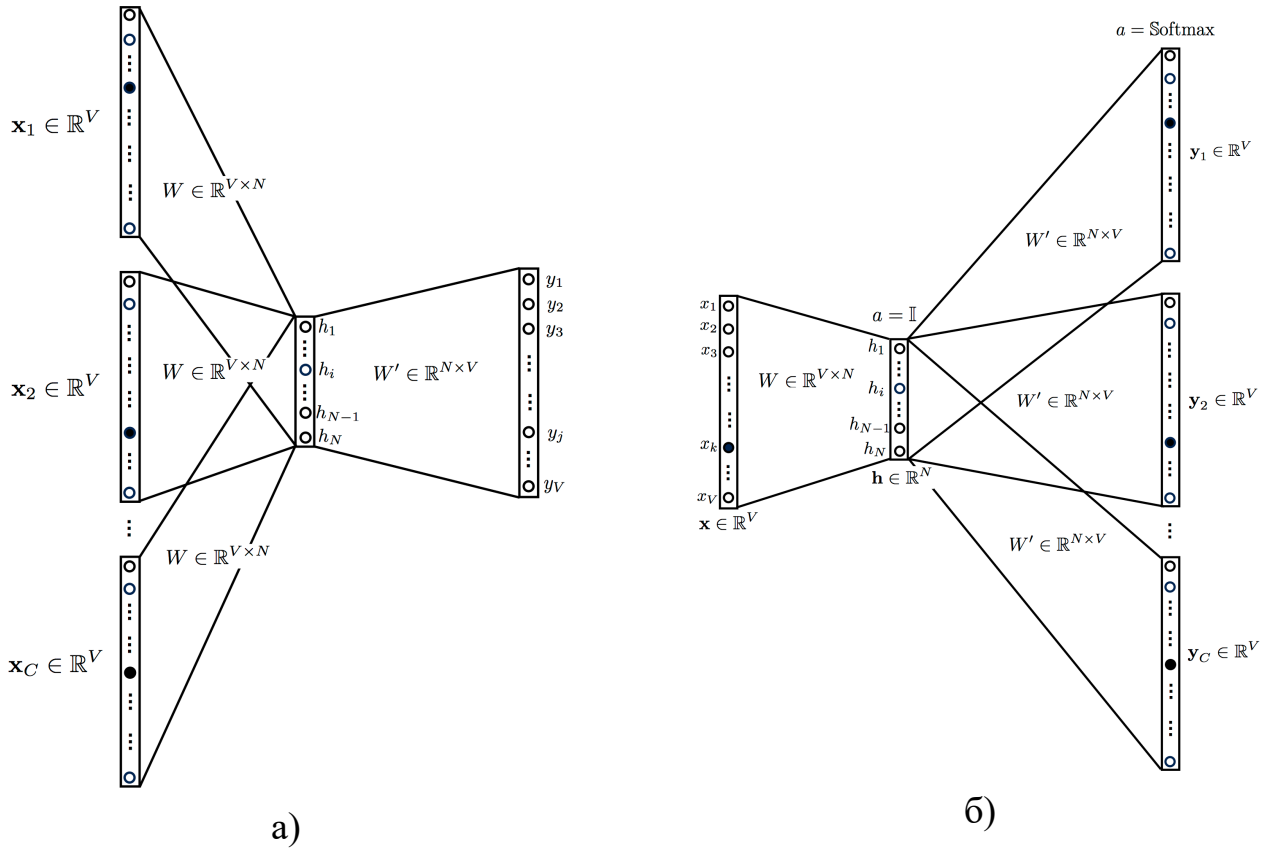


Рис. 1. Схемы архитектур CBOW (а), Skip-gram (б)

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 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 []. 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 по подробнее с LSTM блоками.

### 1.4.1 LSTM

LSTM или Long short-term memory дословно переводится как долгая краткосрочная память, были предложены в статье []. Данные блоки являются разновидностью архитектуры рекуррентных нейронных сетях 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 размываются 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).

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

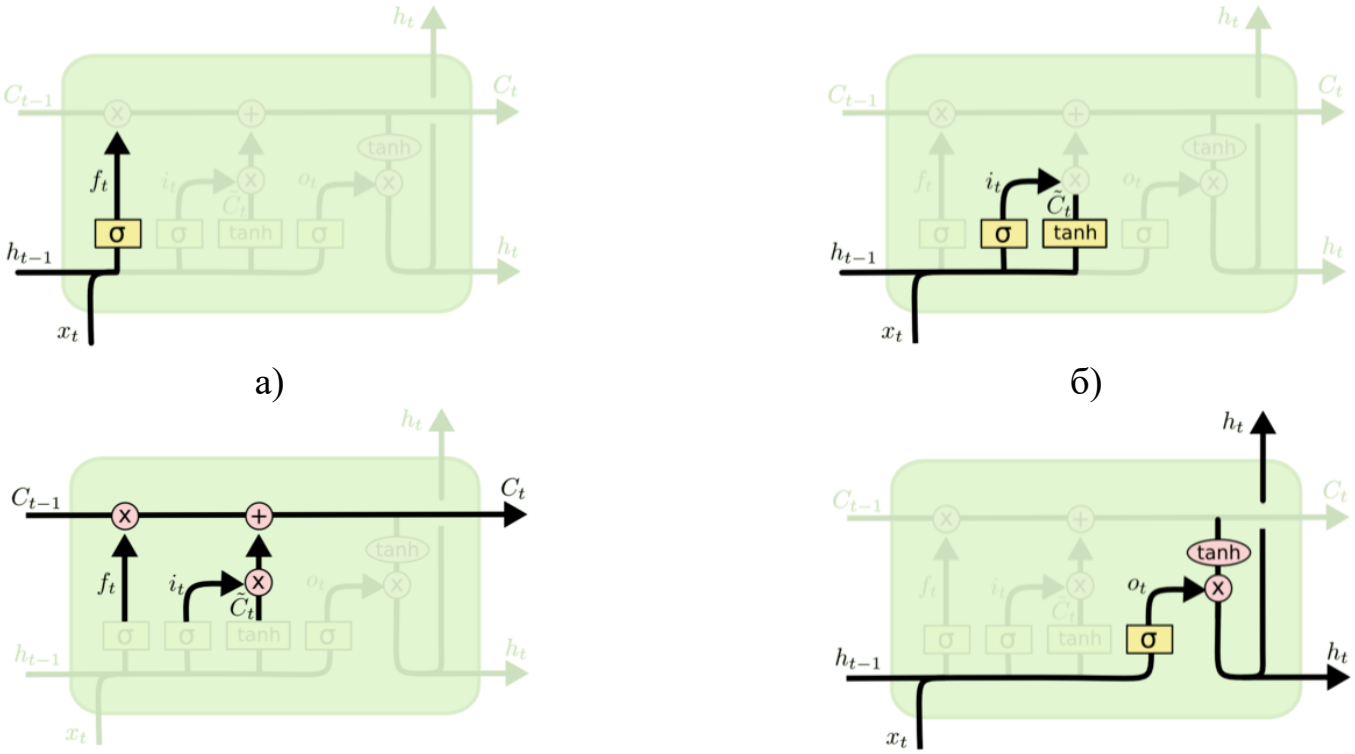


Рис. 2. Виды фильтров: a) слой фильтра забывания, b) слой входного фильтра, c) применение текущих изменений, d) слой выходного фильтра.

На рисунке 2.г 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).

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

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

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.

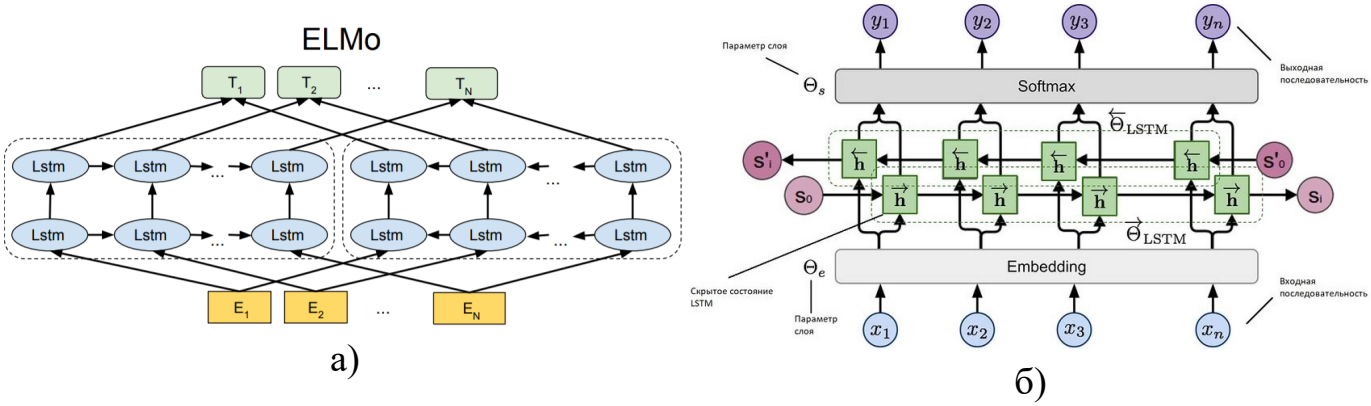


Рис. 3. Архитектура модели ELMo (а), принцип работы модели ELMo (б)

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

Векторным представлением слова будет яаляться 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 или Bidirectional Encoder Representations from Transformers – модель обработки естественного языка выпущенная в 2017 году командой Google AI. В основе работы модели лежит ряд недавних разработок в сфере обработке естественного языка. Можно выделить разработки Semi-supervised Sequence learning [], уже упомянутый ELMo [], ULMFiT [] и OpenAI Transformer [].

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 []. Принцип работы модели трансфомера 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 числи 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.

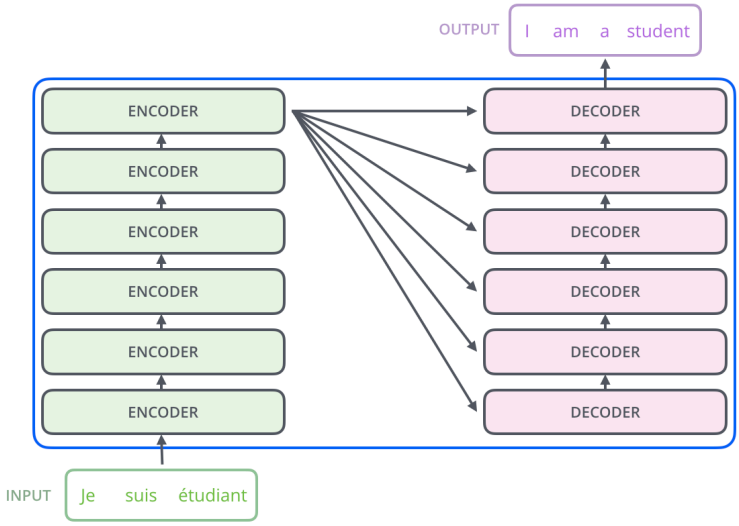


Рис. 4. Структура модели Transformer

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.

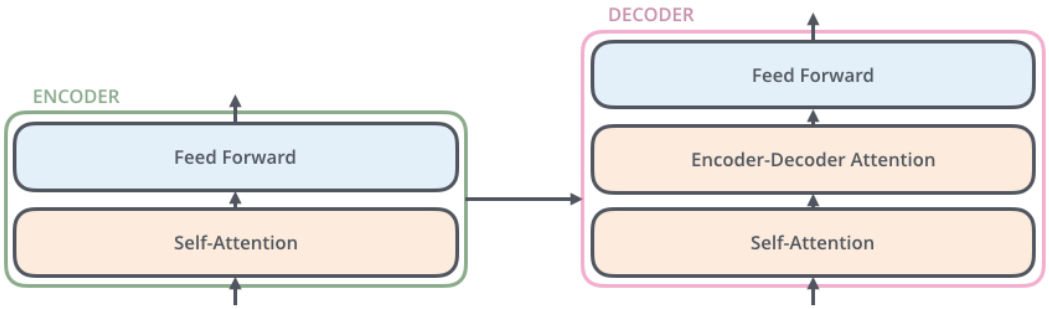


Рис. 5. Структура кодировщика и декодировщика

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

не контекстуализированные 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).

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

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 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.

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 эмбеддниги, следующие за целевым векторным представлением. Упомянутый ранее промежуточный слой внимания Encoder-Decoder attention, помимо взвешенных сумм со слоя внутреннего внимания, принимает нв 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.

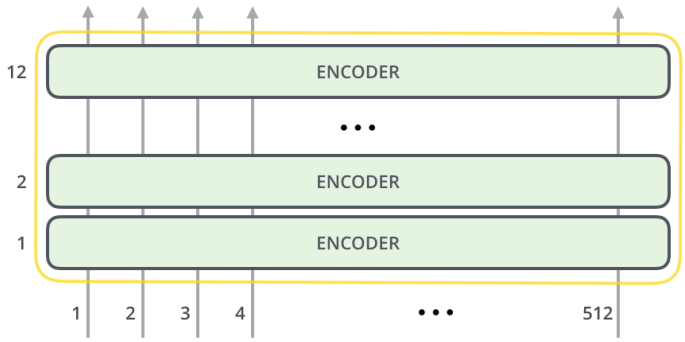


Рис. 6. Структура модели BERT

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.

# 2 Теоретическая часть

## 2.1 Выводы

1.

# 3 Практическая часть

## 3.1 Выводы

1.

# 4 Заключение

## Список использованных источников

1.