МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра математической кибернетики и компьютерных наук

ГЕНЕРАЦИЯ СУММАРИЗАЦИЙ ТЕКСТОВ ПРИ ПОМОЩИ НЕЙРОСЕТЕВОЙ МОДЕЛИ POINTER-GENERATOR С МЕХАНИЗМОМ ПОКРЫТИЯ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 411 группы
направления 02.03.02 — Фундаментальная информатика и информационные технологии
факультета КНиИТ
Лезгян Артема Саркисовича

Научный руководитель
зав. каф. техн. прогр, к. ф.-м. н. _____ И. А. Батраева

Заведующий кафедрой
к. ф.-м. н., доцент _____ А. С. Иванов

Саратов 2020

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире неуклонно увеличивается объем информации, которую так или иначе ежедневно обрабатывает каждый человек. Поэтому все более важной становится задача автоматической суммаризации текстовых данных. Краткое содержание длинных документов, новостных статей или переписок может помочь существенно сократить время, затрачиваемое на изучение или поиск той или иной информации.

Задача суммаризации обычно заключается в том, чтобы сжать некоторый текст до более короткого фрагмента, который при этом должен содержать основную информацию из оригинала. Данная задача сложна тем, что состоит не только в выделении наиболее значимых смысловых моментов исходного текста, но также обычно требует, чтобы получаемый на выходе текст был связным, построенным по правилам естественного языка. Суммаризация является одной из основных задач NLP и может применяться например для:

— непосредственно сжатия текста, что существенно снижает время, затрачиваемое человеком на чтение;
— решения задач информационного поиска, где краткое резюме может упростить процесс выбора документа или увеличить эффективность индексации;
— персонализированные резюме могут использоваться в вопросно-ответных системах, так как позволяют обобщать запросы пользователя и строить по ним соответствующий ответ.

Начиная с 50-х годов XX века, когда впервые была сформулирована проблема извлечения некоторой общей информации из текста, для решения данной задачи использовались методы, основанные на: выделении тематических слов, подсчете частоты встречаемости слов (TF-IDF), базах знаний, графовых моделях и многие другие [1]. Однако в последние годы наибольшие успехи в решении различных NLP-задач достигаются благодаря развитию машинного обучения, в особенности рекуррентных нейронных сетей и механизмов внимания [2]. Поэтому целью данной работы является реализация нейросетевой архитектуры Pointer-Generator и применение ее в задаче автоматической суммаризации текстов.

Поставленная задача решается в несколько этапов:

— анализ различных способов предобработки данных и приведение текстов

к виду, пригодному для обучения нейронной сети;
- изучение основ архитектур seq2seq и Pointer;
- реализация архитектуры Pointer-Generator, алгоритма Beam Search и необходимых для обучения и использования модели инструментов;
- обучение модели с различными гиперпараметрами и по-разному обработанными данными;
- анализ результатов экспериментов и оценка эффективности предложенного решения.

Бакалаврская работа состоит из введения, двух глав, заключения, списка использованных источников и пяти приложений. Объем работы 44 страницы. Список литературы включает 42 наименования.

В первой главе изложены некоторые элементы теории: архитектура реализованной модели Pointer-Generator, теоретические основы способов предобработки данных, алгоритма Beam Search и используемых метрик.

Во второй главе описаны особенности предобработки данных, реализации нейросетевой модели и необходимых для ее обучения и использования алгоритмов, а так же результаты экспериментов.

1 Описание работы

1.1 Подходы к задаче суммаризации

Существует два основных подхода к решению данной задачи: экстрактивный и абстрактивный. Экстрактивные методы предполагают формирование аннотации путем извлечения некоторых наиболее значимых частей (обычно предложений) из исходного текста. Абстрактивные же методы обладают возможностью генерировать новые слова или целые фразы, не встречающиеся в исходном тексте. Такой подход ближе к используемому человеком при написании резюме.

Экстрактивный метод проще, так как копирование текста исходного документа обеспечивает высокие показатели грамотности и точности. С другой стороны, этот метод лишен таких преимуществ, как возможность перефразировать исходный текст или обобщать информацию.

Из-за довольно высокой сложности абстрактивной суммаризации в основном используются экстрактивные модели, например суммаризатор, предоставляемый библиотекой для обработки естественных языков Gensim. Однако развитие рекуррентных нейронных сетей, способных к генерации текста, сделало возможным применение абстрактивного метода суммаризации. Такие модели также не лишены недостатков: они могут искажать фактические детали и не могут приемлемо обрабатывать слова, отсутствующие в их словаре, к тому же подобные модели могут несколько раз генерировать один и тот же текст. Для решения этой проблемы была разработана гибридная архитектура, включающая в себя оба подхода к суммаризации [3]. К рекуррентной нейронной сети был добавлен механизм указывания, который позволяет извлекать неизвестные модели слова напрямую из исходного текста, при этом сохраняя способность к обобщению и самостоятельной генерации текста.

1.2 Создание архитектуры модели

1.2.1 Гибридная архитектура Pointer-Generator

Рассмотренная в данной работе архитектура Pointer-Generator (рисунок 1) представляет собой гибрид между seq2seq с механизмом внимания [4] и архитектурой Pointer [5]. Она позволяет как генерировать описание на основе фиксированного словаря модели, так и копировать слова из исходного текста с помощью механизма указывания.

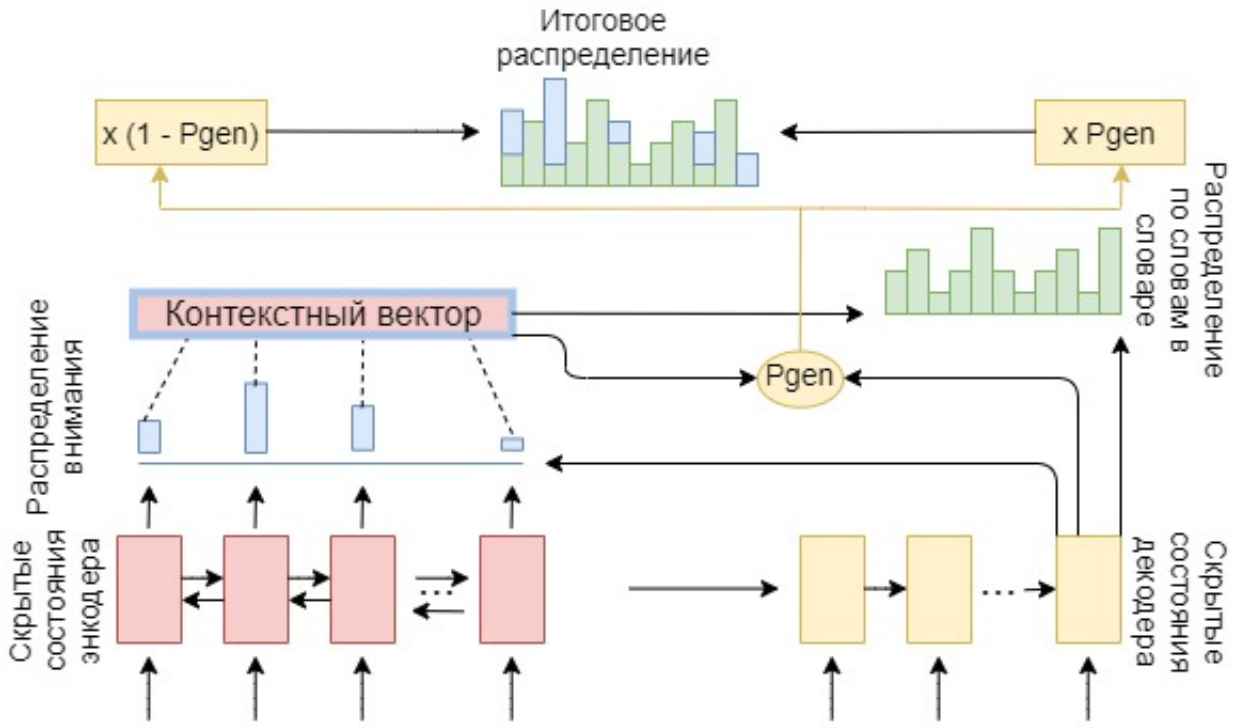

Рисунок 1 – Архитектура модели Pointer-Generator

В модели указатель-генератор распределение внимания a^t и контекстный вектор h_t^* рассчитываются так же, как и в seq2seq. В дополнение к этому на каждом шаге t на основе контекстного вектора h_t^*, скрытых состояний декодировщика s_t и входных данных декодировщика x_t вычисляется вероятность генерации $p_{gen} \in [0, 1]$:

$$p_{gen} = \sigma(w_{h^*}^T h_t^* + w_s^T s_t + w_x^T x_t + b_{ptr}), \tag{1}$$

где $w_{h^*}^T, w_s^T, w_x^T$ — векторы, b_{ptr} — обучаемый параметр, а σ — сигмоидная функция. Вероятность генерации используется для дальнейшего выбора между генерацией слова из словаря на основе их вероятностей P_{vocab} и копированием слова из исходной последовательности на основе распределения внимания. Для каждого документа сохраняется его расширенный словарь, который является объединением словаря модели и всех слов, встречающихся в исходном документе. Таким образом, итоговая функция распределения вероятности для расширенного словаря принимает следующий вид:

$$P(w) = p_{gen} P_{vocab}(w) + (1 - p_{gen}) \sum_{i:w_i=w} a_i^t, \tag{2}$$

Так, если слово w отсутствует в словаре модели (OOV), то $P_{vocab}(w)$ будет

равна нулю, в то же время, если слово отсутствует в исходном документе, то $\sum_{i:w_i=w} a_i^t$ обратится в ноль. Способность обрабатывать OOV слова является основным преимуществом архитектуры Pointer-Generator перед другими моделями, ограниченными их собственным словарным запасом. Функция потерь при добавлении механизма указателя не изменяется, только теперь вероятность целевого слова вычисляется по формуле 2.

1.2.2 Механизм покрытия

Довольно распространенной проблемой seq2seq моделей является повторение одних и тех же слов или целых предложений. В качестве способа борьбы с этим явлением может быть использован механизм покрытия coverage [6]. В модели Pointer-Generator используется вектор покрытия c^t, который является суммой распределения внимания на всех предыдущих шагах.

Фактически, вектор покрытия это не нормализированное распределение по словам исходного документа, которое отображает степень покрытия, полученную каждым словом в результате работы механизма внимания на предыдущих шагах. Вектор покрытия используется как дополнительный аргумент при вычислении распределения внимания.

Такой подход позволяет механизму внимания при принятии текущего решения учитывать и распределения, построенные на предыдущих шагах. Это дает возможность лучше избегать концентрации на одних и тех же участках исходной последовательности и, как следствие, с меньшей вероятностью генерировать повторяющийся текст. Также некоторые дополнения вносятся в функцию потерь, чтобы дополнительно штрафовать модель за генерацию повторяющихся слов.

1.3 Предварительная обработка данных

На вход модели должен подаваться заранее обработанный корпус текстов. В рамках данной работы было проведено четыре эксперимента по обучению построенной модели на данных, предобработанных различными способами, и сравнение результатов обучения.

В связи с ограниченностью вычислительных ресурсов модель обучалась генерации новостных заголовков, которые так же, как и аннотация, отражают основное содержание текста, но при этом значительно короче полной аннотации. В качестве датасета использовался корпус новостей сайта ria.ru.

1.3.1 Базовая предобработка

Базовая предварительная обработка состоит из следующих этапов:

— Разделение единого файла датасета на набор небольших блоков по 1000 документов в каждом. Это необходимо для упрощения дальнейшей параллельной обработки данных и для экономии оперативной памяти.
— Разбиение текста на предложения и предложений на отдельные слова, отделение пробелами знаков препинания.

Также в каждом эксперименте после предварительной обработки данных составлялся словарь, отсортированный в порядке убывания частотности слов.

1.3.2 Векторные представления слов

Распределенное (или векторное) представление слов — это способ представления слов в виде векторов евклидова пространства, размерность которого обычно равна нескольким сотням. Основная идея заключается в том, что геометрические отношения между точками евклидова пространства будут соответствовать семантическим отношениям между словами. Например, слова, представленные двумя близко расположенными точками векторного пространства, будут, скорее всего, синонимами или просто тесно связанными по смыслу словами.

Обычно вместе с обучением рекуррентных слоев нейронной сети обучаются и входные Embedding слои, отвечающие за преобразование слов естественного языка в их векторное представление, пригодное для анализа нейронной сетью. Однако для этого можно использовать векторные представления слов построенные заранее, например другими моделями машинного обучения.

В данной работе были использованы вектора FastText, обученные на корпусе статей Wikipedia. Для этого были выполнены следующие действия:

— К предварительно обработанному корпусу текстов была применена лемматизация, то есть приведение всех слов к начальной форме.
— В качестве итогового словаря взято пересечение между словарем модели FastText и словарем, построенным на основе лемматизированного датасета.

1.3.3 Алгоритмы стемминга

Стемминг — это поиск основы слова, причем не обязательно совпадающей с корнем. Он является одной из основных составляющих нормализации

текста, позволяющей как анализировать только семантику слова, игнорируя словоформы, так и существенно сокращать необходимый для работы словарь.

В данной работе был использован Snowball stemmer, обладающий достаточными точностью и быстродействием. Он использовался для того, чтобы разделить каждое слово на его неизменяемую основу — стемму и изменяемую часть — флексию. После этого стеммы и флексии рассматриваются как отдельные слова: из них составляется словарь, для каждого слова строится его векторное представление и подается на вход модели. Такой подход позволяет существенно сократить словарь модели, при этом сохранив возможность без каких-либо дополнительных вычислений восстанавливать формы слов.

1.3.4 Byte pair encoding

Впервые алгоритм BPE был опубликован Филиппом Гейджем в статье «Новый алгоритм сжатия данных» в февральском выпуске C Users Journal 1994 года, как алгоритм сжатия данных. Идея алгоритма заключалась в нахождении наиболее часто встречающихся пар байтов и замене их на некоторый новый байт, отсутствующий в исходных данных.

Однако в дальнейшем идея алгоритма была применена для токенизации текстов [7]. Для этого исходный алгоритм модифицируется таким образом, чтобы объединять часто встречающиеся пары буквенных n-грамм, а не заменять их. Это приводит к тому, что любое сложное или редкое слово в итоге оказывается разбито на несколько более часто встречающихся n-грамм. Кратко алгоритм можно описать так:

1. Инициализируется словарь;
2. Каждое слово в корпусе представляется как комбинация символов со специальным символом конца слова `</w>`;
3. Производится подсчет пар символов во всех словах корпуса;
4. Наиболее часто встречающаяся пара символов объединяется в один токен и добавляется в словарь, после чего во всем корпусе данная пара символов заменяется этим токеном;
5. Шаг 3 повторяется до тех пор, пока не будет произведено необходимое число операций слияния или пока не будет достигнут желаемый размер словаря (который является единственным гиперпараметром алгоритма).

Таким образом, BPE является чем-то средним между кодированием текстов на уровне букв и кодированием на уровне слов. Как и побуквенное ко-

дирование, он позволяет существенно снизить размерность словаря, при этом длина каждой последовательности остается приемлемой, как и при кодировании по словам. Дополнительным преимуществом этого подхода является то, что он, как и разделение на стеммы и флексии, сохраняет исходные формы слов. К тому же этот подход, как и использование указателя, позволяет избавиться от проблемы OOV слов.

Главным недостатком оригинального алгоритма является его низкая скорость работы. Для решения этой проблемы была применена библиотека YouTokenToMe, реализующая оптимизированную версию данного алгоритма. Библиотека предоставляет методы для построения словаря токенов и последующей токенизации текстовых последовательностей.

1.4 Лучевой поиск

На этапе генерации выходной последовательности декодировщик циклично получает на вход сгенерированное им же на предыдущем шаге слово (на первом шаге это специальный символ начала генерации `[START]`) и генерирует следующее слово последовательности. Так продолжается до тех пор, пока не будет сгенерирован символ конца последовательности `[STOP]` или пока последовательность не превысит определенной длины. Результатом работы декодировщика на каждом шаге генерации последовательности является распределение вероятности по всем словам словаря (возможно расширенного). Конечной целью является генерация наиболее вероятной последовательности слов.

Оптимальным выбором будет использование эвристических алгоритмов, а именно, алгоритма лучевого поиска Beam Search, наиболее часто применяющегося в задачах декодирования выходной последовательности seq2seq моделей. Он использует обход в ширину для построения дерева поиска. На каждом шаге алгоритм выбирает все достижимые из элементов текущего уровня состояния и сортирует их в порядке возрастания стоимости эвристики, после чего сохраняет определенное число β (называемое шириной луча) наилучших состояний. На следующем уровне дерева поиск будет осуществляется только из этих состояний. Ширина луча является единственным гиперпараметром алгоритма, при $\beta = 1$ лучевой поиск превращается в жадный алгоритм, при неограниченной ширине луча — в поиск в ширину. Этот алгоритм также не гарантирует того, что будет найдено оптимальное значение, однако существен-

но увеличивает вероятность этого. В общем случае Beam Search возвращает первое найденное значение, однако в задачах генерации текстовых последовательностей алгоритм продолжает работу до тех пор, пока не будет достигнута определенная глубина поиска или заданное количество завершившихся токеном [STOP] последовательностей . После завершения поиска алгоритм оценит все полученные решения и выдаст наиболее вероятное.

1.5 Существующие метрики

Важной проблемой для всех моделей, основанных на генерации нового текста по исходному (диалоговые модели, машинный перевод, автоматическая суммаризация), является оценка их качества. Существует довольно большое количество различных метрик, предназначенных для решения этой задачи. Рассмотрим некоторые из них:

- BLEU [8] — определенного класса метрик, разработанного для машинного перевода, но также успешно применяемых и для других задач. BLEU является модификацией точности (precision) совпадения ответа модели и целевой последовательности, перевзвешенной таким образом, чтобы ответ, состоящий из одного слова, не получил идеальную оценку. BLEU стала одной из первых метрик, показавших высокую корреляцию с человеческими представлениями о качестве, и до сих пор остается одной из самых популярных и вычислительно недорогих метрик.
- ROUGE — еще один класс метрик, более ориентированных именно на оценку качества автоматической суммаризации [9]. ROUGE подсчитывает долю пересечения множеств n-грамм, последовательностей слов и пар слов между сгенерированной и эталонной аннотациями.

2 Программная реализация

2.1 Реализация архитектуры

В основе реализации архитектуры модели Pointer-Generator лежат классы: `Encoder` и `Decoder`.

Для эффективной работы с последовательностями различной длины в кодировщике используются предоставляемые фреймворком PyTorch функции `pack_padded_sequence` и `pad_packed_sequence`, позволяющие обрабатывать тензоры произвольной длины (важным для работы этих функция является то, что входные данные должны быть отсортированы в порядке невозрастания). Выход LSTM-слоя дополнительно подается на вход полносвязному слою (необходимо для дальнейшего вычисления распределения внимания). Кодировщик возвращает исходный выход LSTM, выход полносвязного слоя и свои скрытые состояния. Во время шага декодировщика текущее слово конкатенируется с вектором контекста и подается на вход LSTM, после чего на основе скрытых состояний рекуррентного слоя, вектора покрытия и выходов кодировщика в классе `Attention`, реализующем механизм внимания, вычисляются новый контекстный вектор, распределение внимания и вектор покрытия. Далее вычисляется вероятность генерации, позволяющая переключаться между извлечением слова из оригинального текста и генерацией на основе словаря. Также на основе выходов декодировщика, которые подаются на вход двум полносвязным слоям, строится распределение вероятности по словам в словаре модели. Последними шагом является получение итогового распределения вероятностей по словам расширенного словаря. Следует отметить, что во всех моделях, кроме основанной на векторах FastText, векторные представления слов обучаются одновременно с самой моделью. В данной работе для кодировщика и декодировщика использовался один и тот же словарь, веса Embedding также были общими.

2.2 Цикл обучения

В процессе обучения, на вход кодировщику подается набор исходных текстовых последовательностей, которые могут иметь разную длину. После завершения работы кодировщика запускается цикл декодирования. Начальные скрытые состояния декодировщика (однонаправленной LSTM) инициализируются весами кодировщика, в дальнейшем на каждом шаге веса будут иници-

ализироваться данными с предыдущей итерации. Проблемой является то, что размерность скрытых состояний кодировщика из-за прохода в двух направлениях в два раза больше, чем у декодировщика. Для решения этой проблемы веса кодировщика подаются на вход полносвязному слою с функцией активации $relu$, который учится извлекать наиболее важные признаки из обоих направлений прохода и понижает размерность до необходимой. Во время обучения при каждой итерации декодирования, на вход декодировщику подается предыдущее слово оригинальной аннотации. Данный прием обучения называется teacher forcing. В декодировщик передаются выходы кодировщика, векторы контекста и покрытия, а также дополнительная информация, необходимая для работы механизма указывания. После каждого шага декодирования вычисляются потери как отрицательный логарифм от полученной вероятности следующего слова эталонной последовательности. В конце обработки мини-батча сумма потерь на каждом шаге нормализуется по длине декодированной последовательности и выбирается среднее из потерь для всех последовательностей в батче. После вычисления обратного распространения ошибки применяется gradient clipping, необходимый для предотвращения «взрыва» градиентов [10].

2.3 Лучевой поиск

Для более точной генерации суммаризаций текстов в данной работе был реализован алгоритм Beam Search. Для удобства работы с каждой гипотезой создан класс `Beam`, используемый для хранения расшифрованных токенов, а также дополнительной информации: вероятностей каждого из сгенерированных токенов, текущих скрытых состояний декодировщика, векторов контекста и покрытия. На вход кодировщику подается мини-батч размером β (ширина луча). Это необходимо для дальнейшей оптимизации лучевого поиска: гипотезы вычисляются не последовательно, а параллельно, как до этого при обучении обрабатывались мини-батчи. После создаются β пустых гипотез, скрытые состояния декодировщика инициализируются скрытыми состояниями кодировщика. Далее, на каждом шаге декодирования выходной последовательности, из гипотез извлекаются последние расшифрованные токены, скрытые состояния декодировщика с предыдущего шага, контекстный вектор и покрытие, после чего подаются на вход декодировщику. Из распределения вероятностей по словам, построенного декодировщиком, извлекаются $top(\beta * 2)$ наиболее вероятных токенов. Для каждой из гипотез с предыдущего шага сохраняют-

ся $\beta * 2$ новых, дополненных сгенерированными на текущем шаге словами и обновленными данными, после чего список новых гипотез сортируется по невозрастанию. Если какая-то последовательность в этом списке заканчивается символом `[STOP]`, то она добавляется в массив результатов, иначе — в массив лучших на этом шаге. После того, как набрано β лучших гипотез, алгоритм переходит к следующей итерации. Декодирование завершается, если была достигнута максимальная заданная длина или набрано β завершившихся последовательностей. После завершения алгоритма выбранные гипотезы сортируются и возвращается лучшая.

2.4 Эксперименты по обучению

Во время обучения каждые 5000 батчей производилась валидация модели на тестовых данных. Для борьбы с переобучением использовался метод ранней остановки. Для этого был реализован класс `Checkpoint`, который после каждой валидации производил сравнение текущего значения потерь и наилучшего. В случае, если значение функции потерь не улучшалось в течении четырех валидаций, обучение останавливалось. Класс поддерживает сохранение n последних моделей, а также отдельное сохранение лучшей. При обучении использовался оптимизатор Adam [11].

Лучшей оказалась модель, обученная на приведенных к виду стемма + флексия данных без использования механизма покрытия. Модель, обученная на предобработанных BPE данных, показала второй результат по точности, но довольно быстро переобучилась.

Таким образом, на основе данной работы можно сделать вывод, что архитектура Pointer-Generator, даже в базовой версии, вполне успешно справляется с задачей генерации суммаризаций текстов. Абстрактивный метод позволяет лаконично передать главную мысль, тогда как экстрактивный при необходимости дополняет ее некоторой важной информацией. Механизм покрытия также работоспособен и позволяет избегать повторений и излишней концентрации на одном участке текста. Таким образом, модель может быть использована на практике, например, для построения кратких аннотаций текстов в поисковых системах.

ЗАКЛЮЧЕНИЕ

В ходе выполнения дипломной работы были решены поставленные задачи:

- проанализированы и применены различные способы предобработки исходных данных;
- изучены основы архитектур seq2seq и Pointer;
- реализована гибридная архитектура Pointer-Generator с механизмом покрытия, алгоритм Beam Search, а также дополнительные средства для обучения и валидации модели;
- произведено несколько экспериментов по обучению модели на по-разному предобработанных данных;
- проанализированы результаты экспериментов, предложены некоторые возможные пути улучшения модели.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 *Allahyari, M.* Text summarization techniques: A brief survey / M. Allahyari, S. Pouriyeh, M. Assefi, S. Safaei, E. Trippe, J. Gutierrez, K. Kochut // *International Journal of Advanced Computer Science and Applications (IJACSA)*. — July 2017. — Vol. 8. — Pp. 397–405.

2 *Головко, В. А.* Нейронные сети: обучение, организация и применение / В. А. Головко, В. В. Краснопрошин. — Москва: Радиотехника, 2001.

3 *See, A.* Get to the point: Summarization with pointer-generator networks / A. See, P. J. Liu, C. D. Manning. — 2017.

4 *Николенко, С. И.* Глубокое обучение / С. И. Николенко, С. А. Кадурин, Е. О. Архангельская. — СПб.: Питер, 2018.

5 *Vinyals, O.* Pointer networks / O. Vinyals, M. Fortunato, N. Jaitly. — 2015.

6 *Meng, F.* Interactive attention for neural machine translation / F. Meng, Z. Lu, H. Li, Q. Liu. — 2016.

7 *Sennrich, R.* Neural machine translation of rare words with subword units / R. Sennrich, B. Haddow, A. Birch. — 2015.

8 *Papineni, K.* Bleu: a method for automatic evaluation of machine translation / K. Papineni, S. Roukos, T. Ward, W. J. Zhu. — October 2002.

9 *Lin, C.-Y.* Automatic evaluation of machine translation quality using longest common subsequence and skip-bigram statistics / C.-Y. Lin. — July 2004. https://www.microsoft.com/en-us/research/publication/automatic-evaluation-of-machine-translation-quality-using-longest/-common-subsequence-and-skip-bigram-statistics/.

10 *Брайан, М.* Знакомство с PyTorch: глубокое обучение при обработке естественного языка / М. Брайан, Р. Делип. — СПб.: Питер, 2020.

11 *Ян, Г.* Глубокое обучение / Г. Ян, Б. Иошуа, К. Аарон. — Москва: ДМК, 2018.