[Введение 2](#_Toc110408469)

[1 Что такое реферирование в NLP 3](#_Toc1192039116)

[1.1 Экстрактивное обобщение 3](#_Toc1270891916)

[1.2 Абстрактное обобщение 4](#_Toc2129351721)

[2 Введение в Seq2seq моделирование 5](#_Toc1082545670)

[3 Архитектура кодировщика - дешифровщика 6](#_Toc1528842611)

[3.1 Обучение 6](#_Toc1232589223)

[3.1.1 Кодировщик 6](#_Toc1245663219)

[3.1.2 Декодер 7](#_Toc1385568589)

[3.2 Вывод 8](#_Toc198270101)

[4 Ограничения архитектуры кодировщика-дешифровщика 10](#_Toc1097221085)

[5 Концепция работы механизма Внимания 11](#_Toc1675709131)

[5.1 Глобальное внимание 11](#_Toc2130415721)

[5.2 Локальное внимание 12](#_Toc1162630605)

[6 Задача 13](#_Toc279811154)

[7 Реализация модели с использованием Keras 14](#_Toc266726418)

[Вывод 15](#_Toc1165286707)

[Список литературы 16](#_Toc1891007770)

# 

# Введение

Во многих документах зачастую содержится большое количество текста, который не несет существенную информацию. Хорошим примером таких документов являются различные новостные статьи. Зачастую людям, у которых нет времени на прочтение полного текста, нужно ознакомиться с кратким содержанием новости. Также очень удобно прочитать лишь краткую аннотацию новости для того, чтобы понять стоит ли читать новость полностью. Таким образом, появляется необходимость сокращать объём документа, выделяя наиболее значимую часть текста, называемую рефератом. Ручное реферирование — сложная, рутинная работа, требующая дополнительных сотрудников, поэтому целесообразно использовать системы автоматического реферирования текстов.

Задача автоматического реферирования текстов очень популярна среди исследователей. Существует большое количество публикаций, в которых описываются различные алгоритмы автоматического реферирования. Однако, различные авторы используют различные метрики для оценки предложенных ими алгоритмов. Кроме того, оценка алгоритмов производится, в основном, на англоязычных наборах документов. В связи с этим применение алгоритмов автоматического реферирования текстов к русскоязычному набору документов и их сравнение является актуальной задачей.

Один из подходов реализации реферирования текстов является методы глубокого обучения. Результаты достигаемые при применение данного подхода имеют замечательные показатели. В этой работе будут рассмотрен пошаговый процесс создания “Автоматического реферирования” с использованием глубокого обучения, охватывая все концепции необходимые для его создания и реализована модель реферирования текста на языке программирования Python.

# Что такое реферирование в NLP

Давайте сначала разберемся, что такое резюмирование текста, прежде чем мы посмотрим, как оно работает. Вот краткое определение, с которого можно начать: «Автоматическое резюмирование текста - это задача создания краткого и понятного резюме при сохранении ключевого информационного содержания и общего смысла».

Существует два основных подхода к резюмированию текста (Рисунок 1): экстрактивное и абстрактное обобщение.

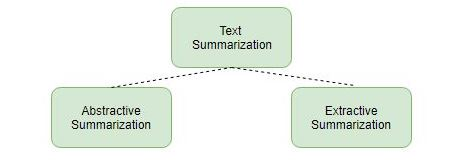


Рисунок 1-Виды автоматического реферирования

## Экстрактивное обобщение

По названию данного подхода можно предположить что он делает. В данном методе выделяются только ключевые важные слова или предложения из исходного текста. Эти извлеченные предложения и есть резюме. На диаграмме показан концепт (Рисунок 2).

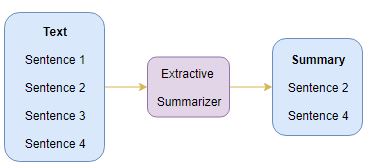


Рисунок 2-Схема экстрактивного реферирования

## Абстрактное обобщение

Этот подход интересен тем что на основе исходного текста осуществляется генерация новых, которые в совокупности будут составлять резюме.

В отличии от экстрактивного подхода в котором мы использовали только те предложения которые присутствовали в данном случает мы создаём новую последовательность. Предложения сформированные посредством абстрактного реферирования могут отсутствовать в исходном тексте (Рисунок 3).

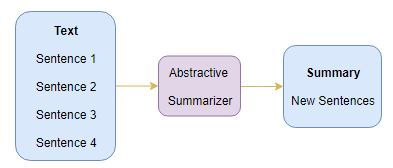


Рисунок 3-Схема абстрактного реферирования

В данном работе мы будем создавать программу реферирования текстов основанную на абстрактной модели, но сперва нужно разобраться с моделями и основными концепциями, необходимыми для построения модели.

# Введение в Seq2seq моделирование

Для любой информации которая носит последовательный характер мы можем построить Seq2seq модель.

Данная модель применяется при анализе тональности текста, нейронном машинном переводе и распознавании именованных сущностей, которое широко применяется при работе с информацией последовательного характера.

В случае нейронного машинного перевода - это текст на одном языке, а выход - также текст, но на другом языке. Пример: “I love playing sports”-”Me encanta hacer deporte”.

При распознавании сущностей входом является последовательность слов, а выходом-последовательность тегов для каждого слова из входной последовательности.



Проблему автоматического реферирования можно сформулировать в терминах Seq2seq. На рисунке показана типичная архитектура модели Seq2seq(Рисунок 4).

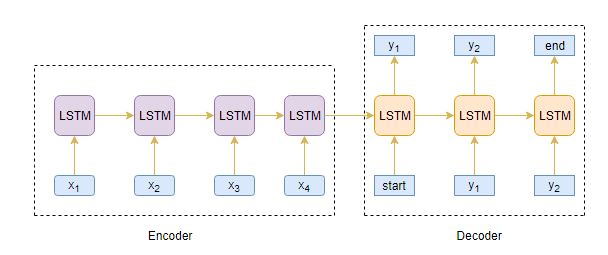


Рисунок 4-Архитектура модели Seq2seq

В данной архитектура можно выделить два компонента: ***Кодировщик, Декодер***.

Для понимания того как писать код и как это будет работать нужно разобраться с этими двумя понятиями.

# Архитектура кодировщика - дешифровщика

Архитектура кодировщика-декодера используется для решения задач Seq2seq, когда входные и выходные последовательности имеют разную длину.

Вход - это длинная последовательность слов, а выход- короткая версия входной последовательности слов (Рисунок 5).

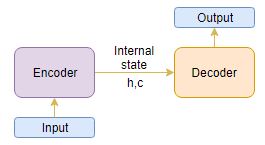


Рисунок 5-Схема потока данных

На практике в качестве компонентов кодировщика и декодера предпочтительно используют различные варианты рекуррентных нейронных сетей (RNN), стробируемых нейронных сетей (GPU) или долгосрочную-краткосрочную память (LSTM). Это потому что они способны фиксировать долгосрочные зависимости, преодолевая проблему исчезающего градиента.

Настройка кодировщика-декодера происходит в 2 этапа:

- Обучение;

- Вывод.

Разбираться в этих концепция будем основываясь на модель LSTM.

## Обучение

На этапе обучения происходит настройка кодировщика и декодера. Следующий шаг обучение модели предсказыванию смещения целевой последовательности на один шаг.

Рассмотрим подробно как настроить кодировщик и декодер.

### Кодировщик

Модель долгосрочной краткосрочной памяти кодировщика (LSTM) считывает всю входную последовательность, при этом на каждом временном шаге в кодировщик подается одно слово. Затем он обрабатывает информацию на каждом временном шаге и фиксирует контекстную информацию, присутствующую во входной последовательности (Рисунок 6).

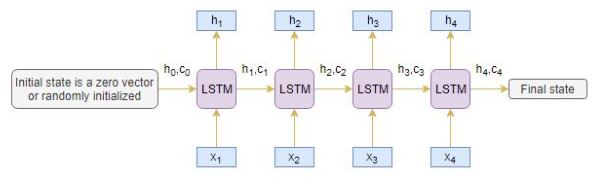


Рисунок 6-Схема работы LSTM кодировщика

Скрытое состояние (hi) и состояние ячейки (ci) последнего временного шага используются для инициализации декодера. Помните, это связано с тем, что кодировщик и декодер - это два разных набора архитектуры LSTM.

### Декодер

Декодер также представляет собой часть LSTM, которая по словам считывает всю целевую последовательность и предсказывает то же смещение последовательности на один временной шаг. Декодер обучен предсказывать следующее слово в последовательности с учетом предыдущего слова ().

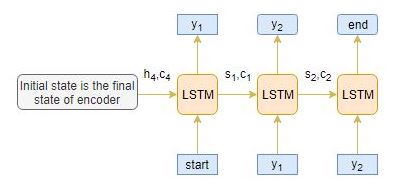


Рисунок 7-Схема работы LSTM декодера

<start> и <end> являются специальными токенами, которые добавляются к целевой последовательности перед подачей ее в декодер. Целевая последовательность неизвестна при декодировании тестовой последовательности. Итак, мы начинаем предсказывать целевую последовательность, передавая в декодер первое слово, которое всегда будет токеном <start>. И токен <end> сигнализирует об окончании предложения.

## Вывод

После обучения модель тестируется на новых входных последовательностях, для которых выход неизвестен. Итак, нам нужно настроить архитектуру вывода для декодирования тестовой последовательности (Рисунок 8).

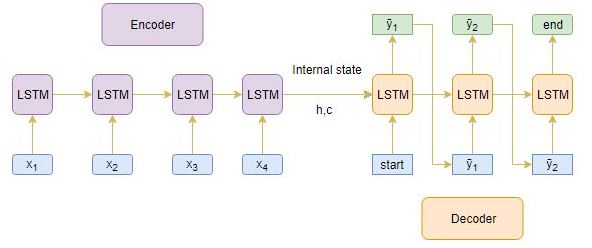


Рисунок 8-Общая схема LSTM

Алгоритм декодирования тестовой последовательности:

1. Кодировать всю входную последовательность и инициализировать декодер внутренними состояниями кодировщика.
2. Передать токен <start> как вход в декодер.
3. Запустить декодер на одну итерацию с внутренними состояниями.
4. Результатом будет вероятность следующего слова. Будет выбрано слово с максимальной вероятностью.
5. Передайте выбранное слово в качестве входных данных для декодера на следующем шаге и обновите внутренние состояния с текущим шагом.
6. Повторяйте шаги 3-5, пока мы не сгенерируем токен или не достигнем максимальной длины целевой последовательности.

Рассмотрим пример, в котором тестовая последовательность задается как [x1, x2, x3, x4].

Процесс вывода для тестовой последовательности.

1. Закодируйте тестовую последовательность во внутренние векторы состояния.
2. Посмотрите, как декодер предсказывает целевую последовательность на каждом временном шаге (Рисунок 9):

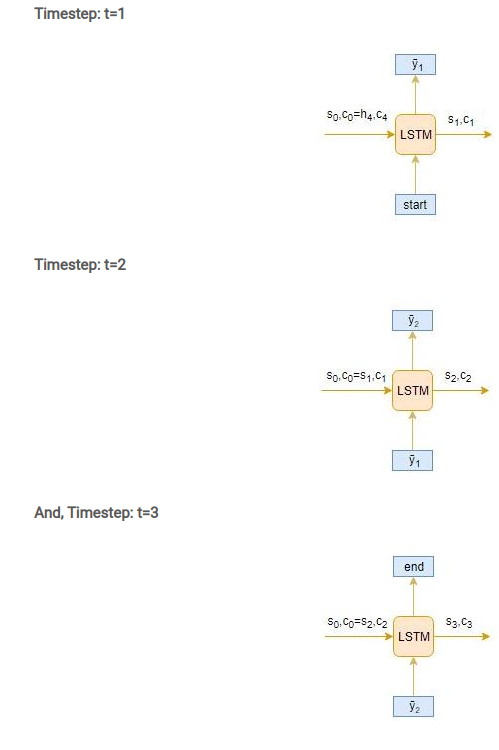


Рисунок 9-Процесс вывода тестовой последовательности

# Ограничения архитектуры кодировщика-дешифровщика

Какой бы полезной ни была эта архитектура кодировщика-декодера, она имеет определенные ограничения. Кодировщик преобразует всю входную последовательность в вектор фиксированной длины, а затем декодер предсказывает выходную последовательность. Это работает только для коротких последовательностей, поскольку декодер просматривает всю входную последовательность для предсказания. Здесь возникает проблема с длинными последовательностями. Кодировщику сложно запомнить длинные последовательности в вектор фиксированной длины.

Потенциальная проблема с этим подходом кодер-декодер заключается в том, что нейронная сеть должна иметь возможность сжимать всю необходимую информацию исходного предложения в вектор фиксированной длины. Это может затруднить работу нейронной сети с длинными предложениями. Производительность базового кодера-декодера быстро ухудшается по мере увеличения длины входного предложения.

Чтобы преодолеть проблему длинных последовательностей нужно использовать концепцию механизма внимания. Она нацелена на предсказание слова, глядя только на несколько конкретных частей последовательности, а не на всю последовательность.

# Концепция работы механизма Внимания

Сколько внимания нам нужно уделять каждому слову во входной последовательности для генерации слова на шаге t? Это ключевая интуиция, лежащая в основе концепции механизма внимания.

Давайте рассмотрим простой пример, чтобы понять, как работает механизм внимания: Исходная последовательность: «Which sport do you like the most?» Целевая последовательность: «I love cricket».

Первое слово «I» в целевой последовательности связано с четвертым словом «you» в исходной последовательности. Точно так же второе слово «love» в целевой последовательности связано с пятым словом «like» в исходной последовательности.

Таким образом, вместо того, чтобы смотреть на все слова в исходной последовательности, мы можем повысить важность определенных частей исходной последовательности, которые приводят к целевой последовательности. Это основная идея механизма внимания.

В зависимости от способа получения вектора контекста присутствия существует 2 различных класса механизмов внимания: глобальное внимание и локальное внимание.

## Глобальное внимание

Здесь внимание уделено всем исходным позициям. Другими словами, все скрытые состояния кодировщика учитываются для получения вектора сопровождаемого контекста (Рисунок 10).

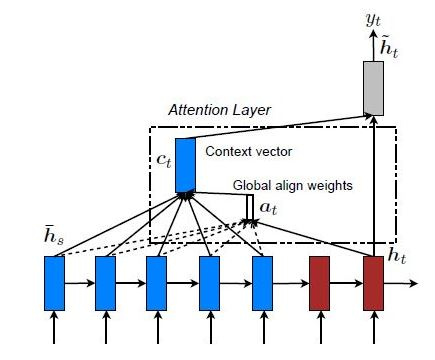


Рисунок 10-Схема глобального механизма внимания

## Локальное внимание

Здесь внимание уделено лишь нескольким исходным позициям. Только несколько скрытых состояний кодировщика учитываются для получения вектора сопровождаемого контекста (Рисунок 11).

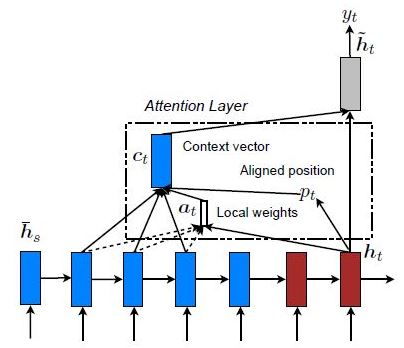


Рисунок 11-Схема локального механизма внимания

# Задача

# Реализация модели с использованием Keras

# Вывод

# Список литературы