

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное
учреждение высшего образования «Самарский национальный
исследовательский университет имени академика С.П. Королева
(Самарский университет)»

Институт _____ информатики, математики и электроники

Факультет _____ информатики

Кафедра _____ программных систем

ОТЧЁТ

_____ к курсовой работе по дисциплине
_____ «Нейронные сети глубокого обучения»

Студент _____ В.А. Артамонов

Студент _____ Д.А. Смирнов

Преподаватель _____ А.Н. Жданова

СОДЕРЖАНИЕ

1	Постановка задачи	4
2	Библиотеки python NLTK (Natural Language Toolkit) и Word2Vec	5
3	RNN (Recurrent neural network)	8
4	Вычислительные эксперименты	11
5	Результаты работы	24
6	Вывод.....	25
	Список использованных источников	26
	Приложение А.....	27
	Исходный код.....	27

ВВЕДЕНИЕ

Частотный анализ является одним из методов обработки текста на языке NLP (Neuro-linguistic programming). Результатом является список слов, наиболее часто встречающихся в тексте. Частотный анализ слов также позволяет получить представление о тематике и основных понятиях текста.

В NLP пометка POS подвергается синтаксическому анализу, где наша цель - понять роли, которые играют слова в предложении, взаимосвязь между словами и проанализировать грамматическую структуру предложений.

Пометка POS - это процесс пометки слова в корпусе соответствующей части речевого тега на основе его контекста и определения. Теги POS определяют лингвистическую роль слова в предложении.

Обработку текста на языке NLP проще всего производить с помощью языка программирования Python, поскольку он имеет развитую инфраструктуру, зарекомендовал себя в сфере анализа данных и машинного обучения. Так же сообществом разработано несколько библиотек для решения задач NLP на Python. Мы в своей работе будем использовать библиотеку NLTK для анализа текста [1].

1 Постановка задачи

Цель курсовой работы: по произвольному тексту составить словарь текста, определив слова по частоте их употребления, распределив по частям речи.

В данной курсовой работе мы изучим и используем три вида нейронных сетей для решения поставленной задачи такие как RNN, LSTM и GRU. И исходя из порученного времени на обучение каждой нейронной сети и их точности выберем самую оптимальную.

2 Библиотеки python NLTK (Natural Language Toolkit) и Word2Vec

NLTK (Natural Language Toolkit) – ведущая платформа для создания NLP-программ на языке программирования Python. У нее есть легкие в использовании интерфейсы для многих, а также библиотеки для обработки текстов и классификации, токенизации, стемминга, разметки, фильтрации и семантических рассуждений. Ну и еще это бесплатный опенсорсный проект, который развивается с помощью коммьюнити [2].

2.1 Токенизация по предложениям

Токенизация по предложениям – это процесс разделения письменного языка на предложения – компоненты. В английском и некоторых других языках мы можем выбирать предложение каждый раз, когда находим определенный знак пунктуации – точку.

Но даже в английском эта задача не такая простая, так как точка используется так же и в сокращениях. Таблица сокращений может помочь во время обработки текста, чтобы расставить границы предложений верно. В большинстве случаев для этого используются библиотеки.

Пример:

Возьмем текст:

Backgammon is one of the oldest known board games. Its history can be traced back nearly 5,000 years to archeological discoveries in the Middle East. It is a two player game where each player has fifteen checkers which move between twenty-four points according to the roll of two dice.

После использования токенизации на выходе мы получим 3 отдельных предложения:

Backgammon is one of the oldest known board games.

Its history can be traced back nearly 5,000 years to archeological discoveries in the Middle East.

It is a two player game where each player has fifteen checkers which move between twenty-four points according to the roll of two dice.

2.2 Токенизация по словам

Для токенизация по словам в многих языках в том числе и английском используют пробел как разделитель слов.

Хотя могут возникнуть проблемы, если мы будем использовать только пробел – так как в английском составные существительные пишутся по-разному и иногда через пробел.

Пример:

Возьмем предложения и применим к ним метод токенизации по словам:

['Backgammon', 'is', 'one', 'of', 'the', 'oldest', 'known', 'board', 'games', '.']

['Its', 'history', 'can', 'be', 'traced', 'back', 'nearly', '5,000', 'years', 'to', 'archeological', 'discoveries', 'in', 'the', 'Middle', 'East', '.']

['It', 'is', 'a', 'two', 'player', 'game', 'where', 'each', 'player', 'has', 'fifteen', 'checkers', 'which', 'move', 'between', 'twenty-four', 'points', 'according', 'to', 'the', 'roll', 'of', 'two', 'dice', '.']

2.3 Лемматизация и стемминг текста

Тексты содержат разные грамматические формы одного и того же слова, а также встречаются и однокоренные слова. Лемматизация и стемминг способны привести все встречающиеся словоформы к одной, нормальной словарной форме.

Примеры:

Приведение разных словоформ к одной:

dog, dogs, dog's, dogs' => dog

Пример целого предложения:

the boy's dogs are different sizes => the boy dog be differ size

2.4 Стоп – слова

Стоп-слова – это слова, которые удаляются из текста до и после обработки текста. Применяя машинное обучение к текстам, такие слова

могут добавить много лишнего шума, поэтому необходимо избавляться от ненужных слов.

Стоп-слова это обычно понимают междометия, артикли, союзы и т.д., которые не несут смысловой нагрузки и не нужны для нашей задачи. При этом надо понимать, что не существует всеобщего списка стоп-слов, все зависит от конкретного случая [2].

2.5 Word2vec

Word2Vec работает с огромным текстовым корпусом и по определенным правилам присваивает каждому слову индивидуальный набор чисел — так называемый семантический вектор. Это так же как складывать и вычитать вектора так же, если из вектора слова *король* вычесть вектор *мужчина* и прибавить вектор *женщина*, получатся числа, соответствующие слову *королева*.

Вся идея с семантическими векторами основана на дистрибутивной гипотезе. Она состоит в том, что смысл слова заключается не в наборе его собственных букв и звуков, а в том, среди каких слов оно чаще всего встречается. Получается смысл слова не хранится где-то внутри него, он распределен между элементами его возможных контекстов, отсюда и следует название — дистрибутивная гипотеза [3].

3 RNN (Recurrent neural network)

RNN (рекуррентная нейронная сеть), так как такая сеть имеет «память» и в основном используется в обработке естественного языка (NLP) и речи. Конкретная форма рекуррентной нейронной сети (RNN) заключается в том, что сеть сохраняет предыдущую информацию и применяет ее к вычислению данного выхода, то есть узлы между скрытыми слоями больше не не соединены, а соединены, а вход скрытого слоя включает не только входной слой, а вывод также включает вывод скрытого слоя в предыдущий момент.

В теории RNN может обрабатывать данные любой длины, но, поскольку структура сети имеет проблему «исчезновения градиента», в практических приложениях методы решения проблемы исчезновения градиента включают: Clipping Gradient и LSTM (Long Short-Term Memory) [4].

На рисунке 1 представлена простая классическая структура RNN:

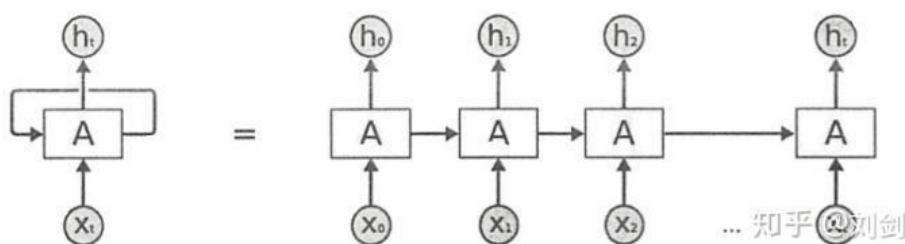


Рисунок 1 – Классическая структура RNN

LSTM был предложен "Hochreiter & Schmidhuber" в 1997 году и стал стандартной формой RNN для решения упомянутой выше проблемы "долгосрочной зависимости" модели RNN (Рисунок 2).

Long short Term Memory

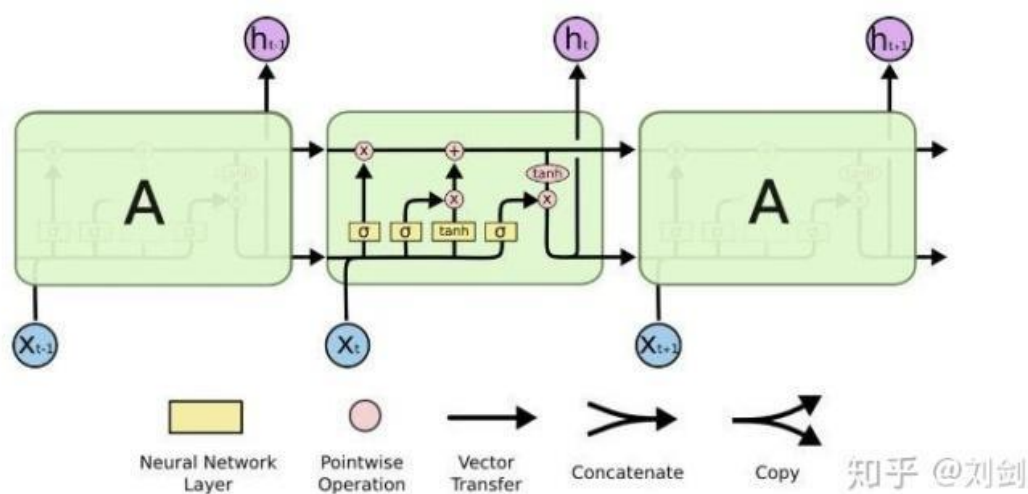


Рисунок 2 – LSTM

LSTM контролирует состояние и вывод в разное моменты времени через три «вентильные» структуры. А структура «ворот» представляет собой полностью связанную нейронную сеть, которая использует функцию активации сигмоида и операцию побитового умножения. Функция активации сигмоида выдает значение от 0 до 1. Это значение передает, сколько информации сколько может пройти ток » Дверь », 0 означает, если информация не может пройти, 1 означает, что вся информация имеет возможность пройти. Среди них «забытый вентиль» и «входной вентиль» есть центр или так называемое ядро структуры модуля LSTM. Разберем подробно следующие три «дверные» конструкции.

- Шлюз забывания, он нужен для того, чтобы LSTM «забыл» ранее ненужную информацию. Он определит, какая информация будет удалена (забыта), на основе входа X_t узла в текущий момент времени, состояние узла в предыдущий момент времени S_{t-1} и выходных данных узла в предыдущий момент времени h_{t-1} .

- Входной шлюз, LSTM, необходим для решения, какая информация останется. После того, как LSTM использует шлюз забывания, чтобы «забыть» часть информации, так же нужно оставить последнюю память из текущего входа. Входной вентиль будет решать, какая информация войдет в

текущее состояние узла C_t в соответствии с входом X_t узла в текущий момент времени, состоянием узла в предыдущий момент времени C_{t-1} и выходом узла в предыдущий момент времени h_{t-1} .

- Выходной вентиль, после того как LSTM получает последнее состояние узла C_t , он объединяет выходные данные узла в предыдущий момент времени h_{t-1} и входные данные X_t узла в текущее время, чтобы определить выход узла в текущее время.

GRU (Gated Recurrent Unit) – архитектура RNN, предложенная в 2014 году. Это облегченная версия LSTM. На рисунке 3 приводится структурная сравнительная диаграмма LSTM и GRU [4]:

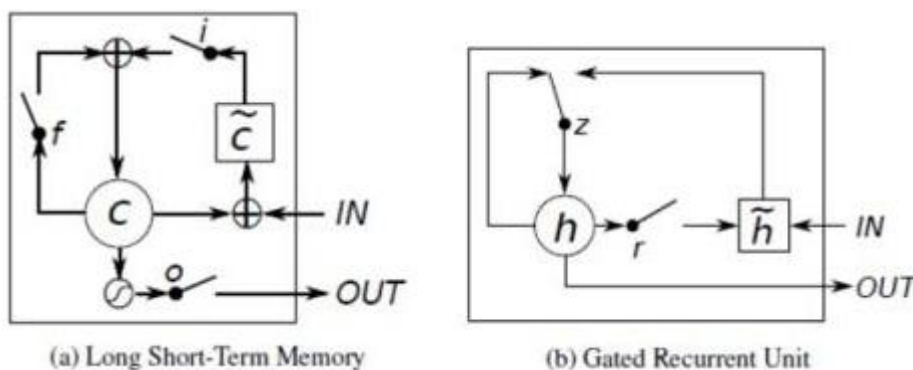
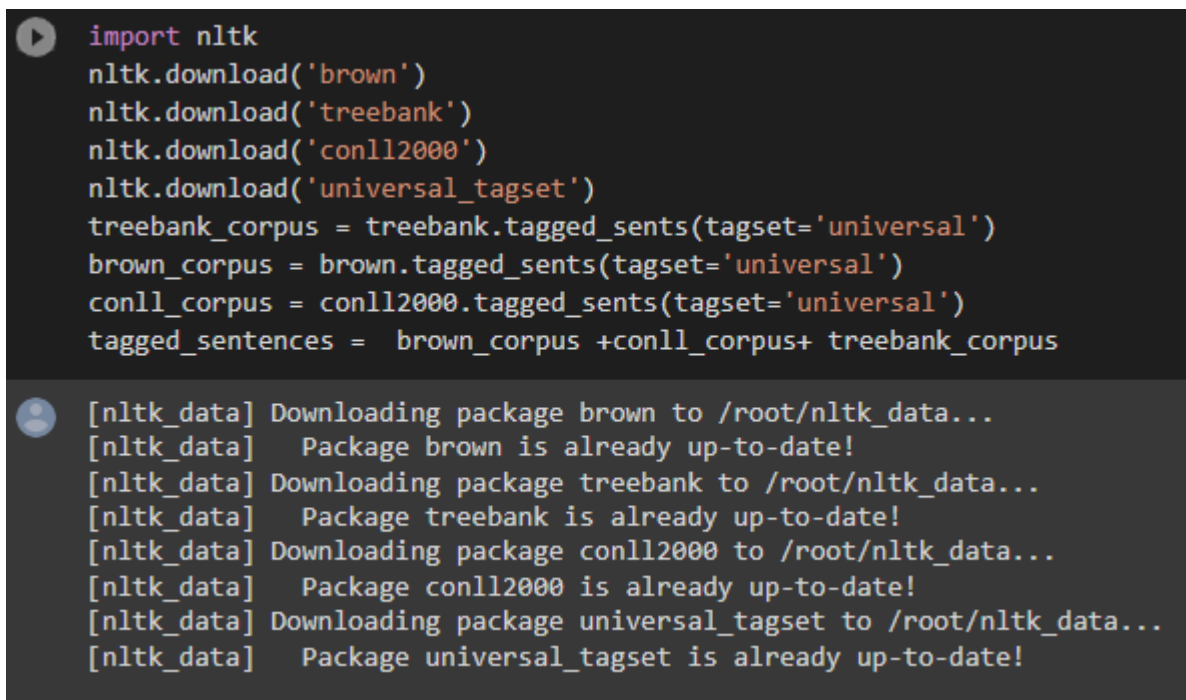


Рисунок 3 – Структурная сравнительная диаграмма LSTM и GRU

4 Вычислительные эксперименты

4.1 Импортирование библиотек



```
import nltk
nltk.download('brown')
nltk.download('treebank')
nltk.download('conll2000')
nltk.download('universal_tagset')
treebank_corpus = treebank.tagged_sents(tagset='universal')
brown_corpus = brown.tagged_sents(tagset='universal')
conll_corpus = conll2000.tagged_sents(tagset='universal')
tagged_sentences = brown_corpus + conll_corpus + treebank_corpus
```

```
[nltk_data] Downloading package brown to /root/nltk_data...
[nltk_data] Package brown is already up-to-date!
[nltk_data] Downloading package treebank to /root/nltk_data...
[nltk_data] Package treebank is already up-to-date!
[nltk_data] Downloading package conll2000 to /root/nltk_data...
[nltk_data] Package conll2000 is already up-to-date!
[nltk_data] Downloading package universal_tagset to /root/nltk_data...
[nltk_data] Package universal_tagset is already up-to-date!
```

Рисунок 4 – Импортирование библиотек NLTK

4.2 Предварительная обработка данных

Разделим данные на слова и теги, а также выполним векторизацию X и Y и запеним последовательности.

Поскольку это проблема "многие ко многим", каждая точка данных будет представлять собой отдельное предложение корпусов.

Каждая точка данных будет содержать несколько слов во входной последовательности. Это то, что мы будем называть X .

Каждое слово будет иметь свой соответствующий тег в выходной последовательности. Это то, что мы будем называть Y [4].

Набор данных представлен на рисунке 5:

```
print('sample X: ', X[0], '\n')
print('sample Y: ', Y[0], '\n')

sample X:  ['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', 'Friday', 'an', 'investigation',
sample Y:  ['DET', 'NOUN', 'NOUN', 'ADJ', 'NOUN', 'VERB', 'NOUN', 'DET', 'NOUN', 'ADP', 'NOUN',
```

Рисунок 5 – Набор данных (X) (Y)

Закодируем X и Y в целочисленные значения

Мы будем использовать функцию Tokenizer() из библиотеки Keras для кодирования текстовой последовательности в целочисленную последовательность (Рисунок 6).

```
** Raw data point **
-----
X:  ['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', 'Friday', 'an', 'investigation', 'of', 'Atlanta's', 'recent', 'primary', 'election',
Y:  ['DET', 'NOUN', 'NOUN', 'ADJ', 'NOUN', 'VERB', 'NOUN', 'DET', 'NOUN', 'ADP', 'NOUN', 'ADJ', 'NOUN', 'NOUN', 'VERB', '.', 'DET', 'NOUN', '.']

** Encoded data point **
-----
X:  [1, 5731, 778, 2326, 1842, 39, 853, 34, 1944, 4, 16831, 379, 1343, 1523, 1116, 12, 67, 569, 14, 9, 89, 10208, 252, 205, 3]
Y:  [5, 1, 1, 6, 1, 2, 1, 5, 1, 4, 1, 6, 1, 1, 2, 3, 5, 1, 3, 4, 5, 1, 2, 1, 3]
```

Рисунок 6 – Закодированный набор данных (X) (Y)

Следующим шагом после кодирования данных является определение длины последовательности. На данный момент предложения, представленные в данных, имеют различную длину. Нам нужно либо дополнить короткие предложения, либо сократить длинные предложения до фиксированной длины. Однако эта фиксированная длина является гиперпараметром.

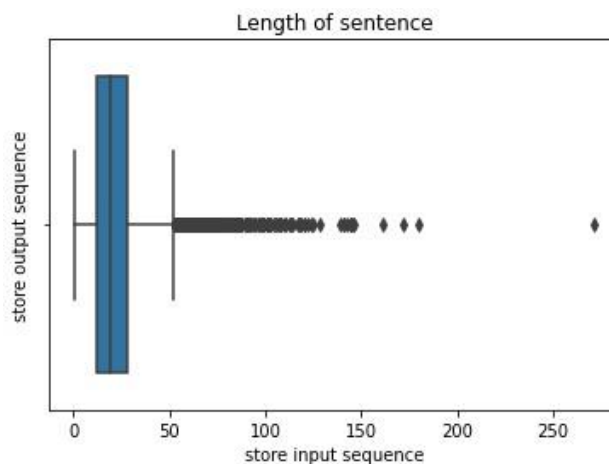


Рисунок 7 – График длины предложений

В настоящее время каждое слово и каждый тег кодируются как целое число.

Мы будем использовать более сложную технику для представления входных слов (X), используя так называемые вложения слов.

Однако, чтобы представить каждый тег в Y, мы просто будем использовать схему однократного кодирования, поскольку в наборе данных всего 13 тегов, и у LSTM не будет проблем с изучением собственного представления этих тегов.

На данный момент каждое слово и каждый тег кодируются как целое число. Мы будем использовать более сложную технику для представления входных слов (X) с использованием так называемых вложений слов. Мы используем модель word2vec (king-man = queen)

```
[ ] EMBEDDING_SIZE = 300 # each word in word2vec model is represented using a 300 dimensional vector
    VOCABULARY_SIZE = len(word_tokenizer.word_index) + 1

    # create an empty embedding matrix
    embedding_weights = np.zeros((VOCABULARY_SIZE, EMBEDDING_SIZE))

    # create a word to index dictionary mapping
    word2id = word_tokenizer.word_index

    # copy vectors from word2vec model to the words present in corpus
    for word, index in word2id.items():
        try:
            embedding_weights[index, :] = word2vec[word]
        except KeyError:
            pass
```

Рисунок 8 – Модель Word2vec

На данном этапе мы определили размер вложения как 300. Теперь вложение имеет свой вес и другой уровень.

Перед использованием RNN мы должны убедиться, что размеры данных соответствуют ожиданиям RNN. В общем, RNN ожидает следующую форму

Shape of X: (#samples, #timesteps, #features)

Shape of Y: (#samples, #timesteps, #features)

Теперь могут быть различные варианты формы, которые вы используете для подачи RNN, в зависимости от типа архитектуры.

4.3 Разделение данных в наборах обучения, проверки и тестирования

На рисунке ниже показаны три датасета: brown (обучение), conll2000 (валидация и проверка) и threebank (тестирование).

```
TRAINING DATA
Shape of input sequences: (52165, 100)
Shape of output sequences: (52165, 100, 13)
-----
VALIDATION DATA
Shape of input sequences: (9206, 100)
Shape of output sequences: (9206, 100, 13)
-----
TESTING DATA
Shape of input sequences: (10831, 100)
Shape of output sequences: (10831, 100, 13)
```

Рисунок 9 – Датасеты

4.4 Обучение RNN (Recurrent neural network)

Теперь, во время обучения модели, мы также можем обучать вложения слов вместе с весами сети. Их часто называют весами вложения. Во время обучения веса встраивания будут обрабатываться как обычные веса сети, которые обновляются на каждой итерации. Если мы не позволим им тарировать, мы получим меньшую точность.

4.4.1 Измерение гиперпараметров

Ниже представлен график сравнения моделей RNN (rnn_model, rnn_model_t1 и rnn_model_t3)

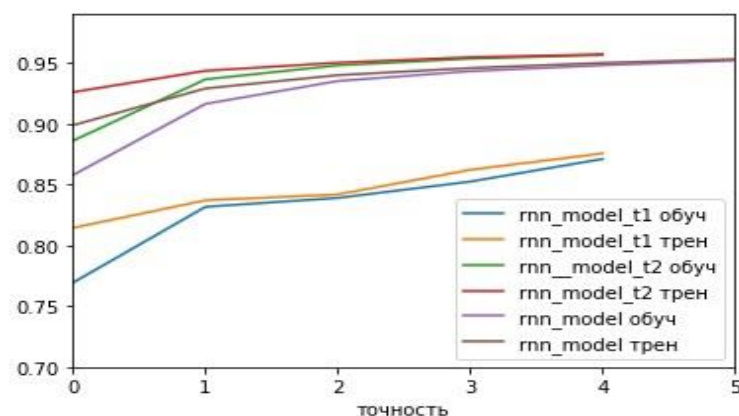


Рисунок 10 – График сравнения моделей RNN

Таблица 1 – Гиперданные моделей RNN

	Epochs	batch_size
rnn_model	10	128
rnn_model_t1	5	128
rnn_model_t2	5	64
rnn_model_t3	10	64

Исходя из рисунка 10 и таблицы 1, который представлен выше можно сказать что модель `rnn_model_t2` показала лучшие результаты. Осталось сравнить с финальной сборкой рекуррентной нейронной сети.

На рисунке ниже представлен график сравнения `rnn_model_t3` и `rnn_model_t2`

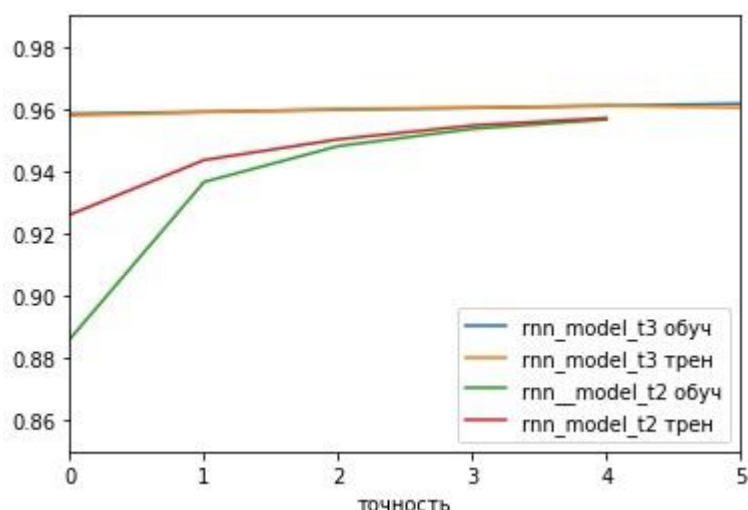


Рисунок 11 – График сравнения моделей RNN

Из графика выше видно, что модель `rnn_model_t3` показала лучшие результаты, поэтому для нас интересны именно ее гиперданные (64 `batch_size` и 10 `epochs`).

В последующих моделях будем применять такие `EPOCHS` и `BATCH_SIZE` при обучении LSTM, GRU и Bidirectional LSTM

4.4.2 Функция потерь

Функция потерь – функция, что показывает результат, оценку, на сколько хорошо справился классификатор на обучающей выборке. Был выбран `sparse_categorical_crossentropy` в качестве функции потерь.

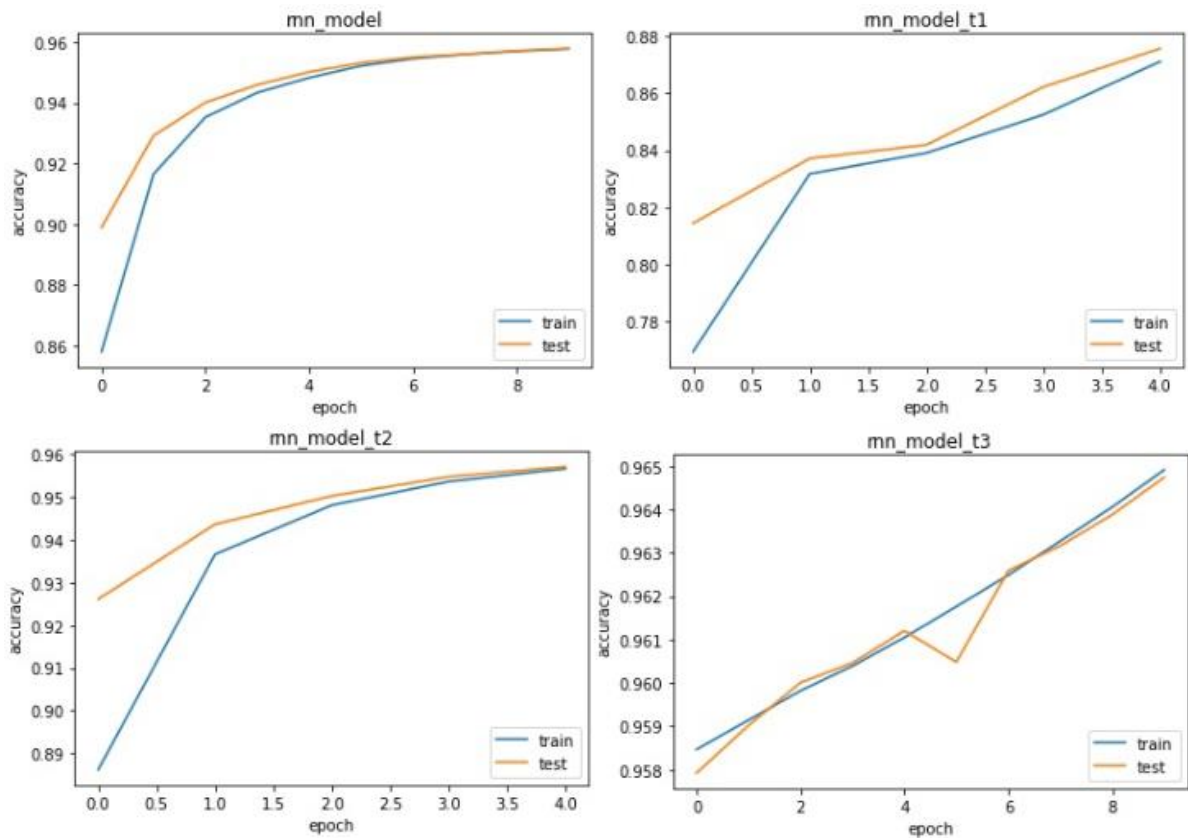


Рисунок 12 – Графики потерь

Таблица 2 – Значение векторов потерь

Epochs	rnn_model	rnn_model_t1	rnn_model_t2	rnn_model_t3
1	0.8988659	0.8144069	0.9260960	0.9579219
2	0.9290820	0.8370779	0.9436389	0.9590180
3	0.9400532	0.8418803	0.9502737	0.9600032
4	0.9458701	0.8621637	0.9547512	0.9604594
5	0.9501118	0.8756126	0.9571464	0.9611992
6	0.9531066	-	-	0.9604790
7	0.9549804	-	-	0.9625841
8	0.9559156	-	-	0.9631599
9	0.9570986	-	-	0.9638768
10	0.9578242	-	-	0.9647371

Исходя из рисунка 12 и таблицы 2, который представлен выше можно сказать что модель `rnn_model_t3` показала лучшие результаты и поэтому мы берем ее гиперданные для обучения последующих моделей.

На рисунках ниже показана скомпилированная модель и модель сходимости RNN.


```
rnn_model_t3.summary()
```

```
Model: "sequential_3"
```

Layer (type)	Output Shape	Param #
embedding_3 (Embedding)	(None, 100, 300)	17834700
simple_rnn_3 (SimpleRNN)	(None, 100, 64)	23360
time_distributed_3 (TimeDistributed)	(None, 100, 13)	845

```

Total params: 17,858,905
Trainable params: 17,858,905
Non-trainable params: 0

```

Рисунок 13 – Скомпилированная модель RNN

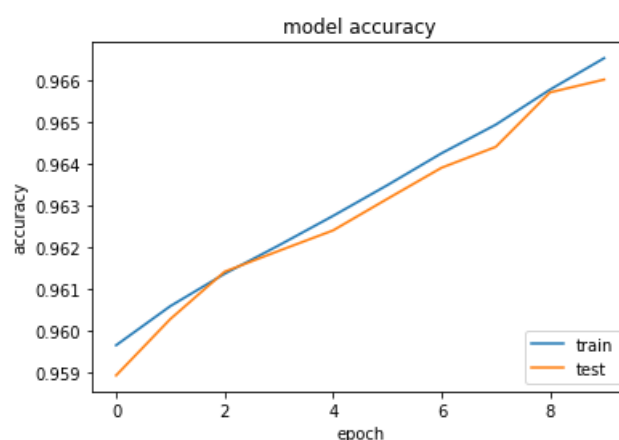


Рисунок 14 – Модель сходимости RNN

4.5 Обучение с помощью LSTM (долговременная сеть кратковременной памяти)

Основное радикальное улучшение, которое принесли LSTM, связано с новым изменением в структуре самого нейрона. Это было введено для решения проблемы увеличения / уменьшения градиента в RNN. В случае LSTM нейроны называются клетками, и клетка LSTM отличается от нормального нейрона во многих отношениях.

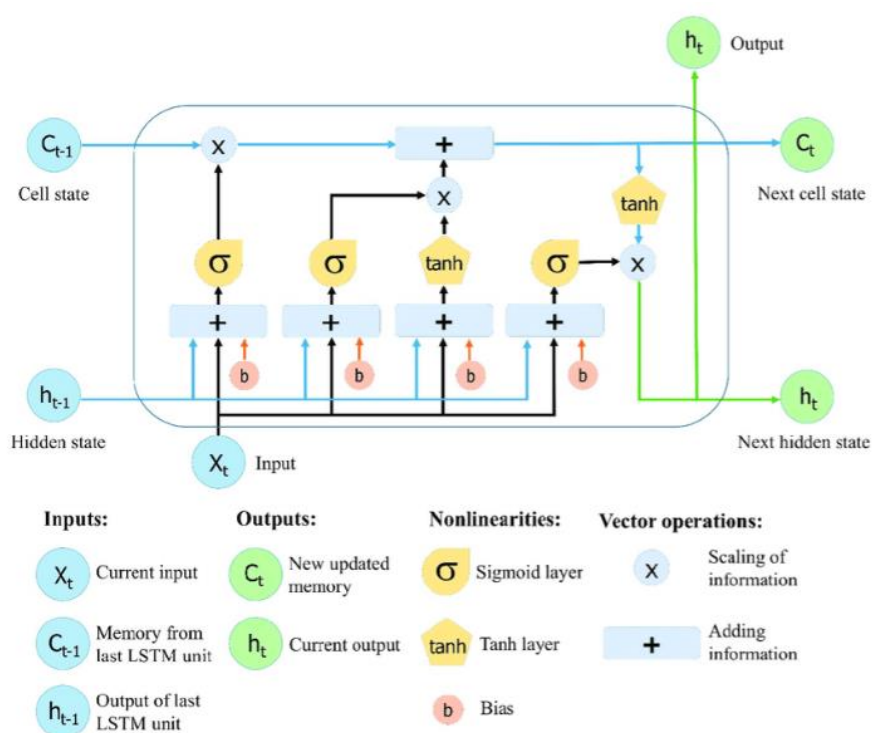


Рисунок 15 – Схема LSTM

LSTM имеет явный блок памяти, в котором хранится информация, относящаяся к изучению некоторой задачи, он имеет стробирующие механизмы, регулирующие информацию структура ячейки LSTM, позволяет сети LSTM иметь плавный и непрерывный поток градиентов при обратном распространении, который сеть сохраняет (и передает на следующий уровень) или забывает. Этот поток также называется каруселью постоянных ошибок.

На рисунках ниже показана скомпилированная модель и модель сходимости LSTM:

```
lstm_model.summary()

Model: "sequential_4"
Layer (type)                Output Shape              Param #
-----
embedding_4 (Embedding)     (None, 100, 300)         17834700
lstm (LSTM)                  (None, 100, 64)          93440
time_distributed_4 (TimeDis  (None, 100, 13)          845
tributed)

Total params: 17,928,985
Trainable params: 17,928,985
Non-trainable params: 0
```

Рисунок 16 – Скомпилированная модель LSTM

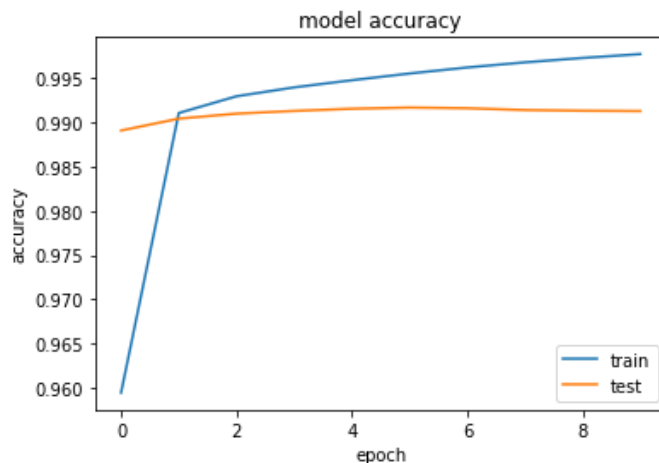


Рисунок 17 – Модель сходимости LSTM

4.6 Обучение с GRU (закрытое повторяющееся подразделение)

GRU, известный как закрытый рекуррентный модуль, представляет собой архитектуру RNN, которая похожа на модули LSTM. GRU состоит из элемента сброса и элемента обновления вместо элемента ввода, вывода и забывания LSTM. Элемент сброса определяет, как объединить новый ввод с предыдущей памятью, а элемент обновления определяет, какой объем предыдущей памяти следует сохранить [5].

На рисунках ниже представлена скомпилированная модель и модель сходимости GRU:

```
gru_model.summary()
```

Model: "sequential_5"		
Layer (type)	Output Shape	Param #
embedding_5 (Embedding)	(None, 100, 300)	17834700
gru (GRU)	(None, 100, 64)	70272
time_distributed_5 (TimeDistributed)	(None, 100, 13)	845
Total params: 17,905,817		
Trainable params: 17,905,817		
Non-trainable params: 0		

Рисунок 18 – Скомпилированная модель GRU

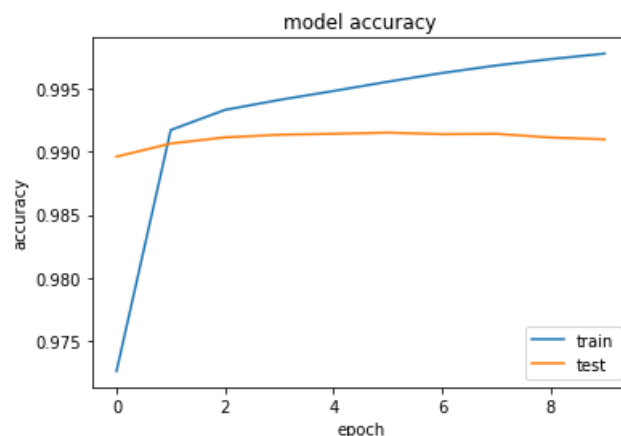


Рисунок 19 – Модель сходимости GRU

На рисунках представлена скомпилированная модель и модель сходимости Bidirectional LSTM:

```
bidirect_model.summary()

Model: "sequential_6"
-----
Layer (type)                Output Shape          Param #
-----
embedding_6 (Embedding)      (None, 100, 300)      17834700
bidirectional (Bidirectiona  (None, 100, 128)      186880
l)
time_distributed_6 (TimeDis  (None, 100, 13)       1677
tributed)
-----
Total params: 18,023,257
Trainable params: 18,023,257
Non-trainable params: 0
```

Рисунок 20 – Скомпилированная модель Bidirectional LSTM

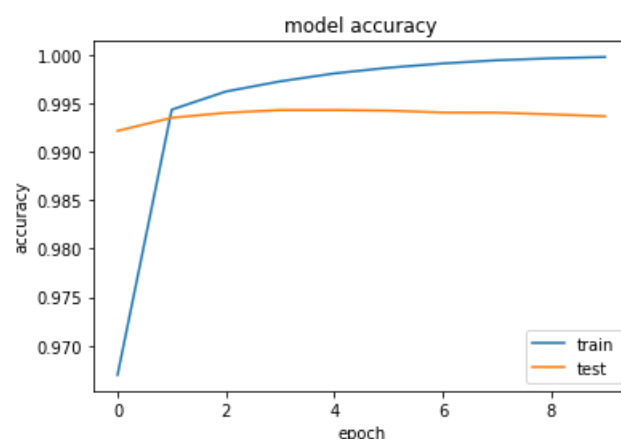


Рисунок 21 – Модель сходимости Bidirectional LSTM

Судя по графикам, выборка гиперпараметров была мала и количество обучаемых моделей rnn превышало необходимое. Из-за чего модели начали

«передумывать» себя. Осталось провести последний этап – запуск модели на тестовых данных.

На рисунке 22 изображены итоговые значения тестовых испытаний моделей, на котором видно, что у двунаправленной LSTM показатели значительно лучше, чем у половины испытываемых моделей.

Таблица 3 – Итоговые значения испытания моделей на тестовых данных

Epochs	rnn_model_t3	lstm_model	gru_model	bidirect_model
1	0.9579219	0.9889973	0.9894633	0.9920899
2	0.9590180	0.9906267	0.9906028	0.9935259
3	0.9600032	0.9910210	0.9910601	0.9938681
4	0.9604594	0.9913045	0.9912220	0.9941603
5	0.9611992	0.9913936	0.9913132	0.9941940
6	0.9604790	0.9915902	0.9912763	0.9941885
7	0.9625841	0.9914588	0.9911883	0.9936367
8	0.9631599	0.9911590	0.9910373	0.9939452
9	0.9638768	0.9911535	0.9910058	0.9938105
10	0.9647371	0.9908689	0.9908418	0.9937595

Двунаправленный LSTM значительно повысил точность (учитывая, что точность уже достигла предела). Это показывает мощь двунаправленных LSTM. Время в полтора раза больше предыдущих моделей, но и разница в точности на таком уровне – запредельная.

Getdict ('ссылка на файл расширением .txt') - функция выдающая словарь с частотным анализом слов из выборки.

На рисунке 23 и таблице 4 будет показан словарь и его график

getdictWithoutStopWords ('ссылка на файл расширением .txt') - функция выдающая словарь с частотным анализом слов из выборки (без стоп-слов)

На рисунке 24 и таблице 5 будет показан словарь (без стоп-слов) и его график

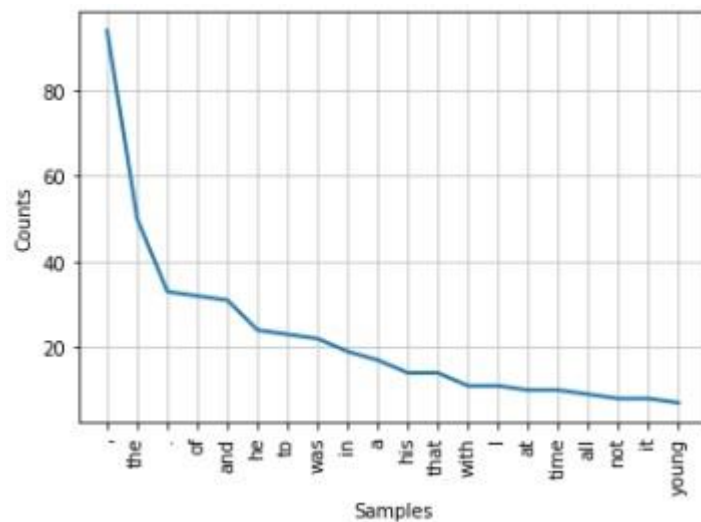


Рисунок 23 – Словарь частоты употребления частей речи

Таблица 4 – Словарь на 20 токенов

Номер токена	По частоте употребления	По частям речи
1	, (94)	At (ADP)
2	the (50)	the (DET)
3	. (33)	beginning (VERB)
4	of (32)	of (ADP)
5	and (31)	July (NOUN)
6	he (24)	, (,)
7	to (23)	at (ADP)
8	was (22)	an (DET)
9	in (19)	extremely (ADVERB)
10	a (17)	hot (ADJ)
11	hes (14)	time (NOUN)
12	that (14)	, (,)
13	with (11)	in (ADP)
14	I (11)	the (DET)
15	at (10)	evening (VERB)
16	time (10)	, (,)
17	all (9)	one (NUM)
18	not (8)	young (ADJ)
19	it (8)	man (NOUN)
20	young (7)	came (VERB)

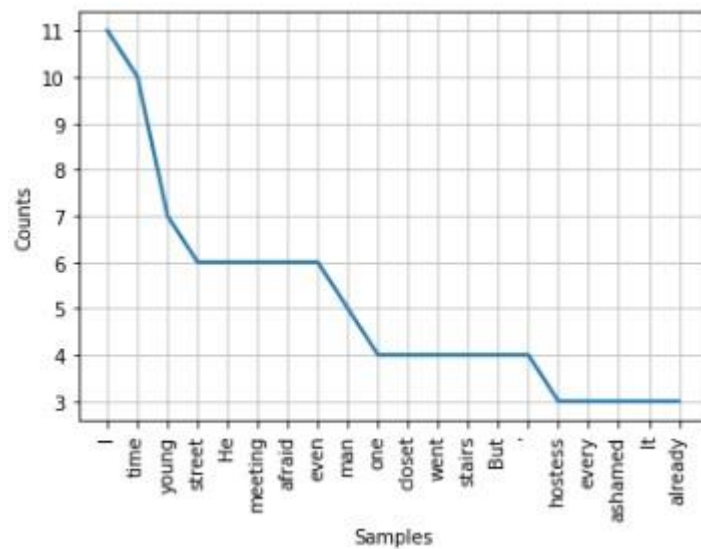


Рисунок 24 – Словарь частоты употребления частей речи без стоп-слов

Таблица 5 – Словарь на 20 токенов без стоп-слов

Номер токена	По частоте употребления	По частям речи
1	I (11)	At (ADP)
2	time (10)	the (DET)
3	young (7)	beginning (VERB)
4	street (6)	of (ADP)
5	He (6)	July (NOUN)
6	meeting (6)	, (,)
7	afraid (6)	at (ADP)
8	even (6)	an (DET)
9	man (5)	extremely (ADVERB)
10	one (4)	hot (ADJ)
11	closet (4)	time (NOUN)
12	went (4)	, (,)
13	stairs (4)	in (ADP)
14	But (4)	the (DET)
15	' (4)	evening (VERB)
16	hostess (3)	, (,)
17	every (3)	one (NUM)
18	ashamed (3)	young (ADJ)
19	it (3)	man (NOUN)
20	already (3)	came (VERB)

5 Результаты работы

Была построена и обучена модель которая по произвольному тексту составляет словарь текста и определяет слова по частоте их употребления, распределив по частям речи.

Исходя из графиков, полученных в результате работы можно сделать вывод что разница во времени значительная, исходя из ее архитектуры (Двунаправленный LSTM), точность гораздо выше, несмотря на предельный порог, и регрессия в последних моделях показала, что выборка гиперданных была мала и было использовано излишнее количество тренировочных RNN. Также, не использовался GPU для расчетов, поэтому времени на обучение ушло гораздо больше.

Исходный код курсовой работы представлен в приложении А и доступен по ссылке – <https://colab.research.google.com/drive/1Kkspc7FoalbEdSK2JzFvFhDVIU9Y4Wb3?usp=sharing#scrollTo=C04p1Wleh7i3>

6 Вывод

В ходе выполнения курсового проектирования были изучены возможности рекуррентных нейронных сетей и библиотек NLTK.

В первой главе приведена математическая постановка задачи и ее обоснование относительно поставленных целей в курсовом проекте.

В второй главе приведены теоретические обоснования и принципы работы python библиотек NLTK и Word2Vec в работе с текстом.

В третьей главе говорится об принципах работы, архитектуры, структуры, параметров и методов обучения рекуррентных нейронных сетей.

В четвертой главе приводятся описания вычислительных экспериментов, выходные данные, графики и таблицы, с целью эффективной оценки модели и ее сравнительный анализ.

Разработанная система может быть использована для получения словарей с частотным анализом используемых слов с выделением по частям речи, по ссылке на англоязычный текст.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 Частотный анализ текста [Электронный ресурс]. URL: <https://habr.com/ru/post/517410/> (дата обращения: 18.06.2022).
- 2 Основы Natural Language Processing для текста [Электронный ресурс]. URL: <https://habr.com/ru/company/Voximplant/blog/446738/> (дата обращения: 18.06.2022).
- 3 Word2vec [Электронный ресурс]. URL: <https://sysblok.ru/knowhow/word2vec-pokazhi-mne-svoj-kontekst-i-ja-skazhu-kto-ty/>
- 4 Принципы RNN [Электронный ресурс]. URL: <https://russianblogs.com/article/61561184544/> (дата обращения: 18.06.2022).
- 5 Маркировка POS с использованием вариантов RNN [Электронный ресурс]. URL: <https://iprathore71.medium.com/pos-tagging-using-rnn-variants-53311de58ca9> (дата обращения: 18.06.2022).

Приложение А

Исходный код

```
import warnings
warnings.filterwarnings("ignore")

import numpy as np

from matplotlib import pyplot as plt

from nltk.corpus import brown
from nltk.corpus import treebank
from nltk.corpus import conll2000

import seaborn as sns

from gensim.models import KeyedVectors

from keras.preprocessing.sequence import pad_sequences
from keras.utils.np_utils import to_categorical
from keras.models import Sequential
from keras.layers import Embedding
from keras.layers import Dense, Input
from keras.layers import TimeDistributed
from keras.layers import LSTM, GRU, Bidirectional, SimpleRNN, RNN
from keras.models import Model
from keras.preprocessing.text import Tokenizer

from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle

import nltk
nltk.download('brown')
nltk.download('treebank')
nltk.download('conll2000')
nltk.download('universal_tagset')
treebank_corpus = treebank.tagged_sents(tagset='universal')
brown_corpus = brown.tagged_sents(tagset='universal')
conll_corpus = conll2000.tagged_sents(tagset='universal')
tagged_sentences = brown_corpus + conll_corpus + treebank_corpus

tagged_sentences[2]

X = [] # store input sequence
Y = [] # store output sequence
```

```

for sentence in tagged_sentences:
    X_sentence = []
    Y_sentence = []
    for entity in sentence:
        X_sentence.append(entity[0]) # entity[0] contains the word
        Y_sentence.append(entity[1]) # entity[1] contains corresponding tag

    X.append(X_sentence)
    Y.append(Y_sentence)

num_words = len(set([word.lower() for sentence in X for word in sentence]))
num_tags = len(set([word.lower() for sentence in Y for word in sentence]))

print("Total number of tagged sentences: {}".format(len(X)))
print("Vocabulary size: {}".format(num_words))
print("Total number of tags: {}".format(num_tags))

print('sample X: ', X[0], '\n')
print('sample Y: ', Y[0], '\n')

print("Length of first input sequence : {}".format(len(X[0])))
print("Length of first output sequence : {}".format(len(Y[0])))

word_tokenizer = Tokenizer()
word_tokenizer.fit_on_texts(X)
X_encoded = word_tokenizer.texts_to_sequences(X)

tag_tokenizer = Tokenizer()
tag_tokenizer.fit_on_texts(Y)
Y_encoded = tag_tokenizer.texts_to_sequences(Y)

print("*** Raw data point ***", "\n", "-"*100, "\n")
print('X: ', X[0], '\n')
print('Y: ', Y[0], '\n')
print()
print("*** Encoded data point ***", "\n", "-"*100, "\n")
print('X: ', X_encoded[0], '\n')
print('Y: ', Y_encoded[0], '\n')

different_length = [1 if len(input) != len(output) else 0 for input, output in zip(X_encoded, Y_encoded)]
print("{} sentences have disparate input-output lengths.".format(sum(different_length)))

lengths = [len(seq) for seq in X_encoded]
print("Length of longest sentence: {}".format(max(lengths)))

sns.boxplot(lengths)
plt.show()

```

```

MAX_SEQ_LENGTH = 100 # sequences greater than 100 in length will be truncated

X_padded = pad_sequences(X_encoded, maxlen=MAX_SEQ_LENGTH, padding="pre", truncating="post")
Y_padded = pad_sequences(Y_encoded, maxlen=MAX_SEQ_LENGTH, padding="pre", truncating="post")

print(X_padded[0], "\n"*3)
print(Y_padded[0])

X, Y = X_padded, Y_padded

path = "/content/drive/MyDrive/LrNK/GoogleNews-vectors-negative300.bin"
word2vec = KeyedVectors.load_word2vec_format(path, binary=True, limit=100000)

word2vec.most_similar(positive = ["King", "Woman"], negative = ["Man"])

EMBEDDING_SIZE = 300 # each word in word2vec model is represented using a 300 dimensional vector
VOCABULARY_SIZE = len(word_tokenizer.word_index) + 1

# create an empty embedding matrix
embedding_weights = np.zeros((VOCABULARY_SIZE, EMBEDDING_SIZE))

# create a word to index dictionary mapping
word2id = word_tokenizer.word_index

# copy vectors from word2vec model to the words present in corpus
for word, index in word2id.items():
    try:
        embedding_weights[index, :] = word2vec[word]
    except KeyError:
        pass

print("Embeddings shape: {}".format(embedding_weights.shape))

embedding_weights[word_tokenizer.word_index['joy']]

Y = to_categorical(Y)

print(Y.shape)

TEST_SIZE = 0.15
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=TEST_SIZE, random_state=4)

VALID_SIZE = 0.15
X_train, X_validation, Y_train, Y_validation = train_test_split(X_train, Y_train, test_size=VALID_SIZE,
random_state=4)

print("TRAINING DATA")
print('Shape of input sequences: {}'.format(X_train.shape))

```

```

print('Shape of output sequences: {}'.format(Y_train.shape))
print("-"*50)
print("VALIDATION DATA")
print('Shape of input sequences: {}'.format(X_validation.shape))
print('Shape of output sequences: {}'.format(Y_validation.shape))
print("-"*50)
print("TESTING DATA")
print('Shape of input sequences: {}'.format(X_test.shape))
print('Shape of output sequences: {}'.format(Y_test.shape))

NUM_CLASSES = Y.shape[2]

rnn_model = Sequential()

# create embedding layer - usually the first layer in text problems
rnn_model.add(Embedding(input_dim = VOCABULARY_SIZE,      # vocabulary size - number of unique
words in data
                        output_dim = EMBEDDING_SIZE,      # length of vector with which each word is represented
                        input_length = MAX_SEQ_LENGTH,     # length of input sequence
                        trainable = False                 # False - don't update the embeddings
))

# add an RNN layer which contains 64 RNN cells
rnn_model.add(SimpleRNN(64,
                        return_sequences=True # True - return whole sequence; False - return single output of the end of the
sequence
))

# add time distributed (output at each sequence) layer
rnn_model.add(TimeDistributed(Dense(NUM_CLASSES, activation='softmax'))

rnn_model.compile(loss = 'categorical_crossentropy',
                  optimizer = 'adam',
                  metrics = ['acc'])

rnn_model.summary()

rnn_training = rnn_model.fit(X_train, Y_train, batch_size=128, epochs=10, validation_data=(X_validation,
Y_validation))

plt.plot(rnn_training.history['acc'])
plt.plot(rnn_training.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc="lower right")
plt.show()

```

```

rnn_model = Sequential()

rnn_model.add(Embedding(input_dim = VOCABULARY_SIZE,
                        output_dim = EMBEDDING_SIZE,
                        input_length = MAX_SEQ_LENGTH,
                        trainable = True
                    ))

rnn_model.add(SimpleRNN(64,
                        return_sequences=True
                    ))

rnn_model.add(TimeDistributed(Dense(NUM_CLASSES, activation='softmax'))

rnn_model.compile(loss = 'categorical_crossentropy',
                  optimizer = 'adam',
                  metrics = ['acc'])

rnn_model.summary()

rnn_training = rnn_model.fit(X_train, Y_train, batch_size=128, epochs=10, validation_data=(X_validation,
Y_validation))

plt.plot(rnn_training.history['acc'])
plt.plot(rnn_training.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc="lower right")
plt.show()

rnn_model = Sequential()

# create embedding layer - usually the first layer in text problems
rnn_model.add(Embedding(input_dim = VOCABULARY_SIZE, # vocabulary size - number of unique
words in data
                    output_dim = EMBEDDING_SIZE, # length of vector with which each word is represented
                    input_length = MAX_SEQ_LENGTH, # length of input sequence
                    weights = [embedding_weights], # word embedding matrix
                    trainable = True # True - update the embeddings while training
                ))

# add an RNN layer which contains 64 RNN cells
rnn_model.add(SimpleRNN(64,
                        return_sequences=True # True - return whole sequence; False - return single output of the end of the
sequence

```

```

))

# add time distributed (output at each sequence) layer
rnn_model.add(TimeDistributed(Dense(NUM_CLASSES, activation='softmax')))

rnn_model.compile(loss = 'categorical_crossentropy',
                  optimizer = 'adam',
                  metrics = ['acc'])

rnn_model.summary()

rnn_training = rnn_model.fit(X_train, Y_train, batch_size=128, epochs=10, validation_data=(X_validation,
Y_validation))

plt.plot(rnn_training.history['acc'])
plt.plot(rnn_training.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc="lower right")
plt.show()

"""LSTM"""

lstm_model = Sequential()
lstm_model.add(Embedding(input_dim = VOCABULARY_SIZE, # vocabulary size - number of unique
words in data
                    output_dim = EMBEDDING_SIZE, # length of vector with which each word is represented
                    input_length = MAX_SEQ_LENGTH, # length of input sequence
                    weights = [embedding_weights], # word embedding matrix
                    trainable = True # True - update embeddings_weight matrix
))

lstm_model.add(LSTM(64, return_sequences=True))
lstm_model.add(TimeDistributed(Dense(NUM_CLASSES, activation='softmax')))

lstm_model.compile(loss = 'categorical_crossentropy',
                  optimizer = 'adam',
                  metrics = ['acc'])

lstm_model.summary()

lstm_training = lstm_model.fit(X_train, Y_train, batch_size=128, epochs=10, validation_data=(X_validation,
Y_validation))

plt.plot(lstm_training.history['acc'])
plt.plot(lstm_training.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')

```



```

plt.xlabel('epoch')
plt.legend(['train', 'test'], loc="lower right")
plt.show()

"""GRU"""

gru_model = Sequential()
gru_model.add(Embedding(input_dim = VOCABULARY_SIZE,
                        output_dim = EMBEDDING_SIZE,
                        input_length = MAX_SEQ_LENGTH,
                        weights = [embedding_weights],
                        trainable = True
))
gru_model.add(GRU(64, return_sequences=True))
gru_model.add(TimeDistributed(Dense(NUM_CLASSES, activation='softmax'))

gru_model.compile(loss='categorical_crossentropy',
                  optimizer='adam',
                  metrics=['acc'])

gru_model.summary()

gru_training = gru_model.fit(X_train, Y_train, batch_size=128, epochs=10, validation_data=(X_validation,
Y_validation))

plt.plot(gru_training.history['acc'])
plt.plot(gru_training.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc="lower right")
plt.show()

"""Bidirectional LSTM"""

bidirect_model = Sequential()
bidirect_model.add(Embedding(input_dim = VOCABULARY_SIZE,
                             output_dim = EMBEDDING_SIZE,
                             input_length = MAX_SEQ_LENGTH,
                             weights = [embedding_weights],
                             trainable = True
))
bidirect_model.add(Bidirectional(LSTM(64, return_sequences=True)))
bidirect_model.add(TimeDistributed(Dense(NUM_CLASSES, activation='softmax'))

bidirect_model.compile(loss='categorical_crossentropy',
                      optimizer='adam',
                      metrics=['acc'])

```

```

bidirect_model.summary()

bidirect_training = bidirect_model.fit(X_train, Y_train, batch_size=128, epochs=10,
validation_data=(X_validation, Y_validation))

plt.plot(bidirect_training.history['acc'])
plt.plot(bidirect_training.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc="lower right")
plt.show()

"""Model evaluation"""

loss, accuracy = rnn_model.evaluate(X_test, Y_test, verbose = 1)
print("Loss: {0},\nAccuracy: {1}".format(loss, accuracy))

loss, accuracy = lstm_model.evaluate(X_test, Y_test, verbose = 1)
print("Loss: {0},\nAccuracy: {1}".format(loss, accuracy))

loss, accuracy = gru_model.evaluate(X_test, Y_test, verbose = 1)
print("Loss: {0},\nAccuracy: {1}".format(loss, accuracy))

loss, accuracy = bidirect_model.evaluate(X_test, Y_test, verbose = 1)
print("Loss: {0},\nAccuracy: {1}".format(loss, accuracy))

```