МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №7

по дисциплине «Искусственные нейронные сети»

Тема: Классификация обзоров фильмов

Студент гр.8382 _____ Терехов А.Е.

Преподаватель _____ Жангриров Т.Р.

Санкт-Петербург

2021

Цель работы

Классификация последовательностей - это проблема прогнозирующего моделирования, когда у вас есть некоторая последовательность входных данных в пространстве или времени, и задача состоит в том, чтобы предсказать категорию для последовательности.

Проблема усложняется тем, что последовательности могут различаться по длине, состоять из очень большого словарного запаса входных символов и могут потребовать от модели изучения долгосрочного контекста или зависимостей между символами во входной последовательности.

В данной лабораторной работе также будет использоваться датасет IMDb, однако обучение будет проводиться с помощью рекуррентной нейронной сети.

Задачи

- Ознакомиться с рекуррентными нейронными сетями.
- Изучить способы классификации текста.
- Ознакомиться с ансамблированием сетей.
- Построить ансамбль сетей, который позволит получать точность не менее 97%.

Требования

1. Найти набор оптимальных ИНС для классификации текста.
2. Провести ансамблирование моделей.
3. Написать функцию/функции, которые позволят загружать текст и получать результат ансамбля сетей.
4. Провести тестирование сетей на своих текстах (привести в отчете).

Основные теоретические положения

Мы отобразим каждый обзор фильма в реальную векторную область, популярную технику при работе с текстом, которая называется встраивание слов. Это метод, в котором слова кодируются как действительные векторы в многомерном пространстве, где сходство между словами в смысле смысла приводит к близости в векторном пространстве.

Keras предоставляет удобный способ преобразования положительных целочисленных представлений слов во вложение слов слоем Embedding.

Мы сопоставим каждое слово с вещественным вектором длиной 32. Мы также ограничим общее количество слов, которые нам интересны в моделировании, до 5000 наиболее часто встречающихся слов и обнуляем остальные. Наконец, длина последовательности (количество слов) в каждом обзоре варьируется, поэтому мы ограничим каждый обзор 500 словами, укорачивая длинные обзоры и дополняя короткие обзоры нулевыми значениями.

Теперь, когда мы определили нашу проблему и то, как данные будут подготовлены и смоделированы, мы готовы разработать модель LSTM.

Архитектура сети

Первый слой - это Embedded, который использует 32 вектора длины для представления каждого слова. Следующий уровень - это слой LSTM с 100 единицами памяти (умными нейронами). Наконец, поскольку это проблема классификации, мы используем плотный выходной слой с одним нейроном и сигмоидной функцией активации, чтобы сделать 0 или 1 прогноз для двух классов (хорошего и плохого) в задаче.

Поскольку это проблема двоичной классификации, в качестве функции потерь используется журнал потерь (binary$_c rossentropy$)$.ADAM$.2, .64.

Этот простой LSTM с небольшими настройками достигает почти современных результатов по проблеме IMDB. Важно, что это шаблон, который

вы можете использовать для применения сетей LSTM к вашим собственным задачам классификации последовательностей.

Рассмотрим некоторые расширения этой простой модели.

LSTM и сверточная нейронная сеть для классификации последовательностей.

Сверточные нейронные сети превосходно изучают пространственную структуру во входных данных.

Данные обзора IMDB действительно имеют одномерную пространственную структуру в последовательности слов в обзорах, и CNN может быть в состоянии выбрать инвариантные особенности для хорошего и плохого настроения. Эти изученные пространственные особенности могут затем быть изучены как последовательности уровнем LSTM.

Мы можем легко добавить одномерный слой CNN и максимальный пул после слоя Embedding, которые затем передают объединенные элементы в LSTM. Мы можем использовать небольшой набор из 32 объектов с небольшой длиной фильтра 3. Слой пула может использовать стандартную длину 2, чтобы вдвое уменьшить размер карты объектов.

Мы достигнем результатов, аналогичных первому примеру, но с меньшими весами и меньшим временем обучения.

Рекуррентные нейронные сети, такие как LSTM, обычно имеют проблему переобучения. Необходимо будет решить эту проблему путем добавления в архитектуру сети слоев Dropout.

Ход работы

В работе были использованы следующие зависимости.

```
import re
from typing import Callable, List, Tuple
```

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from keras.datasets import imdb
from keras.layers import Dense, LSTM, Conv1D, MaxPooling1D,
   Dropout
from keras.layers.embeddings import Embedding
from keras.models import Sequential
from keras.preprocessing import sequence
from sklearn.model_selection import train_test_split
```

Для загрузки и подготовки датасета была написана функция.

```
def get_data():
    (X_train, y_train), (X_test, y_test) = imdb.load_data(
       num_words=MAX_WORDS)
    data = np.concatenate((X_train, X_test), axis=0)
    targets = np.concatenate((y_train, y_test), axis=0)
    split = train_test_split(data, targets, test_size=0.1)
    split[0] = sequence.pad_sequences(split[0], maxlen=
       MAX_REVIEW_LEN)
    split[1] = sequence.pad_sequences(split[1], maxlen=
       MAX_REVIEW_LEN)
    return split
```

В данной функции помимо загрузки датасета происходит конкатенация данных, разбиение на тренировочную и тестовую выборки и выравнивение данных по длине.

В работе были рассмотрены две сети: просто LSTM и LSTM + сверточная нейронная сеть. Для создания моделей были написаны соответствующие функции.

```
def build_LSTM_model() -> Sequential:
    model = Sequential()
    model.add(Embedding(MAX_WORDS, EMBEDDING_VECTOR_LEN,
       input_length=MAX_REVIEW_LEN))
    model.add(LSTM(100))
    model.add(Dense(1, activation='sigmoid'))
    return model


def build_LSTM_CNN_model() -> Sequential:
    model = Sequential()
    model.add(Embedding(MAX_WORDS, EMBEDDING_VECTOR_LEN,
       input_length=MAX_REVIEW_LEN))
    model.add(Conv1D(filters=32, kernel_size=3, padding='same',
       activation='relu'))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Dropout(0.5))
    model.add(Conv1D(filters=32, kernel_size=3, padding='same',
       activation='relu'))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Dropout(0.5))
    model.add(LSTM(100))
    model.add(Dense(1, activation='sigmoid'))
    return model
```

Структуры сетей представлены в таблицах 1-2.

Таблица 1 – ИНС №1

Embedding	(None, 512, 32)
LSTM	(None, 100)
Dense	(None, 1)

Таблица 2 – ИНС №2

Embedding	(None, 512, 32)
Conv1D	(None, 512, 32)
MaxPooling1D	(None, 256, 32)
Dropout	(None, 256, 32)
Conv1D	(None, 256, 32)
MaxPooling1D	(None, 128, 32)
Dropout	(None, 128, 32)
LSTM	(None, 100)
Dense	(None, 1)

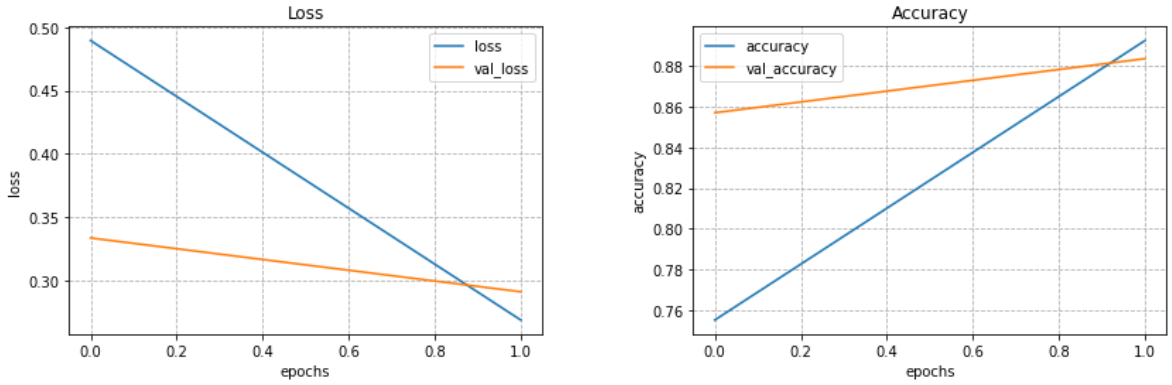
Для обучения и получения дополнительной информации, такой как результаты работы сети на незнакомых данных, был написан декоратор.

```
def fit_model(f: Callable) -> Callable:
    def wrapper() -> (Sequential, float):
        model = f()
        model.compile(loss='binary_crossentropy', optimizer='adam
           ', metrics=['accuracy'])
        print(model.summary())
        results = model.fit(X_train, y_train, validation_split
           =0.2, epochs=2, batch_size=128)
        scores = model.evaluate(X_test, y_test, verbose=0)
        print("Accuracy: %.2f%%" % (scores[1] * 100))
        history = pd.DataFrame(results.history)
        plot(history[['loss', 'val_loss']], 'loss', 'Loss')
        plot(history[['accuracy', 'val_accuracy']], 'accuracy', '
           Accuracy')
        return model, scores[1]
```

```
return wrapper
```

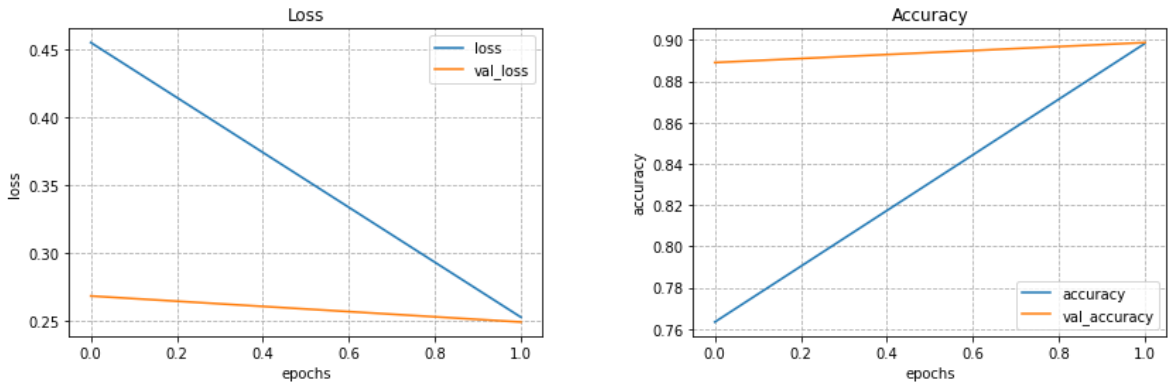
Так как решается задача бинарной классификации используется функция потерь binary_crossentropy. Используемый оптимизатор – Adam. При обучении от тренировочных данных отсекается 20% на валидацию. Обучение происходит в течение 2 эпох. Также вычисляется точность на незнакомых для сети данных.

Результаты обучения сетей представлены на рисунках 1-2.



Потери Точность

Рис. 1 – ИНС №1



Потери Точность

Рис. 2 – ИНС №2

Точность сети №1 на незнакомых данных составила 87.82%. Точность

сети №2 – 89.70%.

Ансамблирование сетей организовано в функции.

```
def ensemble(models: List[Tuple[Sequential, float]], review: np.
   ndarray):
    scores = [m[1] for m in models]
    indices = np.argsort(scores)
    final_pred = 0
    for i in indices[::-1]:
        pred = models[i][0].predict(review)[0][0]
        print(f"Model #{i + 1}: {pred}")
        final_pred += (pred * (i + 1) / sum(range(len(models) +
           1)))
    print(f"Final prediction: {'Positive' if final_pred > .5 else
        'Negative'} ({final_pred})")
```

В функции происходит вычисление взвешенного среднего по следующей формуле:

$$S = \sum_{i=1}^{n} p_i * \frac{i}{\sum_{j=1}^{n} j}$$

Здесь n – количество сетей, p_i – результаты предсказания сетей, сети отсортированы по точности на незнакомых данных.

Для загрузки пользовательского текста написана функция.

```
def load_custom_text(text: str) -> np.ndarray:
    text = text.lower()
    words = re.findall(r"\w+", text)
    index: dict = imdb.get_word_index()
    x = []
    for word in words:
        if word in index and index[word] < MAX_WORDS:
            x.append(index[word] + 3)
    x = sequence.pad_sequences([np.array(x)], maxlen=
       MAX_REVIEW_LEN)
```

```
    return x
```

Текст положительного отзыва: More than Lucas, I admire only the everlasting composer John Williams. I’m just wondering where he gets his ideas for each composition. For a minute he not only composed all the music for these films, but also for Harry Potter, Indiana Jones, Schindler’s List, Jaws, and each of his soundtracks is magical. This film is no exception, as Star Wars is famous for its recognizable music. In addition to the famous screensaver and the Imperial March, they are brilliant: a funeral on Naboo, a march of clones, a battle between a student and a teacher, a great extermination. Star Wars music can be used like the works of Mozart and Chopin, the effect is the same.

Текст негативного отзыва: It is interesting that everyone pays attention to bloopers, bad acting, inconsistencies with subsequent films, but for some reason no one pays attention to ethics. This is awful! Indescribable by any words! How can people even tolerate this? Moreover, they also like it for some reason. Here revenge and betrayal awaits us even worse than in Game of Thrones. The film is called Revenge of the Sith, although it should be called Revenge of the Jedi or Revenge of Obi-Wan - although for what, it is not clear. The most disgusting thing about this film is that the dismemberment and burning of your friends here is shown as moral. Yes, the most notorious villains in other films do not behave the way ’heroes’ behave here.

Ансамбль сетей успешно классифицировал данные отзывы.

```
Model #2: 0.014780102297663689
Model #1: 0.016986679285764694
Final prediction: Negative 0.015515627960364025
Model #2: 0.9283760190010071
Model #1: 0.9599294066429138
Final prediction: Positive 0.9388938148816426
```

Вывод

В данной работе было продолжено изучение анализа настроений. В этот раз задача решалась с помощью рекурентных сетей. Был создан ансамбль из нескольких моделей, корректно отработавший с пользовательскими текстами, для более объективной оценки настроения отзыва к фильму.

ПРИЛОЖЕНИЕ А

```
import re
from typing import Callable, List, Tuple

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from keras.datasets import imdb
from keras.layers import Dense, LSTM, Conv1D, MaxPooling1D,
   Dropout
from keras.layers.embeddings import Embedding
from keras.models import Sequential
from keras.preprocessing import sequence
from sklearn.model_selection import train_test_split

MAX_REVIEW_LEN = 512
EMBEDDING_VECTOR_LEN = 32
MAX_WORDS = 5000
COLAB_PATH_PREFIX = "/content/drive/MyDrive/"


def get_data():
    (X_train, y_train), (X_test, y_test) = imdb.load_data(
       num_words=MAX_WORDS)
    data = np.concatenate((X_train, X_test), axis=0)
    targets = np.concatenate((y_train, y_test), axis=0)
    split = train_test_split(data, targets, test_size=0.1)
    split[0] = sequence.pad_sequences(split[0], maxlen=
       MAX_REVIEW_LEN)
    split[1] = sequence.pad_sequences(split[1], maxlen=
       MAX_REVIEW_LEN)
```

```
    return split


def fit_model(f: Callable) -> Callable:
    def wrapper() -> (Sequential, float):
        model = f()
        model.compile(loss='binary_crossentropy', optimizer='adam
           ', metrics=['accuracy'])
        print(model.summary())
        results = model.fit(X_train, y_train, validation_split
           =0.2, epochs=2, batch_size=128)
        scores = model.evaluate(X_test, y_test, verbose=0)
        print("Accuracy: %.2f%%" % (scores[1] * 100))
        history = pd.DataFrame(results.history)
        plot(history[['loss', 'val_loss']], 'loss', 'Loss')
        plot(history[['accuracy', 'val_accuracy']], 'accuracy', '
           Accuracy')
        return model, scores[1]

    return wrapper


def build_LSTM_model() -> Sequential:
    model = Sequential()
    model.add(Embedding(MAX_WORDS, EMBEDDING_VECTOR_LEN,
       input_length=MAX_REVIEW_LEN))
    model.add(LSTM(100))

    model.add(Dense(1, activation='sigmoid'))
    return model
```

```
def build_LSTM_CNN_model() -> Sequential:
    model = Sequential()
    model.add(Embedding(MAX_WORDS, EMBEDDING_VECTOR_LEN,
       input_length=MAX_REVIEW_LEN))
    model.add(Conv1D(filters=32, kernel_size=3, padding='same',
       activation='relu'))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Dropout(0.5))
    model.add(Conv1D(filters=32, kernel_size=3, padding='same',
       activation='relu'))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Dropout(0.5))
    model.add(LSTM(100))
    model.add(Dense(1, activation='sigmoid'))
    return model


def plot(data: pd.DataFrame, label: str, title: str):
    axis = sns.lineplot(data=data, dashes=False)
    axis.set(ylabel=label, xlabel='epochs', title=title)
    axis.grid(True, linestyle="--")
    plt.show()


def load_custom_text(text: str) -> np.ndarray:
    text = text.lower()
    words = re.findall(r"\w+", text)
    index: dict = imdb.get_word_index()
    x = []
    for word in words:
        if word in index and index[word] < MAX_WORDS:
            x.append(index[word] + 3)
```

```
    x = sequence.pad_sequences([np.array(x)], maxlen=
       MAX_REVIEW_LEN)
    return x


def ensemble(models: List[Tuple[Sequential, float]], review: np.
   ndarray):
    scores = [m[1] for m in models]
    indices = np.argsort(scores)
    final_pred = 0
    for i in indices[::-1]:
        pred = models[i][0].predict(review)[0][0]
        print(f"Model #{i + 1}: {pred}")
        final_pred += (pred * (i + 1) / sum(range(len(models) +
           1)))
    print(f"Final prediction: {'Positive' if final_pred > .5 else
        'Negative'} ({final_pred})")


if __name__ == '__main__':
    X_train, X_test, y_train, y_test = get_data()
    models = [
        fit_model(build_LSTM_model)(),
        fit_model(build_LSTM_CNN_model)(),
    ]
    with open(COLAB_PATH_PREFIX + "neg.txt", "r") as f_neg:
        ensemble(models, load_custom_text(f_neg.read()))

    with open(COLAB_PATH_PREFIX + "pos.txt", "r") as f_pos:
        ensemble(models, load_custom_text(f_pos.read()))
```