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

ОТЧЕТ

по лабораторной работе №7
по дисциплине «Искусственные нейронные сети»
Тема: «Классификация обзоров фильмов»

Студент гр. 8383 _____ Киреев К.А.

Преподаватель _____ Жангиров 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

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

Ход работы

Модели для классификации текста

Были выбраны и реализованы три различные архитектуры нейронной сети для классификации текста. Архитектуры моделей представлены ниже.

Модель 1:

```
model_a = Sequential([
    Embedding(top_words, embedding_vecor_length, input_length=max_review_length),
    Conv1D(64, 3, padding='same', activation='relu'),
    MaxPool1D(2),
    Dropout(0.3),
    Conv1D(128, 3, padding='same', activation='relu'),
    MaxPool1D(2),
    Dropout(0.2),
    LSTM(64, return_sequences=True, dropout=0.2),
    LSTM(32),
    Dense(1, activation='sigmoid')])
```

Модель 2:

```
model_b = Sequential([
    Embedding(top_words, embedding_vecor_length, input_length=max_review_length),
    LSTM(100, return_sequences=True, dropout=0.3),
    LSTM(50, dropout=0.3),
    Dense(1, activation='sigmoid')])
```

Модель 3:

```
model_c = Sequential([
    Embedding(top_words, embedding_vecor_length, input_length=max_review_length),
    Conv1D(32, 3, padding='same', activation='relu'),
    MaxPool1D(2),
    Dropout(0.5),
    Conv1D(64, 3, padding='same', activation='relu'),
    MaxPool1D(2),
    Dropout(0.4),
    Flatten(),
    Dense(128),
    Dropout(0.5),
    Dense(1, activation='sigmoid')])
```

Ансамблирование моделей

Было реализовано обучение моделей и их оценка на тестовых данных. В цикле выводится точность для каждой отдельной модели.

```
members = [model_a, model_b, model_c]
epochs = [2, 2, 3]
for i, mod in enumerate(members):
    x_train = X_train[i * train_size : (i + 1) * train_size]
    y_train = Y_train[i * train_size : (i + 1) * train_size]
    x_test = X_test[i * test_size : (i + 1) * test_size]
    y_test = Y_test[i * test_size : (i + 1) * test_size]
    mod.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    mod.fit(x_train, y_train, validation_split=0.1, epochs=epochs[i], batch_size=64)
    scores = mod.evaluate(x_test, y_test, verbose=2)
    print("Accuracy: %.2f%%" % (scores[1]*100))
```

Была написана функция для получения результата предсказания ансамбля моделей. Далее производится оценка точности ансамбля моделей.

```
def stacked_prediction(members, x_test, load):
    yhat = []
    for i, m in enumerate(members):
        if load:
            print(m.predict(x_test, verbose=0))
        yhat.append(np.round(m.predict(x_test, verbose=0)))
    yhat = np.asarray(yhat)
    yhat = [np.round(np.mean(x)) for x in zip(*yhat)]
    return np.asarray(yhat).astype('int')

yhat = stacked_prediction(members, X_test, False)
acc = accuracy_score(Y_test, yhat)
print("Accuracy: %.2f%%" % (acc*100))
```

Точность отдельных моделей и ансамбля представлена ниже.

```
Epoch 1/2
Epoch 2/2
79/79 - 2s - loss: 0.3043 - accuracy: 0.8708
Accuracy: 87.08%

Epoch 1/2
Epoch 2/2
79/79 - 5s - loss: 0.3302 - accuracy: 0.8592
Accuracy: 85.92%

Epoch 1/3
Epoch 2/3
Epoch 3/3
79/79 - 0s - loss: 0.3651 - accuracy: 0.8548
Accuracy: 85.48%


Accuracy: 88.66%
```

Точность ансамбля моделей составила 88.66%, что превосходит показатели отдельных моделей.

Функция ввода пользовательского текста

Была написана функция для ввода пользовательского текста `load_text()`

```
def load_text():
    dictionary = imdb.get_word_index()
    load_x = []
    words = input()
    words = re.sub(r"[^a-zA-Z0-9']", " ", words)
    words = words.split(' ')
    valid = []
    for word in words:
        word = dictionary.get(word)
        if word in range(1, 10000):
            valid.append(word+3)
    load_x.append(valid)
    result = sequence.pad_sequences(load_x, maxlen=max_review_length)
    print(stacked_prediction(members, result, True))

load_text()
```

Получаем словарь со словами и их индексами. Далее обрабатываем введенный текст, удаляя лишние символы. Заменяем числа на их индексы, оставляя только 10000 самых частых слов. Далее расширяем текст до длины 500 и выводим результат предсказания ансамбля нейронных сетей. Пример работы представлен ниже.

```
It's very silly but very self-aware about it,
which makes it very enjoyable for what it is.
[[0.7633463]]
[[0.8056914]]
[[0.9029985]]
[1]


With so many characters, the movie spends too much time
on discovery and not enough on showing those powers in action.
[[0.24501102]]
[[0.3473688]]
[[0.5906282]]
[0]
```

Выводы

В ходе лабораторной работы был реализован прогноз успеха фильмов по обзорам. Также был применен метод ансамблирования моделей для более точного семантического анализа текста.

Приложение А. Исходный код программы

```
import re
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
os.environ['TF_FORCE_GPU_ALLOW_GROWTH'] = 'true'
from matplotlib import gridspec
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
from keras.utils import to_categorical
from keras.layers import Dense, Dropout, LSTM, Conv1D, MaxPool1D, GRU, Flatten
from keras.models import Sequential
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
from sklearn.metrics import accuracy_score


from keras.datasets import imdb
(training_data,    training_targets),    (testing_data,    testing_targets)    =
imdb.load_data(num_words=10000)
data = np.concatenate((training_data, testing_data), axis=0)
targets = np.concatenate((training_targets, testing_targets), axis=0)


X_test = data[:10000]
Y_test = targets[:10000]
X_train = data[10000:]
Y_train = targets[10000:]


max_review_length = 500 # макс. длина текста
embedding_vecor_length = 32 # 32-мерное векторное представление
top_words = 10000 # количество слов, рассматриваемых как признаки


# print(X_train[0])
X_train = sequence.pad_sequences(X_train, maxlen=max_review_length)
# print(X_train[0])
X_test = sequence.pad_sequences(X_test, maxlen=max_review_length)
```

```
      model_a = Sequential([
          Embedding(top_words, embedding_vecor_length, input_length=max_review_length),
          Conv1D(64, 3, padding='same', activation='relu'),
          MaxPool1D(2),
          Dropout(0.3),
          Conv1D(128, 3, padding='same', activation='relu'),
          MaxPool1D(2),
          Dropout(0.2),
          LSTM(64, return_sequences=True, dropout=0.2),
          LSTM(32),
          Dense(1, activation='sigmoid')])

      model_b = Sequential([
          Embedding(top_words, embedding_vecor_length, input_length=max_review_length),
          LSTM(100, return_sequences=True, dropout=0.3),
          LSTM(50, dropout=0.3),
          Dense(1, activation='sigmoid')])

      model_c = Sequential([
          Embedding(top_words, embedding_vecor_length, input_length=max_review_length),
          Conv1D(32, 3, padding='same', activation='relu'),
          MaxPool1D(2),
          Dropout(0.5),
          Conv1D(64, 3, padding='same', activation='relu'),
          MaxPool1D(2),
          Dropout(0.4),
          Flatten(),
          Dense(128),
          Dropout(0.5),
          Dense(1, activation='sigmoid')])

      ans_len = 4
      train_size, test_size = len(X_train) // ans_len, len(X_test) // ans_len

      members = [model_a, model_b, model_c]
```

```
        epochs = [2, 2, 3]
        for i, mod in enumerate(members):
            x_train = X_train[i * train_size : (i + 1) * train_size]
            y_train = Y_train[i * train_size : (i + 1) * train_size]
            x_test = X_test[i * test_size : (i + 1) * test_size]
            y_test = Y_test[i * test_size : (i + 1) * test_size]
            mod.compile(loss='binary_crossentropy',                    optimizer='adam',
metrics=['accuracy'])
            mod.fit(x_train,    y_train,    validation_split=0.1,    epochs=epochs[i],
batch_size=64)
            scores = mod.evaluate(x_test, y_test, verbose=2)
            print("Accuracy: %.2f%%" % (scores[1]*100))


        def stacked_prediction(members, x_test, load):
            yhat = []
            for i, m in enumerate(members):
                if load:
                    print(m.predict(x_test, verbose=0))
                yhat.append(np.round(m.predict(x_test, verbose=0)))
            yhat = np.asarray(yhat)
            yhat = [np.round(np.mean(x)) for x in zip(*yhat)]
            return np.asarray(yhat).astype('int')


        yhat = stacked_prediction(members, X_test, False)
        acc = accuracy_score(Y_test, yhat)
        print("Accuracy: %.2f%%" % (acc*100))


        def load_text():
            dictionary = imdb.get_word_index()
            load_x = []
            words = input()
            words = re.sub(r"[^a-zA-Z0-9']", " ", words)
            words = words.split(' ')
            valid = []
            for word in words:
                word = dictionary.get(word)
                if word in range(1, 10000):
                    valid.append(word+3)
```

```
    load_x.append(valid)
    result = sequence.pad_sequences(load_x, maxlen=max_review_length)
    print(stacked_prediction(members, result, True))

load_text()
```