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. 7383 _____ Ласковенко Е.А.

Преподаватель _____ Жукова Н. А.

Санкт-Петербург

2020

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)

Ход работы.

1. Были созданы следующие модели нейронных сетей: рекуррентная нейронная сеть (RNN), сверточная нейронная сеть (CNN) и сверточная рекуррентная нейронная сеть (CRNN). Код программы представлен в приложении А.

2. Обучим три модели при одинаковых параметрах обучения и проведем их ансамблирование. Оценим точность нейронных сетей и их ансамблей. Графики точностей для моделей представлены на рис. 1.

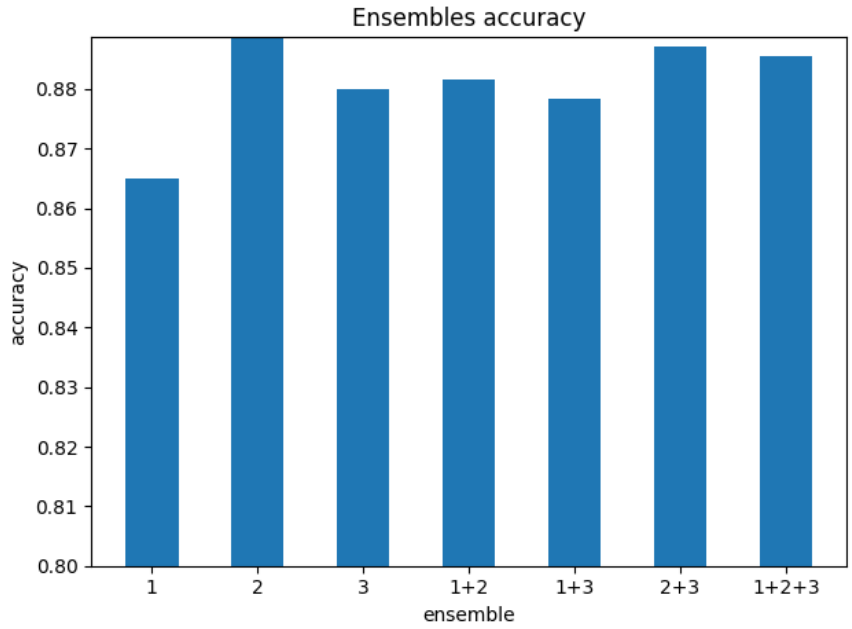

Рисунок 1 – График точностей моделей и их ансамблей

Значения точностей из графика: 86.5% RNN, 88.9% CNN, 88.0% CRNN, 88.2% RNN + CNN, 87.8% RNN + CRNN, 88.7% CNN + CRNN, 88.6% RNN + CNN + CRNN. Все модели и их ансамбли дают примерно одинаковую точность, однако, наибольшую точность более 0.88 дает CNN модель и ансамбль CNN и CRNN моделей.

3. Была написана функция, которая позволяет загружать пользовательский текст из файла. Получим результат ансамбля сетей, загрузив следующий отзыв: The acting was amazing and the film was good overall but I think 'masterpiece' and 'film of the year' are a bit overused

throughout the reviews. In no way did I dislike this film, I thought it was really good, just overrated. Ответ ансамбля сетей: 0.7085517, что говорит о том, что данный отзыв относится к положительному.

Вывод.

В результате выполнения данной работы были созданы несколько моделей нейронных сетей и проведено их ансамблирование. Также были оценены точности каждой из моделей и ансамблей и получен результат оценки пользовательского отзыва ансамблем.

Приложения

Приложение А

```
import numpy as np
from keras.datasets import imdb
from keras.layers import Dense, MaxPooling1D, Conv1D, Dropout,
Flatten
from keras.layers import LSTM
from keras.layers.embeddings import Embedding
from keras.models import Sequential
from keras.preprocessing import sequence
import matplotlib.pyplot as plt

# Setting constants:
top_words = 10000
embedding_vector_length = 32
max_review_length = 500

# Loading data:
(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)

# Fitting data:
training_data = sequence.pad_sequences(training_data,
maxlen=max_review_length)
testing_data = sequence.pad_sequences(testing_data,
maxlen=max_review_length)


# Defying models:
def build_RNN_model():
    model = Sequential()
    model.add(Embedding(top_words, embedding_vector_length,
input_length=max_review_length))
    model.add(LSTM(100))
    model.add(Dropout(0.5))
    model.add(Dense(50, activation='relu'))
    model.add(Dropout(0.25))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy',
optimizer='adam', metrics=['accuracy'])
    return model


def build_CNN_model():
    model = Sequential()
    model.add(Embedding(top_words, embedding_vector_length,
input_length=max_review_length))
```

```
    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=64, kernel_size=3,
padding='same', activation='relu'))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Dropout(0.25))
    model.add(Flatten())
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy',
optimizer='adam', metrics=['accuracy'])
    return model


def build_CRNN_model():
    model = Sequential()
    model.add(Embedding(top_words, embedding_vector_length,
input_length=max_review_length))
    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(Dropout(0.25))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy',
optimizer='adam', metrics=['accuracy'])
    return model


def ensemble_models(models):
    results = []
    accuracies = []
    labels = []
    for model in models:
        results.append(model.predict(testing_data))
        result = np.array(results[-1])
        result = np.reshape(result, result.shape[0])
        result = np.greater_equal(result, np.array([0.5]),
dtype=np.float64)
        accuracy = 1 - np.abs(testing_targets -
result).mean(axis=0)
        accuracies.append(accuracy)
        labels.append(str(len(results)))
    pairs = [(0, 1), (0, 2), (1, 2)]
    for (i, j) in pairs:
        result = np.array([results[i],
results[j]]).mean(axis=0)
        result = np.reshape(result, result.shape[0])
```

```
        result = np.greater_equal(result, np.array([0.5]),
dtype=np.float64)
        accuracy = 1 - np.abs(testing_targets -
result).mean(axis=0)
        accuracies.append(accuracy)
        labels.append(str(i + 1) + '+' + str(j + 1))
    result = np.array(results).mean(axis=0)
    result = np.reshape(result, result.shape[0])
    result = np.greater_equal(result, np.array([0.5]),
dtype=np.float64)

    accuracy = 1 - np.abs(testing_targets -
result).mean(axis=0)
    accuracies.append(accuracy)
    labels.append('1+2+3')
    plot_results(accuracies, labels)
    print(accuracies)


def plot_results(accuracies, labels):
    plt.bar(np.arange(len(accuracies)) * 2, accuracies,
width=1)
    plt.xticks([2 * i for i in range(0, len(accuracies))],
labels=labels)
    plt.ylim([0.8, max(accuracies)])
    plt.title('Ensembles accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('ensemble')
    plt.show()


def load_text(filename):
    # File reading:
    text = []
    file = open(filename, 'rt')
    for line in file.readlines():
        text += [s.strip(''.join(['.', ',', ':', ';', '!',
'?', '(', ')'])).lower() for s in line.strip().split()]
    file.close()
    # Encode words:
    indexes = imdb.get_word_index()
    encoded = []
    for w in text:
        if w in indexes and indexes[w] < 10000:
            encoded.append(indexes[w])
    return np.array(encoded)


# Building RNN model:
RNN_model = build_RNN_model()
```

```
RNN_model.fit(training_data, training_targets,
validation_data=(testing_data, testing_targets), epochs=2,
batch_size=100)
RNN_model.save("rnn_model.h5")

# Building CNN model:
CNN_model = build_CNN_model()
CNN_model.fit(training_data, training_targets,
validation_data=(testing_data, testing_targets), epochs=2,
batch_size=100)
CNN_model.save("cnn_model.h5")

# Building CRNN model:
CRNN_model = build_CRNN_model()
CRNN_model.fit(training_data, training_targets,
validation_data=(testing_data, testing_targets), epochs=2,
batch_size=100)
CRNN_model.save("crnn_model.h5")

# Ensemble models:
ensemble_models([RNN_model, CNN_model, CRNN_model])

# Loading custom text:
text = load_text('review.txt')
text = sequence.pad_sequences([text],
maxlen=max_review_length)

# Predicting custom text:
results = []
results.append(RNN_model.predict(text))
results.append(CNN_model.predict(text))
results.append(CRNN_model.predict(text))
result = np.array(results).mean(axis=0)
result = np.reshape(result, result.shape[0])
print(result)
```