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. 7382 _____ Гаврилов А.В.

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.

Требования к выполнению задания.

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.

Была созданы и обучены 3 модели искусственной нейронной сети. Код предоставлен в приложении А. Реализованы 3 различных модели из которых комбинировались ансамбли. Результат предсказаний ансамбля – среднее предсказаний 2х моделей, его составляющих. Исходя из предсказаний, мы получили точность каждого ансамбля, результат на рис. 1.

```
[[[0, 0], 0.7051], [[0, 1], 0.8817], [[0, 2], 0.8446], [[1, 1], 0.8835], [[1, 2], 0.8706], [[2, 2], 0.8439]]
```

Рисунок 1 – Точность ансамблей.

На рис. 1 список из 2х чисел для каждого элемента основного списка – индексы моделей, если эти числа одинаковые, то ансамбль – сама модель.

Таким образом видим, что наивысшую точность показывает модель 1 без ансамблирования, точность достигает 88,35%.

Точность достаточно низкая, так как для тестирования ансамблей выбраны не лучшие архитектуры моделей и за счет того, что все обзоры обрезаются до 500 слов. Также было выбрано только 5000 наиболее встречаемых слов, что тоже может снизить точность.

Затем была реализована функция, которая кодирует список поданных обзоров и делает предсказание ансамбля. В нашем случае работа этой функции была протестирована на ансамбле из 1 и 2 модели.

Поданные строки:

1) "I love it, it is the best film i have ever seen",
2) "Worst thing i have ever seen",
3) "Bad producer but good actors, so the film is not so bad"

Результат:

[1, 0, 0]

Результат почти верный, так как последнюю строку сложно классифицировать, модель ее классифицирует, как отрицательный отзыв.

Выводы.

В ходе работы были построены и изучены методы построения ансамблей моделей нейронных сетей. Разработана функция для предсказания результатов ансамблем для собственных текстовых данных.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
import matplotlib.pyplot as plt
import numpy as np
from tensorflow.keras import layers
from tensorflow.keras.datasets import imdb
from tensorflow.keras import Sequential
from tensorflow.keras.preprocessing import sequence

class lab6:
    def __init__(self):
        self.batch_size = 70
        self.epochs = 2
        self.max_review_length = 500
        self.embedding_vecor_length = 32
        self.top_words = 5000

    def get_data(self, length=10000):
        (X_train, y_train), (X_test, y_test) =
imdb.load_data(num_words=length)
        X_test = X_test[:10000]
        y_test = y_test[:10000]
        X_train = sequence.pad_sequences(X_train,
maxlen=self.max_review_length)
        X_test = sequence.pad_sequences(X_test,
maxlen=self.max_review_length)
        return X_train, y_train, X_test, y_test

    def build_models(self):
        models = []
        model = Sequential()
        model.add(layers.Embedding(self.top_words,
self.embedding_vecor_length, input_length=self.max_review_length))
        model.add(layers.LSTM(100))
        model.add(layers.Dense(32, activation="relu"))
        model.add(layers.Dropout(0.3, noise_shape=None, seed=None))
        model.add(layers.Dense(70, activation="relu"))
        model.add(layers.Dropout(0.4, noise_shape=None, seed=None))
        model.add(layers.Dense(70, activation="relu"))
        model.add(layers.Dense(1, activation="sigmoid"))
        model.compile(optimizer="adam", loss="binary_crossentropy",
metrics=["accuracy"])
        models.append(model)

        model = Sequential()
        model.add(layers.Embedding(self.top_words,
self.embedding_vecor_length, input_length=self.max_review_length))
        model.add(layers.Conv1D(filters=32, kernel_size=3,
padding='same', activation='relu'))
```

```
        model.add(layers.MaxPooling1D(pool_size=2))
        model.add(layers.Dropout(0.3, noise_shape=None, seed=None))
        model.add(layers.LSTM(100))
        model.add(layers.Dropout(0.25, noise_shape=None, seed=None))
        model.add(layers.Dense(1, activation='sigmoid'))
        model.compile(optimizer="adam", loss="binary_crossentropy",
metrics=["accuracy"])
        models.append(model)

        model = Sequential()
        model.add(layers.Embedding(self.top_words,
self.embedding_vecor_length, input_length=self.max_review_length))
        model.add(layers.LSTM(100))
        model.add(layers.Dense(1, activation='sigmoid'))
        model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
        models.append(model)

        return models

    def fit(self, models):
        x_train, y_train, x_test, y_test = self.get_data()
        for model in models:
            history = model.fit(x_train, y_train, epochs=self.epochs,
batch_size=self.batch_size,
                                validation_data=(x_test, y_test))
            self.plot_acc(history.history['acc'],
history.history['val_acc'])
        return models

    def plot_acc(self, acc, val_acc):
        plt.plot(acc, 'b', label='train')
        plt.plot(val_acc, 'r', label='validation')
        plt.title('accuracy')
        plt.ylabel('accuracy')
        plt.xlabel('epochs')
        plt.legend()
        plt.show()
        plt.clf()

    def start(self):
        models = self.build_models()
        models = self.fit(models)
        self.test_ensemble(models)
        self.predict_reviews(models[1], models[2], ["I love it, it is
the best film i have ever seen",
                                                    "Worst thing i
have ever seen",
                                                    "Bad producer but
good actors, so the film is not so bad"])
```

```
    def test_ensemble(self, models):
        x_train, y_train, x_test, y_test = self.get_data()
        x_test = x_test[:10000]
        y_test = y_test[:10000]
        results = []
        print("Starting testing ensembles")
        for i in range(len(models)):
            for j in range(i, len(models)):
                pred = self.predict_ensemble(models[i], models[j],
x_test)
                res = [1 if pred[i] == y_test[i] else 0 for i in
range(len(pred))]
                res = res.count(1)/len(res)
                results.append([[i, j], res])
        print(results)

    def predict_ensemble(self, model1, model2, data):
        pred1 = model1.predict(data)
        pred2 = model2.predict(data)
        pred = [1 if (pred1[i] + pred2[i])/2 > 0.5 else 0 for i in
range(len(pred1))]
        return np.array(pred)

    def predict_reviews(self, model1, model2, rev):
        encoded_rev = self.encode_review(rev)
        print(self.predict_ensemble(model1, model2, encoded_rev))

    def encode_review(self, rev):
        res = []
        for i, el in enumerate(rev):
            el = el.lower()
            delete_el = [',', '!', '.', '?']
            for d_el in delete_el:
                el = el.replace(d_el, '')
            el = el.split()
            for j, word in enumerate(el):
                code = imdb.get_word_index().get(word)
                if code is None:
                    code = 0
                el[j] = code
            res.append(el)
        for i, r in enumerate(res):
            res[i] = sequence.pad_sequences([r],
maxlen=self.max_review_length)
        res = np.array(res)
        return res.reshape((res.shape[0], res.shape[2]))


lab = lab6()
lab.start()
```