Отчёт о выполнении работы по классификации отзывов к фильмам.

Было предложено разработать и обучить модель нейронной сети для классификации отзывов к фильмам. Датасет с открытого источника https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz .

ДАТАСЕТ.

Основной набор данных содержит 50 000 отзывов, разделённых равномерно на 25 000 обучающих и 25 000 тестовых наборов. Распределение меток сбалансировано по 12 500 положительных и отрицательных отзывов в каждом наборе **датасета**.

В задании было предложено оценить отзыв к фильму с присвоением рейтинга (от 1 до 10). В связи с этим необходимо преобразовать **датасет**, разбить датасет с двумя классами на датасет с восемью классами (5 и 6 не учитываются).

```
# функция разбивает исходный датасет по классам (по рейтингу)

def func1(mas):
    for i in range(len(mas)):
        str_n = mas[i]
        str_1 = str_name(str_n)

        name_faile, num_class = inv1(str_1)
        name_faile1 = name_faile + '.txt'

        if num_class == '0':
            path_new = c10
        elif num_class == '1':
            path_new = c1
        elif num_class == '2':
            path_new = c2
        elif num_class == '3':
            path_new = c3
        elif num_class == '4':
            path_new = c4
        elif num_class == '5':
            path_new = c5
        elif num_class == '6':
            path_new = c
        elif num_class == '7':
            path_new = c7
        elif num_class == '8':
            path_new = c8
        elif num_class == '9':
            path_new = c9


        shutil.copy2(str_n, os.path.join(path_new, name_faile1))
```

Рисунок 1. Функция разбиения исходного датасета по рейтингам.

Для преобразования **датасета** были разработаны дополнительные функции (Рис. 1).

Функция считывает названия файлов с выбранных папок, из названия файла получает класс (рейтинг) отзыва, и сохраняет файл в папку с названием соответствующим данному рейтингу.

Получили **датасет** разбитый на восемь классов, но классы разбиты не поровну. В дальнейшем все классы были уравновешены, из каждого класса удалены лишние файлы, получилось по 4 000 файлов в каждом классе. Оставшиеся файлы используются в тестовом датасете, для проверки итоговой точности. На данном датасете предложенная мной модель не обучилась, на валидационном наборе данных показывала переобучение уже достигнув точности только 0,27. Для улучшения точности испробовал различные методы:

Менял размер **batch**.

Применял различный **Dropout**.

Кроссвалидацию.

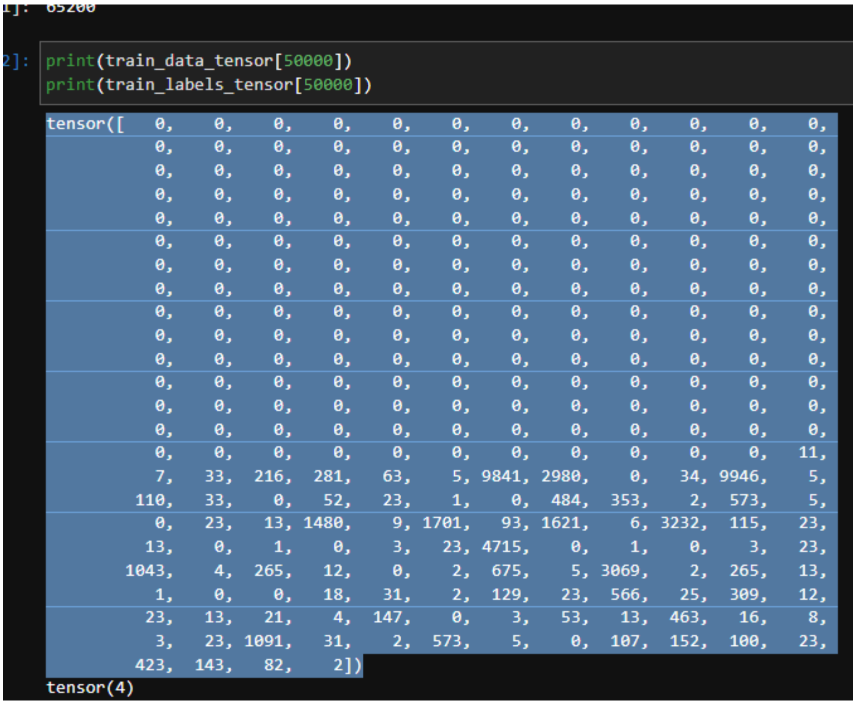
Регуляризацию.

Так же менял количество слоёв, но точность на валидационном наборе не выросла.

Из чего сделал вывод что, скорее всего не достаточно данных датасета. Решил расширить данный датасет, применив аугментацию. Расширял датасет путём добавления случайным образом символа заполнителя, в данном случае единицей, символ заполнитель нуль использовался для заполнения пустого места (если отзыв был короче необходимого размера).

Так как отзывы все разного размера, слишком длинные отзывы обрезались с конца, а слишком короткие дополнялись нулями в начале отзыва, таким образом, весь отзыв находится в конце после нулей. Так же неизвестные слова, которые не попали в словарь, заменялись нулём. Для упрощения модели (для уменьшения входного слоя) исходный словарь

кодировки слов, был сокращён до 39998 слов, обрезав конец словаря (так как самые популярные слова находятся в начале словаря), плюс два символа заполнителя, в итоге получил размерность входного слоя 40000 (Рис. 2).

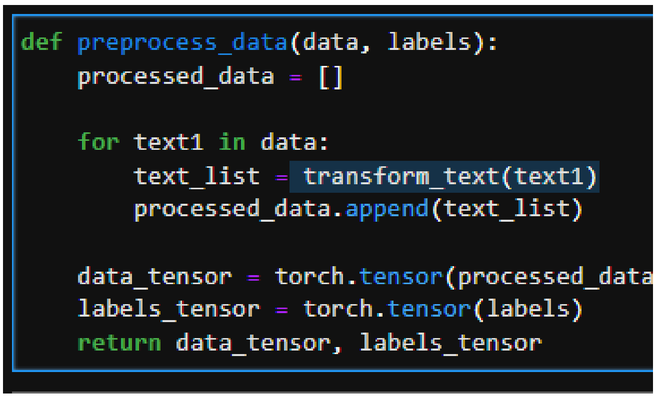


```
1]: 65200

2]: print(train_data_tensor[50000])
    print(train_labels_tensor[50000])

tensor([   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,   11,
           7,   33,  216,  281,   63,    5, 9841, 2980,    0,   34, 9946,    5,
         110,   33,    0,   52,   23,    1,    0,  484,  353,    2,  573,    5,
           0,   23,   13, 1480,    9, 1701,   93, 1621,    6, 3232,  115,   23,
          13,    0,    1,    0,    3,   23, 4715,    0,    1,    0,    3,   23,
        1043,    4,  265,   12,    0,    2,  675,    5, 3069,    2,  265,   13,
           1,    0,    0,   18,   31,    2,  129,   23,  566,   25,  309,   12,
          23,   13,   21,    4,  147,    0,    3,   53,   13,  463,   16,    8,
           3,   23, 1091,   31,    2,  573,    5,    0,  107,  152,  100,   23,
         423,  143,   82,    2])
tensor(4)
```

Рисунок 2. Тензоры, отзыв и рейтинг.

После предобработки тексты отзывов и метки (рейтинги), преобразованы в тензор и сохранены в файл, для удобства дальнейшего использования. В итоге получил файл содержащий тензор с 65200 отзывов, а так же файл содержащий тензор меток.

Для токенизации текста использовался словарь **imdb.vocab**.



```
def preprocess_data(data, labels):
    processed_data = []

    for text1 in data:
        text_list = transform_text(text1)
        processed_data.append(text_list)

    data_tensor = torch.tensor(processed_data
    labels_tensor = torch.tensor(labels)
    return data_tensor, labels_tensor
```

Рисунок 3. Функция преобразования в тензоры.

```python
# функция преобразования текста в числа

def transform_text(text1, vocab=vocab, len_text=len_text):
    mass = []
    str1 =''
    for ch in text1:
        if len(mass)>len_text-1: # если слов больше чем нужно, выходим
            return mass
        if ch !=' ':
            str1 = str1 + ch
        if ch ==' ':
            if str1 !='':

                fl = False

                for i in range(len(vocab)):
                    if str1.lower() == vocab[i].lower():
                        fl = True
                        mass.append(i+1)
                        str1 =''
                        break
                if fl == False: #  если слово не найдено, заменяем нулями
                    mass.append(0)
                    str1 =''
    for i in range(len(vocab)): # Проверяем последнее слово
        if str1.lower() == vocab[i].lower():
            fl = True
            mass.append(i+1)
            str1 =''
            break
    if fl == False: #  если слово не найдено, заменяем нулями
        mass.append(0)
        str1 =''

    if len(mass)<len_text:  #  если слов меньше чем нужно, добавляем нулями.
        while len(mass)<len_text:
            mass.append(0)
            str1 =''

    return mass
```

Рисунок 4. Функция преобразования текста в числа.

МОДЕЛЬ СЕТИ.

Для классификации отзывов использовалась модель нейронной сети

LSTM (архитектура рекуррентной нейронной сети). Модель создана с использованием фреймворка **pytorch** (Рис.3)**.**

```
# Определение класса модели
class SentimentClassifier(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(SentimentClassifier, self).__init__()
        self.hidden_size = hidden_size
        self.embedding = nn.Embedding(input_size, hidden_size)
        self.lstm1 = nn.LSTM(hidden_size, hidden_size, batch_first=True)
        self.lstm2 = nn.LSTM(hidden_size, hidden_size, batch_first=True)
        self.dropout = nn.Dropout(p=0.2)  # Добавление слоя Dropout
        self.fc = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        embedded = self.embedding(x)
        output, _ = self.lstm1(embedded)
        output, _ = self.lstm2(output)
        output = self.dropout(output)  # Применение Dropout к выходу LSTM
        output = self.fc(output[:, -1, :])  # Используется только последн
        return output
```

Рисунок 3. Модель сети.

Модель содержит два **LSTM** слоя. Применялся **Dropout** для борьбы с переобучением. Функция потерь **nn.CrossEntropyLoss**(), оптимизатор **optim.Adam.**

Оптимальное значение размерности скрытого слоя **hidden_size = 256.**

При таких показателях модель показала лучшую точность 0,75.

```
Epoch 9/20 | Train Loss: 0.0270 | Train Accuracy: 0.9917 | Val Loss: 1.7076 |
Epoch 10/20 | Train Loss: 0.0297 | Train Accuracy: 0.9906 | Val Loss: 1.6481
Epoch 11/20 | Train Loss: 0.0371 | Train Accuracy: 0.9882 | Val Loss: 1.5671
Epoch 12/20 | Train Loss: 0.0239 | Train Accuracy: 0.9927 | Val Loss: 1.6081
Epoch 13/20 | Train Loss: 0.0254 | Train Accuracy: 0.9917 | Val Loss: 1.5688
Epoch 14/20 | Train Loss: 0.0267 | Train Accuracy: 0.9914 | Val Loss: 1.5454
Epoch 15/20 | Train Loss: 0.0320 | Train Accuracy: 0.9903 | Val Loss: 1.5839
Epoch 16/20 | Train Loss: 0.0257 | Train Accuracy: 0.9919 | Val Loss: 1.6192
Epoch 17/20 | Train Loss: 0.0248 | Train Accuracy: 0.9919 | Val Loss: 1.6492
Epoch 18/20 | Train Loss: 0.0256 | Train Accuracy: 0.9917 | Val Loss: 1.8426
Epoch 19/20 | Train Loss: 0.0291 | Train Accuracy: 0.9904 | Val Loss: 1.6065
Epoch 20/20 | Train Loss: 0.0232 | Train Accuracy: 0.9924 | Val Loss: 1.6941
train -   точность: 0.9962
test -   точность: 0.7440
train -   точность: 0.9955
test -   точность: 0.7592
```

Рисунок 4. Обучение сети.

На пятом рисунке видна хорошая обучаемость модели на 20 эпохах.

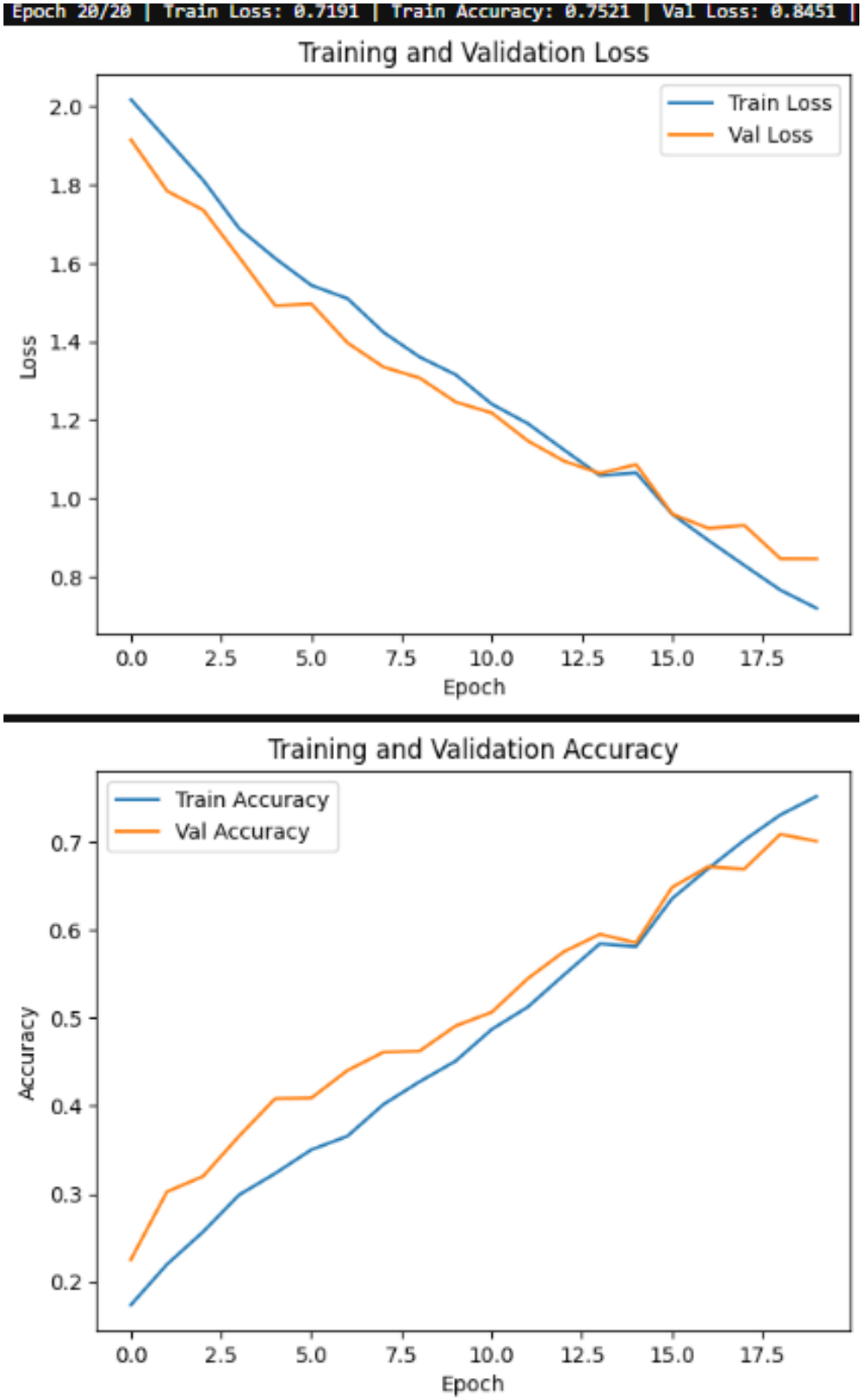


Рисунок 5. Начало обучения.

На рис. 6 видно что сеть переобучилась, но точность предсказаний уже высока около 90 процентов.

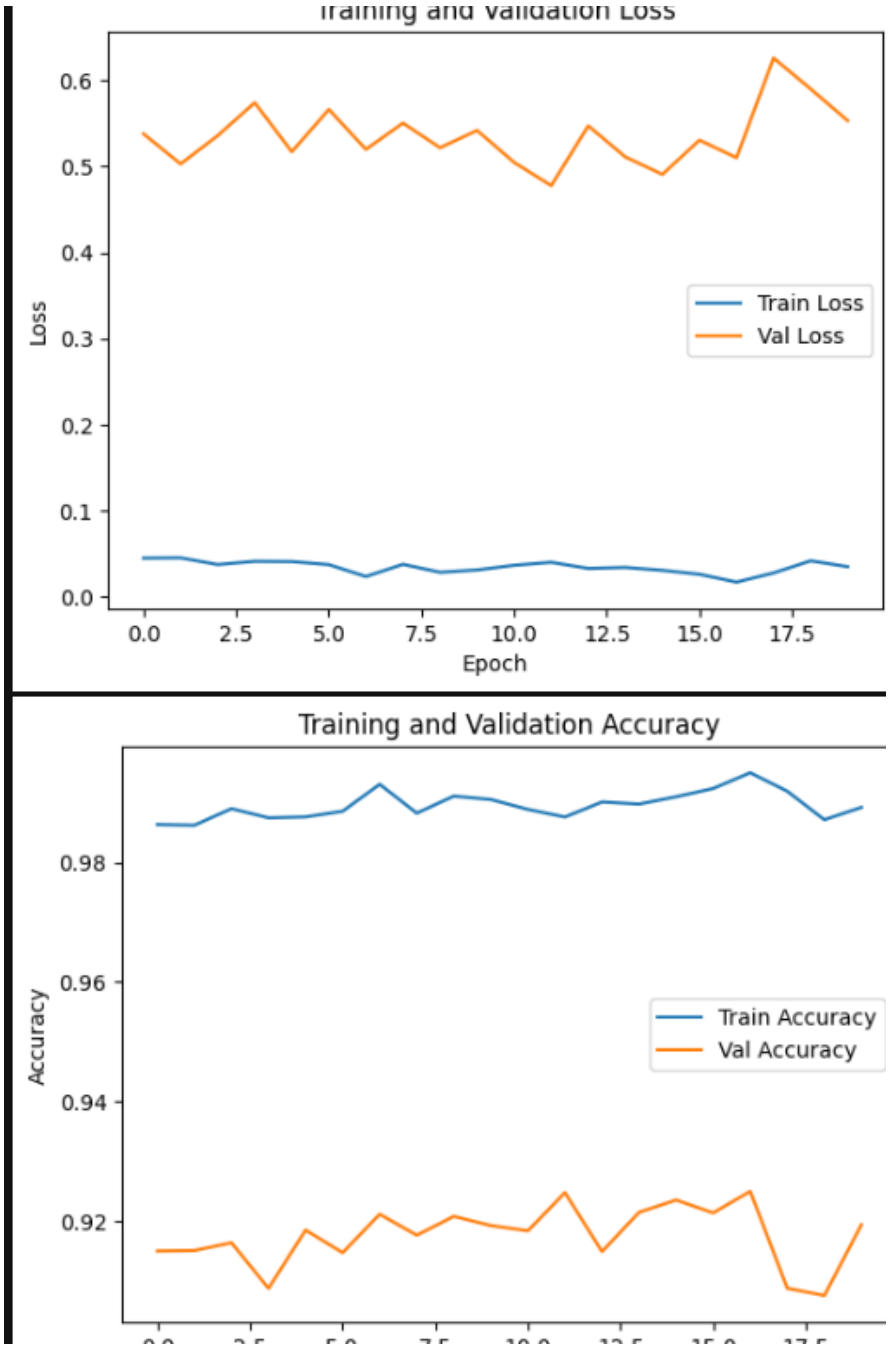


Рисунок 6. График обучения.

Вывод.

При увеличении датасета проблема переобучаемости сети была решена!