Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования «Южно-Уральский государственный университет»
(национальный исследовательский университет)
Институт естественных и точных наук
Кафедра прикладной математики и программирования

Отчет по лабораторной работе № 4
по дисциплине «Современные нейросетевые технологии»

Авторы работы
Студент группы ЕТ-122
_____/ Матвеева А.В.
«____» _____2025 г.

Руководитель работы,
_____/ Кичеев Д.М.
«____» _____2025 г.

Челябинск 2025

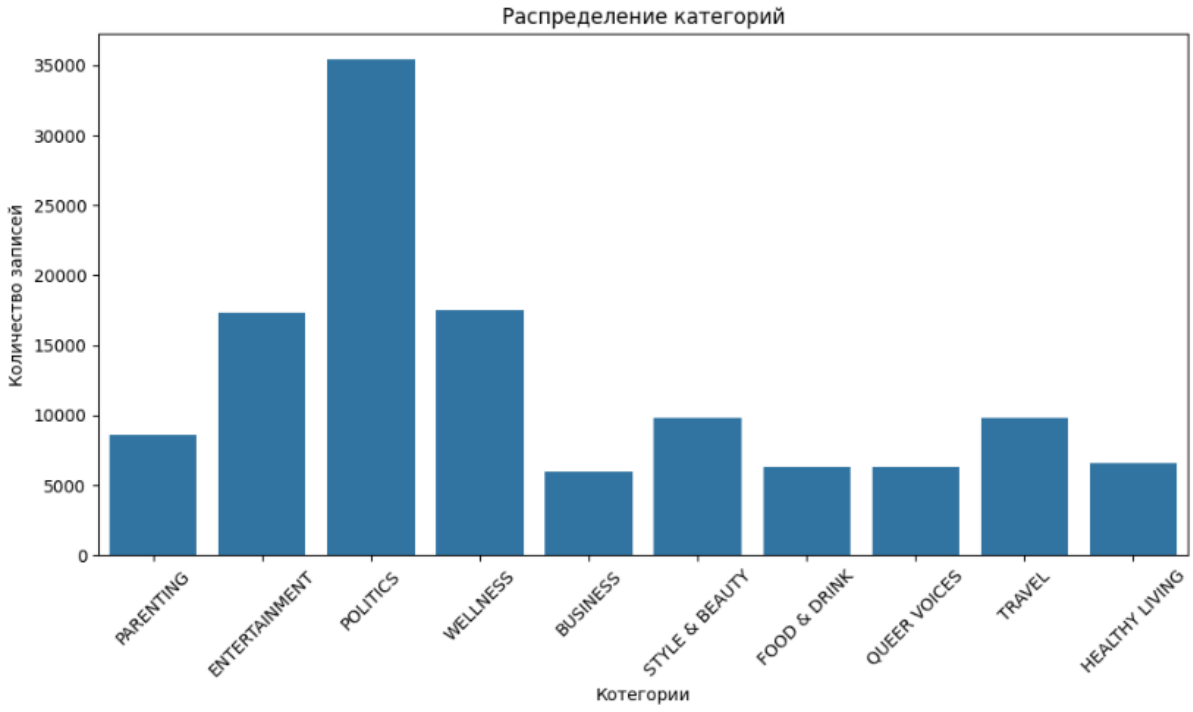
Задание:

Цель: настоящей работы состоит в том, чтобы построить архитектур рекуррентных нейронных сетей, которые позволяют решить практическую задачу с высокими показателями качества.

Выполнение задания:

1. В ходе данного проекта я освоила базовые навыки работы с библиотекой глубокого обучения TensorFlow на примере построения и обучения рекурентных нейронных сетей для задачи многоклассовой классификации.

2. Так как задача состояла в использовании именно рекурентных нейронных сетей мной был взят датасет с текстами. Были использованы данные с kaggle (https://www.kaggle.com/datasets/rmisra/news-category-dataset/data) Данные были предобработаны, в результате чего в датасете остались текстовые новости, разделенные по 10 категориям. Размер валидационного набора: 18552, размер тренировочного набора: 86574, размер тестового набора: 18552.



Тексты имеют разные параметры. После предобработки средняя длина: 6.45, медиана длины: 6.00, максимальная длина: 36, минимальная длина: 2. Исходя из длины текстов были подобраны гиперпараметры и модели.

3. При тестировании моделей я использую метрику качества Accuracy, которая определяется следующим образом:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

где:

- *TP* (True Positives) — количество истинных положительных предсказаний (правильно классифицированные объекты класса 1).
- *TN* (True Negatives) — количество истинных отрицательных предсказаний (правильно классифицированные объекты класса 0).
- *FP* (False Positives) — количество ложных положительных предсказаний (объекты класса 0, ошибочно классифицированные как класс 1).
- *FN* (False Negatives) — количество ложных отрицательных предсказаний (объекты класса 1, ошибочно классифицированные как класс 0).

4. Данные мной были скачаны с сайта kaggle, хранятся локально в файле json.
5. Перед анализом данных я предобработала тексты: привела к нижнему регистру, разбила на отдельные слова (токены), удалила небуквенно-цифровые символы и стоп-слова, применила стеммер Портера к оставшимся словам, чтобы получить их основу.

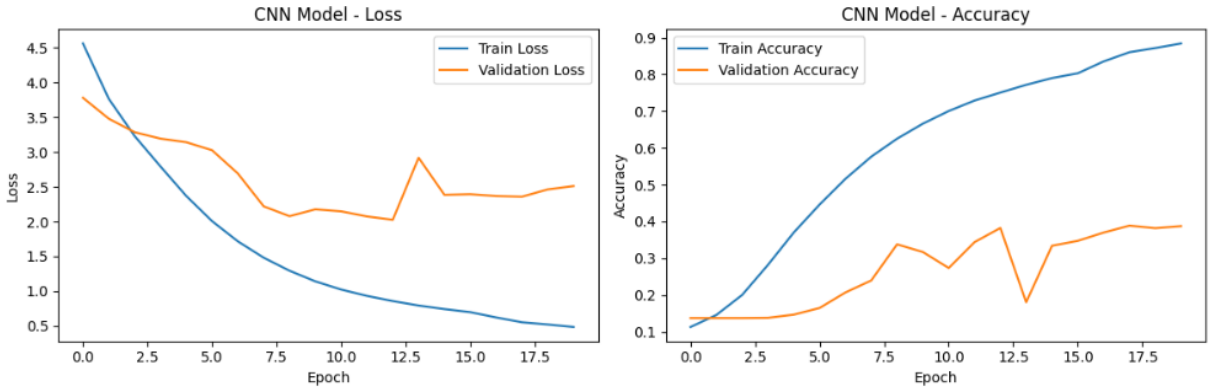
   Далее, я токенизоровала текст и кодировала таргет.

   При составлении обучающего датасета я также аугментировала данные, оставив (создав) за 9000 строк каждого класса.
6. Я использовала стандартную функция fit для обучения нейронных сетей, а после обучения я графически выводила данные обучения (метрику и ошибку) с тренировочной и валидационной выборки. В конце я вывела сравнение всех моделей.
7. Мной были разработаны несколько нейронных сетей для тестирования.
   1) Первая модель - это одномерная сверточная нейронную сеть (1D CNN). Модель начинается со слоя эмбеддинга, преобразующего слова в векторные представления. Затем идет четыре последовательных сверточных слоя (Conv1D) с уменьшающимся количеством фильтров и размером ядра, каждый из которых сопровождается нормализацией данных (BatchNormalization) и функцией активации ReLU. После сверточных слоев применяется глобальное макс-пулингование (GlobalMaxPooling1D) для уменьшения размерности. Далее следуют два полносвязных слоя (Dense) с ReLU активацией, L2 регуляризацией и нормализацией с Dropout для предотвращения

переобучения. Финальный полносвязный слой имеет функцию активации softmax для получения вероятностей классов.

```
def build_cnn_model():
    model = Sequential([
        Embedding(vocab_size, embedding_dim),
        Conv1D(filters=32, kernel_size=3, activation='relu'),
        BatchNormalization(),
        Conv1D(filters=16, kernel_size=2, activation='relu'),
        BatchNormalization(),
        Conv1D(filters=8, kernel_size=2, activation='relu'),
        BatchNormalization(),
        Conv1D(filters=4, kernel_size=2, activation='relu'),
        BatchNormalization(),
        GlobalMaxPooling1D(),
        Dense(128, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.02)),
        BatchNormalization(),
        Dropout(0.6),
        Dense(64, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.02)),
        BatchNormalization(),
        Dropout(0.4),
        Dense(num_categories, activation='softmax')
    ])
    return model
```

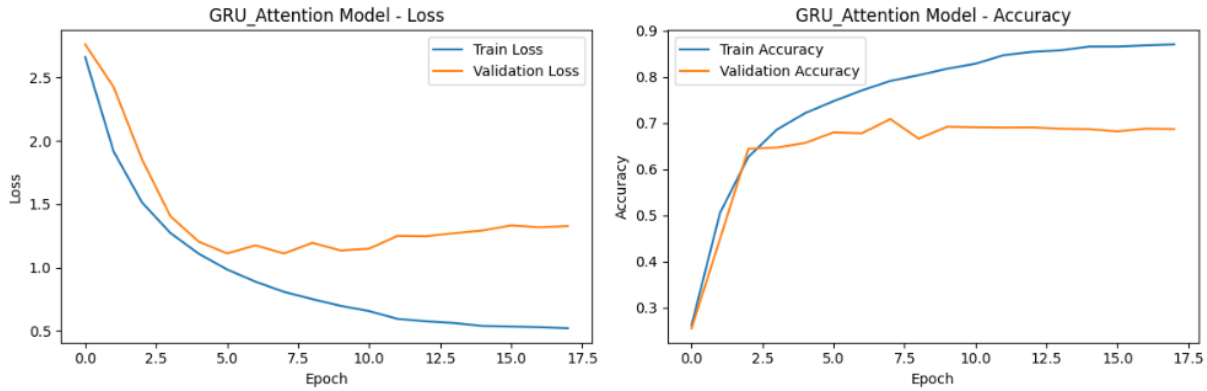
Модель довольно плохо справилась с задачей.



Метрика 0.38 на тестовых данных показывает, что сверточная сеть не улавливает все необходимые признаки.

2) Вторая модель - это рекурентная нейронная сеть. Эта функция строит модель с двумя слоями GRU и механизмом внимания (Attention). Модель начинается со слоя эмбеддинга. Затем два последовательных слоя GRU обрабатывают последовательность, причём второй GRU выдаёт все промежуточные состояния. Слой внимания взвешивает различные части выходных данных второго GRU, выделяя наиболее важные моменты. Результат внимания проходит через слой глобального макс-пулинга. Далее следует полносвязный слой с ReLU активацией и L2 регуляризацией, за которым идут нормализация данных и Dropout. Финальный полносвязный слой с активацией softmax выдает вероятности классов.

```python
def build_gru_attention_model():
    inputs = Input(shape=(max_length,))
    embedding = Embedding(vocab_size, embedding_dim)(inputs)
    gru_out1 = GRU(128, return_sequences=True)(embedding)
    gru_out2 = GRU(32, return_sequences=True)(gru_out1)
    attention = Attention()([gru_out2, gru_out2])
    context = GlobalMaxPooling1D()(attention)
    dense = Dense(32, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.02))(context)
    batch_norm = BatchNormalization()(dense)
    dropout = Dropout(0.6)(batch_norm)
    outputs = Dense(num_categories, activation='softmax')(dropout)
    model = Model(inputs=inputs, outputs=outputs)
    return model
```



Метрика на тестовых данных -0.70. В модели также заметно переобучение. Это скорее всего связано с аугментацией данных. Метрики на валидационных и тестовых данных похожи.

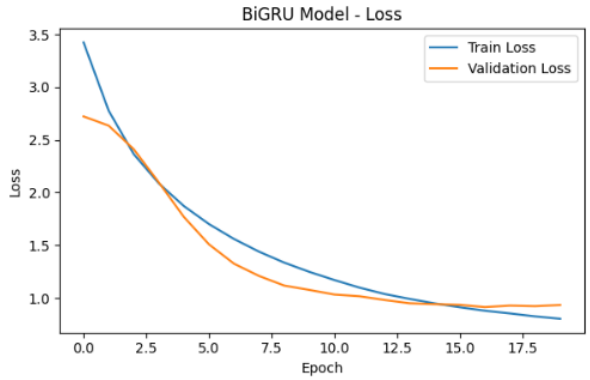
3) Третья модель тоже рекурентная нейронная сеть. Эта функция создает модель для классификации последовательностей, использующую двунаправленный LSTM и механизм внимания. Сначала входные данные проходят через слой эмбеддинга. Затем двунаправленный LSTM обрабатывает последовательность в обоих направлениях, улавливая контекст как до, так и после каждого элемента. Слой внимания взвешивает выходные данные LSTM, выделяя наиболее важные части. Результат внимания подвергается глобальному макс-пулингу. Далее следуют два полносвязных слоя с ReLU активацией, L2 регуляризацией, нормализацией данных и Dropout для предотвращения переобучения. Финальный полносвязный слой с активацией softmax выдает вероятности классов.

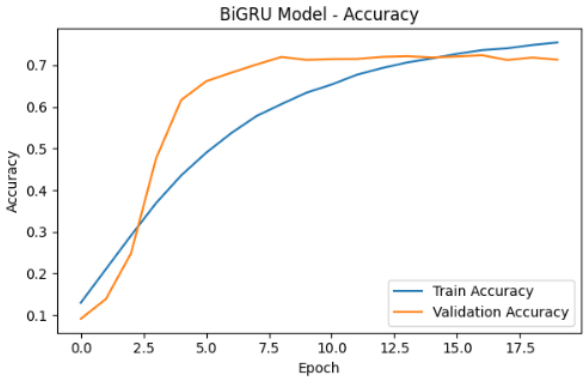
```python
def build_lstm_attention_model():
    inputs = Input(shape=(max_length,))
    embedding = Embedding(vocab_size, embedding_dim)(inputs)
    lstm_out = Bidirectional(LSTM(64, return_sequences=True))(embedding)
    attention = Attention()([lstm_out, lstm_out])
    context = GlobalMaxPooling1D()(attention)
    dense1 = Dense(32, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.02))(context)
    batch_norm = BatchNormalization()(dense1)
    dropout = Dropout(0.6)(batch_norm)
    dense2 = Dense(32, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.02))(dropout)
    batch_norm2 = BatchNormalization()(dense2)
    outputs = Dense(num_categories, activation='softmax')(batch_norm2)
    model = Model(inputs=inputs, outputs=outputs)
    return model
```

Полученая метрика на тестовых данных 0.67. В данной модели не видно сильного переобучения, с учетом аугментации. Возможно, при большем количестве эпох метрика может улучшится.

4) Четвертая модель - третья рекурентная сеть. Эта функция строит модель на основе двух двунаправленных GRU-слоев (BiGRU) для классификации последовательностей. Сначала входные данные проходят через слой эмбеддинга. Затем два последовательных слоя BiGRU обрабатывают последовательность в обоих направлениях. Первый BiGRU возвращает все промежуточные состояния, а второй — только последнее. После каждого BiGRU следует слой нормализации данных. Далее идет полносвязный слой с ReLU активацией и L2 регуляризацией, за которым следуют нормализация данных и Dropout. Финальный полносвязный слой с активацией softmax выдает вероятности классов.

```python
def build_bigru_model():
    model = Sequential([
        Embedding(vocab_size, embedding_dim),
        Bidirectional(GRU(32, return_sequences=True, dropout=0.3, recurrent_dropout=0.1)),
        BatchNormalization(),
        Bidirectional(GRU(16, return_sequences=False, dropout=0.3, recurrent_dropout=0.1)),
        BatchNormalization(),
        Dense(16, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.02)),
        BatchNormalization(),
        Dropout(0.6),
        Dense(num_categories, activation='softmax')
    ])
    return model
```





Модель показала лучший результат (0.72), несмотря

8. Результаты:

Несмотря на большое количество слоев, сверточная нейронная сеть не может уловить все нужные закономерности и показывает на том же количестве эпох результат хуже, чем рекурентные нейронные сети. Так как текст в датасете не длинный, хорошую роль сыграли механизм внимания и нормализация данных.

Модель	Время обучения (сек)	Количество слоёв	Особенности	Accuracy
model_cnn	389,09	4 сверточных слоя	Сверточная нейронная сеть	0.37
model_gru_attention	2303,94	2 слоя GRU	Рекуррентная нейронная сеть, используется GRU и слой внимания	0.70
model_lstm_attention	1867,83	1 двунаправленный слой LSTM	Рекуррентная нейронная сеть, используется двунаправленный слой LSTM и механизм внимания	0.67
model_bigru	1546,16	2 слоя BiGRU	Рекуррентная нейронная сеть. Используются слои	0.72

		BiGRU, возвращающие промежуточные и последние состояния