Цель работы

Целью данной работы является изучение рекуррентных нейронных сетей и их применение. В процессе работы будет рассмотрено несколько различных архитектур рекуррентных нейронных сетей, включая RNN, LSTM и GRU.

Задание

1. Подготовка данных
2. Архитектура RNN
3. Обучение RNN
4. Настройка гиперпараметров и выбор модели
5. Сравнение с существующими решениями

Выполнение работы

В качестве исходного набора данных выбран датасет, отражающий временную последовательность значений величины электроэнергии, вырабатываемой ветряными турбинами в Германии.

Первые 5 строк набора данных приведены на рис.1. dt – дата и время замера (через каждые 15 минут), MW – количество энергии в MB. Размер выборки – 50000 строк.

	dt	MW
0	2011-01-01 00:00:00	3416.0
1	2011-01-01 00:15:00	4755.0
2	2011-01-01 00:30:00	4939.0
3	2011-01-01 00:45:00	4939.0
4	2011-01-01 01:00:00	4998.0

Рисунок 1

В качестве индекса назначена дата, по ней выполнена сортировка данных (рис.2).

	MW
dt	
2011-01-01 00:00:00	3416.0
2011-01-01 00:15:00	4755.0
2011-01-01 00:30:00	4939.0
2011-01-01 00:45:00	4939.0
2011-01-01 01:00:00	4998.0

Рисунок 2

На рис.3 приведен график значений параметра.

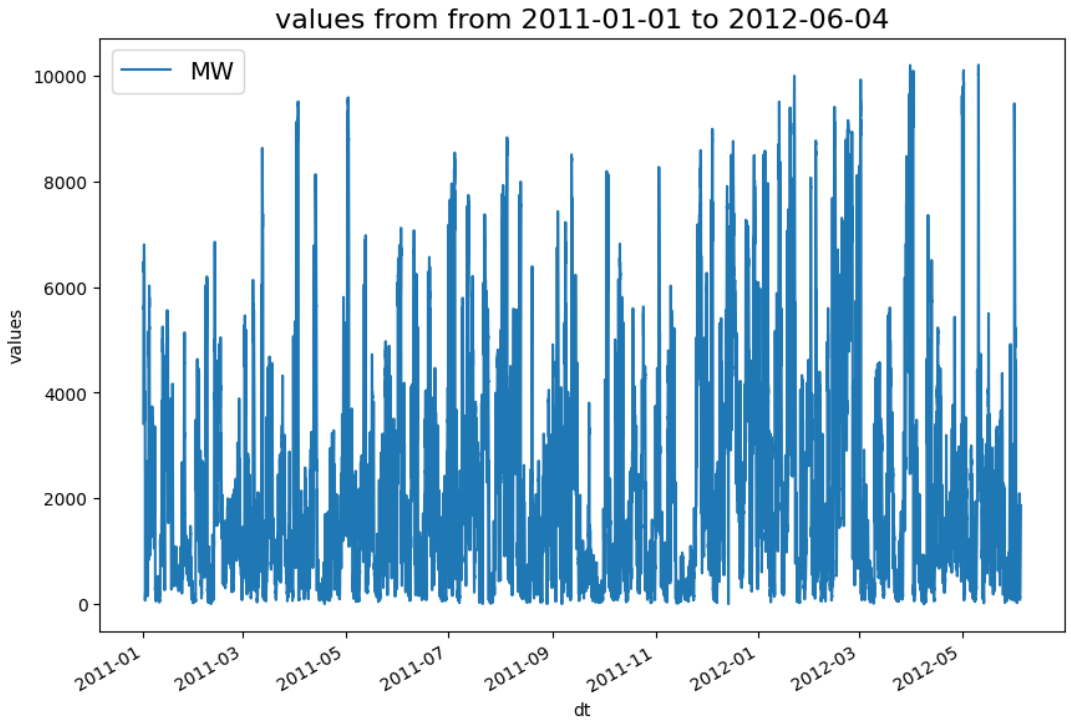

Рисунок 3

Данные разделены на обучающую и валидационную выборку (рис.4). График приведен на рис.5.

```
# Define the split time
split_ratio = 0.8  # 80% for the training set
split_time = int(len(df) * split_ratio)

# Get the train set
time_train = df['dt'].iloc[:split_time]
x_train = df['MW'].iloc[:split_time]

# Get the validation set
time_valid = df['dt'].iloc[split_time:]
x_valid = df['MW'].iloc[split_time:]
```

Рисунок 4

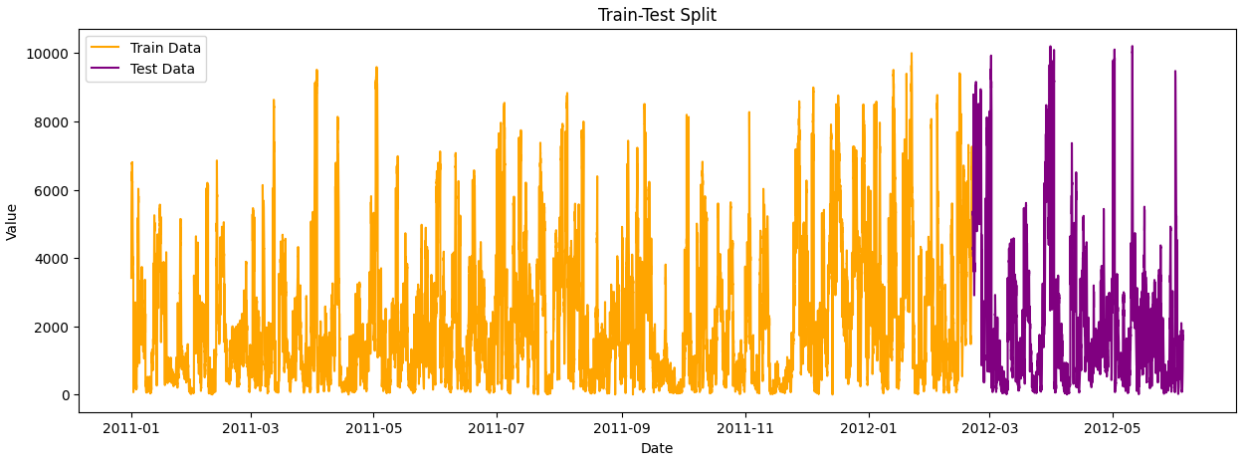

Рисунок 5

Архитектуры RNN

1. SimpleRNN

Рекуррентные нейронные сети (Recurrent Neural Networks,RNN) – сети с обратными или перекрестными связями между различными слоями нейронов. Описание архитектуры модели приведена на рис.6.

```
model_rn = tf.keras.models.Sequential([
  tf.keras.layers.Lambda(lambda x: tf.expand_dims(x, axis=-1),
                      input_shape=[window_size]),
  tf.keras.layers.SimpleRNN(256, return_sequences=True),
  tf.keras.layers.SimpleRNN(128),
  tf.keras.layers.Dropout(0.5),
  tf.keras.layers.Dense(1, kernel_regularizer=regularizers.l1_l2(l1=0.1, l2=0.1)),
  tf.keras.layers.Lambda(lambda x: x * 100.0)
])
```

Рисунок 6

Модель обучалась в течение 10 эпох.

Результат запуска с оптимизатором Adam приведен на рис.7. Видно, что модель не переобучается, и достигает лучшего значения функции потерь. На рисунке 8 приведен график функции потерь.

```
Epoch 1/10
1250/1250 [==============================] - 51s 39ms/step - loss: 911.4874 - val_loss: 781.5694
Epoch 2/10
1250/1250 [==============================] - 26s 21ms/step - loss: 545.1022 - val_loss: 592.5516
Epoch 3/10
1250/1250 [==============================] - 27s 21ms/step - loss: 394.3986 - val_loss: 464.2535
Epoch 4/10
1250/1250 [==============================] - 27s 22ms/step - loss: 317.5264 - val_loss: 461.0796
Epoch 5/10
1250/1250 [==============================] - 28s 22ms/step - loss: 268.5903 - val_loss: 349.1442
Epoch 6/10
1250/1250 [==============================] - 27s 21ms/step - loss: 237.9586 - val_loss: 315.0747
Epoch 7/10
1250/1250 [==============================] - 27s 21ms/step - loss: 216.7818 - val_loss: 276.0563
Epoch 8/10
1250/1250 [==============================] - 27s 22ms/step - loss: 201.5307 - val_loss: 254.6733
Epoch 9/10
1250/1250 [==============================] - 27s 21ms/step - loss: 190.5519 - val_loss: 237.3799
Epoch 10/10
1250/1250 [==============================] - 27s 21ms/step - loss: 182.9746 - val_loss: 222.2040
```

Рисунок 7

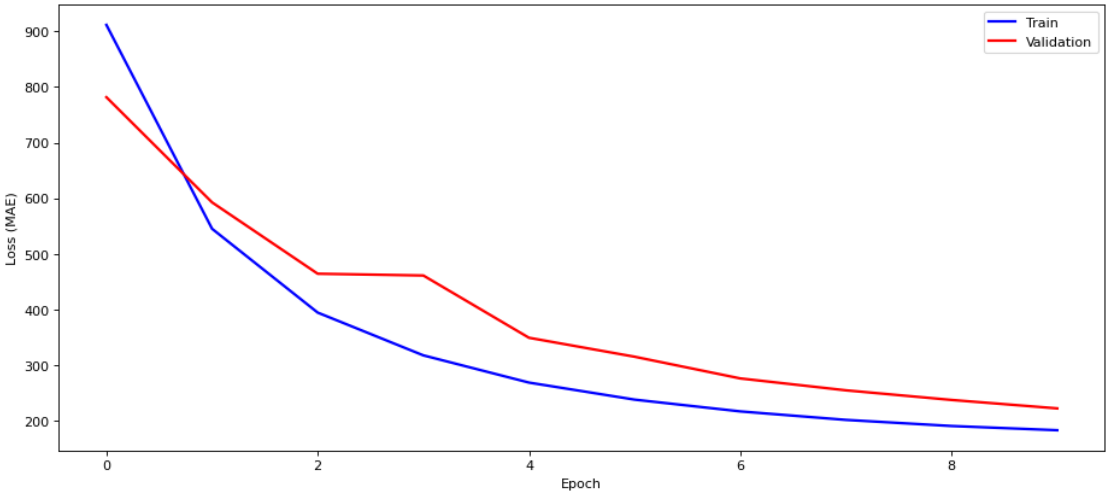

Рисунок 8

Результат запуска с оптимизатором SGD приведен на рис.9. График функции потерь приведен на рис.10.

```
Epoch 1/10
1250/1250 [==============================] - 30s 22ms/step - loss: 1415.7988 - val_loss: 1841.2465
Epoch 2/10
1250/1250 [==============================] - 27s 22ms/step - loss: 1419.5812 - val_loss: 1794.5596
Epoch 3/10
1250/1250 [==============================] - 26s 20ms/step - loss: 1416.8240 - val_loss: 1784.9937
Epoch 4/10
1250/1250 [==============================] - 26s 21ms/step - loss: 1418.9570 - val_loss: 1797.8635
Epoch 5/10
1250/1250 [==============================] - 26s 21ms/step - loss: 1418.3126 - val_loss: 1761.1338
Epoch 6/10
1250/1250 [==============================] - 26s 21ms/step - loss: 1419.1707 - val_loss: 1811.4818
Epoch 7/10
1250/1250 [==============================] - 27s 22ms/step - loss: 1420.6250 - val_loss: 1810.0569
Epoch 8/10
1250/1250 [==============================] - 25s 20ms/step - loss: 1419.6006 - val_loss: 1759.7286
Epoch 9/10
1250/1250 [==============================] - 28s 22ms/step - loss: 1416.8566 - val_loss: 1795.5972
Epoch 10/10
1250/1250 [==============================] - 26s 21ms/step - loss: 1421.0708 - val_loss: 1789.4392
```

Рисунок 9

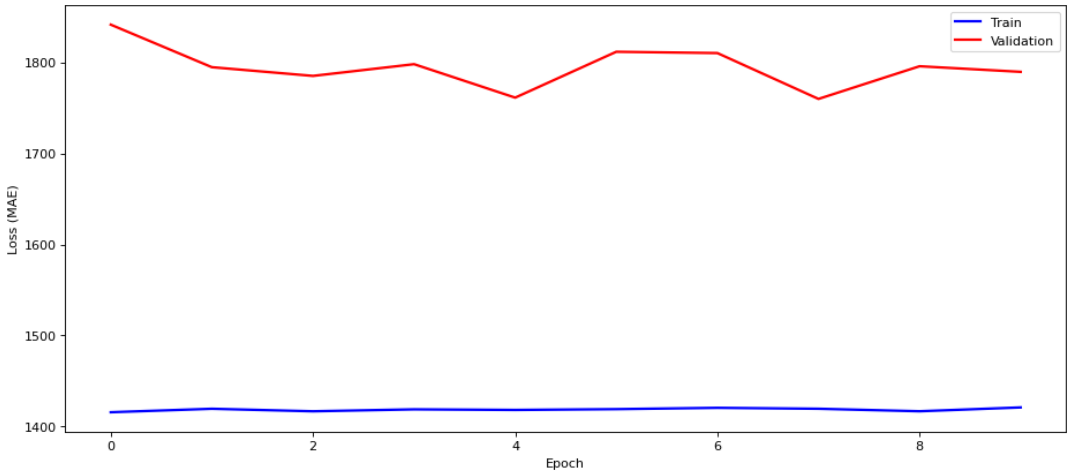

Рисунок 10

Результат запуска с оптимизатором Momentum приведен на рис.11. График функции потерь приведен на рис.12.

```
Epoch 1/10
1250/1250 [==============================] - 41s 31ms/step - loss: 1413.2865 - val_loss: 1794.9297
Epoch 2/10
1250/1250 [==============================] - 26s 21ms/step - loss: 1413.3262 - val_loss: 1888.8490
Epoch 3/10
1250/1250 [==============================] - 27s 21ms/step - loss: 1418.3740 - val_loss: 1892.2015
Epoch 4/10
1250/1250 [==============================] - 30s 24ms/step - loss: 1416.2640 - val_loss: 1844.4237
Epoch 5/10
1250/1250 [==============================] - 28s 23ms/step - loss: 1415.7500 - val_loss: 1896.0460
Epoch 6/10
1250/1250 [==============================] - 29s 23ms/step - loss: 1413.8885 - val_loss: 1876.0752
Epoch 7/10
1250/1250 [==============================] - 28s 22ms/step - loss: 1415.2937 - val_loss: 1917.9990
Epoch 8/10
1250/1250 [==============================] - 28s 22ms/step - loss: 1415.1982 - val_loss: 1779.0243
Epoch 9/10
1250/1250 [==============================] - 28s 22ms/step - loss: 1414.8553 - val_loss: 1837.7406
Epoch 10/10
1250/1250 [==============================] - 28s 22ms/step - loss: 1416.9137 - val_loss: 1846.2150
```

Рисунок 11

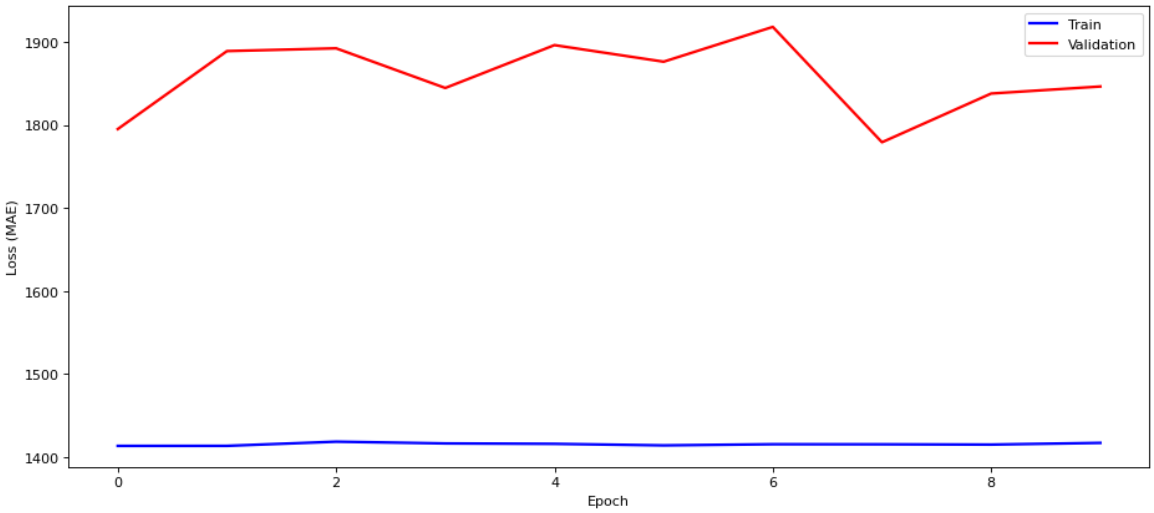

Рисунок 12

Результат запуска с оптимизатором RMSprop приведен на рис.13. График функции потерь приведен на рис.14.

```
Epoch 1/10
1250/1250 [==============================] - 31s 23ms/step - loss: 870.6238 - val_loss: 782.4727
Epoch 2/10
1250/1250 [==============================] - 28s 22ms/step - loss: 535.3803 - val_loss: 604.4910
Epoch 3/10
1250/1250 [==============================] - 28s 23ms/step - loss: 394.0846 - val_loss: 493.2908
Epoch 4/10
1250/1250 [==============================] - 29s 23ms/step - loss: 322.6066 - val_loss: 405.2794
Epoch 5/10
1250/1250 [==============================] - 27s 22ms/step - loss: 281.6136 - val_loss: 411.3339
Epoch 6/10
1250/1250 [==============================] - 27s 21ms/step - loss: 255.1040 - val_loss: 348.4691
Epoch 7/10
1250/1250 [==============================] - 27s 22ms/step - loss: 237.9112 - val_loss: 300.2887
Epoch 8/10
1250/1250 [==============================] - 27s 21ms/step - loss: 223.6962 - val_loss: 347.0493
Epoch 9/10
1250/1250 [==============================] - 27s 22ms/step - loss: 215.0981 - val_loss: 304.6273
Epoch 10/10
1250/1250 [==============================] - 27s 21ms/step - loss: 208.3069 - val_loss: 253.8154
```

Рисунок 13

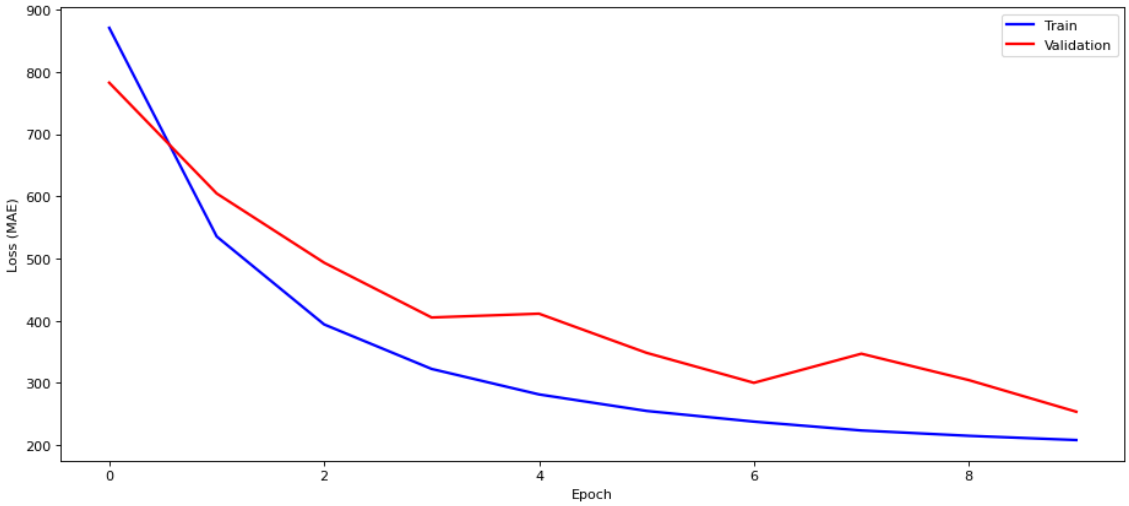

Рисунок 14

Делаем вывод о том, что наилучший оптимизатор для модели SimpleRNN применимо к нашей задаче – Adam.

2. LSTM

Вариация RNN, в которой объем памяти для модели увеличивается для размещения данных за более длительный период.

Описание архитектуры модели приведено на рис.15.

```
model_bl = tf.keras.models.Sequential([
  tf.keras.layers.Lambda(lambda x: tf.expand_dims(x, axis=-1),
                      input_shape=[window_size]),
  tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(128, return_sequences=True)),
  tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(128)),
  tf.keras.layers.Dense(256, activation='relu'),
  tf.keras.layers.Dropout(0.4),
  tf.keras.layers.Dense(1),
#   tf.keras.layers.Lambda(lambda x: x * 100.0)
])
```

Рисунок 15

Модель обучалась в течение 10 эпох.

Результат запуска с оптимизатором Adam приведен на рис.16. Видно, что модель достигает лучшего значения функции потерь. На рисунке 17 приведен график функции потерь.

```
Epoch 1/10
1250/1250 [==============================] - 34s 19ms/step - loss: 1755.3640 - val_loss: 1492.9065
Epoch 2/10
1250/1250 [==============================] - 16s 12ms/step - loss: 988.2839 - val_loss: 795.8721
Epoch 3/10
1250/1250 [==============================] - 15s 12ms/step - loss: 524.8098 - val_loss: 553.7314
Epoch 4/10
1250/1250 [==============================] - 16s 12ms/step - loss: 335.9855 - val_loss: 371.4703
Epoch 5/10
1250/1250 [==============================] - 16s 13ms/step - loss: 260.5960 - val_loss: 314.6245
Epoch 6/10
1250/1250 [==============================] - 16s 13ms/step - loss: 229.7703 - val_loss: 237.8281
Epoch 7/10
1250/1250 [==============================] - 16s 13ms/step - loss: 220.5684 - val_loss: 199.8091
Epoch 8/10
1250/1250 [==============================] - 20s 16ms/step - loss: 203.5006 - val_loss: 179.1137
Epoch 9/10
1250/1250 [==============================] - 20s 16ms/step - loss: 205.5265 - val_loss: 165.8624
Epoch 10/10
1250/1250 [==============================] - 19s 15ms/step - loss: 196.3822 - val_loss: 166.1301
```

Рисунок 16

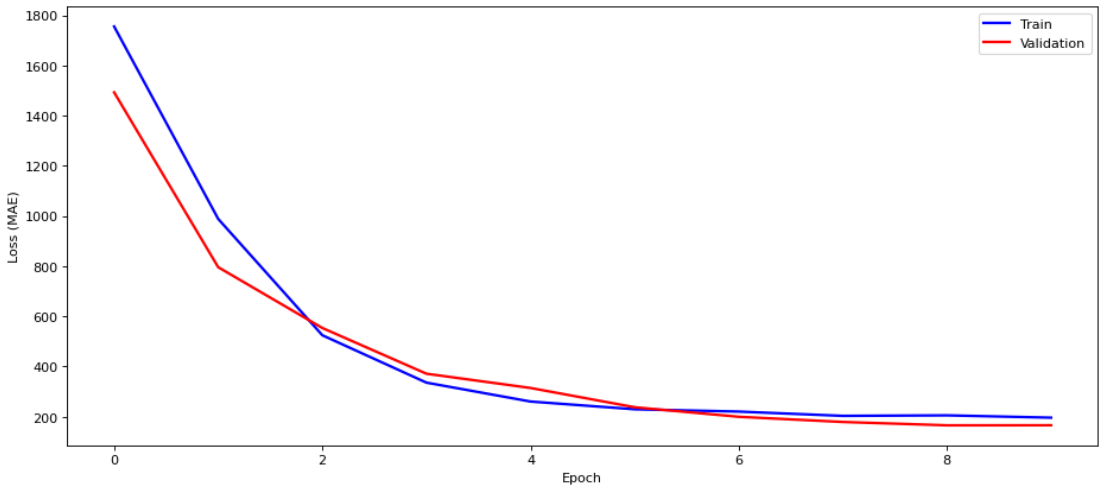

Рисунок 17

Результат запуска с оптимизатором SGD приведен на рис.18. Благодаря наличию слоя Dropout, модель не переобучается. На рисунке 19 приведен график функции потерь.

```
Epoch 1/10
1250/1250 [==============================] - 23s 13ms/step - loss: 2066.7205 - val_loss: 2205.8704
Epoch 2/10
1250/1250 [==============================] - 15s 12ms/step - loss: 2030.6051 - val_loss: 2103.1943
Epoch 3/10
1250/1250 [==============================] - 15s 12ms/step - loss: 1739.8458 - val_loss: 1650.3113
Epoch 4/10
1250/1250 [==============================] - 17s 14ms/step - loss: 1485.9275 - val_loss: 1612.7251
Epoch 5/10
1250/1250 [==============================] - 15s 12ms/step - loss: 1465.1100 - val_loss: 1637.6294
Epoch 6/10
1250/1250 [==============================] - 15s 12ms/step - loss: 1458.6093 - val_loss: 1653.9775
Epoch 7/10
1250/1250 [==============================] - 15s 12ms/step - loss: 1459.9058 - val_loss: 1657.6047
Epoch 8/10
1250/1250 [==============================] - 17s 13ms/step - loss: 1461.5278 - val_loss: 1656.5599
Epoch 9/10
1250/1250 [==============================] - 16s 13ms/step - loss: 1461.3877 - val_loss: 1655.3562
Epoch 10/10
1250/1250 [==============================] - 15s 12ms/step - loss: 1454.2848 - val_loss: 1661.2969
```

Рисунок 18

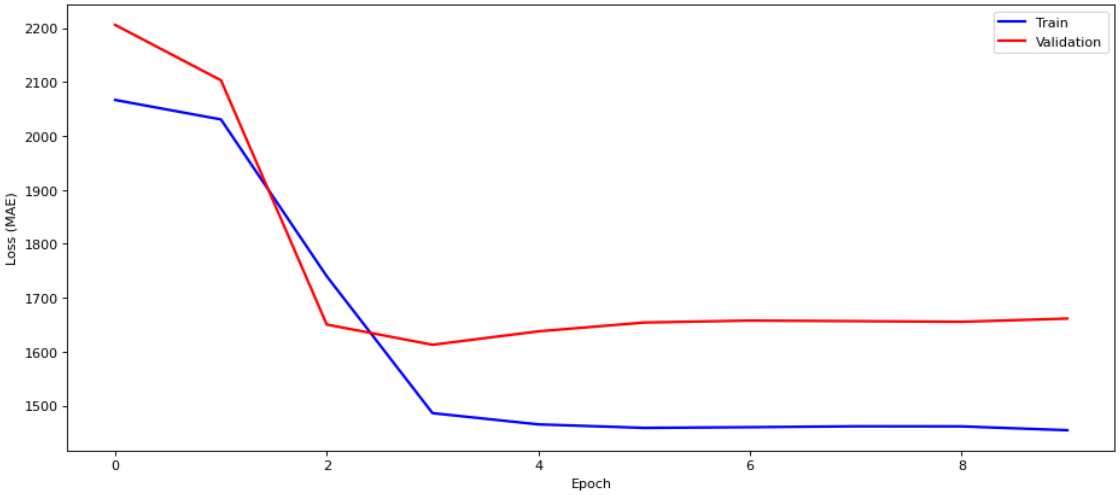

Рисунок 19

Результат запуска с оптимизатором Momentum приведен на рис.20. Потери практически не изменяются в процессе обучения. На рисунке 21 приведен график функции потерь.

```
Epoch 1/10
1250/1250 [==============================] - 23s 13ms/step - loss: 2058.9824 - val_loss: 2161.8132
Epoch 2/10
1250/1250 [==============================] - 17s 14ms/step - loss: 1668.7278 - val_loss: 1615.0780
Epoch 3/10
1250/1250 [==============================] - 20s 16ms/step - loss: 1444.8323 - val_loss: 1686.0717
Epoch 4/10
1250/1250 [==============================] - 18s 14ms/step - loss: 1458.5604 - val_loss: 1681.7443
Epoch 5/10
1250/1250 [==============================] - 15s 12ms/step - loss: 1458.5488 - val_loss: 1681.3964
Epoch 6/10
1250/1250 [==============================] - 15s 12ms/step - loss: 1456.3630 - val_loss: 1685.7772
Epoch 7/10
1250/1250 [==============================] - 16s 13ms/step - loss: 1460.1407 - val_loss: 1680.8990
Epoch 8/10
1250/1250 [==============================] - 16s 12ms/step - loss: 1463.9998 - val_loss: 1676.9437
Epoch 9/10
1250/1250 [==============================] - 15s 12ms/step - loss: 1466.8365 - val_loss: 1672.5204
Epoch 10/10
1250/1250 [==============================] - 15s 12ms/step - loss: 1463.5878 - val_loss: 1674.7776
```

Рисунок 20

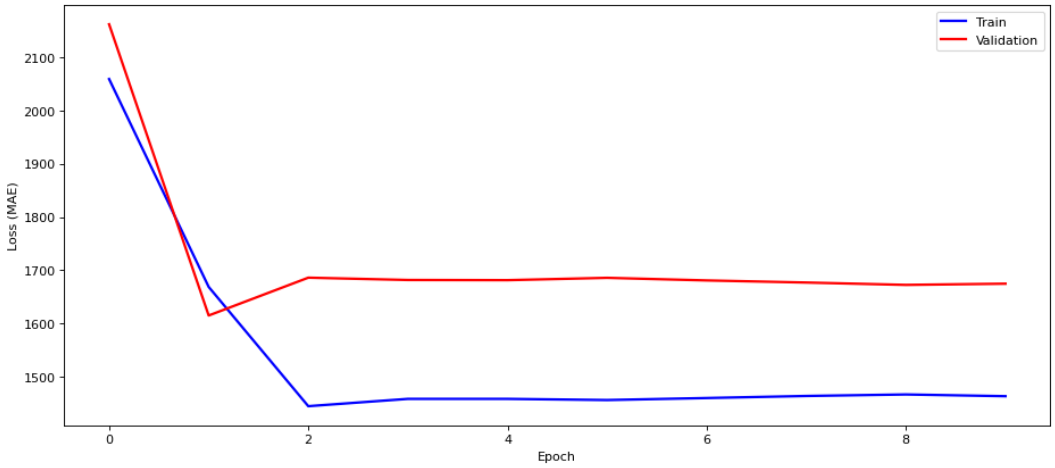

Рисунок 21

Результат запуска с оптимизатором RMSprop приведен на рис.22. Потери значительно снижаются в процессе обучения. На рисунке 23 приведен график функции потерь.

```
Epoch 1/10
1250/1250 [==============================] - 26s 14ms/step - loss: 1860.3030 - val_loss: 1732.1401
Epoch 2/10
1250/1250 [==============================] - 16s 13ms/step - loss: 1268.1632 - val_loss: 1104.9458
Epoch 3/10
1250/1250 [==============================] - 17s 13ms/step - loss: 755.5355 - val_loss: 742.7878
Epoch 4/10
1250/1250 [==============================] - 19s 15ms/step - loss: 488.4181 - val_loss: 531.4764
Epoch 5/10
1250/1250 [==============================] - 17s 14ms/step - loss: 366.2580 - val_loss: 431.9541
Epoch 6/10
1250/1250 [==============================] - 15s 12ms/step - loss: 308.7145 - val_loss: 403.0281
Epoch 7/10
1250/1250 [==============================] - 15s 12ms/step - loss: 278.9854 - val_loss: 299.5013
Epoch 8/10
1250/1250 [==============================] - 15s 12ms/step - loss: 260.0689 - val_loss: 244.8537
Epoch 9/10
1250/1250 [==============================] - 15s 12ms/step - loss: 250.0439 - val_loss: 275.7071
Epoch 10/10
1250/1250 [==============================] - 15s 12ms/step - loss: 237.5375 - val_loss: 224.5082
```

Рисунок 22

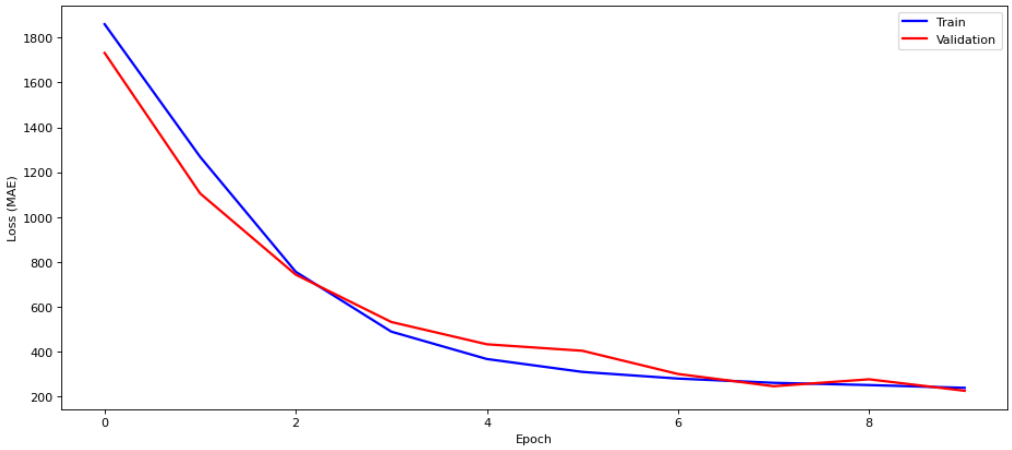

Рисунок 23

Наилучших результатов модель достигает с применением оптимизаторов Adam и RMSprop.

3. GRU

Вариация RNN, поддерживающая выборочное сохранение памяти. В эту модель добавлены шлюзы обновления и забывание в скрытом слое, что позволяет хранить информацию в памяти и удалять ее.

Описание архитектуры модели приведено на рис.24.

```
model_gru = tf.keras.models.Sequential([
  tf.keras.layers.Lambda(lambda x: tf.expand_dims(x, axis=-1),
                      input_shape=[window_size]),
  tf.keras.layers.GRU(256, return_sequences=True),
  tf.keras.layers.GRU(128),
  tf.keras.layers.Dropout(0.4),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 100.0)
])
```

Рисунок 24

Результат запуска с оптимизатором Adam приведен на рис.25. Потери значительно снижаются в процессе обучения. На рисунке 26 приведен график функции потерь.

```
Epoch 1/10
1250/1250 [==============================] - 15s 9ms/step - loss: 937.3234 - val_loss: 783.4882
Epoch 2/10
1250/1250 [==============================] - 11s 9ms/step - loss: 556.0599 - val_loss: 594.3708
Epoch 3/10
1250/1250 [==============================] - 11s 9ms/step - loss: 400.9240 - val_loss: 476.6862
Epoch 4/10
1250/1250 [==============================] - 11s 9ms/step - loss: 321.6938 - val_loss: 383.9645
Epoch 5/10
1250/1250 [==============================] - 11s 9ms/step - loss: 275.3958 - val_loss: 340.0323
Epoch 6/10
1250/1250 [==============================] - 12s 10ms/step - loss: 247.3547 - val_loss: 290.3085
Epoch 7/10
1250/1250 [==============================] - 11s 9ms/step - loss: 224.8487 - val_loss: 254.8476
Epoch 8/10
1250/1250 [==============================] - 17s 13ms/step - loss: 212.1335 - val_loss: 245.8351
Epoch 9/10
1250/1250 [==============================] - 13s 10ms/step - loss: 207.0831 - val_loss: 220.1118
Epoch 10/10
1250/1250 [==============================] - 12s 9ms/step - loss: 204.1248 - val_loss: 226.0244
```

Рисунок 25

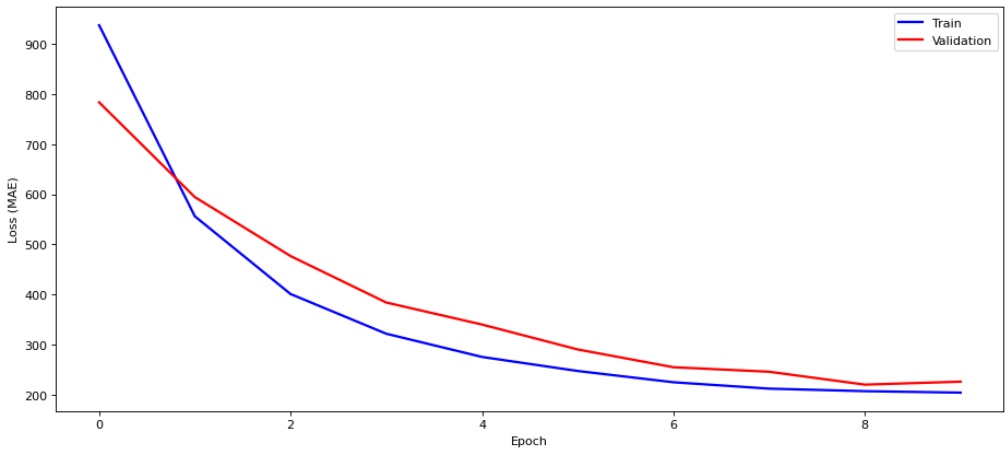

Рисунок 26

Результат запуска с оптимизатором SGD приведен на рис.27. На рисунке 28 приведен график функции потерь.

```
Epoch 1/10
1250/1250 [==============================] - 17s 12ms/step - loss: 1416.4540 - val_loss: 1779.4608
Epoch 2/10
1250/1250 [==============================] - 10s 8ms/step - loss: 1416.7770 - val_loss: 1828.8011
Epoch 3/10
1250/1250 [==============================] - 12s 10ms/step - loss: 1421.6748 - val_loss: 1732.8766
Epoch 4/10
1250/1250 [==============================] - 11s 9ms/step - loss: 1424.0653 - val_loss: 1764.5333
Epoch 5/10
1250/1250 [==============================] - 11s 9ms/step - loss: 1424.7257 - val_loss: 1766.2535
Epoch 6/10
1250/1250 [==============================] - 10s 8ms/step - loss: 1424.4111 - val_loss: 1782.7727
Epoch 7/10
1250/1250 [==============================] - 13s 10ms/step - loss: 1427.8888 - val_loss: 1743.0944
Epoch 8/10
1250/1250 [==============================] - 19s 15ms/step - loss: 1429.2142 - val_loss: 1762.4691
Epoch 9/10
1250/1250 [==============================] - 15s 12ms/step - loss: 1424.5660 - val_loss: 1762.1002
Epoch 10/10
1250/1250 [==============================] - 11s 9ms/step - loss: 1426.4012 - val_loss: 1756.2324
```

Рисунок 27

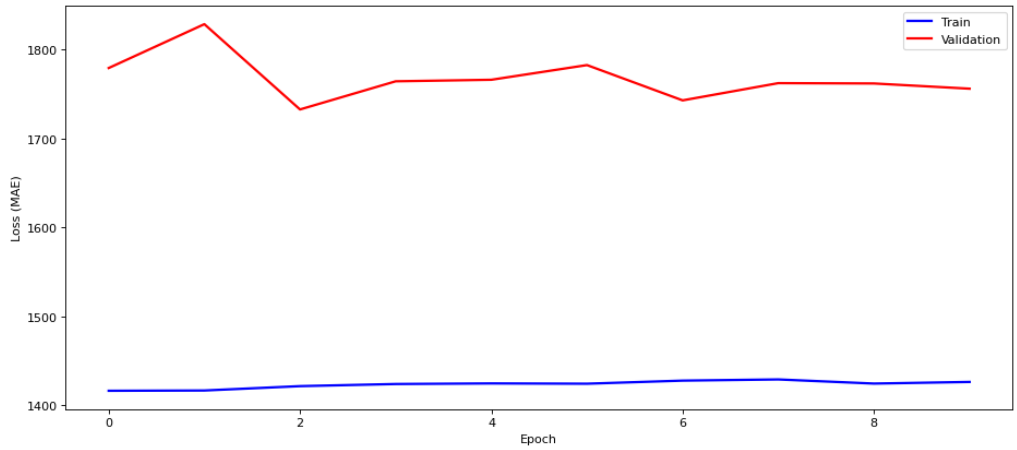

Рисунок 28

Результат запуска с оптимизатором Momentum приведен на рис.29. На рисунке 30 приведен график функции потерь.

```
Epoch 1/10
1250/1250 [==============================] - 14s 9ms/step - loss: 1416.1448 - val_loss: 1773.3597
Epoch 2/10
1250/1250 [==============================] - 11s 9ms/step - loss: 1419.7971 - val_loss: 1864.7727
Epoch 3/10
1250/1250 [==============================] - 12s 10ms/step - loss: 1422.4601 - val_loss: 1801.7390
Epoch 4/10
1250/1250 [==============================] - 11s 8ms/step - loss: 1417.6733 - val_loss: 1917.5557
Epoch 5/10
1250/1250 [==============================] - 11s 9ms/step - loss: 1422.5762 - val_loss: 1858.1591
Epoch 6/10
1250/1250 [==============================] - 14s 11ms/step - loss: 1421.0002 - val_loss: 1790.6622
Epoch 7/10
1250/1250 [==============================] - 13s 10ms/step - loss: 1422.3748 - val_loss: 1807.3695
Epoch 8/10
1250/1250 [==============================] - 16s 12ms/step - loss: 1421.0228 - val_loss: 1845.9700
Epoch 9/10
1250/1250 [==============================] - 11s 9ms/step - loss: 1421.3523 - val_loss: 1819.7897
Epoch 10/10
1250/1250 [==============================] - 10s 8ms/step - loss: 1420.6249 - val_loss: 1832.0741
```

Рисунок 29

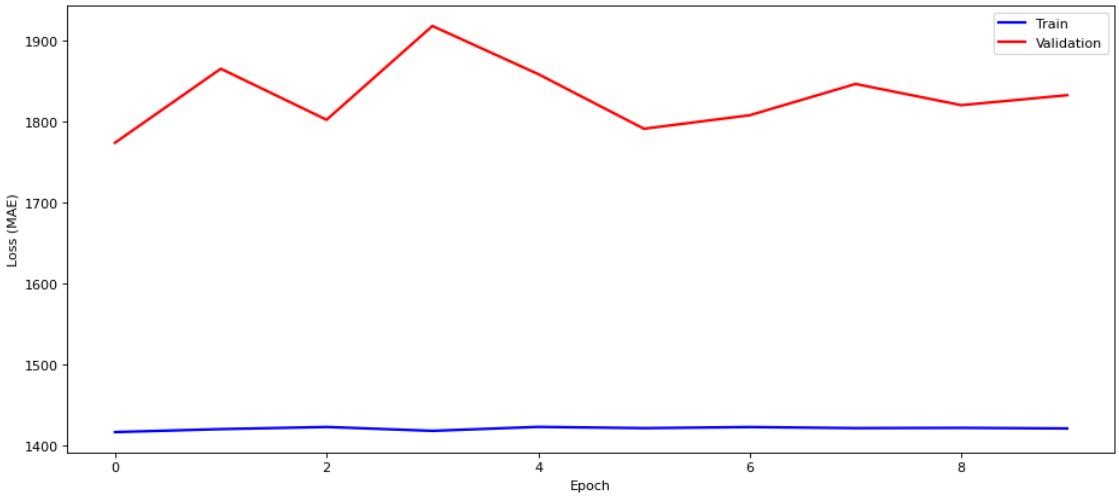

Рисунок 30

Результат запуска с оптимизатором RMSprop приведен на рис.31. На рисунке 32 приведен график функции потерь.

```
Epoch 1/10
1250/1250 [==============================] - 24s 15ms/step - loss: 914.1360 - val_loss: 785.5862
Epoch 2/10
1250/1250 [==============================] - 11s 9ms/step - loss: 556.9304 - val_loss: 569.5705
Epoch 3/10
1250/1250 [==============================] - 12s 10ms/step - loss: 410.0522 - val_loss: 549.3918
Epoch 4/10
1250/1250 [==============================] - 11s 9ms/step - loss: 340.3200 - val_loss: 451.3372
Epoch 5/10
1250/1250 [==============================] - 11s 9ms/step - loss: 297.5688 - val_loss: 366.7338
Epoch 6/10
1250/1250 [==============================] - 11s 9ms/step - loss: 267.6719 - val_loss: 396.8355
Epoch 7/10
1250/1250 [==============================] - 11s 9ms/step - loss: 247.7889 - val_loss: 290.2387
Epoch 8/10
1250/1250 [==============================] - 13s 10ms/step - loss: 237.9822 - val_loss: 273.1628
Epoch 9/10
1250/1250 [==============================] - 11s 9ms/step - loss: 227.2674 - val_loss: 335.9984
Epoch 10/10
1250/1250 [==============================] - 10s 8ms/step - loss: 221.3728 - val_loss: 257.2447
```

Рисунок 31

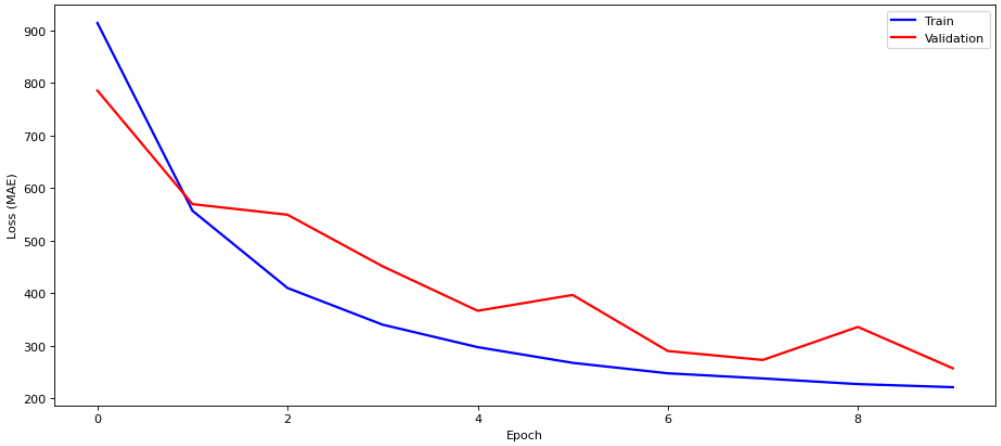

Рисунок 32

Наилучших результатов модель достигает с применением оптимизатора Adam.

Сравним результаты MAE (Mean Absolute Error) разных моделей:

```
simpleRNN + adam: 182.9746
LSTM + adam: 196.3822
GRU + adam: 204.248
```

Adam - эффективный метод оптимизации, который адаптирует скорость обучения для каждого параметра. Наилучший результат показала модель simpleRNN в силу особенностей исследуемой последовательности.

Настройка гиперпараметров

Random Search - метод оптимизации гиперпараметров, который заключается в случайном выборе значений гиперпараметров из заданного диапазона и оценке производительности модели с этими значениями. Это простой и эффективный способ настройки гиперпараметров модели машинного обучения.

Определяем параметры, создаем модель и запустим обучение (рис.33).

```
import sklearn
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test =  train_test_split(df, df['MW'], test_size=0.2, shuffle=False)
```

```
param_dist = {
'n_estimators': randint(50, 100),
'max_depth': [None, 10, 20],
'min_samples_leaf': [1, 2, 4]
}
```

```
rf_model = RandomForestRegressor()
random_search = RandomizedSearchCV(rf_model, param_distributions=param_dist, cv=5, verbose=2, n_jobs=-1)
```

```
random_search.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
▸ RandomizedSearchCV
▸ estimator: RandomForestRegressor
  ▸ RandomForestRegressor
```

Рисунок 33

Результаты обучения приведены на рис.34.

```
print("Best Hyperparameters:", random_search.best_params_)
print("Best Cross-Validation Score:", random_search.best_score_)
```

```
Best Hyperparameters: {'max_depth': 20, 'min_samples_leaf': 1, 'n_estimators': 96}
Best Cross-Validation Score: 0.9999975486511982
```

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
y_pred = random_search.predict(x_test)
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("Mean Absolute Error:", mae)
print("R^2 Score:", r2)
```

```
Mean Squared Error: 42.17927606336771
Mean Absolute Error: 0.47514791666666223
R^2 Score: 0.9999921739543299
```

MSE - метрика, которая сообщает среднеквадратичную разницу между прогнозируемыми значениями и фактическими значениями в наборе данных. R^2 оценивает, насколько хорошо регрессионная модель соответствует фактическим данным.

Получены результаты, сигнализирующие об удачном подборе гиперпараметров.

Ссылки на Google Colab:

https://colab.research.google.com/drive/18jK5_7Wj0PxNpRmDw5E9f1J3N7g5yyvs?usp=sharing

https://colab.research.google.com/drive/10sr9EDM2_p8rk0PHGVBibaaHmMCtMfgX?usp=sharing

Выводы

В результате выполнения работы было проведено исследование рекуррентных нейронных сетей. Построены различные архитектуры – RNN, GRU, LSTM, с применением различных оптимизаторов – Adam, SGD, Momentum, RMSprop. Изучены методы работы моделей и оптимизаторов, определены оптимальные значения гиперпараметров для данного набора данных.