Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Компьютерных сетей и систем

Кафедра Информатики

ЛАБОРАТОРНАЯ РАБОТА №8
«Рекуррентные нейронные сети для анализа временных рядов»

Магистрант:
гр. 956241
Шуба И.А.

Проверил:
Заливако С. С.

Минск, 2020

ХОД РАБОТЫ

Задание.

Данные: Набор данных для прогнозирования временных рядов, который состоит из среднемесячного числа пятен на солнце, наблюдаемых с января 1749 по август 2017. Данные в виде csv-файла можно скачать на сайте Kaggle -> https://www.kaggle.com/robervalt/sunspots/

Задание 1.

Загрузите данные. Изобразите ряд в виде графика. Вычислите основные характеристики временного ряда (сезонность, тренд, автокорреляцию).

Задание 2.

Для прогнозирования разделите временной ряд на обучающую, валидационную и контрольную выборки.

Задание 3.

Примените модель ARIMA для прогнозирования значений данного временного ряда.

Задание 4.

Повторите эксперимент по прогнозированию, реализовав рекуррентную нейронную сеть (с как минимум 2 рекуррентными слоями).

Задание 5.

Сравните качество прогноза моделей. Какой максимальный результат удалось получить на контрольной выборке?

Результат выполнения:

Задание 1. Загрузите данные. Изобразите ряд в виде графика. Вычислите основные характеристики временного ряда (сезонность, тренд, автокорреляцию).

Укажем путь и загрузим датасет.

```
path = 'sunspots/Sunspots.csv'
```

```
df = pd.read_csv(path, index_col=['Date'], usecols=['Date', 'Monthly Mean Total Sunspot Number'])
df.index = pd.to_datetime(df.index.values)
df_quarter = df.resample('Q').mean()
```

```
df.head()
```

Out[11]:

	Monthly Mean Total Sunspot Number
1749-01-31	96.7
1749-02-28	104.3
1749-03-31	116.7
1749-04-30	92.8
1749-05-31	141.7

Рисунок 1 – Загрузка данных

Отобразим временной ряд в виде графика.

```
plt.plot(pd.to_datetime(df.index.values), df.values)
plt.xlabel("Time (Years)")
plt.ylabel("Sunspots")
plt.title('Monthly Mean Total Sunspot Number')
plt.show()
```

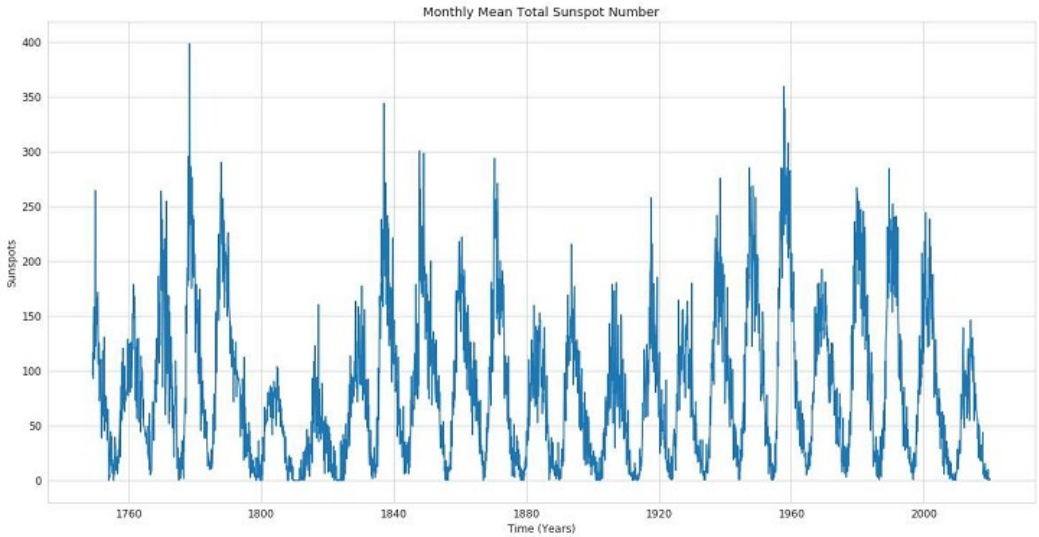


Рисунок 2 – Временной ряд в виде графика

Заметим, что явных пиков на рисунке 2 примерно 25, разделив весь период наблюдения на количество пиков можно приблизительно узнать период повторения. Таким образом 11x12 месяцев = 132 – получим период повторения пика в месяцах.

```
# 25 peaks
(df.index[-1].year - df.index[0].year) / 25 # Each 11 years repeat peak
```

```
Out[14]:

10.8
```

Рисунок 3 – Расчет сезонности

```
sm.tsa.seasonal_decompose(df['Monthly Mean Total Sunspot Number']).plot()
print("Критерий Дики-Фуллера: p=%f" % sm.tsa.stattools.adfuller(df['Monthly Mean Total
 Sunspot Number'])[1])
```

```
Критерий Дики-Фуллера: p=0.000000
```

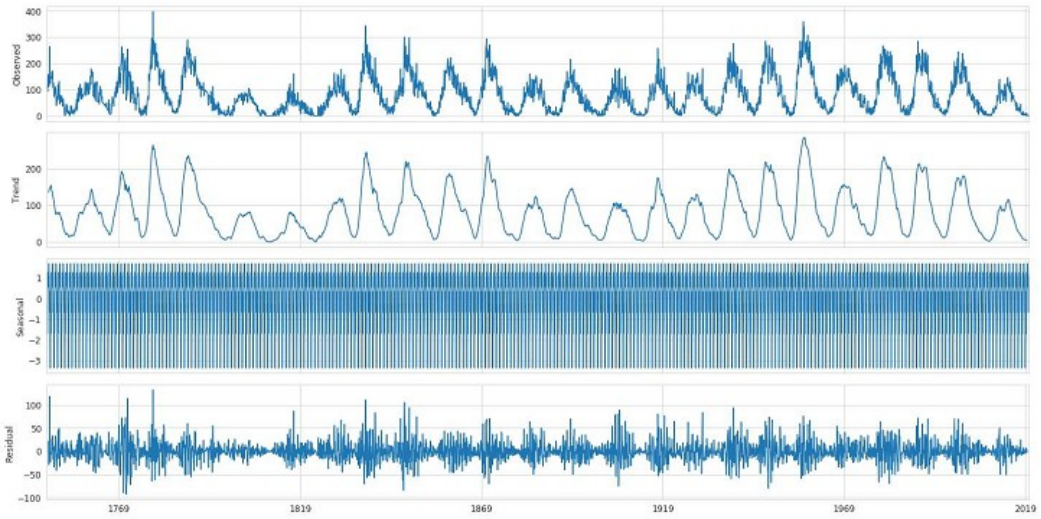


Рисунок 4 – Декомпозиция временного ряда

Построим графики автокорреляции и частичной автокорреляции. Укажем количество лагов 132, равное периоду повторения характерного пика, вычисленного выше.

```
plot_acf(df,lags=132, title='Sunspots ACF')
plot_pacf(df,lags=132, title='Sunspots PACF')
plt.show()
```

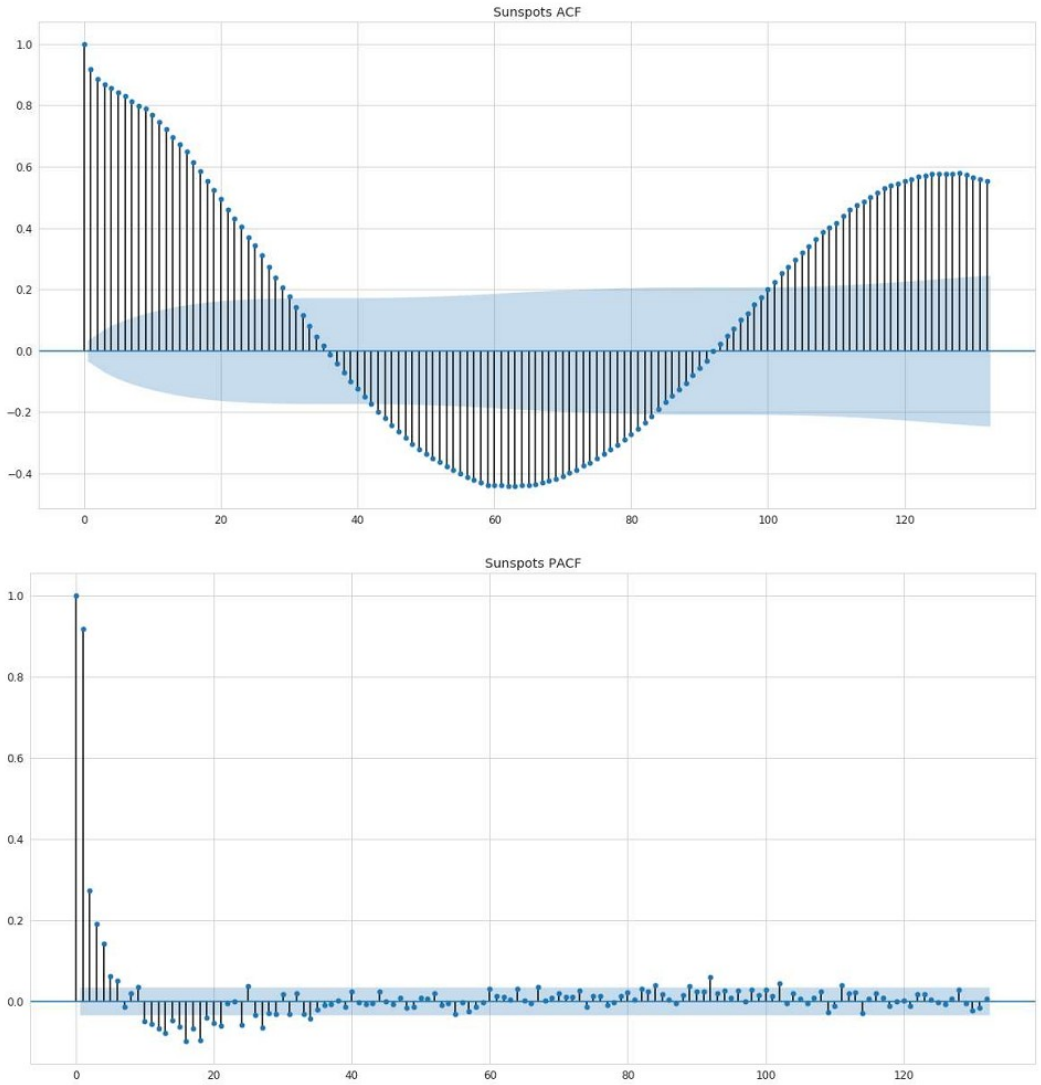


Рисунок 5 – Графики автокорреляции и частичной автокорреляции

Задание 2. Для прогнозирования разделите временной ряд на обучающую, валидационную и контрольную выборки.

```
train_data = data[:int(len(data)*0.8)]
val_data = data[int(len(data)*0.8):int(len(data)*0.9)]
test_data = data[int(len(data)*0.9):]
```

```
(len(train_data) + len(val_data) + len(test_data)) == len(data)
```

```
Out[24]:

True
```

```
print('Train', len(train_data))
print('Validation:', len(val_data))
print('Test', len(test_data))
```

```
Train 2601
Validation: 325
Test 326
```

Рисунок 6 – Разделение временного ряда

Задание 3. Примените модель ARIMA для прогнозирования значений данного временного ряда.

Многие методы и модели основаны на предположениях о стационарности ряда, но как было замечено ранее наш ряд таковым, скорее всего, не является. Поэтому для проверки стационарности проведем обобщенный тест Дикки-Фуллера на наличие единичных корней. Напишем функцию для определения стационарности ряда.

```
from statsmodels.tsa.stattools import adfuller
def printADFTest(serie):
    test = adfuller(serie, autolag='AIC')
    print('adf: ', test[0])
    print('p-value: ', test[1])
    print('Critical values: ', test[4])
    if test[0]> test[4]['5%']:
        print ('Есть единичные корни, ряд не стационарен')
    else:
        print('Единичных корней нет, ряд стационарен')
```

Рисунок 7 – Функция для определения стационарности ряда

Применим реализованную функцию к данному временному ряду.

```
#d = 0
printADFTest(df['Monthly Mean Total Sunspot Number'])
#d = 1
printADFTest(df['Monthly Mean Total Sunspot Number'].diff(1).dropna())
```

```
adf:  -10.48086843266985
p-value:  1.2147141586504414e-18
Critical values:  {'1%': -3.4323805665026175, '5%': -2.8624371819849372,
'10%': -2.5672475896829807}
Единичных корней нет, ряд стационарен
adf:  -9.091685375801628
p-value:  3.8092149110860715e-15
Critical values:  {'1%': -3.432381197225056, '5%': -2.8624374605672154, '1
0%': -2.567247737994645}
Единичных корней нет, ряд стационарен
```

Рисунок 8 – Вывод функции по определению стационарности ряда

В обоих случаях ряд является стационарным, поэтому можно пробовать подбирать параметры для модели ARIMA.

Для прогнозирования временного ряда с очевидной сезонностью будем использовать модель SARIMA(Seasonal Autoregressive Integrated Moving Average).

Определим параметры модели.

S = 132 – количество точек (месяцев) в периоде

D = 1 – у ряда есть устойчивая сезонная составляющая во времени.

По графику частичной автокорреляции (PACF) определяем p – первый лаг, когда значение входит в уровень значимости, таким образом, **p = 7**.

По графику автокорреляции (ACF) определяем q – первый лаг, когда значение входит в уровень значимости, таким образом, **q = 26**. Также по этому графику определяем P и Q. Если значение ACF на 132 лаге, то **P = 1**, а **Q = 0**, при этом должно выполняться условие $P+Q \leq 2$.

Построим модель SARIMA.

```
model = sm.tsa.statespace.SARIMAX(train_data, trend='n', order=(7,0,26), seasonal_order
=(1,1,0,11*12))
results = model.fit()
print(results.summary())
```

Рисунок 9 – Обучение модели SARIMA

```
val_pred = results.predict(val_data.index[0], val_data.index[-1])
test_pred = results.predict(test_data.index[0], test_data.index[-1])
print('Mean Absolute Error on validation data:', mean_absolute_error(val_data, val_pred
))
print('Mean Absolute Error on test data:', mean_absolute_error(test_data, test_pred))
```

```
Mean Absolute Error on validation data: 37.57887990932546
Mean Absolute Error on test data: 63.50573112993852
```

Рисунок 10 – Результаты прогнозирования

Метрика Mean Absolute Error на валидационных данных несколько выше, чем на контрольных. Построим график временного ряда для валидационных и контрольных данных.

```
plt.plot(train_data, label='Train data')
plt.plot(val_data, label='Real data', linewidth=5, alpha=0.4)
plt.plot(val_pred, label='Prediction data', linewidth=5, alpha=0.6)
plt.vlines(val_data.index[0], train_data.values.min(), train_data.values.max(), 'red',
'--')
plt.xlabel("Time (Years)")
plt.ylabel("Sunspots")
plt.title('Prediction on validation data')
plt.legend()
```

```
Out[31]:

<matplotlib.legend.Legend at 0x7f6f9d7eafd0>
```

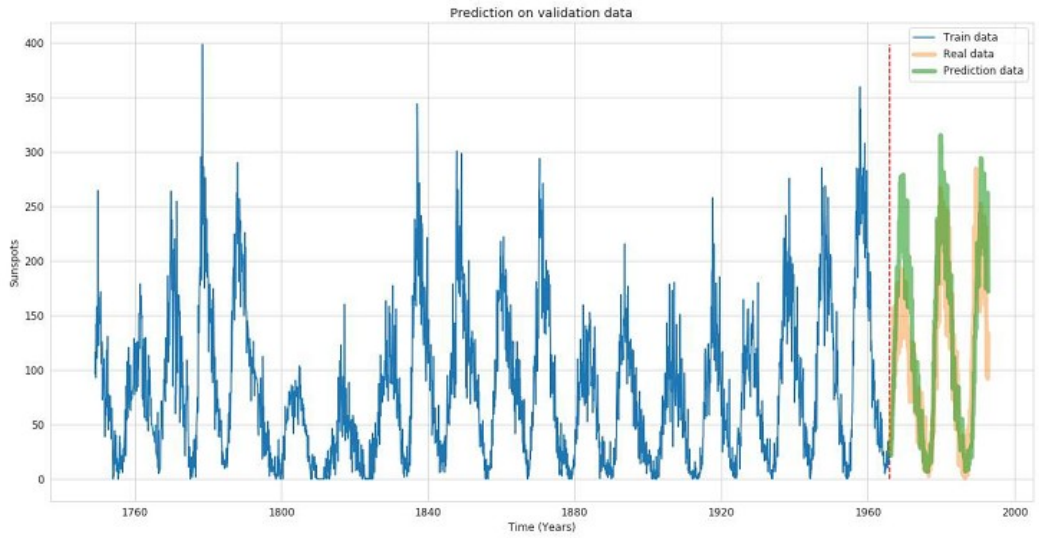


Рисунок 11 – Предсказание на валидационных данных

```
plt.plot(train_data.append(val_data), label='Train data')
plt.plot(test_data, label='Real data', linewidth=5, alpha=0.4)
plt.plot(test_pred, label='Prediction data', linewidth=5, alpha=0.6)
plt.vlines(test_data.index[0], train_data.values.min(), train_data.values.max(), 'red',
'--')
plt.xlabel("Time (Years)")
plt.ylabel("Sunspots")
plt.title('Prediction on test data')
plt.legend()
```

```
Out[32]:

<matplotlib.legend.Legend at 0x7f6f9e20cf90>
```

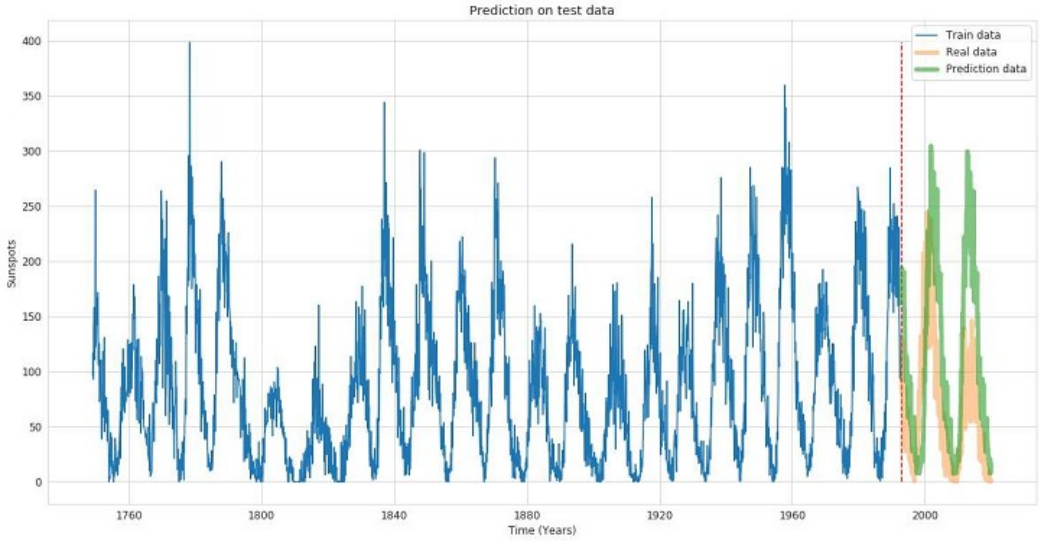


Рисунок 12 – Предсказание на контрольных данных

Задание 4. Повторите эксперимент по прогнозированию, реализовав рекуррентную нейронную сеть (с как минимум 2 рекуррентными слоями).

Преобразуем данные для подачи их в нейронную сеть.

```
y = df['Monthly Mean Total Sunspot Number']
x = df.index.values
```

```
scaler = MinMaxScaler()
```

```
mms_y = scaler.fit_transform(y.values.reshape(-1,1))
```

```
train_data = mms_y[:int(len(mms_y)*0.8)]
val_data = mms_y[int(len(mms_y)*0.8):int(len(mms_y)*0.9)]
test_data = mms_y[int(len(mms_y)*0.9):]
```

Рисунок 13 – Разделение и нормализация исходного датасета

```python
def simple_look(dataset, look_back=1):
    dataX = dataset[:-(look_back + 1), 0]
    dataY = dataset[look_back:-1, 0]
    return np.reshape(dataX, (len(dataX), 1)), np.reshape(dataY, (len(dataY), 1))
```

```python
#reshape into X=t and Y=t+1
look_back = 1
trainXb, trainYb = simple_look(train_data, look_back)
valXb, valYb = simple_look(val_data, look_back)
testXb, testYb = simple_look(test_data, look_back)
```

```python
#reshape input to be [samples, time steps, features]
trainX = np.reshape(trainXb, (trainXb.shape[0], 1, trainXb.shape[1]))
valX = np.reshape(valXb, (valXb.shape[0], 1, valXb.shape[1]))
testX = np.reshape(testXb, (testXb.shape[0], 1, testXb.shape[1]))
```

Рисунок 14 – Изменение формы данных

```python
model = Sequential()
model.add(LSTM(256, input_shape=(1, look_back), return_sequences=True))
model.add(LSTM(128))
model.add(Dense(1))
keras2ascii(model)
```

Рисунок 15 – Реализация рекуррентной нейронной сети

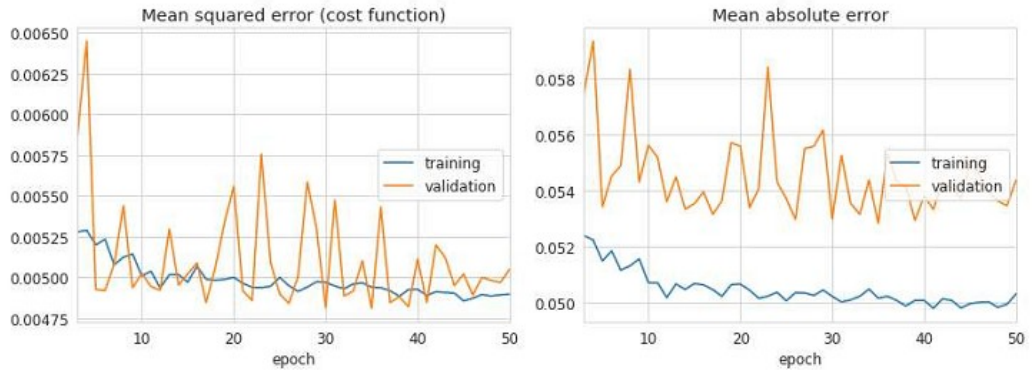
```
 OPERATION          DATA DIMENSIONS   WEIGHTS(N)   WEIGHTS(%)

     Input   #####      1    1
      LSTM   LLLLL -------------------    264192     57.3%
      tanh   #####      1  256
      LSTM   LLLLL -------------------    197120     42.7%
      tanh   #####         128
     Dense   XXXXX -------------------       129      0.0%
             #####           1
```

Рисунок 16 – Схема рекуррентной нейронной сети

```
model.compile(loss='mean_squared_error', optimizer='adam',  metrics=['mae'])
```

```
model.fit(trainX,
          trainYb,
          epochs=50,
          validation_data=(valX, valYb),
          callbacks=[PlotLossesKeras()],
          batch_size=1,
          verbose=2)
```




```
Mean squared error (cost function):
training   (min:    0.005, max:    0.006, cur:    0.005)
validation (min:    0.005, max:    0.007, cur:    0.005)

Mean absolute error:
training   (min:    0.050, max:    0.057, cur:    0.050)
validation (min:    0.053, max:    0.067, cur:    0.054)
```

Рисунок 17 – Обучение рекуррентной нейронной сети

Из графиков видно, что существует достаточно большой разброс в метрике MSE, что присуще рекуррентным сетям, однако еще можно заметить, что ниспадающего тренда при обучении уже, приблизительно, после 10 эпохи нет, что означает, что для получения примерно такого же результата точности модели, хватилось бы приблизительно 10 эпох.

```
trainPredict = model.predict(trainX)
valPredict = model.predict(valX)
testPredict = model.predict(testX)

trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform(trainYb)

valPredict = scaler.inverse_transform(valPredict)
valY = scaler.inverse_transform(valYb)

testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform(testYb)

valScore = mean_absolute_error(valY, valPredict)
print('Val Score: %.2f MSE' % (valScore))
testScore = mean_absolute_error(testY, testPredict)
print('Test Score: %.2f MSE' % (testScore))
```

```
Val Score: 21.66 MSE
Test Score: 15.95 MSE
```

Рисунок 18 – Результаты модели на валидационной и контрольной выборке

```
val_pred_df = pd.DataFrame(valPredict)
val_pred_df.index = val_data.index[2:]
test_pred_df = pd.DataFrame(testPredict)
test_pred_df.index = test_data.index[2:]
```

```
plt.plot(train_data.index, train_data.values, label='Train data')
plt.vlines(train_data.index[-1], train_data.values.min(), train_data.values.max(), 'red', '--')
plt.plot(val_data.index, val_data.values, label='Real data')
plt.plot(val_pred_df, label='Prediction data', linewidth=5, alpha=0.6)
plt.xlabel("Time (Years)")
plt.ylabel("Sunspots")
plt.title('Prediction on validation data')
plt.legend()
```

```
Out[84]:

<matplotlib.legend.Legend at 0x7fa763359250>
```

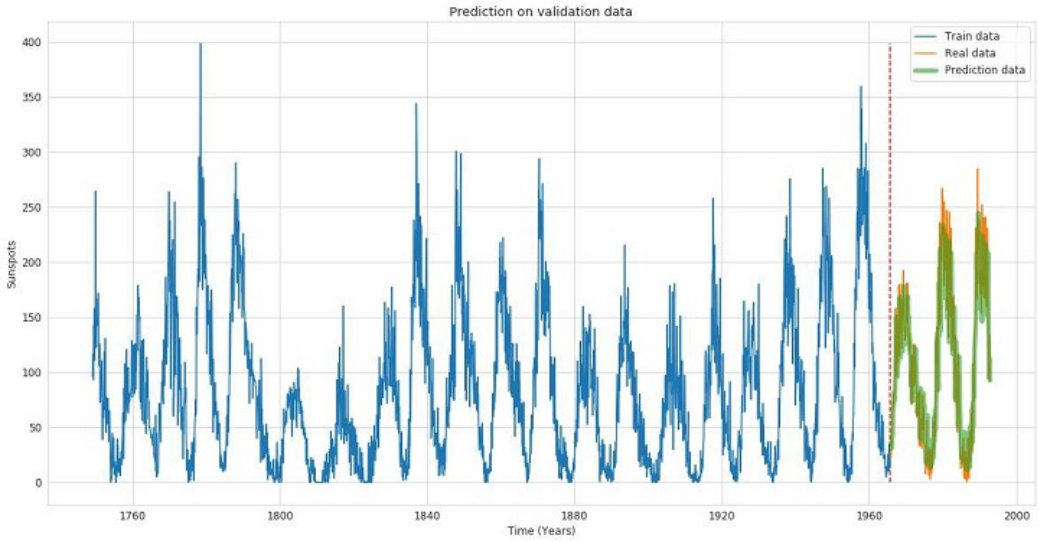


Рисунок 19 – Предсказание на валидационных данных

В целом, с помощью нейронной сети получился результат намного лучше, нежели с подбором параметров для модели SARIMA. Также можно было бы написать автоматический перебор параметров для SARIMA модели. А лучшую модель можно было бы выбрать с помощью наименьшего информационного критерия Акаике (AIC). Однако данный процесс занял бы очень много времени, так как по графику автокорреляции q составило 26.

```
plt.plot(train_data.append(val_data), label='Train data')
plt.vlines(test_data.index[0], train_data.values.min(), train_data.values.max(), 'red',
'--')
plt.plot(test_data.index, test_data.values, label='Real data')
plt.plot(test_pred_df, label='Prediction data', linewidth=5, alpha=0.6)
plt.xlabel("Time (Years)")
plt.ylabel("Sunspots")
plt.title('Prediction on test data')
plt.legend()
```

```
Out[85]:

<matplotlib.legend.Legend at 0x7fa7632c9b50>
```

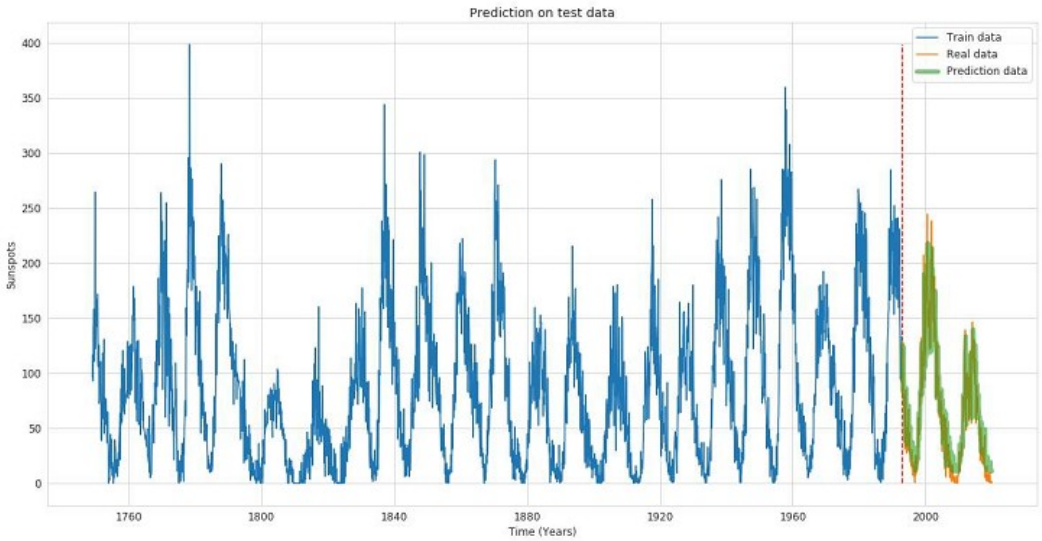


Рисунок 20 – Предсказание на контрольных данных

Вывод:

В ходе выполнения лабораторной работы были изучены подходы для работы с временными рядами. Были построены графики декомпозиции, автокорреляции, частичной корреляции для исходного набора данных. Были подобраны оптимальные параметры для модели SARIMA. В результате метрика MAE для контрольной выборки составила 63, что на самом деле еще является оптимальной ошибкой, так как значение было получено суммой числа пятен за 326/12 = 27 лет. Также были изучены рекуррентные нейронные сети. Была написана простейшая архитектура, состоящая из 2 рекуррентных слоев. В результате метрика MAE для контрольной выборки составила 15,95, что практически в 4 раза меньше ошибки полученной с помощью модели SARIMA.