Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра информатики

|  |
| --- |
|  |

**ЛабоРАТОРНАЯ РАБОТА №8**

«Рекуррентные нейронные сети для анализа временных рядов»

|  |  |  |
| --- | --- | --- |
| Студент |  | К. Д. Зюсько |
| Преподаватель |  | М. В. Стержанов |

Минск 2020

ХОД РАБОТЫ

**Задание.**

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/

1. Загрузите данные. Изобразите ряд в виде графика. Вычислите основные характеристики временного ряда (сезонность, тренд, автокорреляцию).
2. Для прогнозирования разделите временной ряд на обучающую, валидационную и контрольную выборки.
3. Примените модель ARIMA для прогнозирования значений данного временного ряда.
4. Повторите эксперимент по прогнозированию, реализовав рекуррентную нейронную сеть (с как минимум 2 рекуррентными слоями).
5. Сравните качество прогноза моделей.

**Результат выполнения:**

1. Код реализации:

!mkdir ~/.kaggle

!cp /content/.kaggle/kaggle.json ~/.kaggle/kaggle.json

!kaggle datasets download robervalt/sunspots -p robervalt/sunspots

!unzip robervalt/sunspots/sunspots.zip -d data

import pandas as pd

DATA\_PATH = './data/Sunspots.csv'

VALUES = 'Monthly Mean Total Sunspot Number'

def get\_dataset():

df = pd.read\_csv(DATA\_PATH)

df.drop(['Unnamed: 0'], axis=1, inplace=True)

return df

df = get\_dataset()

df.head()

import matplotlib.pyplot as plt

plt.rcParams['figure.figsize'] = [30, 13]

df\_plot = get\_dataset()

df\_plot['Date']= pd.to\_datetime(df\_plot['Date'])

df\_plot.plot(x = 'Date', y = VALUES)

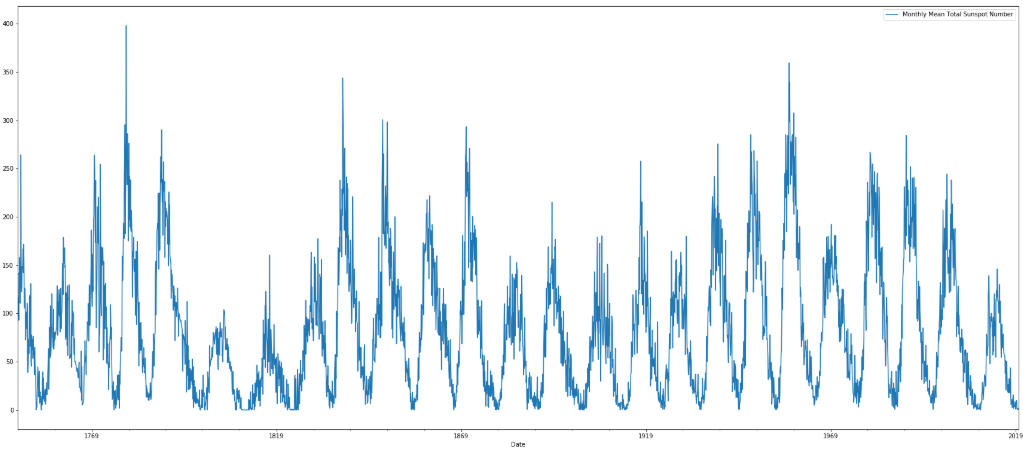


Рисунок 1 – визуализация исходного ряда

# autocorrelation

from pandas.plotting import autocorrelation\_plot

plt.rcParams['figure.figsize'] = [7, 3]

series = get\_dataset()

autocorrelation\_plot(series[VALUES])

plt.show()

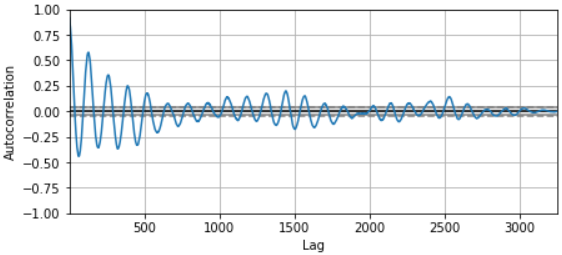


Рисунок 2 – график автокорреляции

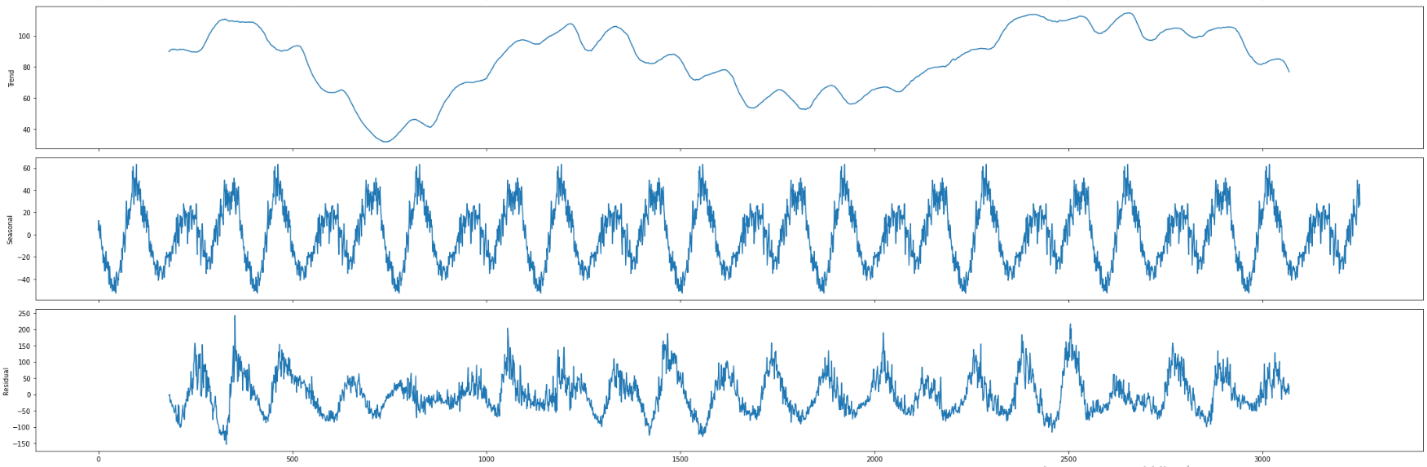


Рисунок 3 – тренд, сезонность, остаток

2. Код реализации:

def ts\_split(data,percentage=0.7):

n = int(data.shape[0]\*percentage)

return data[:n], data[n+1:]

train, test = ts\_split(df)

train, val = ts\_split(train)

3. Код реализации:

from pandas import read\_csv

from pandas import datetime

from matplotlib import pyplot

from statsmodels.tsa.arima\_model import ARIMA

from sklearn.metrics import mean\_squared\_error

arima = ARIMA(train[VALUES], (6,0,1), dates=train['Date']).fit()

# print summary of fit model

print(arima.summary())

history = [x for x in train[VALUES]]

tst = [x for x in test[VALUES]]

predictions = list()

for t in range(len(test[VALUES])):

model = ARIMA(history, order=(6,0,1))

model\_fit = model.fit(disp=0)

output = model\_fit.forecast()

yhat = output[0]

predictions.append(yhat)

obs = tst[t]

history.append(obs)

print('predicted=%f, expected=%f' % (yhat, obs))

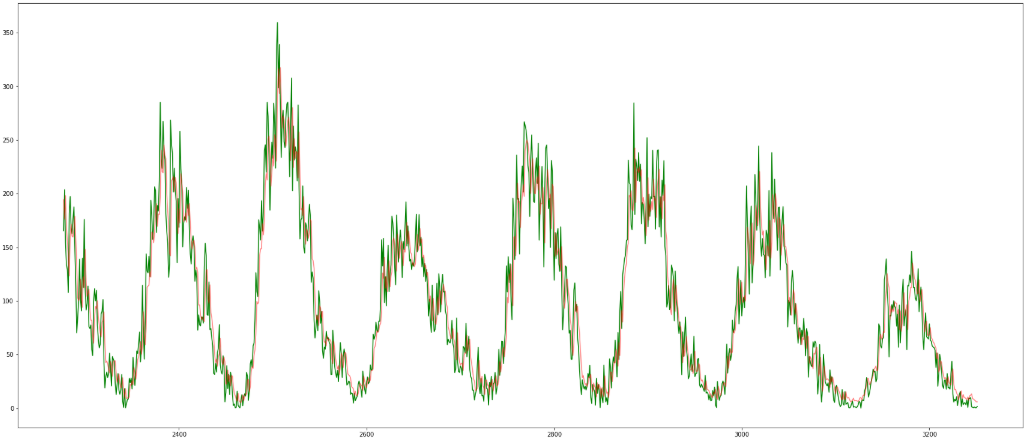


Рисунок 4 – предсказанные значения ARIMA

4. Код реализации:

# univariate data preparation

from numpy import array

# split a univariate sequence into samples

def split\_sequence(sequence, n\_steps):

X, y = list(), list()

for i in range(len(sequence)):

# find the end of this pattern

end\_ix = i + n\_steps

# check if we are beyond the sequence

if end\_ix > len(sequence)-1:

break

# gather input and output parts of the pattern

seq\_x, seq\_y = sequence[i:end\_ix], sequence[end\_ix]

X.append(seq\_x)

y.append(seq\_y)

return array(X), array(y)

from keras.models import Sequential

from keras.layers import LSTM

from keras.layers import Dense

n\_features = 1

# choose a number of time steps

n\_steps = 14

model = Sequential()

model.add(LSTM(50, activation='relu', return\_sequences=True, input\_shape=(n\_steps, n\_features)))

model.add(LSTM(50, activation='relu'))

model.add(Dense(30, activation="relu"))

model.add(Dense(10, activation="relu"))

model.add(Dense(1))

model.compile(optimizer='adam', loss='mse')

model.save\_weights('model.h5')

ds = get\_dataset()

train, test = ts\_split(ds)

# split into samples

X, y = split\_sequence(train[VALUES], n\_steps)

# reshape from [samples, timesteps] into [samples, timesteps, features]

X = X.reshape((X.shape[0], X.shape[1], n\_features))

# reset weights

model.load\_weights('model.h5')

# fit model

model.fit(X, y, epochs=30, verbose=1)

# demonstrate prediction

x\_input = X[0]

x\_input = x\_input.reshape((1, n\_steps, n\_features))

yhat = model.predict(x\_input, verbose=0)

print(yhat, y[0])

import numpy as np

def predict(last\_n\_steps, replaceable):

dfc = pd.DataFrame([], columns=('Date', VALUES))

last\_values = last\_n\_steps.to\_numpy()

n = replaceable.first\_valid\_index()

for index, row in replaceable.iterrows():

cp = row[VALUES]

row[VALUES] = model.predict(last\_values.reshape((1, n\_steps, n\_features)), verbose=0)[0][0]

last\_values = np.append(last\_values, [cp])[-n\_steps:]

dfc = dfc.append(row)

return dfc

predictions = predict(train[VALUES][:n\_steps], test)

import matplotlib.pyplot as plt

plt.rcParams['figure.figsize'] = [30, 13]

train, test = ts\_split(ds)

plt.plot(test[VALUES], color='green')

plt.plot(predictions[VALUES], color='red', alpha=0.5)

plt.show()

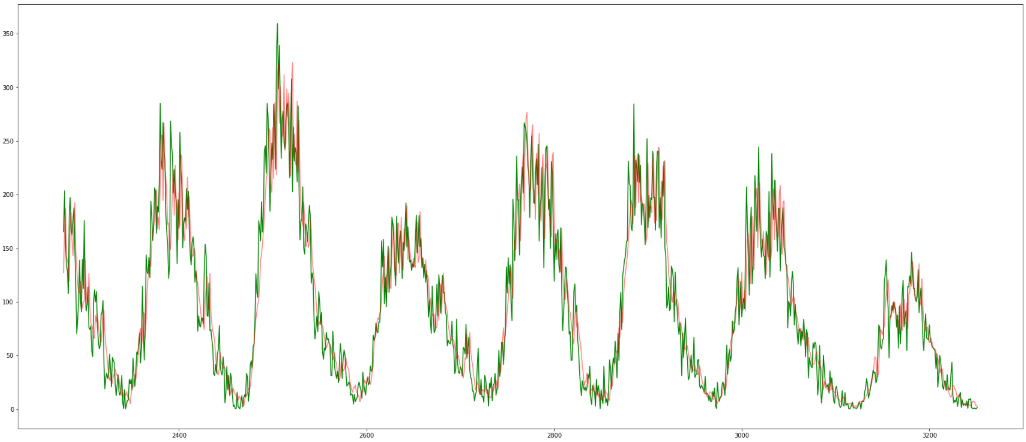


Рисунок 5 – график предсказаний на тестовой выборке

1. Результаты прогнозирования нейронной сети получились практически такими же, как и у ARIMA (611 vs 610 MSE). Однако нейронная сеть обучается намного быстрее. Поэтому можно сделать вывод, что нейронная сеть работает лучше.