# Лабораторная работа №8

## Рекуррентные нейронные сети для анализа временных рядов

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/ (https://www.kaggle.com/robervalt/sunspots/)

### Задание 1

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

In [1]:

```
from google.colab import drive

drive.mount('/content/drive', force_remount = True)
```

```
Mounted at /content/drive
```

In [0]:

```
BASE_DIR = '/content/drive/My Drive/Colab Files/mo-2'

import sys

sys.path.append(BASE_DIR)

import os
```

In [0]:

```
DATA_ARCHIVE_NAME = 'sunspots.zip'

LOCAL_DIR_NAME = 'sunspots'
```

In [0]:

```
from zipfile import ZipFile

with ZipFile(os.path.join(BASE_DIR, DATA_ARCHIVE_NAME), 'r') as zip_:
    zip_.extractall(LOCAL_DIR_NAME)
```

In [0]:

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

In [0]:

```
import pandas as pd

all_df = pd.read_csv(DATA_FILE_PATH, parse_dates = ['Date'], index_col = 'Date')
```

In [7]:

```
print(all_df.shape)
```

```
(3252, 2)
```

In [8]:

```
all_df.keys()
```

Out[8]:

```
Index(['Unnamed: 0', 'Monthly Mean Total Sunspot Number'], dtype='object')
```

In [9]:

```
from statsmodels.tsa.seasonal import seasonal_decompose

additive = seasonal_decompose(all_df['Monthly Mean Total Sunspot Number'], model = 'additiv
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: Fut
ureWarning: pandas.util.testing is deprecated. Use the functions in the publ
ic API at pandas.testing instead.
  import pandas.util.testing as tm
```

In [0]:

```
%matplotlib inline

import matplotlib.pyplot as plt
```

In [0]:

```
import seaborn as sns

from matplotlib import rcParams

rcParams['figure.figsize'] = 11.7, 8.27

sns.set()

sns.set_palette(sns.color_palette('hls', 8))
```

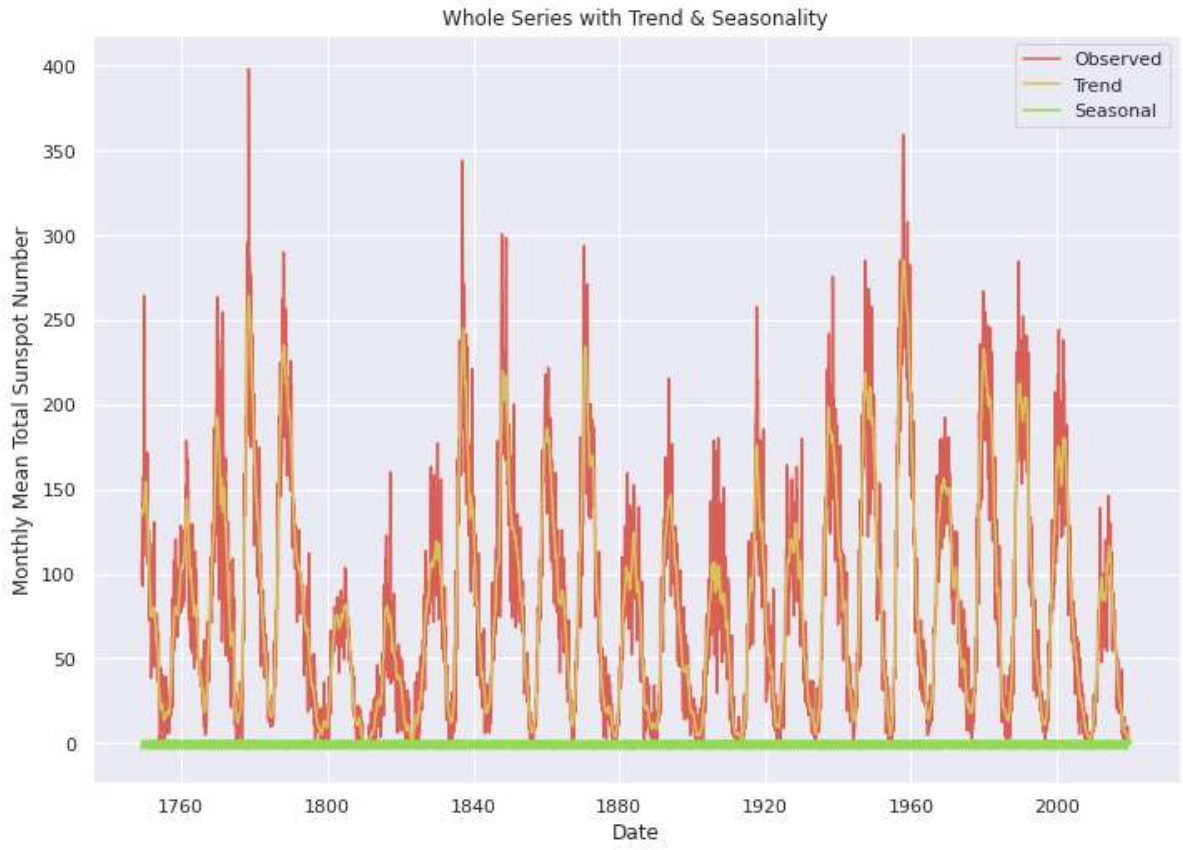
In [12]:

```
sns.lineplot(data = additive.observed, label = 'Observed')
sns.lineplot(data = additive.trend, label = 'Trend')
sns.lineplot(data = additive.seasonal, label = 'Seasonal')

plt.xlabel('Date')
plt.ylabel('Monthly Mean Total Sunspot Number')

plt.title('Whole Series with Trend & Seasonality')

plt.show()
```




Рассмотрим подробнее на небольшом промежутке:

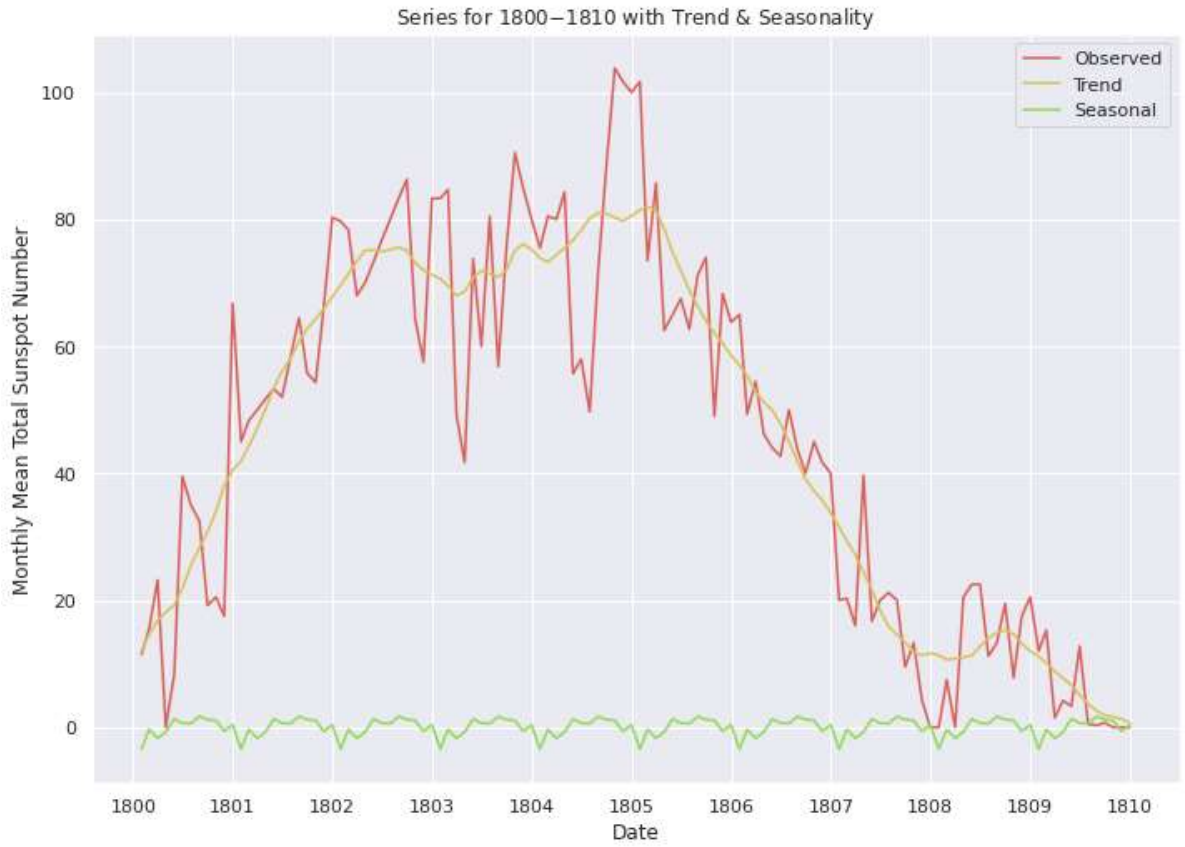
In [13]:

```
sns.lineplot(data = additive.observed['1800-01-01':'1810-01-01'], label = 'Observed')
sns.lineplot(data = additive.trend['1800-01-01':'1810-01-01'], label = 'Trend')
sns.lineplot(data = additive.seasonal['1800-01-01':'1810-01-01'], label = 'Seasonal')

plt.xlabel('Date')
plt.ylabel('Monthly Mean Total Sunspot Number')

plt.title('Series for 1800$-$1810 with Trend & Seasonality')

plt.show()
```



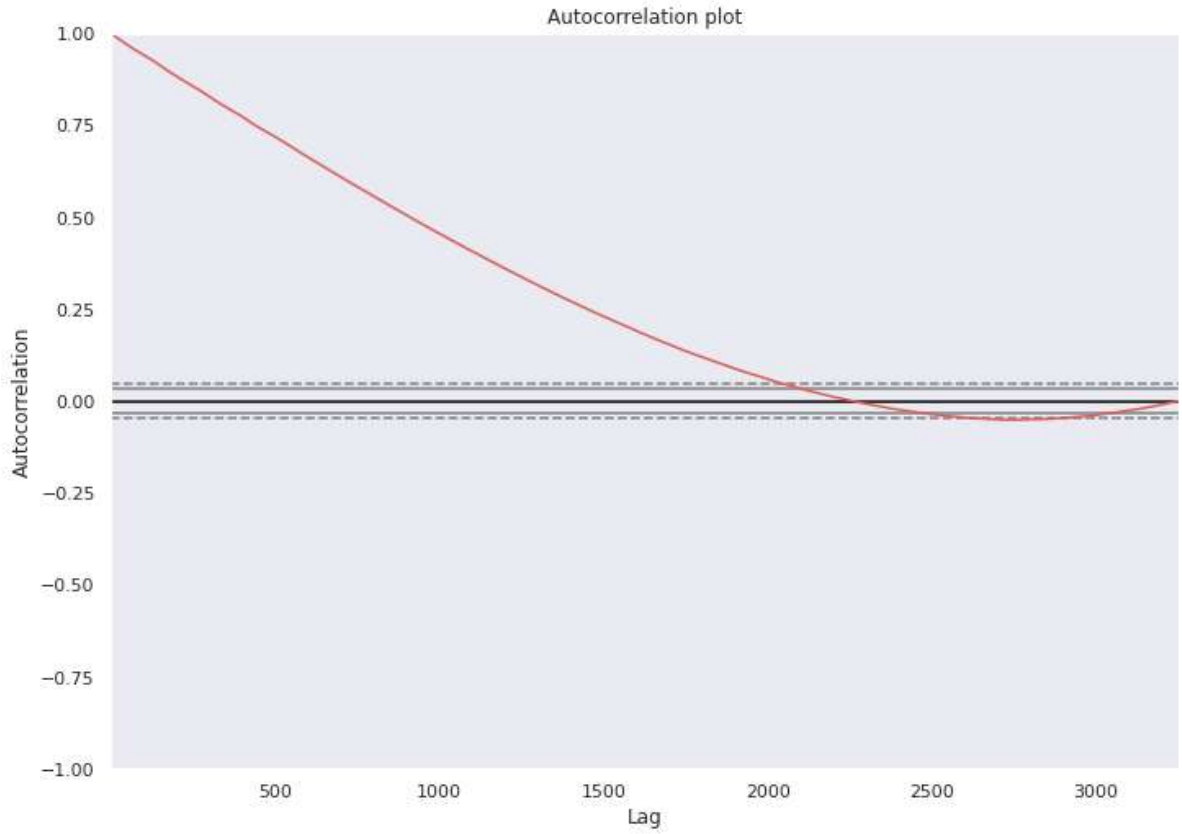
In [14]:

```
from pandas.plotting import autocorrelation_plot

autocorrelation_plot(all_df.values.tolist())

plt.title('Autocorrelation plot')

plt.show()
```




## Задание 2

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

Этот шаг будет применён автоматически как параметр `validation_split` метода `model.fit()` .

## Задание 3

Примените модель *ARIMA* для прогнозирования значений данного временного ряда.

In [15]:

```
! pip install pmdarima

! pip show pmdarima
```

```
Requirement already satisfied: pmdarima in /usr/local/lib/python3.6/dist-pac
kages (1.5.3)
Requirement already satisfied: Cython>=0.29 in /usr/local/lib/python3.6/dist
-packages (from pmdarima) (0.29.16)
Requirement already satisfied: pandas>=0.19 in /usr/local/lib/python3.6/dist
-packages (from pmdarima) (1.0.3)
Requirement already satisfied: scikit-learn>=0.22 in /usr/local/lib/python3.
6/dist-packages (from pmdarima) (0.22.2.post1)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.6/dis
t-packages (from pmdarima) (1.18.2)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.6/dist
-packages (from pmdarima) (1.4.1)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.6/dist
-packages (from pmdarima) (0.14.1)
Requirement already satisfied: statsmodels>=0.10.2 in /usr/local/lib/python
3.6/dist-packages (from pmdarima) (0.10.2)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.6/dist-pack
ages (from pmdarima) (1.24.3)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/dist
-packages (from pandas>=0.19->pmdarima) (2018.9)
Requirement already satisfied: python-dateutil>=2.6.1 in /usr/local/lib/pyth
on3.6/dist-packages (from pandas>=0.19->pmdarima) (2.8.1)
Requirement already satisfied: patsy>=0.4.0 in /usr/local/lib/python3.6/dist
-packages (from statsmodels>=0.10.2->pmdarima) (0.5.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.6/dist-pac
kages (from python-dateutil>=2.6.1->pandas>=0.19->pmdarima) (1.12.0)
Name: pmdarima
Version: 1.5.3
Summary: Python's forecast::auto.arima equivalent
Home-page: http://alkaline-ml.com/pmdarima (http://alkaline-ml.com/pmdarima)
Author: None
Author-email: None
License: MIT
Location: /usr/local/lib/python3.6/dist-packages
Requires: joblib, Cython, pandas, urllib3, statsmodels, numpy, scikit-learn,
scipy
Required-by:
```

In [0]:

```
test_period = 6 * 12
```

In [17]:

```
from pmdarima.arima import auto_arima

arima = auto_arima(all_df['Monthly Mean Total Sunspot Number'][:-test_period],
                   trace = True, error_action = 'ignore', suppress_warnings = True, seasona
```

```
Performing stepwise search to minimize aic
Fit ARIMA: (2, 0, 2)x(1, 0, 1, 12) (constant=True); AIC=29588.066, BIC=2963
6.583, Time=23.372 seconds
Fit ARIMA: (0, 0, 0)x(0, 0, 0, 12) (constant=True); AIC=35872.649, BIC=3588
4.778, Time=0.065 seconds
Fit ARIMA: (1, 0, 0)x(1, 0, 0, 12) (constant=True); AIC=30025.545, BIC=3004
9.804, Time=5.185 seconds
Fit ARIMA: (0, 0, 1)x(0, 0, 1, 12) (constant=True); AIC=32380.500, BIC=3240
4.758, Time=5.341 seconds
Fit ARIMA: (0, 0, 0)x(0, 0, 0, 12) (constant=False); AIC=38764.997, BIC=3877
1.062, Time=0.049 seconds
Fit ARIMA: (2, 0, 2)x(0, 0, 1, 12) (constant=True); AIC=29586.807, BIC=2962
9.260, Time=13.302 seconds
Fit ARIMA: (2, 0, 2)x(0, 0, 0, 12) (constant=True); AIC=29595.958, BIC=2963
2.345, Time=2.032 seconds
Fit ARIMA: (2, 0, 2)x(0, 0, 2, 12) (constant=True); AIC=29587.374, BIC=2963
5.891, Time=45.942 seconds
Fit ARIMA: (2, 0, 2)x(1, 0, 0, 12) (constant=True); AIC=29587.365, BIC=2962
9.817, Time=16.549 seconds
Fit ARIMA: (2, 0, 2)x(1, 0, 2, 12) (constant=True); AIC=29582.451, BIC=2963
7.033, Time=67.935 seconds
Fit ARIMA: (2, 0, 2)x(2, 0, 2, 12) (constant=True); AIC=29552.494, BIC=2961
3.140, Time=71.844 seconds
Near non-invertible roots for order (2, 0, 2)(2, 0, 2, 12); setting score to
inf (at least one inverse root too close to the border of the unit circle:
0.993)
Fit ARIMA: (2, 0, 2)x(2, 0, 1, 12) (constant=True); AIC=29581.341, BIC=2963
5.923, Time=64.230 seconds
Fit ARIMA: (2, 0, 2)x(2, 0, 0, 12) (constant=True); AIC=29587.014, BIC=2963
5.531, Time=52.283 seconds
Fit ARIMA: (1, 0, 2)x(2, 0, 1, 12) (constant=True); AIC=29579.106, BIC=2962
7.623, Time=43.009 seconds
Fit ARIMA: (1, 0, 2)x(1, 0, 1, 12) (constant=True); AIC=29586.021, BIC=2962
8.473, Time=18.368 seconds
Fit ARIMA: (1, 0, 2)x(2, 0, 0, 12) (constant=True); AIC=29585.061, BIC=2962
7.513, Time=39.560 seconds
Fit ARIMA: (1, 0, 2)x(2, 0, 2, 12) (constant=True); AIC=29551.544, BIC=2960
6.126, Time=72.061 seconds
Near non-invertible roots for order (1, 0, 2)(2, 0, 2, 12); setting score to
inf (at least one inverse root too close to the border of the unit circle:
0.995)
Fit ARIMA: (1, 0, 2)x(1, 0, 0, 12) (constant=True); AIC=29585.380, BIC=2962
1.768, Time=8.778 seconds
Fit ARIMA: (1, 0, 2)x(1, 0, 2, 12) (constant=True); AIC=29580.389, BIC=2962
8.906, Time=49.857 seconds
Fit ARIMA: (0, 0, 2)x(2, 0, 1, 12) (constant=True); AIC=31382.620, BIC=3142
5.073, Time=33.173 seconds
Near non-invertible roots for order (0, 0, 2)(2, 0, 1, 12); setting score to
inf (at least one inverse root too close to the border of the unit circle:
1.000)
Fit ARIMA: (1, 0, 1)x(2, 0, 1, 12) (constant=True); AIC=29629.733, BIC=2967
2.185, Time=54.417 seconds
Fit ARIMA: (1, 0, 3)x(2, 0, 1, 12) (constant=True); AIC=29580.861, BIC=2963
5.443, Time=48.949 seconds
```

```
Fit ARIMA: (0, 0, 1)x(2, 0, 1, 12) (constant=True); AIC=32043.176, BIC=3207
9.563, Time=28.353 seconds
Near non-invertible roots for order (0, 0, 1)(2, 0, 1, 12); setting score to
inf (at least one inverse root too close to the border of the unit circle:
1.000)
Fit ARIMA: (0, 0, 3)x(2, 0, 1, 12) (constant=True); AIC=31074.278, BIC=3112
2.795, Time=42.232 seconds
Near non-invertible roots for order (0, 0, 3)(2, 0, 1, 12); setting score to
inf (at least one inverse root too close to the border of the unit circle:
1.000)
Fit ARIMA: (2, 0, 1)x(2, 0, 1, 12) (constant=True); AIC=29592.648, BIC=2964
1.165, Time=51.681 seconds
Fit ARIMA: (2, 0, 3)x(2, 0, 1, 12) (constant=True); AIC=29579.013, BIC=2963
9.659, Time=62.712 seconds
Fit ARIMA: (2, 0, 3)x(1, 0, 1, 12) (constant=True); AIC=29586.998, BIC=2964
1.579, Time=22.034 seconds
Fit ARIMA: (2, 0, 3)x(2, 0, 0, 12) (constant=True); AIC=29585.838, BIC=2964
0.420, Time=60.607 seconds
Fit ARIMA: (2, 0, 3)x(2, 0, 2, 12) (constant=True); AIC=29543.930, BIC=2961
0.641, Time=80.082 seconds
Near non-invertible roots for order (2, 0, 3)(2, 0, 2, 12); setting score to
inf (at least one inverse root too close to the border of the unit circle:
0.995)
Fit ARIMA: (2, 0, 3)x(1, 0, 0, 12) (constant=True); AIC=29586.591, BIC=2963
5.108, Time=17.100 seconds
Fit ARIMA: (2, 0, 3)x(1, 0, 2, 12) (constant=True); AIC=29580.547, BIC=2964
1.194, Time=72.056 seconds
Fit ARIMA: (3, 0, 3)x(2, 0, 1, 12) (constant=True); AIC=29578.087, BIC=2964
4.798, Time=79.448 seconds
Fit ARIMA: (3, 0, 3)x(1, 0, 1, 12) (constant=True); AIC=29586.766, BIC=2964
7.413, Time=22.877 seconds
Fit ARIMA: (3, 0, 3)x(2, 0, 0, 12) (constant=True); AIC=29584.745, BIC=2964
5.391, Time=79.798 seconds
Fit ARIMA: (3, 0, 3)x(2, 0, 2, 12) (constant=True); AIC=29546.099, BIC=2961
8.874, Time=82.080 seconds
Near non-invertible roots for order (3, 0, 3)(2, 0, 2, 12); setting score to
inf (at least one inverse root too close to the border of the unit circle:
0.995)
Fit ARIMA: (3, 0, 3)x(1, 0, 0, 12) (constant=True); AIC=29585.293, BIC=2963
9.875, Time=25.847 seconds
Fit ARIMA: (3, 0, 3)x(1, 0, 2, 12) (constant=True); AIC=29579.288, BIC=2964
5.999, Time=71.712 seconds
Fit ARIMA: (3, 0, 2)x(2, 0, 1, 12) (constant=True); AIC=29587.135, BIC=2964
7.781, Time=73.021 seconds
Fit ARIMA: (4, 0, 3)x(2, 0, 1, 12) (constant=True); AIC=29579.706, BIC=2965
2.482, Time=82.381 seconds
Fit ARIMA: (3, 0, 4)x(2, 0, 1, 12) (constant=True); AIC=29579.646, BIC=2965
2.421, Time=91.857 seconds
Fit ARIMA: (2, 0, 4)x(2, 0, 1, 12) (constant=True); AIC=29576.620, BIC=2964
3.331, Time=70.104 seconds
Fit ARIMA: (2, 0, 4)x(1, 0, 1, 12) (constant=True); AIC=29583.960, BIC=2964
4.607, Time=29.800 seconds
Fit ARIMA: (2, 0, 4)x(2, 0, 0, 12) (constant=True); AIC=29582.698, BIC=2964
3.345, Time=71.533 seconds
Fit ARIMA: (2, 0, 4)x(2, 0, 2, 12) (constant=True); AIC=29547.505, BIC=2962
0.281, Time=100.620 seconds
Near non-invertible roots for order (2, 0, 4)(2, 0, 2, 12); setting score to
inf (at least one inverse root too close to the border of the unit circle:
0.994)
Fit ARIMA: (2, 0, 4)x(1, 0, 0, 12) (constant=True); AIC=29568.222, BIC=2962
2.804, Time=23.695 seconds
```

```
Fit ARIMA: (2, 0, 4)x(0, 0, 0, 12) (constant=True); AIC=29479.053, BIC=2952
7.570, Time=8.133 seconds
Fit ARIMA: (2, 0, 4)x(0, 0, 1, 12) (constant=True); AIC=29472.268, BIC=2952
6.849, Time=31.449 seconds
Fit ARIMA: (2, 0, 4)x(0, 0, 2, 12) (constant=True); AIC=29573.231, BIC=2963
3.878, Time=81.292 seconds
Fit ARIMA: (2, 0, 4)x(1, 0, 2, 12) (constant=True); AIC=29578.088, BIC=2964
4.799, Time=84.294 seconds
Fit ARIMA: (1, 0, 4)x(0, 0, 1, 12) (constant=True); AIC=29579.628, BIC=2962
8.145, Time=15.428 seconds
Fit ARIMA: (2, 0, 3)x(0, 0, 1, 12) (constant=True); AIC=29585.945, BIC=2963
4.462, Time=10.997 seconds
Fit ARIMA: (3, 0, 4)x(0, 0, 1, 12) (constant=True); AIC=29523.347, BIC=2958
3.993, Time=30.866 seconds
Fit ARIMA: (2, 0, 5)x(0, 0, 1, 12) (constant=True); AIC=29583.600, BIC=2964
4.247, Time=15.339 seconds
Fit ARIMA: (1, 0, 3)x(0, 0, 1, 12) (constant=True); AIC=29586.789, BIC=2962
9.241, Time=10.503 seconds
Fit ARIMA: (1, 0, 5)x(0, 0, 1, 12) (constant=True); AIC=29579.796, BIC=2963
4.378, Time=21.199 seconds
Fit ARIMA: (3, 0, 3)x(0, 0, 1, 12) (constant=True); AIC=29584.591, BIC=2963
9.173, Time=20.767 seconds
Fit ARIMA: (3, 0, 5)x(0, 0, 1, 12) (constant=True); AIC=29449.139, BIC=2951
5.850, Time=34.889 seconds
Near non-invertible roots for order (3, 0, 5)(0, 0, 1, 12); setting score to
inf (at least one inverse root too close to the border of the unit circle:
0.997)
Total fit time: 2443.158 seconds
```

In [0]:

```
arima_forecast = arima.predict(n_periods = test_period)
```

In [19]:

```
from sklearn.metrics import mean_squared_error

mean_squared_error(all_df['Monthly Mean Total Sunspot Number'][-test_period:], arima_foreca
```

Out[19]:

```
3196.9943474969787
```

## Задание 4

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

Сначала нужно создать датасет из данных.

In [20]:

```
! pip install tensorflow-gpu --pre --quiet

! pip show tensorflow-gpu
```

```
Name: tensorflow-gpu
Version: 2.2.0rc3
Summary: TensorFlow is an open source machine learning framework for everyon
e.
Home-page: https://www.tensorflow.org/ (https://www.tensorflow.org/)
Author: Google Inc.
Author-email: packages@tensorflow.org
License: Apache 2.0
Location: /usr/local/lib/python3.6/dist-packages
Requires: google-pasta, opt-einsum, tensorboard, six, h5py, tensorflow-estim
ator, termcolor, wrapt, wheel, keras-preprocessing, protobuf, gast, astunpar
se, scipy, numpy, grpcio, absl-py
Required-by:
```

In [0]:

```
TIME_STEPS = 100

TEST_PERIOD = 1000
```

In [0]:

```
import numpy as np
from datetime import timezone

def timeseries_to_dataset(_X_ts, _time_steps):

    samples_n_ = len(_X_ts) - _time_steps

    print( len(_X_ts))

    X_ = np.zeros((samples_n_, _time_steps))
    y_ = np.zeros((samples_n_, ))

    for i in range(samples_n_):

        X_[i] = _X_ts[i:(i + _time_steps)]

        y_[i] = _X_ts[(i + _time_steps)]

    return X_[..., np.newaxis], y_
```

In [31]:

```
X, y = timeseries_to_dataset(all_df['Monthly Mean Total Sunspot Number'][:-TEST_PERIOD].val

X_test, y_test = timeseries_to_dataset(all_df['Monthly Mean Total Sunspot Number'][-TEST_PE
```

```
2252
1000
```

In [0]:

```
import tensorflow as tf
from tensorflow import keras
```

In [37]:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

model = tf.keras.Sequential()

model.add(LSTM(8, activation = 'relu', return_sequences = True, input_shape = X.shape[-2:]))
model.add(LSTM(8, activation = 'relu'))
model.add(Dense(1))
```

```
WARNING:tensorflow:Layer lstm_4 will not use cuDNN kernel since it doesn't m
eet the cuDNN kernel criteria. It will use generic GPU kernel as fallback wh
en running on GPU
WARNING:tensorflow:Layer lstm_5 will not use cuDNN kernel since it doesn't m
eet the cuDNN kernel criteria. It will use generic GPU kernel as fallback wh
en running on GPU
```

In [38]:

```
model.compile(optimizer = 'adam',
              loss = 'mse',
              metrics = ['accuracy'])

model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_4 (LSTM)                (None, 100, 8)            320
_________________________________________________________________
lstm_5 (LSTM)                (None, 8)                 544
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 9
=================================================================
Total params: 873
Trainable params: 873
Non-trainable params: 0
_________________________________________________________________
```

In [39]:

```
model.fit(x = X, y = y, validation_split = 0.15, epochs = 20, verbose = 0)
```

Out[39]:

```
<tensorflow.python.keras.callbacks.History at 0x7f162e959fd0>
```

In [40]:

```
results = model.evaluate(X_test, y_test)

print('Test mse, test accuracy:', results)
```

```
29/29 [==============================] - 1s 21ms/step - loss: 651.9083 - acc
uracy: 0.0011
Test mse, test accuracy: [651.9083251953125, 0.0011111111380159855]
```

## Задание 5

Сравните качество прогноза моделей.

Какой максимальный результат удалось получить на контрольной выборке?

Нейронная сеть дала среднеквадратичную ошибку в 4 раза больше, чем ARIMA, а точность предсказания вообще близка к нулю. Можно сделать вывод, что предсказание временных рядов требует более тонкой настройки архитектуры сетей.