

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение
высшего образования
«Уральский федеральный университет имени
первого Президента России Б. Н. Ельцина»

**ПРОГНОЗИРОВАНИЕ ВРЕМЕННЫХ РЯДОВ
НА ОСНОВЕ LSTM-СЕТЕЙ**

**Методические указания к выполнению
лабораторной работы № 8**

Екатеринбург
2020

Содержание

1. Введение.....	3
2. Задание на лабораторную работу	3
3. Требования к оформлению отчета.....	9

1. Введение

LSTM – особая разновидность архитектуры рекуррентных нейронных сетей, способная к обучению долговременным зависимостям. Вместо стандартных слоёв нейронных сетей данная сеть использует блоки ячеек. Эти ячейки имеют различные компоненты, называемые входными данными, выходными данными и компонентами памяти. В данной работе рассматривается реализация LSTM-прогнозной сети на основе фреймворка *keras* над *tensorflow* (для доп. информации см. сайт keras.io)

2. Задание на лабораторную работу

Результатом выполнения лабораторной работы является оформленный отчет в виде *Jupyter*-тетради, в котором должны быть представлены и отражены все нижеперечисленные пункты:

- 1) Сначала импортируйте в свой код нужные библиотеки, функции и т.д.

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
import h5py
```

```
%matplotlib inline
```

```
from sklearn.preprocessing import MinMaxScaler
```

```
from keras.models import Sequential
```

```
from keras.layers import Dense
```

```
from keras.layers import LSTM
```

```
from keras.layers import Dropout
```

- 2) Загрузите из mat-файла **Fort.mat** ряд, содержащий отсчеты некоторого реального ВР, всего 174 отсчета в вектор-строке, и отмасштабируйте его в диапазон от 0 до 1, так как функция активации слоя LSTM корректно обрабатывает значения только в данном диапазоне:

```
file = h5py.File('Fort.mat', 'r')
data = file.get('Fort')
Fort = np.array(data)
F = Fort
scaler = MinMaxScaler(feature_range=(0, 1))
F = scaler.fit_transform(F)
F_tr = F[:150]
F_test = F[144:]
plt.figure(figsize = (10, 5))
plt.plot(F, 'k')
plt.plot(np.r_[150:], F_test, 'b')
plt.plot(np.r_[144:174], F_test, 'r')
plt.show()
```

- 3) Важно произвести предобработку исходных данных в формат, понимаемый слоем LSTM-сети, в виде «порций» (batches) для обучения/валидации. Ниже приведен пример для модели сети 6 порядка авторегрессии на $(150-6) = 144$ смежных точках ряда.

```
from keras.preprocessing.sequence import TimeseriesGenerator
data_gen = TimeseriesGenerator(F_tr, F_tr,
                                length=6, sampling_rate=1,
                                batch_size=150)
```

```
batch_0 = data_gen[0]
x, y = batch_0 # вход и обучающий выход для сети
print(x.shape) # 144 точки обучения, прогноз 1 точки по 6 прошлым
xx=np.reshape(x, (x.shape[0], 1, x.shape[1]))
yy=y
print(xx.shape) # меняем местами размерности
print(yy.shape) #
```

- 4) Затем составляется сама модель прогнозной сети. В простейшем случае нам понадобится только 1 внутренний LSTM-слой и 1 выходной слой. Тогда модель строится как:

```
from keras.utils import plot_model
model = Sequential() # слои соединены последовательно
model.add(LSTM(units=20, input_shape=(1, 6))) # 20 нейронов
model.add(Dense(units = 1)) # выход одномерный
model.compile(optimizer = 'adam', loss = 'mean_squared_error')
plot_model(model, to_file='model.png') # рисунок полученной сети
```

Но слои LSTM замечательны тем, что могут иметь связи между собой кроме стандартных входов-выходов для реализации «долгой памяти». В этом случае в коде модели необходимо это указывать. Также в такие модели зачастую добавляют слои “*Dropout*”, которые со случайной заданной вероятностью обнуляют входы следующего слоя при обучении, тем самым позволяя избежать переобучения всей нейронной сети в целом. Например, модель из 3 слоев LSTM может быть построена примерно следующим образом:

```
from keras.utils import plot_model  
  
model = Sequential()  
model.add(LSTM(units=20, return_sequences=True, input_shape=(1, 6)))  
model.add(Dropout(0.2))  
model.add(LSTM(units=20, return_sequences=True))  
model.add(Dropout(0.2))  
model.add(LSTM(units=20))  
model.add(Dense(units = 1))  
model.compile(optimizer = 'adam', loss = 'mean_squared_error')  
plot_model(model, to_file='model.png')
```

- 5) В качестве функции оптимизации здесь задан алгоритм **Adam**. **Adam** — adaptive moment estimation, оптимизационный алгоритм, который сочетает в себе и идею накопления движения и идею более слабого обновления весов для типичных признаков. Отличительная особенность в том, что функция использует сглаженные версии среднего и среднеквадратичного градиента. Алгоритм используется для градиентной оптимизации стохастических целевых функций первого порядка, основанный на адаптивных оценках моментов более низкого порядка. Данный способ хорошо подходит для нестационарных целей и задач с очень шумными и / или разреженными градиентами. Во время процедуры обучения мы минимизируем ошибку между прогнозом и фактическими наблюдениями в терминах корня среднеквадратичной ошибки **'mean_squared_error'**.

6) Наконец, производим обучение нашей модели.

```
model.fit(xx, yy, epochs = 100)    # 100 эпох по 144 точки
```

7) Чтобы посмотреть, что же у нас получилось обучить, построим ретроспективный прогноз, с переходом обратно к исходному масштабу данных:

```
trainPredict = model.predict(xx)  
trainPredict = scaler.inverse_transform(trainPredict)  
plt.figure(figsize = (10, 5))  
plt.plot(Fort, 'k')  
plt.plot(np.r_[6:150],trainPredict, 'b')  
plt.show()
```

8) А для тестовой проверки прогноза придется исходные точки вновь переработать в формат, понятный для модели LSTM-сети:

```
data_gen = TimeseriesGenerator(F_test, F_test,  
                                length=6, sampling_rate=1,  
                                batch_size=150)  
batch_0 = data_gen[0]  
x, y = batch_0  
xx=np.reshape(x, (x.shape[0], 1, x.shape[1]))  
yy = y  
print(xx.shape)           # прогноз на 24 точки по 6 наблюдениям  
print(yy.shape)           #
```

- 9) Строим получившийся тестовый прогноз в нужном масштабе:

```
testPredict = model.predict(xx)
testPredict = scaler.inverse_transform(testPredict)
plt.figure(figsize = (10, 5))
plt.plot(Fort, 'k')
plt.plot(np.r_[150:174],testPredict, 'b')
plt.show()
```

- 10) Теперь самостоятельно попробуйте подобрать такую модель и ее параметры (число нейронов в слоях, общая структура, коэффициенты Dropout, число предыдущих точек для прогноза, число наблюдений для обучения и т.д.), чтобы получить наиболее удачный результат с Вашей точки зрения. **Длина прогноза** должна быть в **24** отсчета. Графики исходного ряда *Fort* и прогноза строятся вместе, так как они имеют малую длину и вполне могут поместиться рядом с достаточной точностью.

- 11) Постройте прогноз на **256** точек для следующего модельного временного ряда и самостоятельно выберите для него параметры:

```
t = np.linspace(0, 1, 1024)
f1 = 10
f2 = 50
F=1.7*np.sin(2*np.pi*f1*t)+np.sin(2*np.pi*f2*t)+0.2*rand.randn(len(t))
plt.figure(figsize = (10, 5))
plt.plot(t, F, 'k')
plt.plot(t, 1.7*np.sin(2*np.pi*f1*t), 'b')
plt.plot(t, np.sin(2*np.pi*f2*t), 'r')
plt.show()
```


- 12) Постройте прогноз на **256** точек для следующего модельного временного ряда и самостоятельно выберите для него параметры:

```
t = np.linspace(0,1,1024)
f1 = 10
f2 = 40
f3 = 100
f4 = 150
F=2.0*np.sin(2*np.pi*f1*t)+1.5*np.sin(2*np.pi*f2*t)
    +0.8*np.sin(2*np.pi*f3*t)
    +0.5*np.sin(2*np.pi*f4*t)+0.2*rand.randn(len(t))
plt.figure(figsize = (10, 15))
plt.subplot(5,1,1)
plt.plot(t, F, 'k')
plt.subplot(5,1,2)
plt.plot(t, 2.0*np.sin(2*np.pi*f1*t), 'b')
plt.subplot(5,1,3)
plt.plot(t, 1.5*np.sin(2*np.pi*f2*t), 'r')
plt.subplot(5,1,4)
plt.plot(t, 0.8*np.sin(2*np.pi*f3*t), 'g')
plt.subplot(5,1,5)
plt.plot(t, 0.5*np.sin(2*np.pi*f4*t), 'm')
plt.show()
```

3. Требования к оформлению отчета

Отчет в Jupyter-тетради должен обязательно содержать: номер лабораторной работы, ФИО студента, номер варианта (либо студенческий номер), номер группы, результаты выполнения работы с комментариями студента (комментарии пишутся после #) и изображениями.