МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Уральский федеральный университет имени
первого Президента России Б. Н. Ельцина»

ПРИМЕНЕНИЕ ВЕЙВЛЕТ МЕТОДОВ ДЛЯ ДЕКОМПОЗИЦИИ ВРЕМЕННЫХ РЯДОВ ПРОГНОЗИРОВАНИЕ НА ОСНОВЕ ИСКУССТВЕННЫХ НЕЙРОННЫХ СЕТЕЙ

Методические указания к выполнению
практического задания № 6

Екатеринбург
2023

Содержание

Введение

Целью данной лабораторной работы является знакомство и изучение средств *Python* для работы с вейвлетами и нейронными сетями. Студенты приобретут навыки по использованию этого инструмента для анализа и декомпозиции временных рядов, а также получения оценки их частотно-временных характеристик. 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 входным 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

1. 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 numpy.random as rand
import matplotlib.pyplot as plt
import h5py
import pywt
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
```

```
%matplotlib inline
```

2. Создадим зашумленный временной ряд с 2 периодиками, отобразите полученный ряд:

```
t = np.linspace(0, 1, 1024)
f1 = 10
f2 = 50
F=np.sin(2*np.pi*f1*t)+np.sin(2*np.pi*f2*t)+0.2*rand.randn(len(t))
```

3. Наиболее широкой библиотекой для работы с вейвлетами в *Python* является *PyWavelets*. При работе с вейвлетами надо понимать, что это очень гибкий инструмент со множеством параметров. Во-первых, самым главным параметр вейвлет-разложения является сам материнский (базисный) вейвлет. Множество доступных в библиотеке *PyWavelets* семейств вейвлетов и их модификаций доступно на сайте: http://wavelets.pybytes.com. Внимательно изучите эти семейства, так как выбор базового вейвлета существенно влияет на конечный результат декомпозиции по принципу самоподобия выделенных компонент и выбранного вейвлета.

   Для начала используем вейвлет Мейера:

```
wvlt = pywt.Wavelet('dmey')
```

4. Во-вторых, декомпозиция с помощью дискретных вейвлетов происходит до определенного уровня (**level**), ограниченного размером доступных данных. В нашем случае возможно разложение до 4 уровня:

```
pywt.dwt_max_level(len(F), wvlt) # будет выведено число 4
```

5. В-третьих, по краям временного интервала вейвлет может по-разному трактовать конструируемые точки для экстраполяции (**mode**): простое дополнение нулями, константами, симметрично/асимметрично, периодически и т.д. В зависимости от вида исходных данных, лучше подходит тот или иной режим.
6. Ну и наконец, декомпозиция всегда происходит в виде комбинации коэффициентов **Аппроксимации** (*cA*) плюс **Детали** (*cD*). Всегда есть одна

аппроксимация, а число деталей равно уровню декомпозиции. Меняя выбор группировки коэффициентов аппроксимации и деталей, будут меняться восстановленные компоненты и соответствующая декомпозиция ряда.

7. Разобьем наш исходный ряд на компоненты с помощью вейвлета Мейера, в режиме периодизации, до 4 уровня декомпозиции:

```
cA4, cD4, cD3, cD2, cD1 = pywt.wavedec(F, wvlt, mode='periodization', level=4)
```

8. Обратите внимание, как выглядят выходные результаты: одна аппроксимация **cA4** с номером уровня и 4 детали с убывающими номерами от 4 уровня **cD4** до 1 уровня **cD1**. Если бы мы декомпозировали ряд на 3 уровня (**level=3**), то тогда выходные значения следовало бы записать как: **cA3, cD3, cD2, cD1**. Можно на выходе функции использовать и одну переменную, но ее все равно придется разбивать на отдельные элементы. Также следует помнить, что в результате декомпозиции получаются не новые временные ряды, а только вейвлет-коэффициенты декомпозиции.
9. Восстановим две периодики исходного модельного ряда:

```
Fre = pywt.waverec((cA4, None, None, None, None), wvlt, mode='periodization')
Fre2 = pywt.waverec((None, cD4, None, None, None), wvlt, mode='periodization')
plt.figure(figsize = (10, 5))
plt.plot(t, F, 'k')
plt.plot(t, Fre, 'b') # это будет первая периодика
plt.plot(t, Fre2, 'r') # это будет вторая периодика
plt.show()
```

10. Снова обратите внимание на форму записи восстанавливаемых компонент. Те коэффициенты, которые мы не хотим использовать для реконструкции компонент (лишние детали и т.д.), мы заменяем на **None**. Меняя комбинации используемых и неиспользуемых аппроксимаций и деталей, мы будем получать разные восстановленные компоненты.
11. Проведите аналогичную декомпозицию для 3 уровня (**level = 3**). Вейвлет не меняйте. Сравните полученные результаты.
12. Повторите все проделанные шаги по декомпозиции ряда и восстановлению его компонент **для своего варианта** базисного вейвлета. Наименование

вейвлета указано в таблице ниже. Используйте любой возможный уровень декомпозиции – выберите тот, который окажется наиболее точным среди них.

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
db8	*coif3*	*haar*	*sym4*	*coif1*	*db4*	*db2*	*sym5*
9	**10**	**11**	**12**	**13**	**14**	**15**	**16**
coif5	*haar*	*sym6*	*db6*	*sym7*	*db10*	*sym8*	*db5*
17	**18**	**19**	**20**	**21**	**22**	**23**	**24**
coif4	*sym10*	*db16*	*haar*	*db20*	*sym16*	*coif2*	*db18*

13. Есть и другие модификации простого вейвлет-преобразования, позволяющие декомпозировать временные ряды. Например, есть **Стационарное Вейвлет Преобразование** (Stationary Wavelet Transform = **SWT**). Этот метод дает гораздо большие возможности декомпозиции по уровню и по комбинации аппроксимаций и деталей:

```
(cA5, cD5), (cA4, cD4), (cA3, cD3), (cA2, cD2), (cA1, cD1) = pywt.swt(F, wvlt, level=5)
```

14. Как видно, вместо одной аппроксимации и множества деталей мы теперь получаем пары коэффициентов аппроксимации и детали. Причем, на самом деле, при восстановлении их можно использовать уже в совершенно разных комбинациях. Восстановим искомые периодические компоненты следующей комбинацией пар, при этом нам еще потребуется нормировка, так как часть коэффициентов мы полностью выбросили:

```
rr1 = pywt.iswt([(cA5, cD5)], wvlt)
rr2 = pywt.iswt([(cD4, cD3)], wvlt)
```

```
plt.figure(figsize = (10, 5))
plt.plot(t, F, 'k')
plt.plot(t, rr1/5, 'b') # перенормируем
plt.plot(t, rr2/4, 'r') # перенормируем
plt.show()
```

15. Проделайте декомпозицию SWT **для своего варианта** вейвлета. Уровень декомпозиции может отличаться.
16. Ну и остался самый мощный инструмент вейвлет-декомпозиции, называемый **Пакетной Вейвлет Декомпозицией** (Wavelet Packet Decomposition = **WPD**). Этот метод опирается на дискретное преобразование, но позволяет произвести полный перебор комбинаций коэффициентов аппроксимации **a** и деталей **d**. Например, для 4 уровня это будет 16 комбинаций вида: 'aaaa', 'aaad', 'aada', 'aadd', 'adaa', 'adad', 'adda', 'addd', 'daaa', 'daad', 'dada', 'dadd', 'ddaa', 'ddad', 'ddda', 'dddd'

    Для сравнения, на 4 уровне обычной дискретной вейвлет декомпозиции из пункта 7 доступна только одна комбинация 'addd'.
17. При таком большом числе возможных вариантов, декомпозиция представляется в виде бинарного дерева, выборка или удаление узлов из которого и будет менять после реконструкции полученные временные ряды. Но для начала создадим общую пакетную декомпозицию:

```
wp = pywt.WaveletPacket(data=F, wavelet='dmey', mode='periodization')
print([node.path for node in wp.get_level(4, 'freq')]) # выводим все комбинации
узлов, упорядоченные по их частотной ширине спектра
```

18. Попробуем удалить один из узлов и посмотреть, что получится:

```
del wp['aaaa'] # удалим самый «глубокий» узел
reF = wp.reconstruct() # и восстановим ряд …
plt.figure(figsize = (10, 5))
plt.plot(t, F, 'k')
plt.plot(t, reF, 'b') # получим нечто периодическое, плохого качества
plt.show()
```

19. Если удаление узлов не приводит к желаемым результатам, возможно есть смысл делать отдельную выборку ветвей этих узлов:

```
wp = pywt.WaveletPacket(data=F, wavelet='dmey', mode='periodization')
new_wp = pywt.WaveletPacket(data=None, wavelet='dmey', mode='periodization')
new_wp['aaaa'] = wp['aaaa'].data  # выбираем первую ветвь
new_wp.reconstruct(update=True)   # обновляем данные
reF1 = new_wp.data                # восстанавливаем под нее ряд 1
new_wp = pywt.WaveletPacket(data=None, wavelet='dmey', mode='periodization')
new_wp['aaad'] = wp['aaad'].data  # выбираем вторую ветвь
new_wp.reconstruct(update=True)   # обновляем данные
reF2 = new_wp.data                # восстанавливаем под нее ряд 2
plt.figure(figsize = (10, 5))
plt.plot(t, F, 'k')
plt.plot(t, reF1, 'b')      # компонента 1
plt.plot(t, reF2, 'r')            # компонента 2
plt.show()
```

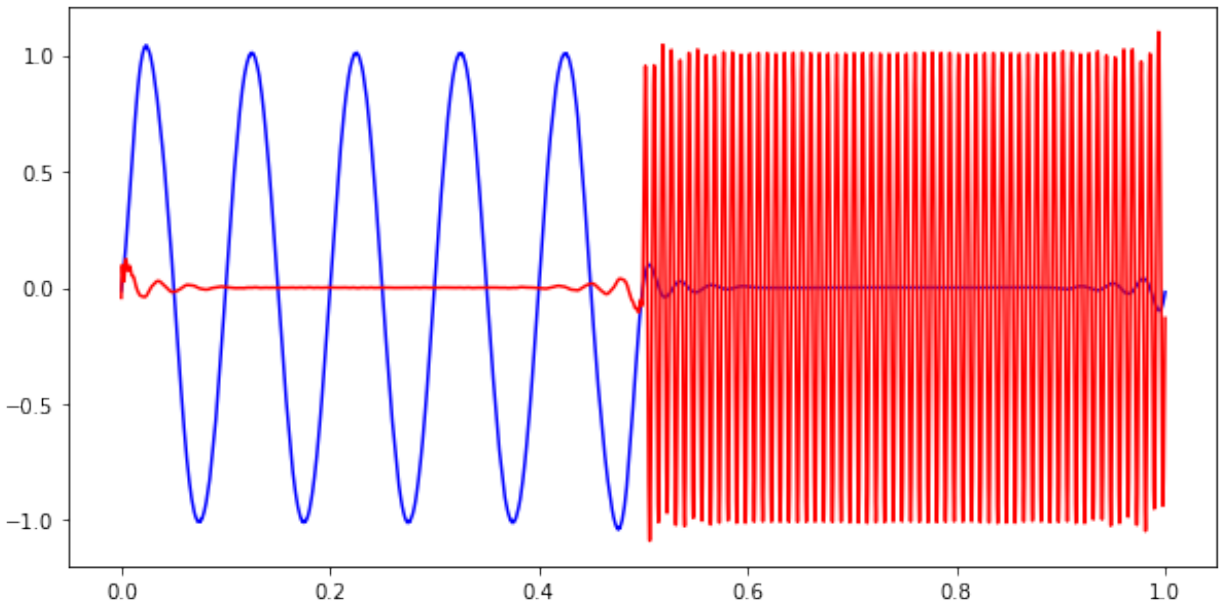
20. Проделайте подобную декомпозицию WPD **для своего варианта** вейвлета с выбором/удалением нужных узлов. Уровень декомпозиции может отличаться.
21. Теперь примените полученные навыки вейвлет-декомпозиции всех видов, на основе базисного вейвлета **для своего варианта** из таблицы, для получения компонент следующих временных рядов.

22. Декомпозируйте сигнал с частотным изломом на 2 периодические компоненты, разделенные по времени:

```
t = np.linspace(0, 1, 4096)
xf = np.zeros(4096)
for i in range(0, len(t)//2):
    xf[i] = np.sin(2*np.pi*10*t[i])
for i in range(len(t)//2, len(t)):
    xf[i] = np.sin(2*np.pi*120*t[i])
```

```
plt.figure(figsize = (10, 5))
plt.plot(t, xf)
plt.show()
```

Хороший результат выглядит примерно так:



23. Попробуйте выделить **экспоненциальный тренд** из следующего зашумленного временного ряда:

```
t = np.linspace(0, 4, 4096)
Fexp = np.exp(-0.4*np.pi*t) + 0.2*rand.randn(len(t))
```

Внимание! Для выделения тренда опцию **mode='periodization'** внутри функций вейвлетов нужно будет убрать, так как для тренда не нужен режим периодического дополнения точек по краям.

24. Смоделируйте временной ряд из **4 гармоник с шумом**, и разделите его на **4 гармоники** с помощью вейвлет декомпозиции (отобразите полученные результаты):

$$u(t) = 2\sin(2\pi f_1 t),\ 1{,}5\sin(2\pi f_2 t),\ 0{,}8\sin(2\pi f_3 t),\ 0{,}5\sin(2\pi f_4 t) + \xi$$

Ряд строится для `t = np.linspace(0,1,1024)` строятся полученные результаты. Результаты оформить в виде графиков полученных компонент и компоненты, описанные в формуле $2sin(2\pi f_1 t)$, $1,5sin(2\pi f_2 t)$, $0,8sin(2\pi f_3 t)$, $0,5sin(2\pi f_4 t)$.

Вариант	1,20	2,19	3,18	4,17	5,16	6,15	7,14	8,13	9,12	10,11
f1	10	20	30	40	50	60	10	20	30	10
f2	40	30	15	10	20	10	50	40	10	30
f3	100	110	90	130	115	100	105	100	125	100
f4	150	120	170	150	160	180	150	150	160	150

Внимание! Здесь Вам потребуется выделять более двух компонент, из которых только одна будет определяться на основе коэффициента аппроксимации, а остальные – на основе коэффициентов детализации. Но коэффициенты детализации низких уровней имеют 2^n больше отсчетов (из-за квадратичной двоичной фильтрации), чем самый верхний уровень. Как тогда восстановить ряд? Тогда нужно просто использовать при восстановлении меньше **None**, чем необходимо. Например, для 5 уровня можно построить более 6 компонент (аппроксимация + 5 деталей + комбинации деталей) следующим образом:

```
cA5, cD5, cD4, cD3, cD2, cD1 = pywt.wavedec(F, wvlt, mode='periodization', level=5)
pywt.waverec((cA5, None, None, None, None, None), wvlt, mode='periodization')
pywt.waverec((None, cD5, None, None, None, None), wvlt, mode='periodization')
pywt.waverec((None, cD4, None, None, None), wvlt, mode='periodization')
pywt.waverec((None, cD3, None, None), wvlt, mode='periodization')
pywt.waverec((None, cD2, None), wvlt, mode='periodization')
pywt.waverec((None, cD1), wvlt, mode='periodization')'periodization')
```

или разные комбинации деталей:

```
pywt.waverec((None, cD4, cD3, None, None), wvlt, mode='periodization')
pywt.waverec((None, cD4, cD3, cD2, None), wvlt, mode='periodization')
pywt.waverec((None, cD3, None, cD1), wvlt, mode='periodization')
```

и т.д.

25. Загрузите временной ряд из файла doppler.mat и отобразите:

```
file = h5py.File('doppler.mat','r')
data = file.get('data')
```

```
data = np.array(data)
data.ravel()
```

Декомпозируйте этот временной ряд с помощью вейвлетов таким образом, чтобы максимально очистить его от шума, но полностью сохранить его возрастающую периодику. Используйте только базисный вейвлет для своего варианта.

26. Загрузите и отобразите 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 = np.ravel(Fort)
scaler = MinMaxScaler(feature_range=(0, 1))
F = scaler.fit_transform(F)
F_tr = F[:150]
F_test = F[144:]
```

27. 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     # пояснить
print(x.shape)     # пояснить
xx=np.reshape(x, (x.shape[0], 1, x.shape[1]))
yy=y
print(xx.shape)    # пояснить
print(yy.shape)
```

28. 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()     # пояснить
model.add(LSTM(units=20, input_shape=(1, 6))) # пояснить
model.add(Dense(units = 1))              # пояснить
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')
```

29. 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'**.

30. Наконец, производим обучение нашей модели.

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

31. 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()
```

32. 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)
print(yy.shape)
```

33. 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()
```

34. 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.

35. Постройте и отобразите 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
))
```

36. 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))
```

37. Оформите итоговый отчет о проделанной работе.

2. 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 и выводами.