Московский авиационный институт
(Национальный исследовательский университет)
Институт №8 «Информационные технологии и прикладная математика»

Кафедра вычислительной математики и программирования

Лабораторная работа № 8

по курсу «Нейроинформатика».

Тема: «Динамические сети».

Студент: Вельтман Л.Я.
Группа: 80-407Б
Преподаватели: Тюменцев Ю.В.
Аносова Н.П.

Вариант: 7

Оценка:

Москва, 2020

Цель работы.

Целью работы является исследование свойств некоторых динамических нейронных сетей, алгоритмов обучения, а также применение сетей в задачах аппроксимации функций и распознавания динамических образов.

Основные этапы работы.

1. Использовать сеть прямого распространения с запаздыванием для предсказания значений временного ряда и выполнения многошагового прогноза.

2. Использовать сеть сеть прямого распространения с распределенным запаздыванием для распознавания динамических образов.

3. Использовать нелинейную авторегрессионную сеть с внешними входами для аппроксимации траектории динамической системы и выполнения многошагового прогноза.

Оборудование.

Операционная система: macOS Catalina version 10.15.5

Процессор: 2,3 GHz 2-ядерный процессор Intel Core i5

Оперативная память: 8 ГБ 2133 MHz LPDDR3

Программное обеспечение.

Работа выполнена на Python3 с применением библиотек numpy (для вычислений), pandas и matplotlib (для графиков) при помощи командной оболочки Jupyter Notebook.

Сценарий выполнения работы.

Базовая сеть FTDNN состоит из двух компонентов: структуры памяти и нелинейного ассоциатора. Структура памяти представляет собой линию временной задержки, которая содержит p самых последних входов, сгенерированных элементом задержки, представленным оператором d, в то время как ассоциатором является обычная сеть с прямой связью. Структура памяти хранит

актуальную информацию о прошлом, и ассоциатор использует память для предсказания будущих событий.

Особенностью FTDNN является то, что структура памяти сосредоточена на входном уровне; это отличает ее от общей нейронной сети с временной задержкой (TDNN). Основным преимуществом FTDNN является то, что он менее сложен, чем обычный TDNN, и имеет те же возможности обработки временных шаблонов.

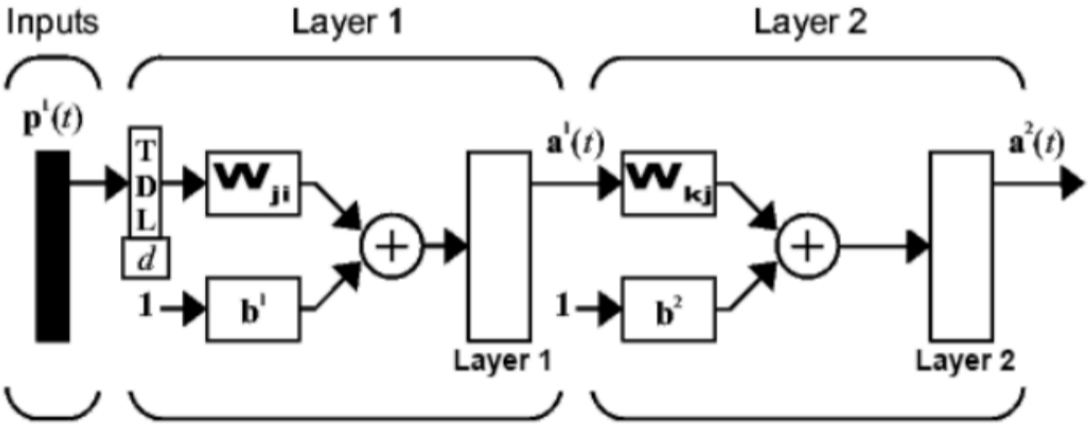


Fig. 2. Focused Time Delay Neural Network Structure [6].

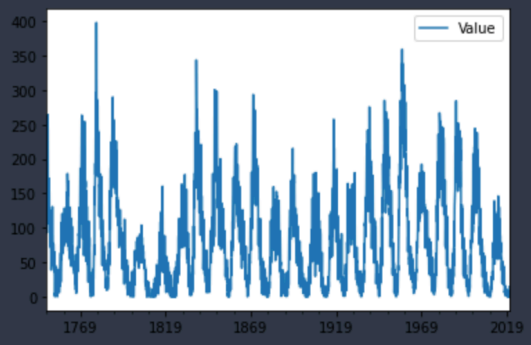
Сети с распределенной задержкой похожи на сети с прямой связью, за исключением того, что каждый вход и веса уровня имеют связанную с ним линию задержки отвода. Это позволяет сети иметь конечный динамический отклик на входные данные временного ряда. Эта сеть также похожа на нейронную сеть с временной задержкой (timedelaynet), которая имеет задержки только на входном весе.

Нелинейная авторегрессионная сеть с экзогенными (внешним) входами (NARX) представляет собой рекуррентную динамическую сеть с обратными связями, охватывающими несколько уровней сети. Они могут научиться предсказывать один временной ряд с учетом прошлых значений того же временного ряда, входной обратной связи и другого временного ряда, называемого внешним или экзогенным временным рядом.

Входные данные и результаты

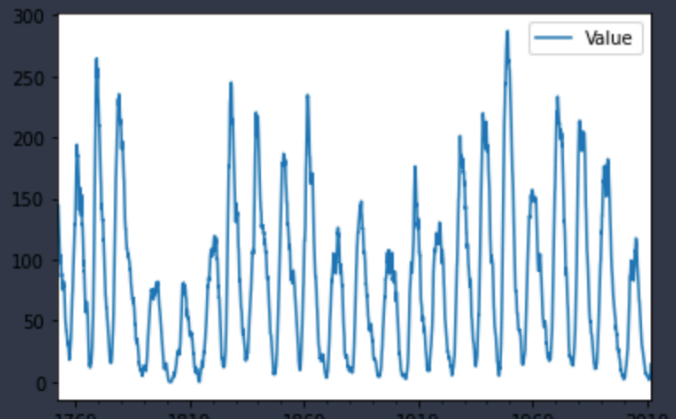
Задание 1

```
In [7]: 1 df.plot();
```



Сглаживание траектории с помощью усредняющего фильтра smooth

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff352763c18>
```



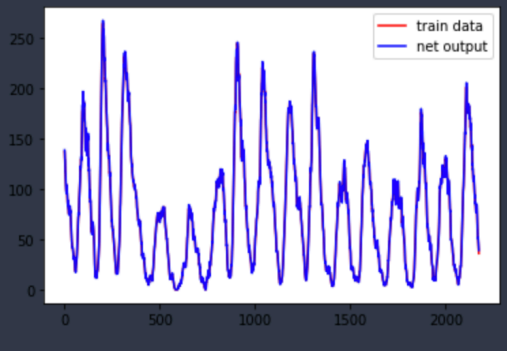
Рассчитать выход сети для обучающего подмножества.

In [15]:
```
predictTrain = model.predict(xTrain)
```

In [16]:
```
MSE = mean_squared_error(yTrain, predictTrain)
print('MSE = {}'.format(MSE))
print('RMSE = {}'.format(np.sqrt(MSE)))
```

```
MSE = 8.210838029299524
RMSE = 2.8654559897683867
```

In [17]:
```
plt.plot(yTrain, color='red')
plt.plot(predictTrain, color='blue')
plt.legend(['train data', 'net output'])
plt.show()
```



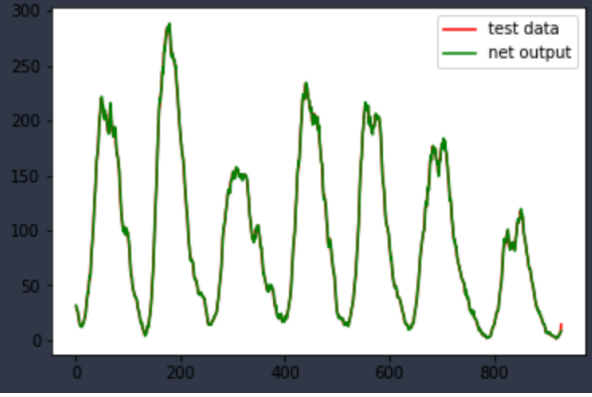
Рассчитать выход сети для тестового подмножества.

In [18]:
```
predictTest = model.predict(xTest)
```

In [19]:
```
MSE = mean_squared_error(yTest, predictTest)
print('MSE = {}'.format(MSE))
print('RMSE = {}'.format(np.sqrt(MSE)))
```

```
MSE = 7.831346190185805
RMSE = 2.7984542501505727
```

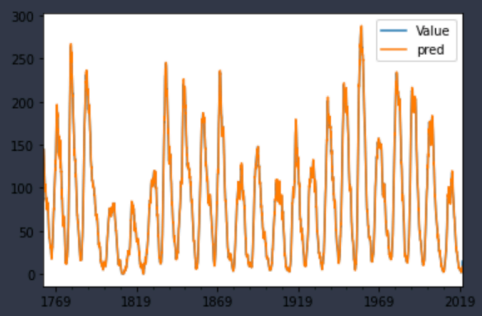
In [20]:
```
plt.plot(yTest, color='red')
plt.plot(predictTest, color='green')
plt.legend(['test data', 'net output'])
plt.show()
```



In [21]:
```
df['pred'] = np.concatenate([df[:deep].values.flatten(),
                             predictTrain.flatten(),
                             df[trainSize:trainSize+deep].values.flatten(),
                             predictTest.flatten()])
```

In [22]:
```
df.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff352117dd8>
```

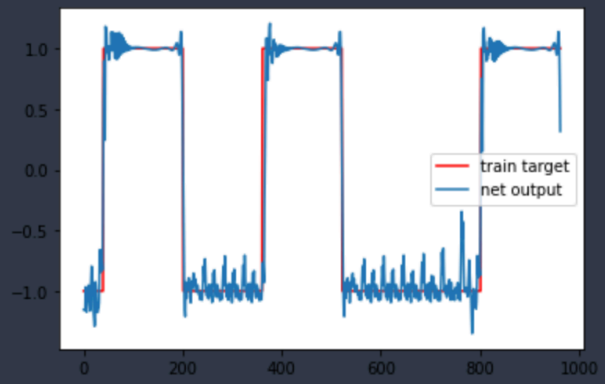


Задание 2

Рассчитываем выход сети

In [29]:
```
output = pyrenn.NNOut(P, nn)
```

In [30]:
```
pl.plot(T, color='red')
plt.plot(output)
plt.legend(['train target', 'net output'])
plt.show()
```



Преобразуем предсказанные значения.

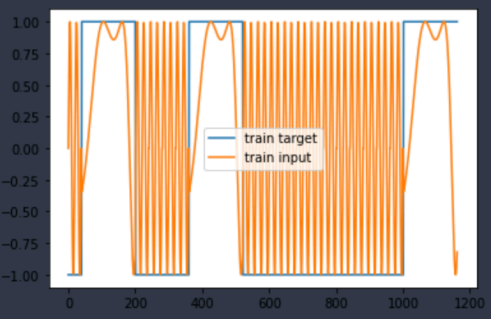
In [31]:
```
output[output >= 0] = 1.0
output[output < 0] = -1.0

MSE = mean_squared_error(T.reshape(T.shape[0]), output)
print('MSE = {}'.format(MSE))
print('RMSE = {}'.format(np.sqrt(MSE)))
```

```
MSE = 0.020768431983385256
RMSE = 0.14411256705570563
```

Для проверки качества распознавания сформируем новое обучающее множество, изменив одно из значений R.

In [32]:
```
p1_k = np.linspace(0, 1, int(1 / 0.025), endpoint=True)

# Основной сигнал
p1 = np.sin(4 * np.pi * p1_k)

t1 = np.ones(len(p1_k)) * (-1)

p2_k = np.linspace(2.16, 4.04, int(4.04 / 0.025), endpoint=True)

# Сигнал, подлежащий распознаванию
p2 = np.cos(np.cos(p2_k) * p2_k * p2_k + 5 * p2_k)

t2 = np.ones(len(p2_k))

# Длительность основного сигнала
R = np.array([1, 4, 12])
```

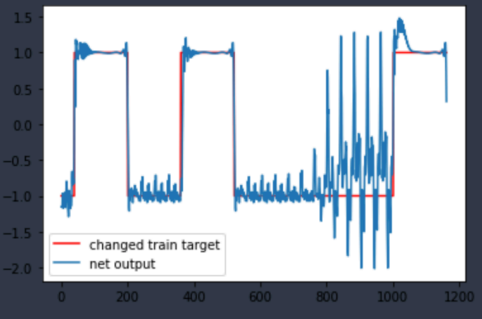
In [35]:
```
pl.plot(T2)
pl.plot(P2)
pl.legend(['train target', 'train input'])
pl.show()
```



Рассчитываем выход сети

In [36]:
```
# Simulate network
output2 = pyrenn.NNOut(P2, nn)
```

In [37]:
```
pl.plot(T2, color='red')
pl.plot(output2)
pl.legend(['changed train target', 'net output'])
pl.show()
```



In [38]:

```
output2[output2 >= 0] = 1.0
output2[output2 < 0] = -1.0

MSE = mean_squared_error(T2, output2)
print('MSE = {}'.format(MSE))
print('RMSE = {}'.format(np.sqrt(MSE)))
```

```
MSE = 0.10662080825451418
RMSE = 0.32652841875480637
```

Задание 3

Задание 3

Построить и обучить нелинейную авторегрессионную сеть с внешними входами (Non-linearAutoRegressive network with eXogeneous inputs, NARX), которая будет выполнять аппроксимациютраектории динамической системы, также выполнить многошаговый прогноз значений системы.

Входная последовательность формируется из входного управляющего сигнала u(k) и выходного сигнала y(k). Последовательность целевых выходов задает выходной сигнал y(k).

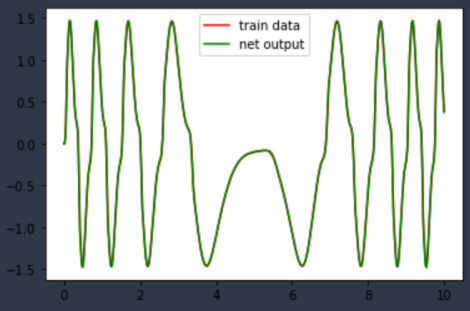
In [39]:

```
def u(k):
    return np.sin(k * k - 10 * k + 3)

k = np.linspace(0, 10, (int)(10/0.01))

y = [0.]
for i in k:
    y.append(y[-1] / (1 + y[-1]**2) + u(i)**3)
y = np.array(y[:-1])
```

Сравним выход сети с соответствующими эталонными подмножествами

In [45]:

```
pl.plot(k, y, color='red')
plt.plot(k, output, color='green')
plt.legend(['train data', 'net output'])
plt.show()
```



Число скрытого слоя 10. Одношаговый прогноз. Сеть имеет 2 выхода.

In [41]:
```
narx = NARX(MLPRegressor(hidden_layer_sizes=(10, 10)), solver='lbfgs', max_iter=600,
                         auto_order=2, exog_order=[2], exog_delay=[1])
```

In [42]:
```
input = u(k)[:, np.newaxis]
target = y
narx.fit(input, target)
```

Рассчитаем выход сети

In [43]:
```
output = narx.predict(input, target, step=1)
```

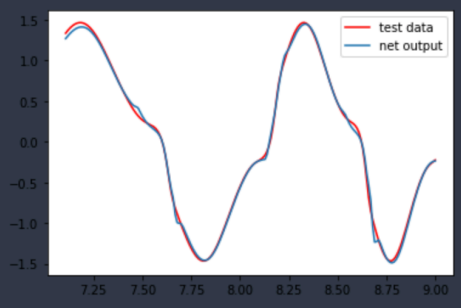
In [44]:
```
output[np.isnan(output)] = 0
MSE = mean_squared_error(target, output)
print('MSE = {}'.format(MSE))
print('RMSE = {}'.format(np.sqrt(MSE)))
```

```
MSE = 0.00010975944844669733
RMSE = 0.010476614359930279
```

In [48]:
```
outputTest[np.isnan(outputTest)] = 0
MSE = mean_squared_error(targetTest, outputTest)
print('MSE = {}'.format(MSE))
print('RMSE = {}'.format(np.sqrt(MSE)))
```

```
MSE = 0.04766468352839287
RMSE = 0.2183224301999061
```

In [49]:
```
pl.plot(xTest[shift:], yTest[shift:], color='red')
plt.plot(xTest[shift:], outputTest[shift:])
plt.legend(['test data', 'net output'])
plt.show()
```



Код программы.

```
#!/usr/bin/env python
# coding: utf-8


import neurolab as nl
import numpy as np
import numpy.matlib
from neupy import algorithms
import pylab as pl
from sklearn.metrics import mean_squared_error
import pyrenn
from matplotlib import pyplot as plt
import math
import neurolab as nl
import seaborn as sns
import random
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from sklearn.neural_network import MLPRegressor
from fireTS.models import NARX


# #### Задание 1
#
# Построить и обучить сеть прямого распространения с запаздыванием (Focused Time-
DelayNeural Network, FTDNN), которая будет аппроксимировать последовательность чисел
Вольфа, а также выполнить многошаговый прогноз.



df = pd.read_csv('SN_m_tot_V2.0.csv', sep=';', header=None)

df = df.iloc[:, 0:4]
df = df.drop([2], axis=1)
df.columns = ['Year', 'Month', 'Value']
df.Year = df.Year.astype(str)
df.Month = df.Month.astype(str)
df.index = pd.to_datetime(df.Year + '-' + df.Month)
df.drop(['Year'], axis=1, inplace=True)
df.drop(['Month'], axis=1, inplace=True)


df.plot();


date = '1761-04-01'
deep = 5
trainSize = 500
testSize = 100
validSize = 50


# Сглаживание траектории с помощью усредняющего фильтра smooth с шириной окна 12.


def smooth(a, windowWidth):
    # a: NumPy 1-D array containing the data to be smoothed
    # windowWidth: smoothing window size needs, which must be odd number,
    # as in the original MATLAB implementation
    out0 = np.convolve(a, np.ones(windowWidth, dtype=int), 'valid') / windowWidth
    r = np.arange(1, windowWidth-1,2)
    start = np.cumsum(a[:(windowWidth-1)-1])[::2] / r
    stop = (np.cumsum(a[:-(windowWidth-1):-1])[::2] / r)[::-1]
    return np.concatenate((start, out0, stop))

values = df.values.flatten()
widthWindow = 12
```

```
smoothValues = smooth(values, widthWindow)

shift = df.values.size - smoothValues.size
df.iloc[shift:] = smoothValues[:, np.newaxis]

df = df[df.index >= pd.to_datetime(date)]
df.plot()


trainSize = int(len(df) * 0.7)
train = df[:trainSize]
test = df[trainSize:]

trainData = train.values.squeeze()
xTrain = np.array([trainData[i:i + deep] for i in range(len(trainData) - deep)])
yTrain = train.iloc[deep:].values

testData = test.values.squeeze()
xTest = np.array([testData[i:i + deep] for i in range(len(testData) - deep)])
yTest = test.iloc[deep:].values


# Чисто скрытого слоя 8. Для обучения сети использовать метод Левенберга-Марквардта.


model = Sequential()
model.add(Dense(12, input_dim=deep, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1))

model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(xTrain, yTrain, epochs=60, batch_size=2, verbose=2)


mse1 = model.evaluate(xTrain, yTrain, verbose=0)
print('Test Score')
print('MSE: {}'.format(mse1))
print('RMSE: {}'.format(np.sqrt(mse1)))

mse2 = model.evaluate(xTest, yTest, verbose=0)
print('Test Score')
print('MSE: {}'.format(mse2))
print('RMSE: {}'.format(np.sqrt(mse2)))


# Рассчитать выход сети для обучающего подмножества.

predictTrain = model.predict(xTrain)

MSE = mean_squared_error(yTrain, predictTrain)
print('MSE = {}'.format(MSE))
print('RMSE = {}'.format(np.sqrt(MSE)))

plt.plot(yTrain, color='red')
plt.plot(predictTrain, color='blue')
plt.legend(['train data', 'net output'])
plt.show()


# Рассчитать выход сети для тестового подмножества.

predictTest = model.predict(xTest)

MSE = mean_squared_error(yTest, predictTest)
print('MSE = {}'.format(MSE))
print('RMSE = {}'.format(np.sqrt(MSE)))

plt.plot(yTest, color='red')
plt.plot(predictTest, color='green')
plt.legend(['test data', 'net output'])
plt.show()
```

```
df['pred'] = np.concatenate([df[:deep].values.flatten(),
                             predictTrain.flatten(),
                             df[trainSize:trainSize+deep].values.flatten(),
                             predictTest.flatten()])


df.plot()


# #### Задание 2
#
# Построить и обучить сеть прямого распространения с распределенным
запаздыванием(Distributed Time-Delay Neural Network, TDNN), которая будет выполнять
распознавание динамического образа.

# Обучающее множество взять из лабораторной работы №5.

p1_k = np.linspace(0, 1, int(1 / 0.025), endpoint=True)

# Основной сигнал
p1 = np.sin(4 * np.pi * p1_k)

t1 = np.ones(len(p1_k)) * (-1)

p2_k = np.linspace(2.16, 4.04, int(4.04 / 0.025), endpoint=True)

# Сигнал, подлежащий распознаванию
p2 = np.cos(np.cos(p2_k) * p2_k * p2_k + 5 * p2_k)

t2 = np.ones(len(p2_k))

# Длительность основного сигнала
R = np.array([1, 4, 7])


p2 = p2.reshape(1, p2.shape[0])
t2 = t2.reshape(1, t2.shape[0])


P = np.concatenate((numpy.matlib.repmat(p1, 1, R[0]), p2,
                    numpy.matlib.repmat(p1, 1, R[1]), p2,
                    numpy.matlib.repmat(p1, 1, R[2]), p2), axis=1).reshape(-1, 1)

T = np.concatenate((numpy.matlib.repmat(t1, 1, R[0]), t2,
                    numpy.matlib.repmat(t1, 1, R[1]), t2,
                    numpy.matlib.repmat(t1, 1, R[2]), t2), axis=1).reshape(-1, 1)



T = T.reshape(T.shape[0])
P = P.reshape(P.shape[0])


# Задать задержки[0 : 4] для входного и скрытого слоев. Число нейронов скрытого слоя
задать равным 8.

# Система с 1 входом и выходом
nn = pyrenn.CreateNN([1, 8, 1], dIn=[4], dIntern=[4])


# Обучаем сеть

nn = pyrenn.train_LM(P, T, nn, E_stop=1e-5, k_max=100)


# Рассчитываем выход сети

output = pyrenn.NNOut(P, nn)


pl.plot(T, color='red')
plt.plot(output)
```

```
plt.legend(['train target', 'net output'])
plt.show()


# Преобразуем предсказанные значения.


output[output >= 0] = 1.0
output[output < 0] = -1.0

MSE = mean_squared_error(T.reshape(T.shape[0]), output)
print('MSE = {}'.format(MSE))
print('RMSE = {}'.format(np.sqrt(MSE)))


# Для проверки качества распознавания сформируем новое обучающее множество, изменив одно
из значений R.


p1_k = np.linspace(0, 1, int(1 / 0.025), endpoint=True)

# Основной сигнал
p1 = np.sin(4 * np.pi * p1_k)

t1 = np.ones(len(p1_k)) * (-1)

p2_k = np.linspace(2.16, 4.04, int(4.04 / 0.025), endpoint=True)

# Сигнал, подлежащий распознаванию
p2 = np.cos(np.cos(p2_k) * p2_k * p2_k + 5 * p2_k)

t2 = np.ones(len(p2_k))

# Длительность основного сигнала
R = np.array([1, 4, 12])


p2 = p2.reshape(1, p2.shape[0])
t2 = t2.reshape(1, t2.shape[0])

P2 = np.concatenate((numpy.matlib.repmat(p1, 1, R[0]), p2,
                     numpy.matlib.repmat(p1, 1, R[1]), p2,
                     numpy.matlib.repmat(p1, 1, R[2]), p2), axis=1).reshape(-1, 1)

T2 = np.concatenate((numpy.matlib.repmat(t1, 1, R[0]), t2,
                     numpy.matlib.repmat(t1, 1, R[1]), t2,
                     numpy.matlib.repmat(t1, 1, R[2]), t2), axis=1).reshape(-1, 1)


T2 = T2.reshape(T2.shape[0])
P2 = P2.reshape(P2.shape[0])

pl.plot(T2)
pl.plot(P2)
pl.legend(['train target', 'train input'])
pl.show()


# Рассчитываем выход сети


# Simulate network
output2 = pyrenn.NNOut(P2, nn)


pl.plot(T2, color='red')
pl.plot(output2)
pl.legend(['changed train target', 'net output'])
pl.show()

output2[output2 >= 0] = 1.0
output2[output2 < 0] = -1.0
```

```
MSE = mean_squared_error(T2, output2)
print('MSE = {}'.format(MSE))
print('RMSE = {}'.format(np.sqrt(MSE)))


# #### Задание 3
#
# Построить и обучить нелинейную авторегрессионную сеть с внешними входами (Non-
linearAutoRegressive network with eXogeneous inputs, NARX), которая будет выполнять
аппроксимациютраектории динамической системы, также выполнить многошаговый прогноз
значений системы.

# Входная последовательность формируется из входного управяющего сигнала u(k) и выходного
сигнала y(k). Последовательность целевых выходов задает выходной сигнал y(k).

def u(k):
    return np.sin(k * k - 10 * k + 3)

k = np.linspace(0, 10, (int)(10/0.01))

y = [0.]
for i in k:
    y.append(y[-1] / (1 + y[-1]**2) + u(i)**3)
y = np.array(y[:-1])


# Обучающее, тестовое и валидационное множество - число временных отсчетов 700, 200 и 97
соответственно.

delay = 3
trainSize = 700
testSize = 200
validSize = 97
shift=10

xTrain = k[:700]
xTest = k[700:900]
xValid = k[900:997]

yTrain = y[:700]
yTest = y[700:900]
yValid = y[900:997]


# Число скрытого слоя 10. Одношаговый прогноз. Сеть имеет 2 выхода.

narx = NARX(MLPRegressor(hidden_layer_sizes=(10, 10)), solver='lbfgs', max_iter=600,
                         auto_order=2, exog_order=[2], exog_delay=[1])


input = u(k)[:, np.newaxis]
target = y
narx.fit(input, target)


# Рассчитаем выход сети

output = narx.predict(input, target, step=1)


output[np.isnan(output)] = 0
MSE = mean_squared_error(target, output)
print('MSE = {}'.format(MSE))
print('RMSE = {}'.format(np.sqrt(MSE)))


# Сравним выход сети с соответствующими эталонными подмножествами

pl.plot(k, y, color='red')
plt.plot(k, output, color='green')
plt.legend(['train data', 'net output'])
```

```
plt.show()


# Выполним многошаговый прогноз: рассчитаем выход сети для тестового подмножества.

narx = NARX(MLPRegressor(hidden_layer_sizes=(10, 10)), solver='lbfgs', max_iter=600,
                         auto_order=2, exog_order=[3], exog_delay=[3])
narx.fit(input, target)


# Рассчитаем выход сети

inputTest = u(xTest)[:, np.newaxis]
targetTest = yTest
outputTest = narx.predict(inputTest, targetTest, step=3)

outputTest[np.isnan(outputTest)] = 0
MSE = mean_squared_error(targetTest, outputTest)
print('MSE = {}'.format(MSE))
print('RMSE = {}'.format(np.sqrt(MSE)))

pl.plot(xTest[shift:], yTest[shift:], color='red')
plt.plot(xTest[shift:], outputTest[shift:])
plt.legend(['test data', 'net output'])
plt.show()
```

Выводы:

Выполнив лабораторную работу, я научилась применять динамические сети для предсказания значений временного ряда и выполнения многошагового прогноза, распознавания динамических образов и аппроксимации траектории динамической систем.