Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №8 по курсу «Нейроинформатика»

Студент: М. А. Бронников
Преподаватель: Н. П. Аносова
Группа: М8О-407Б

Дата:
Оценка:
Подпись:

Москва, 2021

Динамические сети

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.

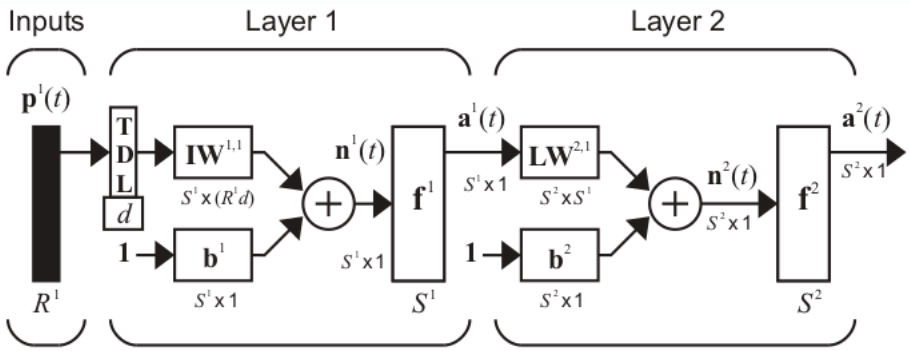
Вариант №4:

1. 10/1848
2. $g(k) = \sin(\sin(k)k^3 - 10),\ k \in [1.56, 3.12]$
3. $u(k) = \sin(k^2 - 7k)$

1 Структура моделей

Focused Time-Delay Neural Network, FTDNN

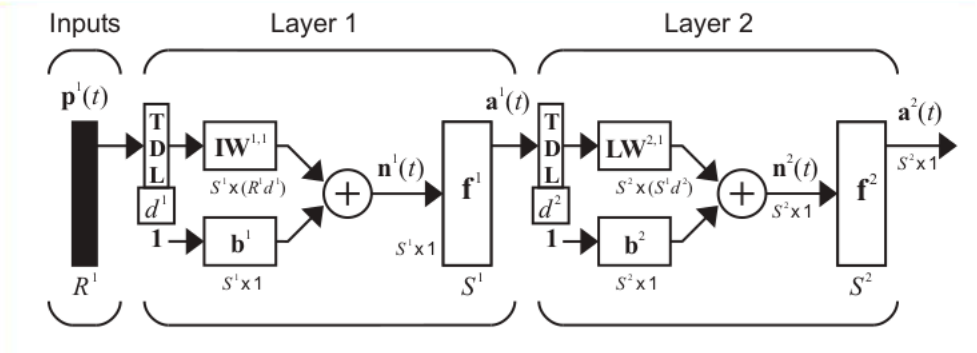
Сеть прямого распространения с запаздыванием похожа на адаптивный фильтратор, однако имеет 2 полносвязанных слоя вместо одного, что делает ее более гибкой.



Такая система позволяет успешно справляться с задачами, связанными с динамическими процессами, такими как распознование звукового или видео потока.

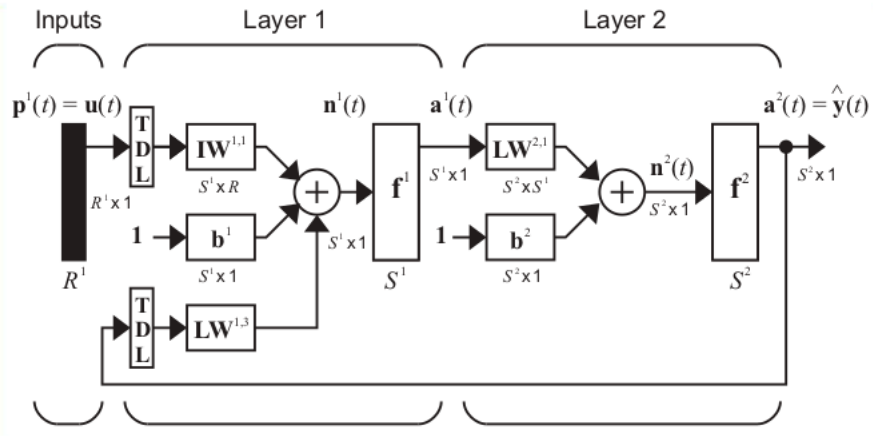
Distributed Time-Delay Neural Network, TDNN

В отличие от *FTDNN* сеть прямого распространения с распределенным запаздыванием имеет *TDL* блок не только перед первым слоем, н и перед вторым.



Non-linearAutoRegressive network with eXogeneous inputs, NARX

Нелинейная авторегрессионная сеть с внешними входами отличается от базовой *FTDNN* тем, что выход первого слоя формируется не только из перемножения матрицы весов на входные значения, но и из перемножения другой матрицы весов с выходом *TDL* блока, сформированного из предыдущих выходов нейронной сети.



Такая структура дает сети адаптироваться не только за счет известных значений текущего момента, но и предсказанных сетью до этого, что позволяет ей справляться с управлением динамическими системами, где требуется адаптация.

2 Ход работы

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.

Формируем входные данные:

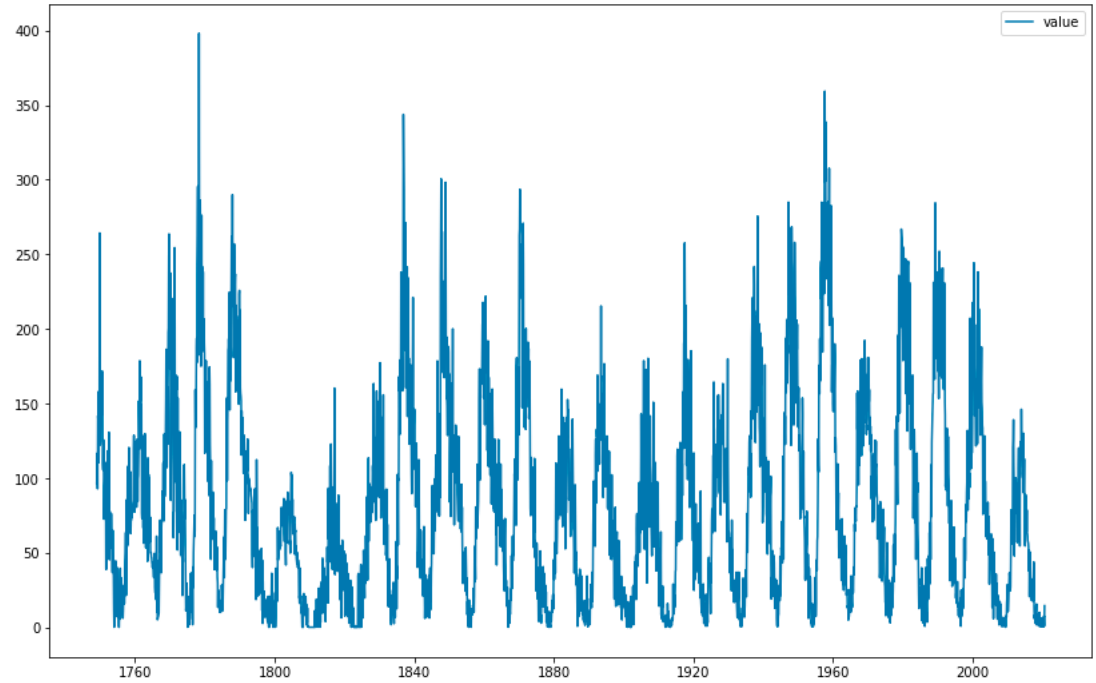
```
df = pd.read_csv('wolfie.csv', sep=';', header=None)
df.head()
```

	0	1	2	3	4	5	6
0	1749	1	1749.042	96.7	-1.0	-1	1
1	1749	2	1749.123	104.3	-1.0	-1	1
2	1749	3	1749.204	116.7	-1.0	-1	1
3	1749	4	1749.288	92.8	-1.0	-1	1
4	1749	5	1749.371	141.7	-1.0	-1	1

```
df = df.iloc[:, 0:4]
df.drop([2], axis=1, inplace=True)

df[0] = df[0].astype(str)
df[1] = df[1].astype(str)
df.index = pd.to_datetime(df[0] + '-' + df[1])

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

df.head()
```

	3
1749-01-01	96.7
1749-02-01	104.3
1749-03-01	116.7
1749-04-01	92.8
1749-05-01	141.7

```
plt.figure(figsize=(14, 9))
plt.plot(df, label="value")
plt.legend()
plt.show()
```




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

Усредняющий фильтр **smooth** с заданной шириной окна:

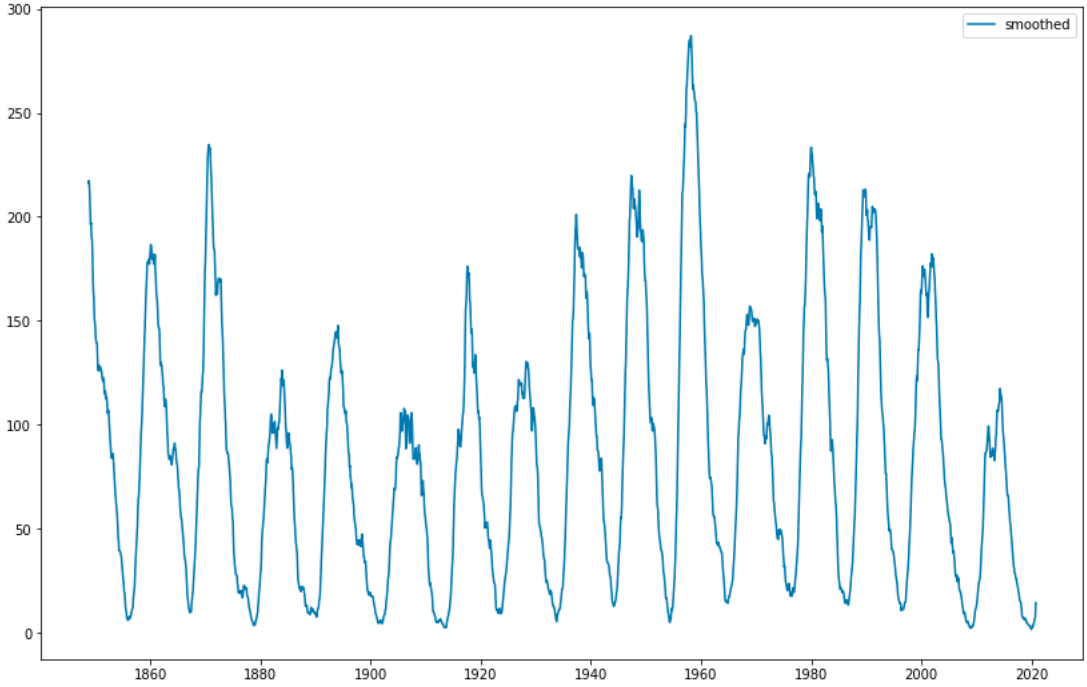
```
def smooth(a, windowWidth):
    out0 = np.convolve(a, np.ones(windowWidth, dtype=int), 'valid') / windowWidth
    r = np.arange(1, windowWidth-1,2)
    start = np.cumsum(a[:(windowWidth-1)-1])[::2] / r
    stop = (np.cumsum(a[:-(windowWidth-1):-1])[::2] / r)[::-1]
    return np.concatenate((start, out0, stop))
```

Зададим ширину окна и сгладим данные:

```
widthWindow = 12
smoothValues = smooth(values, widthWindow)
```

Обновим данные в датафрейме на сглаженные и зададим начало в соответствии с вариантом.

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

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

plt.figure(figsize=(14, 9))
plt.plot(df, label="smoothed")
plt.legend()
plt.show()
```


smoothed
300
250
200
150
100
50
0
1860
1880
1900
1920
1940
1960
1980
2000
2020

Сформируем обучающее множество (сразу с задержками) и тестовое множество для многошагового прогноза.

```
deep = 5
```

```
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
```

Создадим и обучим нейронную сеть:

```
model = Sequential()
model.add(Dense(12, input_dim=deep, activation='tanh'))
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=0)
```

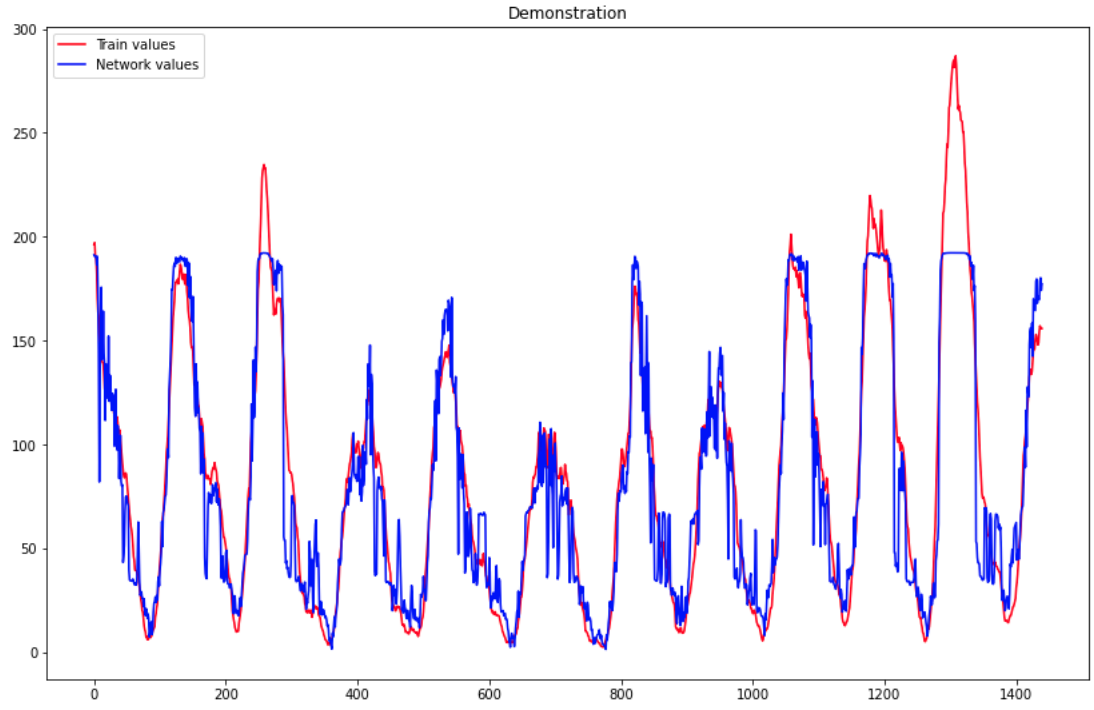
```
<tensorflow.python.keras.callbacks.History at 0x7f8feab36210>
```

Выход сети:

```
predictTrain = model.predict(xTrain)

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

```
MSE = 445.16319754391355
RMSE = 21.098890907910622
```

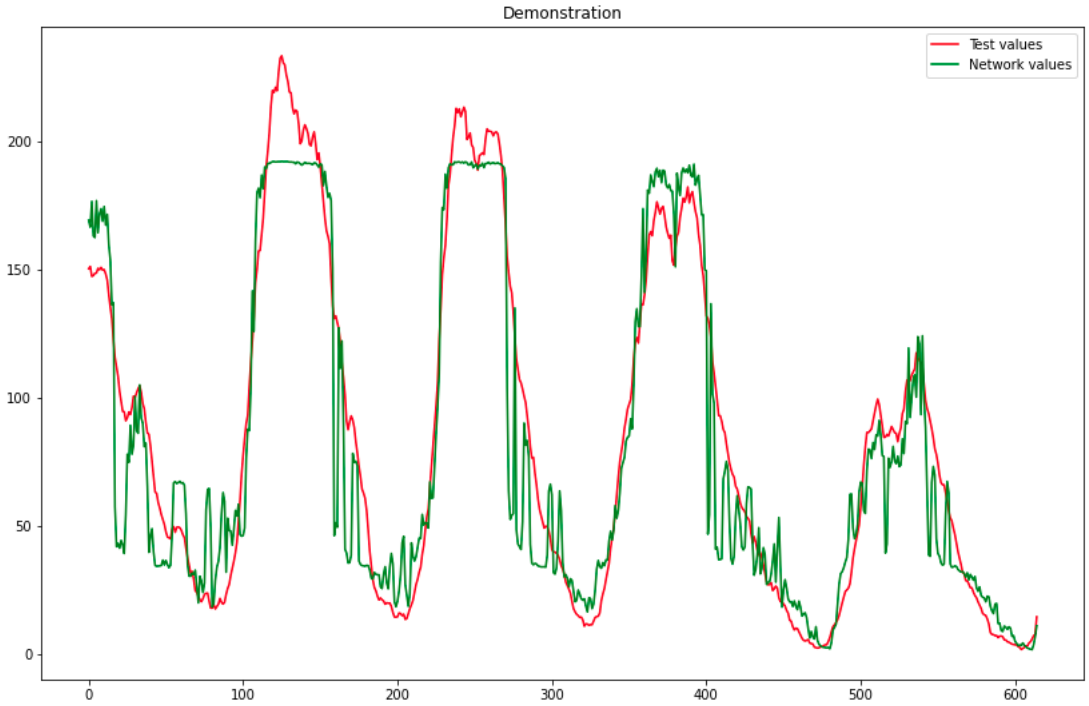
```
plt.figure(figsize=(14, 9))

plt.plot(yTrain, color='red')
plt.plot(predictTrain, color='blue')

plt.legend(['Train values', 'Network values'])
plt.title("Demonstration")
plt.show()
```




```
predictTest = model.predict(xTest)

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

```
MSE = 484.71250713043275
RMSE = 22.01618738861097
```

```
plt.figure(figsize=(14, 9))

plt.plot(yTest, color='red')
plt.plot(predictTest, color='green')

plt.legend(['Test values', 'Network values'])
plt.title("Demonstration")
plt.show()
```


Demonstration
Test values
Network values
200
150
100
50
0
0
100
200
300
400
500
600

Задание №2

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

Сгенерируем набор из точек:

```
p1_k = np.linspace(0, 1, int(1 / 0.025), endpoint=True)
p2_k = np.linspace(a, b, int((b - a) / 0.025), endpoint=True)

p1 = np.sin(4 * np.pi * p1_k) # main signal
p2 = g(p2_k) # signal to detect

t1 = np.ones(len(p1_k)) * (-1)
t2 = np.ones(len(p2_k))

p2 = p2.reshape(1, -1)
t2 = t2.reshape(1, -1)
```

```
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.ravel() # makes 1-D array
P = P.ravel()
```

Создадим и обучим сеть с задержками, равными 5:

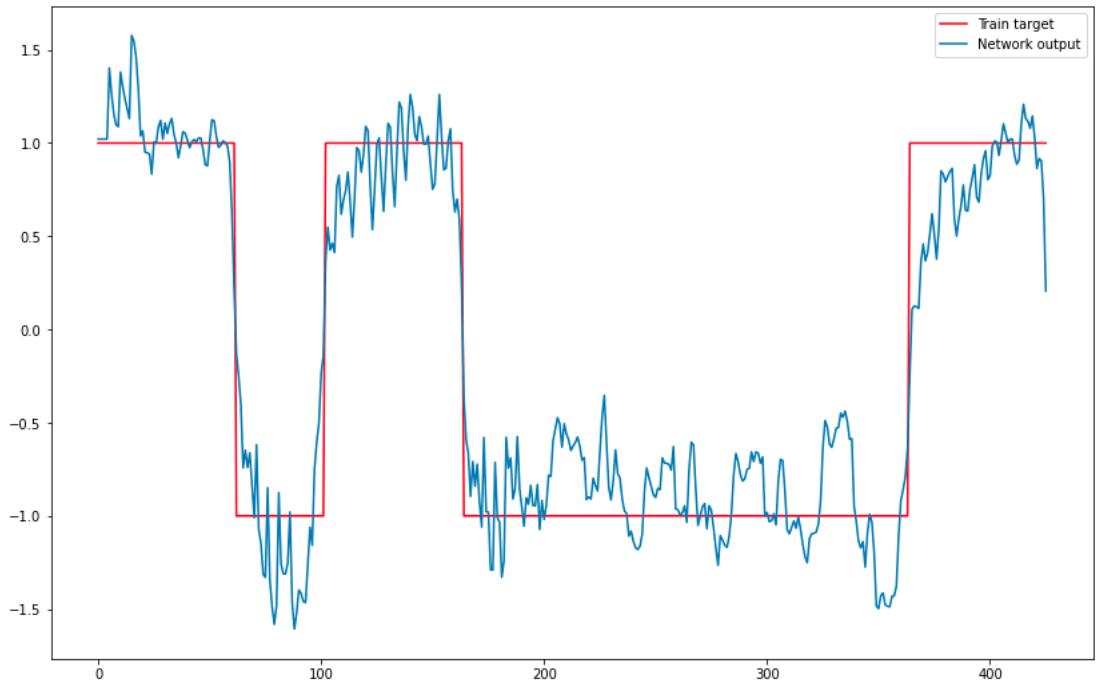
```
nn = pyrenn.CreateNN([1, 8, 1], dIn=[5], dIntern=[5])

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

```
Maximum number of iterations reached
```

Выход сети:

```
output = pyrenn.NNOut(P, nn)
```

```
plt.figure(figsize=(14, 9))
plt.plot(T, color='red')
plt.plot(output)
plt.legend(['Train target', 'Network output'])
plt.show()
```



```
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)))
```

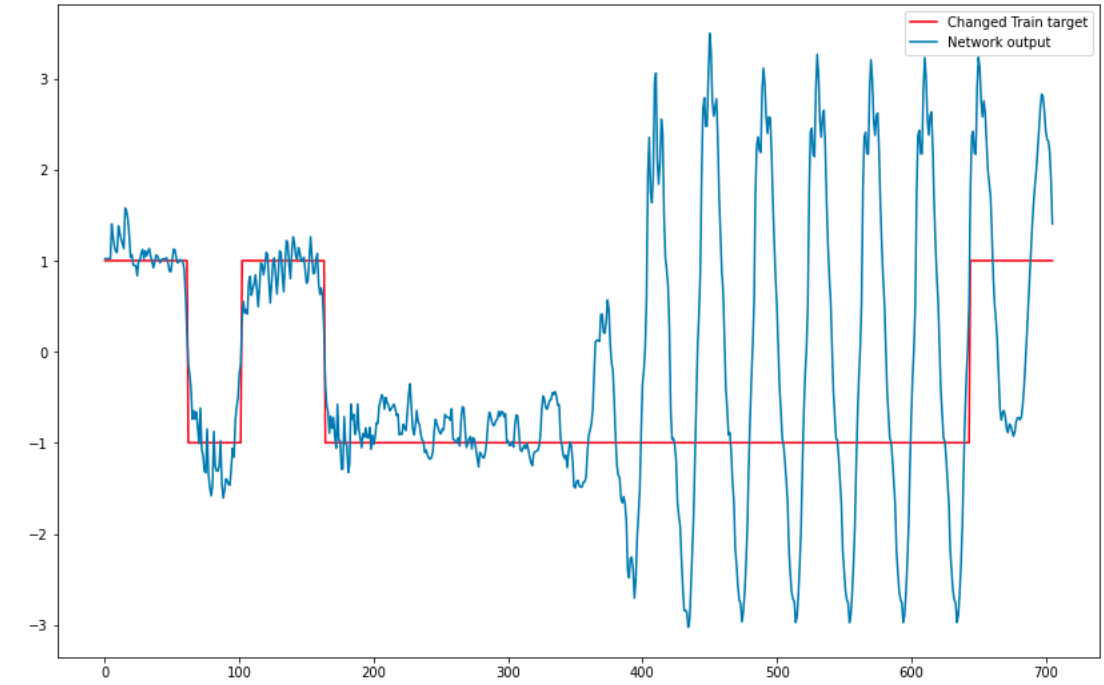
```
MSE = 0.009389671361502348
RMSE = 0.09690031662230185
```

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

```
P2 = np.concatenate((numpy.matlib.repmat(p1, 1, R[0]), p2,
                     numpy.matlib.repmat(p1, 1, R[1]), p2,
                     numpy.matlib.repmat(p1, 1, 12), 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, 12), t2), axis=1).reshape(-1, 1)

T2 = T2.ravel() # makes 1-D array
P2 = P2.ravel()
```

```
output2 = pyrenn.NNOut(P2, nn)
```

```
plt.figure(figsize=(14, 9))

plt.plot(T2, color='red')
plt.plot(output2)
plt.legend(['Changed Train target', 'Network output'])
plt.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)))
```

```
MSE = 0.8781869688385269
RMSE = 0.9371163048621697
```

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.

Сформируем данные:

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

y = f(k)
```

```
inpt = u(k)[:, np.newaxis]
target = y
```

```
# data parameters:
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]
```

Создадим и обучим сеть:

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

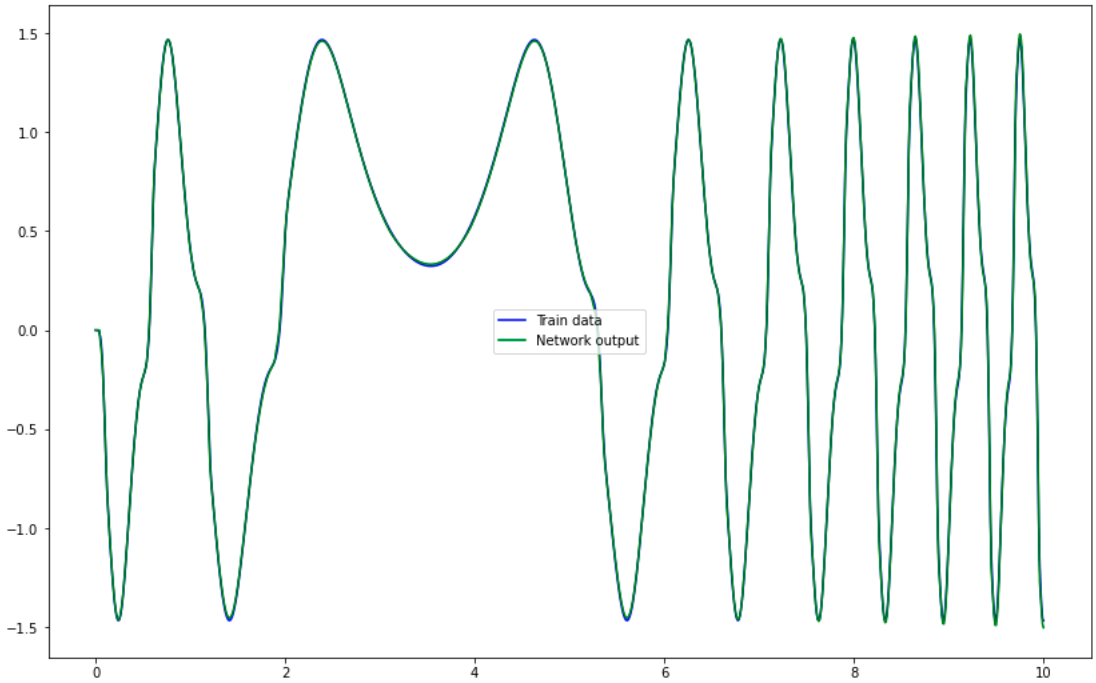
```
narx.fit(inpt, target)
```

Аппроксимируем систему:

```
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)))
```

```
MSE = 0.0002189926411106784
RMSE = 0.014798399951031138
```

```
plt.figure(figsize=(14, 9))

plt.plot(k, y, color='blue')
plt.plot(k, output, color='green')
plt.legend(['Train data', 'Network output'])
plt.show()
```



Выполним многошаговый прогноз: обучим и рассчитаем выход сети для тестового подмножества.

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

```
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)))
```

```
MSE = 0.00559475479824596
RMSE = 0.07479809354686763
```

```
plt.figure(figsize=(14, 9))

plt.plot(xTest[shift:], yTest[shift:], color='blue')
plt.plot(xTest[shift:], outputTest[shift:])
plt.legend(['Test data', 'Network output'])
plt.show()
```

3 Выводы

Выполнив восьмую лабораторную работу по курсу «Нейроинформатика», я познакомился с Динамическими сетями которые отлично справляются с динамическими данными, т.е. когда входные данные, подаваемые в сеть упорядоченны по какому-либо динамическому параметру, такому как время.

У этих сетей есть множество применений, начиная от обработки звукового сигнала с микроконтроллера и заканчивая системами управления летательными средствами. Поэтому изученные знания еще наверняка понадобятся мне в моей будущей карьере.

Эта работа является последней в этом курсе и я считаю она достойно завершила столь интересный и прикладной темой этот увлекательный предмет, который познакомил нас со всевозможными нейронными сетями с огромным спектром их применений.