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

Кафедра вычислительной математики и программирования

Лабораторная работа № 2

по курсу «Нейроинформатика».

Тема: «Линейная нейронная сеть. Правило обучения Уидроу-Хоффа».

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 линейной нейронной сети и алгоритмов ее обучения, применение сети в задачах аппроксимации и фильтрации.

Основные этапы работы.

1. Использовать линейную нейронную сеть с задержками для аппроксимации функции. В качестве метода обучения использовать адаптацию.

2. Использовать линейную нейронную сеть с задержками для аппроксимации функции и выполнения многошагового прогноза.

3. Использовать линейную нейронную сеть в качестве адаптивного фильтра для подавления помех. Для настройки весовых коэффициентов использовать метод наименьших квадратов.

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) и matplotlib (для графиков) при помощи командной оболочки Jupyter Notebook.

Сценарий выполнения работы.

Задание 1

Используем линейную нейронную сеть Adaline с задержками для аппроксимации функции.

In [5]:
```
delay = 5
learningRate = 0.01
epochs = 50
```

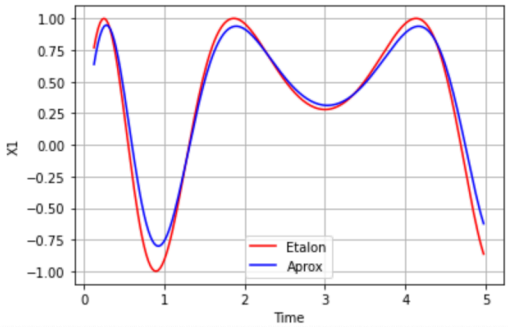
In [6]:
```
w1, out, error = Adaline(x1, epochs, delay, learningRate)
```

In [7]:
```
print('MSE = {}'.format(round(error, 5)))
print('RMSE = {}'.format(round(np.sqrt(error), 5)))
```

```
MSE = 0.05642
RMSE = 0.23753
```

In [8]:
```
plt.grid(True)
plt.xlabel('Time')
plt.ylabel('X1')
plt.plot(t1[delay:], x1[delay:], 'r', label='Etalon')
plt.plot(t1[delay:], out[delay:], 'b', label='Aprox')
plt.legend()
```

Out[8]:
```
<matplotlib.legend.Legend at 0x7fb76a6b10b8>
```



Задание 2

Используем линейную нейронную сеть Adaline с задержками для аппроксимации функции и выполнения многошагового прогноза.

In [9]:
```
delay = 3
learningRate = 0.01
epochs = 600
```

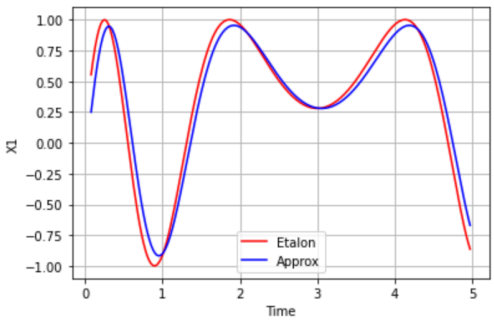
In [10]:
```
w2, out2, error2 = Adaline(x1, epochs, delay, learningRate)
```

In [11]:
```
print('Error = {}'.format(round(error2, 5)))
```

```
Error = 0.03891
```

In [12]:
```
plt.grid(True)
plt.xlabel('Time')
plt.ylabel('X1')
plt.plot(t1[delay:], x1[delay:], 'r', label='Etalon')
plt.plot(t1[delay:], out2[delay:], 'b', label='Approx')
plt.legend()
```

Out[12]:
```
<matplotlib.legend.Legend at 0x7fb76a7482b0>
```



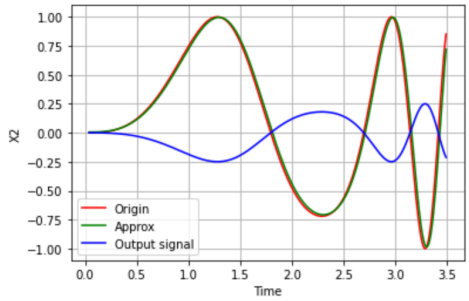
Многошаговый прогноз

```
In [13]: test_time = get_t(5, 5 + 10 * h1, h1)
         target_x = input_f1(test_time)

In [14]: w2_, out2_, error2_ = Adaline(target_x, epochs, delay, learningRate)

In [15]: print('Error = {}'.format(round(error2_, 5)))

         Error = 0.23096
```

Задание 3

Используем линейную нейронную сеть Adaline в качестве адаптивного фильтра для подавления помех.

```
In [16]: delay = 4
         learningRate = 0.0055
         epochs = 60

In [17]: w3, out3, error3 = Adaline(x2, epochs, delay, learningRate)

In [18]: print('Error = {}'.format(round(error3, 5)))

         Error = 0.00439

In [19]: plt.grid(True)
         plt.xlabel('Time')
         plt.ylabel('X2')
         plt.plot(t2[delay:], x2[delay:], 'r', label='Origin')
         plt.plot(t2[delay:], out3[delay:], 'g', label='Approx')
         plt.plot(t2[delay:], y2[delay:], 'b', label='Output signal')
         plt.legend()

Out[19]: <matplotlib.legend.Legend at 0x7fb76a960198>
```



Входные данные и результаты

Вход	Выход
Аппроксимация $x = \sin(t^2 - 6t + 3), \quad t \in [0, 6],\ h = 0.025$ $x = \sin(\sin(t)t^2), \quad t \in [0, 3.5],\ h = 0.01$	**Ошибка = 0.05642** **RMSE = 0.23753**
Аппроксимация и многошаговый прогноз $x = \sin(t^2 - 6t + 3), \quad t \in [0, 6],\ h = 0.025$ $x = \sin(\sin(t)t^2), \quad t \in [0, 3.5],\ h = 0.01$	**Ошибка = 0.003891** **Ошибка (многошаговый прогноз) = 0.23096**

Подавление помех $x = \sin\left(t^2 - 6t + 3\right), \quad t \in [0, 6],\ h = 0.025$ $x = \sin\left(\sin(t)t^2\right), \quad t \in [0, 3.5],\ h = 0.01$	**Ошибка = 0.00439**

Код программы.

```
import numpy as np
import matplotlib.pyplot as plt
import math


def input_f1(t):
    return np.sin(t*t - 6*t + 3)

def input_f2(t):
    return np.sin(np.sin(t) * t*t)

def get_t(a, b, h):
    return np.arange(a, b, h)

def output_f(t):
    return 1/4 * np.sin(np.sin(t) * t*t - np.pi)


h1 = 0.025
h2 = 0.01

t1 = get_t(0, 5, h1)
t2 = get_t(0, 3.5, h2)

x1 = input_f1(t1)
x2 = input_f2(t2)

y1 = output_f(t1)
y2 = output_f(t2)


def Adaline(target, epochs, delay, learningRate):
    weights = np.random.sample(delay)
    bias = np.random.random()
    out = np.zeros(target.size)
    errMin = 10**(-10)
    err = 99999
    errors = []

    for epoch in range(epochs):
        if err < errMin:
            for i in range(len(target) - delay):
                x = np.array([target[i + j] for j in range(delay)], dtype=np.float32)
                out[i + delay] = np.dot(x, weights) + bias
            break
        else:
            err = 0
            for i in range(len(target) - delay):
                x = np.array([target[i + j] for j in range(delay)], dtype=np.float32)
                out[i + delay] = np.dot(x, weights) + bias
```

```
            err = (target[i + delay] - out[i + delay])
            errors.append(err * err)
            weights += learningRate * np.dot(x, err)
            bias += learningRate * err
    if (learningRate > 10**(-6)):
        learningRate -= 0.0001
return weights, out, sum(errors) / len(errors)
```

Выводы:

Выполнив лабораторную работу, я научилась строить линейную нейронную сеть, применяя правило обучения Уидроу-Хоффа, использовать сеть для аппроксимации функции, выполнения многошагового прогноза и использовать линейную нейронную сеть в качестве адаптивного фильтра для фильтрации помех.