**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

**Лабораторная работа № 2**

по курсу «Нейроинформатика»

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

Студент: Куликов А.В.

Группа: М80-408Б-17

Преподаватель: Аносова Н.П.

Дата: 22 октября 2020

Оценка:

Москва, 2020

**Цель работы:**

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:**

Процессор: AMD Ryzen 5 Mobile 3550H

Объем оперативной памяти: 8 Гб

**Программное обеспечение:**

Python 3.8.5

**Сценарий выполнения работы:**

### Этап №1

Реализация линейной НС ADALINE (файл linear\_NN.py)

import numpy as np

class ADALINE:

    def fit\_one(self, x, y):

        prediction = self.predict(x)

        error = y - prediction

        self.W += self.lr \* np.dot(x[np.newaxis].T, error[np.newaxis])

        self.b += self.lr \* error

    def fit(self, x\_train, y\_train, learning\_rate=0.01, epochs=20):

        self.lr = learning\_rate

        x\_dim = x\_train.shape[1]

        y\_dim = y\_train.shape[1]

        self.W = np.random.sample((x\_dim, y\_dim))

        self.b = np.random.random(y\_dim)

        for i in range(epochs):

            for x, y in zip(x\_train, y\_train):

                self.fit\_one(x, y)

    def predict(self, x\_to\_predict):

        res = np.dot(x\_to\_predict, self.W) + self.b

        return res

    def weights(self):

        return self.W

    def biases(self):

        return self.b

Данные для задания (вариант 8)

t\_begin1 = 1

t\_end1 = 6

h1 = 0.025

def phi1(t):

    return np.sin(t\*t - 10\*t + 3)

t\_begin2 = 0

t\_end2 = 3.5

h2 = 0.01

def phi2(t):

    return np.sin(-2\*t\*t + 7\*t)

def phi(t):

    return 1/8 \* np.sin(-2\*t\*t + 7\*t - m.pi)

Вспомогательные функции

def gen\_data(x, D):

    x\_train\_list = []

    y\_train\_list = []

    for i in range(D, x.shape[0]-1):

        x\_train\_list.append(x[i-D:i])

        y\_train\_list.append([x[i+1]])

    x\_train = np.array(x\_train\_list)

    y\_train = np.array(y\_train\_list)

    return x\_train, y\_train

def gen\_filter\_data(x, y, D):

    x\_train\_list = []

    y\_train\_list = []

    for i in range(D, x.shape[0]):

        x\_train\_list.append(x[i-D:i])

        y\_train\_list.append([y[i]])

    x\_train = np.array(x\_train\_list)

    y\_train = np.array(y\_train\_list)

    return x\_train, y\_train

Метрики

def mse(A, B):

    return ((A - B) \*\* 2).sum() / A.shape[0]

def rmse(A, B):

    return np.sqrt(mse(A, B))

Задание №1

print('--- TASK 1 ---')

n1 = int((t\_end1 - t\_begin1) / h1) + 1

t1 = np.linspace(t\_begin1, t\_end1, num=n1)

x1 = phi1(t1)

D = 5

x\_train, y\_train = gen\_data(x1, D)

approximator = ADALINE()

approximator.fit(x\_train, y\_train, learning\_rate=0.01, epochs=50)

print("WEIGHTS:")

print(approximator.weights())

print("BIASES:")

print(approximator.biases())

predicted = approximator.predict(x\_train)

expected = y\_train

rmse\_metric = rmse(predicted, expected)

print(f'RMSE: {rmse\_metric}')

fig, ax = plt.subplots()

ax.plot(t1[D:-1], expected, 'g', label='reference')

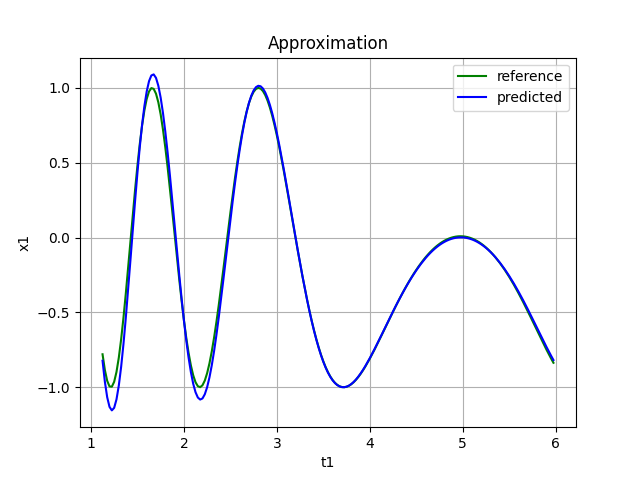
ax.plot(t1[D:-1], predicted, 'b', label='predicted')

ax.set(xlabel='t1', ylabel='x1', title='Approximation')

ax.grid()

ax.legend()

plt.savefig('ex1.png')



Вывод программы:

--- TASK 1 ---

WEIGHTS:

[[-0.45275971]

[-0.19147798]

[ 0.23901133]

[ 0.13306742]

[ 1.23267737]]

BIASES:

[-0.01425813]

RMSE: 0.052782836573605295

Задание №2

print('--- TASK 2 ---')

D = 3

x\_train, y\_train = gen\_data(x1, D)

predictor = ADALINE()

predictor.fit(x\_train, y\_train, epochs=600)

print("WEIGHTS:")

print(approximator.weights())

print("BIASES:")

print(approximator.biases())

predicted = predictor.predict(x\_train)

expected = y\_train

error = predicted - expected

fig, ax = plt.subplots()

ax.plot(t1[D:-1], expected, 'g', label='reference')

ax.plot(t1[D:-1], predicted, 'b', label='predicted')

ax.plot(t1[D:-1], error, 'r', label='error')

ax.set(xlabel='t1', ylabel='x1', title='Approximation after training')

ax.grid()

ax.legend()

plt.savefig('ex2.png')

rmse\_metric = rmse(predicted, expected)

print(f'RMSE on train data: {rmse\_metric}')

print(f'{D=}')

n\_append = 10

t1 = np.append(t1, np.array([t\_end1 + (i+1) \* h1 for i in range(n\_append)]))

x1 = phi1(t1)

x, y = gen\_data(x1, D)

predicted = predictor.predict(x)

expected = y

fig, ax = plt.subplots()

ax.plot(t1[-n\_append:], expected[-n\_append:], 'g', label='reference')

ax.plot(t1[-n\_append:], predicted[-n\_append:], 'b', label='predicted')

ax.set(xlabel='t1', ylabel='x1', title='Prediction for next 10 measurments')

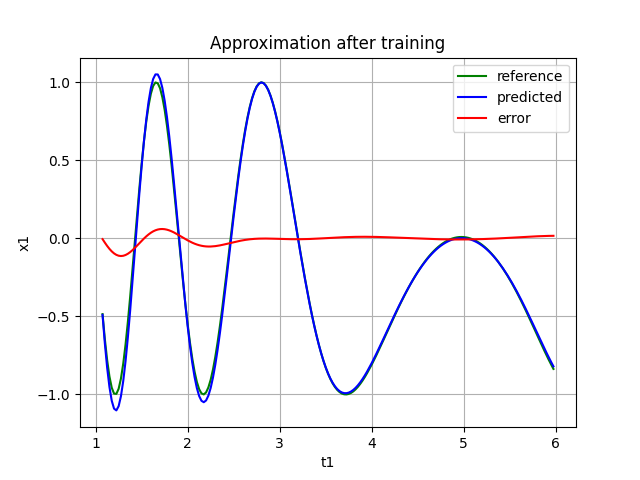
ax.grid()

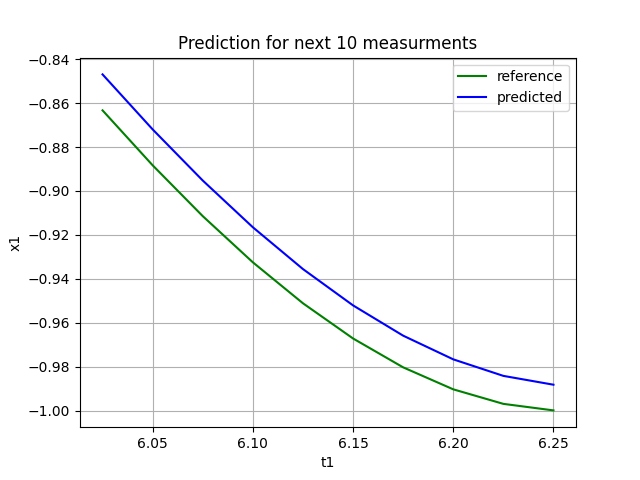
ax.legend()

rmse\_metric = rmse(expected[-n\_append:], predicted[-n\_append:])

print(f'RMSE on appended data: {rmse\_metric}')

plt.savefig('ex3.png')





Вывод программы:

WEIGHTS:

[[-0.45275971]

[-0.19147798]

[ 0.23901133]

[ 0.13306742]

[ 1.23267737]]

BIASES:

[-0.01425813]

RMSE on train data: 0.030947483817309215

D=3

RMSE on appended data: 0.014881409671749012

Задание №3

print('--- TASK 3 ---')

n2 = int((t\_end2 - t\_begin2) / h2) + 1

print(f'{n2=}')

t2 = np.linspace(t\_begin2, t\_end2, num=n2)

x2 = phi2(t2)

y2 = phi(t2)

D = 4

x\_train, y\_train = gen\_filter\_data(x2, y2, D)

print(x\_train[:5])

print(y\_train[:5])

adaptive\_filter = ADALINE()

adaptive\_filter.fit(x\_train, y\_train, learning\_rate=0.01, epochs=50)

print("WEIGHTS:")

print(approximator.weights())

print("BIASES:")

print(approximator.biases())

predicted = adaptive\_filter.predict(x\_train)

expected = y\_train

error = predicted - expected

fig, ax = plt.subplots()

ax.plot(t2[D:], expected, 'g', label='reference')

ax.plot(t2[D:], predicted, 'b', label='filtered')

ax.plot(t2[D:], error, 'r', label='error')

ax.set(xlabel='t1', ylabel='x1', title='')

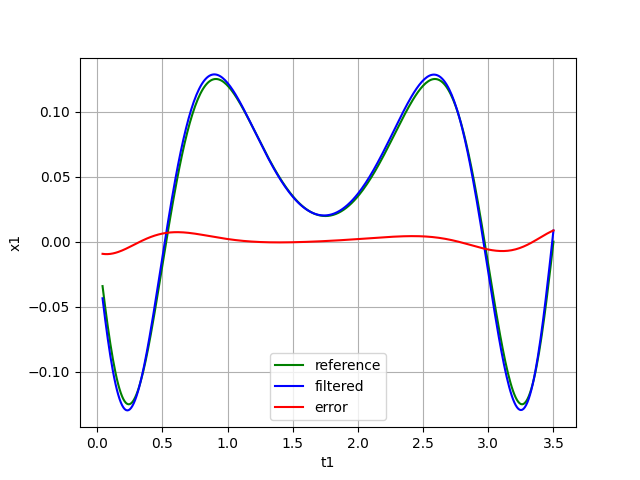
ax.grid()

ax.legend()

plt.savefig('ex4.png')

rmse\_metric = rmse(predicted, expected)

print(f'RMSE: {rmse\_metric}')

****

--- TASK 3 ---

n2=351

[[0. 0.06974334 0.1387509 0.20669911]

[0.06974334 0.1387509 0.20669911 0.27327886]

[0.1387509 0.20669911 0.27327886 0.33819668]

[0.20669911 0.27327886 0.33819668 0.4011757 ]

[0.27327886 0.33819668 0.4011757 0.46195657]]

[[-0.03415986]

[-0.04227458]

[-0.05014696]

[-0.05774457]

[-0.06503726]]

WEIGHTS:

[[-0.59673427]

[-0.06422892]

[ 0.29050225]

[ 0.22414986]

[ 1.10419541]]

BIASES:

[-0.01512013]

RMSE: 0.00421140575986584

**Выводы:**

Линейные сети по своей структуре аналогичны персептрону и отличаются лишь функцией активации. В случае линейной нейронной сети она, как ни странно, линейна. Таким образом выход линейной сети может принимать любое значение, в то время как выход персептрона ограничен значениями 0 или 1.

К сожалению, персептрону, и ADALINE присущ один и тот же минус — они могут работать только с линейно разделимыми классами.