**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: Якимович А.И.

Группа: 80-408Б

Вариант: 20

Оценка:

Москва, 2018

Постановка задачи.

Целью работы является исследование свойств линейной нейронной сети и алгоритмов ее обучения, применение сети в задачах аппроксимации и фильтрации.

Ход 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.

Запуск программы.

Чтобы воспользоваться программой, необходимо запустить программу на любом интерпретаторе Python версии 3.x.x.

Входные данные и результаты.

Вариант 20.

|  |  |
| --- | --- |
| **Вход** | **Выход** |
| **C:\Users\now20\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Скриншот 24-12-2018 114809.png**  Аппроксимация | 0.166  **C:\Users\now20\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Figure_1.png** |
| **C:\Users\now20\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Скриншот 24-12-2018 114809.png**  Аппроксимация и многошаговый прогноз | **Ошибка = 0.04** |
| **C:\Users\now20\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Скриншот 24-12-2018 114809.png**  Подавление помех | Ошибка = 0.436 |

Выводы:

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

Программа работает правильно, прошла все тесты, и ее можно легко применять для задач любой размерности без существенного переписывания кода.

Исходный код.

import numpy as np

import matplotlib.pyplot as plt

import sys

import math

def out2F(t):

return (1/4) \* math.cos(-math.cos(t) \* t\*\*2 + t + 2\*math.pi)

def in1F(t):

return math.sin(-2\*math.sin(t) \* t\*\*2 + 7\*t)

def in2F(t):

return math.cos(math.cos(t)\*t\*\*2 + t)

out2F = np.vectorize(out2F)

in1F = np.vectorize(in1F)

in2F = np.vectorize(in2F)

in2 = (np.linspace(0.5, 4.0, num = (4.0 - 0.5) / 0.01))

out2 = out2F(in2)

in2 = in2F(in2)

'''

'''

ls1\_x = np.linspace(0.5, 3.2, num = 1 + (3.2 - 0.5) / 0.01)

ls1 = in1F(ls1\_x)

#out1 = in1[3::4]

#in1 = np.delete(in1, np.arange(3, in1.size, 4))

w1 = np.array([1, 1, 1, 1, 1], float)

b1 = 0

learn\_rate1 = 0.01

number\_of\_era1 = 5

e1 = 99999

out1 = np.zeros(ls1.size)

for i in range(50):

e1 = 0

for i in range(ls1.size - number\_of\_era1):

x1 = np.array([ls1[i], ls1[i + 1], ls1[i + 2], ls1[i + 3], ls1[i + 4]], float)

out1[i + number\_of\_era1] = w1.dot(x1) + b1

w1 -= learn\_rate1 \* (out1[i + number\_of\_era1] - ls1[i + number\_of\_era1]) \* x1

b1 -= learn\_rate1 \* (out1[i + number\_of\_era1] - ls1[i + number\_of\_era1])

e1 += ((out1[i + number\_of\_era1] - ls1[i + number\_of\_era1]) \*\* 2) / 2

print("e1 = ", e1)

plt.subplot(311)

plt.plot(ls1\_x, ls1, color = 'red')

plt.plot(ls1\_x, out1, color = 'blue')

###################################################################

ls2\_x = np.linspace(0.5, 3.21, num = 11 + (3.2 - 0.5) / 0.01)

ls2 = in1F(ls2\_x)

w2 = np.array([1, 1, 1], float)

b2 = 0

learn\_rate2 = 0.01

number\_of\_era2 = 3

e2\_m = 10\*\*(-6)

e2 = 99999

out2 = np.zeros(ls2.size)

#while e1 > e1\_m:

for i in range(600):

if e2 < e2\_m:

for i in range(ls2.size - number\_of\_era2):

x2 = np.array([ls2[i], ls2[i + 1], ls2[i + 2]], float)

out2[i + 3] = w2.dot(x2) + b2

break

else:

e2 = 0

for i in range(ls2.size - number\_of\_era2):

x2 = np.array([ls2[i], ls2[i + 1], ls2[i + 2]], float)

out2[i + 3] = w2.dot(x2) + b2

w2 -= learn\_rate2 \* (out2[i + number\_of\_era2] - ls2[i + number\_of\_era2]) \* x2

b2 -= learn\_rate2 \* (out2[i + number\_of\_era2] - ls2[i + number\_of\_era2])

e2 += ((out2[i + number\_of\_era2] - ls2[i + number\_of\_era2]) \*\* 2) / 2

print("e2 = " , e2)

plt.subplot(312)

plt.plot(ls2\_x, ls2, color = 'red')

plt.plot(ls2\_x, out2, color = 'blue')

##########################################################################

ls3\_x = np.linspace(0.5, 4., num = 1 + (4. - 0.5) / 0.01)

ls3 = in2F(ls3\_x)

ls3\_out = out2F(ls3\_x)

w3 = np.array([0, 0, 0, 0, 0, 0, 0], float)

b3 = 0

learn\_rate3 = 0.001

number\_of\_era3 = 7

e3\_m = 10\*\*(-6)

e3 = 99999

out3 = np.zeros(ls3.size)

for i in range(1000):

e3 = 0

for i in range(ls3.size - number\_of\_era3):

x3 = np.array([ls3[i], ls3[i + 1], ls3[i + 2], ls3[i + 3], ls3[i+4], ls3[i+5], ls3[i +6]], float)

out3[i + number\_of\_era3] = w3.dot(x3) + b3

w3 -= learn\_rate3 \* (out3[i + number\_of\_era3] - ls3\_out[i + number\_of\_era3]) \* x3

b3 -= learn\_rate3 \* (out3[i + number\_of\_era3] - ls3\_out[i + number\_of\_era3])

e3 += ((out3[i + number\_of\_era3] - ls3\_out[i + number\_of\_era3]) \*\* 2) / 2

print("e3 = " , e3)

plt.subplot(313)

plt.plot(ls3\_x, ls3\_out, color = 'red')

plt.plot(ls3\_x, out3, color = 'blue')

plt.plot(ls3\_x, ls3, color = 'green')

plt.show()