**Министерство образования и науки Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технический университет**

**имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

ФАКУЛЬТЕТ «Информатика и системы управления» (ИУ)

КАФЕДРА «Информационная безопасность» (ИУ8)

**ИНТЕЛЛЕКТУАЛЬНЫЕ ТЕХНОЛОГИИ ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ**

**ЛАБОРАТОРНАЯ РАБОТА № 3**

**Изучение алгоритма обратного распространения ошибки (метод Back Propagation)**

**Вариант № 7**

**Преподаватель:** Коннова Н.С.

**Студент:** Кустов И. А.

**Группа:** ИУ8-61

Оглавление

[Цель работы 3](#_Toc5313934)

[Постановка задачи 3](#_Toc5313935)

[Практическая часть 4](#_Toc5313936)

[Вывод 10](#_Toc5313937)

[Приложения 10](#_Toc5313938)

# **Цель работы** – исследовать функционирование многослойной нейронной сети (МНС) прямого распространения и ее обучение методом обратного распространения ошибки

**Постановка задачи** – на примере МНС архитектуры N – J – M реализовать ее обучение методом BP, проведя настройку весов нейронов скрытого и выходного слоев.

# **Практическая часть**

Обучение проводится на архитектуре 1-2-1. Требуется обучить МНС на восстановление по входному вектору

х = (1, 4)

целевого значения

t = -0.2

с погрешностью не более 0.001.

Начальные веса скрытого и выходного слоев примем равные 1, норму обучения равной 0.3.

В таблице 1 приведены результаты обучения МНС методом B. В таблице 2 приведены веса скрытого и выходного слоев на каждой эпохе. На рисунке 1 изображен график зависимости суммарной ошибки от номера эпохи.

Таблица 1

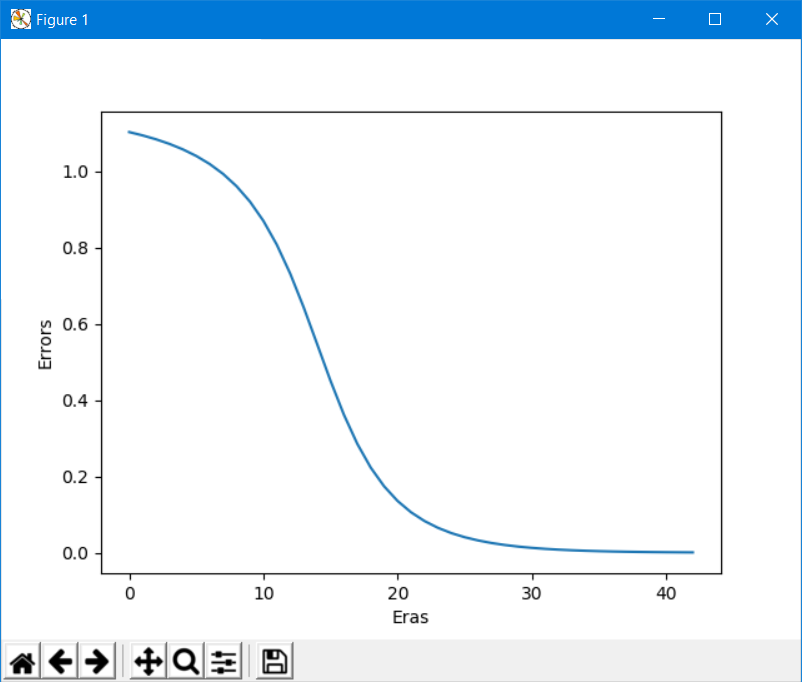
**Параметры НС на последовательных эпохах (пороговая ФА)**

|  |  |  |
| --- | --- | --- |
| **k** | **Выходной вектор y** | **Суммарная ошибка E** |
| 0 | 0.903 | 1.1027 |
| 1 | 0.894 | 1.094 |
| 2 | 0.884 | 1.0838 |
| 3 | 0.872 | 1.0717 |
| … | … | … |
| 40 | -0.1987 | 0.0013 |
| 41 | -0.1990 | 0.001 |
| 42 | -0.1992 | 0.0008 |

Таблица 2

**Веса скрытого и выходного слоев на эпохах обучения**

|  |  |  |  |
| --- | --- | --- | --- |
| **k** | **Веса скрытого слоя** | | **Веса выходного слоя** |
| 1 нейрон | 2 нейрон |
| 0 | [1, 1] | [1, 1] | [1, 1, 1] |
| 1 | [0.9996, 0.9984] | [1, 0.998] | [0.969, 0.97, 0.97 ] |
| 2 | [0.9992, 0.9967] | [0.999, 0.997] | [0.936, 0.937, 0.937] |
| 3 | [0.9987, 0.9949] | [0.999, 0.995] | [0.901, 0.902, 0.902] |
| … |  |  |  |
| 40 | [0.99253, 0.97018] | [0.99253, 0.97018] | [-0.14671, -0.13, -0.13] |
| 41 | [0.99254, 0.97019] | [0.99254, 0.97019] | [-0.1469, -0.13018, -0.13018] |
| 42 | [0.99255, 0.9702] | [0.99255, 0.9702] | [-0.14705, -0.13033, -0.13033] |



**Рисунок 1.** График зависимости суммарной ошибки от количества эпох

# **Вывод**

В процессе выполнения лабораторной работы было проведено обучение МНС методом BP на архитектуре 1-2-1. Такая МНС имеет достаточно высокую эффективность. Так же по каждому нейрону проходит информация, связанная именно с ним, что позволяет реализовать распараллеливание.

# **Приложения**

Ссылка на гитхаб с кодом:

<https://github.com/Kustov-Ilya/lab_work_II/blob/master/lab4.py>

Листинг №1

import numpy as np

import os

Function = lambda net: (1-np.exp(-net))/(1+np.exp(-net))

Deriv\_func = lambda net: (1 - Function(net)\*\*2)/2

class Neuron:

def \_\_init\_\_(self, number\_w, nu):

self.w = np.ones(number\_w)

self.nu = nu

self.net = 0

self.sigma = 0

def Get\_out(self, x):

self.net = np.dot(self.w, x)

return Function(self.net)

def Get\_mistake(self, sigma):

self.sigma = Deriv\_func(self.net)\*sigma

return [self.sigma\*wi for wi in self.w[1:]]

def Correct\_w(self, x):

self.w+= [self.nu\*self.sigma\*xi for xi in x]

self.net, self.sigma = 0, 0

class Neuron\_Net:

def \_\_init\_\_(self, x, J, t, nu, eps):

self.era = 0

self.x = x

self.t = t

self.J = J

self.eps = eps

self.hide\_level = [Neuron(len(x), nu) for \_ in range(J)]

self.output\_level = [Neuron(J+1, nu) for \_ in range(len(t))]

def Era(self):

self.era+=1

hide\_out = [1] + [hide\_neuron.Get\_out(self.x)

for hide\_neuron

in self.hide\_level]

output\_out = [output\_neuron.Get\_out(hide\_out)

for output\_neuron

in self.output\_level]

primary\_sigma = [self.t[i] - output\_out[i]

for i in range(len(self.t))]

output\_mistake = [output\_neuron.Get\_mistake(primary\_sigma[i])

for i, output\_neuron

in enumerate(self.output\_level)]

output\_mistake = [sum(line[i] for line in output\_mistake) for i in range(self.J)]

\_ = [hide\_neuron.Get\_mistake(output\_mistake[i])

for i, hide\_neuron

in enumerate(self.hide\_level)]

\_ = [hide\_neuron.Correct\_w(self.x)

for hide\_neuron

in self.hide\_level]

\_ = [output\_neuron.Correct\_w(hide\_out)

for output\_neuron

in self.output\_level]

squared\_mistake = sum(i\*\*2 for i in primary\_sigma)\*\*0.5

print(f"Era: {self.era} y: {', '.join(str(np.around(i, 3)) for i in output\_out)} Mistake: {np.around(squared\_mistake, 4)}", end = ' ')

print("Hide", end = ' ')

for i, neuron in enumerate(self.hide\_level):

print(f"{i+1}. {np.around(neuron.w, 3)}", end = ' ')

print("Output", end = ' ')

for i, neuron in enumerate(self.output\_level):

print(f"{i+1}. {np.around(neuron.w, 3)}", end = ' ')

print('\n')

return squared\_mistake

def Work(self):

while self.Era() > self.eps:

pass

if \_\_name\_\_ == "\_\_main\_\_":

Nn = Neuron\_Net([1, 4], 2, [-0.2], 0.3, 0.001)

Nn.Work()

os.system("pause")