|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение высшего образования**  **Московский государственный технический университет**  **имени Н.Э. Баумана** |

Факультет: Информатика и системы управления

Кафедра: Информационная безопасность

**Интеллектуальные технологии информационной безопасности**

**Лабораторная работа №4 на тему:**

«Изучение алгоритма обратного распространения ошибки

(метод Back Propagation)»

Вариант 4

Студент: Евула А. С.

Группа: ИУ8-63

Москва 2021

Цель работы:

Исследовать функционирование многослойной нейронной сети (МНС) прямого распространения и ее обучение методом обратного распространения ошибки.

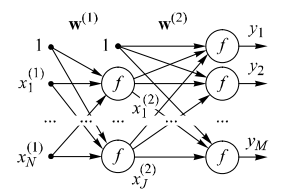
Постановка задачи:

На примере МНС архитектуры N-J-M реализовать ее обучение методом ВР, проведя настройку весов нейронов скрытого и выходного слоев.

**Архитектура**: 1-2-1

**X**: (1, 3)

**10t**: 1



Ход работы

Требуется обучить МНС на восстановление по входному вектору

X = (1, 3)

Целевого вектора T = (0.1)

С погрешностью не более

Исходные веса примем нулевыми, норма обучения равна 1 (исходя из тестов, лучшее время обучения).

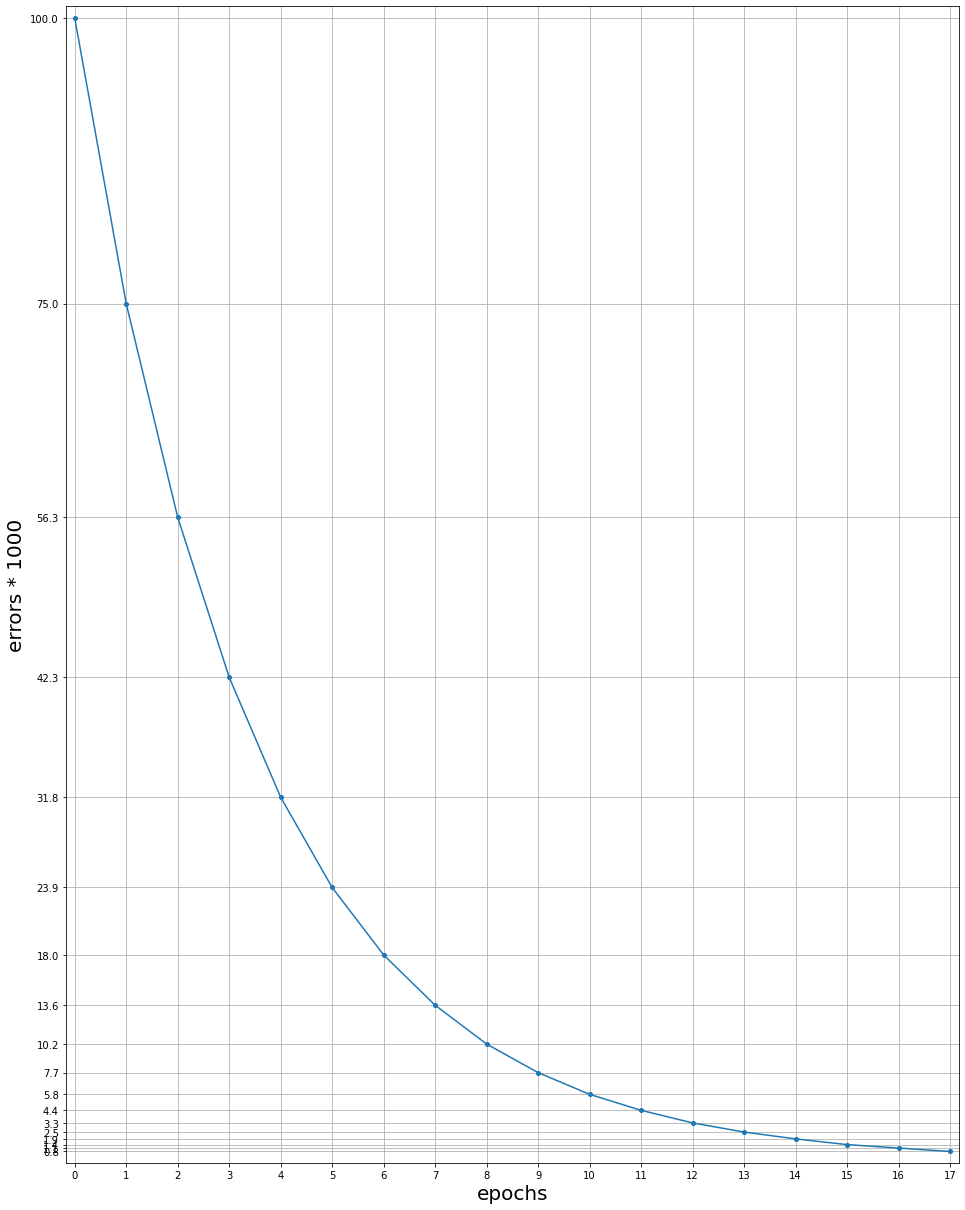


Рисунок - график зависимости ошибки от эпохи

По исходным данным, нейронная обучается за ~17 эпох.

Исходный код программы и ее вывод представлены в Приложении.

Приложение

Исходный код программы (НС)

# Copyright 2021, Evula A. S., All rights reserved.

# IC8-63 BMSTU

from math import exp, sqrt

from matplotlib import pyplot as plt

class NN:

def \_\_init\_\_(self):

self.N = 1

self.x = [1, 3]

self.J = 2

self.W\_hidden = [[0] \* (self.J) for \_ in range(self.N+1)]

self.M = 1

self.W\_exit = [[0] \* (self.M) for \_ in range(self.J+1)]

self.t = [0.1]

self.Rate = 1

self.MinError = 0.001

self.MaxEpoch = 1000

def train(self):

epoch = 0

E = 1

self.res = [[], [], []]

while E > self.MinError:

if epoch > self.MaxEpoch:

return print('ERR: epoch')

res = self.tick()

E, y = round(res[0], 4), res[1]

self.res[0].append(epoch)

self.res[1].append(round(E,4)\*1000)

self.res[2].append(y)

epoch += 1

return 0

def tick(self):

# Calculate y

net\_hidden = [0] \* self.J

for j in range(self.J):

net\_hidden[j] = self.W\_hidden[0][j]

for i in range(self.N):

net\_hidden[j] += self.W\_hidden[i+1][j] \* self.x[i+1]

out = []

for j in range(self.J):

out.append(self.f\_net(net\_hidden[j]))

net\_exit = [0] \* self.M

for m in range(self.M):

net\_exit[m] = self.W\_exit[0][m]

for j in range(self.J):

net\_exit[m] += self.W\_exit[j+1][m] \* out[j]

y = []

for m in range(self.M):

y.append(self.f\_net(net\_exit[m]))

# Errors

d\_exit = []

for m in range(self.M):

d\_exit.append(self.d\_f\_net(net\_exit[m]) \* (self.t[m]-y[m]))

d\_hidden = []

for j in range(self.J):

tmp = 0

for m in range(self.M):

tmp += d\_exit[m] \* self.W\_exit[j][m]

d\_hidden.append(self.d\_f\_net(net\_exit[m]) \* tmp)

# Correction

for i in range(self.N+1):

for j in range(self.J):

self.W\_hidden[i][j] += self.Rate\*self.x[i]\*d\_hidden[j]

out = [1] + out

for j in range(self.J+1):

for m in range(self.M):

self.W\_exit[j][m] += self.Rate\*out[j]\*d\_exit[m]

E = 0

for m in range(self.M):

E += (self.t[m]-y[m])\*(self.t[m]-y[m])

return sqrt(E), y

def f\_net(self, net):

return (1-exp(-net))/(1+exp(-net))

def d\_f\_net(self, net):

return 0.5\*(1 - self.f\_net(net)\*self.f\_net(net))

def print(self, graph=False, table=False):

print(f't: {self.t}')

print(end='\nВеса скрытого слоя\n')

for j in range(len(self.W\_hidden)):

print(f'w{j}(1) = [', end='')

for i in range(len(self.W\_hidden[j])):

print(round(self.W\_hidden[j][i], 5), end='')

if i != len(self.W\_hidden[j])-1:

print(', ', end='')

print(']')

print(end='\nВеса выходного слоя\n')

for j in range(len(self.W\_exit)):

print(f'w{j}(2) = [', end='')

for i in range(len(self.W\_exit[j])):

print(round(self.W\_exit[j][i], 5), end='')

if i != len(self.W\_exit[j])-1:

print(', ', end='')

print(']')

if graph:

plt.rcParams["figure.figsize"] = (16,22)

plt.plot(self.res[0], self.res[1], marker='o', markersize=4)

plt.subplots\_adjust(bottom = 0.15)

plt.grid()

plt.xlabel('epochs', fontsize=20)

plt.ylabel('errors \* 1000', fontsize=20)

plt.xticks(self.res[0])

plt.yticks(self.res[1])

plt.margins(0.01)

plt.show()

if table:

print('Эпоха K\tВыходной Y')

for i in range(len(self.res[0])):

rounded\_y = []

for m in range(self.M):

rounded\_y.append(round(self.res[2][i][m], 4))

print(f' {self.res[0][i]}\t {rounded\_y}')

def main():

nn = NN()

if nn.train() != None:

nn.print(graph=True, table=True)

if \_\_name\_\_ == '\_\_main\_\_':

main()

Вывод программы

t: [0.1]

Веса скрытого слоя

w0(1) = [0.00846, 3e-05]

w1(1) = [0.02539, 9e-05]

Веса выходного слоя

w0(2) = [0.19936]

w1(2) = [0.00205]

w2(2) = [0.0]

Эпоха K Ошибка E Выходной Y

0 100.0 [0.0]

1 75.0 [0.025]

2 56.300000000000004 [0.0437]

3 42.3 [0.0577]

4 31.8 [0.0682]

5 23.900000000000002 [0.0761]

6 18.0 [0.082]

7 13.6 [0.0864]

8 10.200000000000001 [0.0864]

9 7.7 [0.0864]

10 5.8 [0.0898]

11 4.4 [0.0923]

12 3.3 [0.0942]

13 2.5 [0.0956]

14 1.9 [0.0967]

15 1.4 [0.0975]

16 1.1 [0.0981]

17 0.8 [0.0992]