|  |  |
| --- | --- |
| 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**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ «Информатика и системы управления» (ИУ)

КАФЕДРА «Информационная безопасность» (ИУ8)

Отчёт

по лабораторной работе № 1

по дисциплине «Интеллектуальные технологии информационной безопасности»

**Тема: «Исследование однослойных неронных сетей на примере моделирования булевых выражений»**

Вариант 3

Выполнил: Горбачев А.А.,

студент группы ИУ8-63

Проверил: Волосова Н.К.,

преподаватель каф. ИУ8

г. Москва,

2021 г.

# Цель работы

Исследовать функционирование простейшей нейронной сети (НС) на базе нейрона с нелинейной функцией активации и обучить ее по правилу Видроу-Хоффа.

# Условия

Условия согласно варианту 3:

Булевая функция от 4ех переменных:

Функция активации 1:

Функция активации 2:

Норма обучения

# Аналитическая часть

Алгоритм функционирования НС с пороговой ФА имеет вид

Где net – сетевой(комбинированный) вход, а y – реальный выход НС.

Алгоритм функционирования НС с логической ФА выглядит следующим образом:

Где out – сетевой (недискретизированный) выход НС

Для необученной НС ее реальный выход у в общем случае отличается от целевого выхода t, представляющего собой значения заданной БФ нескольких переменных

, т. е. имеется хотя бы один набор сигналов (х1 ,х2 ,х3 ,х4), для которого ошибка

Правило Видроу – Хоффа (дельта правило):

На каждой эпохе k суммарная квадратичная ошибка E(k) равна расстоянию Хемминга между векторами целевого и реального выходов по всем входным векторам x1, x2, x3, x4

# Ход работы

Получим нейросетевую модель булевой функции (таблица 1)

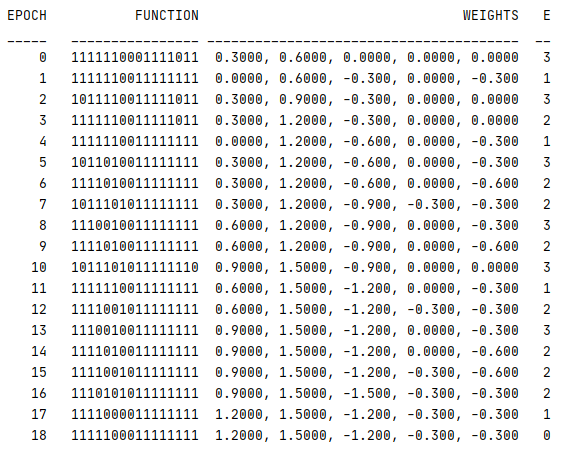
Таблица 1. Таблица истинности БФ

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **F** | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| **X4** | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| **X3** | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| **X2** | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| **X1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

На начальном шаге весовые коэффициенты беруться в виде:

Используя ФА 1. Динамика НС представлена в таблице 2, график суммарной ошибки приведен на рисунке 2.

Таблица 2. Параметры НС на послежовательных эпохах (Пороговая ФА)



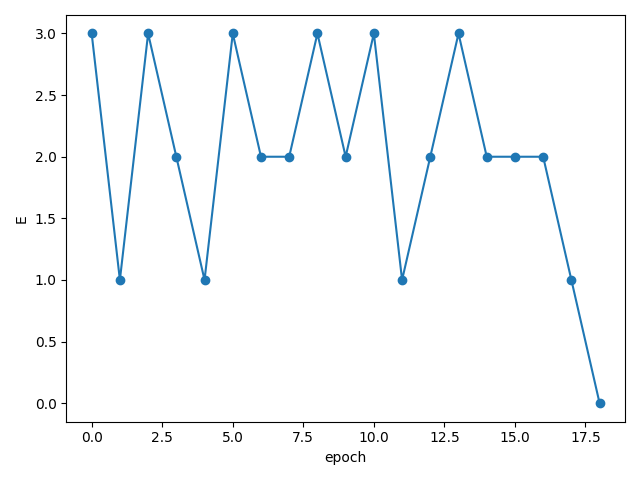
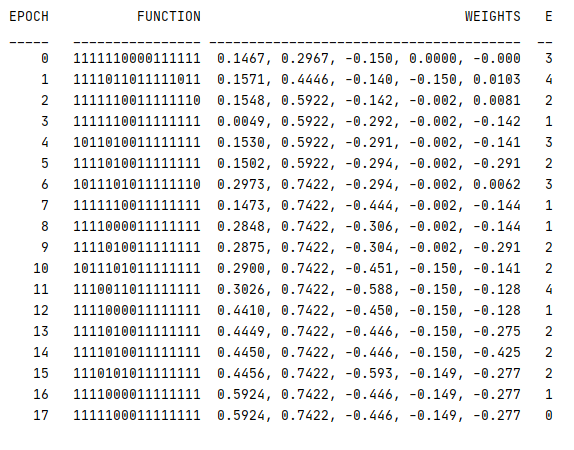
**

Рисунок 2 – График суммарной ошибки НС по эпохам обучения (пороговая ФА)

Используя логическую ФА и считая получим результаты приведенные в таблице 3 и на рисунке 3.

Таблица 3. Параметры НС на последовательных эпохах (Логическая ФА)



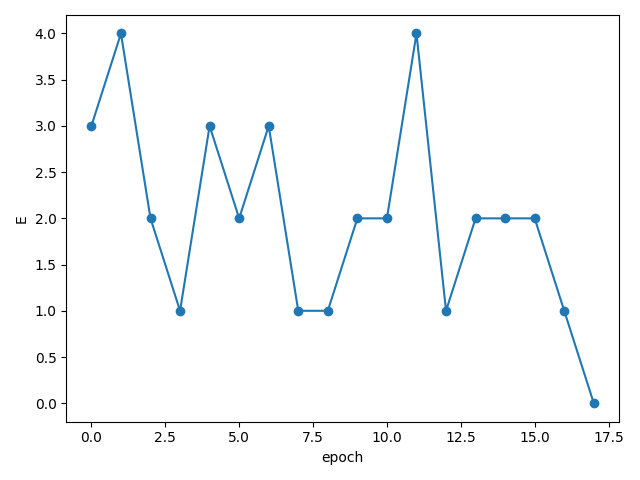
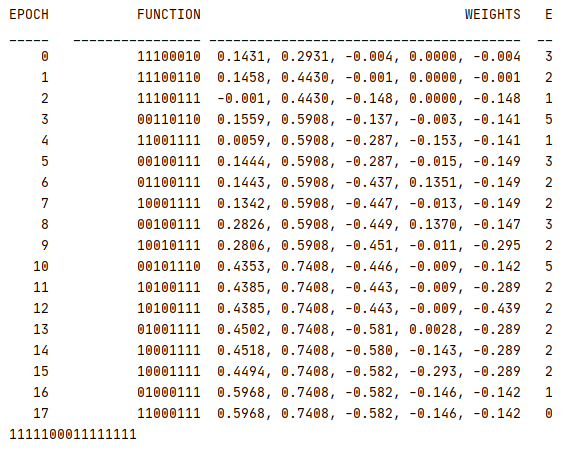
**

Рисунок 3 – График суммарной ошибки НС по эпохам обучения (логическая ФА)

Уменьшим размер выборки до 5 наборов

Результаты приведены в таблице 4 и на рисунке 4

Таблица 3. Параметры НС на последовательных эпохах (Логическая ФА) с уменьшенной выборкой



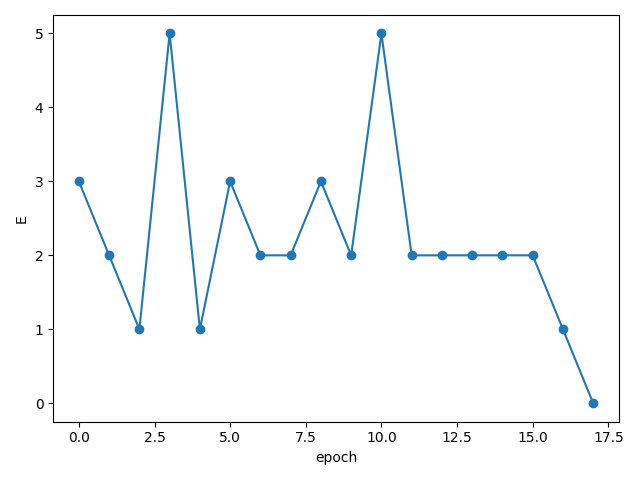
**

Рисунок 4 – График суммарной ошибки НС по эпохам обучения (логическая ФА) с уменьшенной выборкой

Код программы приведен в Приложении А.

# Выводы

В ходе выполнения лабораторной работы, мною было исследованно функционирование простейшей нейронной сети (НС) на базе нейрона с нелинейной функцией активации. Также она была обучена по правилу Видроу-Хоффа. Результаты совпали с ожидаемыми, что говорит о корректности работы программы.

Приложение А. Исходный код программы

*Файл main.py*

import numpy as np  
import math  
import matplotlib.pyplot as plt  
  
  
class NeuralNetwork:  
 def \_\_init\_\_*(*self, type, n, func*)*:  
 self.\_\_type = type  
 self.\_\_n = n  
 self.\_\_net = 0  
 self.\_\_weights = *[*0, 0, 0, 0, 0*]* self.\_\_func = func  
 self.\_\_errors = *[]* def function\_one*(*self*)*:  
 return 1 if self.\_\_net >= 0 else 0  
  
 def function\_two*(*self*)*:  
 return 0.5 \* *(*np.tanh*(*self.\_\_net*)* + 1*)* def function\_two\_dxdy*(*self*)*:  
 return 1 / *(*np.cosh*(*2 \* self.\_\_net*)* + 1*)* def net\_calculator*(*self, x*)*:  
 self.\_\_net = self.\_\_weights*[*0*]* + self.\_\_weights*[*1*]* \* x*[*1*]* + self.\_\_weights*[*2*]* \* x*[*2*]* + self.\_\_weights*[*3*]* \* x*[*3*]* + self.\_\_weights*[*4*]* \* x*[*4*]* def weights\_correction*(*self, sigma, x, dfdnet*)*:  
 for i in range*(*5*)*:  
 self.\_\_weights*[*i*]* = self.\_\_weights*[*i*]* + self.\_\_n \* sigma \* dfdnet \* x*[*i*]* def study*(*self*)*:  
 print*(***"EPOCH"**.rjust*(*5*)*, **"FUNCTION"**.rjust*(*18*)*, **"WEIGHTS"**.rjust*(*39*)*, **"E"**.rjust*(*3*))* print*(***"\_"** \* 5, **" "**, **"\_"** \* 16, **"\_"** \* 39, **" \_\_"***)* epoch = 0  
 while True:  
 error = 0  
 predicted\_y = False  
 y = **""** for i in range*(*16*)*:  
 x = *[*1, math.floor*(*i // 8*)* % 2, math.floor*(*i // 4*)* % 2, math.floor*(*i // 2*)* % 2, math.floor*(*i // 1*)* % 2*]* if self.\_\_type == 1:  
 self.net\_calculator*(*x*)* predicted\_y = True if self.function\_one*()* == 1 else False  
 elif self.\_\_type == 2:  
 self.net\_calculator*(*x*)* predicted\_y = True if self.function\_two*()* >= 0.5 else False  
 if predicted\_y is not self.\_\_func*[*i*]*:  
 error += 1  
 y += str*(*int*(*predicted\_y*))* dfdnet = 1  
 tn = int*(*self.\_\_func*[*i*])* yn = int*(*predicted\_y*)* sigm = tn - yn  
 self.net\_calculator*(*x*)* if self.\_\_type == 2:  
 dfdnet = self.function\_two\_dxdy*()* self.weights\_correction*(*sigm, x, dfdnet*)* w\_string = **', '**.join*([*str*(***"%.3f"** % it*)* if it < 0 else str*(***"%.4f"** % it*)* for it in self.\_\_weights*])* self.\_\_errors.append*([*epoch, error*])* print*(*str*(*epoch*)*.rjust*(*5*)*, str*(*y*)*.rjust*(*18*)*, str*(*w\_string*)*.rjust*(*39*)*, str*(*error*)*.rjust*(*3*))* if error == 0:  
 break  
 epoch += 1  
  
 def study\_selected*(*self*)*:  
 SELECTED = *[  
 [*1, 0, 0, 1, 1*]*,  
 *[*1, 0, 1, 0, 0*]*,  
 *[*1, 0, 1, 0, 1*]*,  
 *[*1, 0, 1, 1, 1*]*,  
 *[*1, 0, 1, 1, 0*]*,  
 *[*1, 1, 0, 0, 0*]*,  
 *[*1, 1, 0, 1, 0*]*,  
 *[*1, 1, 1, 0, 1*]*,  
 *]* SELECTED\_FUNC = *[* True,True, False, False, False, True, True, True*]* print*(***"EPOCH"**.rjust*(*5*)*, **"FUNCTION"**.rjust*(*18*)*, **"WEIGHTS"**.rjust*(*39*)*, **"E"**.rjust*(*3*))* print*(***"\_"** \* 5, **" "**, **"\_"** \* 16, **"\_"** \* 39, **" \_\_"***)* epoch = 0  
 while True:  
 error = 0  
 predicted\_y = False  
 y = **""** for i in range*(*len*(*SELECTED*))*:  
 if self.\_\_type == 1:  
 self.net\_calculator*(*SELECTED*[*i*])* predicted\_y = True if self.function\_one*()* == 1 else False  
 elif self.\_\_type == 2:  
 self.net\_calculator*(*SELECTED*[*i*])* predicted\_y = True if self.function\_two*()* >= 0.5 else False  
 if predicted\_y is not SELECTED\_FUNC*[*i*]*:  
 error += 1  
 y += str*(*int*(*predicted\_y*))* dfdnet = 1  
 tn = int*(*SELECTED\_FUNC*[*i*])* yn = int*(*predicted\_y*)* sigm = tn - yn  
 self.net\_calculator*(*SELECTED*[*i*])* if self.\_\_type == 2:  
 dfdnet = self.function\_two\_dxdy*()* self.weights\_correction*(*sigm, SELECTED*[*i*]*, dfdnet*)* w\_string = **', '**.join*([*str*(***"%.3f"** % it*)* if it < 0 else str*(***"%.4f"** % it*)* for it in self.\_\_weights*])* self.\_\_errors.append*([*epoch, error*])* print*(*str*(*epoch*)*.rjust*(*5*)*, str*(*y*)*.rjust*(*18*)*, str*(*w\_string*)*.rjust*(*39*)*, str*(*error*)*.rjust*(*3*))* if error == 0:  
 break  
 epoch += 1  
  
 predicted = *[]* for i in range*(*16*)*:  
 x = *[*1, math.floor*(*i // 8*)* % 2, math.floor*(*i // 4*)* % 2, math.floor*(*i // 2*)* % 2, math.floor*(*i // 1*)* % 2*]* if self.\_\_type == 1:  
 self.net\_calculator*(*x*)* predicted.append*(*True if self.function\_one*()* == 1 else False*)* elif self.\_\_type == 2:  
 self.net\_calculator*(*x*)* predicted.append*(*True if self.function\_two*()* >= 0.5 else False*)* y\_string = **''**.join*([***'1'** if it else **'0'** for it in predicted*])* print*(*y\_string*)* def printgraph*(*self*)*:  
 err = np.array*(*self.\_\_errors*)* x, y = err.T  
 plt.ylabel*(***'E'***)* plt.xlabel*(***'epoch'***)* plt.scatter*(*x, y*)* plt.plot*(*x, y*)* plt.show*()* def reset*(*self, type, n, func*)*:  
 self.\_\_type = type  
 self.\_\_n = n  
 self.\_\_net = 0  
 self.\_\_weights = *[*0, 0, 0, 0, 0*]* self.\_\_func = func  
 self.\_\_errors = *[]*def main*()*:  
 FUNCTION = *[*True, True, True, True, True, False, False, False, True, True, True, True, True, True, True, True*]* nw = NeuralNetwork*(*1, 0.3, FUNCTION*)* print*(***'-'** \* 30, **'FA 1st type'**, **'-'** \* 30*)* print*()* nw.study*()* nw.printgraph*()* print*()* print*()* print*(***'-'** \* 30, **'FA 2nd type'**, **'-'** \* 30*)* print*()* nw.reset*(*2, 0.3, FUNCTION*)* nw.study*()* nw.printgraph*()* print*()* print*()* print*(***'-'** \* 30, **'FA 2nd type'**, **'-'** \* 30*)* print*()* nw.reset*(*2, 0.3, FUNCTION*)* nw.study\_selected*()* nw.printgraph*()* print*()* print*()*if \_\_name\_\_ == **'\_\_main\_\_'**:  
 main*()*