**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

КАФЕДРА Информационная безопасность (ИУ8)

# ИНТЕЛЛЕКТУАЛЬНЫЕ ТЕХНОЛОГИИ ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ

## Лабораторная работа №1 на тему:

«Исследование однослойных нейронных сетей на примере моделирования булевых выражений»

Вариант 8

Выполнил: Песоцкий А. А.,

студент группы ИУ8-61

Проверила: Коннова Н.С.,

доцент каф. ИУ8

Москва 2020 г.

## Цель работы

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

## Постановка задачи

Получить модель булевой функции (БФ) на основе однослойной НС (единичный нейрон) с двоичными входами $x_1, x_2, x_3, x_4 \in \{0,1\}$ единичным входом смещения $x_0, = 1$, синаптическими весами $w_0, w_1, w_2, w_3, w_4$, двоичным выходом $y \in \{0,1\}$ и заданной нелинейной функцией активации $f: R \to (0,1)$.

Для заданной БФ реализовать обучение НС для двух случаев:

1. с использованием всех комбинаций переменных $x_1, x_2, x_3, x_4$;
2. с использованием части возможных комбинаций переменных $x_1, x_2, x_3, x_4$; остальные комбинации используются в качестве тестовых.

## Ход работы

Получим таблицу истинности для моделируемой БФ:

$$F(x_1, x_2, x_3, x_4) = (\text{x1} + \text{x2} + \text{x4})\ \text{x3}$$

| $x_1$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $x_2$ | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| $x_3$ | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| $x_4$ | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| $F$ | **0** | **0** | **0** | **1** | **0** | **0** | **1** | **1** | **0** | **0** | **1** | **1** | **0** | **0** | **1** | **1** |

На начальном шаге $l = 0$ (эпоха $k = 0$) весовые коэффициенты берутся в виде:

$$w_0^{(0)} = w_1^{(0)} = w_2^{(0)} = w_3^{(0)} = w_4^{(0)} = 0$$

Норму обучения для всех случаев выберем $\eta = 0.3$

1. Обучение НС с использованием всех комбинаций переменных $x_1, x_2, x_3, x_4$.

1.1. Используем пороговую ФА:

$$f(net) = \begin{cases} 1, net \geq 0 \\ 0, net < 0 \end{cases}$$

*Таблица 1 Параметры НС на последовательных эпохах (пороговая ФА)*

| K | W | Y | E |
|---|---|---|---|
| 0 | [0, 0, 0, 0, 0] | [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1] | 9 |
| 1 | [−0.3, −0.3, 0.0, 0.6, 0.3] | [0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1] | 3 |
| 2 | [−0.3, 0.0, 0.0, 1.2, 0.0] | [0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1] | 1 |
| 3 | [−0.6, 0.0, 0.0, 0.9, 0.0] | [0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1] | 1 |
| 4 | [−0.6, 0.0, 0.0, 0.9, 0.3] | [0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1] | 1 |
| 5 | [−0.6, 0.0, 0.3, 1.2, 0.0] | [0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1] | 1 |
| 6 | [−0.9, 0.0, 0.3, 0.9, 0.0] | [0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1] | 1 |
| 7 | [−0.9, 0.0, 0.3, 0.9, 0.3] | [0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1] | 1 |
| 8 | [−0.9, 0.0, 0.3, 1.2, 0.3] | [0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1] | 1 |
| 9 | [−1.2, 0.0, 0.0, 1.2, 0.0] | [0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1] | 1 |
| 10 | [−1.2, 0.0, 0.0, 1.2, 0.3] | [0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1] | 1 |
| 11 | [−1.2, 0.0, 0.0, 1.2, 0.6] | [0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1] | 1 |
| 12 | [−1.2, 0.0, 0.3, 1.2, 0.6] | [0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1] | 1 |
| 13 | [−1.2, 0.0, 0.6, 1.5, 0.3] | [0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1] | 1 |
| 14 | [−1.5, 0.0, 0.3, 1.5, 0.0] | [0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1] | 1 |
| 15 | [−1.5, 0.0, 0.3, 1.5, 0.3] | [0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1] | 1 |
| 16 | [−1.5, 0.0, 0.3, 1.5, 0.6] | [0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1] | 1 |
| 17 | [−1.5, 0.0, 0.3, 1.5, 0.9] | [0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1] | 1 |
| 18 | [−1.5, 0.0, 0.6, 1.8, 0.6] | [0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1] | 1 |
| 19 | [−1.5, 0.3, 0.6, 1.8, 0.6] | [0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1] | 1 |
| 20 | [−1.8, 0.3, 0.6, 1.5, 0.6] | [0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1] | 0 |

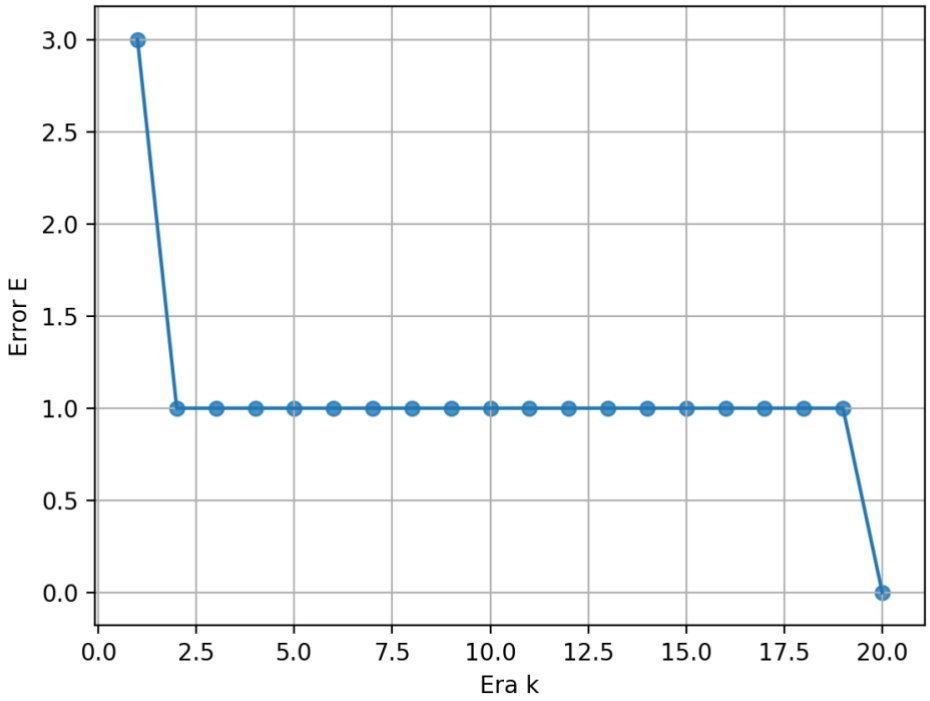


*Рисунок 1 График суммарной ошибки НС по эпохам обучения (пороговая ФА)*

1.2. Используем сигмоидальную (логистическую) ФА и её производную:

$$f(net) = \frac{1}{1 + \exp(-net)}, \frac{df(net)}{dnet} = f(net)[1 - f(net)]$$

*Таблица 2 Параметры НС на последовательных эпохах (логистическая ФА)*

| K | W | Y | E |
|---|---|---|---|
| 0 | [0, 0, 0, 0, 0] | [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1] | 9 |
| 1 | [-0.0, 0.0, -0.0, 0.15, -0.0] | [0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1] | 1 |
| 2 | [-0.07, 0.0, -0.0, 0.08, -0.0] | [0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1] | 1 |
| 3 | [-0.07, -0.07, 0.0, 0.22, -0.0] | [0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1] | 1 |
| 4 | [-0.07, 0.0, 0.0, 0.23, -0.0] | [0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1] | 1 |
| 5 | [-0.15, 0.0, 0.0, 0.15, -0.0] | [0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1] | 1 |
| 6 | [-0.15, 0.0, 0.0, 0.15, 0.07] | [0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1] | 1 |
| 7 | [-0.15, 0.0, 0.0, 0.15, 0.15] | [0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1] | 1 |
| 8 | [-0.15, 0.0, -0.0, 0.22, 0.15] | [0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1] | 1 |
| 9 | [-0.15, 0.0, 0.07, 0.3, 0.07] | [0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1] | 1 |
| 10 | [-0.22, 0.0, 0.07, 0.22, 0.07] | [0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1] | 1 |
| 11 | [-0.22, 0.0, 0.07, 0.22, 0.15] | [0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1] | 1 |
| 12 | [-0.22, 0.0, 0.07, 0.3, 0.15] | [0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1] | 1 |
| 13 | [-0.3, -0.07, 0.07, 0.3, 0.07] | [0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1] | 2 |
| 14 | [-0.22, 0.0, 0.07, 0.37, 0.15] | [0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1] | 1 |
| 15 | [-0.3, 0.0, 0.07, 0.3, 0.15] | [0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1] | 1 |
| 16 | [-0.3, 0.0, 0.15, 0.3, 0.15] | [0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1] | 1 |
| 17 | [-0.3, 0.07, 0.07, 0.37, 0.15] | [0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1] | 1 |
| 18 | [-0.37, 0.07, 0.07, 0.3, 0.15] | [0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1] | 0 |

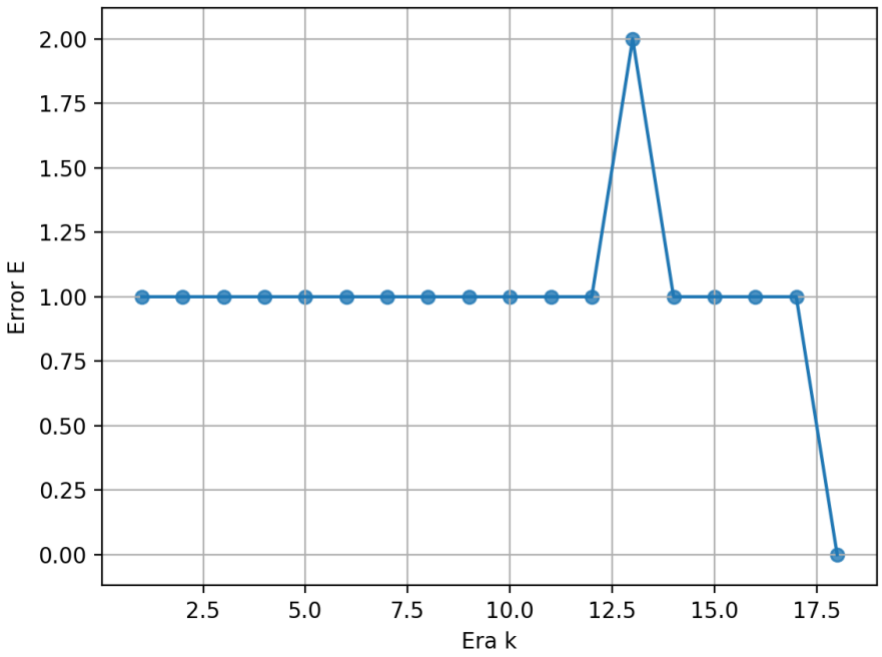


*Рисунок 2 График суммарной ошибки НС по эпохам обучения (логистическая ФА)*

2. Обучение НС с использованием части комбинаций переменных $x_1, x_2, x_3, x_4$.

   Последовательно уменьшая выборку количества векторов, найдём наименьшее количество необходимых для обучения векторов.

   2.1. Используя пороговую ФА.

   Минимальный набор из четырёх векторов:

$x^{(1)}$ = [0, 0, 1, 0] $x^{(2)}$ = [0, 1, 1, 0] $x^{(3)}$ = [1, 0, 1, 1] $x^{(4)}$ = [1, 1, 0, 1]

   Даёт следующие синаптические коэффициенты:

W = (-1.8, 0.3, 0.6, 1.5, 0.3)

   Для полного обучения потребовалось 24 эпохи.

*Таблица 3 Параметры НС на последовательных эпохах (пороговая ФА) при наборе из 4 векторов*

| K | W | Y | E |
|---|---|---|---|
| 1 | [-0.3, -0.3, 0.0, 0.0, -0.3] | [0, 0, 0, 0] | 2 |
| 2 | [0.0, -0.3, 0.0, 0.6, -0.3] | [1, 1, 1, 0] | 1 |
| 3 | [-0.3, -0.3, -0.3, 0.6, -0.3] | [1, 1, 0, 0] | 2 |
| 4 | [-0.3, -0.3, -0.3, 0.9, -0.3] | [1, 1, 0, 0] | 2 |
| 5 | [-0.3, -0.3, -0.3, 1.2, -0.3] | [1, 1, 1, 0] | 1 |
| 6 | [-0.3, -0.3, 0.0, 1.2, -0.3] | [1, 1, 1, 0] | 1 |
| 7 | [-0.3, 0.0, 0.0, 1.2, 0.0] | [1, 1, 1, 0] | 1 |
| 8 | [-0.6, 0.0, 0.0, 0.9, 0.0] | [1, 1, 1, 0] | 1 |
| 9 | [-0.6, 0.0, 0.3, 0.9, 0.0] | [1, 1, 1, 0] | 1 |
| 10 | [-0.9, 0.0, 0.0, 0.9, 0.0] | [1, 1, 1, 0] | 1 |
| 11 | [-0.9, 0.0, 0.3, 0.9, 0.0] | [1, 1, 1, 0] | 1 |
| 12 | [-0.9, 0.0, 0.6, 0.9, 0.0] | [1, 1, 1, 0] | 1 |
| 13 | [-1.2, -0.3, 0.6, 0.9, -0.3] | [0, 1, 0, 0] | 1 |
| 14 | [-0.9, 0.0, 0.6, 1.2, 0.0] | [1, 1, 1, 0] | 1 |
| 15 | [-1.2, 0.0, 0.3, 1.2, 0.0] | [1, 1, 1, 0] | 1 |
| 16 | [-1.2, 0.0, 0.6, 1.2, 0.0] | [1, 1, 1, 0] | 1 |
| 17 | [-1.2, 0.0, 0.9, 1.2, 0.0] | [1, 1, 1, 0] | 1 |
| 18 | [-1.5, 0.0, 0.6, 1.2, 0.0] | [0, 1, 0, 0] | 1 |
| 19 | [-1.5, 0.0, 0.3, 1.5, 0.0] | [1, 1, 1, 0] | 1 |
| 20 | [-1.5, 0.0, 0.6, 1.5, 0.0] | [1, 1, 1, 0] | 1 |
| 21 | [-1.5, 0.0, 0.9, 1.5, 0.0] | [1, 1, 1, 0] | 1 |
| 22 | [-1.8, 0.0, 0.6, 1.5, 0.0] | [0, 1, 0, 0] | 1 |
| 23 | [-1.5, 0.3, 0.6, 1.8, 0.3] | [1, 1, 1, 0] | 1 |
| 24 | [-1.8, 0.3, 0.6, 1.5, 0.3] | [0, 1, 1, 0] | 0 |

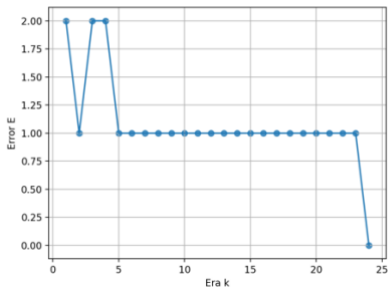


*Рисунок 3 График суммарной ошибки НС по эпохам обучения (пороговая ФА) при наборе из 4-х векторов*

2.2. Используя сигмоидальную (логистическую) ФА.

Минимальный набор из четырёх векторов:

$x^{(1)}$ = [0, 0, 1, 0] $x^{(2)}$ = [0, 0, 1, 1] $x^{(3)}$ = [1, 1, 0, 1] $x^{(4)}$ = [1, 1, 1, 0]

Даёт следующие синаптические коэффициенты:

W = (-0.3743, 0.1486, 0.1486, 0.2998, 0.0745)

Для полного обучения потребовалось 17 эпох.

*Таблица 4 Параметры НС на последовательных эпохах (логистическая ФА) при наборе из 4 векторов*

| K | W | Y | E |
|---|---|---|---|
| 1 | [-0.0013, -0.0008, -0.0008, 0.0736, -0.0003] | [1, 1, 0, 1] | 1 |
| 2 | [-0.0016, -0.0012, -0.0012, 0.1482, -0.0003] | [1, 1, 0, 1] | 1 |
| 3 | [-0.0012, -0.0012, -0.0012, 0.2235, -0.0002] | [1, 1, 0, 1] | 1 |
| 4 | [-0.0753, -0.0012, -0.0012, 0.1494, -0.0002] | [1, 1, 0, 1] | 1 |
| 5 | [-0.0753, -0.0012, -0.0012, 0.1494, 0.0747] | [1, 1, 0, 1] | 1 |
| 6 | [-0.0756, -0.0016, -0.0016, 0.224, 0.0747] | [1, 1, 0, 1] | 1 |
| 7 | [-0.0752, 0.0734, 0.0734, 0.2244, 0.0747] | [1, 1, 1, 1] | 2 |
| 8 | [-0.1498, 0.0734, 0.0734, 0.2248, -0.0002] | [1, 1, 0, 1] | 1 |
| 9 | [-0.1497, 0.0735, 0.0735, 0.2998, -0.0002] | [1, 1, 0, 1] | 1 |
| 10 | [-0.2242, 0.0735, 0.0735, 0.2252, -0.0002] | [1, 1, 0, 1] | 1 |
| 11 | [-0.2247, 0.0735, 0.0735, 0.2248, 0.0744] | [1, 1, 0, 1] | 1 |
| 12 | [-0.2248, 0.0735, 0.0735, 0.2995, 0.0744] | [1, 1, 0, 1] | 1 |
| 13 | [-0.2246, 0.0736, 0.0736, 0.3746, 0.0745] | [1, 1, 0, 1] | 1 |
| 14 | [-0.2992, 0.0736, 0.0736, 0.3001, 0.0745] | [1, 1, 0, 1] | 1 |
| 15 | [-0.2993, 0.0736, 0.0736, 0.3, 0.1494] | [1, 1, 0, 1] | 1 |
| 16 | [-0.2993, 0.1486, 0.1486, 0.3, 0.1494] | [1, 1, 1, 1] | 2 |
| 17 | [-0.3743, 0.1486, 0.1486, 0.2998, 0.0745] | [0, 1, 0, 1] | 0 |

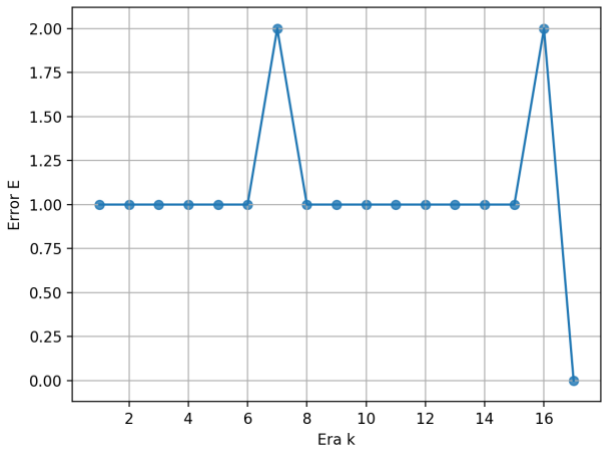


*Рисунок 4 График суммарной ошибки НС по эпохам обучения (логистическая ФА) при наборе из 4-х векторов*

## Выводы

В ходе выполнения работы было изучено функционирование простейшей НС на базе нейрона с нелинейной функцией активации и ее обучение по правилу Видроу-Хоффа В качестве ФА были даны – пороговая и логистическая. В ходе обучения на полных наборах было выявлено, что с использованием логистической ФА понадобилось меньше эпох, чем для обучения с использованием пороговой.

Кроме того, для случаев пороговой и логистической функций активации были найдены минимально возможные наборы векторов, на которых можно обучить НС. В обоих случаях удалось найти наборы, состоящие из четырёх векторов. Как и в обучении на полных наборах с использованием пороговой функции активации понадобилось меньшее количество эпох, чем с использованием логистической.

# Приложение А.

*Файл ‘computation.py’.*

```
from math import exp
import matplotlib.pyplot as plt
import matplotlib as mpl


truth_table = [[0, 0, 0, 0], [0, 0, 0, 1], [0, 0, 1, 0], [0, 0, 1, 1],
               [0, 1, 0, 0], [0, 1, 0, 1], [0, 1, 1, 0], [0, 1, 1, 1],
               [1, 0, 0, 0], [1, 0, 0, 1], [1, 0, 1, 0], [1, 0, 1, 1],
               [1, 1, 0, 0], [1, 1, 0, 1], [1, 1, 1, 0], [1, 1, 1, 1]]
nu = 0.3



'''
Функция подсчитывает net
:param vector_x: булеый набор
:param vector_w: набор весов
:param return: значение net
'''
def get_net(vector_x, vector_w):
    net = 0
    for i in range(1, len(vector_w)):
        net += vector_x[i-1] * vector_w[i]
    net += vector_w[0]
    return net



'''
Функция пересчитывает значения весов
:param vector_w: набор весов
:param delta: дельта
:param n: норма обучения
:param vec: булевый набор
:param net: net
:param func_type: тип функции (пороговая или логист.)
:param return: новый набор весов
```

```
'''
def update_w(vector_w, delta, n, vec, net, func_type):
    for i in range(1, len(vector_w)):
        vector_w[i] += get_dw(delta, nu, vec[i - 1], net, func_type)
    vector_w[0] += get_dw(delta, nu, 1, net, func_type)
    return vector_w


'''
Функция подсчитывает f(net)
:param net: net
:param return: значение f(net)
'''
def l_function(net):
    return 1 / (1 + exp(-net))


'''
Функция подсчитывает y(net) / y(out)
:param net_out:  значение net / out
:param func_type: тип функции (пороговая или логист.)
:param return: значение f(net) / y(out)
'''
def get_function_out(net_out, func_type):
    if func_type == 't':
        return 1 if net_out >= 0 else 0
    else:
        return 1 if net_out >= 0.5 else 0


'''
Функция подсчитывает набор реальных выходов
:param vector_w: набор весов
:param table: все булевы наборы
:param func_type: тип функции (пороговая или логист.)
:param return: набор реальных выходов
```

```
'''
def get_Y_real(vector_w, table, func_type):
    y_list = list()
    for vec in table:
        net = get_net(vec, vector_w)
        if func_type == 't':
            # 2.1
            y = get_function_out(net, 't')
            y_list.append(y)
        if func_type == 'l':
            # 2.2
            out = l_function(net)
            y = get_function_out(out, 'l')
            y_list.append(y)
    return y_list


'''
Функция подсчитывает набор целевых выходов
:param table: все булевы наборы
:param return: набор целевых выходов
'''
def get_Y_target(table):
    t_list = list()
    for vec in table:
        t_list.append(int(bool_function(vec)))
    return t_list


'''
Функция подсчитывает дельта
:param t: целевой выход
:param y: реальный выход
:param return: значение delta
'''
def get_delta(t, y):
    return t - y
```

```
'''
Функция подсчитывает коррекцию веса
:param delta: дельта
:param n: норма обучения
:param x: компонента обучающего вектора
:param net: net
:param func_type: тип функции (пороговая или логист.)
:param return: значение коррекции веса
'''
def get_dw(delta, n, x, net, func_type):
    if func_type == 'l':
        return n * delta * l_function(net) * (1 - l_function(net)) * x
    else:
        return n * delta * x


'''
Функция подсчитывает квадратичную ошибку
:param vec1: целевой набор выходов
:param vec2: реальный набор выходов
:param return: значение квадратичной ошибки
'''
def hamming_distance(vec1, vec2):
    return sum(ch1 != ch2 for ch1, ch2 in zip(vec1, vec2))


'''
Функция подсчитывает значение булевой функции
:param vector: булевый набор
:param return: значение булевой функции
'''
def bool_function(vector):
    return (vector[0] or vector[1] or vector[3]) and vector[2]
    # return not ((vector[0]) and (vector[1])) and vector[2] and vector[3]
```

```
    '''
    Функция строит график E(k)
    :param epoch_list: список эпох
    :param error_list: список ошибок
    '''
    def graph_plot(epoch_list, error_list):
        plt.grid(True)
        plt.plot(epoch_list, error_list)
        plt.xlabel('Era k')
        plt.ylabel('Error E')
        plt.scatter(epoch_list, error_list, alpha=0.8)
        mpl.style.use('bmh')
        plt.show()
```

## *Файл 'neural.py'.*

```
from computation import *
import itertools
import copy
from prettytable import PrettyTable


'''
Функция обучения на всём числе наборов
:param vector_w: набор весов
:param func_type: тип функции (пороговая или логист.)
:param return: таблица обучения
'''
def learn(vector_w, func_type):
    y_list = get_Y_real(vector_w, truth_table, func_type)
    t_list = get_Y_target(truth_table)

    epoch_list = list()
    error_list = list()
    error_sum = hamming_distance(y_list, t_list)
    epoch = 0
```

```
    pt = PrettyTable(['K', 'W', 'Y', 'E'])
    pt.add_row([epoch, copy.copy(vector_w), y_list, error_sum])

    #пока ошибка не равна 0
    while error_sum != 0:
        y_list = list()
        epoch += 1

        for (vec, t) in zip(truth_table, t_list):

            net = get_net(vec, vector_w)
            if func_type == 't':
                y = get_function_out(net, 't')
                y_list.append(y)
            if func_type == 'l':
                out = l_function(net)
                y = get_function_out(out, 'l')
                y_list.append(y)

            delta = get_delta(t, y)
            update_w(vector_w, delta, nu, vec, net, func_type)

        epoch_list.append(epoch)

        y_list = get_Y_real(vector_w, truth_table, func_type)
        error_sum = hamming_distance(t_list, y_list)
        error_list.append(error_sum)
        pt.add_row([epoch, copy.copy([float('{:.2f}'.format(x)) for x in vector_w]), y_list, error_sum])

    graph_plot(epoch_list, error_list)
    return pt


'''
Функция обучения на минимальном числе наборов
:param vector_w: набор весов
:param func_type: тип функции (пороговая или логист.)
:param return: таблица обучения
'''
def learn_min(vector_w, func_type):
    error_sum = 1
```

```
for L in range(1, 17):
    for subset in itertools.combinations(truth_table, L):

        vector_w = [0, 0, 0, 0, 0]
        y_list = get_Y_real(vector_w, subset, func_type)
        t_list = get_Y_target(subset)
        error_list = list()
        epoch = 0
        epoch_list = list()

        pt = PrettyTable(['K', 'W', 'Y', 'E'])

        # пока ошибка не равна 0
        while error_sum != 0 and epoch < 50:
            epoch += 1

            for (vec, t) in zip(subset, t_list):

                net = get_net(vec, vector_w)
                if func_type == 't':
                    y = get_function_out(net, 't')
                    y_list.append(y)
                if func_type == 'l':
                    out = l_function(net)
                    y = get_function_out(out, 'l')
                    y_list.append(y)

                delta = get_delta(t, y)
                update_w(vector_w, delta, nu, vec, net, func_type)

            epoch_list.append(epoch)
            y_list = get_Y_real(vector_w, subset, func_type)
            error_sum = hamming_distance(t_list, y_list)
            error_list.append(error_sum)
            pt.add_row([epoch, copy.copy([float('{:.4f}'.format(x)) for x in vector_w]), y_list, error_sum])
            y_list = list()

        y_list = get_Y_real(vector_w, truth_table, func_type)
        t_list = get_Y_target(truth_table)
        error_sum = hamming_distance(t_list, y_list)
        if error_sum == 0:
```

```
            graph_plot(epoch_list, error_list)
            return pt, subset

    return pt


if __name__ == "__main__":
    W = [0, 0, 0, 0, 0]
    test, combs = learn_min(W, 'l')
    # test = learn(W, 'l')
    print(test)
    print(combs)
```