**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 №4 на тему:

«Исследование нейронных сетей с радиальными базисными функциями (RBF) 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 НС с радиальными базисными функциями (RBF) и обучить её по правилу Видроу-Хоффа.

## Постановка задачи

Получить модель булевой функции (БФ) на основе RBF–НС с двоичными входами $x_1, x_2, x_3, x_4 \in \{0,1\}$, единичным входом смещения $\varphi_0, = 1$, синаптическими весами $v_0, v_1, v_2, v_3, v_4$, двоичным выходом $y \in \{0,1\}$ с пороговой ФА $\varphi: R \to (0,1]$ и координатами центров $c_{j1}, c_{j2}, c_{j3}, c_{j4}$ $(j = \overline{1,J})$ (рисунок 1).

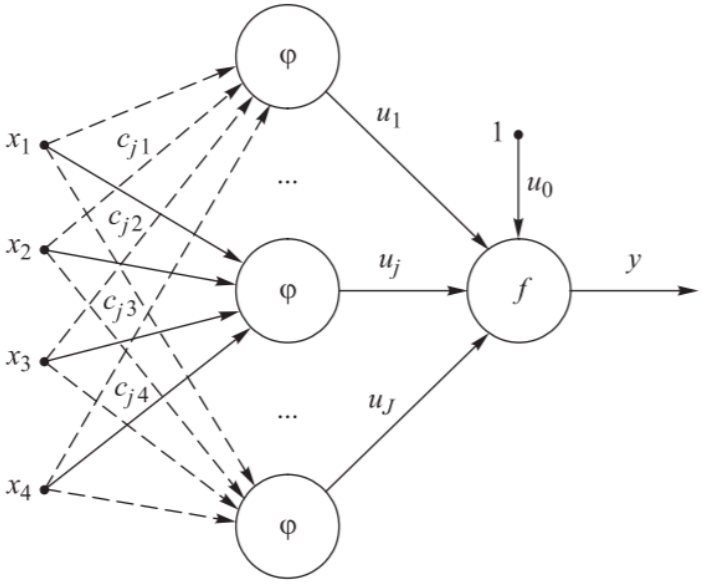


*Рисунок 1. RBF*

Для заданной БФ количество RBF-нейронов необходимо выбирать из соотношения $J = \min\{J_0, J_1\}$, где $J_0, J_1$ – количество векторов **x** = ($x_1$, $x_2$, $x_3$, $x_4$), соответствующих значениям БФ «0» и «1». Центры RBF $c^{(j)} = (c_{j1}, c_{j2}, c_{j3}, c_{j4})$ должны совпадать с концами этих векторов.

Требуется найти минимальный набор векторов **x**, используемых для обуччения.

## 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** |

Находим количество RBF-нейронов: $J = 7$. Центры RBF-нейронов располагаем в точках

$$C^{(1)} = (0,0,1,1), C^{(2)} = (0,1,1,0), C^{(3)} = (0,1,1,1),$$
$$C^{(4)} = (1,0,1,0), C^{(5)} = (1,0,1,1), C^{(6)} = (1,1,1,0), C^{(7)} = (1,1,1,1)$$

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

$$v_0^{(0)} = v_1^{(0)} = v_2^{(0)} = \cdots = w_n^{(0)} = 0$$

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:

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

   Минимальный набор из трёх векторов:

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

   Даёт следующие синаптические веса:

   **V** = (-0.3, 0.4525337806, 0.0291626624, 0.2015837676, 0.0291626624, 0.2015837676, 0.270127759, 0.1695862356)

Для полного обучения потребовалось 5 эпох:

| Epoch | Weights | Y | Error |
|---|---|---|---|
| 0 | [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] | [1, 1, 1] | 1 |
| 1 | [0.0, 0.259399415, 0.0, 0.0954277118, 0.0, 0.0954277118, 0.0, 0.0351058933] | [1, 1, 1] | 1 |
| 2 | [0.0, 0.5187988301, 0.0, 0.1908554237, 0.0, 0.1908554237, 0.0, 0.0702117866] | [1, 1, 1] | 1 |
| 3 | [0.0, 0.4931343656, 0.0697632474, 0.2165198881, 0.0697632474, 0.2165198881, 0.2850638795, 0.1750809273] | [1, 1, 1] | 1 |
| 4 | [-0.3, 0.4525337806, 0.0291626624, 0.2015837676, 0.0291626624, 0.2015837676, 0.270127759, 0.1695862356] | [0, 1, 1] | 0 |

*Рисунок 2. Параметры НС на последовательных эпохах (пороговая ФА) при наборе из 3 векторов*

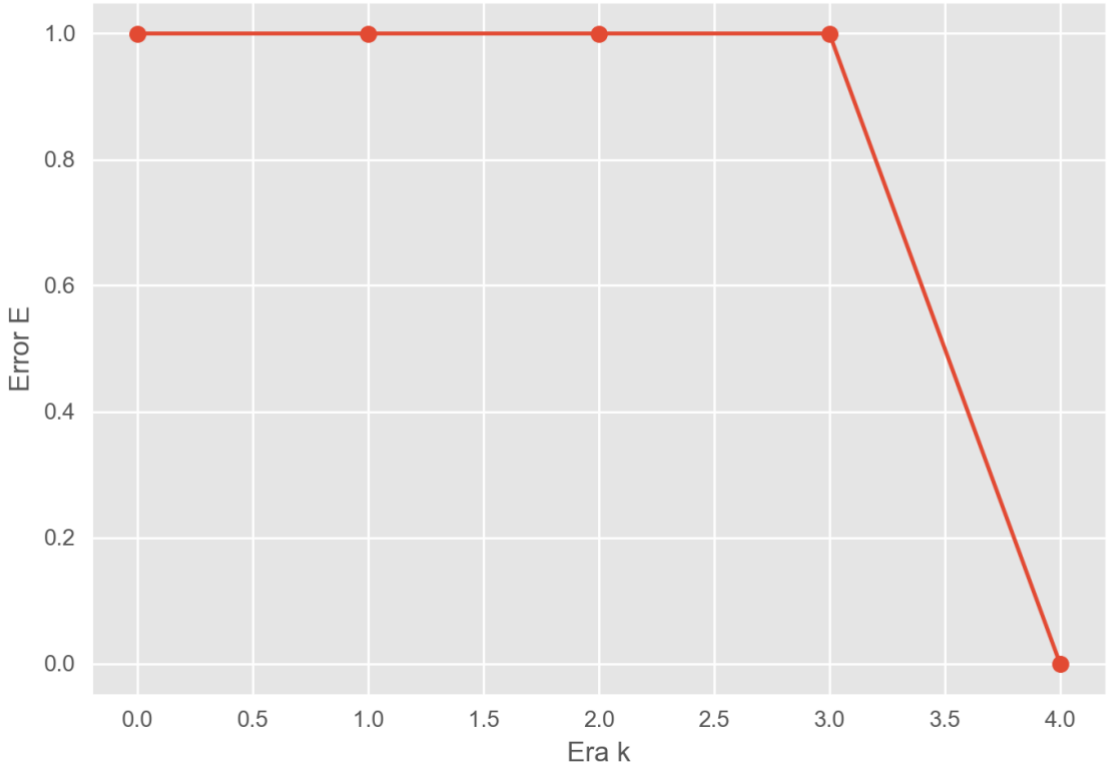


*Рисунок 3. График суммарной ошибки НС по эпохам обучения (логистическая ФА) при наборе из 3 векторов*

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

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

Используя сигмоидальную (логистическую) ФА.

Минимальный набор из четырёх векторов:

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

Даёт следующие синаптические веса:

**V** = (-0.0752466396, 0.0431194053, 0.0329029641, 0.0246333107, 0.0329029641, 0.0246333107, 0.0769105428, 0.0329029641)

Для полного обучения потребовалось 5 эпох.

| Epoch | Weights | Y | Error |
|---|---|---|---|
| 0 | [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] | [1, 1, 1] | 1 |
| 1 | [-0.0002332202, 0.0471758217, 0.0063845532, 0.0173550149, 0.0063845532, 0.0173550149, 0.0023487459, 0.0063845532] | [1, 1, 1] | 1 |
| 2 | [-0.0002688441, 0.0945430704, 0.0127950132, 0.0347804519, 0.0127950132, 0.0347804519, 0.0047070223, 0.0127950132] | [1, 1, 1] | 1 |
| 3 | [-0.0002797805, 0.0706981716, 0.0366353443, 0.0347789718, 0.0366353443, 0.0347789718, 0.0782836087, 0.0366353443] | [1, 1, 1] | 1 |
| 4 | [-0.0752466396, 0.0431194053, 0.0329029641, 0.0246333107, 0.0329029641, 0.0246333107, 0.0769105428, 0.0329029641] | [0, 1, 1] | 0 |

*Рисунок 4. Параметры НС на последовательных эпохах (логистическая ФА) при наборе из 3 векторов*

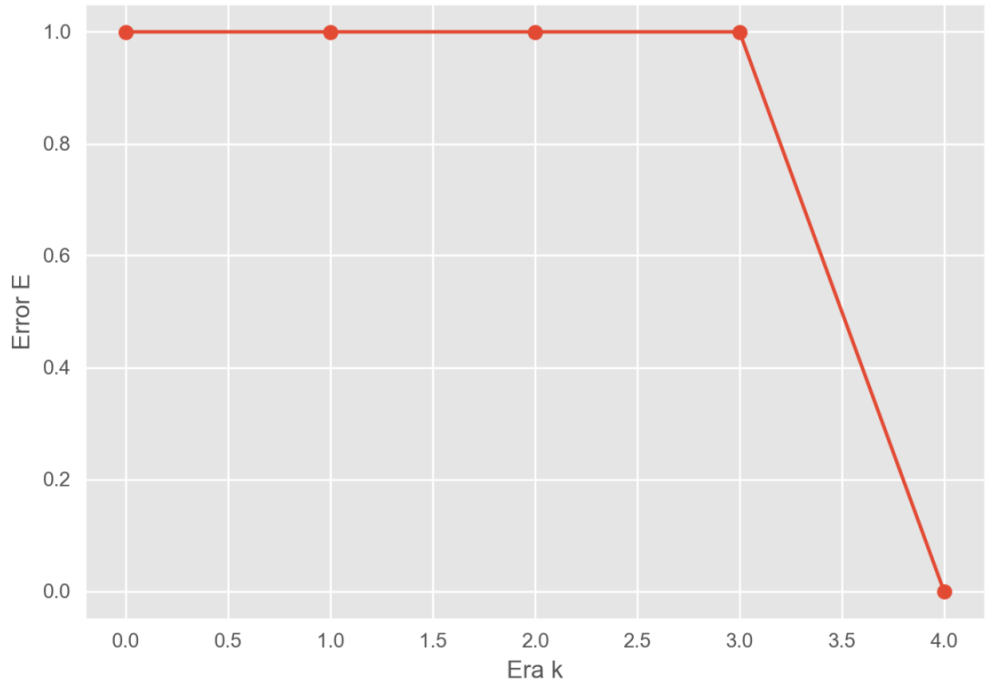


*Рисунок 4. График суммарной ошибки НС по эпохам обучения (логистическая ФА) при наборе из 3 векторов*

## Выводы

В ходе выполнения лабораторной работы было изучено функционирование НС с радиальными базисными функциями (RBF) и было выполнено ее обучение по правилу Видроу-Хоффа.

Были найдены минимально возможные наборы векторов, на которых можно обучить НС.

# Приложение А.

*Файл ‘rbf.py’.*

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


'''
Функция обучения
:param func_type: тип функции (пороговая или логист.)
:param return: таблица обучения и минимальный набор
'''
def learn_rbf(func_type):
    vlist, t1, f1 = get_Y_target(truth_table)

    #выбор центров
    if len(t1) <= 8:
        c_list = t1
    else:
        c_list = f1

    #пробегаем всевозможные комбинации
    for L in range(1, 17):
        for subset in itertools.combinations(truth_table, L):
            #заполняем веса
            vector_v = list()
            for i in range(0, len(c_list) + 1):
                vector_v.append(0)
            #целевой выход
            t_list, t2, f2 = get_Y_target(subset)
            error_list = list()
            epoch = 0
            epoch_list = list()

            pt = PrettyTable(['Epoch', 'Weights', 'Y', 'Error'])
            y_list = get_Y_real(vector_v, subset, func_type, c_list)
            epoch_list.append(epoch)
            error_sum = hamming_distance(t_list, y_list)
            error_list.append(error_sum)
            pt.add_row([epoch, copy.copy([float('{:.7f}'.format(x)) for x in vector_v]), y_list, error_sum])
```

```
        y_list = list()

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

            for (vec, t) in zip(subset, t_list):
                #расчёт фи
                phi_list = list()
                for c in c_list:
                    phi_list.append(get_phi(vec, c))
                net = get_net(phi_list, vector_v) #считаем net

                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)
                #обновляем веса
                delta = get_delta(t, y)
                update_w(vector_v, delta, phi_list, net, func_type)

            epoch_list.append(epoch)
            y_list = get_Y_real(vector_v, subset, func_type, c_list)
            error_sum = hamming_distance(t_list, y_list)
            error_list.append(error_sum)
            pt.add_row([epoch, copy.copy([float('{:.10f}'.format(x)) for x in vector_v]), y_list, error_sum])
            y_list = list()
        #проверка весов на всех наборах
        y_list = get_Y_real(vector_v, truth_table, func_type, c_list)
        t_list, t3, f3 = 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__":
    table, comb = learn_rbf('l')
    print(table)
    print(comb)
```

## *Файл ‘computation.py’.*

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

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    #норма обучения


'''
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 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, 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_v, table, func_type, c_list):
    y_list = list()
    for vec in table:
```

```
        phi_list = list()
        for c in c_list:
            phi_list.append(get_phi(vec, c))
        net = get_net(phi_list, vector_v)

        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)
    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()
    truth_list = list()
    false_list = list()
    for vec in table:
        t_list.append(int(bool_function(vec)))
        if int(bool_function(vec)) == 1:
            truth_list.append(vec)
        else:
            false_list.append(vec)
    return t_list, truth_list, false_list


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

```
def get_phi(vector_x, vector_c):
    sm = 0
    for i in range(0, 4):
        sm += (vector_x[i] - vector_c[i])**2
    return exp(-sm)


'''
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):
    style.use('seaborn')
    style.use('ggplot')
    plt.grid(True)
    plt.plot(epoch_list, error_list)
    plt.xlabel('Era k')
    plt.ylabel('Error E')
    plt.scatter(epoch_list, error_list)
    mpl.style.use('bmh')
    plt.show()
```