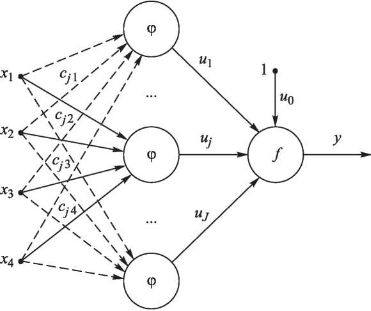
# Практическая работа № 4

# Исследование нейронных сетей  с радиальными базисными функциями (RBF)  на примере моделирования булевых выражений

**Цель работы —** исследовать функционирование НС с радиальными базисными функциями (RBF) и обучить ее по правилу Видроу — Хоффа.

***Постановка задачи.***

Получить модель булевой функции (БФ) на основе RBF-НС с двоичными входами ******, единичным входом смещения ******, синаптическими весами ******, двоичным выходом ****** с пороговой ФА выходного нейрона, *J* скрытыми RBF-нейронами с гауссовой ФА ******и координатами центров ******(рис. 4.1).

******

Для заданной БФ (варианты см. в табл. 4.1) количество RBF- нейронов необходимо выбирать из соотношения , где *—* количество векторов ***,*** соответствующих значениям БФ «О» и «1». Центры RBF  должны совпадать с концами этих векторов.

Требуется найти минимальный набор векторов **х**, используемых для обучения.

***Варианты заданий***

**Варианты БФ и ФА**

|  |  |  |
| --- | --- | --- |
| № Варианта | Моделируемая БФ | ФА |
| 6 |  | 1,4 |

# Выполнение:

## Найдем кол-во RBF нейронов и их центры:

Таблица истинности функции

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  | F = |
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

Количество RBF нейронов = 3, центры RBF нейронов расположим в точках:

## Обучение нейросети на последовательных эпохах

|  |  |  |  |
| --- | --- | --- | --- |
| k | Вектор весов w | Выходной вектор y | Суммарная ошибка Е |
| 0 | [0 0 0 0] | [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0] | 2 |
| 1 | [ 0. -0.259 0.11 0.11 ] | [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0] | 2 |
| 2 | [ 0. -0.329 -0.079 0.221] | [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0] | 2 |
| 3 | [ 0. -0.399 0.031 0.031] | [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0] | 1 |
| 4 | [ 0.3 -0.358 0.141 0.141] | [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0] | 2 |
| 5 | [ 0.3 -0.428 -0.048 0.252] | [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0] | 3 |
| 6 | [ 0. -0.608 -0.238 0.062] | [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0] | 1 |
| 7 | [ 0.3 -0.568 -0.127 0.173] | [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0] | 2 |
| 8 | [ 0.3 -0.637 -0.017 -0.017] | [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0] | 2 |
| 9 | [ 0.3 -0.707 -0.207 0.093] | [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0] | 2 |
| 10 | [ 0.3 -0.777 -0.096 -0.096] | [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1] | 0 |

На графике 1 показано поведение суммарной ошибки нейронной сети по эпохам обучения (пороговая ФА)

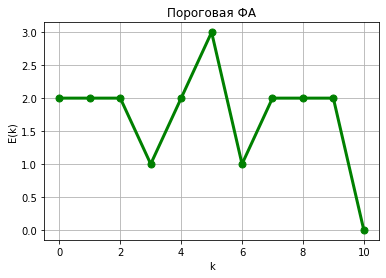


График 1.

## Обучение нейросети с частью комбинации переменных используя пороговую ФА

Последовательно увеличивая выборку количества векторов, найдем наименьшее количество необходимых для обучения векторов.

Минимальный набор обучающих векторов:

Вектор синаптических коэффициентов:

Для полного обучения потребовалось 10 эпох:

|  |  |  |  |
| --- | --- | --- | --- |
| k | Вектор весов w | Выходной вектор y | Суммарная ошибка Е |
| 0 | [0 0 0 0] | [1, 1, 1, 0, 0, 0] | 2 |
| 1 | [ 0. -0.259 0.11 0.11 ] | [1, 1, 0, 1, 0, 0] | 2 |
| 2 | [ 0. -0.329 -0.079 0.221] | [1, 1, 0, 0, 1, 0] | 2 |
| 3 | [ 0. -0.399 0.031 0.031] | [1, 1, 0, 0, 0, 0] | 1 |
| 4 | [ 0.3 -0.358 0.141 0.141] | [1, 1, 0, 1, 0, 0] | 2 |
| 5 | [ 0.3 -0.428 -0.048 0.252] | [1, 1, 0, 1, 1, 0] | 3 |
| 6 | [ 0. -0.608 -0.238 0.062] | [1, 1, 0, 0, 0, 0] | 1 |
| 7 | [ 0.3 -0.568 -0.127 0.173] | [1, 1, 0, 0, 1, 0] | 2 |
| 8 | [ 0.3 -0.637 -0.017 -0.017] | [1, 1, 0, 1, 0, 0] | 2 |
| 9 | [ 0.3 -0.707 -0.207 0.093] | [1, 1, 0, 0, 1, 0] | 2 |
| 10 | [ 0.3 -0.777 -0.096 -0.096] | [1, 1, 0, 0, 0, 1] | 0 |

На графике 2 показано поведение суммарной ошибки НС по эпохам обучения с минимальным количеством наборов (пороговая ФА).

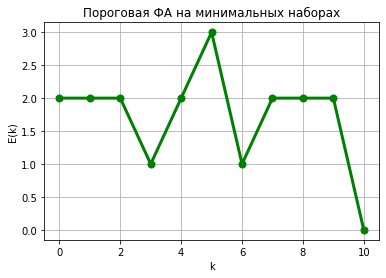


График 2.

# Выводы:

В процессе лабораторной работы было исследовано функционирование НС с радиальными базисными функциями и произведено ее обучение по правилу Видроу – Хофа.

Было произведено обучение НС с использованием пороговой ФА и RBF на всех и на минимальных наборах. Количество RBF-нейронов было равно 3, было найдено 6 минимальных наборов, на которых НС полностью обучилась. В обоих случаях НС обучилась за 10 эпох.

# Контрольные вопросы

1. Расскажите о НС RBF и алгоритме ее функционирования.

НС RBF — это нейронная сеть прямого распространения сигнала, которая содержит промежуточный (скрытый) слой радиально симметричных нейронов. Такой нейрон преобразовывает расстояние от данного входного вектора до соответствующей ему фиксированной точки пространства X по некоторому нелинейному закону, заданному радиальной функцией.

1. Назовите типы радиальных базисных функций.

Часто используемые радиально-базисные функции включают в себя :

Функция Гаусса:

Мультиквадратичная:

Обратная мультиквадратичная:

Полигармонический сплайн:

Тонкий сплайн пластины:

1. Как происходит нахождение параметров и обучение НС RBF?

Для заданной БФ количество RBF-нейронов необходимо выбирать из соотношения J = min{J0, J1}, где J0, J1 – количество векторов х = (х1, х2, х3, х4) соответствующих значениям БФ «0» и «1». Центры RBF должны совпадать с концами этих векторов.

Алгоритм функционирования НС с гауссовой RBF имеет вид:

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

Каждая эпоха обучения включает в себя цикл последовательного предъявления всех образцов обучающей выборки на вход НС. Каждый элементарный шаг обучения производится коррекция весов по правилу Видроу-Хоффа:

На каждой эпохе суммарная квадратичная ошибка Е(к) равна расстоянию Хэмминга между векторами целевого и реального выходов.

# Приложения

import numpy as np

import matplotlib.pyplot as plt

from itertools import combinations

n = 0.3

*def* F(*x*): # возвращает результат моделируемой булевой функции

    return *int*((x[2] and x[3]) or (not x[0]) or (not x[1]))

*def* Y(*net*): # возвращает результат пороговой ФА

    return 1 if net >= 0 else 0

*def* DeltaW(*x*, *q*): # находит величину, на которую изменятся Wi, для пороговой ФА

    return n \* q \* x

*def* Net(*x*, *w*): # находит значение сетевого входа НС

    return sum([w\_i \* x\_i for w\_i, x\_i in zip(w[1:], x)]) + w[0]

*def* Fi(*x*, *c*):

    return np.exp((-1) \* sum([ (x\_i - c\_i) \*\* 2  for x\_i, c\_i in zip(x, c)]))

*def* FindC(*X*):

    RightF = [F(x\_i) for x\_i in X]

    count\_0 = RightF.count(0)

    if (count\_0 <= len(RightF)/2) :

        index = [i for i, e in enumerate(RightF) if e == 0]

    else:

        index = [i for i, e in enumerate(RightF) if e == 1]

    return [X[i] for i in index]

*def* FindFi(*X*, *C*):

    fi = [[Fi(X[i], C[j]) for j in range(len(C))] for i in range(len(X))]

    return fi

*def* MinimazeSet(*X*): # находит минимальные наборы из общей выборки, на которых возможно обучение НС

    RightF = [F(x\_i) for x\_i in X]

    TryF = [0 for i in range(len(X))]

    for min\_num in range(0, len(X) + 1):

        for min\_x in *list*(combinations(X, min\_num)):

            C = FindC(min\_x)

            if (len(C) == 0): continue

            Q = 0

            fi = FindFi(min\_x, C)

            E, w = RBF(min\_x, fi)

            fi = FindFi(X, C)

            for i in range(len(X)):

                TryF[i] = Y(Net(fi[i], w))

                Q += (RightF[i] - TryF[i]) \*\* 2

            if(Q == 0):

                return E

    return []

*def* RBF(*X*, *fi*): #производит обучение НС и возвращает вектор ошибок Е(к)

                            # и вектор синаптических коэффициентов, на которых обучилась НС

    print("X :", X)

    RightF = [F(x\_i) for x\_i in X]

    w = [0 for i in range(len(fi[0]) + 1)]

    TryF = [0 for i in range(len(X))]

    E = [1]

    k = 0

    while E[k] != 0:

        print("\n\nk: ", k)

        print("w: ", np.round(w, 3))

        E.append(0)

        for i in range(len(X)):

            net = Net(fi[i], w)

            TryF[i] = Y(net)

            q = RightF[i] - TryF[i]

            for j in range(len(fi[i])):

                w[j + 1] += DeltaW(fi[i][j], q)

            w[0] += DeltaW(1, q)

            E[k+1] += q \*\* 2

        print("TryF: ", TryF)

        print("E: ", E[k+1])

        k += 1

    return E[1:], w

*def* Graph(*E*, *name*): # строит график зависимости вектора ошибок от эпохи

    if(len(E) == 0): return

    plt.plot(E, 'go-', linewidth=3, markersize=7)

    plt.grid(True)

    plt.title(name)

    plt.xlabel('k')

    plt.ylabel('E(k)')

    plt.show()

if \_\_name\_\_=="\_\_main\_\_":

    X = np.unpackbits(np.array([[j] for j in range(2 \*\* 4)], dtype=np.uint8), axis=1)[:, 4:]

    C = FindC(X)

    fi = FindFi(X, C)

    print("Пороговая ФА")

    E, w = RBF(X, fi)

    Graph(E, "Пороговая ФА")

    print("Пороговая ФА  на минимальных наборах")

    E = MinimazeSet(X)

    Graph(E, "Пороговая ФА на минимальных наборах")