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 №1 на тему:

«Исследование однослойных нейронных сетей на примере моделирования булевых выражений»

Вариант 9

Студент: Овсепян А.Н.

Группа: ИУ8-63

Москва 2021

Цель работы

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

Постановка задачи

Получить модель булевой функции на основе однослойной НС с двоичными входами $x_1, x_2, x_3, x_4 \in \{0, 1\}$, единичным входом смещения $x_0 = 1$, синоптическими весами $w_0 w_1 w_2 w_3 w_4$, двоичным выходом $y \in \{0,1\}$ и заданной функцией активации $f: R \rightarrow \{0,1\}$, реализовать обучение с использованием всех комбинаций входов и с частью возможных комбинаций.

Ход работы

Заданная функция $f = (x_1 + x_2 + x_3)(x_2 + x_3 + x_4)$

Таблица истинности БФ

x_1	x_2	x_3	x_4	f
0	0	0	0	0
0	0	0	1	0
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	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Функции активации:

1. $f(net) = \begin{cases} 1, net \geq 0 \\ 0, net < 0 \end{cases}$
2. $f(net) = \frac{1}{2}(\tanh(net) + 1)$

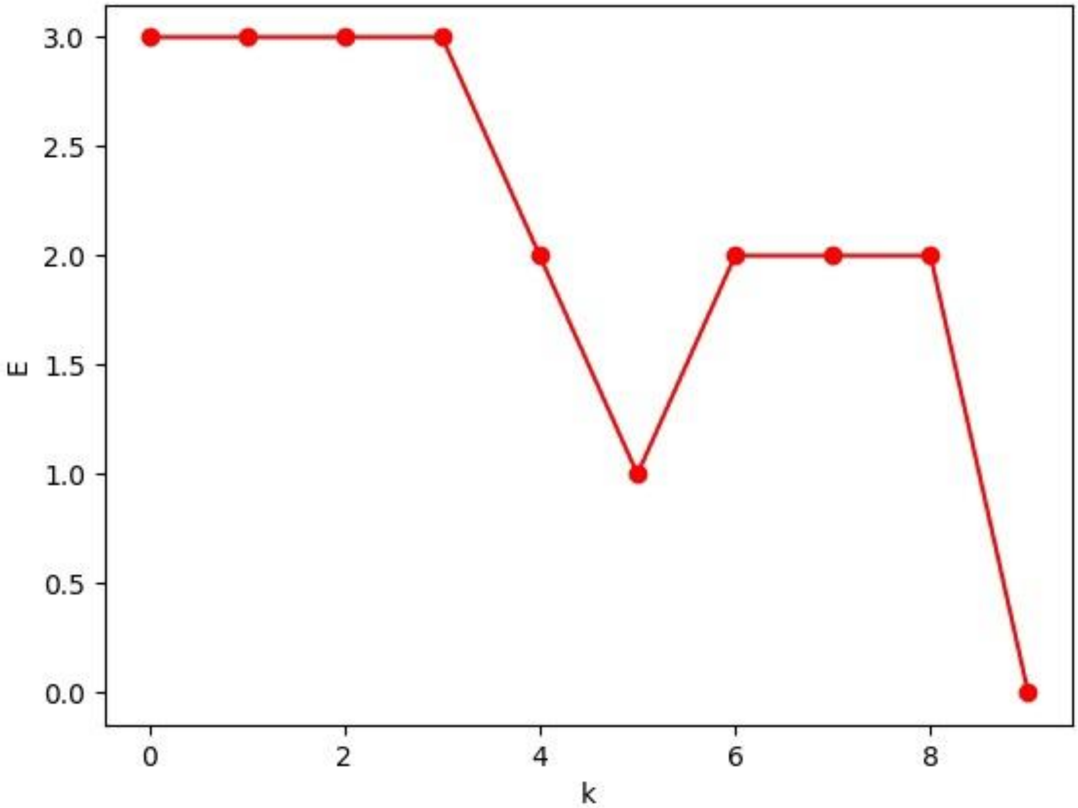
Для обучения использовалась норма обучения $\eta = 0.3$

Использование пороговой функции активации

Параметры НС на последовательных эпохах

Номер эпохи k	Вектор весов w	Выходной вектор y	Суммарная ошибка E
0	0 0 0 0 0	1111111111111111	3
1	0 0 0 0.3 0.3	1111111111111111	3
…	…	…	…
9	-0.9 0.6 0.9 0.9 0.3	0011111101111111	0

График суммарной ошибки НС по эпохам обучения

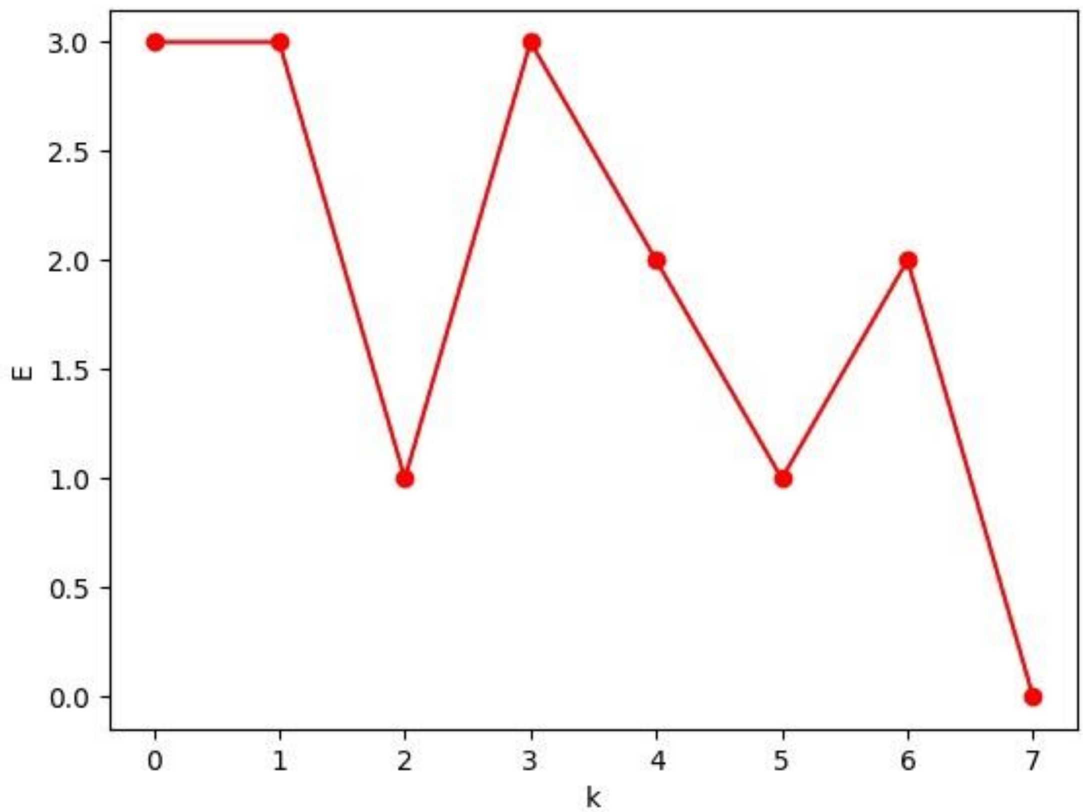


Использование сигмоидальной функции активации

Параметры НС на последовательных эпохах

Номер эпохи k	Вектор весов w	Выходной вектор y	Суммарная ошибка E
0	0 0 0 0 0	1111111111111111	3
1	0.147 0 0.15 0.140 0.147	1111111111111111	3
…	…	…	…
7	-0.294 -0.284 0.5 0.297 0.145	0011111101111111	0

График суммарной ошибки НС по эпохам обучения



Обучение с неполным набором входов

Используя сигмоидальную функцию:

Для заданной функции будет найден минимальный набор из 8 векторов:

$x_1 = (0, 0, 0, 1)$ $x_2 = (1, 0, 0, 0)$ $x_3 = (1, 0, 1, 1)$ $x_4 = (1, 1, 0, 0)$

Для обучения потребовалось 7 эпох, а конечный вектор коэффициентов

$$w = (-0.15, 0.147, 0.297, 0.437, 0.139)$$

Приложение – код программы

```
from numpy import tanh
import matplotlib.pyplot as plt
import pylab
import itertools


def f1_(x):
    return 1


    def f1(net):
        return 1 if net >= 0 else 0


    def f4(net):
        return 0.5 * (tanh(net) + 1)


    def f4_(x):
        return 0.5 - tanh(x)**2/2


    def f_y(net):
        return 1 if net >= 0.5 else 0


    def fun(x1, x2, x3, x4):
        return int((x1 or x2 or x3)*(x2 or x3 or x4))


    xs = [
            [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]
    ]


def print_data():
    true_fun = []
    for i in xs:
        x1, x2, x3, x4 = i
        true_fun.append(fun(x1, x2, x3, x4))
    print('\n F = ', true_fun)


def print_diagram(x, y):
    pylab.xlabel('k')
    pylab.ylabel('E')
    plt.plot(x, y, 'ro-')
    plt.show()


class NeuralS:

    def f_net(self, x, w, w0):
        net = w0
        for (i, j) in zip(x, w):
            net += i * j
        return net

    def __init__(self, fun, xs):
        self.true_fun = []
        self.xs = xs
        for i in self.xs:
```

```
            x1, x2, x3, x4 = i
            self.true_fun.append(fun(x1, x2, x3, x4))

        self.sum_errors = []
        self.weights = []
        self.y_exit = []

    def go(self, mass):

        self.sum_errors = []
        self.weights = []

        w0 = w1 = w2 = w3 = w4 = 0

        self.y_exit = []
        epoch_count = 0
        while True:
            sum_error = 0
            y_ = []

            w0n = w0
            w1n = w1
            w2n = w2
            w3n = w3
            w4n = w4

            for (xi, i_fun) in zip(self.xs, range(0, 16, 1)):

                net = self.f_net(xi, [w1, w2, w3, w4], w0)
                net_y_new = self.f_net(xi, [w1n, w2n, w3n, w4n], w0n)

                y = f_y(mass[0](net))
                y_new = f_y(mass[0](net_y_new))
                y_.append(y)

                error = self.true_fun[i_fun] - y
                error_y_new = self.true_fun[i_fun] - y_new

                sum_error += abs(error)
```

```

                w0n += 0.3 * error_y_new * 1 * mass[1](net_y_new)
                w1n += 0.3 * error_y_new * xi[0] * mass[1](net_y_new)
                w2n += 0.3 * error_y_new * xi[1] * mass[1](net_y_new)
                w3n += 0.3 * error_y_new * xi[2] * mass[1](net_y_new)
                w4n += 0.3 * error_y_new * xi[3] * mass[1](net_y_new)

            self.weights.append([round(w0, 3), round(w1, 3), round(w2, 3), round(w3, 3), round(w4, 3)])
            self.y_exit.append(y_)
            self.sum_errors.append(sum_error)

            w0 = w0n
            w1 = w1n
            w2 = w2n
            w3 = w3n
            w4 = w4n

            if sum_error == 0:
                break
        return self.y_exit, self.weights, self.sum_errors

    def start(self, mass):

        y_exit, weights, sum_errors = self.go(mass)

        k = 0
        print('_' * 150)
        print('||#| iteration |#|            w0          w1          w2          w3          w4          '
              '|#|                      y values                      |#|error|#||')
        print('#' * 150)
        for i in range(0, len(y_exit)):
            print('||#|    ', "%5d" % i, ' |#| ', end='    (')
            for j in weights[i]:
```

```
                print(' ', '%-7s' % str(j), end='  ')
            print(')    |#| ', y_exit[i], ' |#| ', sum_errors[i], ' |#||')
            k += 1
        print('-' * 150)
        print_diagram(range(0, k), sum_errors)


class MinNeuralWeights:
    def __init__(self, fun, xs):
        self.fun = fun
        self.true_fun = []
        self.xs = xs
        for i in self.xs:
            x1, x2, x3, x4 = i
            self.true_fun.append(fun(x1, x2, x3, x4))

    def f_net(self, x, w, w0):
        net = w0
        for (i, j) in zip(x, w):
            net += i * j
        return net

    def check_ok(self, weights, fun):
        for (xi, i_fun) in zip(self.xs, range(0, 16, 1)):
            net = self.f_net(xi, [weights[1], weights[2], weights[3], weights[4]], weights[0])
            y = f_y(fun(net))

            if self.true_fun[i_fun] - y != 0:
                return False
        return True

    def start(self, mass):
        res = []
        for i in range(1, 16, 1):
            for j in itertools.combinations(xs, 16 - i):

                used_x = list(j)
                neural = NeuralS(self.fun, used_x)
```

```
                y_exit, weights, sum_errors = neural.go(mass)

                if self.check_ok(weights[-1], f4):
                    res.append([y_exit, weights[-1], sum_errors, used_x, len(weights)])
                    break

        print(" MIN X_RANGES LEN : ", 16 - len(res))
        print(" EPOCH COUNT : ", res[-1][4])
        for (i, j) in zip(res[-1][3], range(0, len(res[-1][3]))):
            print(" x", j, " = ", i)
        print(" w = ", res[-1][1])


if __name__ == '__main__':
    print_data()

    # 1
    print()

    n1 = NeuralS(fun, xs)
    n1.start([f1, f1_])

    print()
    print('\n')

    # 2
    print()
    n2 = NeuralS(fun, xs)
    n2.start([f4, f4_])

    print()
    print('\n')
    # 3

    print()

    n3 = MinNeuralWeights(fun, xs)
    n3.start([f4, f4_])

```

```
        print()
```