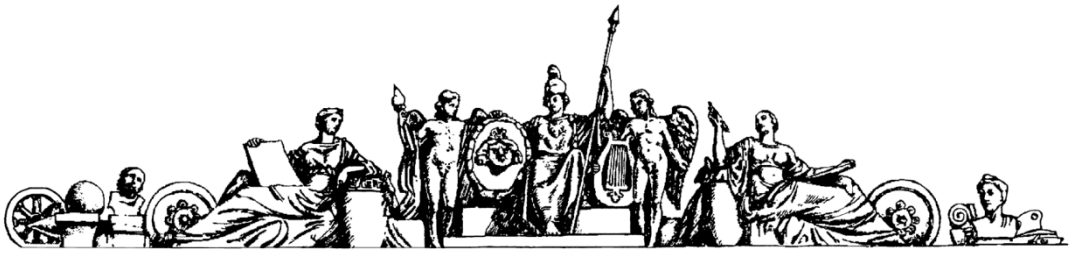

Министерство образования Российской Федерации
МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. БАУМАНА

Факультет: Информатика и системы управления
Кафедра: Информационная безопасность (ИУ8)

ИНТЕЛЛЕКТУАЛЬНЫЕ ТЕХНОЛОГИИ НА ОСНОВЕ ИСКУССТВЕННЫХ НЕЙРОННЫХ СЕТЕЙ

Лабораторная работа №1

Исследование однослойных нейронных сетей на примере моделирования булевых выражений

Вариант 13

Проверяющий: Гурова Е.Б.
Студент: Перескоков В.А.
Группа: ИУ8-61

Москва, 2018

Цель работы

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

Постановка задачи

Постановка задачи. Получить модель булевой функции (БФ) на основе однослойной НС (единичный нейрон) с двоичными входами x1,x2,x3,x4 $\in\{0,1\}$, единичным входом смещения x0 = 1, синаптическими весами , двоичным выходом y $\in\{0,1\}$ и заданной нелинейной функцией активации (ФА) f : R → (0,1) (рис. 1.1).

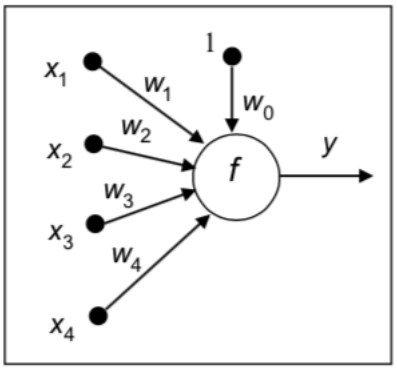

Рис. 1.1. Однослойная НС

Исходные данные в соответствии с вариантом задания

Моделируемая БФ: $(\overline{x_1} + \overline{x_2} + \overline{x_3})(\overline{x_2} + \overline{x_3} + x_4)$

Функции активации:

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

2) $f(\text{net}) = \frac{1}{2}\left(\frac{\text{net}}{1+|\text{net}|} + 1\right);$

Таблица истинности

X1	X2	X3	X4	Y
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	0
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	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

Графики суммарной квадратичной ошибки в зависимости от эпохи

На рисунке показано, что к 26 эпохе НС уже полностью обучена

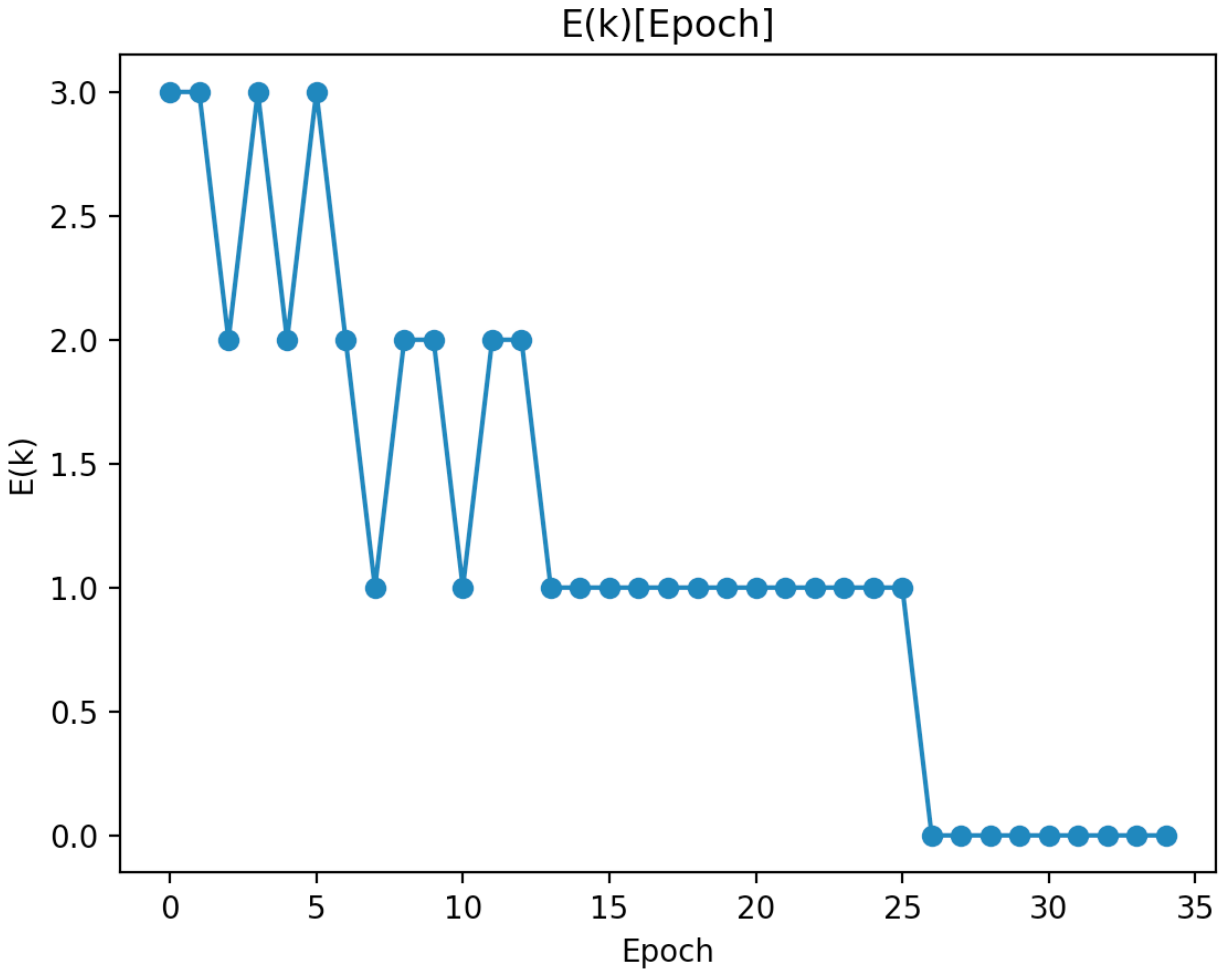

Рис. 1

На рисунке 2 мы видим уже немного другую картину.

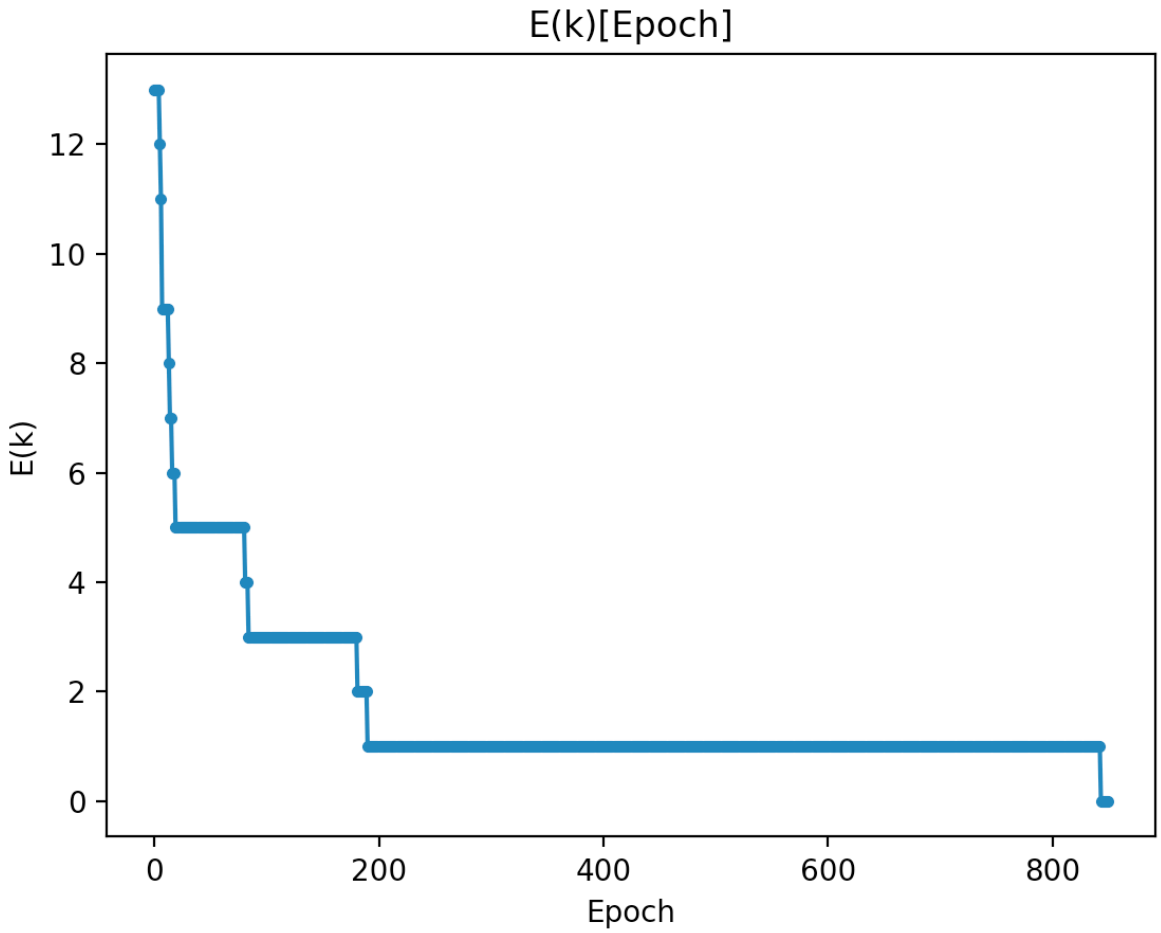

Рис. 2

Выход НС на разных эпохах (2 ФА)

E(k) = 13

X1	X2	X3	X4	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

E(k) = 9

X1	X2	X3	X4	Y
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

E(k) = 3

X1	X2	X3	X4	Y
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	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

E(k) = 1

X1	X2	X3	X4	Y
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	0
0	1	1	1	0
1	0	0	0	1
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	0
1	1	1	1	0

Минимизация

После полного перебора всех векторов было получено, что минимальный набор обучаемой выборки – 9 векторов.

API для получения результата обучения НС

Основной урл (base url) находится по адресу https://bmstu-neural-network.herokuapp.com/api/v1/lab_01

Все запросы отправляются методом POST с json объектом (request описан в таблице 1)

Url	Request	Описание	Пример
/lazy_magic	{“vars”: [X1,…,Xn]}	Для (1) ФА	{“vars”: [0, 1, 1, 1]}
/real_magic	{“vars”: [X1,…,Xn]}	Для (2) ФА	{“vars”: [0, 1, 1, 1]}

Таблица 1

Исходный код (только части, которая реализует обучение НС)

```
class BooleanNeural:
    vars = 2
    truth_table = None
    activate_function = None
    training_nu = None
    epoch_number = None
    weights = []
    info = {
        "epoch": [],
        "error": [],
        "data": [],
        "out": [],
        "net": [],
        "weights": []
    }

    '''
        Конструктор для инициализации полей
    '''
    def __init__(self, vars, truth_table, activate_function, training_nu=1.0,
epoch_number=100):
        self.truth_table = truth_table
        self.activate_function = activate_function
        self.training_nu = training_nu
        self.epoch_number = epoch_number
        self.weights = [0] * (vars + 1)

    '''
        Проведение теста над переменными
    '''
    def test(self, vars):
        reality = None
        for row in self.truth_table:
            if row[0] == vars:
                reality = row[1][0]

        return {
            "out": self.activate_function(self.__calculate_net__(vars)),
            "reality": reality
        }

    def min_search(self):
        for i in range(9):
            top = []
            for j in range(9):
                top.append(self.truth_table[j])
            info = self.training_3(table=top, debug=True)
            if 0 in info["error"]:
```

```
            print(top, info["test"])

    def training_3(self, table, debug=False, simple=False):
        for epoch in range(self.epoch_number):
            for i in range(len(table)):
                row = self.truth_table[i]
                data = row[0]

                net = self.__calculate_net__(data)
                out = self.activate_function(net)
                error = row[1][0] - out

                self.__update_weights__(data, net, out, error, simple)

            error = self.__calculate_error__()
            if debug:
                self.info["error"].append(error)
                self.info["epoch"].append(epoch)
                self.info["test"] = epoch

            if error == 0:
                return self.info if debug else None

        return self.info if debug else None

    '''
        Проведение обучения на полной таблице истинности.
        Таблица истинности генерируется под определенное число переменных
        и под нужную булеву функцию
    '''
    def training(self, debug=False, simple=False):
        for epoch in range(self.epoch_number):
            for i in range(len(self.truth_table)):
                row = self.truth_table[i]
                data = row[0]

                net = self.__calculate_net__(data)
                out = self.activate_function(net)
                error = row[1][0] - out

                self.__update_weights__(data, net, out, error, simple)

            error = self.__calculate_error__()
            if debug:
                self.info["error"].append(error)
                self.info["epoch"].append(epoch)

            if error == 0:
                return self.info if debug else None

        return self.info if debug else None

    def get_info(self):
        return self.info
```

```
    '''
        Вычисление net
    '''
    def __calculate_net__(self, data):
        net = 0
        for j in range(len(data)):
            net = net + data[j] * self.weights[j]

        return net + self.weights[len(data)]

    '''
        Обновление весов
    '''
    def __update_weights__(self, data, net, out, error, simple):
        for index_weight in range(len(self.weights)):
            self.weights[index_weight] = self.weights[index_weight] +
self.training_nu * error * (
                1 if simple else self.__df__(net)) * (
                                            data[index_weight] if index_weight !=
len(data) else 1)

    '''
        Вычисление расстояния Хэмминга
    '''
    def __calculate_error__(self):
        truth_table = []
        for index in range(len(self.truth_table)):
            row = self.truth_table[index]
            out, _, _ = self.__local_test__(row[0])
            truth_table.append([row[0], row[1], [out]])

        error = 0
        for row in truth_table:
            if row[1] != row[2]:
                error += 1

        return error

    '''
        Проверка на значение результата по Y(net)
    '''
    def __local_test__(self, vars):
        result = self.test(vars)
        y = result["out"]

        reality = result["reality"]
        except_result = 1 if y > 0.8 else 0

        return [except_result, y, reality]

    '''
        Производная (добавил в этот класс)
    '''
```

```
def __df__(self, net):
    return 0.5 * (1 / ((abs(net) + 1) ** 2))
```

Вывод

В ходе лабораторной работы было исследовано функционирование простейшей нейронной сети на базе нейрона с нелинейной функцией активации и ее обучение по правилу Видроу-Хоффа.

Было изучено правило Видроу-Хоффа – реализовано на языке python 3.x (https://github.com/vladpereskokov/BMSTU_Neural-network/tree/lab-01). Для доступа к репозиторию, напишите на почту (v.pereskokov@ivpa.ru).

Помимо реализации НС, были добавлены тесты (python.unittest) для 2х переменных и логической функции И, а также для 4х переменных и логической функции моего вариант (13).

Было получено минимальное число векторов, при которых происходит полное обучение.

На Flask'е были реализованы ручки для получения значения функции по 4 значениям переменной. https://bmstu-neural-network.herokuapp.com/api/v1/lab_01

API написано в README репозитория.

Все тесты пройдены. https://travis-ci.com/vladpereskokov/BMSTU_Neural-network/builds/67332090?utm_source=github_status&utm_medium=notification