



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

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

**Интеллектуальные технологии
информационной безопасности**

Лабораторная работа №1

**“Исследование однослойных нейронных сетей на примере
моделирования булевых выражений”**

Вариант 12

Преподаватель: Строганов Иван
Сергеевич
Студент: Кокин Даниил Сергеевич
Группа: ИУ8-61

2021 г.

1. Пороговая функция активации:

Эпоха	Веса W , выходной сигнал Y , суммарная ошибка E
0	$Y=[1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1]$, $W=[0, 0, 0, 0, 0]$, $E=5$
1	$Y=[1\ 1\ 1\ 1\ 0\ 1\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1]$, $W=[0.000\ 0.300\ -0.300\ 0.000\ 0.600]$, $E=3$
2	$Y=[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]$, $W=[0.000\ 0.600\ 0.000\ 0.300\ 0.600]$, $E=5$
...	...
19	$Y=[0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1]$, $W=[-1.500\ 1.500\ 0.300\ 0.600\ 1.200]$, $E=0$

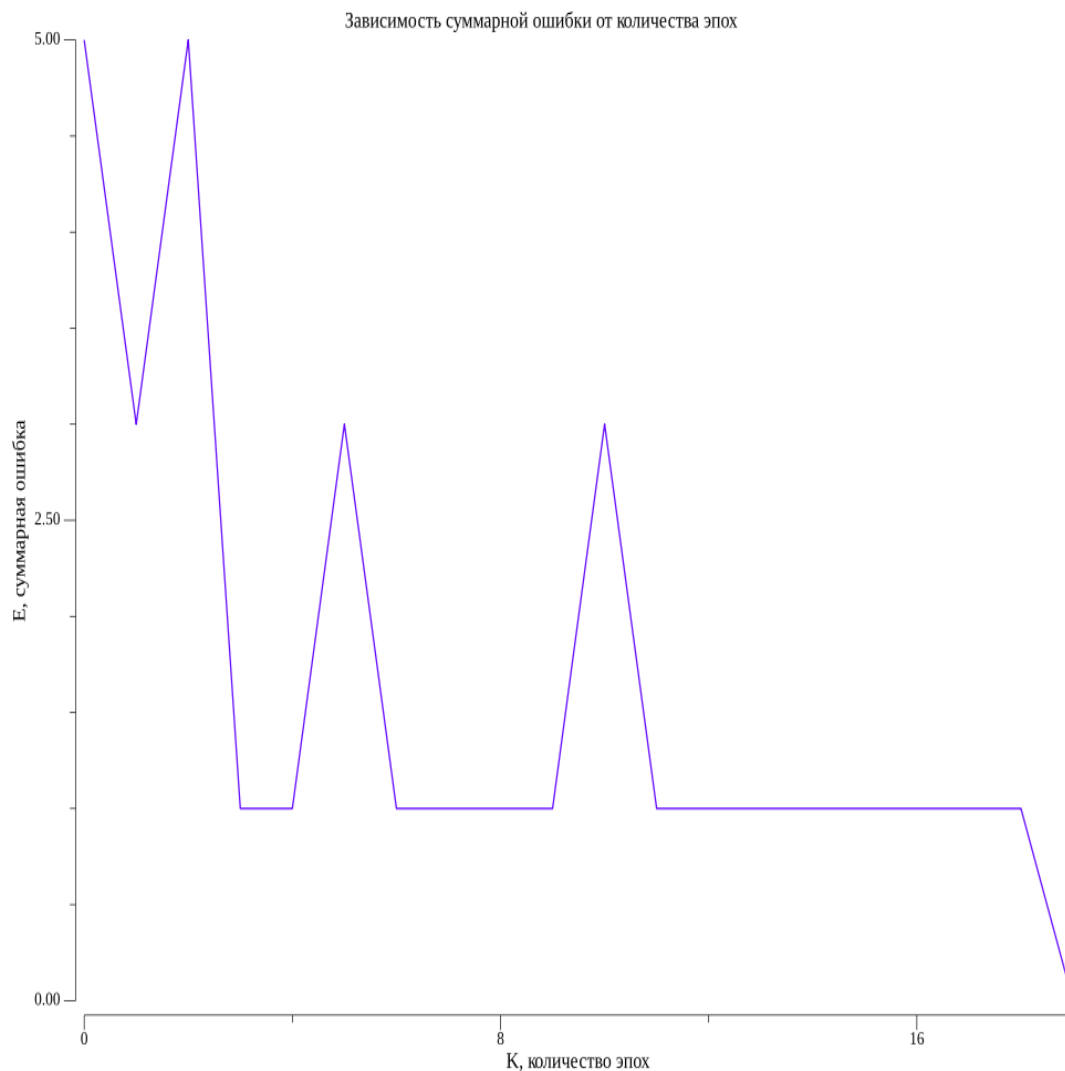


рис 1. Зависимость $E(k)$ с пороговой ФА

2. Логистическая функция активации:

Эпоха	Веса W , выходной сигнал Y , суммарная ошибка E
0	$Y=[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$, $W=[0, 0, 0, 0, 0]$, $E=11$
1	$Y=[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]$, $W=[0.600 \ 0.220 \ 0.013 \ -0.026 \ 0.886]$, $E=5$
2	$Y=[0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]$, $W=[0.300 \ 0.419 \ -0.150 \ 0.122 \ 1.200]$, $E=1$
...	...
7	$Y=[0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1]$, $W=[-0.300 \ 0.973 \ 0.258 \ 0.421 \ 0.566]$, $E=0$

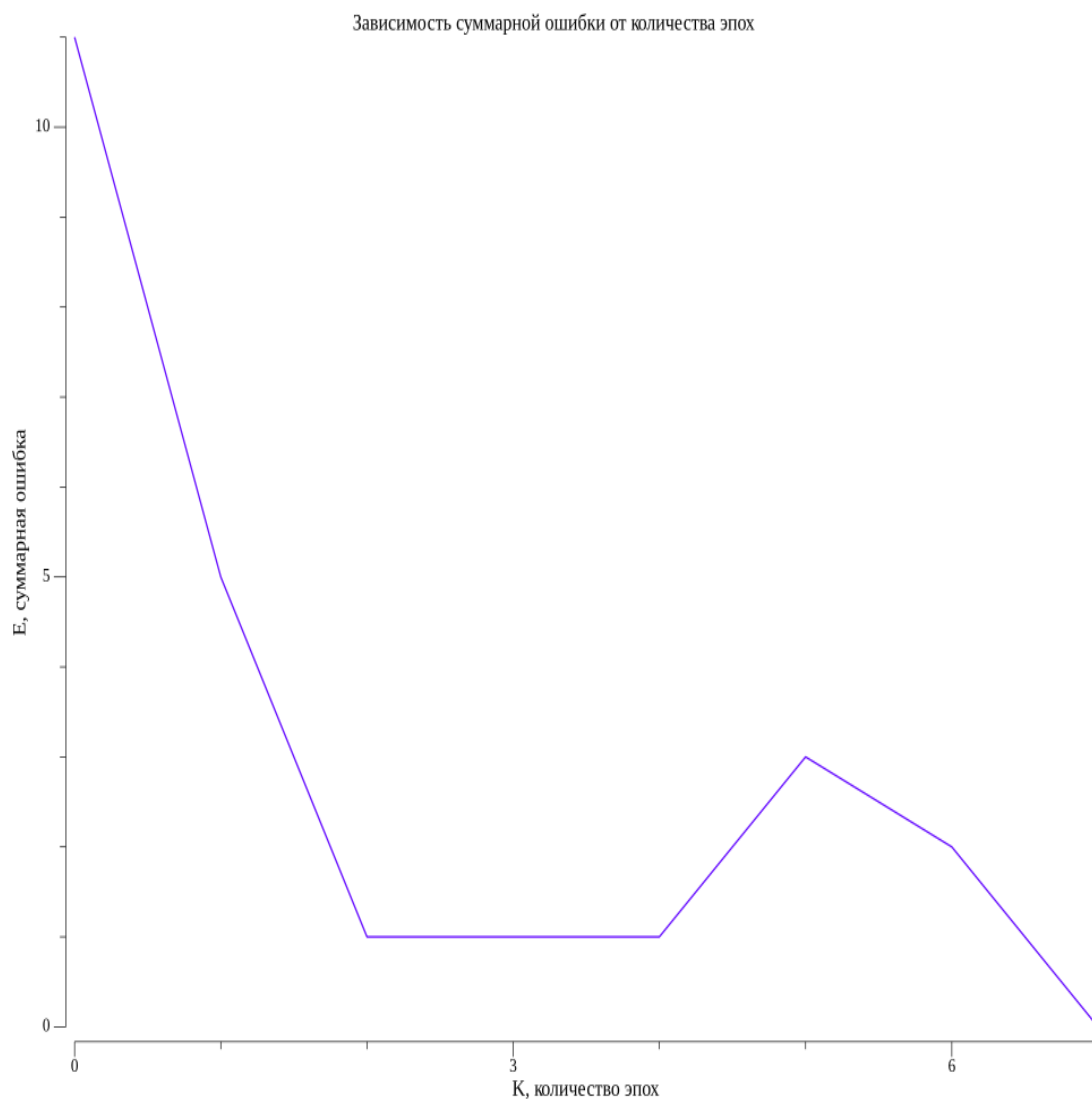
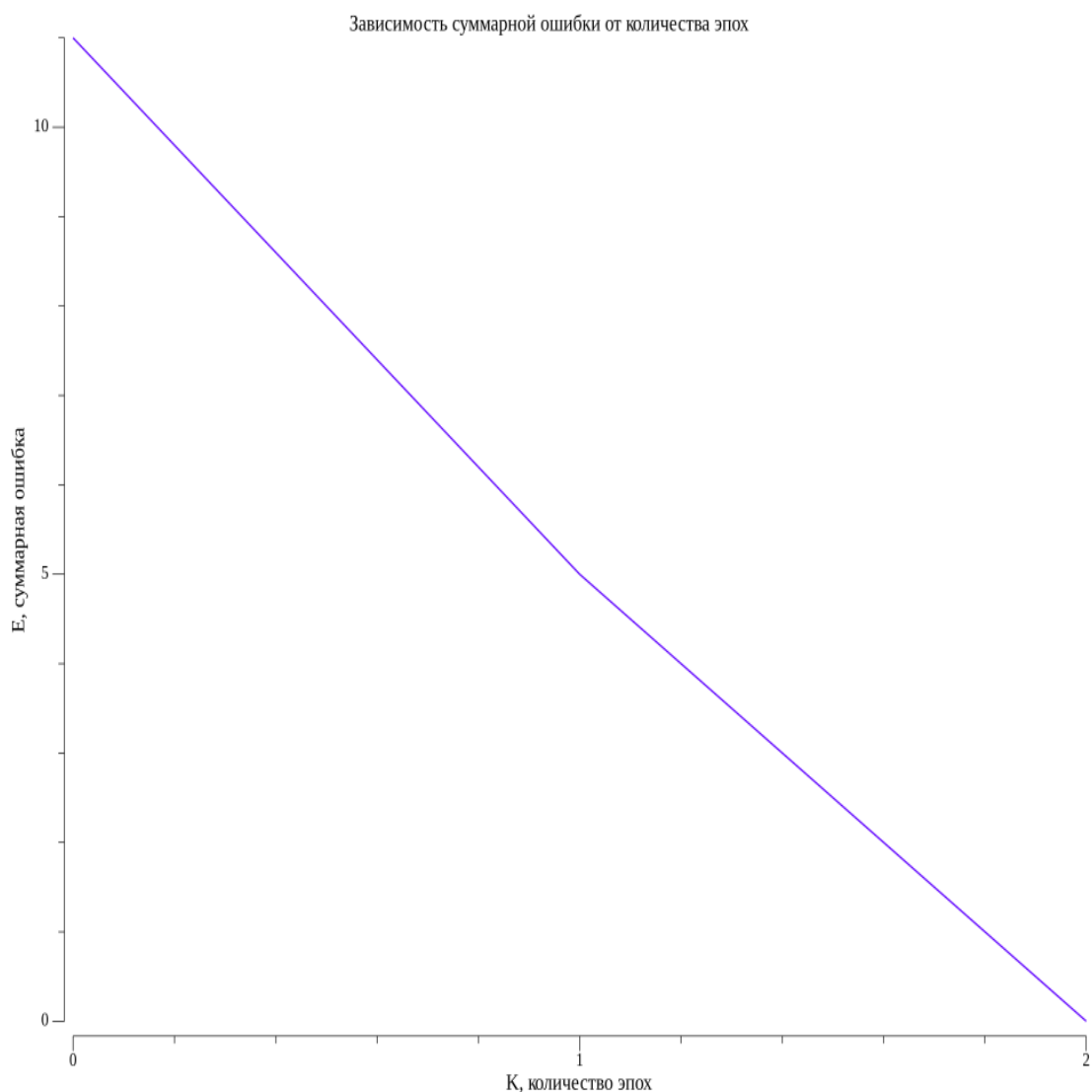


рис 2. Зависимость $E(k)$ с логистической ФА

3. Обучение на 6 наборах [0 0 1 1] [0 1 0 0] [0 1 0 1] [0 1 1 0] [1 0 0 0] [1 0 0 1] с пороговой функцией активации:

Эпоха	Веса W , выходной сигнал Y , суммарная ошибка E
0	$Y=[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$, $W=[0, 0, 0, 0, 0]$, $E=11$
1	$Y=[0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]$, $W=[0.600 \ 0.220 \ 0.164 \ 0.188 \ 0.322]$, $E=5$
2	$Y=[0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1]$, $W=[0.300 \ 0.231 \ 0.155 \ 0.039 \ 0.173]$, $E=0$



Зависимость $E(k)$ с логистической ФА и ограниченной выборкой

Листинг основной части программы, остальное можно найти
здесь: <https://github.com/dankokin/ITIB/tree/main/lab1> (golang)

```
package neuron
import (
    "fmt"
    "io"
    "itib/lab1/combinations"
    "itib/lab1/utils"
)
// Интерфейс для различных ФА с необходимыми двумя методами
// Вычисление ФА и ее производной
type ActivationFunction interface {
    Activate(float64) uint8
    Derivative([]float64, []uint8) float64
}
// Основная структура нейрона
/*
 * activationFunction - интерфейс для ФА
 * teachingRate - норма обучения
 * weights - вектор весов
 * target - вектор значений целевой функции
 * sets - все возможные наборы входных значений для БФ
 * variablesQuantity - кол-во переменных
 * writer - интерфейс для логирования: консоль либо файл
 */
type Neuron struct {
    activationFunction ActivationFunction
    teachingRate        float64
    weights              []float64
    target               []uint8
    sets                 [][]uint8
    variablesQuantity    uint8
    writer               io.Writer
}
// Функция создания объекта структуры Neuron
func CreateNeuron(function ActivationFunction, weights []float64,
    teachRate float64, target []uint8, varsQuantity uint8, writer io.Writer)
*Neuron {
    sets := utils.MakeAllSets(varsQuantity)
    return &Neuron{
        activationFunction: function,
        weights:            weights,
        teachingRate:       teachRate,
        target:             target,
        sets:               sets,
        variablesQuantity:  varsQuantity,
        writer:             writer,
    }
}
```

```

func (n *Neuron) SetOutput(writer io.Writer) {
    n.writer = writer
}
// Вычисление ФА
func (n *Neuron) getActivationFunction(set []uint8) uint8 {
    net := n.weights[0]
    for i, weight := range n.weights[1:] {
        net += weight * float64(set[i])
    }
    return n.activationFunction.Activate(net)
}
// Вычисление выходного вектора
func (n *Neuron) calculateFunctionVector() []uint8 {
    vector := make([]uint8, 0, len(n.target))
    for i := 0; i < len(n.target); i++ {
        value := n.getActivationFunction(n.sets[i])
        vector = append(vector, value)
    }
    return vector
}
// Функция для логирования промежуточных результатов
func (n *Neuron) PrintInfo(epoch uint16, err uint8, out []uint8) {
    info := fmt.Sprintf("Эпоха № %d. Выходной вектор: %v. Вектор  
весов: %.3f. Суммарная ошибка: %d",
        epoch, out, n.weights, err)
    fmt.Fprintln(n.writer, info)
}
// Обучение нейрона
func (n *Neuron) Train(epochs uint16, isPartly bool, graphicName string,
    sets ...[]uint8) bool {
    // Точки для создания графика
    xPoints := make([]float64, 0, epochs)
    yPoints := make([]float64, 0, len(n.target))
    // Если наборы, на которых необходимо обучаться, не заданы,
    берутся все возможные
    var teachSet [][]uint8
    if len(sets) == 0 {
        teachSet = n.sets
    } else {
        teachSet = sets
    }
    for epoch := uint16(0); epoch < epochs; epoch++ {
        vector := n.calculateFunctionVector()
        err := utils.HammingDistance(n.target, vector)
        xPoints = append(xPoints, float64(epoch))
        yPoints = append(yPoints, float64(err))
        // Флаг помогает избежать лишнего логирования в случае с
        нахождением минимального набора
        if !isPartly {
            n.PrintInfo(epoch, err, vector)
        }
        if err == 0 {

```

```

        if isPartly {
            fmt.Fprintf(n.writer, "Минимальный набор из %d векторов:
%v\n", len(sets), sets)
        }
        p := utils.CreatePlotter()
        p.DrawGraph(xPoints, yPoints, len(xPoints), graphicName, 100, 10,
255)
        return true
    }
    for i := 0; i < len(teachSet); i++ {
        y := n.getActivationFunction(teachSet[i])
        for j := uint8(0); j < n.variablesQuantity + 1; j++ {
            if j == 0 {
                n.weights[j] += n.teachingRate * (float64(n.target[i]) -
float64(y))
            } else {
                n.weights[j] += n.teachingRate * (float64(n.target[i]) -
float64(y)) *
                    n.activationFunction.Derivative(n.weights, teachSet[i]) *
                    float64(teachSet[i][j-1])
            }
        }
    }
}
return false
}

// Функция для обучения на неполной выборке
func (n *Neuron) TrainPartly(epochs uint16, graphicName string) bool {
    for i := 2; i < 16; i++ {
        setCombinations := combinations.Combinations(n.sets, i)
        for _, setCombination := range setCombinations {
            n.weights = []float64{0.0, 0.0, 0.0, 0.0, 0.0}
            result := n.Train(100, true, graphicName, setCombination...)
            if result {
                n.weights = []float64{0.0, 0.0, 0.0, 0.0, 0.0}
                n.Train(epochs, false, graphicName, setCombination...)
                return true
            }
        }
    }
    return false
}

```


Вывод

В данной работе реализован перцептрон, обучающий по правилу Видроу-Хоффа простейшую нейронную сеть. В соответствии со входной функцией активации программа выдает отклик на входные данные. Был уменьшен размер обучающей выборки и для БФ были найдены шесть наборов, на которых можно обучить функцию. При изменении нормы обучения меняется количество эпох. При правильном подборе можно избежать большого числа эпох.