# Практическая работа №1

# Исследование однослойных нейронных сетей на примере моделирования булевых выражений

**Цель работы** — исследовать функционирование простейшей нейронной сети (НС) на базе нейрона с нелинейной функцией активации и обучить её по правилу Видроу — Хоффа.

***Постановка задачи.*** Получить модель булевой функции (БФ) на основе однослойной НС (единичный нейрон) с двоичными входами , единичным входом смещения , синаптическими весами , двоичным выходом и заданной нелинейной функцией активации (ФА) (рис. 1.1).

Для заданной БФ (варианты см. в табл. 1.4) реализовать обучение НС с использованием:

1. всех комбинаций переменных ;
2. части возможных комбинаций переменных ;

остальные комбинации являются тестовыми.

Получим нейросетевую модель БФ   
(табл. 1.1):

*Таблица 1.1*

**Таблица истинности БФ**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  | F = (X3\*X4)+!X1+!X2 |
| 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 |

Функции активации:

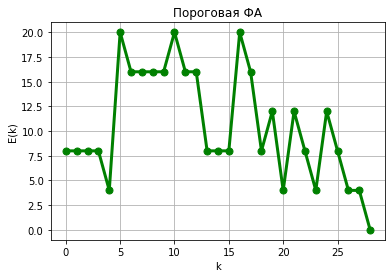
Пороговая ФА:

Сигмоидальная ФА: .

### Этап 1:

Использование пороговой функции активации, обучение ИНС ведется с использованием всех комбинаций переменных

|  |  |  |  |
| --- | --- | --- | --- |
| Но­- мер эпохи  *k* | Вектор весов **w** | Выходной вектор **у** | Сум­марная ошиб- ­ка *E* |
| 0 | (0, 0, 0, 0, 0) | (1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0) | 8 |
| 1 | (1.1102230246251565e^-16, 0.0, 0.0, 0.3, 0.3) | (1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0) | 8 |
| ... | ... | ... | ... |
| 28 | (3.599999999999999, -2.9999999999999996, -2.4, 1.8, 1.5) | (1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1) | 0 |

  
Рис. 1. График суммарной ошибки НС по эпохам обучения (пороговая ФА).

### Этап 2:

Использование логистической функции активации, обучение ИНС ведется с использованием всех комбинаций переменных

|  |  |  |  |
| --- | --- | --- | --- |
| Но­- мер эпохи  *k* | Вектор весов **w** | Выходной вектор **у** | Сум­марная ошиб- ­ка *E* |
| 0 | (0, 0, 0, 0, 0) | (1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0) | 8 |
| 1 | (-0.307849583310995, -0.07696239582774878, -0.07696239582774878, 0.07303760417225122, 0.07303760417225122) | (0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0) | 12 |
| ... | ... | ... | ... |
| 41 | (1.7518525860569518, -1.6131808311090758, -1.2427707136984534, 1.0230748343844145, 0.8233767291844423) | (1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1) | 0 |

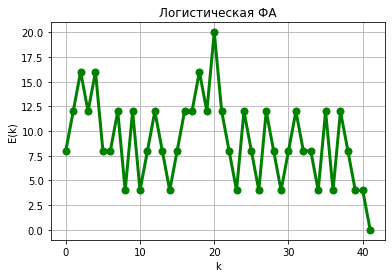


Рис. 2. График суммарной ошибки НС по эпохам обучения (сигмоидальная ФА).

### Этап 3:

Обучение НС с использованием части комбинации переменных используя пороговую ФА. Последовательно увеличивая выборку количества векторов, найдем наименьшее количество необходимых для обучения векторов.

Минимальный набор обучающих векторов:

Вектор синаптических коэффициентов:

W = [2.4, -1.5, -1.2, 0.3, 0.3]

Для обучения ИНС понадобилось 11 эпох.

|  |  |  |  |
| --- | --- | --- | --- |
| Но­- мер эпохи  *k* | Вектор весов **w** | Выходной вектор **у** | Сум­марная ошиб- ­ка *E* |
| 0 | (0, 0, 0, 0, 0) | (1, 1, 1, 1) | 4 |
| 1 | (-1.2, -0.3, -0.3, -0.3, 0) | (0, 0, 1, 1) | 12 |
| ... | ... | ... | ... |
| 11 | (2.4, -1.5, -1.2, 0.3, 0.3) | (1, 1, 1, 0) | 0 |

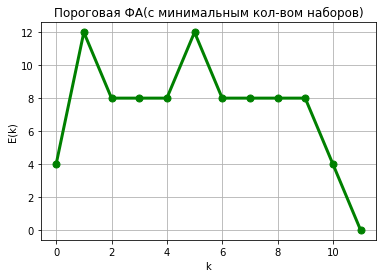


Рис. 3. График суммарной ошибки НС по эпохам обучения с минимальным количеством наборов (пороговая ФА).

### Этап 4:

Обучение НС с использованием части комбинации переменных используя сигмоидальную ФА. Последовательно увеличивая выборку количества векторов, найдем наименьшее количество необходимых для обучения векторов.

Минимальный набор обучающих векторов:

Вектор синаптических коэффициентов:

W = [0.39667643530238905, -0.2982212040765382, -0.2982212040765382, 0.14559606755675975, 0.14559606755675975]

Для обучения ИНС понадобилось 3 эпохи.

|  |  |  |  |
| --- | --- | --- | --- |
| Но­- мер эпохи  *k* | Вектор весов **w** | Выходной вектор **у** | Сум­марная ошиб- ­ка *E* |
| 0 | (0, 0, 0, 0, 0) | (1, 1, 1) | 4 |
| 1 | (-0.6, -0.15, -0.15, 0.0, 0.0) | (0, 0, 1) | 12 |
| ... | ... | ... | ... |
| 3 | (0.39667643530238905, -0.2982212040765382, -0.2982212040765382, 0.14559606755675975, 0.14559606755675975) | (1, 1, 0) | 0 |

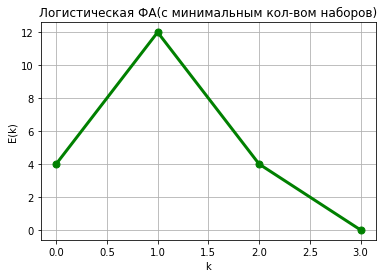


Рис. 4. График суммарной ошибки НС по эпохам обучения с минимальным количеством наборов (логистическая ФА).

### Выводы:

В процессе лабораторной работы было исследовано функционирование простейшей нейронной сети на базе нейрона с нелинейной функцией активации и обучение ее по правилу Видроу – Хоффа. Были обучены НС с использованием пороговой и логистической ФА. А также произведено обучение с использованием части комбинаций переменных. В результате, НС с использованием пороговой ФА была быстрее обучена на минимальном количестве наборов, чем на полных наборах. А НС с использованием логистической ФА была, наоборот, обучена за меньшее количество эпох на полных наборах.

### Приложения:

import numpy as np

import matplotlib.pyplot as plt

from itertools import combinations

n = 0.3 #норма обучения

*def* Hemming(*f1*, *f2*): # возвращает расстояние Хэмминга между наборами  f1 и f2

    e = 0

    for x1, x2 in zip(f1, f2):

        if(x1 != x2) : e += 1

    return e

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

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

*def* Y1(*net*): # возвращает результат пороговой функции активации

    return 1 if net >= 0 else 0

*def* Y2(*net*): # возвращает результат логистической функции активации

    out = 0.5 \* (np.tanh(net) + 1)

    return 1 if out >= 0.5 else 0

*def* DeltaW1(*x*, *q*, *net*): # находит величину, на которую изменятся семантические веса Wi, для ПОРОГОВОЙ функции активации

    return n \* q \* x

*def* DeltaW2(*x*, *q*, *net*): # находит величину, на которую изменятся семантические веса Wi, для ЛОГИСТИЧЕСКОЙ функции активации

    return n \* q \* x \* ((-0.5) \* (np.tanh(net) \*\* 2) + 0.5)

*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]

#производит обучение НС. Возвращает вектор ошибок Е(к) и вектор синаптических коэффициентов, на которых обучилась НС

*def* LearningNN(*X*, *Y*, *DeltaW*):

    RightF = [F(x\_i) for x\_i in X] # Результат моделируемой булевой функции по всем элементам x\_i в списке Х

    w = [0 for i in range(5)] #задаем синаптические веса нулями

    TryF = [0 for i in range(len(X))] #задаем сеетвые значения нулями

    E = [1] #суммарная ошибка

    k = 0  #эпоха

    while E[k]:

        print("\n\nЭпоха =", k, "\n")

        print("Синаптические веса =", w)

        E.append(0)

        for i in range(len(X)):

            net = Net(X[i], w)

            TryF[i] = Y1(net) ###Y

            q = RightF[i] - TryF[i]

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

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

                  w[0] += DeltaW(1, q, net)

                  E[k+1] += q \*\* 2

        k += 1

        print( TryF)

        print("\nСуммарная ошибка : ", E[k])

    return E, w

*def* MinimazeSet(*X*, *Y*, *DeltaW*, *name*): # 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(0, len(X))]

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

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

            E, w = LearningNN(min\_x, Y, DeltaW)

            for i in range(0, len(X)):

                net = Net(X[i], w)

                TryF[i] = Y2(net)

            if(Hemming(RightF, TryF) == 0) :

                Graph(E, name + "(c минимальным кол-вом наборов)")

                return min\_x, w

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

    plt.plot(E[1:], '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(0, pow(2, 4))], dtype=np.uint8), axis=1)[:, 4:]

    E, w = LearningNN(X, Y1, DeltaW1)

    Graph(E, "Пороговая ФА")

    E, w = LearningNN(X, Y2, DeltaW2)

    Graph(E, "Логистическая ФА")

    min\_x, w = MinimazeSet(X, Y1, DeltaW1, "Пороговая ФА")

    print("\n\nmin\_x :", min\_x, "\n\nw =", w)

    min\_x, w = MinimazeSet(X, Y2, DeltaW2, "Логистическая ФА")

    print("\n\nmin\_x :", min\_x, "\n\nw =", w)