**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

отчет

**по лабораторной работе №3**

**по дисциплине «Основы искусственного интеллекта»**

Тема: Нейронная сеть Хемминга

|  |  |  |
| --- | --- | --- |
| Студенты гр. 9302 |  | Ширнин К.В. |
|  |  | Квитко Д.В. |
| Преподаватель |  | Новакова Н.Е. |

Санкт-Петербург

2022

**Содержание**

[Цель работы 3](#_Toc115821449)

[Реализация алгоритма 3](#_Toc115821450)

[Ход работы 5](#_Toc115821451)

[Пример работы программы 6](#_Toc115821452)

[Вывод 8](#_Toc115821453)

[Список используемой литературы 8](#_Toc115821454)

# Цель работы

Приобретение и закрепление знаний и получение практических навыков работы с нейронной сетью Хемминга.

**Теоритические сведенья**

Нейро́нная сеть Хэ́мминга — вид нейронной сети, использующийся для классификации бинарных векторов, основным критерием в которой является расстояние Хэмминга. Является развитием нейронной сети Хопфилда.

Сеть используется для того, чтобы соотнести бинарный вектор

где , с одним из эталонных образов (каждому классу соответствует свой образ), или же решить, что вектор не соответствует ни одному из эталонов. В отличие от сети Хопфилда, выдаёт не сам образец, а его номер.

Сеть Хэмминга — трёхслойная нейронная сеть с обратной связью. Количество нейронов во втором и третьем слоях равно количеству классов классификации. Синапсы нейронов второго слоя соединены с каждым входом сети, нейроны третьего слоя связаны между собой отрицательными связями, кроме синапса, связанного с собственным аксоном каждого нейрона — он имеет положительную обратную связь.

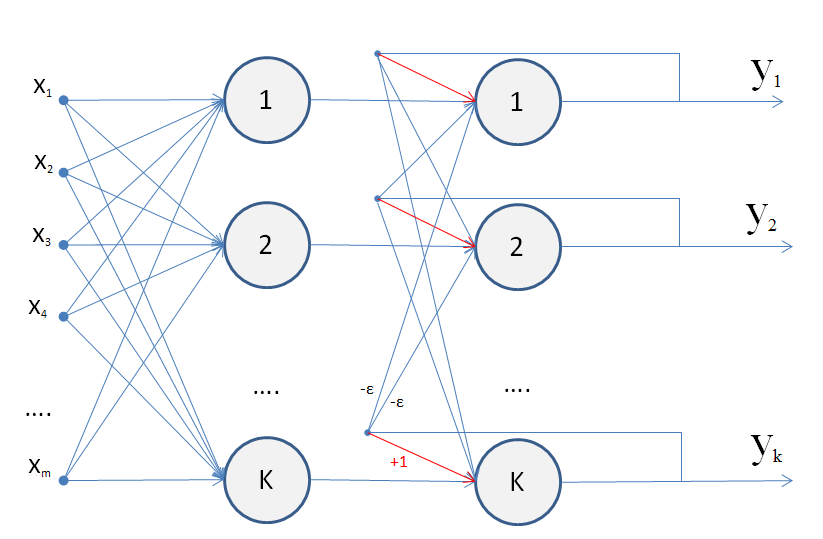


Рис.1.Схема сети Хемминга

Матрица весовых коэффициентов первого слоя получается из матрицы эталонных образов {\displaystyle X}X как {\displaystyle w\_{ij}={\dfrac {x\_{ij}}{2}}} , где матрица эталонных образов — это матрица , {\displaystyle K\times M}, каждая строка которой — соответствующий эталонный бинарный вектор. Функция активация определяется как

{\displaystyle f(s)=\left\{{\begin{matrix}0,&s\leqslant 0,\\s,&0<s\leqslant T;\\T,&s>T\\\end{matrix}}\right.};

где {\displaystyle T={\dfrac {M}{2}}}

Матрица весовых коэффициентов второго слоя имеет размер{\displaystyle K\times K}, и определяется как

,

{\displaystyle {\begin{bmatrix}1&-\epsilon &\cdots &-\epsilon \\-\epsilon &1&\cdots &-\epsilon \\\cdots &\cdots &\cdots &\cdots \\-\epsilon &-\epsilon &\cdots &1\end{bmatrix}},}

где {\displaystyle \epsilon \in (0,{\dfrac {1}{K}}]}

Таким образом, обучение производится за один цикл.

На вход подаётся классифицируемый вектор {\displaystyle {\vec {x^{\*}}}}. Состояние нейронов первого слоя рассчитывается как {\displaystyle s\_{1j}=w\_{ji}x\_{i}^{\*}}. Выход нейронов первого слоя получается путём применения функции активации к состоянию, и становится начальным значением соответствующих нейронов второго слоя. Далее, состояния нейронов второго слоя получаются из их предыдущего состояния, исходя из матрицы весовых коэффициентов второго слоя, и процедура повторяется итерационно до стабилизации вектора состояния второго слоя — пока норма разницы векторов двух последовательных итераций не станет меньше определённого значения {\displaystyle E\_{max}}(на практике достаточно значений порядка 0,1).

В случае, если в итоге один вектор положительный, а остальные отрицательные, то он указывает на подходящий образец. В случае же, если несколько векторов положительны, и при этом, не один из них не превышает  {\displaystyle E\_{max}}, то это значит, что нейросеть не может отнести входящий вектор ни к одному из классов, однако положительные выходы указывают на наиболее схожие эталоны.

# Особенности реализации алгоритма

Для реализации нейронной сети Хэмминга ПО было разделено на две составляющие, а именно на GUI и сам программный код алгоритма. Программный комплекс был написан в рамках правил ООП.

Ход алгоритма делится на несколько этапов – он делится на итерации. После каждой итерации производится обновление количества феромона на дорожках – рёбрах графа – с учетом скорости их испарения и других заданных пользователем параметров. Одна итерация, в свою очередь, делится на проход пачки муравьев. Их количество задается пользователем. Каждый муравей из пачки может пойти в абсолютно любую другую смежную вершину, даже если на этой дорожке большое количество феромона. Это происходит из-за того, что выбор муравья определяется случайной составляющей. С учетом удаленности вершины и рассыпанного по пути к ней феромона муравью более привлекательна та или иная вершина, но даже у самой непривлекательной точки будет иметься вероятность посещения её муравьем. Это позволяет изменить даже, самый лучший маршрут на другой и найти еще более оптимальный путь. После того как все муравьи прошли все вершины - производится обновление количества феромона на дорожках. Сначала испаряется старый феромон с заданным коэффициентом. После, ко всем пройденным маршрутам муравьями на данной итерации добавляется некоторое количество феромона.

# UML-диаграмма

На рисунке 1 представлена UML-диаграмма структуры программы

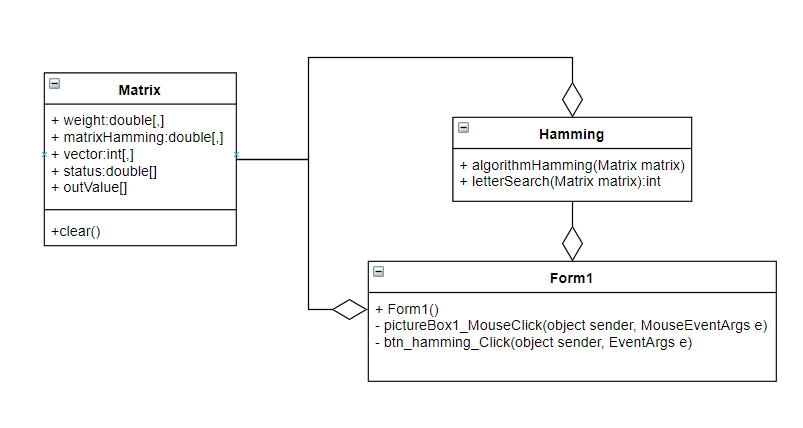


Рис. 2. UML-диаграмма

Таблица 1. Методы и их описание

|  |  |
| --- | --- |
| Метод | Описание |
| public void clear() | Обнуляет значения массивов status и outValue |
| static public int letterSearch(Matrix matrix) | Стабилизация вектора |
| static public void algorithmHamming(Matrix matrix) | Получение состояний нейронов |
| private void pictureBox1\_MouseClick(object sender, MouseEventArgs e) | Формирует входной вектор |
| public Form1() | Конструктор класса Form1 |
| private void btn\_hamming\_Click(object sender, EventArgs e) | Запускает работу алгоритма |

# 

# Ход работы

Для выполнения данной лабораторной работы был выбран язык C#.

Нейронная сеть сразу же обучается распознаванию 6 букв К, В, И. Т, О, Д. Пользователь не может менять буквы которые распознает сеть. Для отрисовки букв был использован PictureBox разделенный на 25 сегментов, нажатие на один из сегментов меняет его цвет на черный, либо белый, это изменение заносится в вектор как 1 или -1 соответственно.

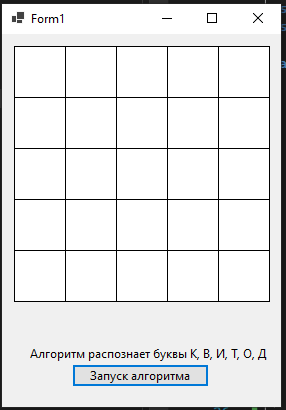


Рис.3. Пользовательский интерфейс

При запуске алгоритма, сеть обрабатывает входной вектор и записывает в массив status статус нейронов, дальше используется функция if для отбора положительных значений в массиве status. Дальше идет цикл который и ищет индекс нарисованной пользователем буквы, функция возвращает индекс найденной буквы, либо -1 если похожая не найдена.

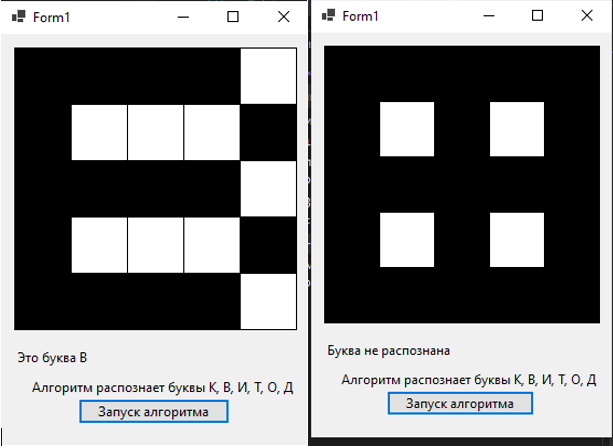


Рис.4. Результат работы программы при обнаружении и не распознании буквы

# Пример работы программы

Пользователь нарисовал букву Т, программа определила, что это буква Т

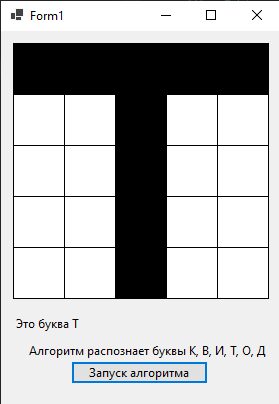


Рис.5. Работа программы при вводе буквы Т

Пользователь нарисовал букву О, программа определила, что это буква О

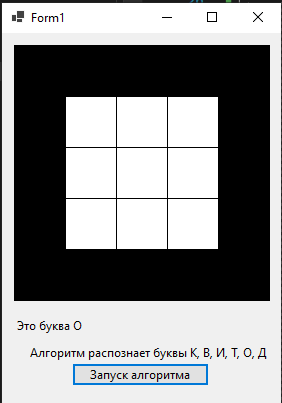


Рис.6. Работа программы при вводе буквы О

Пользователь нарисовал нечеткую букву К, программа определила, что это буква К

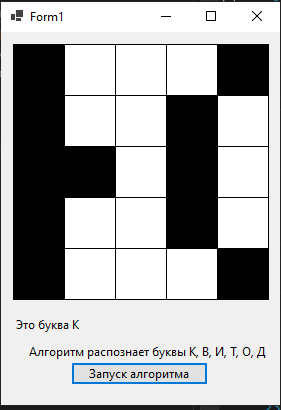


Рис.7. Работа программы при вводе буквы К

Пользователь нарисовал нечеткую букву И, программа определила, что это буква И

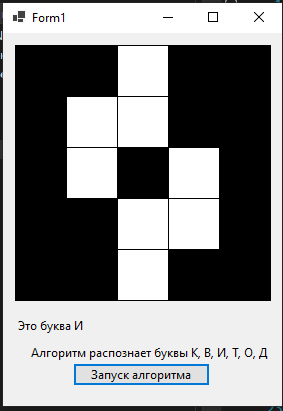


Рис.8. Работа программы при вводе буквы И

# Вывод

В ходе выполнения лабораторной работы была реализована нейронная сеть Хемминга, которая используется для определения чёрно-белого изображения размером 5 на5. Нейронная сеть легко определяет нарисованные пользователем буквы, но если вектора некоторых букв очень сильно схожи, то сеть будет путаться, либо вовсе не сможет найти букву.

# Список используемой литературы

1. <https://neuronus.com/theory/nn/965-nejronnye-seti-khemminga.html>
2. <https://ru.wikipedia.org/wiki/Нейронная_сеть_Хэмминга>

# Листинг

**Form1.cs**

using System.Drawing;

using System.Runtime.ConstrainedExecution;

namespace HammingNeuralNetworks

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

pictureBox1.Image = new Bitmap(pictureBox1.Width,pictureBox1.Height);

using (Graphics g = Graphics.FromImage(pictureBox1.Image))

{

Pen p = new Pen(Color.Black);

for (int y = 0; y < pictureBox1.Height; y += 51)

{

g.DrawLine(p, y, 0, y, pictureBox1.Height);

g.DrawLine(p, 0, y, pictureBox1.Height, y);

}

}

for (int i = 0; i < 6; i++)

{

for (int j = 0; j < 25; j++)

{

matrix.weight[i, j] = matrix.matrixHamming[i, j] / 2;

}

}

label1.Visible = false;

}

Matrix matrix = new Matrix();

private void pictureBox1\_MouseClick(object sender, MouseEventArgs e)

{

int x = e.X / 51;

int y = e.Y / 51;

Graphics g = pictureBox1.CreateGraphics();

SolidBrush solidBrush;

if (matrix.vector[0,x + y \* 5] == -1)

{

solidBrush = new SolidBrush(Color.Black);

matrix.vector[0, x + y \* 5] = 1;

}

else

{

solidBrush = new SolidBrush(Color.White);

matrix.vector[0,x + y \* 5] = -1;

}

g.FillRectangle(solidBrush, x \* 50 + x + 1, y \* 50 + y + 1, 50, 50);

}

private void btn\_hamming\_Click(object sender, EventArgs e)

{

Hamming.algorithmHamming(matrix);

switch (Hamming.letterSearch(matrix))

{

case -1:

label1.Text = "Буква не распознана";

break;

case 0:

label1.Text = "Это буква К";

break;

case 1:

label1.Text = "Это буква В";

break;

case 2:

label1.Text = "Это буква И";

break;

case 3:

label1.Text = "Это буква Т";

break;

case 4:

label1.Text = "Это буква О";

break;

case 5:

label1.Text = "Это буква Д";

break;

}

label1.Visible = true;

}

}

}

**Hamming.cs**

using System;

using System.Collections.Generic;

using System.Diagnostics.Metrics;

using System.Drawing.Drawing2D;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace HammingNeuralNetworks

{

internal class Hamming

{

static public void algorithmHamming(Matrix matrix)

{

for (int i = 0; i < 6; i++)

{

for (int j = 0; j < 25; j++)

matrix.status[i] += matrix.weight[i, j] \* matrix.vector[0, j];

matrix.status[i] += 25 / 2;

}

for (int i = 0; i < 6; i++)

if (matrix.status[i] < 0)

matrix.outValue[i] = 0;

else

matrix.outValue[i] = matrix.status[i];

}

static public int letterSearch(Matrix matrix)

{

double Emax = 0.1;

for (int j = 0; j < 10; j++)

{

double feedbackSynapses = 1.0 / 6.0;

for (int i = 0; i < 6; i++)

{

matrix.status[i] = matrix.outValue[i] - feedbackSynapses \* (matrix.outValue[0] + matrix.outValue[1] + matrix.outValue[2]

+ matrix.outValue[3] + matrix.outValue[4] + matrix.outValue[5] - matrix.outValue[i]);

}

for (int i = 0; i < 6; i++)

if (matrix.status[i] < 0)

matrix.outValue[i] = 0;

else

matrix.outValue[i] = matrix.status[i];

int k = 0, cur = 0;

for (int i = 0; i < 6; i++)

{

if (matrix.outValue[i] != 0)

{

k++;

cur = i;

}

}

if (k == 1)

{

if (matrix.outValue[cur] > Emax)

{

matrix.clear();

return cur;

}

}

}

return -1;

}

}

}

**Matrix.cs**

using System;

using System.Collections.Generic;

using System.Diagnostics.Metrics;

using System.Drawing.Drawing2D;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace HammingNeuralNetworks

{

internal class Hamming

{

static public void algorithmHamming(Matrix matrix)

{

for (int i = 0; i < 6; i++)

{

for (int j = 0; j < 25; j++)

matrix.status[i] += matrix.weight[i, j] \* matrix.vector[0, j];

matrix.status[i] += 25 / 2;

}

for (int i = 0; i < 6; i++)

if (matrix.status[i] < 0)

matrix.outValue[i] = 0;

else

matrix.outValue[i] = matrix.status[i];

}

static public int letterSearch(Matrix matrix)

{

double Emax = 0.1;

for (int j = 0; j < 10; j++)

{

double feedbackSynapses = 1.0 / 6.0;

for (int i = 0; i < 6; i++)

{

matrix.status[i] = matrix.outValue[i] - feedbackSynapses \* (matrix.outValue[0] + matrix.outValue[1] + matrix.outValue[2]

+ matrix.outValue[3] + matrix.outValue[4] + matrix.outValue[5] - matrix.outValue[i]);

}

for (int i = 0; i < 6; i++)

if (matrix.status[i] < 0)

matrix.outValue[i] = 0;

else

matrix.outValue[i] = matrix.status[i];

int k = 0, cur = 0;

for (int i = 0; i < 6; i++)

{

if (matrix.outValue[i] != 0)

{

k++;

cur = i;

}

}

if (k == 1)

{

if (matrix.outValue[cur] > Emax)

{

matrix.clear();

return cur;

}

}

}

return -1;

}

}

}

**Program.cs**

namespace HammingNeuralNetworks

{

internal static class Program

{

/// <summary>

/// The main entry point for the application.

/// </summary>

[STAThread]

static void Main()

{

// To customize application configuration such as set high DPI settings or default font,

// see https://aka.ms/applicationconfiguration.

ApplicationConfiguration.Initialize();

Application.Run(new Form1());

}

}

}