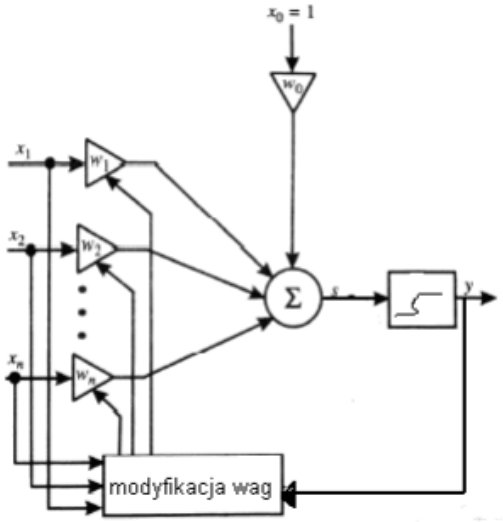
Jan Dudek

Sprawozdanie ze scenariusza 4

1. Cel Ćwiczenia
   Poznanie działania reguły uczenia Hebba na przykładzie rozpoznawania emotikon.

2. Syntetyczny opis
   Model neuronu Hebba:
   Sieć Hebba ma podobną strukturę do sieci Adaline. Różnice tkwi w regule uczenia, która występuje w wersji z nauczycielem oraz bez nauczyciela. W regule Hebba może się też pojawić współczynnik zapominania, który wspomaga proces uczenia.



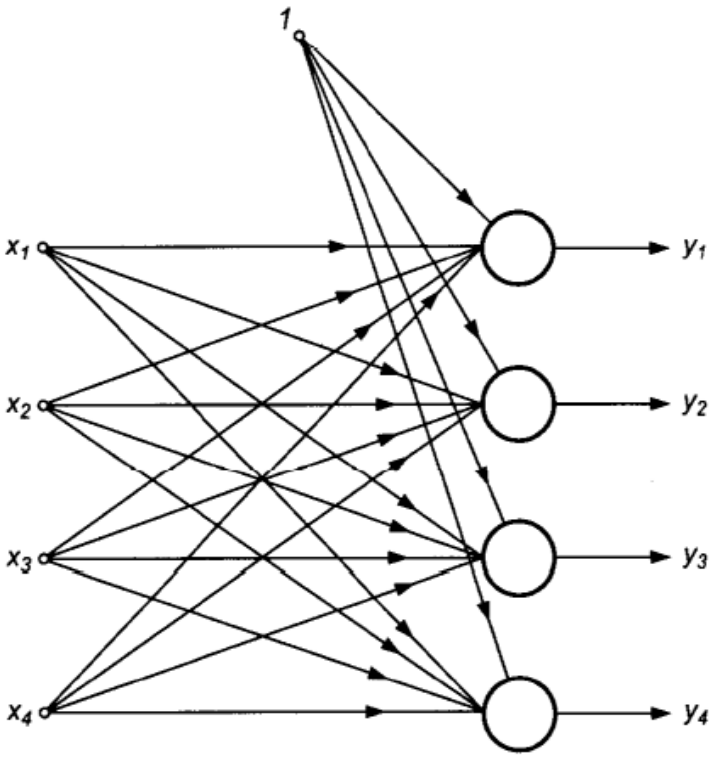
   Algorytm uczenia Hebba (bez nauczyciela):

$$W(t+1) = W(t)+nyx$$

   Gdzie:
   W – waga konkretnego wejścia
   t – numer iteracji
   y – sygnał wyjściowy neuronu
   x – sygnał wejściow neuronu
   n – współczynnik uczenia
   Algorytm ten można wzbogacić o współczynnik uczenia γ
   Oraz zastosować uczenie nadzorowane (sygnał wyjściowy zastępuje się sygnałem oczekiwanym d), wówczas:

$$W(t+1) = (1- \gamma)W(t)+nxd$$

   Opis danych uczących:
   Dane uczące stanowi zestaw 4 emotikon o rozmiarze 15x15 złożonych z liczb -1 oraz 1
   Dane walidujące stanowi identyczny zestaw emotikon, ale każda z nich została zaszumiona poprzez zmianę wartości kilku pikseli na przeciwną.
   Budowa sieci:



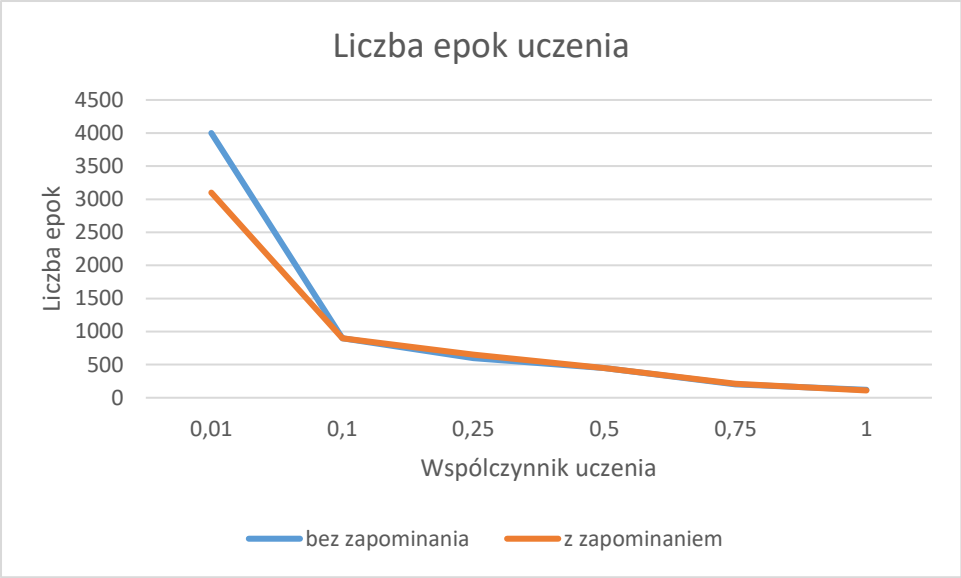
Sieć zbudowana podobnie jak na powyższym rysunku, tzn składa się z 4 neuronów w jednej warstwie, każdy z nich przyjmuje wszystkie piksele emotikony (225 wejść) i produkuje jedna daną wyjściową. Każdy z neuronów korzysta z sigmoidalnej bipolarnej funkcji aktywacji

Łączny błąd sieci neuronowej w danej epoce obliczany jest według wzoru:

$$E = E + \frac{1}{2}\sum (d_i - y_i)^2$$

Uczenie odbywa się dopóki błąd nie będzie niższy lub równy zadanej wartości

3. Zestawienie otrzymanych wyników
   Poniższy wykres przedstawia zestawienie ilości epok potrzebnych do nauczenia sieci (tak aby błąd sieci był niższy niż zadana wartość) w zależności od współczynnika uczenia. Zestawiono uczenie ze współczynnikiem zapominania i bez. Można zauważyć iż uczenie z zapominaniam ułatwia uczenie przy małych wartościach współczynnika uczenia. Dla większych wartości różnica staje się nieznaczna. W przypadku współczynnika uczenia mniejszego niż 0,01 sieć nie osiągała zadanej skuteczności w akceptowalnym czasie (uczenie przerywano ręcznie).



Kolejny wykres przedstawia zestawienie ilości epok potrzebnych do nauczenia sieci przy zadanym współczynniku zapominania. Jak można się było spodziewać siec uczy się szybciej kiedy współczynnik zapominania jest mniejszy. Kiedy współczynnik zwiększymy sieć zapomina to czego się nauczyła.

4. Podsumowanie

   Skuteczność procesu uczenia zależy od wartości współczynnik uczenia. Wraz z jego wzrostem rośnie skuteczność nauczania. Tłumaczy się to faktem iż wyższy współczynnik powoduje szybszą zmianę wag neuronu przez co szybciej osiągają one optymalną wartość, oczywiście pozornie, bo może istnieć wartość bardziej optymalna, ale jej znalezienie wymaga mniejszego współczynnika nauczania.

   Wpływ na szybkość uczenia ma także zastosowany algorytm. Sieć ucząca z zapominaniem jest skuteczniejsza niż bez zapominania, ale trzeba uważać przy doborze wartości współczynnika zapomania, ponieważ wartości zbliżone do 1 powodują, że siec zapomina to czego się do tej pory nauczyła i uczenie staje się nieefektywne.

5. Kod programu

Program główny:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Scenariusz__4
{
    class Program
    {
        static void Main(string[] args)
        {
            int inputSize = 15 * 15;
            double acceptableError = 0.001;
            double Error = 0;
            double neuronError = 0;
            int iloscNeuronow = 4;
            int iloscDanychUczacych = 4;
            double wspolczynnikUczenia = 0.001;
            double wspolczynnikZapominania = 0.01;
            int epochCounter = 0;
            Random random = new Random();
            int ExpectedOutput;
            double wynik;



            List<Data> learnData;
            double[] wyniki = new double[inputSize];

            for (int i = 0; i < inputSize; i++)
            {
                wyniki[i] = 0;
            }



            #region ReadData
            learnData = new List<Data>();
            int loopCounter = 0;
            String line;
            try
            {
                using (StreamReader sr = new StreamReader("inputData.txt"))
                {
                    while (sr.Peek() >= 0)//W tej pętli czytam kolejne linie znaków
                    {
                        List<int> tmp = new List<int>();


                        line = sr.ReadLine();

                        // for (int j = 0; j < line.Length; j++)//W tej pętli
czytam pojedyncze znaki
                        // {
                        string[] words = line.Split(' ');
                        foreach (string word in words)
                        {
```

```
                            tmp.Add(Int32.Parse(word));
                        }
                        //}
                        learnData.Add(new Data(tmp,
(int)(Data.Emoticon)loopCounter));
                        loopCounter++;
                    }
                }
            }
            catch (Exception e)
            {
                Console.WriteLine("The file could not be read:");
                Console.WriteLine(e.Message);
            }

            List<Data> testData = new List<Data>();
            try
            {
                using (StreamReader sr = new StreamReader("testData.txt"))
                {
                    while (sr.Peek() >= 0)//W tej pętli czytam kolejne linie znaków
                    {
                        List<int> tmp = new List<int>();


                        line = sr.ReadLine();

                        // for (int j = 0; j < line.Length; j++)//W tej pętli
czytam pojedyncze znaki
                        // {
                        string[] words = line.Split(' ');
                        foreach (string word in words)
                        {
                            tmp.Add(Int32.Parse(word));
                        }
                        //}
                        testData.Add(new Data(tmp,
(int)(Data.Emoticon)loopCounter));
                        loopCounter++;
                    }
                }
            }
            catch (Exception e)
            {
                Console.WriteLine("The file could not be read:");
                Console.WriteLine(e.Message);
            }
            #endregion


            //stwórz neurony (1. warstwa, 4. neurony)
            List<Neuron> neurons = new List<Neuron>();
            for (int i = 0; i < iloscNeuronow; i++)
            {
                neurons.Add(new Neuron(inputSize, wspolczynnikUczenia,
wspolczynnikZapominania, i + 1));
            }

            //uruchom siec
            double suma = 0;
            do
            {
```

```
                Error = 0;
                epochCounter++;
                for (int i = 0; i < iloscDanychUczacych; i++)
                {
                    for (int j = 0; j < iloscNeuronow; j++)
                    {
                        neuronError = 0;
                        wynik = neurons[j].Run(learnData[i].dataList);
                        if (j == learnData[i].symbol)
                        {
                            ExpectedOutput = 1;
                        }
                        else
                        {
                            ExpectedOutput = -1;
                        }


//neurons[j].TrainSmoothSupervised(learnData[i].dataList,ExpectedOutput);
                        neurons[j].TrainPlainSupervised(learnData[i].dataList,
ExpectedOutput);
                        Error += neuronError;
                    }
                }

                //walidacja neuronu:
                List<List<double>> neuronOutput = new List<List<double>>();
                Error = 0;
                for (int dataIndex = 0; dataIndex < testData.Count; dataIndex++)
                {
                    for (int j = 0; j < iloscNeuronow; j++)
                    {
                        neuronError = 0;
                        //neuron dostaje losowa dana uczaca
                        wynik = neurons[j].Run(learnData[dataIndex].dataList);
                        //zliczam blad jednostkowy
                        if (j == learnData[dataIndex].symbol)
                        {
                            ExpectedOutput = 1;
                        }
                        else
                        {
                            ExpectedOutput = -1;
                        }
                        neuronError = Math.Abs((wynik - ExpectedOutput) / wynik);
                        Console.WriteLine("błąd neuronu " + j + " = " + neuronError
+ "\tOczekiwana wartość = " + ExpectedOutput);
                        Error += neuronError;
                        //wneurons[i].PrintNeuron();
                    }
                }
                //zliczam blad calej sieci
                Error = Error * 100 / iloscNeuronow;
                Console.WriteLine("BŁĄD SIECI = " + Error);

            } while (Error >= acceptableError);


            Console.WriteLine("Uczenie zakończono w epoce " + epochCounter + " z
błędem = " + Error);
            //Wypisz wartości neuronów
            /*
```

```
                for(int i = 0; i < iloscNeuronow; i++)
                {
                    Console.WriteLine("Neuron " + i + " ");
                    Console.WriteLine("Wynik = " + neurons[i].actualAnswer);
                    Console.WriteLine("Oczekiwany wynik" + i);
                }
                */
                Console.ReadLine();
            }

            //napisz liczenie bledów

        }
    }
```

Neuron:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Scenariusz__4
{
    class Neuron
    {
        List<double> W;
        double N;
        double B;
        double beta = 1;
        double lastOutput;
        int expectedAnswer;
        public double actualAnswer;


        public Neuron(int w, double n, double b, int exp)
        {
            W = new List<double>();
            Random rand = new Random();
            for (int i = 0; i < w; i++)
            {
                W.Add(rand.NextDouble());

            }
            N = n;
            B = b;
            expectedAnswer = exp;
            lastOutput = 0;
        }

        public double CalculateInput(int i, int data)
        {
            return W[i] * data;
        }

        public double Run(List<int> data)
        {
            double output = 0;
            for (int i = 0; i < data.Count; i++)
            {
                output += CalculateInput(i, data[i]);
```

```
        }
        output = ActivationFunction(output);
        //output = Normalize(output);
        lastOutput = output;
        return output;
    }



    public double ActivationFunction(double sum)
    {
        //bipolarna funkcja sigmoidlana
        actualAnswer = ((2 / (1 + Math.Exp(-1 * sum * beta))) - 1);
        return actualAnswer;
    }

    public double Normalize(double x)
    {
        if (x >= 0)
            return 1;
        return -1;
    }

    public void TrainPlain(List<int> data)
    {
        //bez wspolczynnika zapominania
        for (int i = 0; i < W.Count; i++)
        {
            W[i] = W[i] + N * data[i] * lastOutput;
        }
    }

    public void TrainSmooth(List<int> data)
    {
        //bez wspolczynnika zapominania
        for (int i = 0; i < W.Count; i++)
        {
            W[i] = W[i] * (1 - B) + N * data[i] * lastOutput;
        }
    }

    public void TrainPlainSupervised(List<int> data, int desiredOutput)
    {
        //bez wspolczynnika zapominania
        for (int i = 0; i < W.Count; i++)
        {
            W[i] = W[i] + N * data[i] * desiredOutput;
        }
    }

    public void TrainSmoothSupervised(List<int> data, int desiredOutput)
    {
        //bez wspolczynnika zapominania
        for (int i = 0; i < W.Count; i++)
        {
            W[i] = W[i] * (1 - B) + N * data[i] * desiredOutput;
        }
    }


    public void PrintNeuron()
    {
```

```
            Console.WriteLine("#####################NEURON####################");
            Console.WriteLine("N = " + N);
            Console.WriteLine("B = " + B);
            for (int i = 0; i < W.Count; i++)
            {
                Console.WriteLine("Waga neuronu " + i + " = " + W[i]);
            }
        }
    }
}
```

Data:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Scenariusz__4
{
    class Data
    {
        public enum Emoticon { laugh, smile, love, cry}

        public List<int> dataList { get; set; }
        public int symbol;

        public Data(List<int> list, int symbol)
        {
            dataList = list;
            this.symbol = symbol;
        }

    }
}
```