Weronika Muszyńska, gr. 03

**Sprawozdanie z ćwiczenia nr 2**

**Temat ćwiczenia:**

Budowa i działanie sieci jednowarstwowej.

**Celem ćwiczenia** jest poznanie budowy i działania jednowarstwowych sieci neuronowych

oraz uczenie rozpoznawania wielkości liter.

**Syntetyczny opis budowy sieci jednowarstwowej**

Sieć neuronowa – sieć, w którym wyjścia każdego neuronu są połączone, poprzez wagi, z wejściami wszystkich neuronów, także z jego własnym wejściem.

Sieci jednokierunkowe to sieci neuronowe, w których nie występuje sprzężenie zwrotne, czyli pojedynczy wzorzec lub sygnał przechodzi przez każdy neuron dokładnie raz w swoim cyklu. Najprostszą siecią neuronową jest pojedynczy perceptron progowy, opracowany przez McCullocha i Pittsa w roku 1943**.**

Elementarna konfoguracja m neuronów połączonych z n wejściami wygląda następująco :



***Rys.1 Schemat połączeń jednowarstwowej sieci neuronowej.***

Wektory wejściowy i wyjściowy mają postać :

x= y =

Wagi wij łączą neuron I z wejściem j. Pobudzeniei-tego neuronu można zapisać jako :

neti = i=1,2,…,m

**Reguła uczenia delta**

Obowiązuje dla neuronów z ciągłymi funkcjami aktywacji i nadzorowanego trybu uczenia. Sygnał uczący, zwany sygnałem delta, jest zdefiniowany :

R= δ [di – f (, f’() oznacza pochodną funkcji aktywacji.

Wzór na korekcję wag ma postać :

δwi= c(di – yi ) f’( neti )x, c- przyjęta stała

****

***Rys.2 Uczenie według reguły delta***

1. W raz z każdym wektorem wejściowym X do neuronu podawany jest sygnał z. Neuron odpowiada na sygnał X sygnałem y = W \* X
2. J eśli neuron nie jest nauczony, sygnał ten jest inny niż wymagany (y≠z). Wewnątrz neuronu istnieje blok oceniający wielkość błędu, δ = z - y
3. Blok ten składa się inwertora oraz sumatora. Na podstawie sygnału błedu oraz wektora wejściowego X możliwe jest takie skorygowanie wektora wag W, by neuron lepiej realizował zadaną funkcję y= f(X).
4. Nowy wektor wag W’ obliczany jest ze wzoru:

W’ = W + ηδX , gdzie η jest współczynnikiem liczbowym, decydującym o szybkości nauki.

**Algorytm wstecznej propagacji błędów :**

metoda uczenia sieci wielowarstwowej, w której błąd ostatniej warstwy jest przesyłany wstecz i wykorzystywany do zmiany wartości wag w poprzednich warstwach.

**Ogólny schemat** procesu trenowania sieci wygląda następująco:

1. Ustalamy topologię sieci, tzn. liczbę warstw, liczbę neuronów w warstwach.
2. Inicjujemy wagi losowo (na małe wartości).
3. Dla danego wektora uczącego obliczamy odpowiedź sieci (warstwa po warstwie).
4. Każdy neuron wyjściowy oblicza swój błąd, oparty na różnicy pomiędzy obliczoną odpowiedzią y oraz poprawną odpowiedzią t.
5. Błędy propagowane są do wcześniejszych warstw.
6. Każdy neuron (również w warstwach ukrytych) modyfikuje wagi na podstawie wartości błędu i wielkosci przetwarzanych w tym kroku sygnałów.
7. Powtarzamy od punktu 3. dla kolejnych wektorów uczących. Gdy wszystkie wektory zostaną użyte, losowo zmieniamy ich kolejność i zaczynamy wykorzystywać powtórnie.
8. Zatrzymujemy się, gdy średni błąd na danych treningowych przestanie maleć. Możemy też co jakiś czas testować sieć na specjalnej puli nieużywanych do treningu próbek testowych i kończyć trenowanie, gdy błąd przestanie maleć.

Aby znależć taki zestaw wag, dla którego błąd sieci jest jak najmniejszy, możemy zapisać ten błąd jako funkcję od wartości wag. Oznaczmy przez:

f: R -> R – funkcję aktywacji w neuronie

w1 ,..., wK – wagi połączeń wchodzących

z1 ,..., zK – sygnały napływające do neuronu z poprzedniej warstwy.

Zwykle błąd liczony jest jako kwadrat odchylenia: d = 1/2 (y-t)2, co możemy rozpisać jako:

**d(w1, ... , wk) = (f(w1z1+... + wkzk)-t)2**

Błąd sieci

W celu ustalenia, o ile powinna zmienić się waga neuronu, powiniśmy "rozłożyć" otrzymany błąd całkowity na połączenia wprowadzające sygnały do danego neuronu. Składową błędu dla każdego j-tego połączenia określamy jako pochodną cząstkową błędu względem j-tej wagi.

Zestaw danych uczących i testujących

Wygenerowanno dane uczące i testujące, zawierające 10 dużych i 10

małych liter wybranych z alfabetu w postaci dwuwymiarowej tablicy 5x7

pikseli dla jednej litery.

Przykład : litera W

**1 0 0 0 1  
1 0 0 0 1  
1 0 0 0 1  
1 0 0 0 1  
1 0 1 0 1  
1 0 1 0 1  
0 1 0 1 0**

**Zestawienie wyników**

Otrzymane wyniki zebrano w postaci tabel i wykresów i przedstawiono poniżej :

Współczynnik uczenia = 0.15

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ilość danych uczących** | **5** | **10** | **15** | **20** |
| **ilość iteracji** | **1943** | **7534** | **9999** | **9999** |
| **Błąd** | **0,02** | **0,04** | **0,06** | **0,03** |

Współczynnik uczenia = 0.6

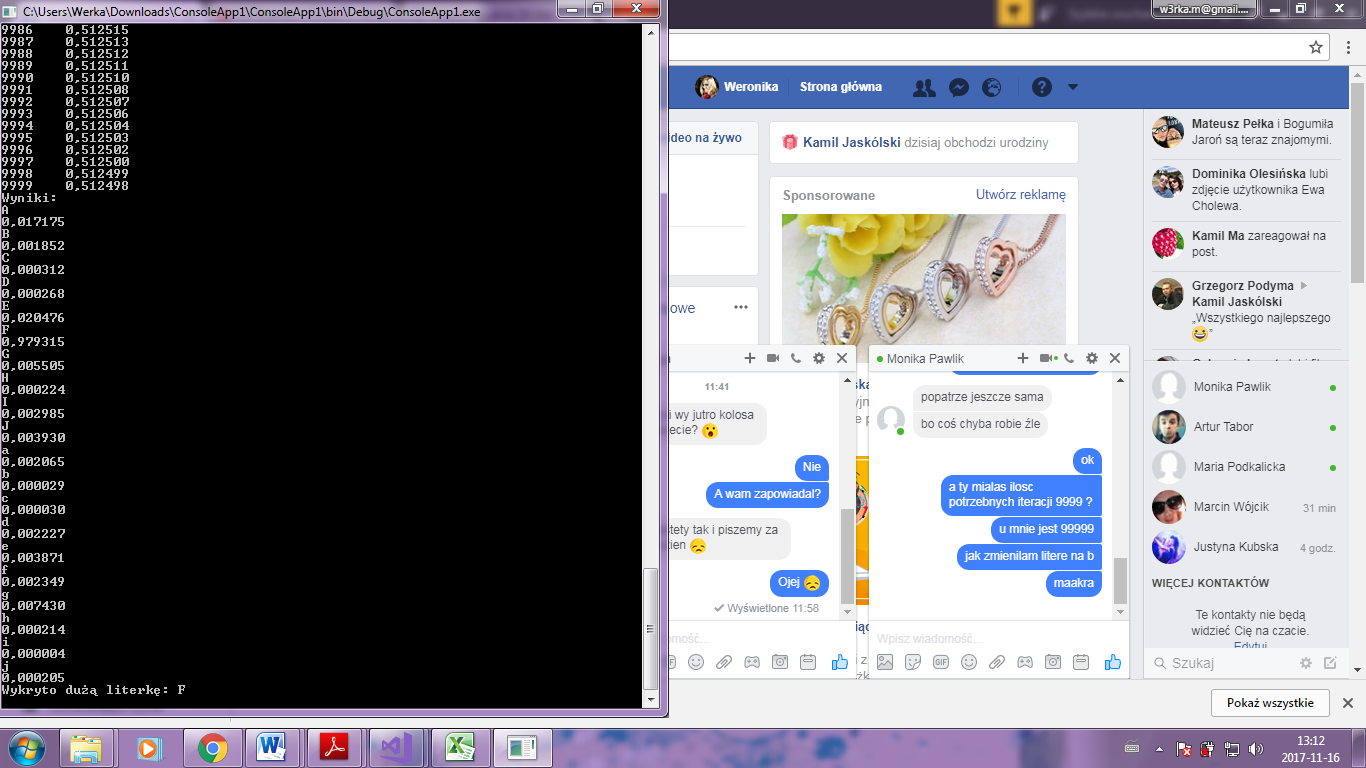
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ilość danych uczących** | **5** | **10** | **15** | **20** |
| **ilość iteracji** | **314** | **1241** | **2100** | **9999** |
| **Błąd** | **0,01** | **0,03** | **0,03** | **0,02** |

Współczynnik uczenia = 0.8

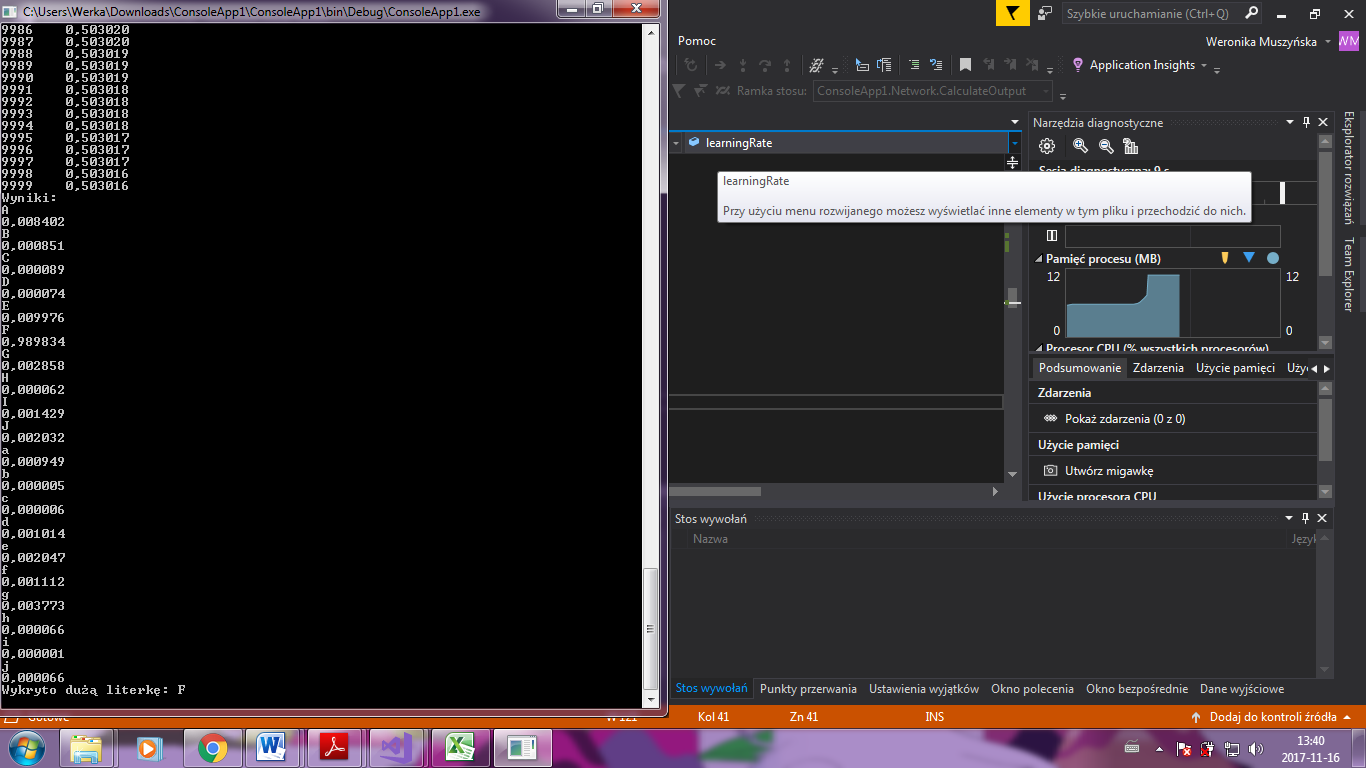
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ilość danych uczących** | **5** | **10** | **15** | **20** |
| **ilość iteracji** | **211** | **823** | **1382** | **9999** |
| **Błąd** | **0,01** | **0,02** | **0,03** | **0,01** |

***Wykres 1, wpływ ilości danych uczących oraz współczynnika uczenia na ilość iteracji***

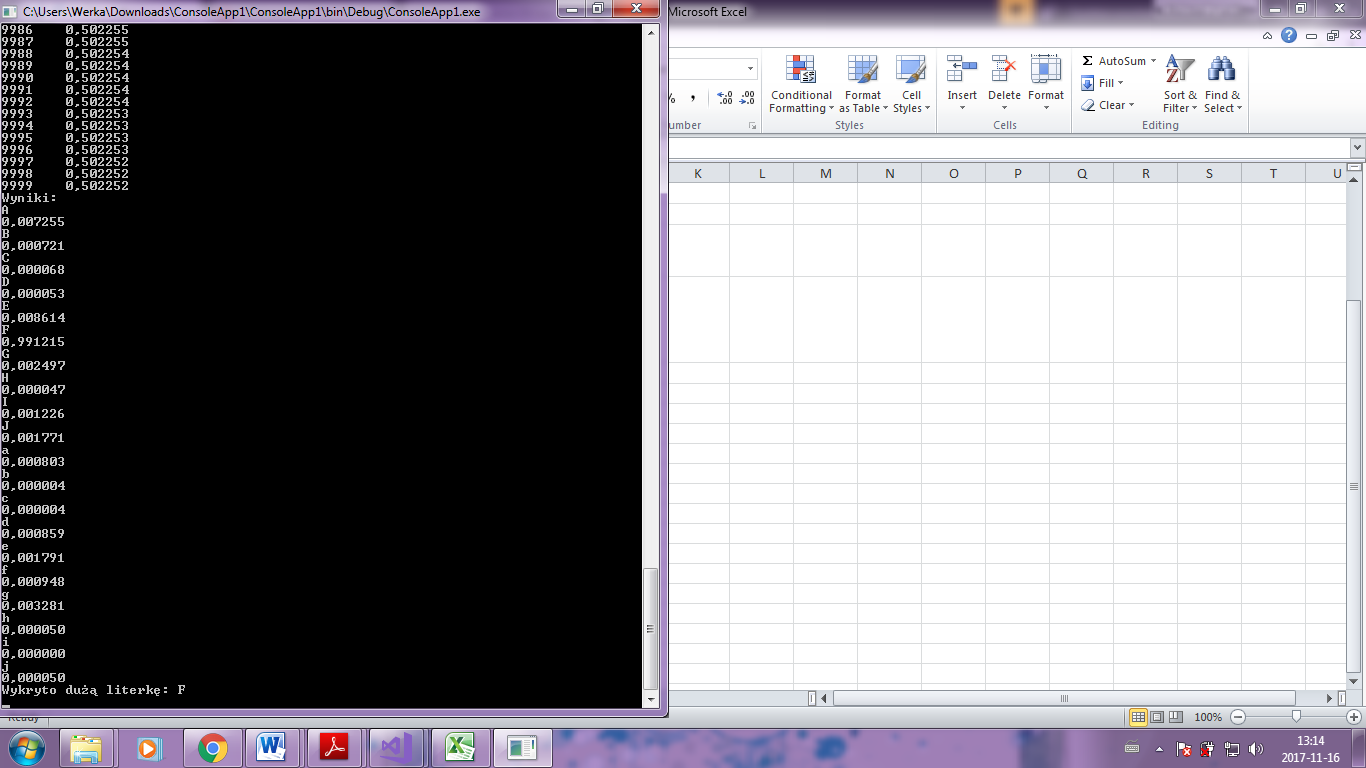
**Przykładowe screeny pokazujące działanie programu oraz odczyt danych :**



***Rys.3 współczynnik uczenia =0.15, ilość danych uczących 20***



***Rys.4 współczynnik uczenia =0.6 , ilość danych uczących 20***



***Rys. 5 współczynnik uczenia =0.8, ilość danych uczących 20***

**Analiza wyników:**

Wybranie małego współczynnika uczenia doprowadza do bardzo powolnego procesu uczenia. Trzeba wykonać dużo kroków, ponieważ były one poprawiane o małe wartości.

Nie można tez wybrać zbyt dużego współczynnika uczenia, gdyż może to prowadzić do bardzo gwałtownych zmiany parametrów sieci.

Im mniejszy współczynnik uczenia, tym większa ilość potrzebnych iteracji do nauki sieci. Najmniejszy błąd podczas testowania zgadywania liter okazał się dla współczynnika 0,6. Okazało się, że największe błędy są dla największej liczby danych do nauki.

Ilość potrzebnych iteracji jest najmniejsza dla największego współczynnika uczenia ( w tym przypadku 0.8) i wzrasta wraz ze wzrostem liczby danych uczących. Jednakże przy 20 danych uczących ilość iteracji potrzebnych do nauki sieci jest taka sama i wynosi 9999.

Dla współczynników uczenia 0.6 oraz 0.8 ilość iteracji w zależności od ilości danych uczących nie różni się zbyt wiele (dla 5, 10 i 15 danych jest to odowiednio : 103, 418, 718). Porównując współczynniki 0.15 oraz 0.8 ta różnica staje się dużo wyższa (1732, 6711, 8617), podawana już w tysiącach.

**Wnioski :**

Po przeprowadzonym eksperymencie można zauważyć, że ważny jest współczynnik uczenia sieci, który powoduje że sieć jest stabilniejsza i popełnia mniej błędów, oraz błąd docelowy do którego zmierza sieć w procesie uczenia. Istotne jest również sposób przygotowania zbioru uczącego, ponieważ nie może zawierać elementów o podobnym wyglądzie a różnych odpowiedziach w procesie uczenia, gdyż sieć jest wówczas źle uczona co powoduje zwiększenie ilości popełnianych błędów. Sieć neuronowa udziela mniej błędnych odpowiedzi kiedy zbiór uczący składa się z uporządkowanych symboli liter niż z przypadkowo ułożonych elementów.

**Źródła:**

1. **S. Osowski. Sieci neuronowe w ujęciu algorytmicznym. WNT, Warszawa 1997.**
2. **J. Żurada, M. Barski, W. Jędruch. Sztuczne sieci neuronowe. PWN, Warszawa 1996.**
3. **R. Tadeusiewicz, Sieci neuronowe. Warszawa : Akademicka Oficyna Wydaw. RM, 1993.**

**Listing kodu :**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

class Program

{

private readonly Network \_network = new Network();

private readonly string[] \_literki = {"A", "B","C","D","E","F","G","H","I","J", "a", "b", "c", "d", "e", "f", "g", "h", "i", "j" };

private static int imageSize = 5;

static void Main(string[] args)

{

var p = new Program();

p.\_network.maximumIteration = int.Parse(10000.ToString());

p.Test();

p.Check();

Console.ReadKey();

}

private void Test()

{

int length = \_literki.Length;

\_network.Initialize(imageSize \* imageSize, length);

double[][] inputs = new double[length][];

double[][] outputs = new double[length][];

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

{

outputs[i] = new double[length];

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

{

outputs[i][j] = i == j ? 1.0 : 0.0;

}

\_network.outputLayer[i].Value = \_literki[i];

}

inputs[0] = new double[25] {1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1};

inputs[1] = new double[25] {1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0};

inputs[2] = new double[25] {1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1};

inputs[3] = new double[25] {1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0};

inputs[4] = new double[25] {1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1};

inputs[5] = new double[25] {1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0};

inputs[6] = new double[25] {1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0};

inputs[7] = new double[25] {1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1};

inputs[8] = new double[25] {1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0};

inputs[9] = new double[25] {1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0};

inputs[10] = new double[25] {1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0};

inputs[11] = new double[25] {1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0};

inputs[12] = new double[25] {0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0};

inputs[13] = new double[25] {0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0};

inputs[14] = new double[25] {0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0};

inputs[15] = new double[25] {0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0};

inputs[16] = new double[25] {1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0};

inputs[17] = new double[25] {0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0};

inputs[18] = new double[25] {0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0};

inputs[19] = new double[25] {0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0};

\_network.TrainNetwork(inputs, outputs);

Console.WriteLine("Errors:");

for (int i = 0; i < \_network.currentIteration; ++i)

{

Console.WriteLine( i + "\t"+ \_network.errors[i].ToString("#0.000000"));

}

}

private void Check()

{

double[] sample = new double[imageSize \* imageSize];

sample = new double[25] { 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0 };

\_network.Recognize(sample);

Console.WriteLine("Wyniki:");

var wyniki = new double[\_literki.Length];

for (int i = 0; i < \_network.outputLayer.Length; ++i)

{

wyniki[i] = \_network.outputLayer[i].Output;

Console.WriteLine(\_network.outputLayer[i].Value);

Console.WriteLine(\_network.outputLayer[i].Output.ToString("#0.000000"));

}

int w = wyniki.ToList().IndexOf(wyniki.Max());

if(w > \_literki.Length/2)

Console.WriteLine("Wykryto małą literkę: " + \_network.outputLayer[w].Value);

else

{

Console.WriteLine("Wykryto dużą literkę: " + \_network.outputLayer[w].Value);

}

}

}

class Network

{

public struct InputLayer

{

public double Value;

public double[] Weights;

}

public struct OutputLayer

{

public double InputSum;

public double Output;

public double Error;

public double Target;

public string Value;

}

public double learningRate = 0.6;

public int ImageSize = 0;

public int InputNum = 0;

public int OutputNum = 0;

public InputLayer[] inputLayer = null;

public OutputLayer[] outputLayer = null;

public double[] errors = null;

public int currentIteration = 0;

public int maximumIteration = 1000;

public void Initialize(int inputSize, int outputSize)

{

InputNum = inputSize;

OutputNum = outputSize;

inputLayer = new Network.InputLayer[inputSize];

outputLayer = new Network.OutputLayer[outputSize];

Random random = new Random();

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

{

inputLayer[i].Weights = new double[OutputNum];

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

{

inputLayer[i].Weights[j] = random.Next(1, 3) / 100.0;

}

}

}

public bool TrainNetwork(double[][] inputs, double[][] outputs)

{

double currentError = 0.0, maximumError = 0.01;

currentIteration = 0;

errors = new double[maximumIteration];

do

{

currentError = 0;

for (int i = 0; i < inputs.Length; ++i)

{

CalculateOutput(inputs[i], outputLayer[i].Value);

BackPropagation();

currentError += GetError();

}

errors[currentIteration] = currentError;

++currentIteration;

}

while (currentError > maximumError && currentIteration < maximumIteration);

if (currentIteration <= maximumIteration)

{

return true;

}

return false;

}

private void CalculateOutput(double[] pattern, string output)

{

for (int i = 0; i < pattern.Length; i++)

{

inputLayer[i].Value = pattern[i];

}

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

{

double total = 0.0;

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

{

total += inputLayer[j].Value \* inputLayer[j].Weights[i];

}

outputLayer[i].InputSum = total;

outputLayer[i].Output = Activation(total);

outputLayer[i].Target = outputLayer[i].Value.CompareTo(output) == 0 ? 1.0 : 0.0;

outputLayer[i].Error = (outputLayer[i].Target - outputLayer[i].Output) \* (outputLayer[i].Output) \* (1 - outputLayer[i].Output);

}

}

private void BackPropagation()

{

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

{

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

{

inputLayer[i].Weights[j] += learningRate \* (outputLayer[j].Error) \* inputLayer[i].Value;

}

}

}

private double GetError()

{

double total = 0.0;

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

{

total += Math.Pow((outputLayer[j].Target - outputLayer[j].Output), 2.0) / 2.0;

}

return total;

}

private double Activation(double x)

{

return (1.0 / (1.0 + Math.Exp(-x)));

}

public void Recognize(double[] input)

{

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

{

inputLayer[i].Value = input[i];

}

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

{

double total = 0.0;

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

{

total += inputLayer[j].Value \* inputLayer[j].Weights[i];

}

outputLayer[i].InputSum = total;

outputLayer[i].Output = Activation(total);

}

}

}

}