**Sprawozdanie z zadania nr 2.**

**Temat:** Budowa i działanie sieci jednowarstwowej.

**Cel:** celem ćwiczenia było poznanie budowy i działania jednowarstwowych sieci neuronowych oraz uczenie rozpoznawania wielkości liter.

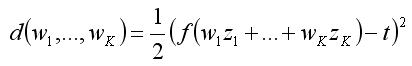
**Wstęp do zagadnienia:**

Rozpoznawanie wzorców liter jest stosunkowo poważnym zadaniem do zrealizowania z wykorzystaniem sztucznej sieci neuronowej. Litery są utworzone na matrycy 5 x 7 czyli na 35 polach. I tyle wejść ma sieć. Ponieważ liter jest 26, prawidłowe rozpoznanie litery będzie polegało na uaktywnieniu jednego z 26 wyjść sieci, lub więcej w przypadku rozpoznawania w prezentowanym wektorze cech kilku liter. Zobaczymy jak znaleźć ilość neuronów w warstwie ukrytej oraz za pomocą jakich mechanizmów można sieć dostroić.

Delta rule:

Adaline:

Algorytm wstecznej propagacji błedu:



**Opis wykonanych kroków w toku realizacji zadania**:

1. Zaimplementowano prostą sieć według instrukcji podanej na ćwiczeniach.
2. Wygenerowano dane uczące :
3. dla kodu java w postaci matryc :

0,0,0,0,0,

0,0,0,0,0,

0,0,0,0,0, co w kodzie było równoznaczne z zapisem

1,0,0,0,1, => 0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0, 1,0,0,0,1, 1,0,0,0,1, 1,0,0,0,1, 0,1,0,1,0,

1,0,1,0,1,

1,0,1,0,1,

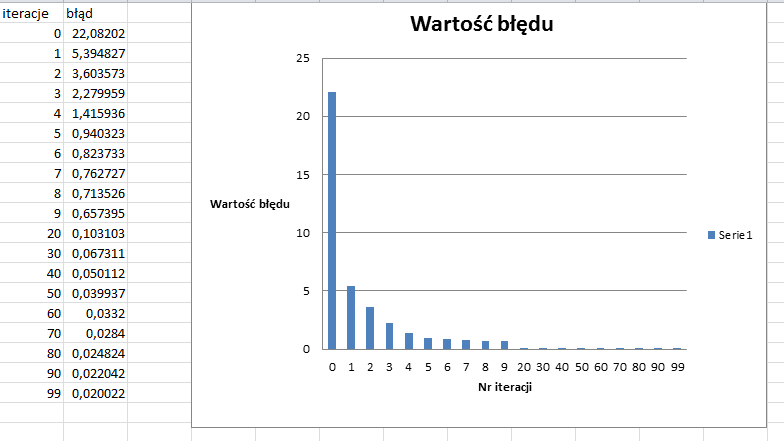
0,1,0,1,0,

b) dla kodu dla cs : matryce- specjalnie przygotowane obrazy w formacie bmp.

1. Wykonano etap uczenia perceptronu dla różnej ilości danych uczących oraz współczynników uczenia „n” a także rożnej ilości iteracji by sprawdzić szybkość uczenia oraz wielkość błędu.
2. Przeprowadzono testy działania za pomocą losowo wybieranego przypadku.

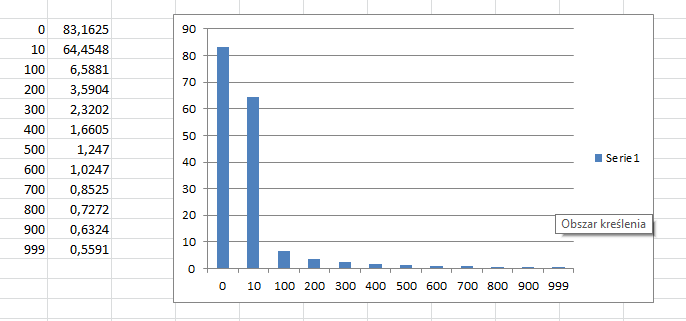
**Zestawienie rezultatów:**

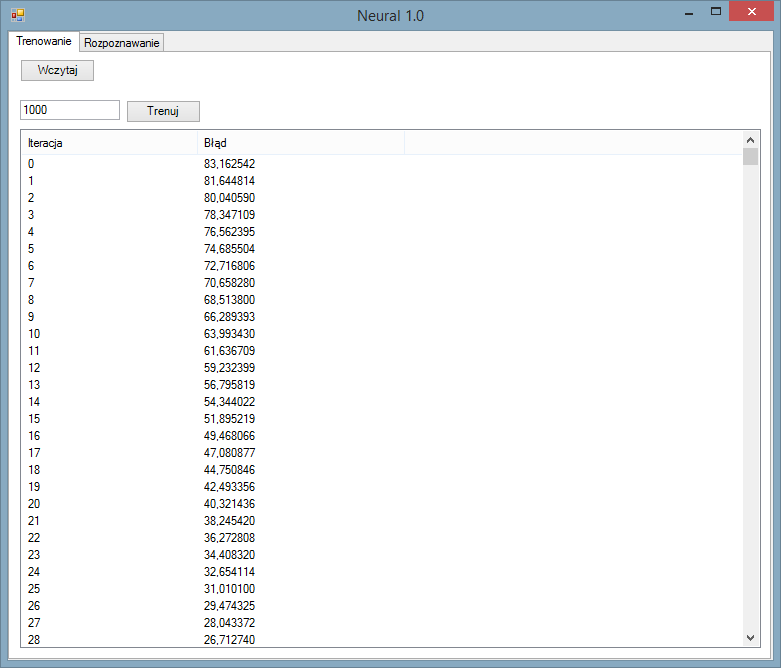
Większość wyników będzie odnosić się do kodu w cs, gdyż działa on stabilnie, sieć daje powtarzalne wyniki i udziela trafnej odpowiedzi .

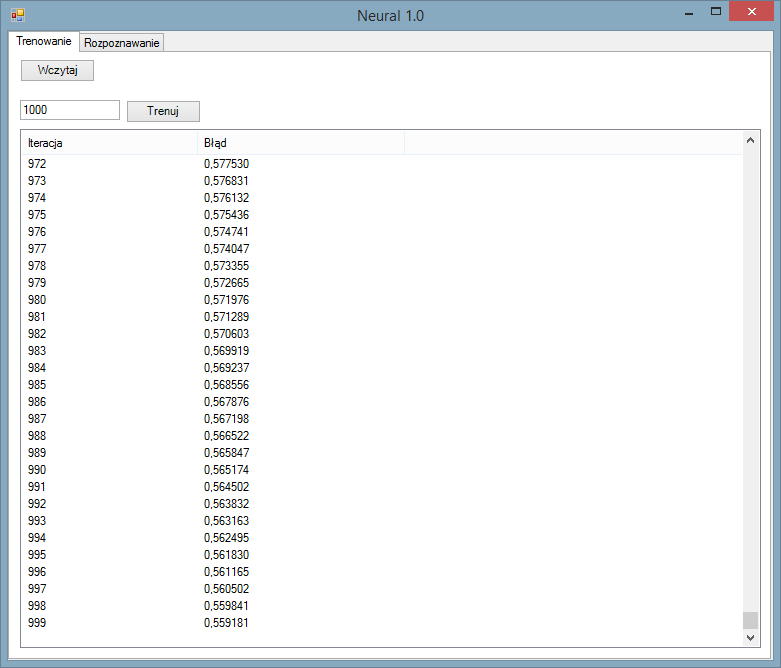
Wynik dla współczynnika uczenia =0,2.

**Spostrzeżenie:**  już przy 50 iteracjach wartość błędu jest na tyle mała, że nie musimy wykonywać ich więcej w procesie uczenia , gdyż proces ten trwa znacznie dłużej a uzyskana korzyść jest praktycznie marginalna- odpowiedź i tak będzie poprawna w obu przypadkach.

Wynik dla współczynnika uczenia się=0,001



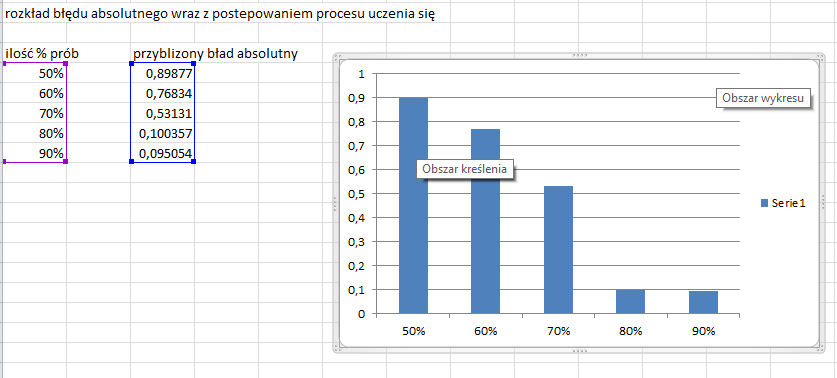
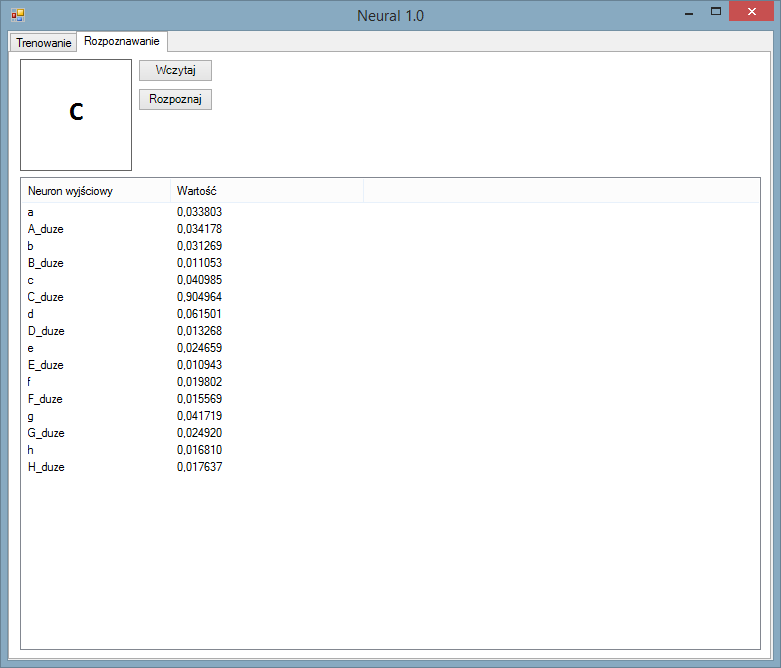




**Spostrzeżenie:**  w przypadku gdy celowo zmniejszono współczynnik uczenia się drastycznie wzrósł błąd . Potrzeba było aż 1000 iteracji by błąd był zbliżony do 10 iteracji z poprzedniego przypadku.

Przykładowy zrzut z działającego programu:

Po wytrenowaniu sieci na zadanym zbiorze liter a(duże i małe) : h( duże i małe), wybrano przykładową matryce: C. Następnie przetestowano odpowiedź sieci i jak widać poprawnie rozpoznano z wartością = 0,904946, gdzie oczekiwaną możemy założyć 1.  
Więc błąd absolutny wyniósł =0,095054.



Dla kodu w java:

Na podstawie dostarczanych matryc udało się tylko w części przypadków uzyskiwać poprawne odpowiedzi ( z reguły w zakresie 60-80%), gdyż w przypadku zadania do wytrenowania :12 matryc wzorcowych dla małych liter i 12 dla dużych liter , odpowiedź sieci z reguły mówiła o dostarczeniu 10 liter małych i 14 dużych- był to najlepszy rezultat jaki udało się osiągnąć.

Przykładowy wydruk:

50.17906495438872

Mala

69.44939827203414

Mala

74.86238584029243

Mala

66.65176374407329

Mala

84.35531355617681

Mala

97.83028714591259

duza

66.1589608672837

Mala

72.61655680133812

Mala

50.17906495438872

Mala

139.8626069215344

duza

90.22922924781932

duza

126.90526269840221

Duza

Wykryte male litery 10

Wykryte duze litery 14

Mogło to być spowodowane mniejszą rozbudową sieci , zbyt małą ilością danych uczących , podobieństwem liter – ich reprezentacją w matrycach, gdzie od pewnego poziomu siec interpretowała wystąpienie” 1” jako fragment dużej litery i po przemnożeniu przez wagi itd zwiększało błąd.

**Listing najważniejszych fragmentów kodu:**

**Dla cs:**public void Initialize(int inputSize, int outputSize)

{

InputNum = inputSize;

OutputNum = outputSize;

inputLayer = new Network.InputLayer[inputSize];

outputLayer = new Network.OutputLayer[outputSize];

// Zainicjuj wagi losowymi wartościami z zakresu 0.01 - 0.02.

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;

// Utwórz tablicę do przechowywania wartości kolejnych błędów.

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);

// Jeżeli maksymalny błąd został osiągnięty w mniejszej liczbie iteracji,

// to nauka sieci zakończyła się pomyślnie.

if (currentIteration <= maximumIteration)

{

return true;

}

return false;

}

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

{

// Przypisz wejście do warstwy wejściowej.

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

{

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

}

// Oblicz wejście, wyjście, wartość oczekiwaną oraz błąd pierwszej warstwy.

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);

}

}

public void BackPropagation()

{

// Popraw wagi warstwy wejściowej.

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;

}

}

}

public 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; }

**Dla java:**

**//utworzenie warstw**

public class Layer {  
 Neuron neurons[];  
 int numberOfNeurons;  
  
 public Layer(int numberOfNeurons){  
 this.numberOfNeurons = numberOfNeurons;  
 this.neurons= new Neuron[numberOfNeurons];  
  
 for(int i=0; i<numberOfNeurons ; i++){  
 this.neurons[i] = new Neuron();  
 }  
 }  
}

List<Double> list = new ArrayList<Double>();  
 //kalibracja  
 for (int i = 10; i < 20; i++) {  
 for (int j = 0; j < 10; j++) {  
 l1.neurons[j].getAnswer(i);  
 n1.input[j] = l1.neurons[j].answer;  
 }  
 n1.getAnswer2();  
 list.add(n1.suma);  
 }  
 Collections.*sort*(list);  
 System.*out*.println(list.get(0));  
 double classifier;  
 classifier = list.get(0);  
  
 //test  
 int dc=0;  
 int mc=0;  
 for (int t = 0; t < 24; t++) {  
 for (int i = 0; i < 10; i++) {  
 l1.neurons[i].getAnswer(t);  
 }  
 for (int i = 0; i < 10; i++) {  
 n1.input[i] = l1.neurons[i].answer;  
 }  
 n1.getAnswer2();  
 System.*out*.println(n1.answer);  
 if(n1.answer<classifier){  
 System.*out*.println("Mala");  
 mc++;  
 }  
  
  
 else{  
 System.*out*.println("duza");  
 dc++;  
 }  
  
 }  
 System.*out*.println("Wykryte male litery "+mc);  
 System.*out*.println("Wykryte duze litery "+dc);  
 }

//obliczanie odpowiedzi z warstw + metoda hebba

public void getAnswer(int j){  
 this.getSuma(j);  
 if(this.suma<0)  
 this.answer=0;  
 else  
 this.answer=this.suma;  
 }  
 public void getAnswer2(){  
 this.getSuma2();  
 if(this.suma<0)  
 this.answer=0;  
 else  
 this.answer=this.suma;  
 }  
 public void hebb(int j){  
 this.getAnswer(j);  
 for(int i=0; i<35; i++){  
 this.weight[i] += 0.000000025 \* Data.*records*[j][i] \* this.answer;  
 }  
 this.getAnswer(j);  
 }  
 public void hebb2(){  
 this.getAnswer2();  
 for(int i=0; i<10; i++){  
 this.weight[i] += 0.000000025 \* this.input[i] \* this.answer;  
 }  
 }  
}

**Wnioski:**

**-**im dokładniejszy zbiór uczący (bez powtórzeń) dostarczy się sieci tym błąd na wyjściu ( oczekiwana – uzyskana wartość) będzie mniejszy.

-nie zawsze odpowiedzi sieci są jednoznaczne ze względu na sposób reprezentacji liter(macierze 01 vs bmp)

- metoda obliczania algorytmem wstecznej propagacji dawała lepsze rezultaty niż metoda adaline/czy też hebb rule.

-współczynnik uczenia się ma wpływ na czas trwania procesu uczenia się a także na jego poprawność.

-dzięki właściwemu dobraniu współczynnika uczenia się i ilości iteracji jesteśmy w stanie „zoptymalizować ‘’ cały proces , uzyskując przy tym niewielki błąd i znaczna część udzielonych odpowiedzi będzie poprawna ( określona poprawnie z dużym prawdopodobieństwem).