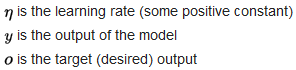
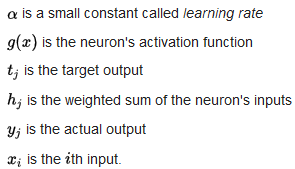
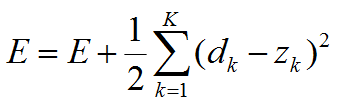
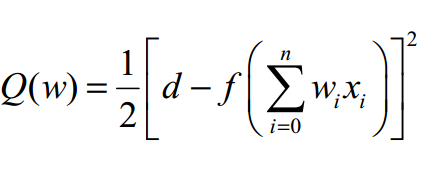
**Mateusz Bożek  
Scenariusz 2**  
**Temat :** Budowa i działanie sieci jednowarstwowej  
  
**Cel ćwiczenia :** Celem ćwiczenia jest zapoznanie się z budową i działaniem jednowarstwowej sieci neuronowej, której zadaniem jest nauczenie się rozpoznawania wielkości liter.  
  
**1.Syntetyczny opis budowy i wykorzystania sieci i algorytmów uczenia.**  
SSN - sztuczna sieć neuronowa, jest to sieć realizująca obliczenia lub przetwarzająca sygnały poprzez rzędy elementów zwanych sztucznymi neuronami.  
  
Do wykonania zadania- uczenia sieci jednowarstwowej - użyto algorytmu uczenia, który polega na podaniu p par uczących  **{ ( y1,d1) , (y2,d2) , (y3,d3) , ... ( yp,dp)}**  gdzie :  
yi - rozmiar J x 1;  
di - rozmiar K x 1;  
yiJ = -1, i = 1,2,3,...,p - wartości sygnału progowego;  
l - numer kroku cyklu uczenia;  
  
Poniżej przedstawiony został cykl uczenia warstwy w kolejnych krokach :  
Krok 1 *: Wybór h>0 oraz Emax >0.*  
Krok 2 : *Wylosowanie losowych wag z przedziału < 0 , 1 > i zapisanie ich do macierzy W o rozmiarze K x J.*Krok 3 : Wyzerowanie wartości błędu E =0 oraz ustawienie wartości początkowej l=1.  
Krok 4 : Podanie obrazu na wejście i obliczenie sygnału wyjściowego:  
 *y = yl*, *d=dl, zk = j(wkT y), k=1,2, ..., K* gdzie *wkT* jest *k*-tym wierszem macierzy *W*.  
Krok 5 : Aktualizacja nowych wag  
  
Adaline :  
  
gdzie :  
  
  
DeltaRule:  
  
gdzie :  
  
  
Krok 6 : Następnie obliczany jest błąd ze wzoru :

  
  
Krok 7 : W przypadku gdy l < p to inkrementujemy l ( l = l +1 ) i przechodzimy ponownie do kroku 4  
Krok 8 : Gdy E < Emax, proces nauki zostaje zakończony, w przeciwnym przypadku przechodzimy ponownie do kroku 3.  
  
W powyższym zagadnieniu użyto **modelu Adaline** - model podobny do budowy perceptronu. W odróżnieniu od niego, model Adaline różni się algorytmem uczenia, ponieważ nie uwzględniana jest funkcja aktywacji przy porównywaniu sygnału wyjściowego z sygnałem wzorcowym.  
Błąd średniokwadratowy w przypadku modelu Adaline obliczany jest ze wzoru :

W celu aktualizacji wag korzystamy ze wzoru :

**Sigmoidalna sieć neuronowa** - sieć podobnie skonstruowana jak w przypadku modelu Adaline. Różnica między modelami jest taka, że w przypadku modelu sigmoidalnego używana jest funkcja sigmoidalna unipolarna (która jest wykorzystywana w projekcie ) lub bipolarna. Błąd wyliczany jest ze wzoru :  
Aktualizacja wag przedstawiana jest wzorem :

**2.Zestawienie wyników:**W projekcie każda litera zapisana jest za pomocą wartości 0 oraz 1 w tablicy 35 elementowej .  
Występuje 10 liter dużych oraz 10 liter małych.  
Przeprowadzono 4 próby dla współczynników : 0,05 ; 0,1 ; 0,2 ; 0,5.

*Współczynnik : 0,05*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Adaline |  |  | Sigmoidalna | |
| nr próby | liczba iteracji | liczba błędów | nr próby | liczba iteracji | liczba błędów |
| 1 | 163 | 3 | 1 | 6903 | 0 |
| 2 | 139 | 6 | 2 | 6790 | 0 |
| 3 | 124 | 4 | 3 | 7084 | 0 |
| 4 | 143 | 4 | 4 | 6945 | 0 |
| 5 | 156 | 5 | 5 | 7012 | 0 |

*Współczynnik : 0,1*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Adaline |  |  | Sigmoidalna |  |
| nr próby | liczba iteracji | liczba błędów | nr próby | liczba iteracji | liczba błędów |
| 1 | 51 | 1 | 1 | 3598 | 0 |
| 2 | 93 | 4 | 2 | 3478 | 0 |
| 3 | 79 | 3 | 3 | 3571 | 0 |
| 4 | 89 | 2 | 4 | 3590 | 1 |
| 5 | 75 | 4 | 5 | 3468 | 0 |

*Współczynnik : 0,2*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Adaline |  |  | Sigmoidalna |  |
| nr próby | liczba iteracji | liczba błędów | nr próby | liczba iteracji | liczba błędów |
| 1 | 331 | 20 | 1 | 1819 | 1 |
| 2 | 331 | 20 | 2 | 1693 | 0 |
| 3 | 332 | 20 | 3 | 1749 | 0 |
| 4 | 331 | 20 | 4 | 1789 | 0 |
| 5 | 331 | 20 | 5 | 1754 | 0 |

*Współczynnik : 0,5*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Adaline |  |  | Sigmoidalna | |
| nr próby | liczba iteracji | liczba błędów | nr próby | liczba iteracji | liczba błędów |
| 1 | 45 | 20 | 1 | 692 | 0 |
| 2 | 45 | 20 | 2 | 722 | 0 |
| 3 | 45 | 20 | 3 | 702 | 1 |
| 4 | 45 | 20 | 4 | 709 | 0 |
| 5 | 45 | 20 | 5 | 716 | 0 |

**3.Analiza:**  
Powyższe wykresu przedstawiają wyniki nauczania sieci jednowarstwowej. Z analizy wynika, że przy małym współczynniku uczenia potrzeba większej liczby epok do nauki. Konfrontując regule delta z modelem adaline widzimy, że przy regule delta sieć prawie w ogóle nie popełnia błędów przy nauce oraz potrzebuje zdecydowanie więcej epok do nauczenia się. Z kolej w przypadku modelu adaline, powyżej współczynnika 0,1 sieć uczyła się błędnie.  
 **4.Wnioski:**  
  
Z przeprowadzonych prób wynika, że im większy współczynnik nauczania tym mniejsza liczba epok potrzebna jest do nauczenia. Liczba epok zależy natomiast od początkowo wylosowanych wag. Porównując oba modele widzimy, że model signoidalny uczy się zdecydowanie lepiej niż model analine jednak potrzebuje na to więcej epok, natomiast efekt uczenia modelu adaline jest słaby.   
  
**5.Listing programu:**

**5.Kod programu :**  
public class Main {  
 public static void main(String[] args) {  
 System.*out*.println("ANALINE");  
 Adaline adaline = new Adaline();  
 adaline.uczenie();  
 adaline.testAdaline();  
  
 System.*out*.println("DELTA SIGNOIDAL");  
 DeltaSigmoidal deltasigmoidal = new DeltaSigmoidal();  
 deltasigmoidal.uczenie();  
 deltasigmoidal.testDelta();  
  
 }  
  
}

Adaline  
import java.util.Random;  
  
public class Adaline {  
  
 public double[] tablica\_wag; //tablica wag  
 public int liczba\_liter; //liczba liter  
 public double blad\_globalny; //blad globalny  
 public int liczba\_wag; //liczba wag  
 public double wspolczynnik\_nauki; //wspolczynnik uczenia  
 public double delta;  
  
 public static char[] *litery* = {'A','B','C','E','G','I','K','M','O','R','T','a','b','c','d','e','f','g','h','l','r'};  
 public static int[] *tablica\_oczekiwanych\_wynikow* = {1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0}; //tablica oczekiwanych wynikow  
 public static int[][] *tablica\_liter\_bit* = { //tablica danych uczacych  
 {0, 1, 1, 1, 0, 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, 1, 0, 0, 0, 1}, //A  
 {1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0}, //B  
 {0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0}, //C  
 {1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0}, //D  
 {1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1}, //E  
 {1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0}, //F  
 {0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0}, //G  
 {1, 0, 0, 0, 1, 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, 1, 0, 0, 0, 1}, //H  
 {0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0}, //I  
 {1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0}, //J  
  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1}, //a  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1}, //b  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1}, //c  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1}, //d  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1}, //e  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0}, //f  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1}, //g  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1}, //h  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1}, //l  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0}, //r  
 };  
  
 public static char[] *litery\_test* = {'K','L','M','N','O','P','R','S','T','U','f','k','m','n','o','p','r','s','u','w'};  
 public static int[][] *tablica\_liter\_bit\_test* = { //tablica danych testowych  
  
 {1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1}, //K  
 {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, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1}, //L  
 {1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1}, //M  
 {1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1}, //N  
 {0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0}, //O  
 {1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0}, //P  
 {1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1}, //R  
 {0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0}, //S  
 {1, 1, 1, 1, 1, 0, 0, 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, 1, 0, 0}, //T  
 {1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0}, //U  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0}, //f  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1}, //k  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1}, //m  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0}, //n  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0}, //o  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0}, //p  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0}, //r  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0}, //s  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1}, //u  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0}, //w  
 };  
  
 public Adaline() { //konstruktor  
 delta=0;  
 liczba\_wag = 35;  
 liczba\_liter = 20;  
 wspolczynnik\_nauki = 0.1;  
 blad\_globalny=0.0;  
 tablica\_wag = new double[liczba\_wag];  
 }  
  
 public void losowanie\_wag(){ //losowanie wag z przedziału <0-1> wraz z ich wyswietlaniem  
 Random randomWeight = new Random();  
 for(int i=0;i<liczba\_wag;i++) {  
 tablica\_wag[i] = randomWeight.nextDouble();  
 }  
  
 /\*for(int i=0;i<liczba\_wag;i++)  
 System.out.println("Weights are: w" + i + " = "+ tablica\_wag[i]);  
 System.out.println("\n");\*/  
 }  
  
 public boolean funkcja\_aktywacji(double sum){ // funkcja aktywacji  
 if(sum > 0.5)  
 return true;  
 else  
 return false;  
 }  
  
 public double funkcja\_sumujaca(int letter[], double[]weights){ //funkcja sumujaca perceptron  
 double sum = 0.0;  
 for(int i=0; i < liczba\_wag; i++)  
 sum += letter[i] \* weights[i];  
 return sum;  
 }  
  
 public void testAdaline() {  
 System.*out*.print("Letters from learning set: \n");  
 for(int i=0;i<liczba\_liter;i++) {  
 System.*out*.print("Letter "+ *litery*[i] +" is: ");  
 if(funkcja\_aktywacji(funkcja\_sumujaca(*tablica\_liter\_bit*[i], tablica\_wag))) {  
 System.*out*.println("DUZA LITERA");  
 }else{  
 System.*out*.println("mala litera");  
 }  
 }  
  
 System.*out*.print("Letters from testing set: \n");  
 for(int i=0;i<liczba\_liter;i++) {  
 System.*out*.print("Letter "+ *litery\_test*[i] +" is: ");  
 if(funkcja\_aktywacji(funkcja\_sumujaca(*tablica\_liter\_bit\_test*[i], tablica\_wag))) {  
 System.*out*.println("DUZA LITERa ");  
 }else{  
 System.*out*.println("mala litera ");  
 }  
 }  
  
 }  
  
 public void uczenie() {  
  
 int epoka=0; //liczba epok potrzebnych do nauki  
 boolean granica\_bledu = false; //zmienna, stwierdzajaca czy blad jest mozliwy do zaakceptowania  
  
 losowanie\_wag(); // losowanie wag  
  
 do{ //podczas jednej epoki nauka z całego zestawu danych uczacych  
  
 epoka++;  
 blad\_globalny=0.0; //zerowanie glownego bledu w celu sprawdzenia bledow podczas jednej iteracji  
 for(int i=0;i<liczba\_liter;i++) {  
 delta = *tablica\_oczekiwanych\_wynikow*[i] - funkcja\_sumujaca(*tablica\_liter\_bit*[i], tablica\_wag); //obliczanie roznicy pomiedzy wynikiem oczekiwanym a wynikiem otrzymanym  
  
 for(int j=0; j < liczba\_wag;j++) // aktualizowanie nowych wag  
 tablica\_wag[j] +=wspolczynnik\_nauki\*delta\**tablica\_liter\_bit*[i][j];  
  
 blad\_globalny+=delta\*delta; //aktualizowanie bledu glownego  
 }  
 blad\_globalny/=2;  
  
 if(blad\_globalny>0.001)  
 { granica\_bledu = false; }  
 else  
 { granica\_bledu = true; }  
  
 }while(!granica\_bledu);  
  
 System.*out*.println("Liczba epok : " + epoka );  
 System.*out*.println("Blad globalny : " + blad\_globalny);  
 for(int i=0;i<liczba\_wag;i++)  
 System.*out*.println("Waga wejscia dla : wejscie" + (i+1) + " = "+ tablica\_wag[i]);  
 System.*out*.println("\n");  
 }  
}

Delta Sigmoidal  
import java.util.Random;  
  
public class DeltaSigmoidal {  
  
 public double[] tablica\_wag; //tablica wag  
 public int liczba\_liter; //liczba liter  
 public double blad\_globalny; //blad globalny  
 public int liczba\_wag; //liczba wag  
 public double wspolczynnik\_nauki; //wspolczynnik uczenia  
 public double delta;  
 public double output;  
  
 public static char[] *litery* = {'A','B','C','E','G','I','K','M','O','R','T','a','b','c','d','e','f','g','h','l','r'};  
 public static int[] *tablica\_oczekiwanych\_wynikow* = {1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0}; //tablica oczekiwanych wynikow  
 public static int[][] *tablica\_liter\_bit* = { //tablica danych uczacych  
 {0, 1, 1, 1, 0, 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, 1, 0, 0, 0, 1}, //A  
 {1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0}, //B  
 {0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0}, //C  
 {1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0}, //D  
 {1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1}, //E  
 {1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0}, //F  
 {0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0}, //G  
 {1, 0, 0, 0, 1, 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, 1, 0, 0, 0, 1}, //H  
 {0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0}, //I  
 {1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0}, //J  
  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1}, //a  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1}, //b  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1}, //c  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1}, //d  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1}, //e  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0}, //f  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1}, //g  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1}, //h  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1}, //l  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0}, //r  
 };  
  
 public static int[][] *tablica\_liter\_bit\_test* = { //tablica danych testowych  
  
 {1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1}, //K  
 {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, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1}, //L  
 {1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1}, //M  
 {1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1}, //N  
 {0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0}, //O  
 {1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0}, //P  
 {1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1}, //R  
 {0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0}, //S  
 {1, 1, 1, 1, 1, 0, 0, 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, 1, 0, 0}, //T  
 {1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0}, //U  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0}, //f  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1}, //k  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1}, //m  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0}, //n  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0}, //o  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0}, //p  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0}, //r  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0}, //s  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1}, //u  
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0}, //w  
 };  
 public static char[] *tablica\_liter\_test* = {'K','L','M','N','O','P','R','S','T','U','f','k','m','n','o','p','r','s','u','w'};  
  
  
  
  
 public DeltaSigmoidal() { // konstruktor  
 delta=0;  
 liczba\_wag = 35;  
 liczba\_liter = 20;  
 wspolczynnik\_nauki = 0.1;  
 blad\_globalny=0;  
 output=0;  
 tablica\_wag = new double[liczba\_wag];  
 }  
  
 public void losowanie\_wag(){ //losowanie losowych wag z przedzialu <0-1> wraz z ich wyswietleniem  
 Random randomWeight = new Random();  
 for(int i=0;i<liczba\_wag;i++) {  
 tablica\_wag[i] = randomWeight.nextDouble();  
 }  
  
 /\*for(int i=0;i<liczba\_wag;i++)  
 System.out.println("Weights are: w" + i + " = "+ tablica\_wag[i]);  
 System.out.println("\n");\*/  
 }  
  
 public double funkcja\_aktywacji(double sum){ //funkcja aktywacji  
 return ( 1 / ( 1 + Math.*exp*(-1.0 \* sum)));  
 }

public double funkcja\_aktywacji(double sum){ //funkcja aktywacji - pochodna  
 return (1\*exp(-1\*sum)) / (pow(exp(-1\*sum)+1, 2));  
 }

//  
 public double funkcja\_sumujaca(int letter[], double[]weights) { // funkcja sumujaca  
 double sum = 0.0;  
 for(int i=0; i < liczba\_wag; i++)  
 sum += letter[i] \* weights[i];  
 return sum;  
 }  
  
 public void testDelta() {  
 System.*out*.print("Letters from learning set: \n");  
 for(int i=0;i<liczba\_liter;i++) {  
 System.*out*.print("Litera "+ *litery*[i] +" jest : ");  
 if(funkcja\_aktywacji(funkcja\_sumujaca(*tablica\_liter\_bit*[i], tablica\_wag))>0.5) {  
 System.*out*.println("DUZA LITERA");  
 }else{  
 System.*out*.println("mala litera");  
 }  
 }  
  
 System.*out*.print("Letters from testing set: \n");  
 for(int i=0;i<liczba\_liter;i++) {  
 System.*out*.print("Litera "+ *tablica\_liter\_test*[i] +" jest : ");  
 if(funkcja\_aktywacji(funkcja\_sumujaca(*tablica\_liter\_bit\_test*[i], tablica\_wag))>0.5) {  
 System.*out*.println("DUZA LITERA");  
 }else{  
 System.*out*.println("mala litera");  
 }  
 }  
  
 }  
  
 public void uczenie() {  
  
 boolean granica\_bledu = false; //zmienna, stwierdzajaca czy blad jest mozliwy do zaakceptowania  
 int epoka=0; //liczba epok potrzebnych do nauki  
  
 losowanie\_wag(); // losowanie wag  
  
 do{ //podczas jednej epoki nauka z całego zestawu danych uczacych  
  
 epoka++; // zwiekszenie numeru iteracji(epoki)  
 blad\_globalny=0.0; //zerowanie glownego bledu w celu sprawdzenia bledow podczas jednej iteracji  
 for(int i=0;i<liczba\_liter;i++) {  
 output=funkcja\_aktywacji(funkcja\_sumujaca(*tablica\_liter\_bit*[i], tablica\_wag));  
  
 delta = *tablica\_oczekiwanych\_wynikow*[i] - output; //obliczanie roznicy pomiedzy wynikiem oczekiwanym a wynikiem otrzymanym  
  
 for(int j=0; j < liczba\_wag;j++)  
 tablica\_wag[j] +=wspolczynnik\_nauki\*delta\**tablica\_liter\_bit*[i][j]\*(1-output)\*output; // aktualizowanie wag  
  
 blad\_globalny+=delta\*delta; //aktualizowanie bledu glownego  
 }  
 blad\_globalny/=2;  
 if(blad\_globalny>0.001)  
 { granica\_bledu = false; }  
 else  
 { granica\_bledu = true; }  
  
  
 }while(!granica\_bledu);  
  
 System.*out*.println("Blad globalny: " + blad\_globalny);  
 System.*out*.println("Liczba epok : " + epoka );  
  
 for(int i=0;i<liczba\_wag;i++)  
 System.*out*.println("Waga dla : wejscie" + (i+1) + " = "+ tablica\_wag[i]);  
 System.*out*.println("\n");  
 }  
}