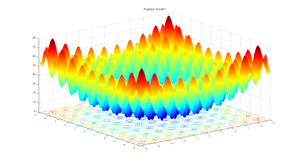
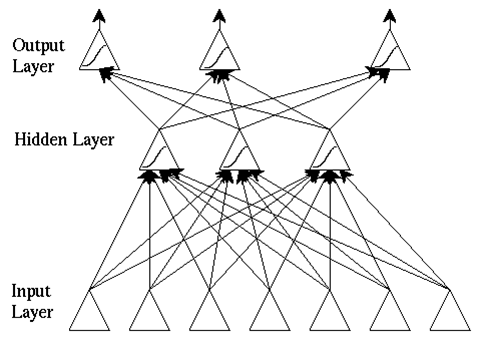
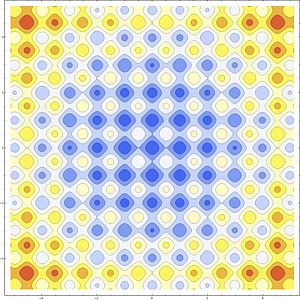
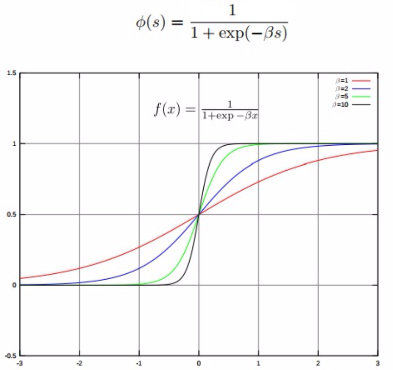
**Mateusz Bożek  
Scenariusz 3**  
**Temat :** Budowa i działanie sieci wielowarstwowej typu feedforward.

**Cel ćwiczenia :** Celem ćwiczenia jest poznanie budowy i działania wielowarstwowych sieci neuronowych poprzez uczenie kształtu wykresu funkcji matematycznej z użyciem algorytmu wstecznej propagacji błędu.  
  
**1.Syntetyczny opis budowy i wykorzystania sieci i algorytmu uczenia.**  
  
W skład sieci wielowarstwowej wchodzi:  
- warstwa wejściowa( Input Layer )  
- warstwa ukryta ( może ich my więcej niż 1 ) (Hidden Layer )  
- warstwa wyjściowa ( Output Layer )  
Kolejne warstwy służą do przetwarzania sygnałów w sieci neuronowej. Sieć neuronowa typu feedforward jest to sieć, w której sygnał ma stały, ustalony kierunek przepływu danych - od warstwy wejściowej, przez warstwy ukryte po warstwę wyjściową. Warstwy te są ze sobą tak połączone, że dane wyjście jednej warstwy jest jednocześnie danymi wejściowymi drugiej warstwy. Sygnał wyjściowy nie jest dzieloną, więc sygnał ten jest podawany wszystkim neuronom kolejnej warstwy.  
Neurony w każdej z warstw nie są ze sobą połączone.  
  
  
   
  **Rysunek przedstawia schemat sieci wielowarstwowej**  
   
  
W celu nauczenia sieci wielowarstwowej użyta została funkcja Rastring. Jej zadaniem jest zwrócenie wartości z, przy podanych wartościach x oraz y. Przyjmują one wartości od -2 do 2. Wzór funkcji Rastring przyjmuje postać :  
 ]  
gdzie :  
n - wymiar ( u nas wynosi oni 2 ze względu na x,y )  
xi - współrzędna   
  
   
 **Rysunek przedstawia funkcje Rastring w wymiarze 3D**



  
 **Rysunek przedstawia funkcje Rastring w wymiarze 2D** Kolor niebieski w centrum - wartość minimalna (0,0,0)  
 Kolor czerwony w rogach - wartość maksymalna  
  
W projekcie użyto 1600 danych uczących, które składają się z 2 danych wejściowych x i y, natomiast daną wyjściową jest wartość z. Wszystkie dane są znormalizowane do przedziału od 0 do 1. Jako funkcja aktywacji używa została funkcja Sigmoidalna unipolarna dana wzorem :  
   
  
  
Schemat uczenia sieci wielowarstwowej :  
Krok 1: Przed przystąpieniem do nauki, podane dane uczące muszą zostać wcześniej normalizowane. Aby to zrobić należy skorzystać ze wzoru :

Krok 2 : Wybór współczynnika uczenia oraz .  
gdzie:  
 - współczynnik uczenia ( 0 , 1 )  
- błąd maksymalny

Krok 3 : Losowanie początkowych wartości wag z zakresu od 0 do 1

Krok 4 : Ustawienie sumy błędu średniokwadratowego .

Krok 5 : Aktualizacja wag za pomocą wzoru :

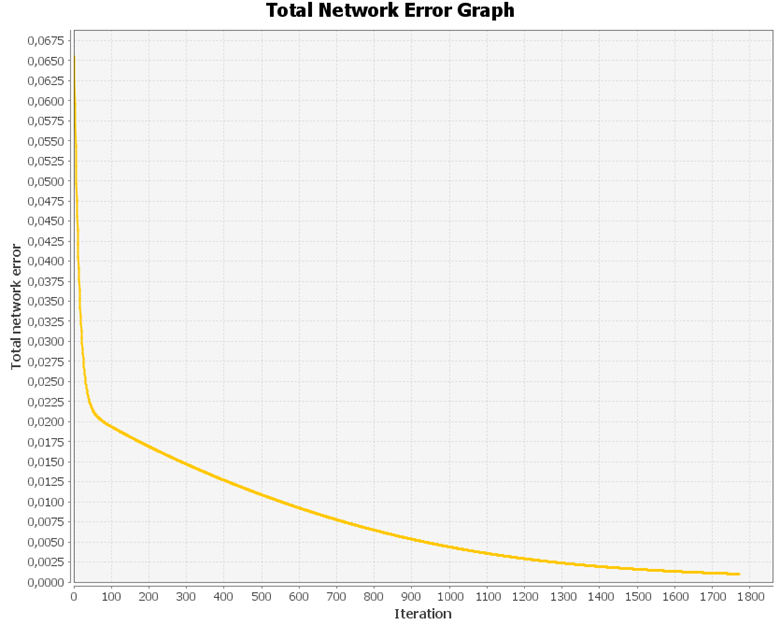
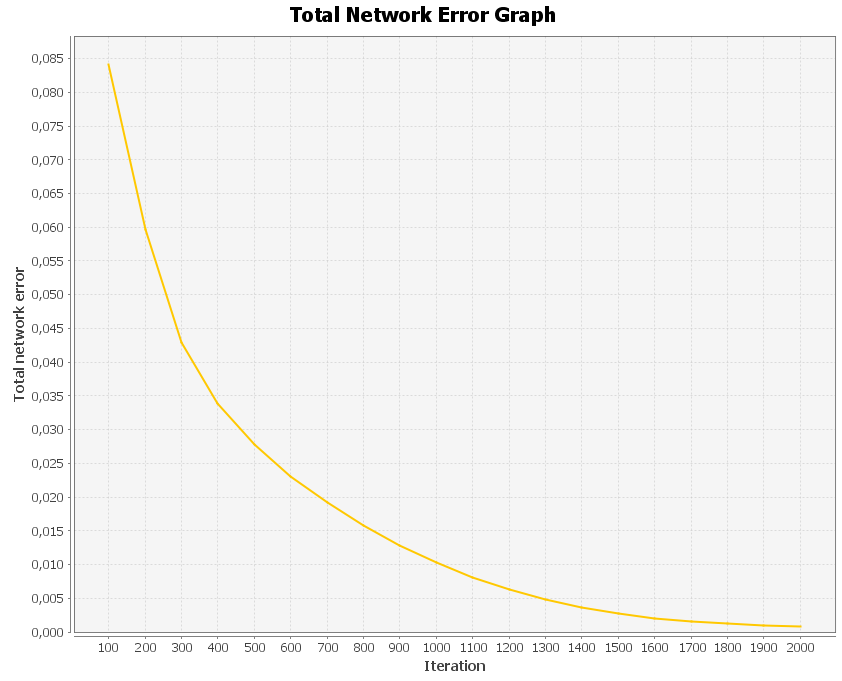
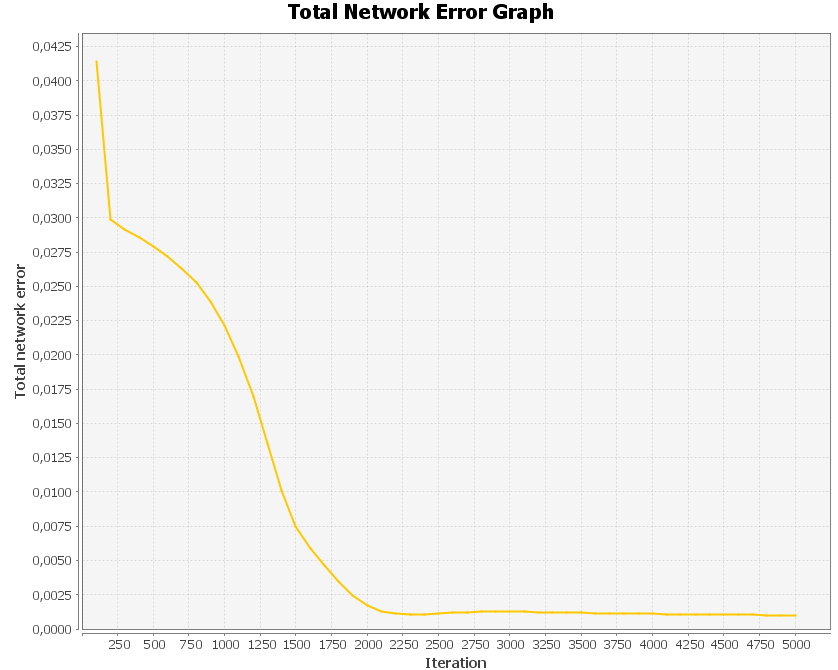
Zmiana wag z warstwy wejściowej i do warstwy ukrytej j:

Zmiana wag z warstwy ukrytej j do warstwy wyjściowej k:

gdzie:

Krok 6 : Obliczenie łącznego błędu epoki:

gdzie:  
 - wartość oczekiwana  
 - wartość na wyjściu

7. Gdy nierówność jest prawidłowa, uczenie jest zakończone. W przypadku gdy nierówność ta nie zachodzi, następuje powrót do kroku numer 4.  
  
  
 **2.Zestawienie wyników:  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
3.Analiza programu :**  
Powyższe wykresy przedstawiają zależności liczby epok od współczynnika uczenia podanego do sieci.   
Na wykresie pierwszym widzimy, że przy niskim współczynniku wynoszącym 0,1 liczba epok dla przedstawionych w legendzie sieci jest duża. Na różnicę między sieciami główny wpływ ma liczba warstw oraz liczba neuronów w danej warstwie. Najmniej epok potrzeba sieci, która posiada 3 warstwy, gdzie liczba neuronów w warstwie wzrasta ( 5, 10, 20 ), natomiast najwięcej potrzeba sieci, która ma zaledwie jedną warstwę ukrytą, która zawiera 5 neutronów. Wraz ze wzrostem współczynnika uczenia, sieci uczą się znacznie szybciej. przy współczynniku 0,3 różnica w liczbie epok nie jest już tak wyraźna jak przy współczynniku 0,2. Drugi z wykresów przedstawia te same sieci co wykres pierwszy jednak różnica polega na zmianie ilości neuronów w poszczególnej warstwie. W przypadku gdy liczba neuronów wzrasta wraz z każdą kolejną warstwą, potrzebna jest mniejsza liczba epok. Oznacza to, że na liczbę epok ( szybkość uczenia ) wpływ ma nie tylko współczynnik uczenia, liczba warstw ale także liczba neuronów w danej warstwie.  
  
  
 **WYKRESY BŁĘDU ŚREDNIOKWADRATOWEGO dla danych sieci :**  
   
 Sieć [2, 5, 10, 20,1]  
  
   
 Sieć [2, 5, 10, 1]  
  
   
 Sieć[2, 10, 1]Powyższe wykresy przedstawiają wartości błędu średniokwadratowego dla każdej z podanych sieci. Przebieg wartości błędu średniokwadratowego dla każdej z nich przebiega w podobny sposób. W początkowej iteracji wartości te są duże jednak wraz z upływem czasu nauki, wartości spadają. W przypadku sieci [2, 10, 1] wykres przebiega nieco inaczej od dwóch pozostałych. Sytuacja ta może być spowodowana stabilnością aktualizacji wag. **4.Wnioski:**Na wyniki uczenia sieci wpływ ma za równo współczynnik uczenia oraz sama struktura sieci. Wraz ze wzrostem współczynnika uczenia, sieć uczy sie szybciej ( potrzeba mniejszej ilości epok ). Dla początkowych jego wartości, sieć potrzebuje bardzo dużo czasu. Optymalnie przy współczynniku 0,3 liczba epok do nauczenia jest podobna. Oprócz współczynnika uczenia duży wpływ na naukę sieci ma również liczba neuronów w danej warstwie. Sieć uczy się szybciej gdy występuje większa liczba warstw. Jednak należy zwrócić uwagę liczbę neuronów w warstwach ponieważ, sieć uczy się lepiej gdy liczba neuronów wzrasta z każdą następną warstwą, niż w przypadku przeciwnym, gdy liczba neuronów w każdej następnej warstwie jest mniejsza. Wówczas sieć uczy się gorzej. **5.Listing programu:**package Rastring;  
import org.neuroph.core.NeuralNetwork;  
import org.neuroph.core.data.DataSet;  
import org.neuroph.core.data.DataSetRow;  
import org.neuroph.nnet.MultiLayerPerceptron;  
import org.neuroph.nnet.learning.BackPropagation;  
import org.neuroph.util.TransferFunctionType;  
  
import java.io.File;  
import java.io.FileWriter;  
import java.io.IOException;  
import java.util.Scanner;  
  
public class Main {  
  
 private static double *xyMin* = -2.0;  
 private static double *xyMax* = 2.0;  
 private static double *zMin* = Double.*MIN\_VALUE*;  
 private static double *zMax* = Double.*MAX\_VALUE*;  
  
 public static void main(String[] args) throws IOException {  
  
 //utworzeine danych  
 DataSet daneUczace = new DataSet(2, 1); //trainingData  
  
 //wczytanie danych wejściowcyh z pliku  
 File newFile = new File("learning\_data.txt");  
 FileWriter zapis = new FileWriter(newFile);  
  
 for (double i=-2.0 ; i<=2.0 ; i+=0.1)  
 for(double j=-2.0 ; j<= 2.0 ; j+=0.1)  
 {  
 double z = 10 \* 2 + Math.*pow*(i, 2) - 10 \* Math.*cos*(2 \* Math.*PI* \* i) + Math.*pow*(j, 2) - 10 \* Math.*cos*(2 \* Math.*PI* \* j);; // obliczenie wartosci z  
  
 zapis.write(i + ";" + j + ";" + z + "\n");   
  
 if(z <= *zMin*) // obliczenie wartości minimum i maksimum na przedziale ( -2 , 2 )  
 *zMin*=z;  
 else if ( z >= *zMax*)  
 *zMax*=z;  
  
 }  
 zapis.close();  
  
 // wczytywanie danych z pliku  
 Scanner wejscie = new Scanner(newFile); //wejscie  
 String[] wynik;  
 do {  
 String line = wejscie.nextLine();  
 wynik = line.split(";");  
  
 //dodanie rzędu danych uczących  
 daneUczace.addRow(new DataSetRow(new double[]{Rastrigin.*normalizacja\_danych*(Double.*parseDouble*(wynik[0]), *xyMin*, *xyMax*), // normalizacja danych - wejscie 1  
 Rastrigin.*normalizacja\_danych*(Double.*parseDouble*(wynik[1]), *xyMin*, *xyMax*) // normalizacja danych wejscie 2  
 }, new double[]{Rastrigin.*normalizacja\_danych*(Double.*parseDouble*(wynik[2]), *zMin*, *zMax*)})); // normalizacja danych - wyjscie  
 } while (wejscie.hasNext());  
  
 //wybranie metody wstecznej propagacji ( backPropagation ) i ustawienie max błędu i współczynnika uczenia  
 BackPropagation backPropagation = new BackPropagation();  
 backPropagation.setMaxError(0.015);  
 backPropagation.setLearningRate(0.03);  
  
 //utworzenie sieci wielowarstwowej i ustalenie ilośći warstw wewnętrznych  
 MultiLayerPerceptron multiLayerPerceptron = new MultiLayerPerceptron(TransferFunctionType.*SIGMOID*, 2, 5,10,20, 1); // między wejscie i wyjscie znajduja sie ukryte wartwy  
 multiLayerPerceptron.setLearningRule(backPropagation);  
  
 System.*out*.println("Rozpoczecie nauki sieci z parametrami : \n" + "Blad maksymalny : " +  
 backPropagation.getMaxError() +"\n"+"Wspolczynnik uczenia : "+backPropagation.getLearningRate());  
 multiLayerPerceptron.learn(daneUczace); // rozpoczecie procesu nauki  
  
 // Wyswietlenie liczby iteracji oraz całkowitego błędu uczenia w sieci neuronowej  
 System.*out*.println("Liczba iteracji : " + backPropagation.getCurrentIteration()+"\n"+"Blad : "+ backPropagation.getErrorFunction().getTotalError());  
  
 //zapisanie utworzonej sieci  
 multiLayerPerceptron.save("siec.nnet");  
  
  
  
 //testowanie nauczonej sieci  
 NeuralNetwork siec\_wielowarstwowa = NeuralNetwork.*createFromFile*("siec.nnet");  
  
 int liczba\_iteracji = 1;  
  
 for (DataSetRow dataSetRow : daneUczace.getRows()) {  
 double[] input = dataSetRow.getInput();  
 System.*out*.println("Epoka : " + liczba\_epok + ": x = " + input[0] + " y = " + input[1]);  
  
 double[] desiredOutput = dataSetRow.getDesiredOutput();  
 siec\_wielowarstwowa.setInput(dataSetRow.getInput());  
 siec\_wielowarstwowa.calculate();  
 double[] output = siec\_wielowarstwowa.getOutput();  
 System.*out*.println("Oczekiwana wartosc wyjsciowa : " + desiredOutput[0]);  
 System.*out*.println("Wyjscie : " + output[0]);  
 System.*out*.println();  
 liczba\_iteracji++;  
 }  
 }  
}  
  
  
  
package Rastring;  
  
public class Rastrigin {  
 //normalizacja danych do przedziału od 0 do 1  
 static double normalizacja\_danych(double wart, double wart\_pocz, double wart\_konc)  
 {  
 return ((wart - wart\_pocz) / (wart\_konc - wart\_pocz)) \* (1.0 - (0.0)) + (0.0);  
 }  
}

