**Scenariusz 3**

Jacek Pozowski, gr. 1

**1. Cel ćwiczenia**

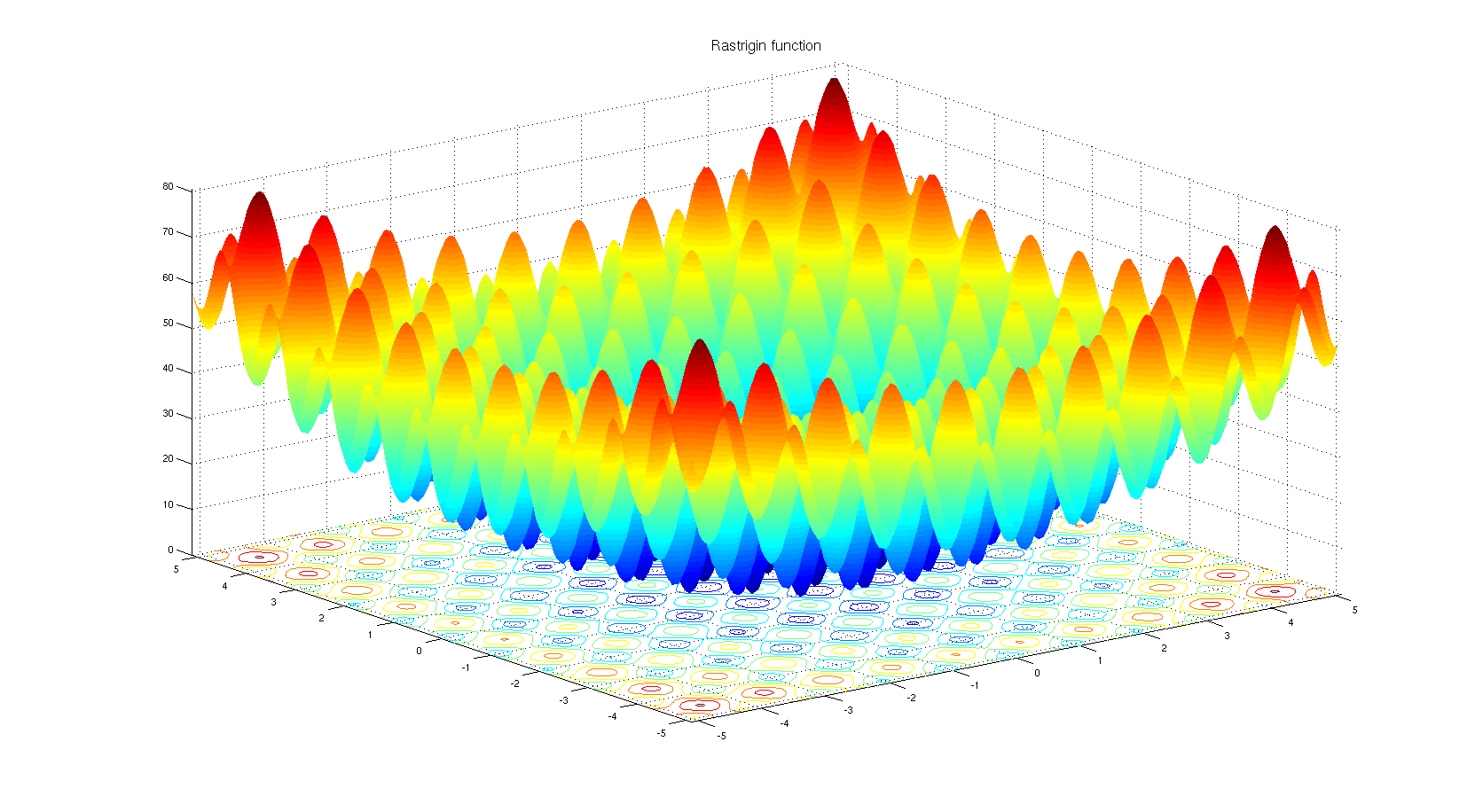
Celem ćwiczenia jest po znanie budowy i działania wielowarstwowych sieci neuronowych po przez uczenie kształtu wykresu funkcji matematycznej z użyciem algorytmu wstecznej propagacji błędu.

**2. Opis budowy sieci i algorytmów uczenia.**

Celem budowanej sieci jest rozpoznawanie funkcji rastrigin. Funkcja ta w naszym przyjmuje za wejście współrzędne x, y z przedziału <-2; 2> i zwraca współrzędną z.

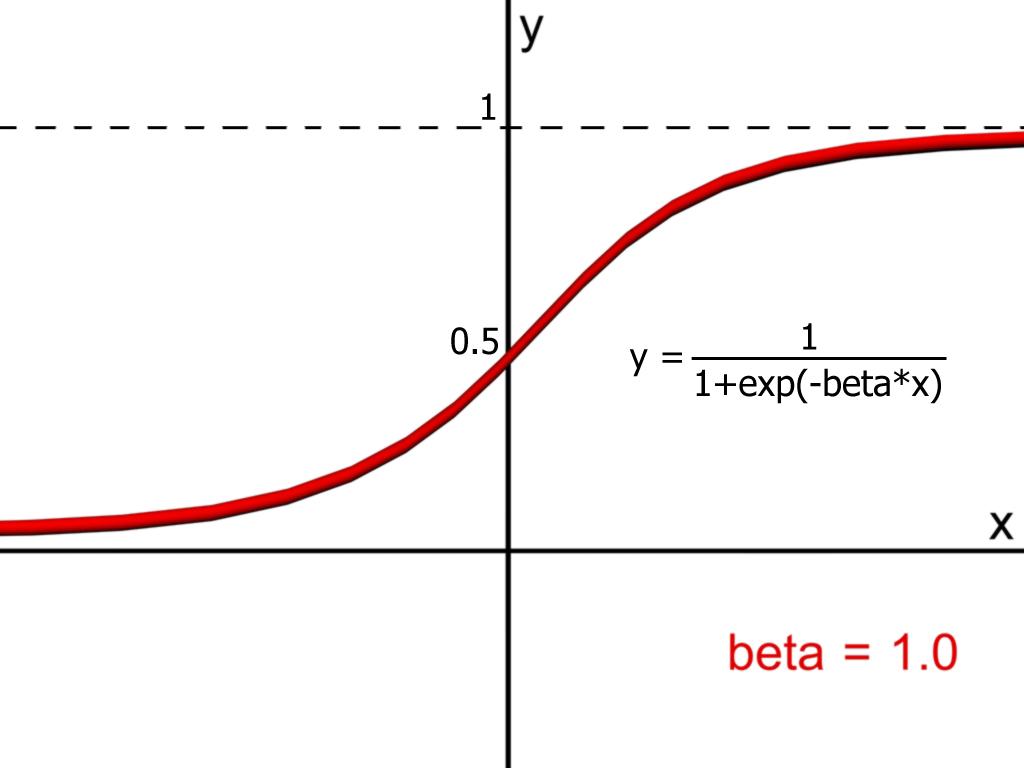
Wzór funkcji rastrigin:

Rastrigin ma minimum globalne w punkcie (0, 0, 0).

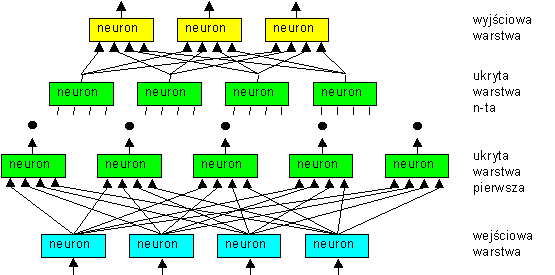


Rys. 1 Funkcja Rastrigin

Jako dane uczące służy zestaw 1600 wierszy, w każdym znajduje się jako wejście współrzędna x, y oraz jako wyjście współrzędna z. Wszystkie wartości są znormalizowane do przedziału <0; 1>, żeby było możliwie uczenie sieci wielowarstwowej. Jako funkcji aktywacji użyto funkcji sigmoidalnej.



Sigmoidalna Funkcja Unipolarna



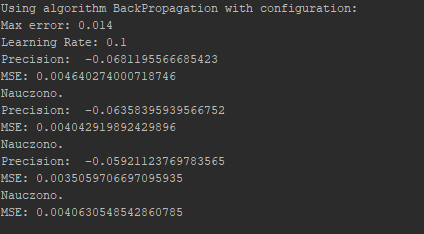
Rys. 3 Ogólny schemat sieci neuronowej wielowarstwowej

Sieć wielowarstwowa składa się z warstwy wejściowej (Input Layer), co najmniej jednej warstwy ukrytej (HiddenLayer) oraz warstwy wyjściowej (OutputLayer). Warstwy ukryte służą do przetwarzania sygnałów w sieci neuronowej.

Zastosowany typ ***feedforward*** oznacza, że w sieci istnieje z góry określony kierunek przepływu danych – dane przechodzą od warstwy wejściowej przez wszystkie warstwy ukryte kończąc na warstwie wyjściowej (dane nie mogą się „cofać” w użytej sieci). Każda z warstw jest powiązana tylko z warstwą poprzednią i następną. Dane wyjściowe każdego neuronu w jednej warstwie są jednocześnie danymi wejściowymi dla neuronów w kolejnej warstwie na zasadzie każdy z każdym. Sygnał wyjściowy nie jest dzielony, więc jest podawany taki sam na wejścia wszystkich neuronów kolejnej warstwy. Natomiast neurony w jednej warstwie nie są ze sobą w żaden sposób połączone.

**3. Otrzymane wyniki i ich analiza**

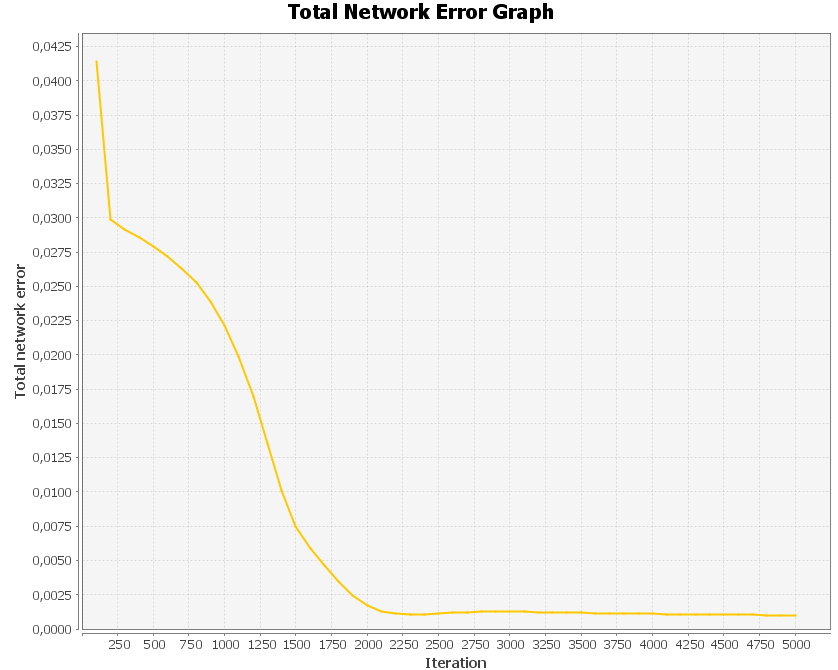
Przykładowe działanie programu:

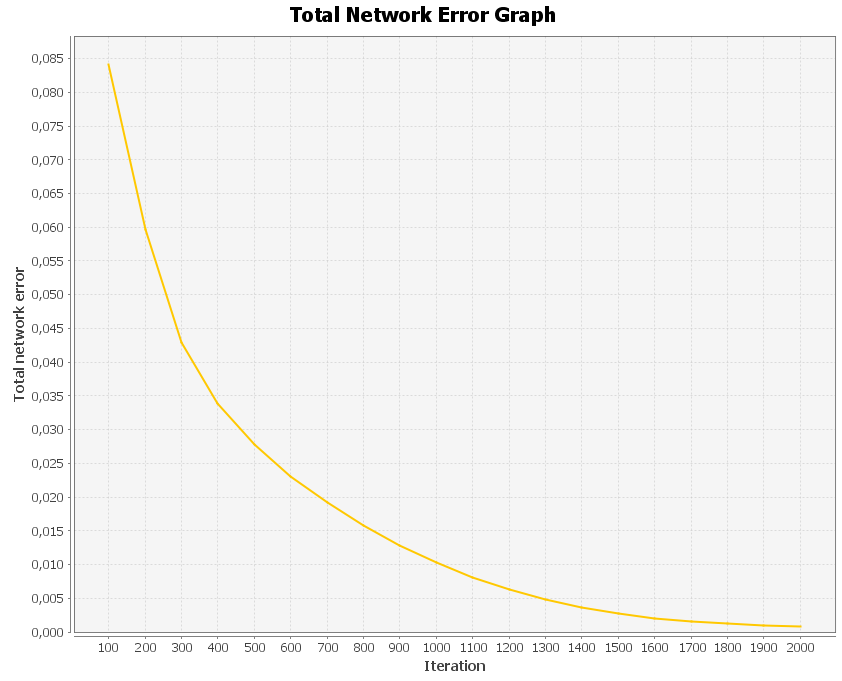


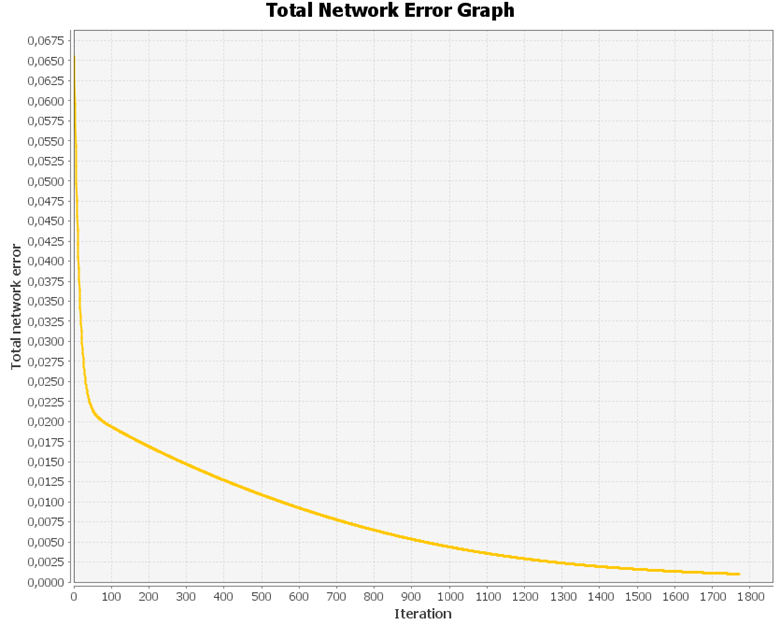
*Analiza*

Z powyższego wykresu można odczytać, że na efektywność uczenia ma wpływ współczynnik uczenia.

Ponadto sama struktura sieci wielowarstwowej ma wpływ na szybkość uczenia. Im więcej warstw, więcej neuronów, tym sieć uczyła się sprawniej.







Błąd średniokwadratowy

Powyższy wykres pokazuje jak maleje błąd średniokwadratowy wraz z kolejnymi iteracjami. Po danym czasie błąd utrzymuje się na danym poziomie i w nim oscyluje. Dla bardziej rozbudowanych sieci błąd średniokwadratowy maleje szybciej i potrzeba mniej iteracji.

**4. Wnioski**

-Podczas tworzenia sieci wielowarstwowej trzeba zwrócić największą uwagę na jej strukturę oraz na współczynnik uczenia. Sieci wielowarstwowe posiadające większą liczbę neuronów oraz warstw ukrytych na ogół uczą się bardziej efektywnie od sieci z ich mniejszą ilością. Jednakże w samej strukturze sieci ważne jest rozmieszczenie danych neuronów.

- Im więcej neuronów w początkowych warstwach ukrytych, tym sieć zaczyna uczyć się gorzej aniżeli w przypadku, gdy miałaby ich mniej.

-mała struktura sieci może skutkować niestabilnością przy aktualizacji wag na co wskazuje uskok w wykresie błędu pierwszej sieci.

-Najbardziej optymalną wersją jest tworzenie sieci z większą ilością warstw, gdzie w każdej kolejnej warstwie ukrytej będzie więcej neuronów w porównaniu do warstwy poprzedniej. Ponadto trzeba zwrócić uwagę na współczynnik uczenia, który pełni także olbrzymią rolę w procesie uczenia.

**5.Kod programu**

**Dane.py**

import math  
import pybrain3  
import sys  
  
daneWejsciowe=pybrain3.datasets.supervised.SupervisedDataSet(2, 1)  
  
x = -2  
  
  
def normalization(c, y, z):  
 v = ((c-y)/(z-y))  
 return v  
  
  
def rastrigin(a, b):  
 return ((10\*2)+math.pow(a,2))-((10\*math.cos(2\*math.pi\*a))+(10\*2)+math.pow(b,2))-(10\*math.cos(2\*math.pi\*b))  
  
  
for i in range(0,40):  
 for j in range(0, 40):  
 daneWejsciowe.addSample(((normalization((x+(i/10)), -2, 2)),  
 (normalization(x+(j/10), -2, 2))),  
 normalization((rastrigin((x + (i / 10)), (x + (j / 10)))), sys.float\_info.min,  
 sys.float\_info.max  
 ))

**siec.py**

#network = RecurrentNetwork() # nowa siec  
import pybrain3  
  
network=pybrain3.FeedForwardNetwork();  
  
inLayer = pybrain3.LinearLayer(2) # warstwa wejsciowa  
hiddenLayer = pybrain3.SigmoidLayer(10) # tworze ukryta warstwe  
#hidden2Layer = pybrain3.SigmoidLayer(10) # tworze ukryta warstwe  
#hidden3Layer = pybrain3.SigmoidLayer(20) # tworze ukryta warstwe  
#hiddenLayer=pybrain3.TanhLayer(5);  
outLayer = pybrain3.LinearLayer(1) # tworze warstwe wyjsciowa  
bias = pybrain3.BiasUnit() # inicjalizuje Bias  
  
network.addInputModule(inLayer)  
network.addModule(bias)  
network.addModule(hiddenLayer)  
network.addOutputModule(outLayer)  
  
bias\_to\_hidden = pybrain3.FullConnection(bias, hiddenLayer) # lacze warstwy  
in\_to\_hidden = pybrain3.FullConnection(inLayer, hiddenLayer)  
#hidden\_to\_hidden2 = pybrain3.FullConnection(hiddenLayer, hidden2Layer)  
#hidden2\_to\_hidden3 = pybrain3.FullConnection(hidden2Layer, hidden3Layer)  
hidden\_to\_out = pybrain3.FullConnection(hiddenLayer, outLayer)  
  
network.addConnection(bias\_to\_hidden) # dodaje polaczenie do sieci  
network.addConnection(in\_to\_hidden)  
network.addConnection(hidden\_to\_out)  
  
network.sortModules()

**Trener.py**

import pybrain3  
from pybrain3.supervised import BackpropTrainer  
from pybrain3.tools.validation import Validator  
  
import dane  
import siec  
  
trener = BackpropTrainer(siec.network, dane.daneWejsciowe, learningrate=0.1)  
  
trener.trainEpochs(1000)  
  
testInput= dane.daneWejsciowe['input']  
testTarget= dane.daneWejsciowe['target']  
  
errorComparator=0.014  
  
MSE=0  
print("Using algorithm BackPropagation with configuration:")  
print("Max error: 0.014")  
print("Learning Rate: 0.1")  
for i in range(3):  
 tmp=siec.network.activate(testInput[i])  
 print("Precision: ",tmp[0])  
 print("MSE:",Validator.MSE(siec.network.activate(testInput[i]),testTarget[i]))  
 MSE+=Validator.MSE(siec.network.activate(testInput[i]),testTarget[i])  
 if errorComparator>tmp:  
 print("Nauczono.")  
 else:  
 print("NIE Nauczono.")  
  
print("MSE:", MSE/3)