Rafał Szyszka

Sprawozdanie 3 – budowa i działanie sieci wielowarstwowej typu feedforward.

1.Wstęp

W ćwiczeniu zostały wykonane sieci wielowarstwowe przy użyciu programu javowego *NeurophStudio*. Jest to darmowy program na licencji Apache 2.0 oparty o framework *Neuroph.*

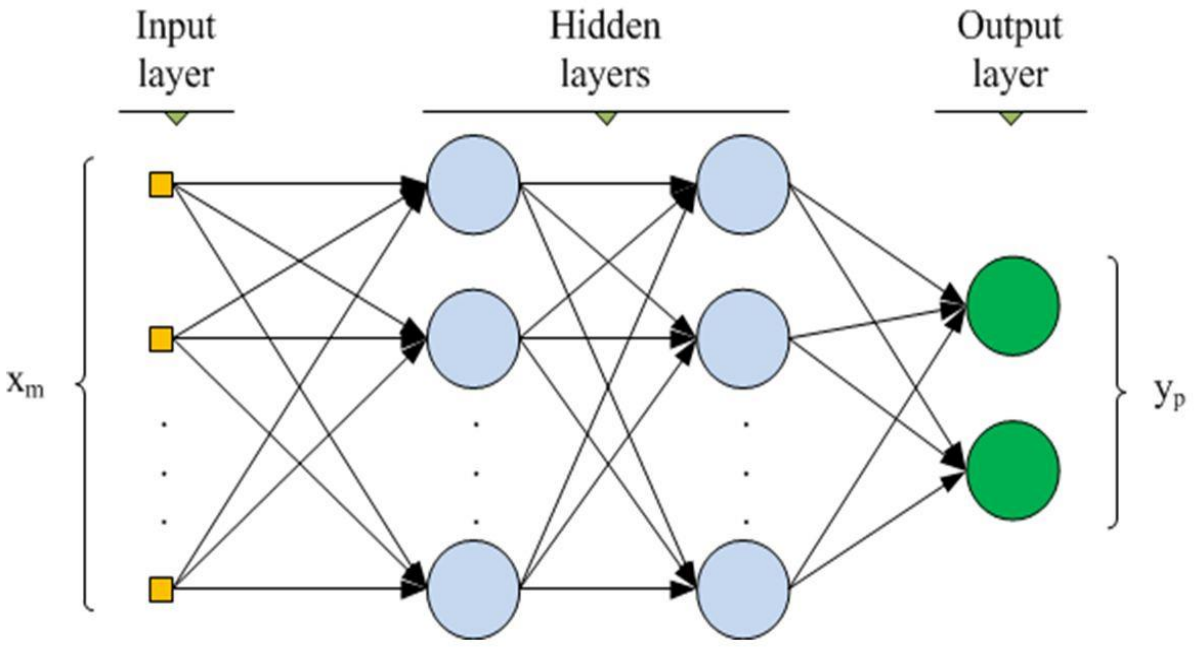
Zaimplementowano także klase generującą dane uczące i testujące dla funkcji Rastrigin 3D z przedziału -2 do 2 znormalizowane do przedziału 0 do 1.

2.Syntetyczny opis budowy użytej sieci i algorytmu uczenia

Typowa struktura sieci typu Multi Layer Neuron zakłada istnienie takiej architektury połączeń pomiędzy neuronami, w której wszystkie wyjścia warstwy wcześniejszej połączone są z odpowiednimi wejściami każdego neuronu warstwy następnej.

W klasycznej terminologii sieci neuronowych wyróżnia się warstwę wejściową, jedną lub dwie warstwy ukryte oraz warstwę wyjściową. Warstwa wejściowa pobiera dane z otoczenia i przesyła je do pierwszej warstwy ukrytej. Następnie sygnał przesyłany jest na wejścia pierwszej warstwy ukrytej, która przetwarza dane i generuje sygnał wyjściowy podawany na wejścia warstwy kolejnej. Przedstawiony na rysunku 1 schemat powtarza się dla wszystkich kolejnych warstw ukrytych i kończy na warstwie wyjściowej, która zgodnie ze wzorcową architekturą oblicza wartości wyjść całej sieci i przekazuje je na zewnątrz.

Rys1 – Przykład sieci wielowarstwowej

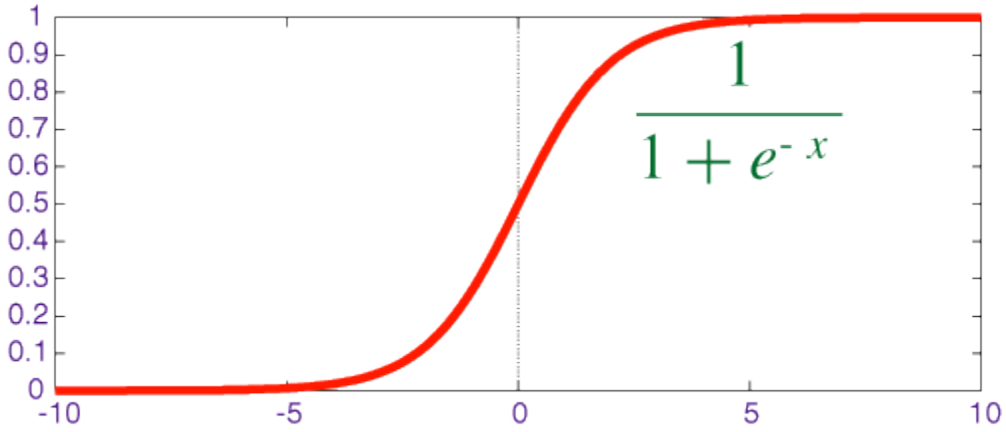


Algorytm wstecznej propagacji błędu (Backpropagation algorithm) sprowadza się do modyfikacji każdej z wag proporcjonalnie do wartości pochodnej cząstkowej funkcji celu. Modyfikacja wag w k-tej iteracji polega na odjęciu od wartości wag z iteracji poprzedniej (k - 1) wektora gradientu obliczonego dla bieżącej obserwacji. Algorytm jest sparametryzowany dzięki użyciu tzw. współczynnika uczenia, lub inaczej długości kroku, oznaczonego w poniższym równaniu symbolem α.

$$\mathbf{w}_i^{(n)}(k) = \mathbf{w}_i^{(n)}(k-1) - \alpha \frac{\partial Q(k)}{\mathbf{w}_i^{(n)}}$$

$$\alpha > 0$$

Wykorzystano unipolarną sigmoidalną funkcję aktywacji:



3. Przebieg ćwiczenia.

a) wygenerowanie 3500 danych uczących i 1500 testujących dla funkcji Rastrigin 3D dla danych wejściowych z przedziału -2 do 2. Formuła funkcji:

$$z = 20 + x^2 + y^2 - 10 * (cos(2\pi x) + cos(2\pi y))$$

b) normalizacja danych wejściowych i wyjściowych za pomocą wzoru:

$$V' = \frac{V - min}{max - min} * (new\_\max - new\_\min) + new\_min$$

Gdzie **[min,max]** jest przedziałem, w którym mieszczą się dane wejściowe, natomiast **[new_min,new_max]** jest nowym przedziałem danych.

c) stworzenie 3 wariantów sieci neuronowych, z algorytmem wstecznej propagacji błędu.

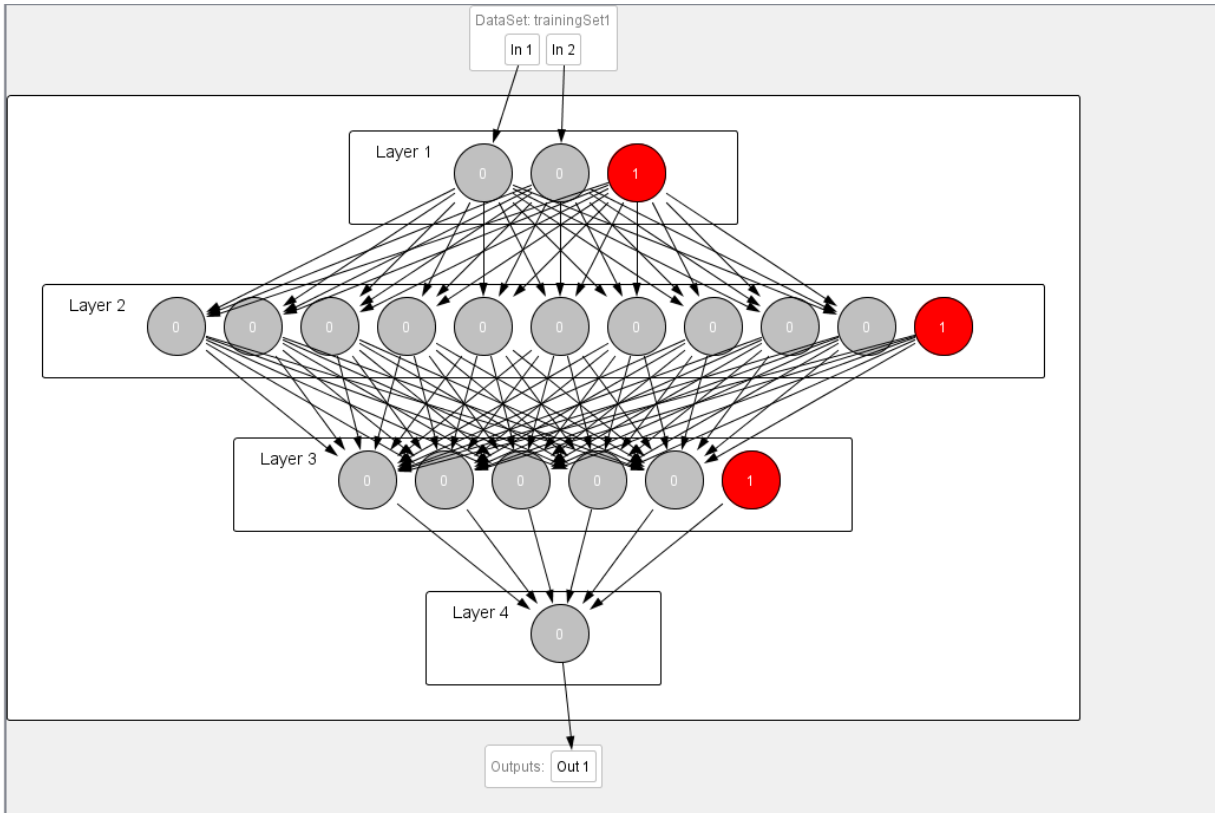
d) uczenie sieci dla różnych współczynników uczenia i różnej ilości warstw.

e) testowanie sieci.

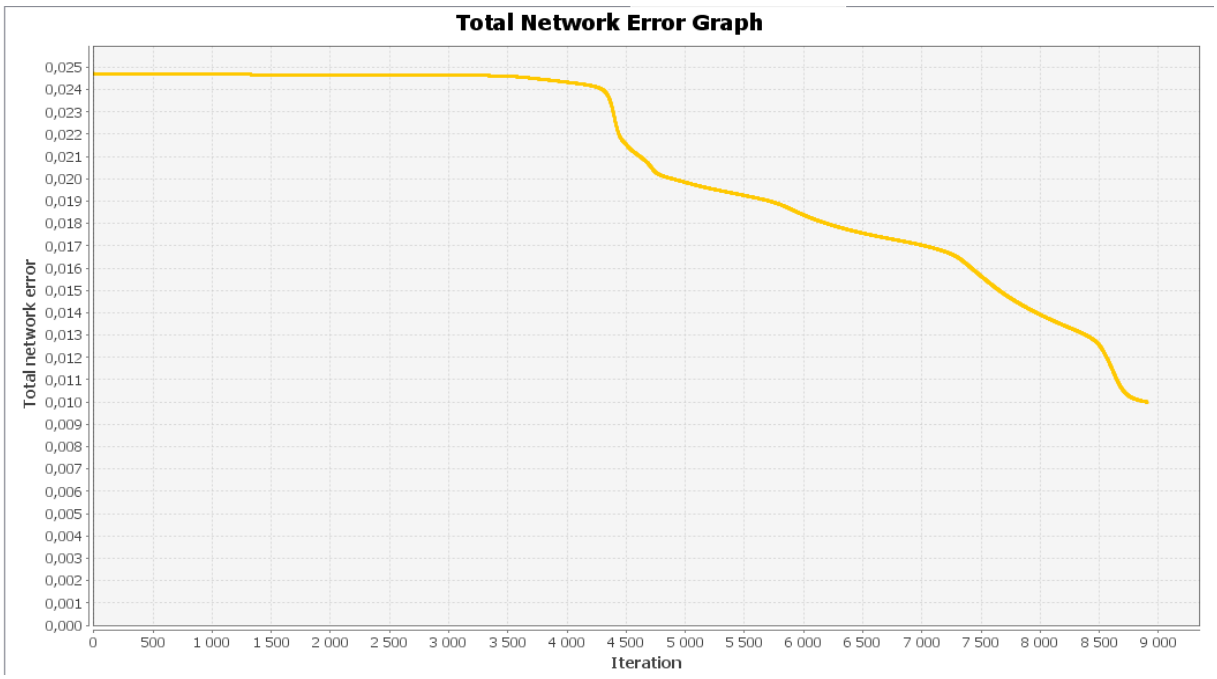
4. Zestawienie i analiza otrzymanych wyników.

Sieci uczone były do momentu uzyskania błędu na poziomie 0,01 lub liczby iteracji równego 10000.

Konfiguracja 1.

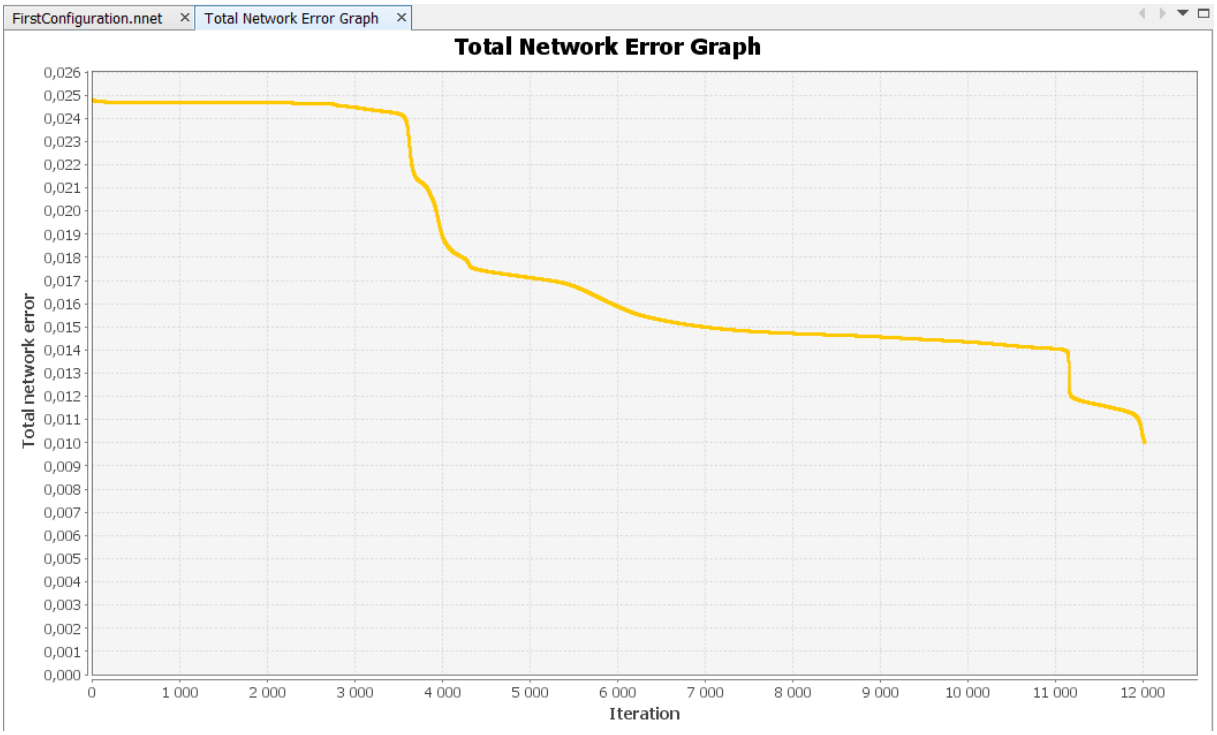


a)Wykres zależności błędu MSE od ilości iteracji dla learn rate = 0,05



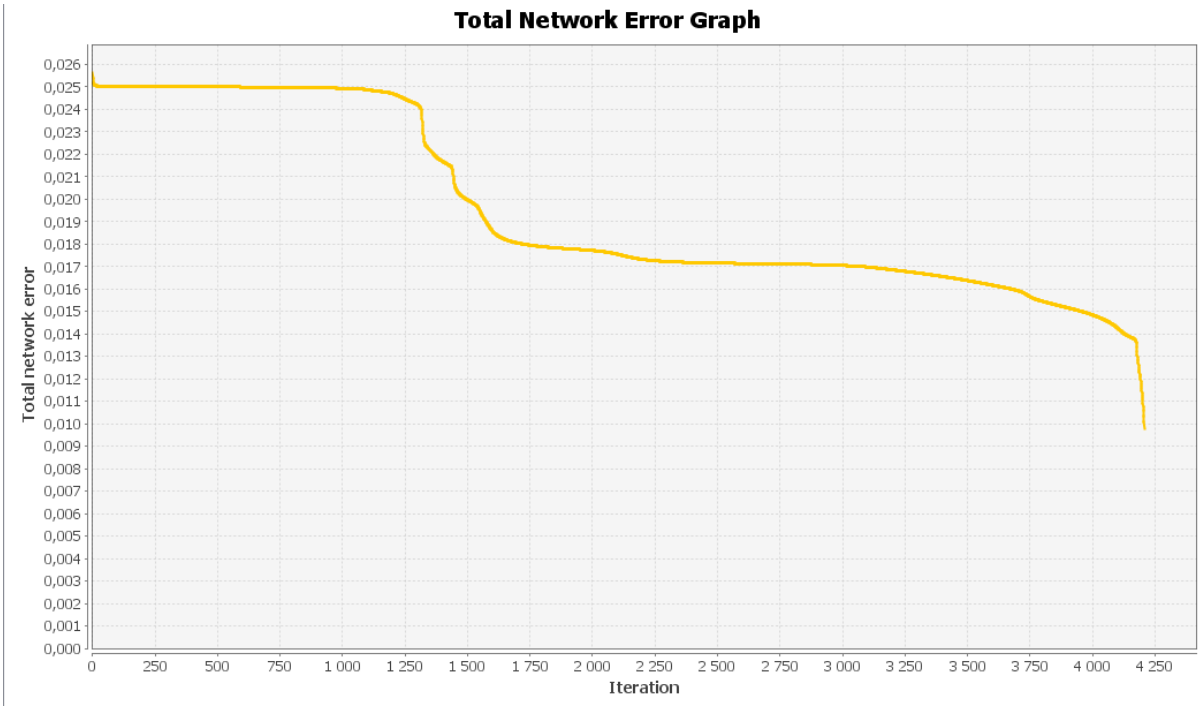
Na powyższym wykresie widzimy, że przez niecałe 4500 iteracji błąd zmniejszał się w niewielkim stopniu, po czym zauważamy przeskok i błąd maleje aż do osiągniecia pożądanej wartości 0,01.

b) Wykres zależności błędu MSE od ilości iteracji dla learn rate = 0,1



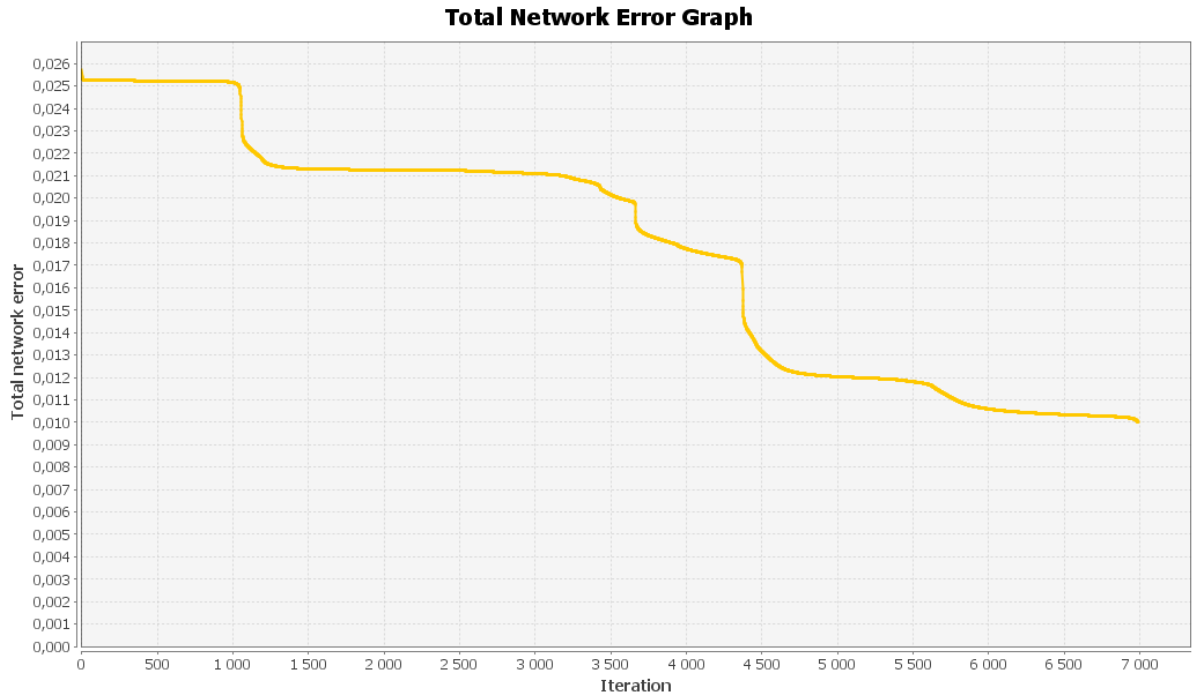
Na tym wykresie już widzimy, że zmniejszenie błędu nastąpiło szybciej, ale mamy więcej nagłych spadków i ilość iteracji to uzyskania pożądanej wartości była większa

c) Wykres zależności błędu MSE od ilości iteracji dla learn rate = 0,5



Wyraźnie widać, że przy learn rate = 0,5 sieć osiągnęła błąd uczenia na poziomie 0,1 już przy około 4200 iteracji. Wyraźnie jednak widać charakterystyczne nagłe spadki.

d) Wykres zależności błędu MSE od ilości iteracji dla learn rate = 0,8



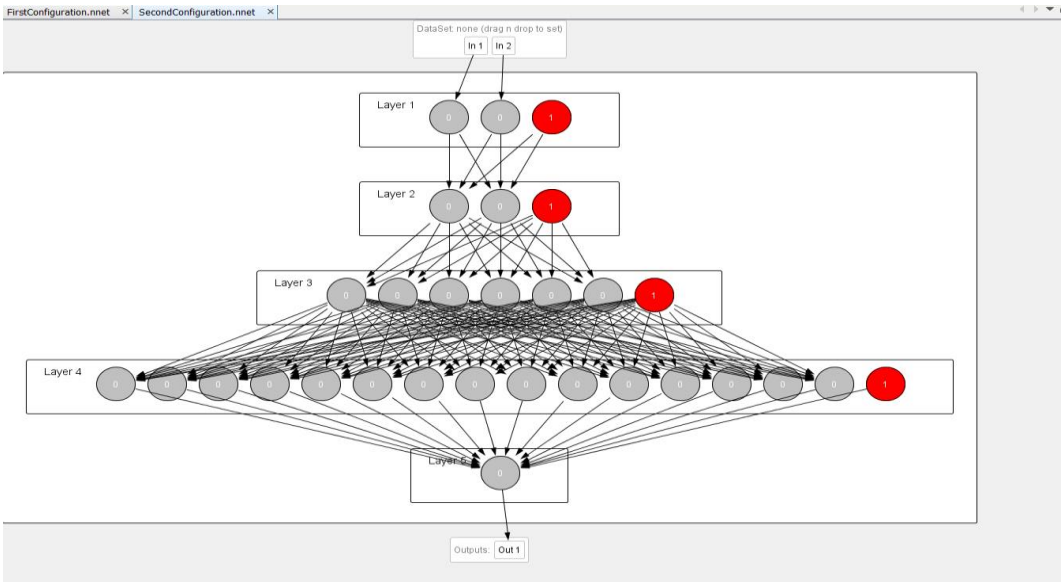
Struktura wykresu w tym przypadku jest również podobna. Sieć uczy się tutaj trochę dłużej co oznacza, że krok uczenia już prawdopodobnie jest za duży.

Testując sieci na innych danych, uzyskano błędy MSE:

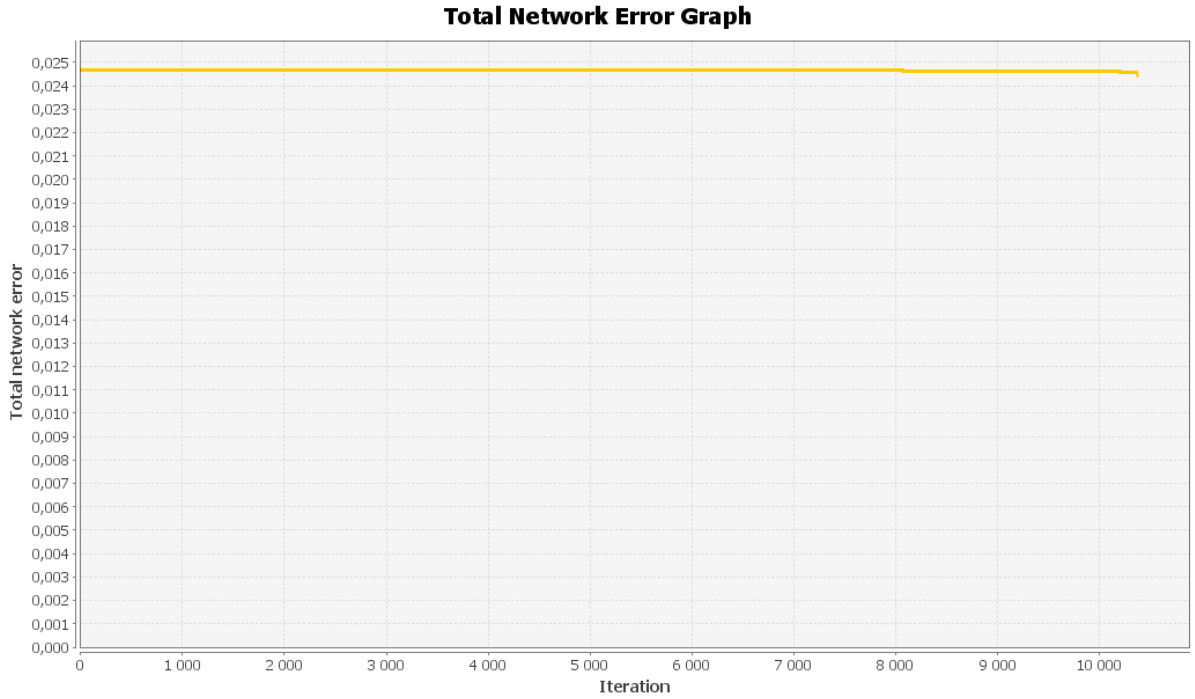
LR	MSE
0,05	0,019729
0,1	0,020776
0,5	0,019098

Zauważyć można że błąd MSE przy danych testujących był około 2 razy większy niż w przypadku uczenia gdzie błąd dochodził do wartości 0,01.

Konfiguracja 2.

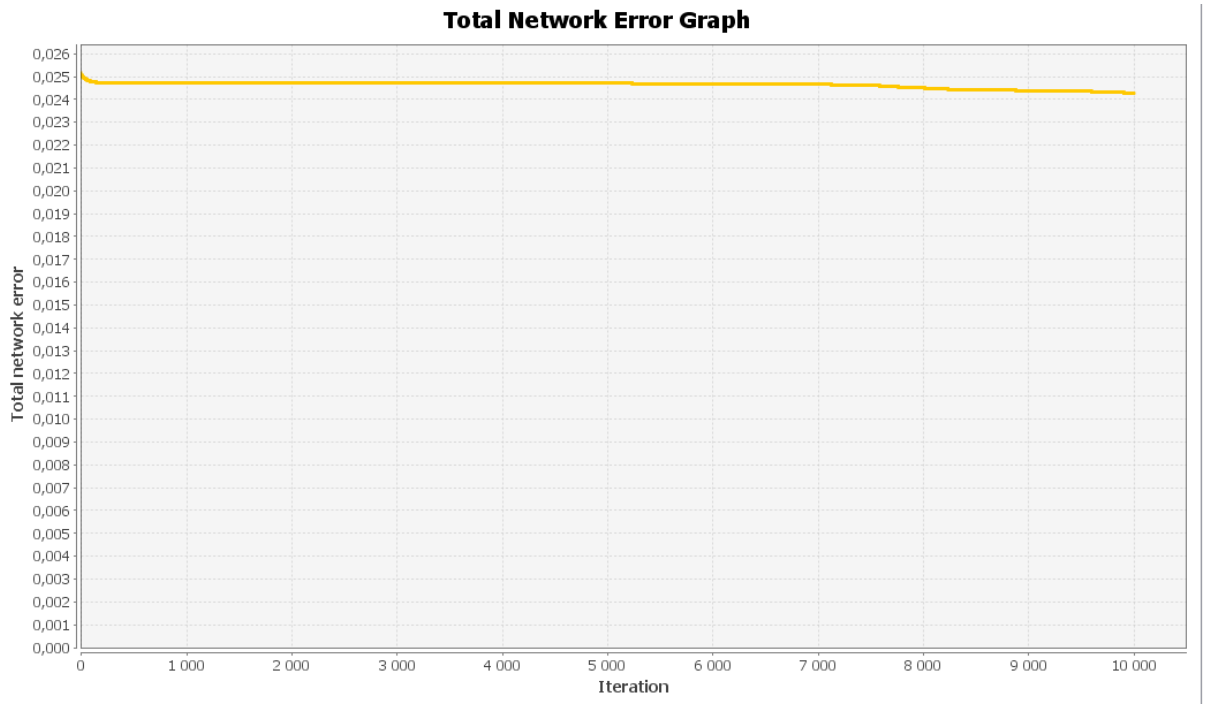


a)Wykres zależności błędu MSE od ilości iteracji dla learn rate = 0,05



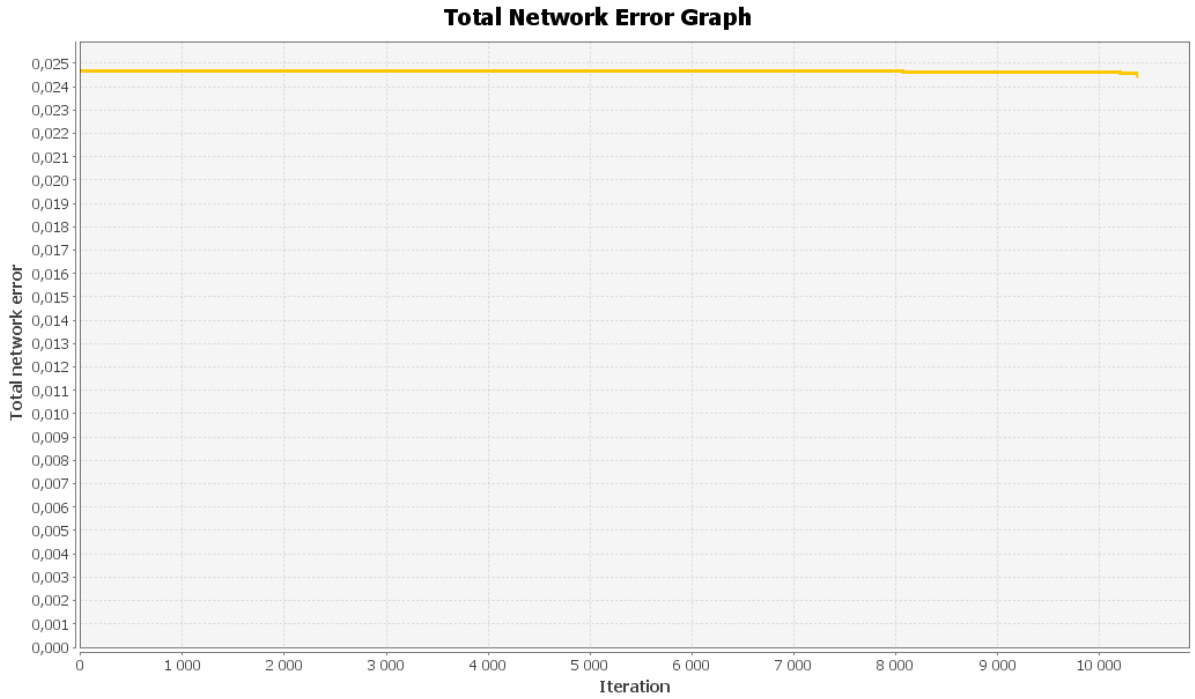
Widzimy, że w tej kofiguracji sieć po ponad 10 000 iteracji nie wykazywała znacznego spadku błędu. Utrzymywał się on na niemalże stałym poziomie.

b)Wykres zależności błędu MSE od ilości iteracji dla learn rate = 0,1



Tutaj obserwujemy podobną sytuacje…

c) Wykres zależności błędu MSE od ilości iteracji dla learn rate = 0,5



W tym przypadku również to samo….

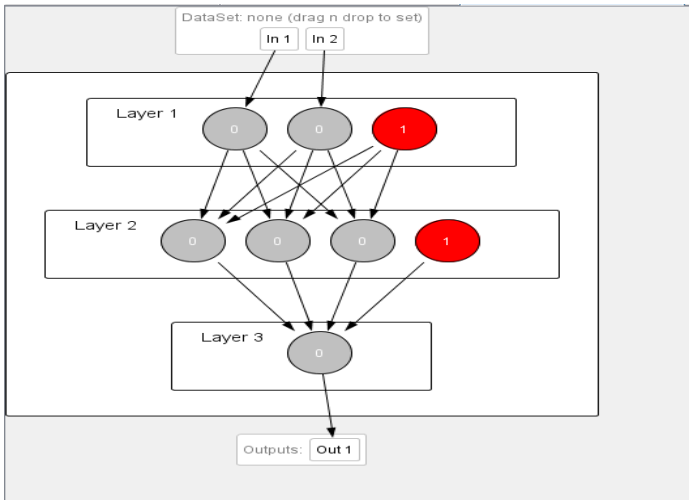
Dodatkowo czas wykonania 10 000 iteracji był znacznie większy niż w poprzednim wariancie co oczywiście wiąże się z większa ilością neuronów składowych.

Testując sieci na innych danych, uzyskano błędy MSE:

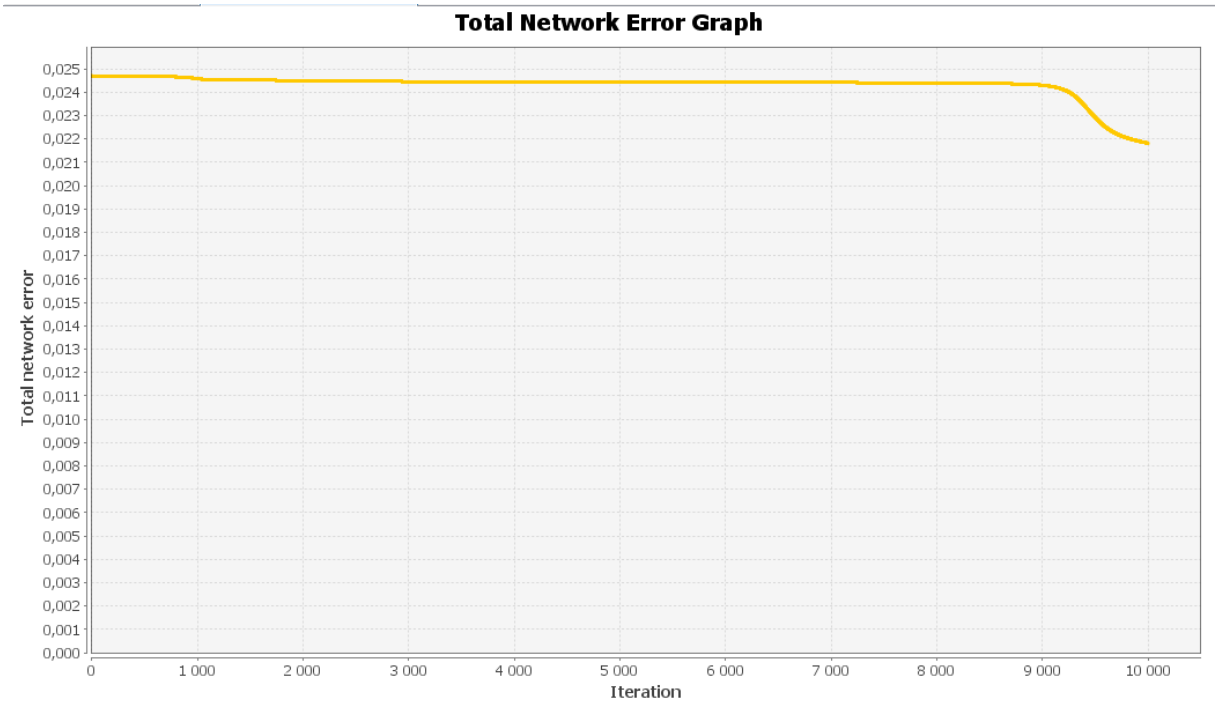
LR	MSE
0,05	0,049776
0,1	0,049702
0,5	0,049682

Tutaj także można zauważyć, że błąd MSE przy danych testujących był około 2 razy większy niż w przypadku uczenia gdzie błąd dochodził do wartości 0,024.

Konfiguracja 3.

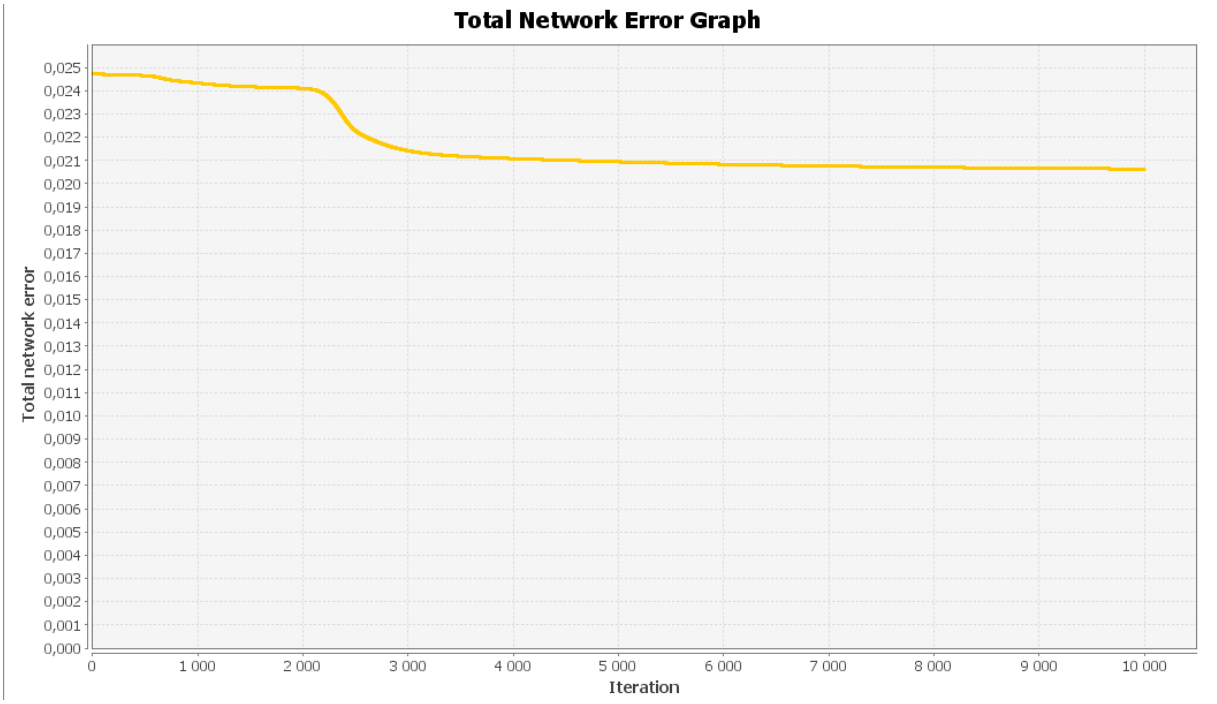


a)Wykres zależności błędu MSE od ilości iteracji dla learn rate = 0,05



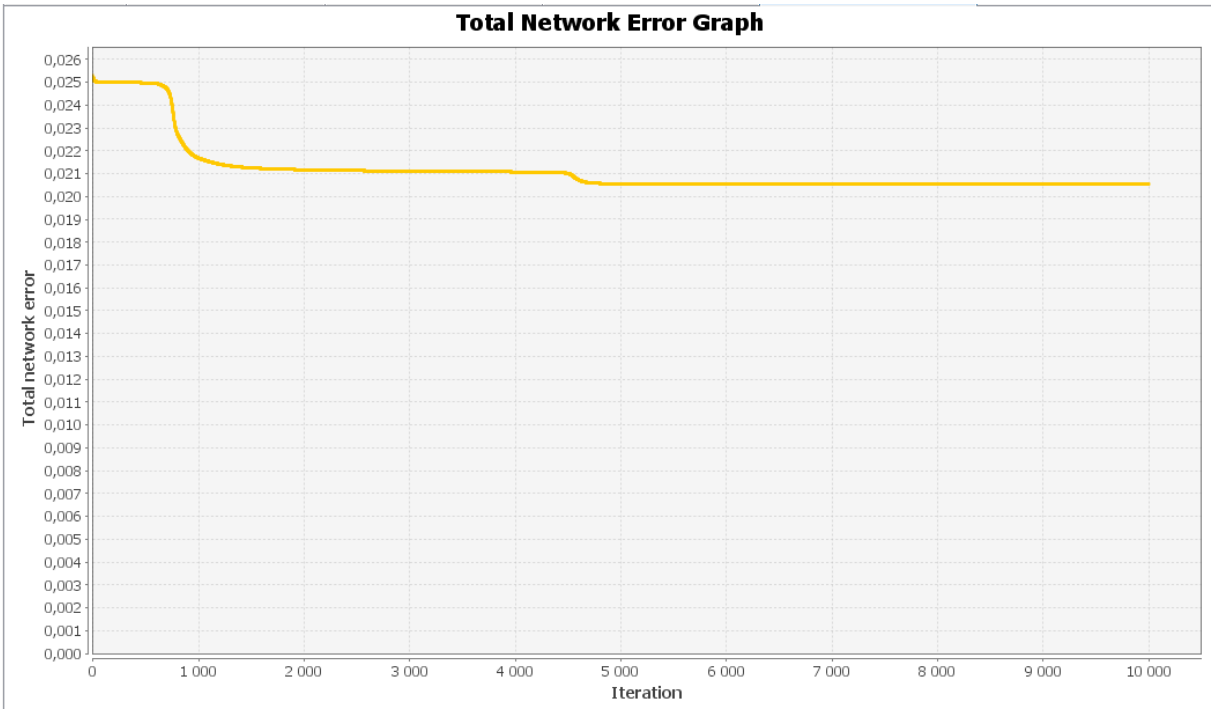
W tej konfiguracji zauważamy pod koniec uczenia niewielki spadek, jednak niewielki.

b) Wykres zależności błędu MSE od ilości iteracji dla learn rate = 0,1



Przy zwiększeniu learn rate spadek możemy zaobserwować dużo wcześniej jednak sieć nie jest w stanie już bardziej zmniejszyć błędu.

c) Wykres zależności błędu MSE od ilości iteracji dla learn rate = 0,5



Przy kolejnym zwiększeniu learn rate, spadek widzimy znowu jeszcze wcześniej, jednak sieć tak samo nie jest w stanie zmniejszyć błędu do pożądanej wartości.

Testując sieci na innych danych, uzyskano błędy MSE:

LR	MSE
0,05	0,045869
0,1	0,042376
0,5	0,043067

Podobnie jak w poprzednich przypadkach błąd MSE dla danych testujących był około 2 razy większy niż w przypadku danych uczących, który osiągał wartość około 0,021.

5. Wnioski.

Na podstawie otrzymanych wyników przy wybranych konfiguracjach można dojść do wniosku, że bardzo ważne jest w odpowiedni sposób dobranie ilości warstw i neuronów w każdej z nich. Przy zbyt małej liczbie, sieć nie jest w stanie nauczyć się odpowiednio dokładnie. Uczy się ona zbyt dokładnie co wpływa na fakt, iż myli się ona na niewystarczająco zadowalającym poziomie. W przypadku zaś zbyt dużej ilości warstw i ilości neuronów w warstwach, dochodzi do podobnej sytuacji, z tą różnicą, że sieć próbuje się nauczyć odwzorowania funkcji zbyt dokładnie co zajmuje za duże ilości czasu.

Na efektywność uczenia duży wpływ miał także learn rate. Ważne jest aby, podobnie jak ilość warstw i neuronów nie był ani za mały, ani za duży, gdyż albo będzie za długo dochodził do rozwiązania, albo go „przeskoczy”.

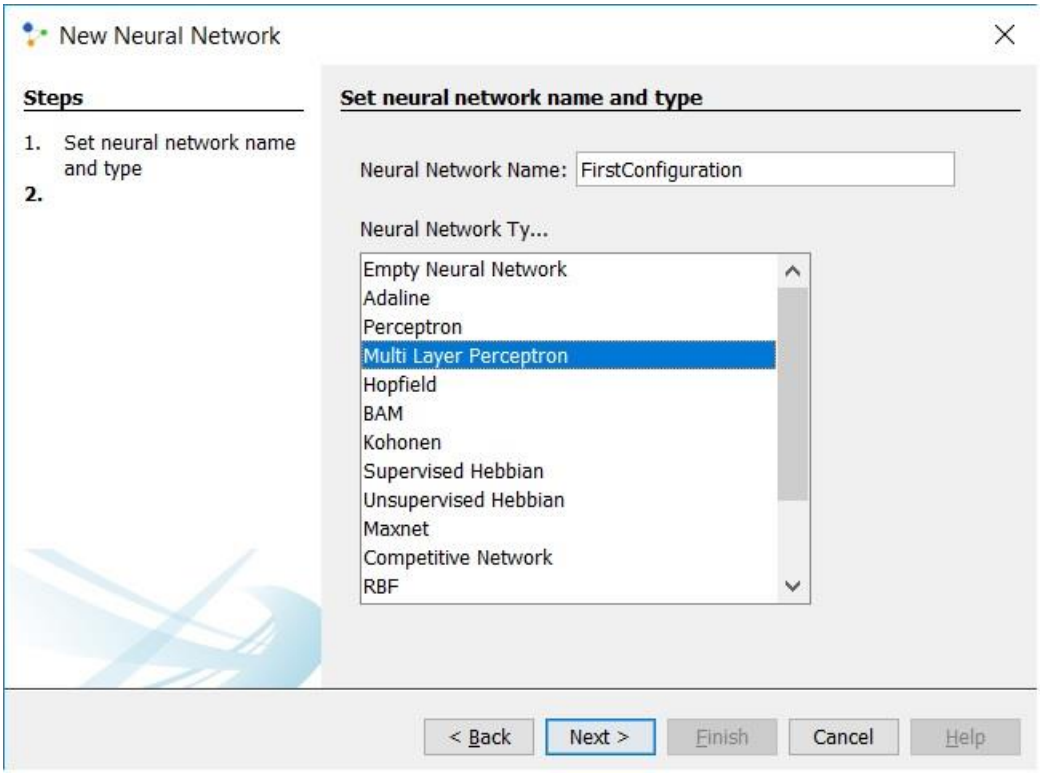
Trudności z uczeniem mogą wynikać także z faktu skomplikowania funkcji Rastrigin 3D, w której występuje dużo minimów lokalnych, co może wpływać negatywnie na działanie algorytmu uczenia.

Ciekawym faktem jest, że testując sieć na innych danych niż przy uczeniu, sieć myli się około 2 razy częściej.

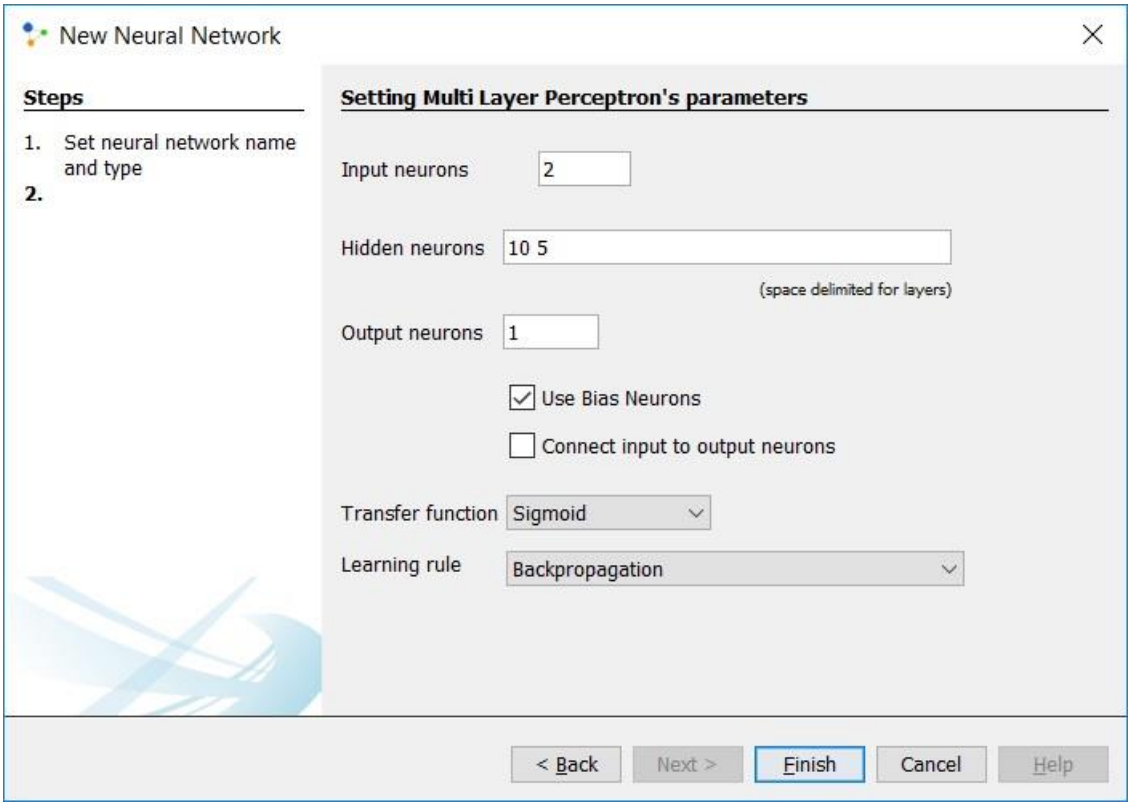
6. Listing kodu generowania danych uczących i zrzuty konfiguracji programu NeurophStudio

```java
/*
 * Copyright (C) 2017 Szysz
 */
package main;

import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.concurrent.ThreadLocalRandom;

/**
 *
 * @author Szysz
 */
public class RastriginDataGenerator {

    public void generateNormalizedData(String fileName, int size) throws IOException {

        double x[] = new double[size];
        double y[] = new double[size];
        double z[] = new double[size];

        File file = new File(fileName); // tworzymy nowy plik o podanej nazwie
        if (!file.exists()) { // jeśli plik nie istnieje należy go stworzyć
            file.createNewFile();
        }

        PrintWriter printWriter = new PrintWriter(file); //writer dzieki ktoremu bedziemy zapisywac dane uczace do pliku
        for (int i = 0; i < size; i++) {
            x[i] = ThreadLocalRandom.current().nextDouble(-2, 2); // losowe wartosci od -2 do 2
            y[i] = ThreadLocalRandom.current().nextDouble(-2, 2);
            z[i] = rastriginFunction(x[i], y[i]); // wartosci funkcji rastrigin3d dla wylosowanych punktów

            normalizeData(x, y, z); //normalizacja danych do przedziali 0 do 1

            printWriter.println(x[i] + "," + y[i] + "," + z[i]); //wpisz dane do pliku
        }
        printWriter.close(); // pamietamy o zamknieciu writera
    }

    private double rastriginFunction(double x, double y) { //funckja rastrigin3d
        return 20 + Math.pow(x, 2) - 10 * Math.cos(2 * Math.PI * x) + Math.pow(y, 2) - 10 * Math.cos(2 * Math.PI * y);
    }

    private void normalizeData(double x[], double y[], double z[]) { // funkcja normalizujaca dane

        for (int i = 0; i < x.length; i++) {
            x[i] = (x[i] + 2.0) / 4.0;
            y[i] = (y[i] + 2.0) / 4.0;
            z[i] = z[i] / 44.5;
        }
    }

```

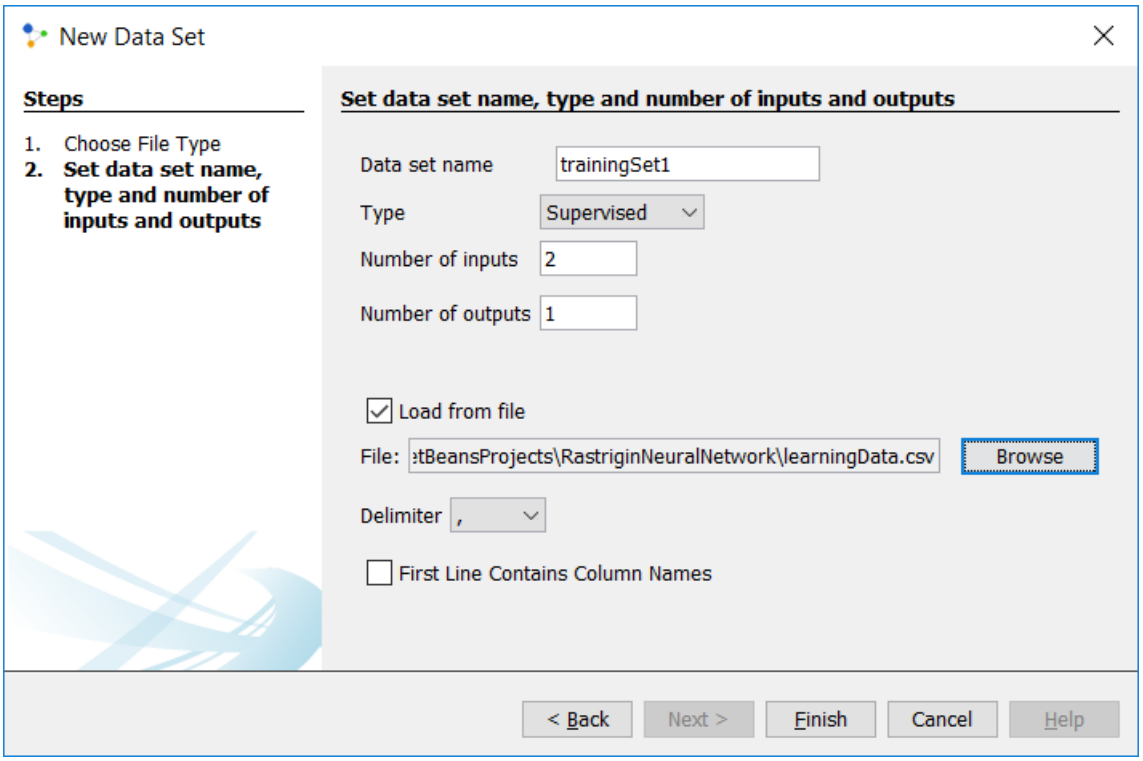
```java
/**
 *
 * @author Szysz
 */
public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) throws InterruptedException {

        RastriginDataGenerator generator = new RastriginDataGenerator(); //tworzymy generator
        try {
            generator.generateNormalizedData("learningData.csv", 3500); //generujemy znormalizowane dane uczace
            generator.generateNormalizedData("testingData.csv", 1500); //generujemy znormalizowane dane testujace
        } catch (IOException ex) {
            System.err.println("IOException cought!");
        }
        finally{
            System.out.println("Generation completed!");
        }
    }

}
```



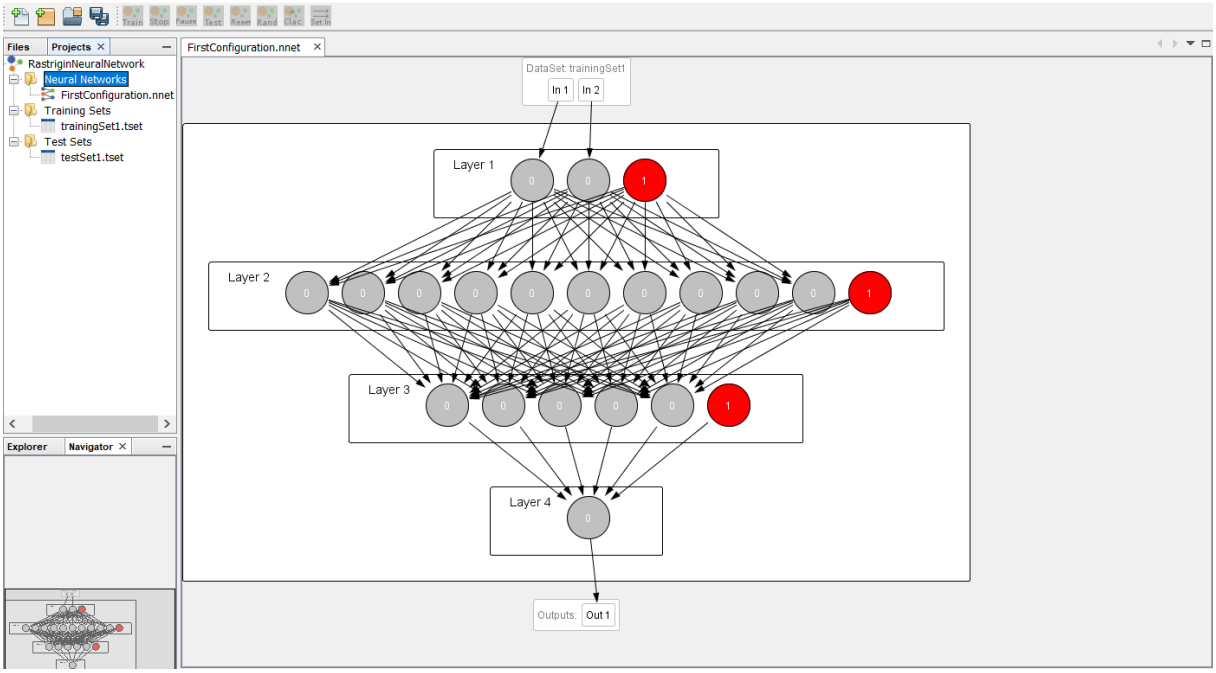
Po uruchomieniu programu tworzymy nowy projekt a następnie tworzymy nowy plik, wybieramy New Neural Network i mamy kreator tworzenia sieci, jak powyżej. W pierwszym kroku wpisujemy nazwę sieci oraz wybieramy jej typ.



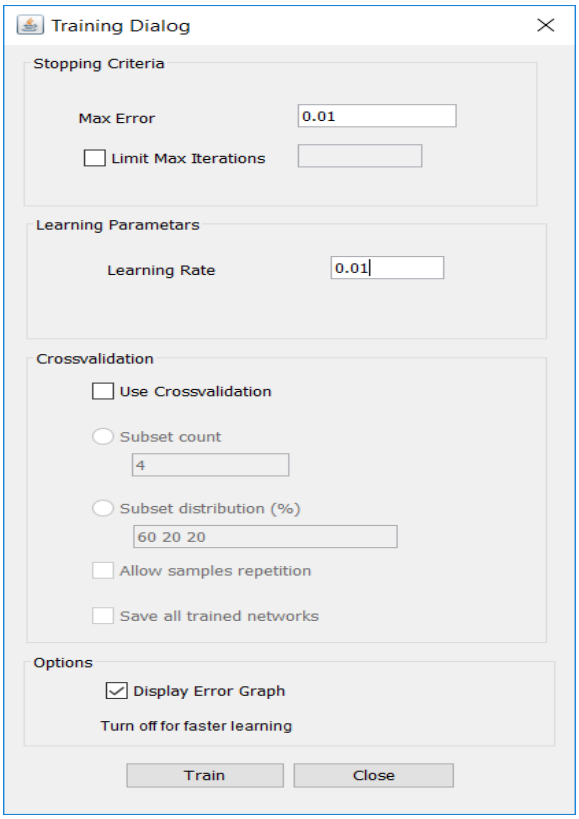
W kolejnym kroku wpisujemy ilość neuronów wejściowych, ukrytych (oddzielone spacjami, jeśli więcej niż jedna) i wyjściowych. Dodatkowo wybieramy czy mają występować neurony z biasem, typ funkcji aktywacji oraz algorytm uczenia.



Musimy także dodać dane uczące poprzez kreator, gdzie wpisujemy jego nazwe, typ(z nauczycielem czy bez), ilość wejść i wyjść. Dodatkowo możemy załadować te dane z pliku.



Po wykonaniu powyższych czynności otrzymujemy sieć do których możemy załadować nasze dane uczące i przystąpić do uczenia i testowania.



Po kliknięciu w górnym menu „train” wpisujemy learning rate, błąd uczenia przy jakim kończymy uczenia oraz ewentualnie maksymalną liczbę iteracji i akceptujemy przyciskiem „Train”

8. Bibliografia.

https://en.wikipedia.org/wiki/Rastrigin_function

http://galaxy.agh.edu.pl/~vlsi/AI/backp_t/backprop.html

http://neuroph.sourceforge.net/

Stanisław Osowski – Sieci neuronowe do przetwarzania informacji, ISBN 83-7207-615-4

http://zsi.ii.us.edu.pl/~nowak/ed/cw4.pdf

https://www.hackerearth.com/practice/machine-learning/machine-learning-algorithms/understanding-deep-learning-parameter-tuning-with-mxnet-h2o-package-in-r/tutorial/