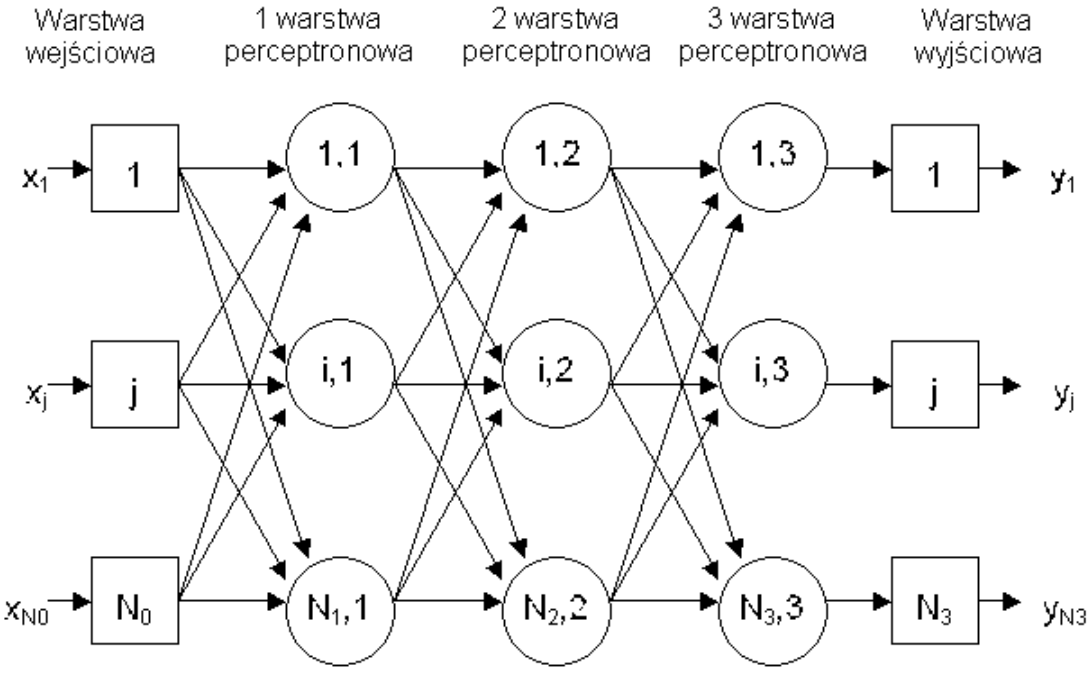
Daniel Wenecki
Podstawy Sztucznej Inteligencji
Sprawozdanie – Projekt nr 2

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 użytej sieci I algorytmu uczenia

Schemat struktury sieci neuronowej z trzema warstwami perceptronowymi oraz warstwą wejściową i wyjściową:



Typowa struktura sieci typu MLP, nazywana siecią w pełni połączoną, 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. Jedyną funkcją warstwy wejściowej jest przesyłanie i rozprowadzanie sygnałów do pierwszej warstwy ukrytej. Jej neurony nie podlegają procesowi uczenia, nie posiadają wag, w których mogłaby być gromadzona wiedza o przybliżanych zależnościach i nie biorą udziału w procesie generalizacji. Następnie sygnał przesyłany jest na wejścia pierwszej warstwy ukrytej, która przetwarza dane i generuje sygnał wyjściowy podawany na wejścia warstwy kolejnej. Powyższy 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. Warstwy ukryte oraz warstwa wyjściowa składają się z tego samego rodzaju neuronów, podlegających procesowi uczenia i na wzór komórki nerwowej, gromadzących wiedzę o przybliżanych zależnościach w wektorze wag.

W sieci użyto sigmoidalnej funkcji aktywacji. Sigmoida ma postać:

$$\phi(s) = \sigma(s) = \frac{1}{1 + exp(-s)},$$

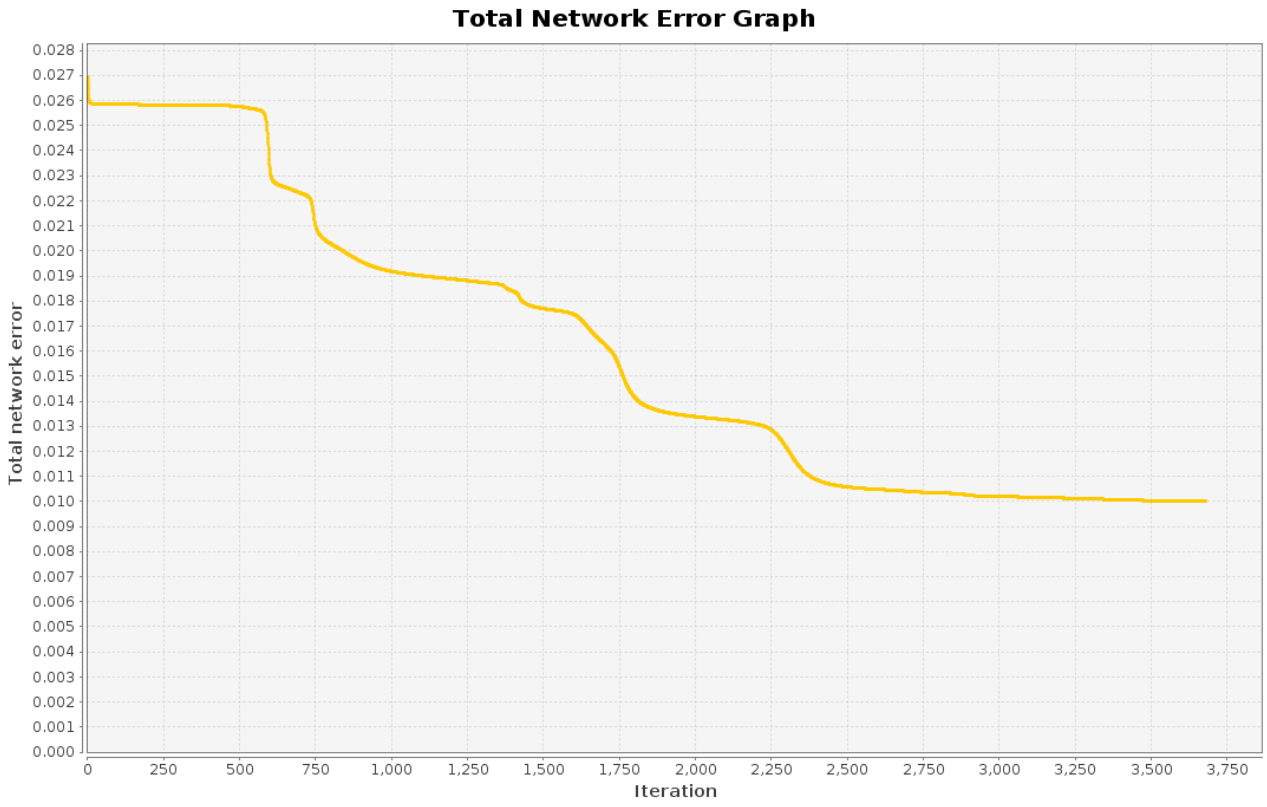
Do aktualizowania wag użyto algorytmu wstacznej propagacji, w której aktualizacja wag przebiega w następujący sposób:

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = -\eta o_i \delta_j$$

$$\delta_j = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial \text{net}_j} = \begin{cases} (o_j - t_j) o_j (1 - o_j) & \text{if } j \text{ is an output neuron,} \\ (\sum_{\ell \in L} \delta_\ell w_{j\ell}) o_j (1 - o_j) & \text{if } j \text{ is an inner neuron.} \end{cases}$$

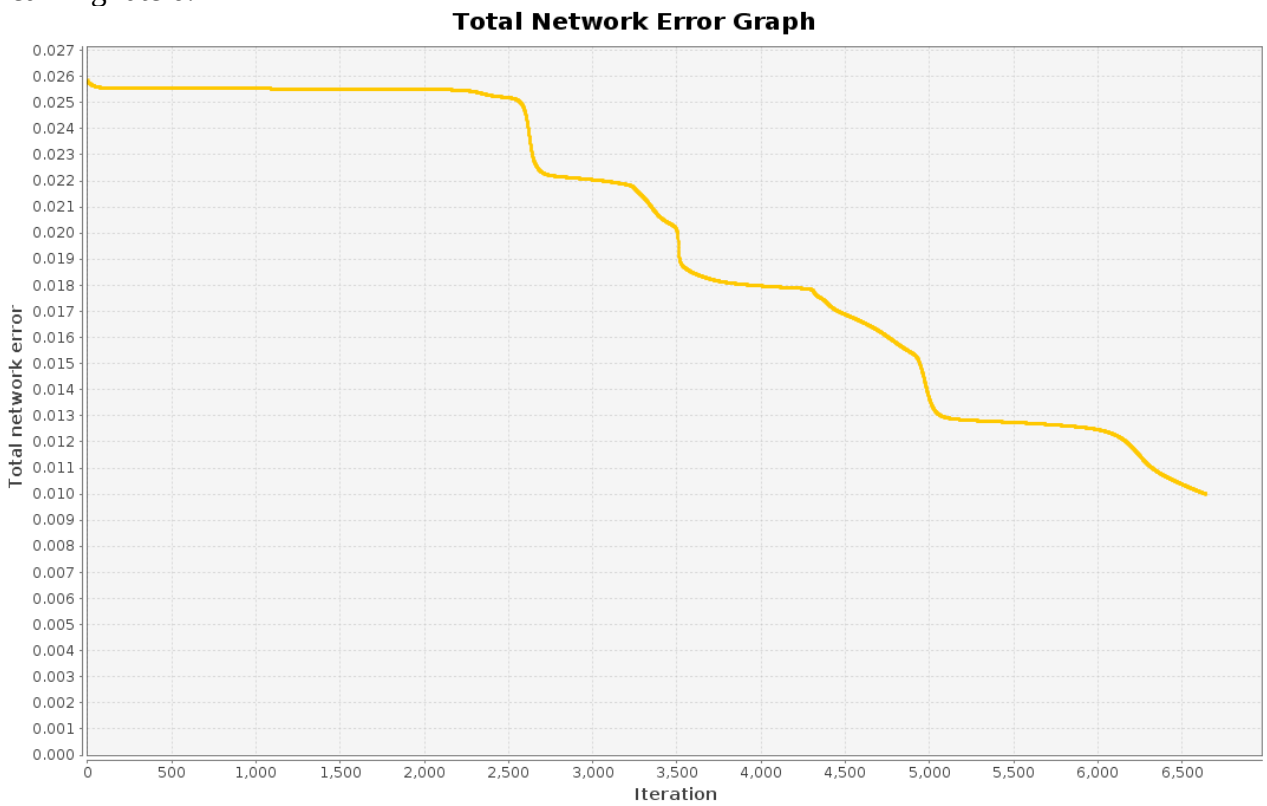
2) Zestawienie otrzymanych wyników

Dane otrzymane dla sieci MLP 10 10, dla różnych współczynników uczenia.
Learning rate 0.5



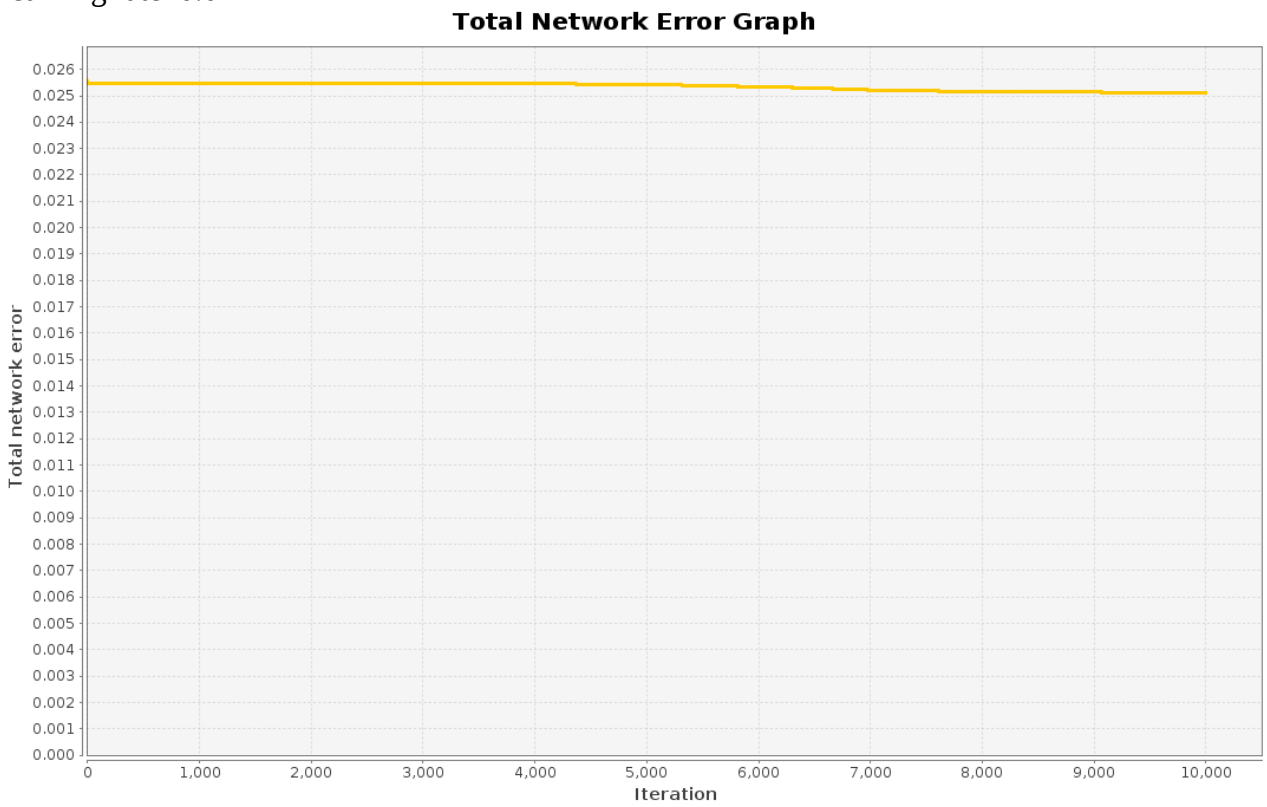
Total Mean Square Error: 0.020455174067925387

Learning rate 0.1



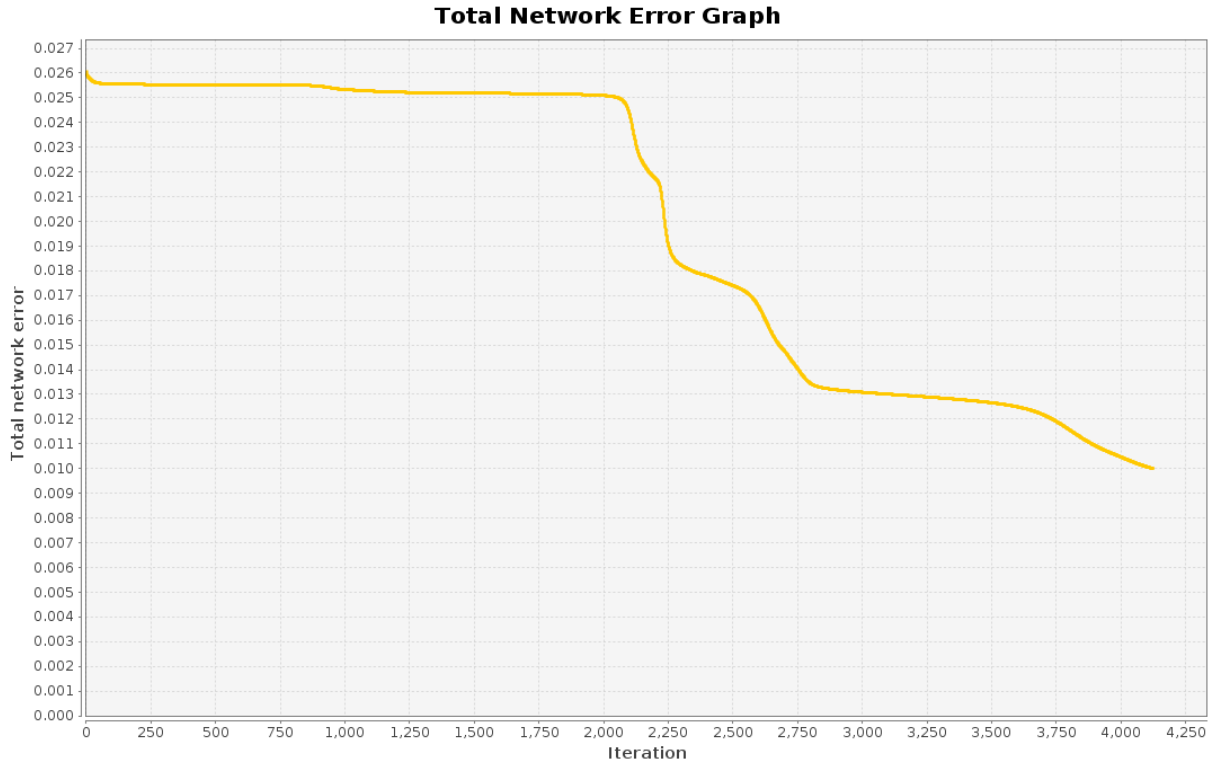
Total Mean Square Error: 0.020265046552216337

Learning rate 0.01

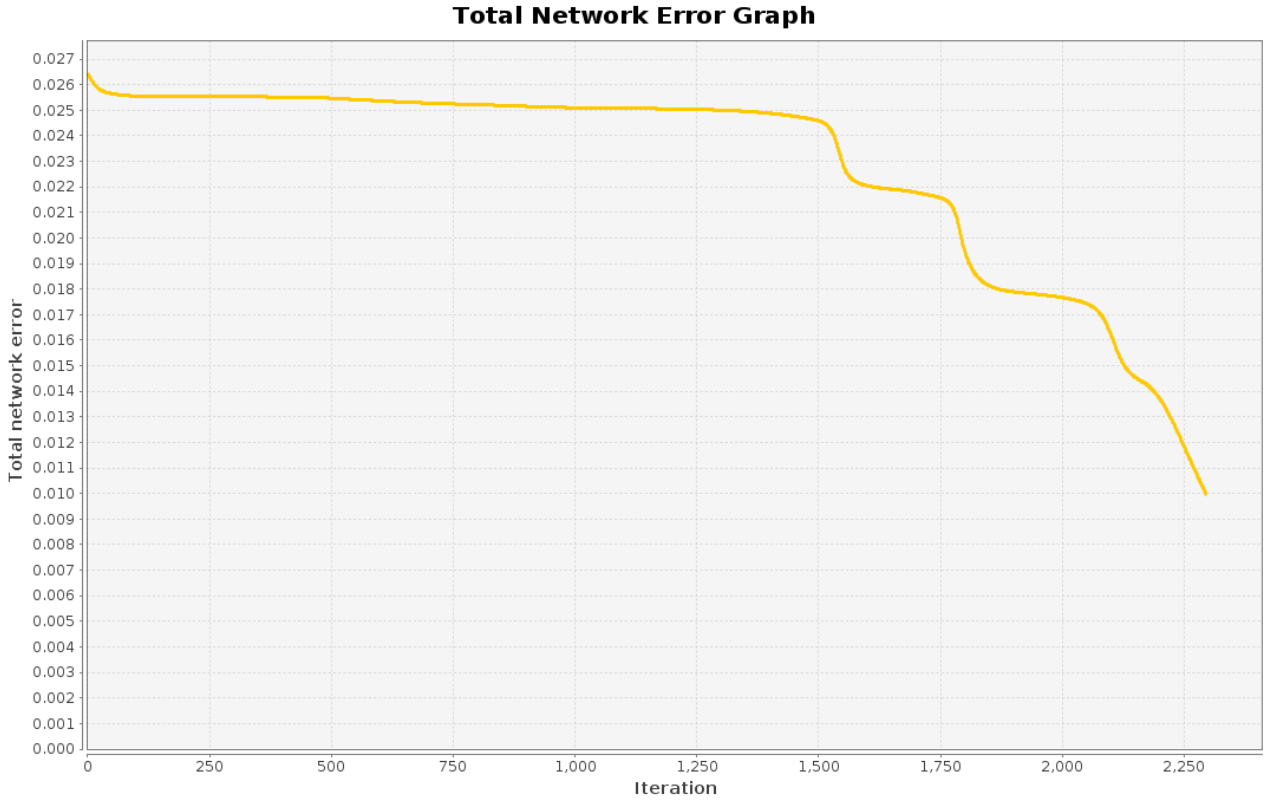


Total Mean Square Error: 0.05081374770891593

Dane otrzymane dla różnych struktur sieci MLP i współczynnika uczenia równego 0.1
Sieć 20 20

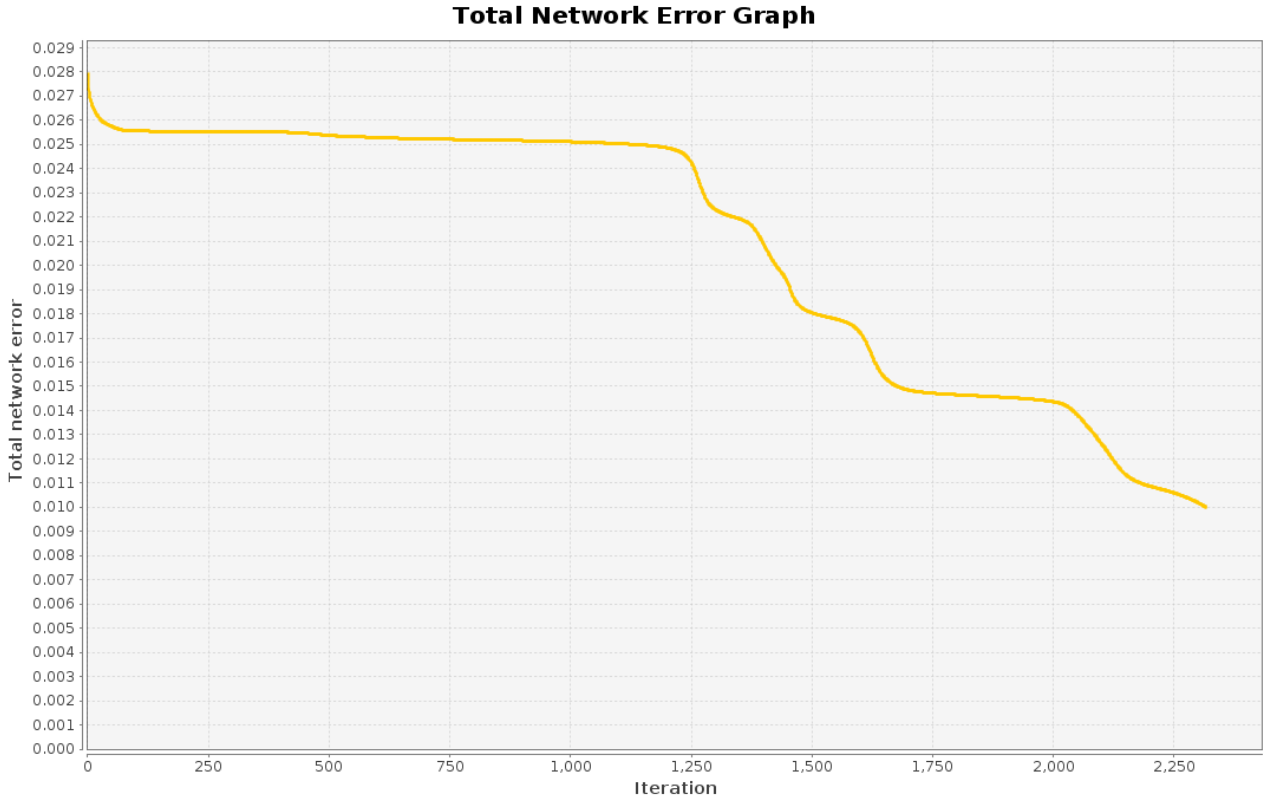


Total Mean Square Error: 0.019831579498332055
Sieć 30 30



Total Mean Square Error: 0.01840591086324019

Sieć 50 50



Total Mean Square Error: 0.01926221002198792

3) Analiza i dyskusja błędów uczenia i testowania opracowanej sieci w zależności od wartości współczynnika uczenia oraz ilości warstw i neuronów,

Sieć dla learning rate 0.01 osiągneła limit 10000 iteracji.

Z uzyskanych wyników możemy zauważyć wyraźny wpływ wartości współczynnika uczenia od ilości iteracji potrzebnych przez sieć do nauczenia się sieci. Struktura sieci także zdaje się mieć wpływ na ilość potrzebnych iteracji.

Średni błąd kwadratowy zmienia się wraz z strukturą. Najmniejszy jest dla środkowej ilości przyjętych przez nas neuronów, a rośnie wraz z malejącą lub zwiększającą się ilością neuronów w sieci.

4) Wnioski

Zwiększająca się ilość iteracji wraz z mniejszym współczynnikiem uczenia jest związana, tak jak w poprzednich scenariuszach, z mniejszą zmianą wag. Ustawienie tej wielkości zbyt małej spowoduje długie szukanie wag, natomiast ustawienie tej wartości zbyt dużej może spowodować, że sieć się nigdy nie nauczy.

Struktura sieci ma wpływ na ilość iteracji, ponieważ w przypadku małej ilości iteracji stopień wielomianu jaki sieć może odwzorować nie jest zbyt duży w związku z tym znalezienie odpowiedniego położenia interpolowanej funkcji trwa dłużej. Należy jednak wziąć pod uwagę, że w przypadku wiekszych sieci, ilość obliczeń na iteracje też jest większy, co oznacza, że mniejsza ilość iteracji niekoniecznie oznacza szybsze otrzymanie poszukiwanego rozwiązania.

Zmieniającą się w wyżej opisany sposób, średnią wartość kwadratową błedu, można wytłumaczyć tym w jaki sposób sieć o określonej ilości neuronów może interpolować funkcje. W przypadku gdy tych neuronów jest mało, sieć będzie tworzyłą funkcje o dość małym stopniu wielomianu. Oznacza to że nie jest w stanie dobrze odwzorować bardziej złożonych funkcji. Natomiast jeśli neuronów jest za dużo, to sieć będzie mogła stworzyć wielomian o zbyt dużym stopniu, co oznacza, że mimo że wartość błędów dla danych uczących będzie w stanie zejść do bardzo niskich wartości, to dane testowania będą generowałly duże błędy.

5) Listing kodu i konfiguracja programu

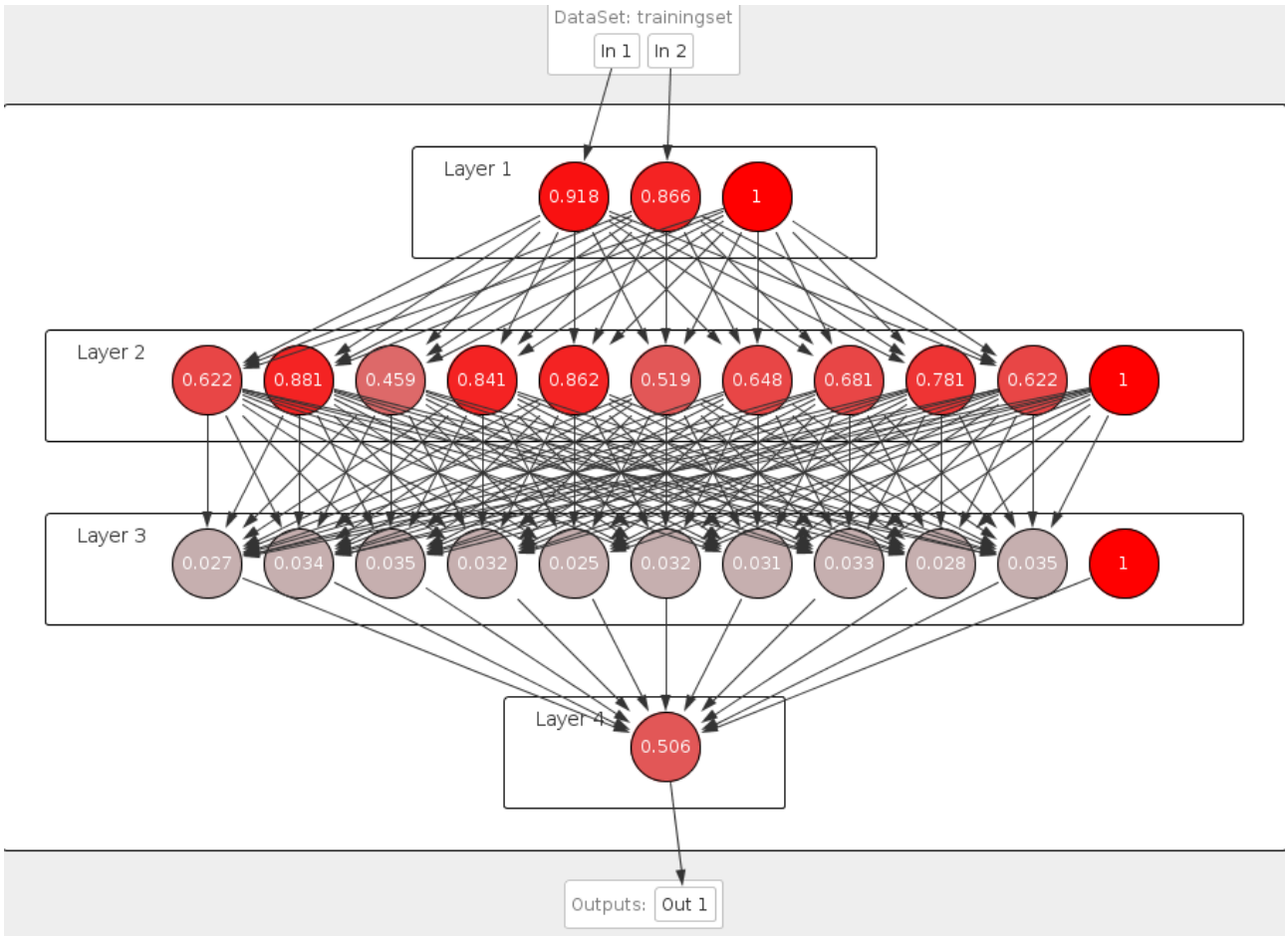
Funkcja generująca dane do uczenia i testowania

```
public class Main {
    private static double calculateRastrigin3D( double x1, double x2 )
//obliczanie funkcji rastrigin
    {
        return 20.0
                + x1 * x1 - 10.0 * Math.cos(2.0 * Math.PI * x1)
                + x2 * x2 - 10.0 * Math.cos(2.0 * Math.PI * x2);
    }
    public static void main( String[] args ) throws FileNotFoundException,
UnsupportedEncodingException {
        Random randomizer = new Random();
        List<double[]> trainingData = new ArrayList<>( 3500 ); // uczące
        List<double[]> testingData = new ArrayList<>( 1500 ); // testujące
        for( int i = 0; i < 3500; ++i ) // 7/10 * 5000
        {
            double x1 = 4.0 * randomizer.nextDouble() - 2.0;
            double x2 = 4.0 * randomizer.nextDouble() - 2.0;
            double y = calculateRastrigin3D( x1, x2 );
            x1 = (x1+2.0)/4.0;//normalizacja
            x2 = (x2+2.0)/4.0;
            y = y/44.5;
            double[] row = new double[] { x1, x2, y };
            trainingData.add( row );
        }
        for( int i = 0; i < 1500; ++i ) // 3/10 * 5000
        {
            double x1 = 4.0 * randomizer.nextDouble() - 2.0;
            double x2 = 4.0 * randomizer.nextDouble() - 2.0;
            double y = calculateRastrigin3D( x1, x2 );
            x1 = (x1+2.0)/4.0;//normalizacja
            x2 = (x2+2.0)/4.0;
            y = y/44.5;
            double[] row = new double[] { x1, x2, y };
            testingData.add( row );
        }
        //zapisywanie do plików w celu późniejszego odczytania przez program
        PrintWriter zapis = new PrintWriter("trainingData.txt");
        for( int i = 0; i < 3500; ++i ) // 7/10 * 5000
        {
            zapis.println(trainingData.get(i)[0]+ " " + trainingData.get(i)
[1] +" "+ trainingData.get(i)[2]);
        }
        zapis.close();
```

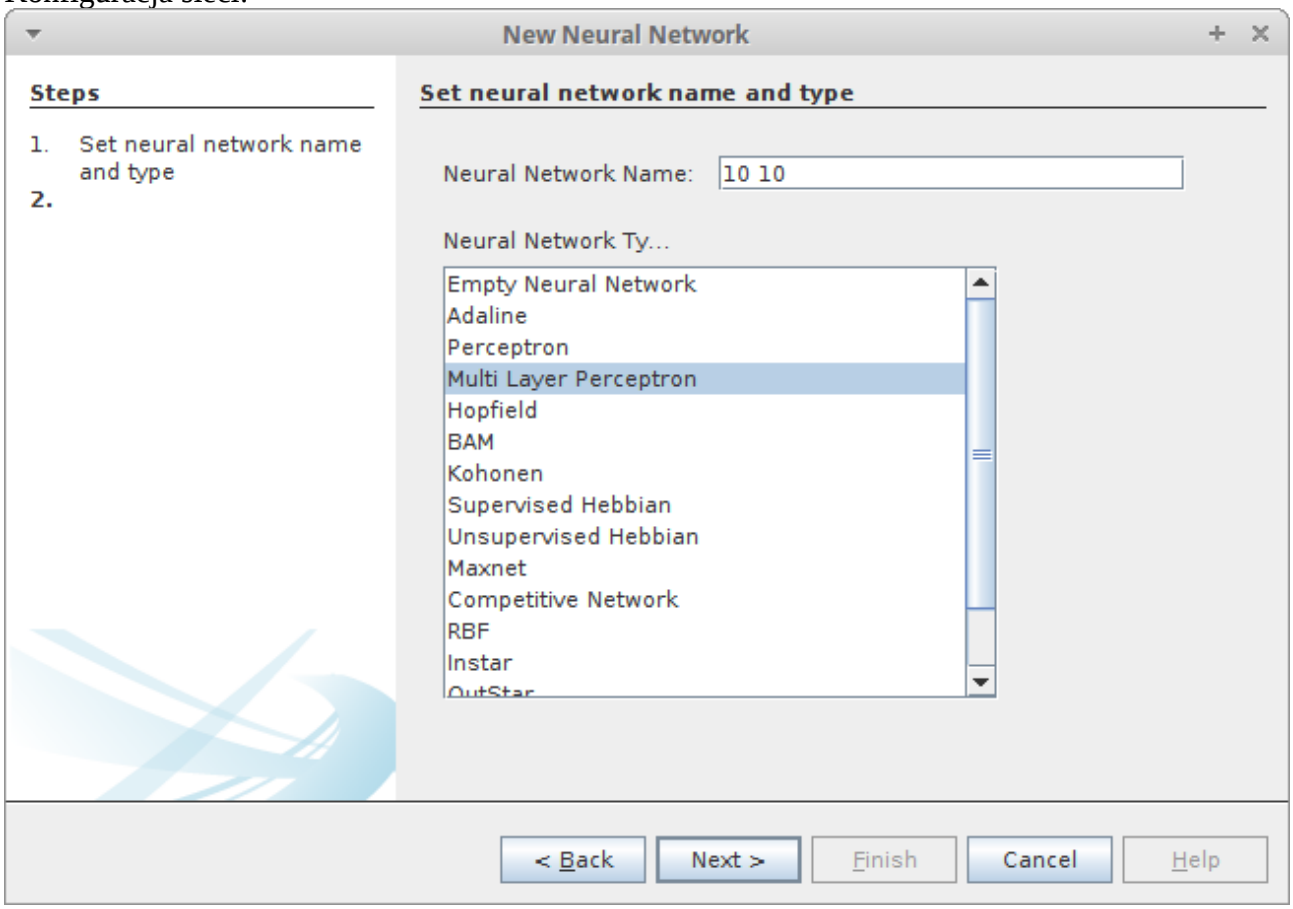
```
        PrintWriter zapis2 = new PrintWriter("testingData.txt");
        for( int i = 0; i < 1500; ++i ) // 3/10 * 5000
        {
            zapis2.println(testingData.get(i)[0]+ " " + testingData.get(i)
[1] +" "+ testingData.get(i)[2]);
        }
        zapis2.close();
    }
}
```

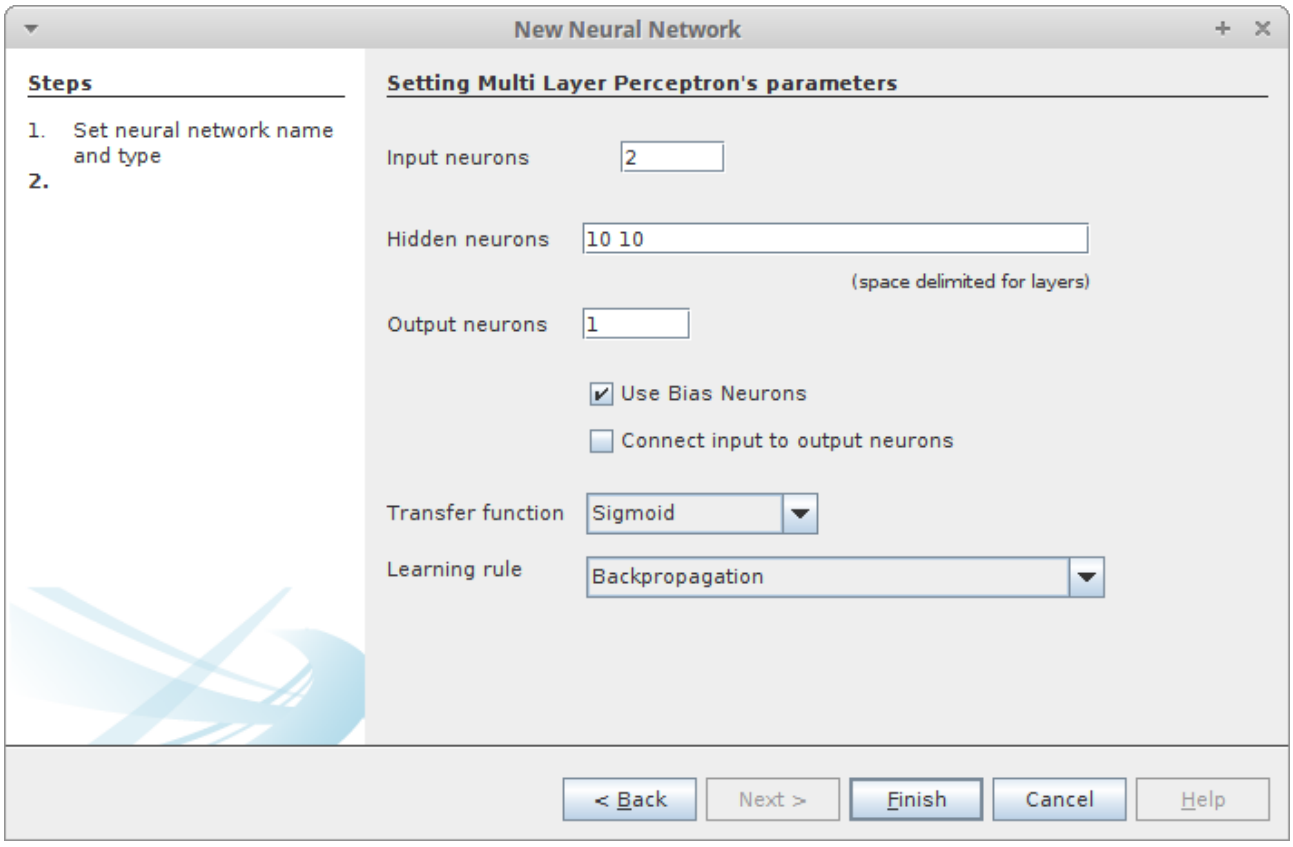
Do scenariusza użyto programu NeurophStudio.

Przykładowa sieć:

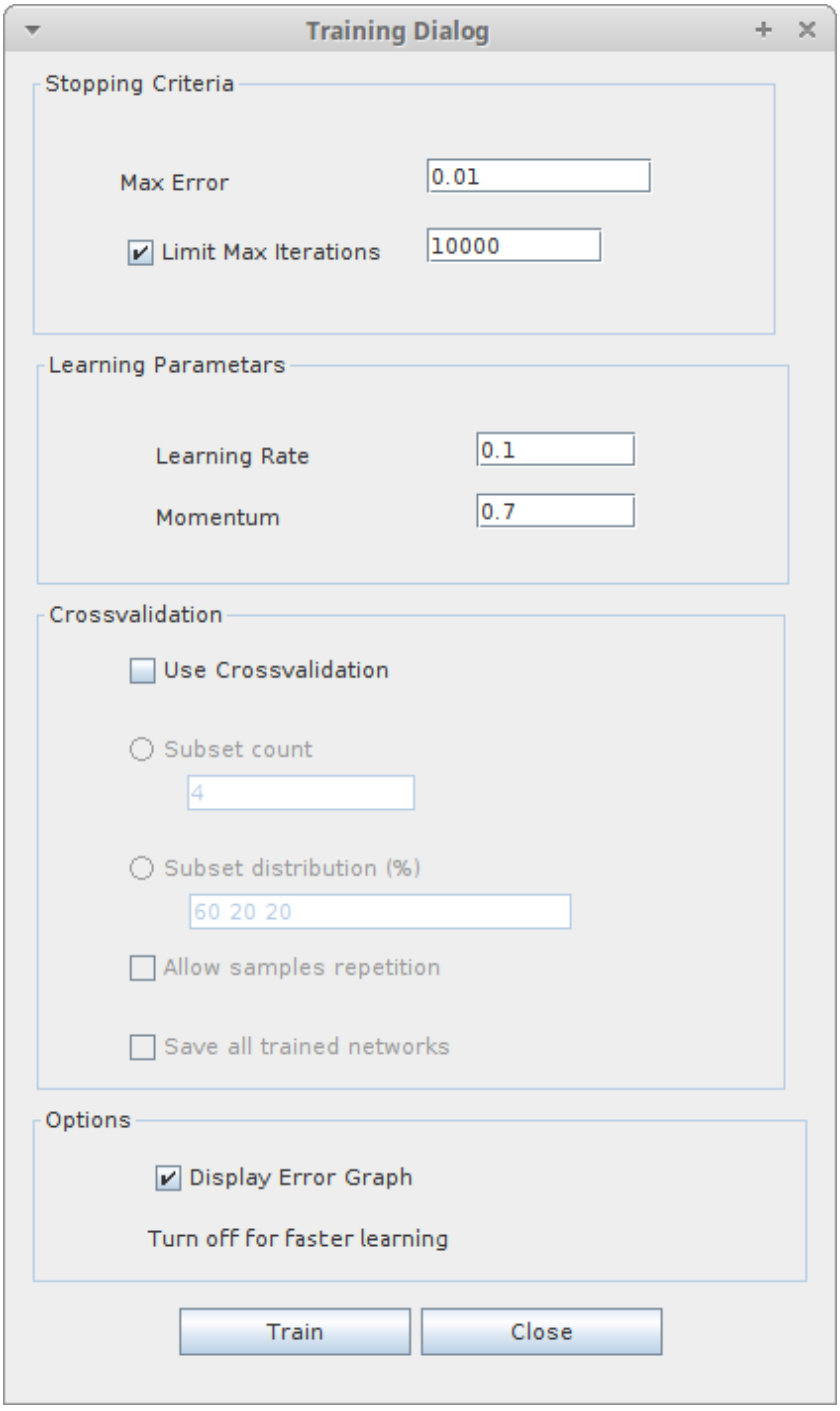


Konfiguracja sieci:





Trenowanie sieci:



Bibliografia:
https://en.wikipedia.org/wiki/Backpropagation
http://www.rosczak.com/mlp/mlp.html