

Celem ćwiczenia było poznanie budowy i działania wielowarstwowych sieci neuronowych poprzez uczenie kształtu wykresu funkcji matematycznej z użyciem algorytmu wstecznej propagacji błędów.

1) Syntetyczny opis budowy wykorzystanego algorytmu uczenia:

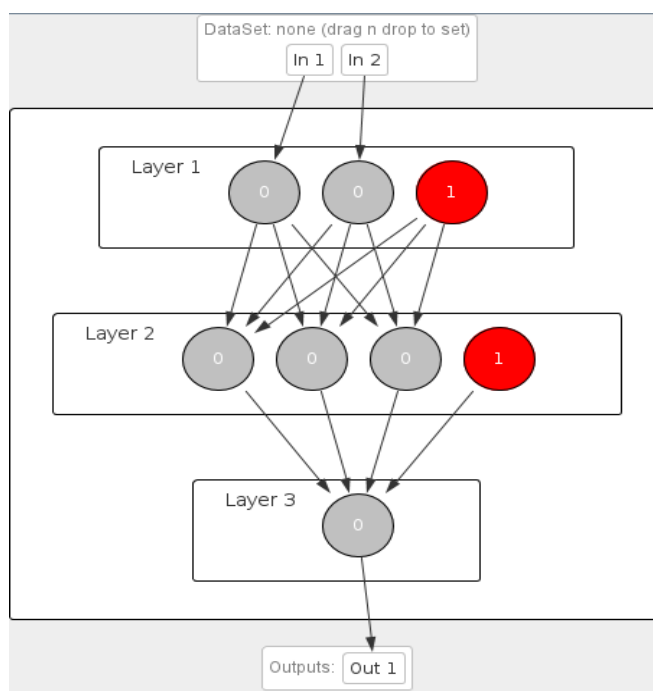
W celu wykonania ćwiczenia skorzystałem z narzędzia jakim jest *NeurophStudio*. Za pomocą kreatora stworzyłem sieci neuronowe, uczyłem je oraz testowałem.

Stworzyłem trzy sieci różniące się ilością warstw ukrytych oraz ilością neuronów w poszczególnych warstwach. Stworzone przeze mnie sieci składają się z perceptronów wykorzystujących sigmoidalną funkcję aktywacji oraz algorytm uczenia *Back Propagation*.

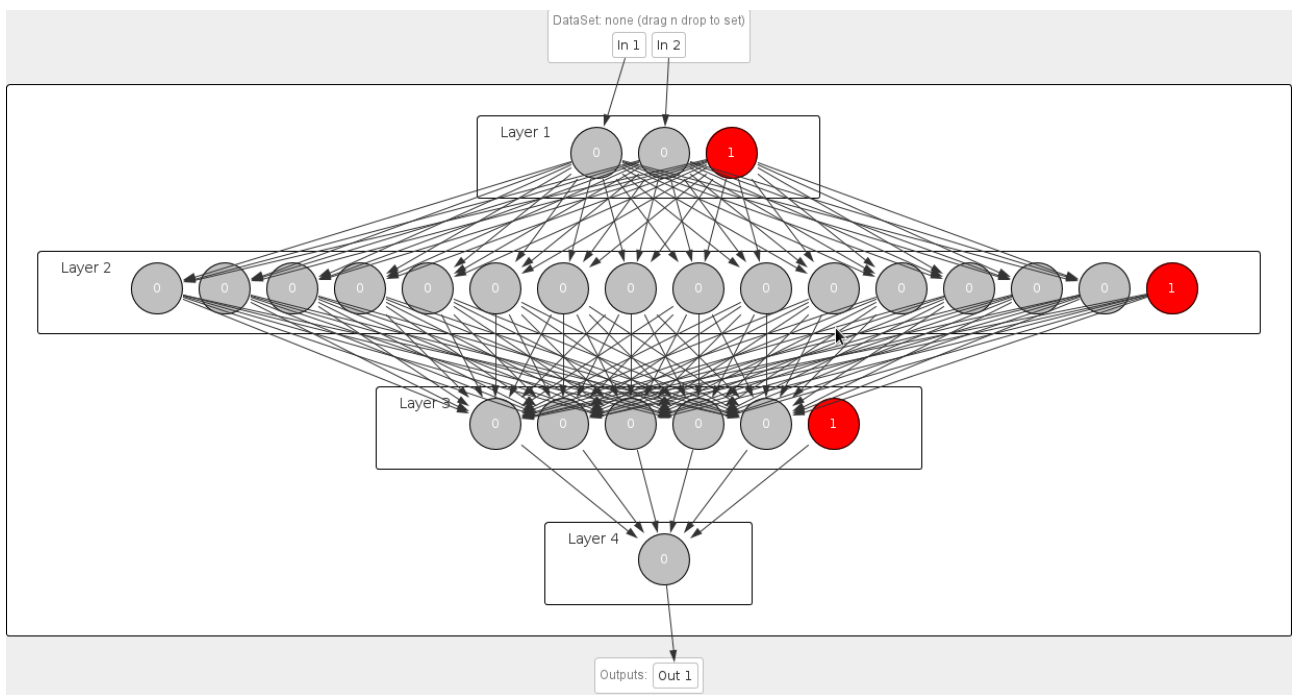
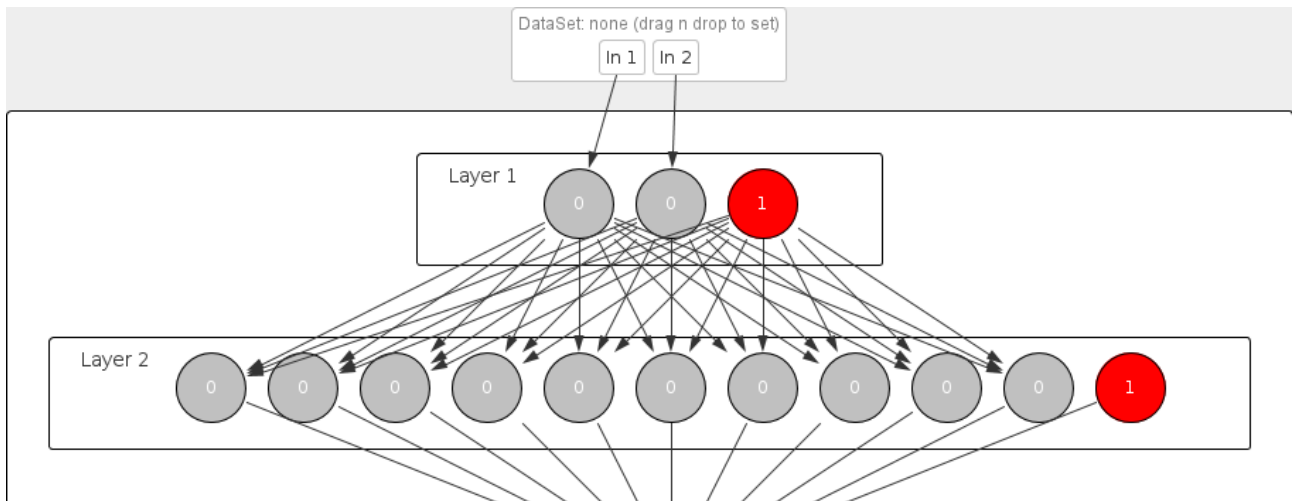
Sam algorytm Back Propagation można podzielić na 3 etapy:

- 1) Przepuszczenie sygnałów wejściowych bez procesu modyfikacji wag przez całą sieć celem uzyskania sygnału wyjściowego całej sieci.
- 2) Obliczenie wartości błędu $\delta = d - y$ dla ostatniej warstwy w sieci, gdzie „ d ” to wartość oczekiwana, a „ y ” to wartość otrzymana jako sygnał wyjściowy w pierwszym etapie, po czym rzutowanie tego błędu od tyłu, aż do samego początku sieci na każdy neuron.
- 3) Po otrzymaniu wartości błędu dla każdego neuronu następuje ponowne przepuszczenie przez całą sieć sygnałów wejściowych, jednak tym razem już z modyfikacją wag.

Poniżej przedstawiam schematy sieci wykorzystanych w ćwiczeniu:



Wariant nr 1 - schemat 2-3-1



Wariant nr 3 - schemat 2-15-5-1

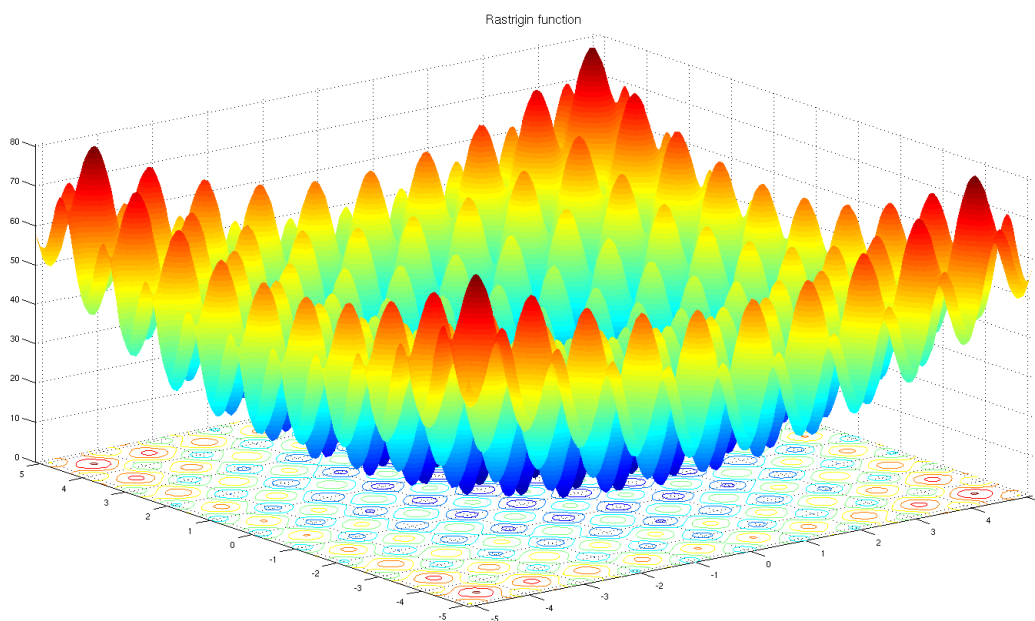
2) Zestawienie otrzymanych wyników:

Do wygenerowania danych uczących i testujących napisałem własny program, w którym funkcja zwracała wynik funkcji Rastrigin dla losowych punktów „x” oraz „y” z zakresu [-2; 2] .

Funkcja Rastrigin ma postać:

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

Jeżeli chodzi o reprezentację graficzną to funkcja ta prezentuje się w następujący sposób:



Reprezentacja graficzna funkcji Rastrigin

Wygenerowałem łącznie 5000 rekordów z czego 70% (3500 rekordów) z nich przeznaczyłem na uczenie sieci a pozostałe 30% (1500 rekordów) na testowanie. Otrzymane dane poddałem procesowi normalizacji, w celu dopasowania ich do sieci.

Normalizacja wykorzystuje następujący wzór:

$$x_{new} = (x_{old} - x_{min}) / (x_{max} - x_{min}) * (x_{newMax} - x_{newMin}) + x_{newMin}$$

gdzie:

x_{new} – nowa wartość x

x_{old} – stara wartość x

x_{min} – minimalna wartość x

x_{max} – maksymalna wartość x

x_{newMin} – nowa minimalna wartość

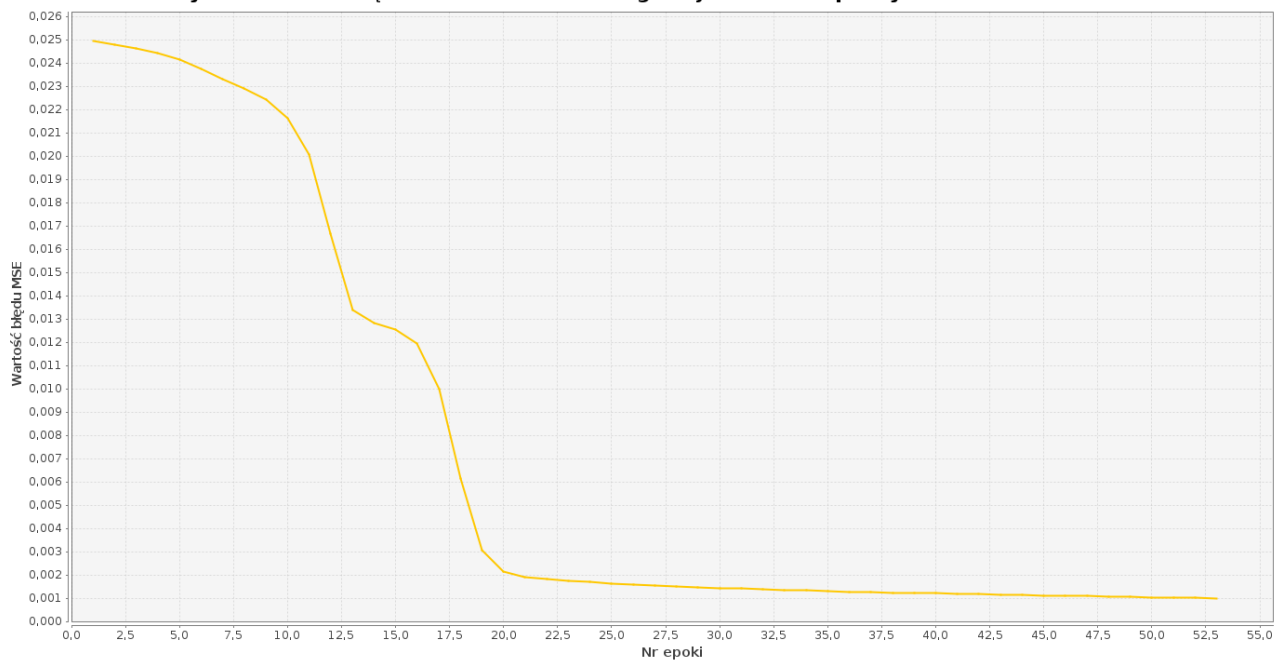
x_{newMax} – nowa maksymalna wartość

Wygenerowane dane zostały zapisane w dwóch osobnych plikach typu csv, by mogły zostać wykorzystane w programie *NeurophStudio*.

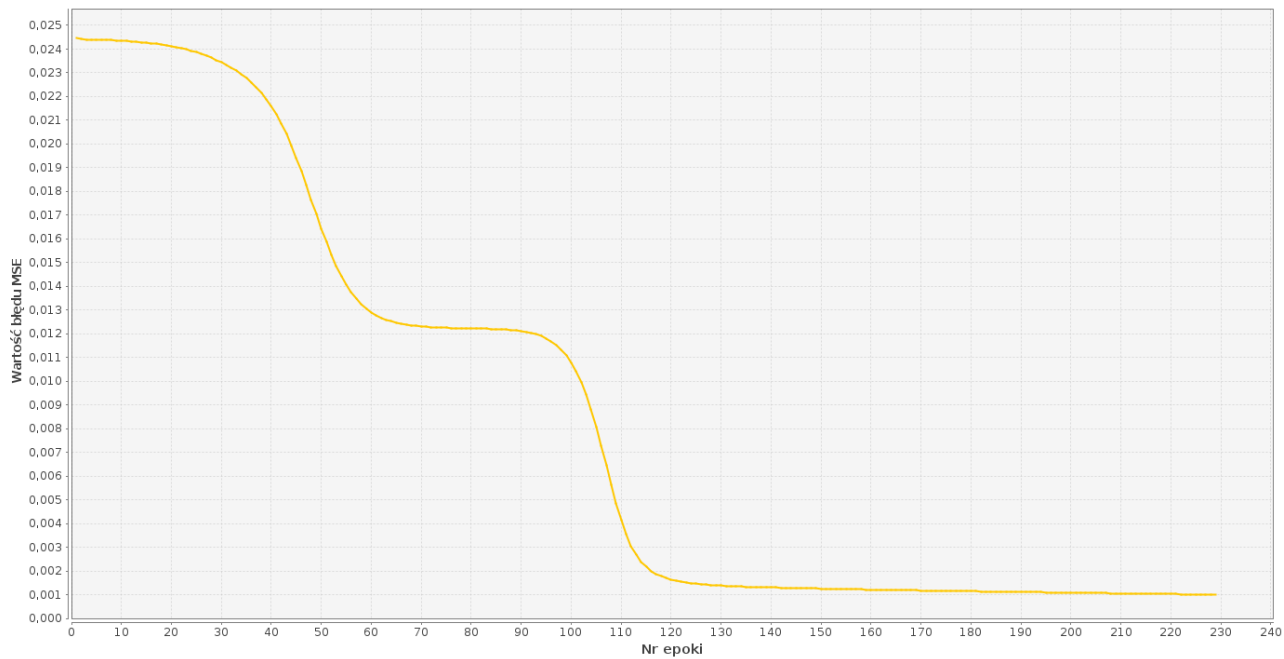
Dla każdej z powyższych trzech sieci wykonałem po trzy warianty uczenia oraz testowania. Każdą dla innego współczynnika uczenia: 0.5, 0.1 oraz 0.01. Jako próg błędu MSE dla którego sieć uważam za nauczoną przyjąłem wartość równą 0.001. Poniżej przedstawiam wykresy błędu MSE uczenia sieci dla 3500 danych uczących:

Wariant nr 1 – schemat 2-3-1

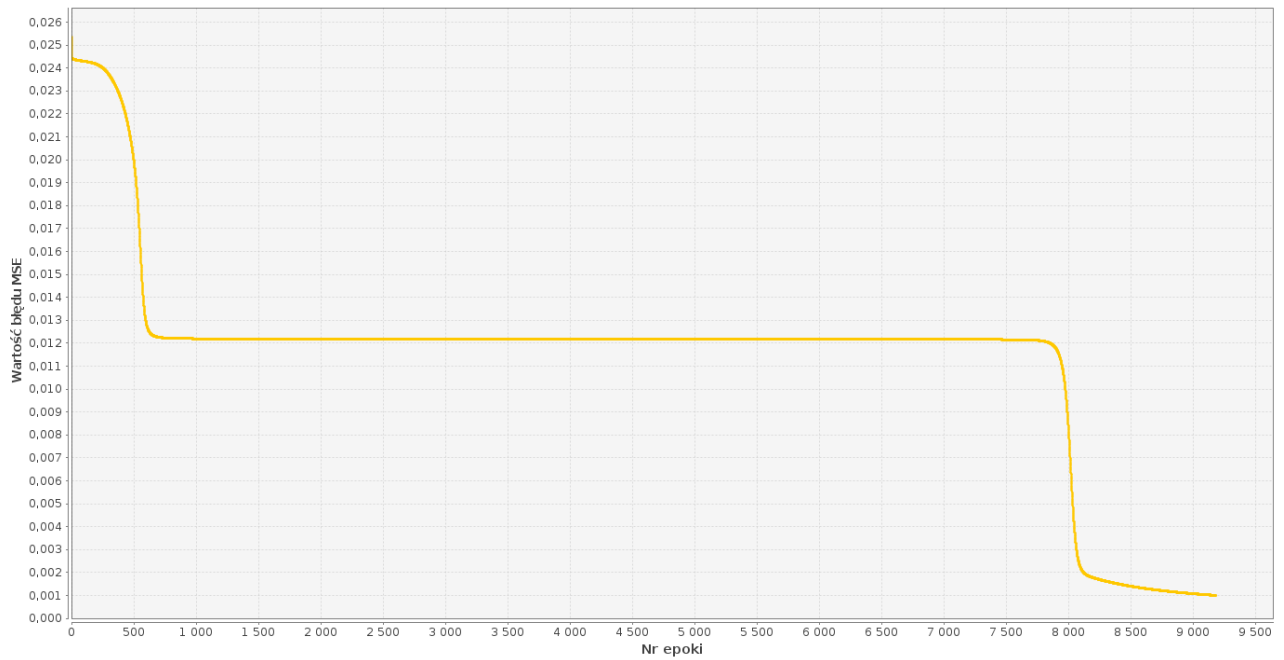
Wykres wartości błędu MSE dla sieci w konfiguracji 2-3-1 ze współczynnikiem uczenia 0.5



Wykres wartości błędu MSE dla sieci w konfiguracji 2-3-1 ze współczynnikiem uczenia 0.1

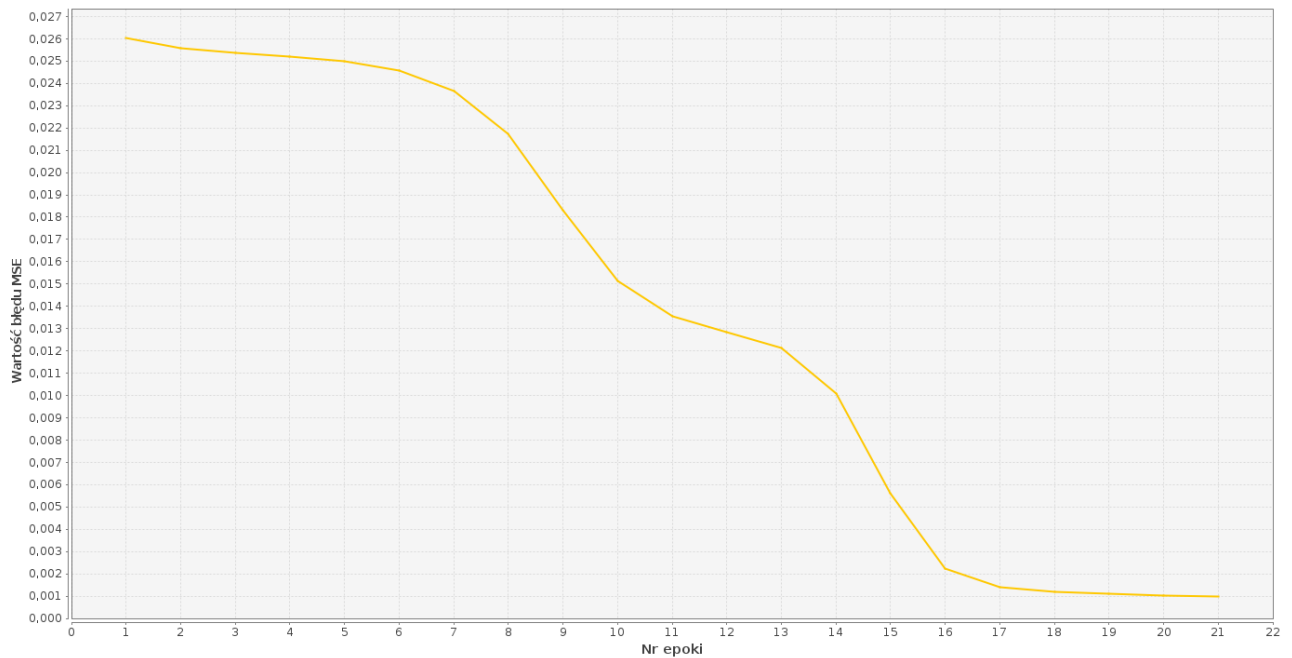


Wykres wartości błędu MSE dla sieci w konfiguracji 2-3-1 ze współczynnikiem uczenia 0.01

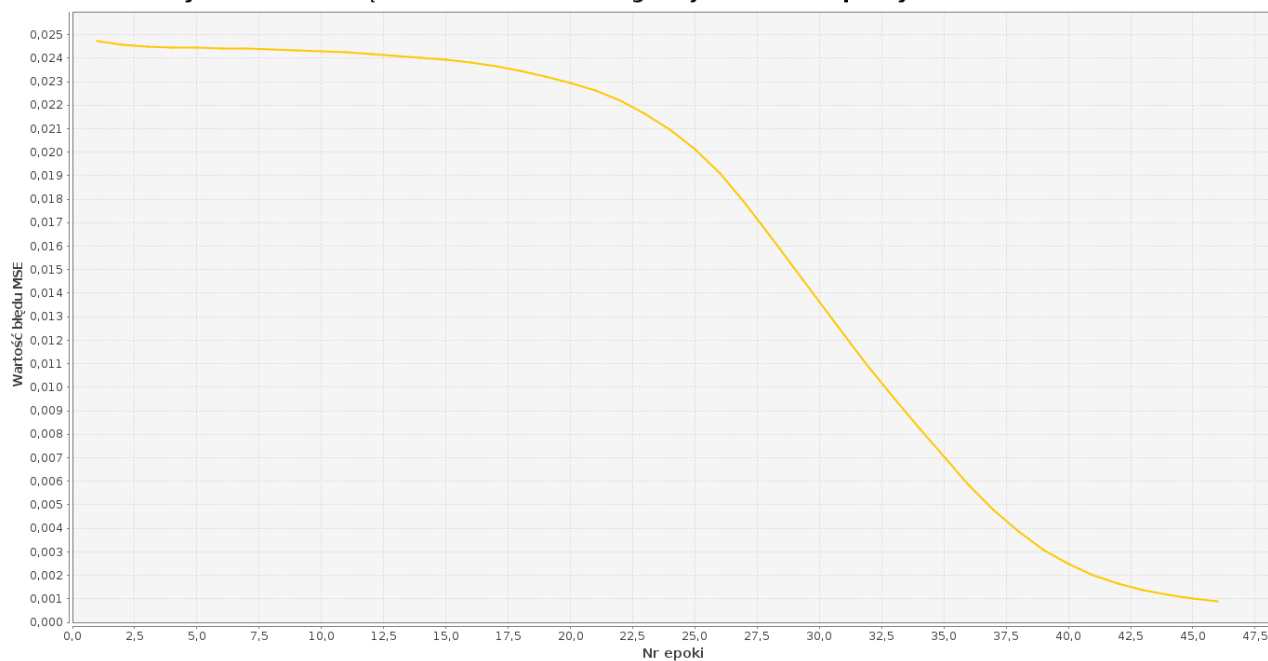


Wariant nr 2 – schemat 2-10-1

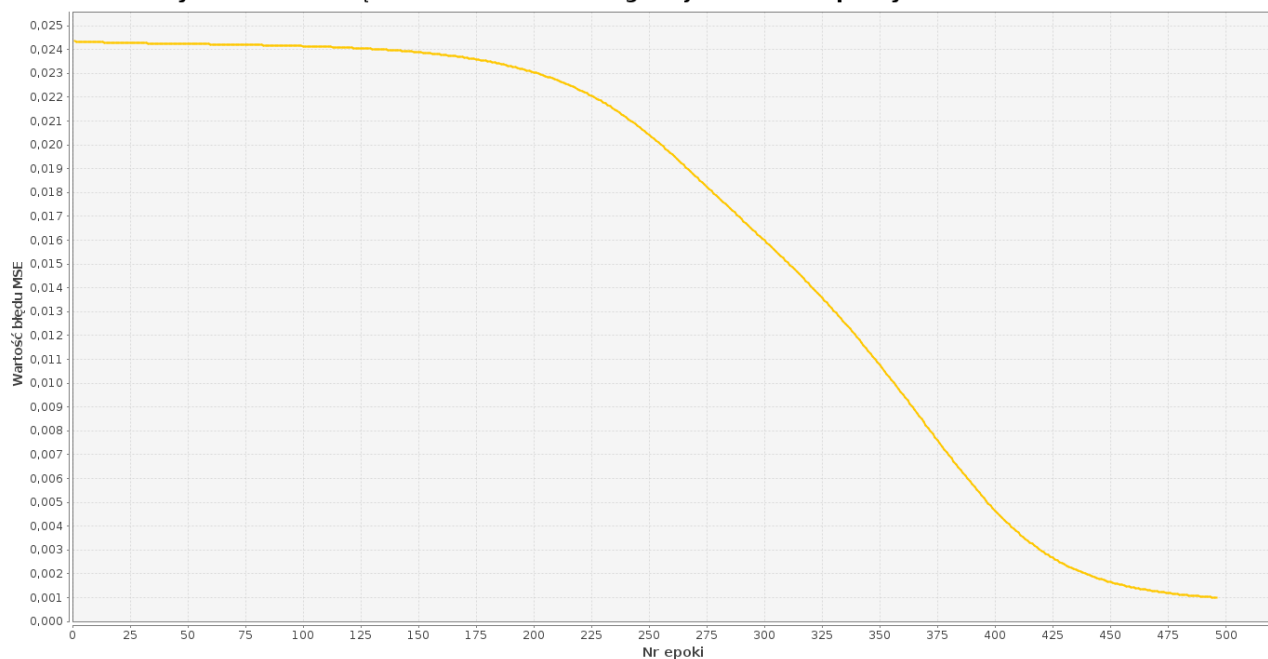
Wykres wartości błędu MSE dla sieci w konfiguracji 2-10-1 ze współczynnikiem uczenia 0.5



Wykres wartości błędu MSE dla sieci w konfiguracji 2-10-1 ze współczynnikiem uczenia 0.1

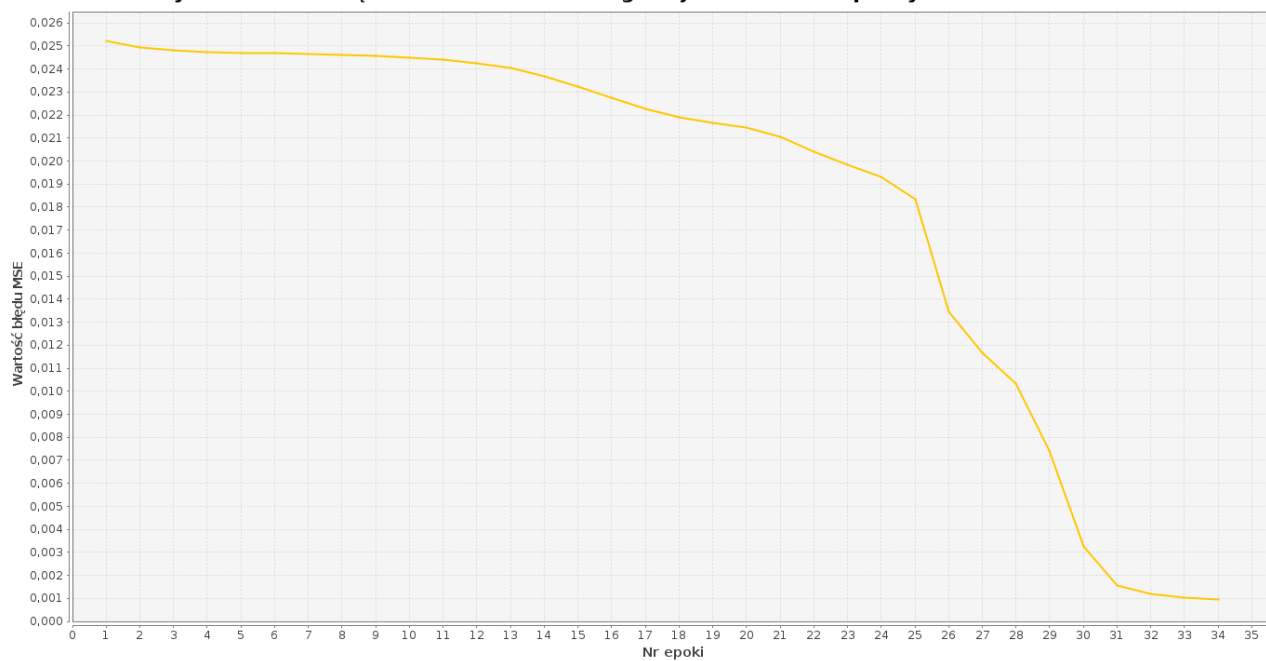


Wykres wartości błędu MSE dla sieci w konfiguracji 2-10-1 ze współczynnikiem uczenia 0.01

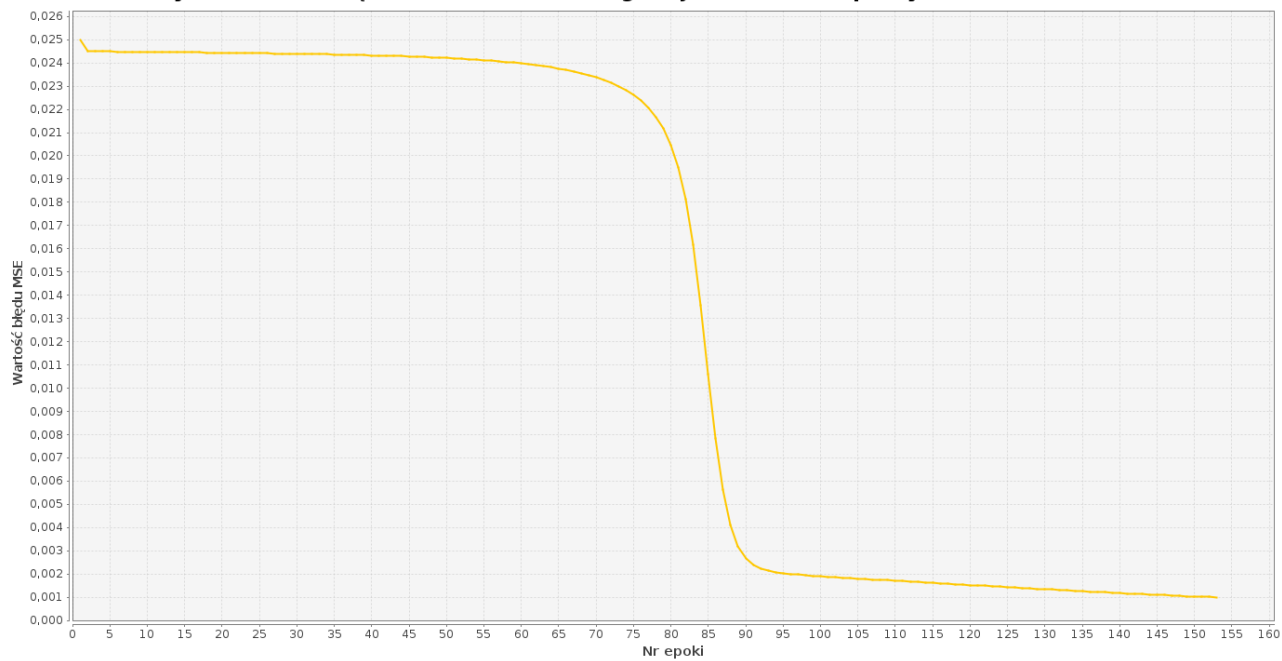


Wariant nr 3 – schemat 2-15-5-1

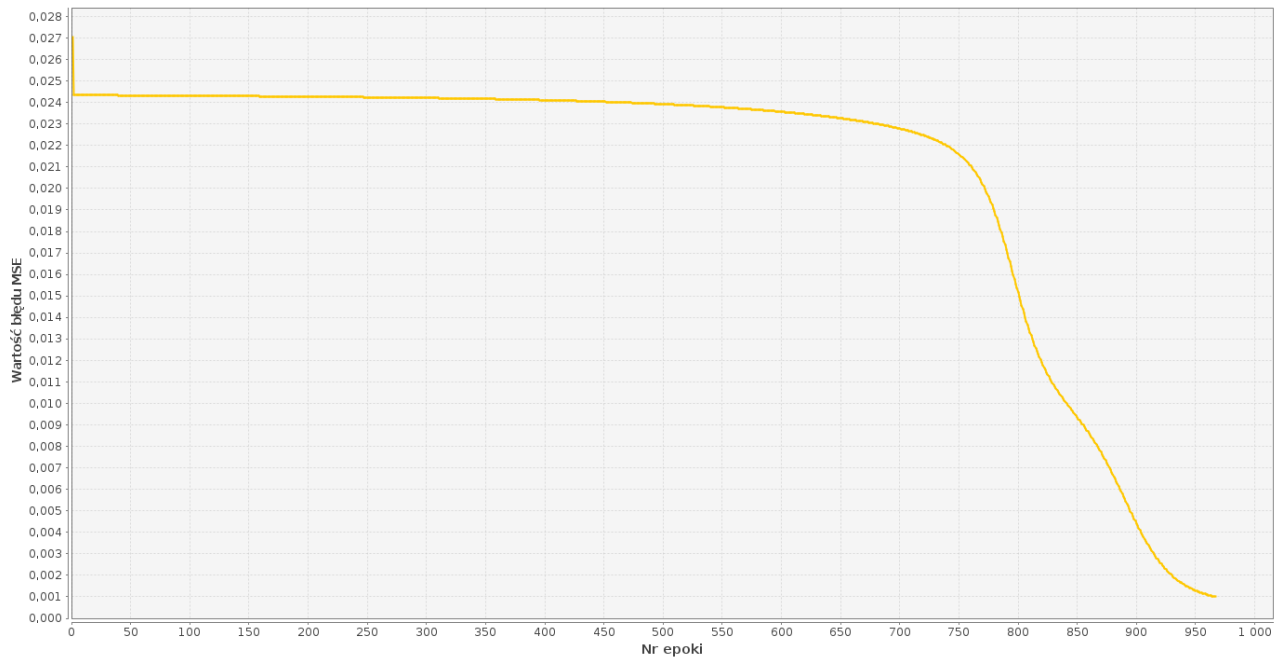
Wykres wartości błędu MSE dla sieci w konfiguracji 2-15-5-1 ze współczynnikiem uczenia 0.5



Wykres wartości błędu MSE dla sieci w konfiguracji 2-15-5-1 ze współczynnikiem uczenia 0.1

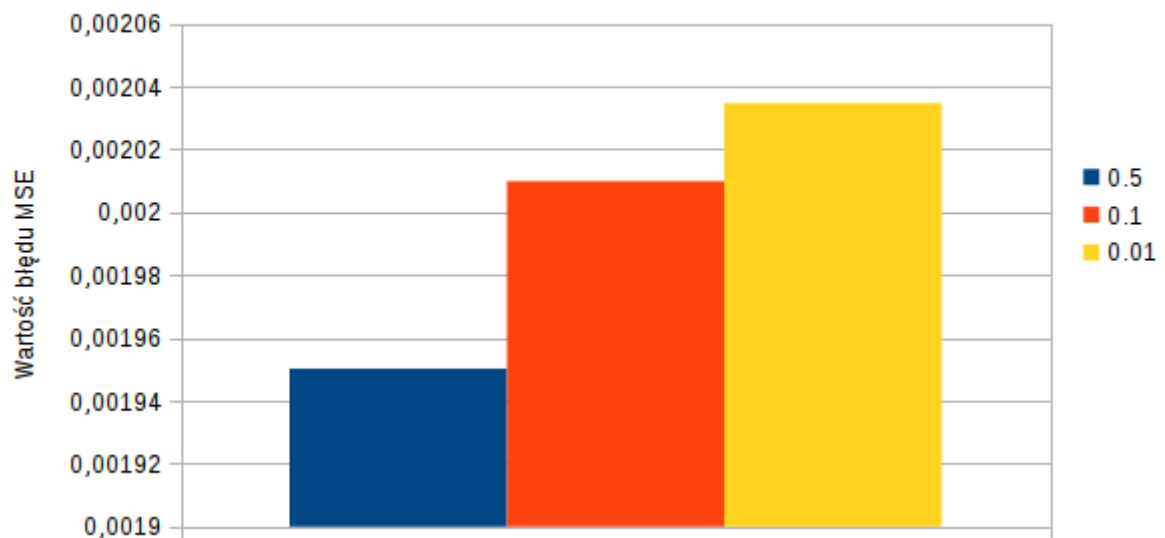


Wykres wartości błędu MSE dla sieci w konfiguracji 2-15-5-1 ze współczynnikiem uczenia 0.01

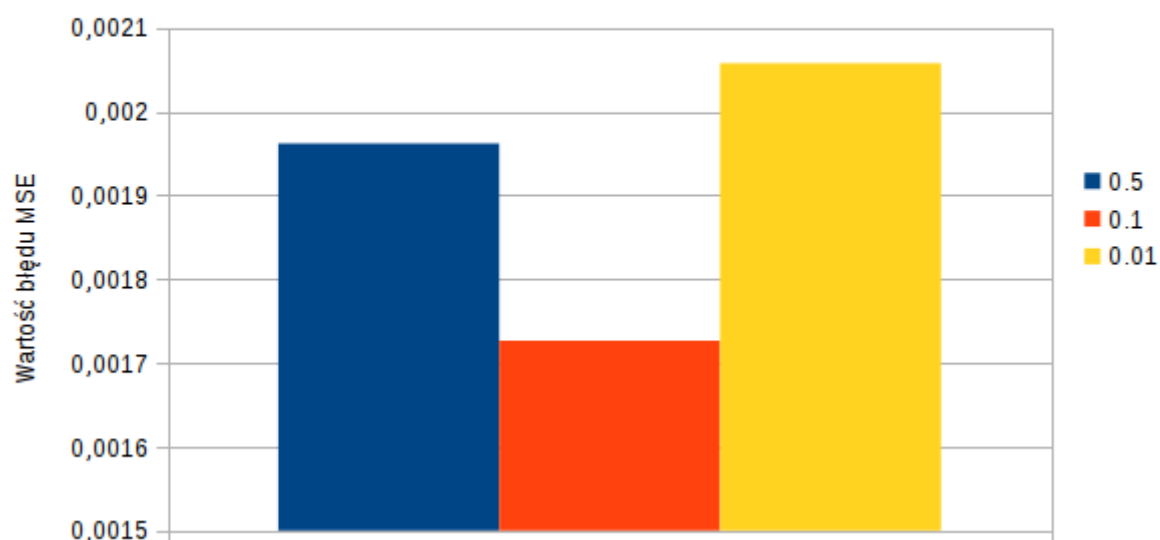


Po nauczeniu każdej sieci dla każdego wariantu współczynnika uczenia przystąpiłem do testowania tychże sieci dla pozostałych 1500 danych testujących. Poniżej przedstawiam histogramy dla porównania błędów MSE dla testowania sieci:

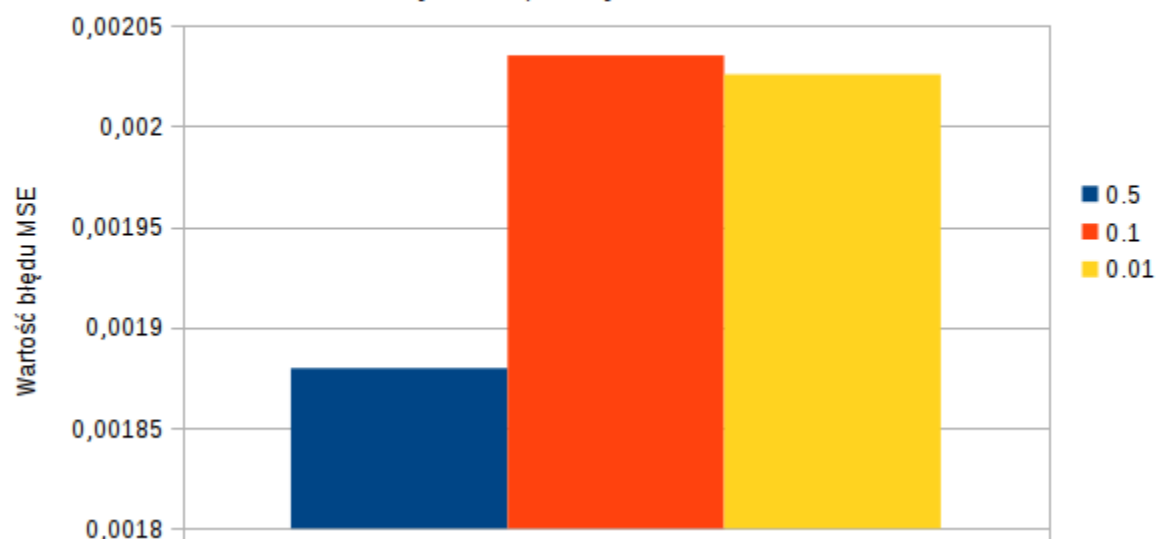
Wartości błędu MSE dla testowania sieci 2-3-1 dla różnych współczynników uczenia



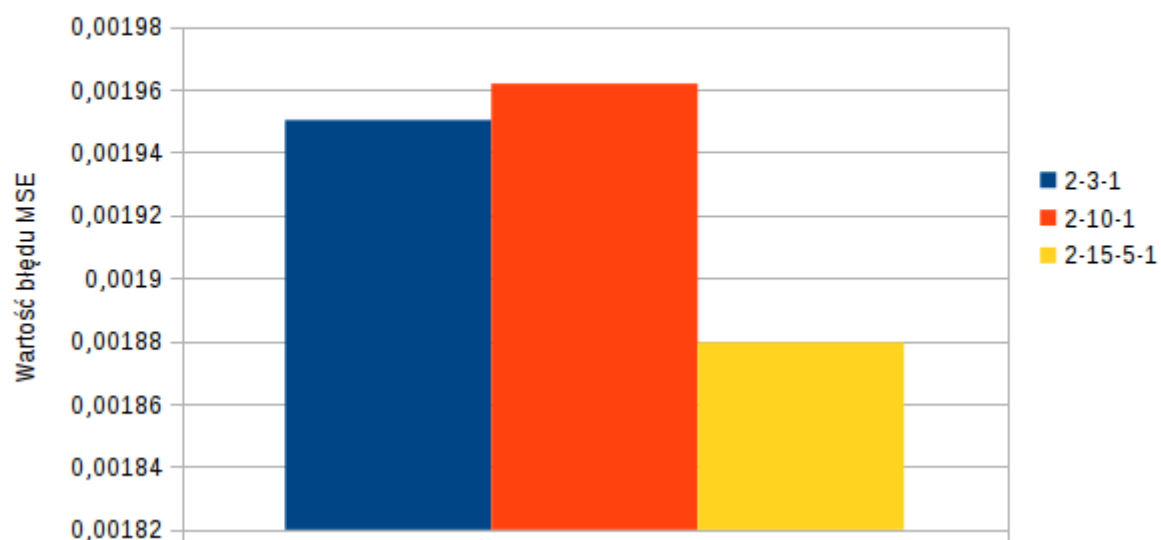
Wartości błędu MSE dla testowania sieci 2-10-1
dla różnych współczynników uczenia



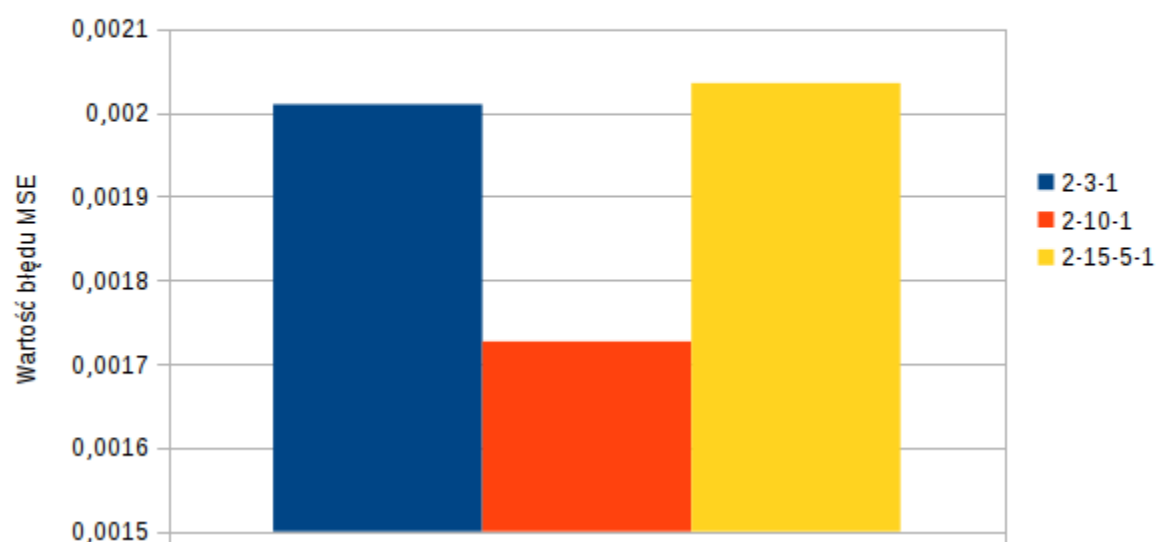
Wartości błędu MSE dla testowania sieci 2-15-5-1
dla różnych współczynników uczenia

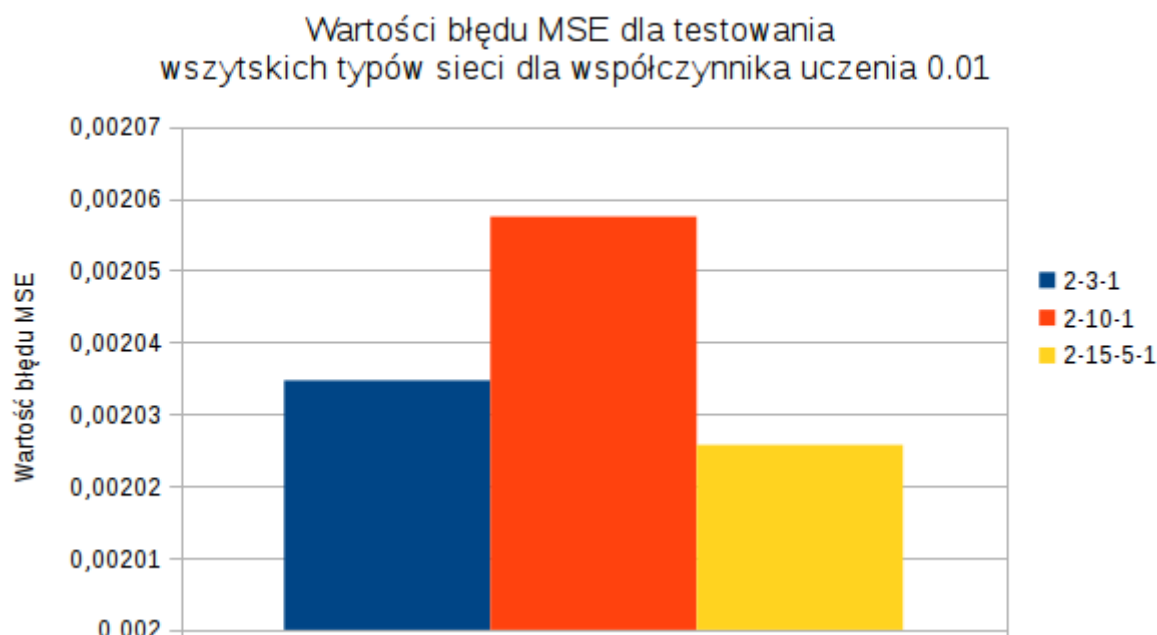


Wartości błędu MSE dla testowania
wszystkich typów sieci dla współczynnika uczenia 0.5



Wartości błędu MSE dla testowania
wszystkich typów sieci dla współczynnika uczenia 0.1





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

Niezależnie od przyjętego schematu budowy sieci widać, że im mniejszy współczynnik uczenia, tym dłużej zajął proces uczenia.

Dla schematu 2-3-1:

- 0.5 – około 50 epok
- 0.1 – około 230 epok
- 0.01 – około 9000 epok

Dla schematu 2-10-1:

- 0.5 – około 20 epok
- 0.1 – około 50 epok
- 0.01 – około 500 epok

Dla schematu 2-15-5-1:

- 0.5 – około 30 epok
- 0.1 – około 150 epok
- 0.01 – około 950 epok

Jednak spoglądając na wykresy błędów MSE podczas uczenia widać jak połączenie różnej konfiguracji sieci, ilości neuronów oraz współczynników uczenia, wpływa na proces uczenia. Niekiedy błąd MSE płynnie zmniejszał się podczas całego uczenia, niekiedy bardzo szybko malał by na samym końcu utrzymywać się prawie na stałym poziomie, a jeszcze w innym przypadku od początku prawie się nie zmieniał, by nagle zacząć drastycznie szybko spadać.

Sytuacja przedstawia się jeszcze inaczej jeśli rozważymy błąd MSE dla testowania sieci. Rozważając błąd MSE w skali sieci jednego rodzaju, widać, że dla wariantu nr1 oraz nr3 najniższą wartość błędu sieć osiągnęła dla współczynnika uczenia równego 0.5. Jednak w wariantcie nr2

najlepszy wynik był przy współczynniku uczenia równym 0.1. Warto również zauważyć, że współczynnik uczenia równy 0.01 zawsze wypadł bardzo źle.

Porównując jednak różne schematy widać, że współczynnik uczenia:

- 0.5 – najlepszy jest przy schemacie 2-15-5-1
- 0.1 – najlepszy jest przy schemacie 2-10-1
- 0.01 – najlepszy jest przy schemacie 2-15-5-1, lecz niewiele lepszy niż w schemacie 2-3-1

4) Sformułowanie wniosków:

Analizując powyższe wyniki można zauważyć, że im mniejszy współczynnik uczenia – tym dłużej sieci zajęła nauka. Jednak istotą uczenia sieci neuronowych nie jest to jak długo się uczy, lecz jak dokładnie to robi. Widać to na histogramach przedstawiających błąd MSE przy testowaniu w zależności od zastosowanego współczynnika uczenia oraz użytego schematu sieci.

Można wyłonić dwóch zwycięzców tych testów: sieć 2-15-5-1 dla współczynnika uczenia 0.5, oraz sieć 2-10-1 dla współczynnika uczenia 0.1. W obu przypadkach sieci nauczyły się bardzo szybko, a podczas testowania wypadły również najlepiej.

Podsumowując, można stwierdzić, że dobranie odpowiedniego współczynnika uczenia jest bardzo ważne jednak istotą uczenia sieci wielowarstwowych jest tak naprawdę odgadnięcie jak sama sieć powinna wyglądać – czyli z ilu warstw powinna się składać oraz z ilu neuronów.

5) Listing całego kodu:

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.util.Random;

public class Rastrigin {
    private static int trainingSize = 3500, testingSize = 1500; //rozmiar danych uczących i testujących
    static double[][] trainingData;
    static double[][] testingData;
    static double[] trainingY;
    static double[] testingY;

    public static void main ( String[] args ) throws FileNotFoundException {
        generateData();
        PrintWriter pw_train = new PrintWriter(new File("train.csv"));
        PrintWriter pw_test = new PrintWriter(new File("test.csv"));
        StringBuilder sb_train = new StringBuilder();
        StringBuilder sb_test = new StringBuilder();

        for ( int i = 0; i < trainingSize; i++ ) {
            sb_train.append( trainingData[i][0] );
            sb_train.append( ',' );
            sb_train.append( trainingData[i][1] );
            sb_train.append( ',' );
            sb_train.append( trainingY[i] );
            sb_train.append( '\n' );
        }
        for ( int i = 0; i < testingSize; i++ ){
            sb_test.append(testingData[i][0]);
            sb_test.append(',');
            sb_test.append(testingData[i][1]);
            sb_test.append(',');
            sb_test.append(testingY[i]);
            sb_test.append('\n');
        }

        pw_train.write(sb_train.toString());
        pw_train.close();
        pw_test.write(sb_test.toString());
        pw_test.close();
    }
}
```

```

private static double calculateRastrigin3D ( double x, double y ) {
    return 20 + Math.pow(x, 2) + Math.pow(y, 2) - 10 * (Math.cos(2 * Math.PI * x) + Math.cos(2 * Math.PI * y));
}

//generowanie danych uczących i testujących
private static void generateData () {
    trainingData = new double[trainingSize][2];
    testingData = new double[testingSize][2];
    trainingY = new double[trainingSize];
    testingY = new double[testingSize];
    Random rand = new Random();

    for ( int i = 0; i < trainingSize; i++ ) {
        trainingData[i][0] = normalization( 4.0 * rand.nextDouble() - 2.0 );
        trainingData[i][1] = normalization( 4.0 * rand.nextDouble() - 2.0 );
        trainingY[i] = normalizationRastrigin( calculateRastrigin3D( trainingData[i][0], trainingData[i][1] ) );
    }

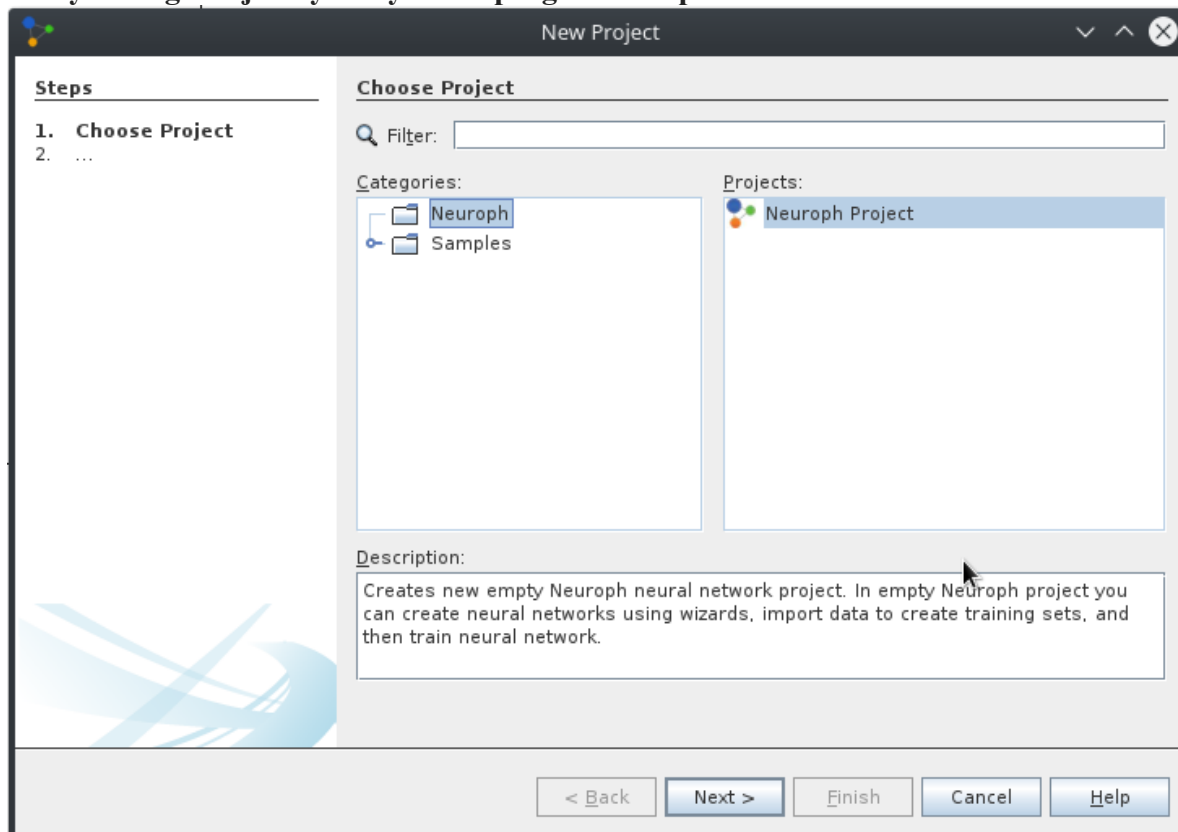
    for ( int i = 0; i < testingSize; i++ ) {
        testingData[i][0] = normalization( 4.0 * rand.nextDouble() - 2.0 );
        testingData[i][1] = normalization( 4.0 * rand.nextDouble() - 2.0 );
        testingY[i] = normalizationRastrigin( calculateRastrigin3D( testingData[i][0], testingData[i][1] ) );
    }
}

//funkcja normalizująca dane "z" funkcji Rastrigin
private static double normalizationRastrigin ( double x ) {
    double min = 0, max = 45;
    double new_min = 0, new_max = 1;
    return ( ( x - min ) / ( max - min ) ) * ( new_max - new_min ) + new_min;
}

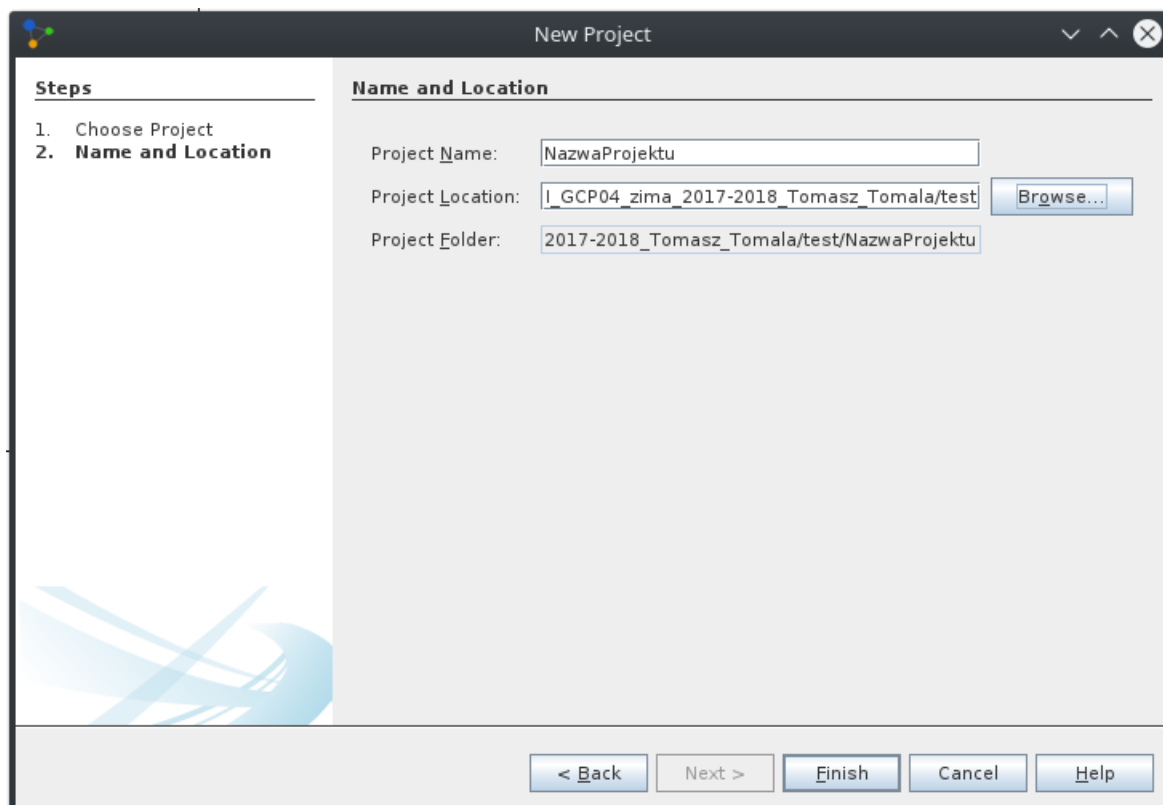
//funkcja normalizująca dane wejściowe "x" i "y" do funkcji Rastrigin
private static double normalization ( double x ) {
    double min = - 2, max = 2;
    double new_min = 0, new_max = 1;
    return ( ( x - min ) / ( max - min ) ) * ( new_max - new_min ) + new_min;
}
}

```

6) Zrzuty konfiguracji i wykorzystania programu z opisem:



Po uruchomieniu programu zaczynamy od utworzenia nowego projektu. Przez kolejne kroki prowadzi nas łatwy w obsłudze kreator.



New Neural Network

Steps

1. Set neural network name and type
- 2.

Set neural network name and type

Neural Network Name:

Neural Network Ty...

- Empty Neural Network
- Adaline
- Perceptron
- Multi Layer Perceptron**
- Hopfield
- BAM
- Kohonen
- Supervised Hebbian
- Unsupervised Hebbian
- Maxnet
- Competitive Network
- RBF
- Instar
- OutStar

< Back Next > Finish Cancel Help

Po utworzeniu projektu tworzymy w nim nową sieć. Podajemy nazwę sieci oraz wybieramy z listy co to ma być za typ sieci.

New Neural Network

Steps

1. Set neural network name and type
- 2.

Setting Multi Layer Perceptron's parameters

Input neurons

Hidden neurons (space delimited for layers)

Output neurons

☒ Use Bias Neurons

☐ Connect input to output neurons

Transfer function

Learning rule

< Back Next > Finish Cancel Help

Następnie podajemy z ilu warstw będzie składać się sieć i z ilu neuronów będzie składać się poszczególne warstwa. Wybieramy również funkcję aktywacji i metodę uczenia.

New Data Set

Steps

1. Choose File Type
2. **Set data set name, type and number of inputs and outputs**

Set data set name, type and number of inputs and outputs

Data set name:

Type:

Number of inputs:

Number of outputs:

☒ Load from file

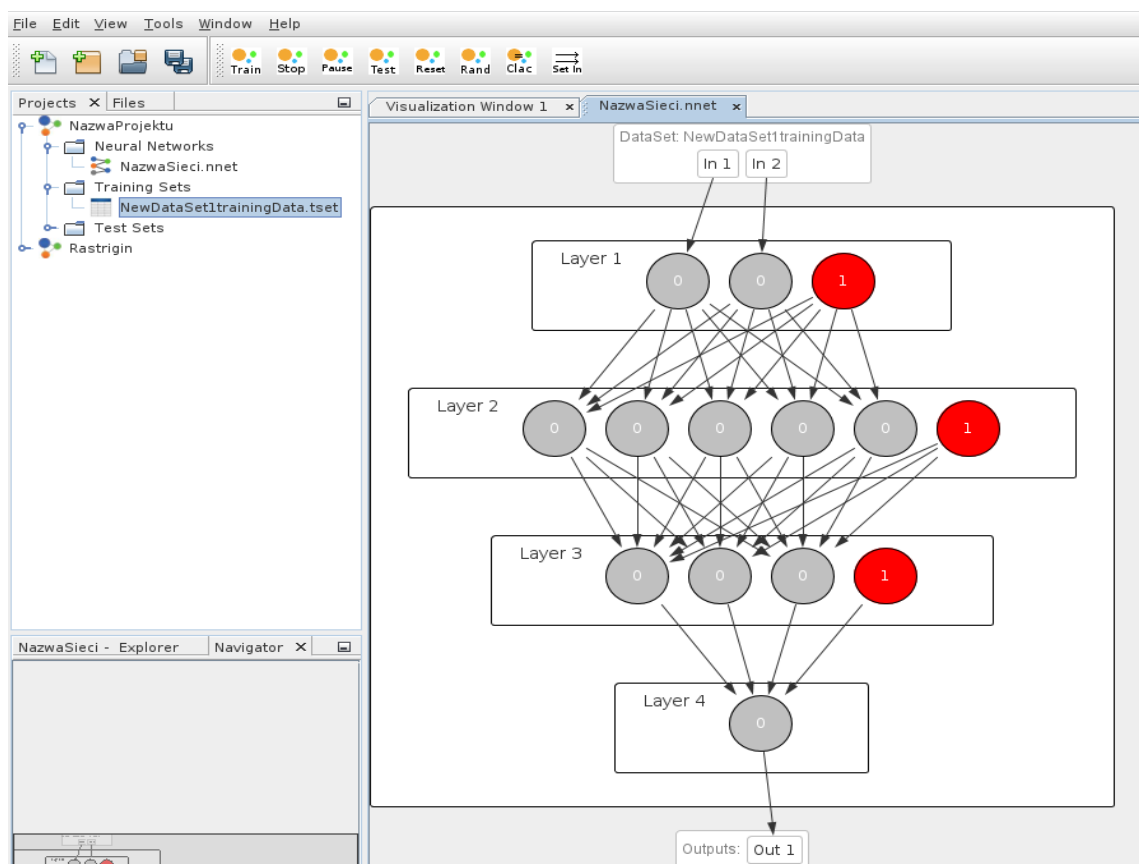
File:

Delimiter:

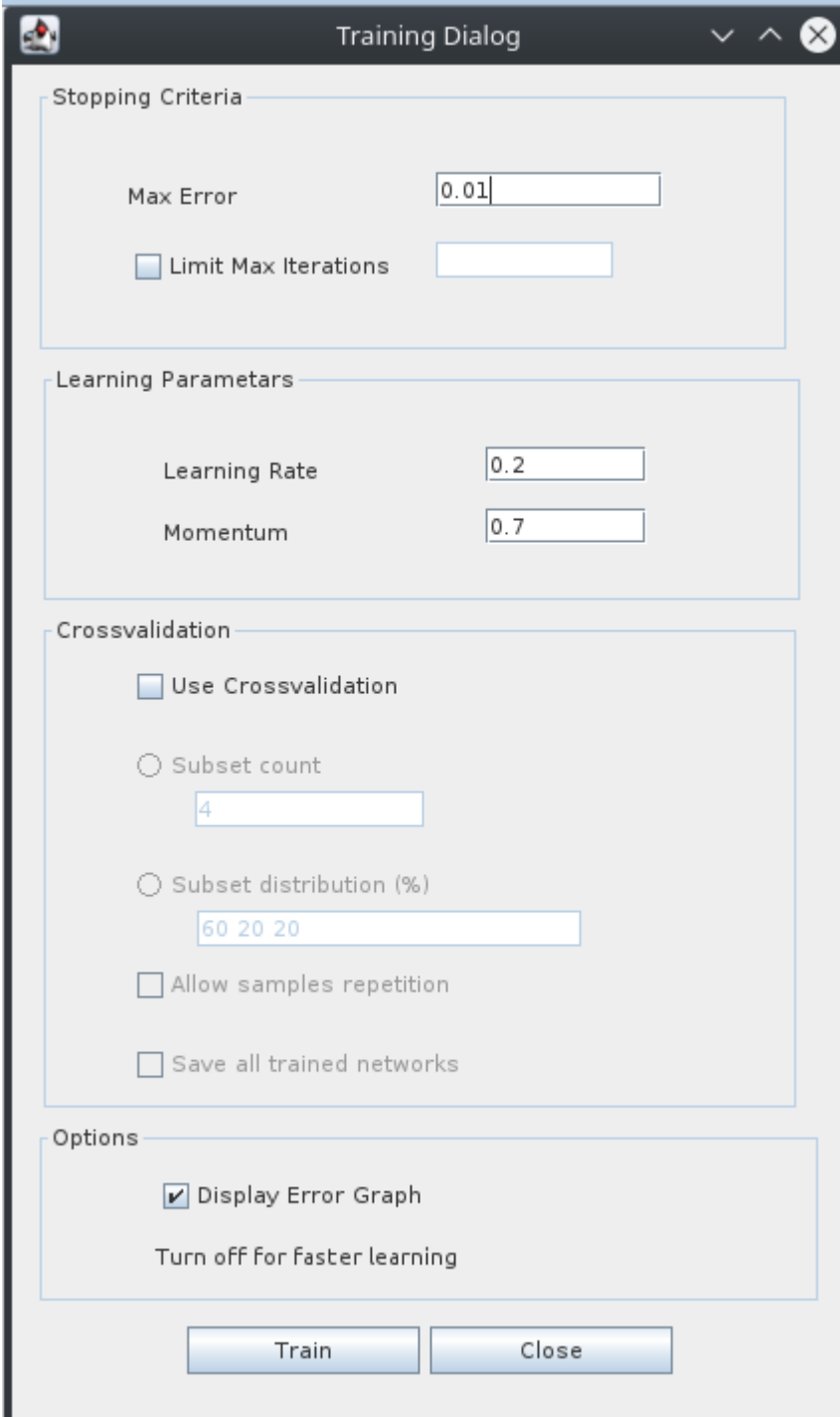
☐ First Line Contains Column Names

< Back Next > Finish Cancel Help

Następnie musimy wczytać dane uczące i testujące. Można wpisać je ręcznie, lub wczytać z pliku, tak jak pokazano na powyższym zrzucie ekranu.



Następnie przeciągamy wybrane dane na sieć, oraz wybrany z buttonów na pasku czy chcemy sieć uczyć, czy też testować.

A screenshot of a 'Training Dialog' window. The window has a title bar with a small icon on the left and standard window controls (minimize, maximize, close) on the right. The main area is divided into four sections: 'Stopping Criteria', 'Learning Parameters', 'Crossvalidation', and 'Options'. In 'Stopping Criteria', 'Max Error' is set to 0.01, and 'Limit Max Iterations' is unchecked. In 'Learning Parameters', 'Learning Rate' is 0.2 and 'Momentum' is 0.7. In 'Crossvalidation', 'Use Crossvalidation' is unchecked, 'Subset count' is 4, 'Subset distribution (%)' is 60 20 20, and both 'Allow samples repetition' and 'Save all trained networks' are unchecked. In 'Options', 'Display Error Graph' is checked, and there is a note 'Turn off for faster learning'. At the bottom are 'Train' and 'Close' buttons.

Training Dialog

Stopping Criteria

Max Error

☐ Limit Max Iterations

Learning Parameters

Learning Rate

Momentum

Crossvalidation

☐ Use Crossvalidation

☐ Subset count

☐ Subset distribution (%)

☐ Allow samples repetition

☐ Save all trained networks

Options

☒ Display Error Graph

Turn off for faster learning

Jeżeli wybierzemy uczenie sieci, musimy podać końcowy błąd uczenia, opcjonalnie maksymalną liczbę iteracji uczenia, oraz współczynnik uczenia. Po kliknięciu buttona train sieć zacznie się uczyć. W przypadku testowania sieć po prostu zwróci nam wyniki testowania w postaci tekstu na ekranie.

Bibliografia:

<http://neuroph.sourceforge.net/download.html>

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