

Konrad Tabiś

Laboratorium nr 3

Sprawozdanie - Podstawy Sztucznej Inteligencji – Scenariusz 3

Temat ćwiczenia: Budowa i działanie sieci wielowarstwowej typu feedforward.

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

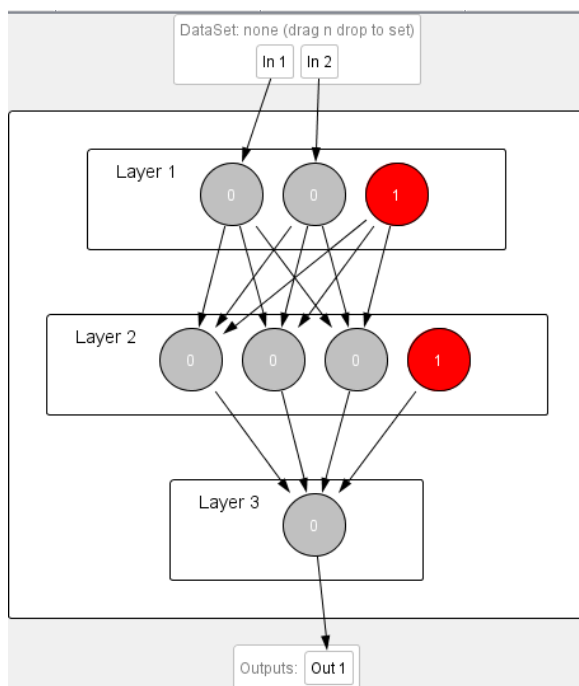
1) Syntetyczny opis budowy użytej sieci i algorytmu uczenia.

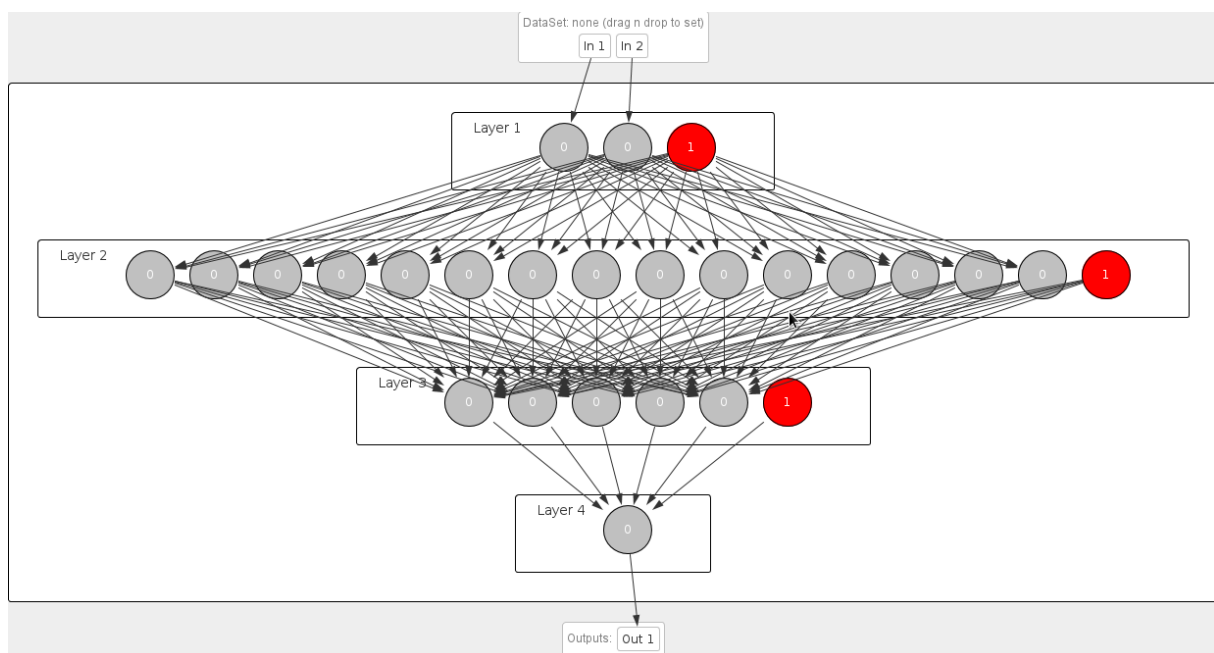
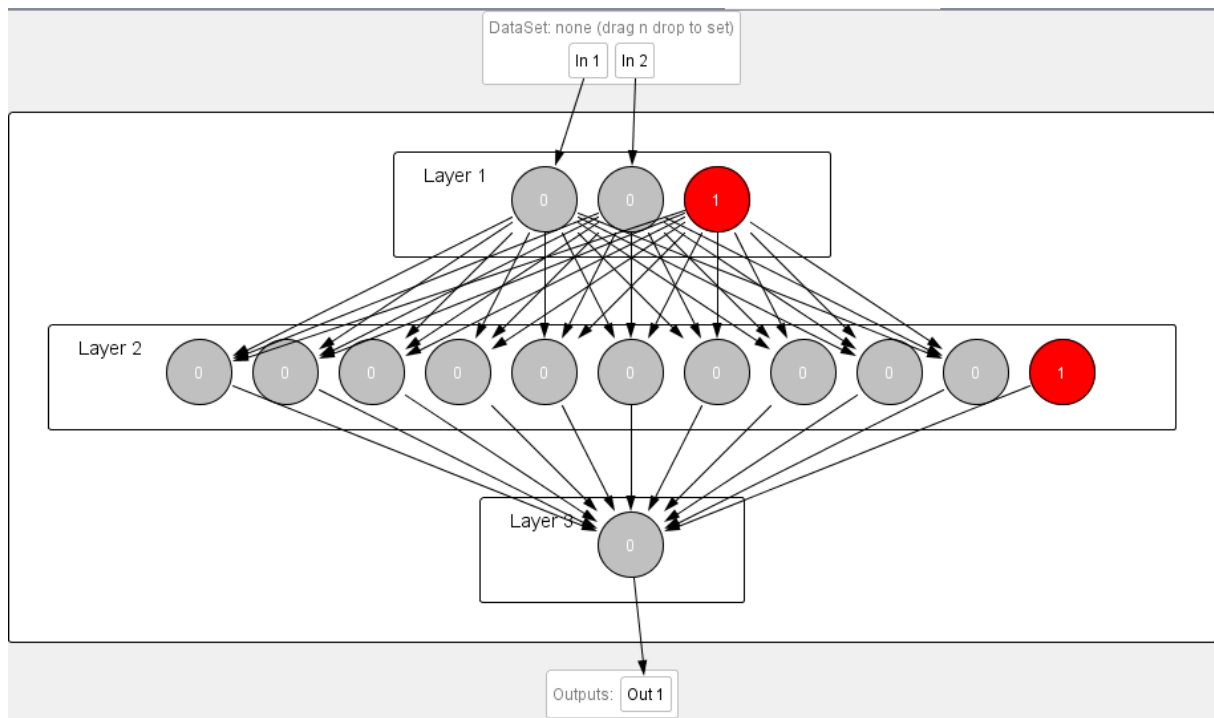
W celu wykonania ćwiczenia skorzystałem z narzędzia jakim jest *NeurophStudio*. Za pomocą kreatora stworzyłem sieci neuronowe, nauczyłem je oraz testowałem. Sieci które stworzyłem wykorzystują sigmoidalną funkcję aktywacyjną oraz algorytm uczenia Back Propagation.

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 błęd $\delta = d - y$ 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 błęd dla każdego neuronu następuje ponowne przepuszczenie przez całą sieć sygnałów wejściowych, jednak tym razem już z modyfikacją wag.

Stworzone sieci :



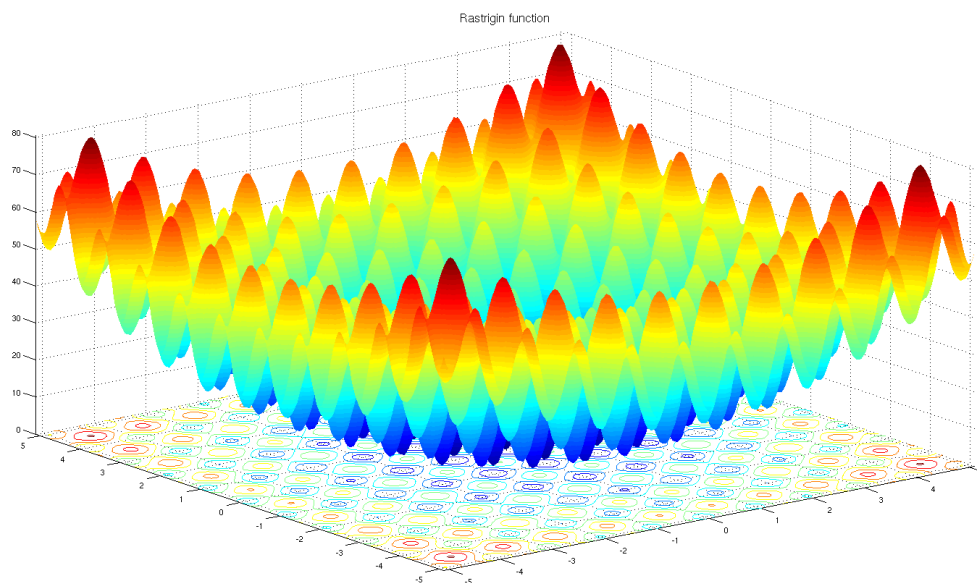


Do wygenerowania danych uczących i testujących napisałem własny program, w którym funkcja zwracała wynik funkcji Rastrigin dla losowych punktów „x” oraz „y” z zakresu [-2; 2] .

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:



2) Zestawienie otrzymanych wyników.

Wygenerowałem 3500 rekordów przeznaczonych na uczenie sieci oraz 1500 rekordów na testowanie. Otrzymane dane poddałem procesowi normalizacji.

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

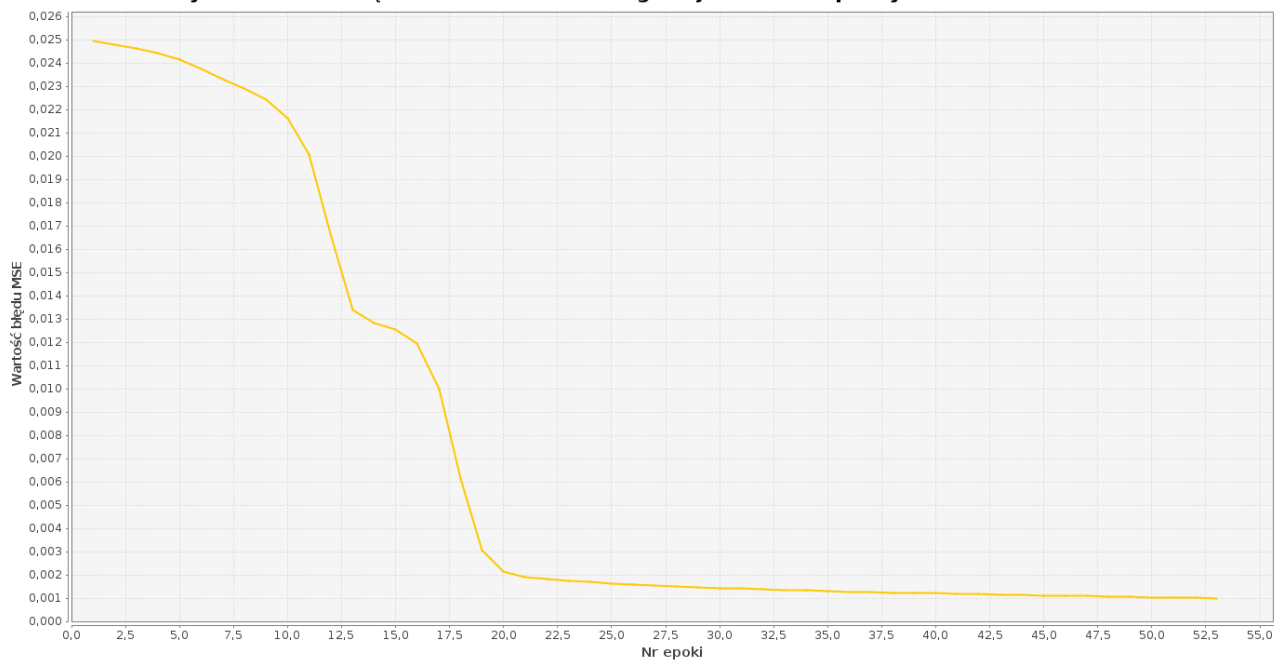
X_{newMax} – nowa maksymalna wartość

Wygenerowane dane zostały zapisane w plikach, by mogły zostać wykorzystane w programie NeurophStudio.

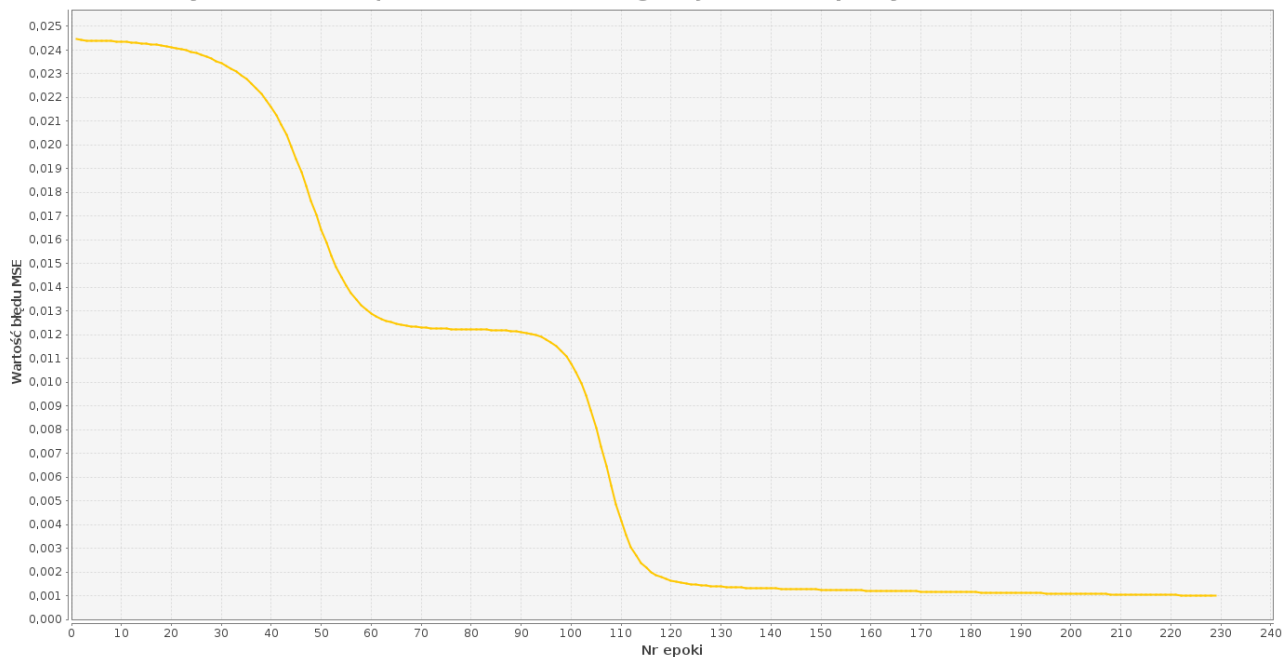
Dla każdej z powyższych trzech sieci wykonałem po trzy warianty uczenia oraz testowania. Każdą dla innego współczynnika uczenia: 0.5, 0.1 oraz 0.01. Jako próg błędu MSE dla którego sieć uważam za nauczoną przyjąłem wartość równą 0.001. Poniżej przedstawiam wykresy błędu MSE uczenia sieci dla 3500 danych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

Wariant nr 1 – schemat 2-3-1

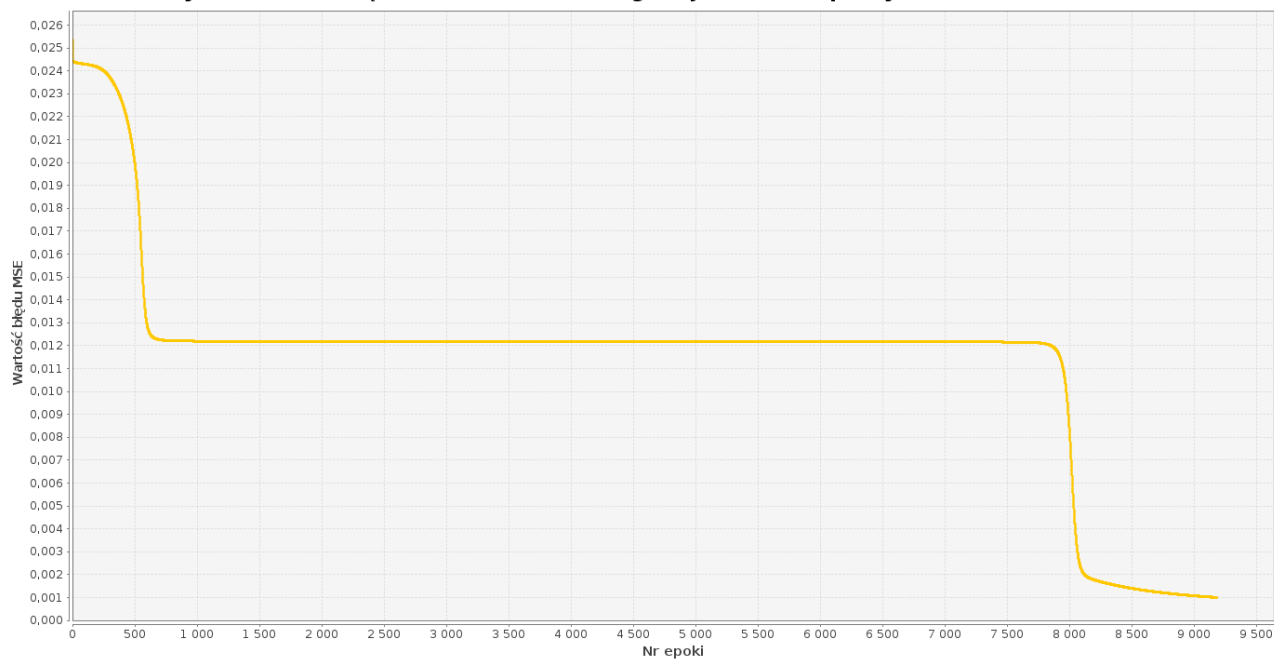
Wykres wartości błędu MSE dla sieci w konfiguracji 2-3-1 ze współczynnikiem uczenia 0.5



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

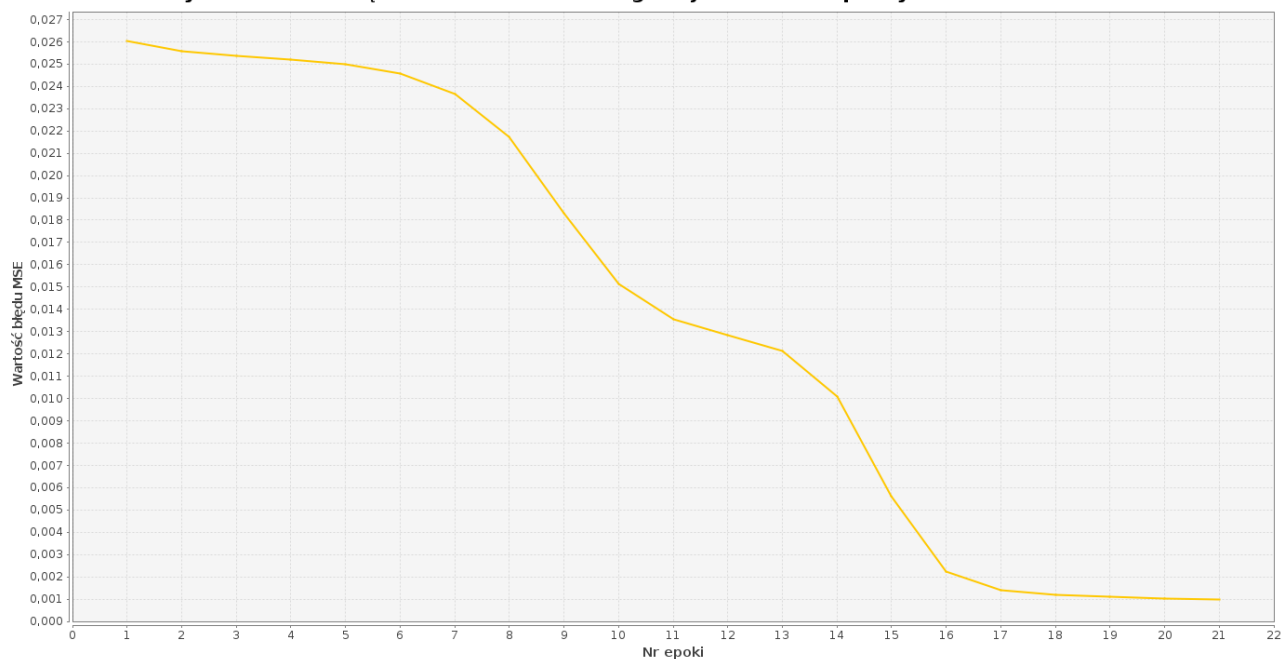


Wykres wartości błędu MSE dla sieci w konfiguracji 2-3-1 ze współczynnikiem uczenia 0.01

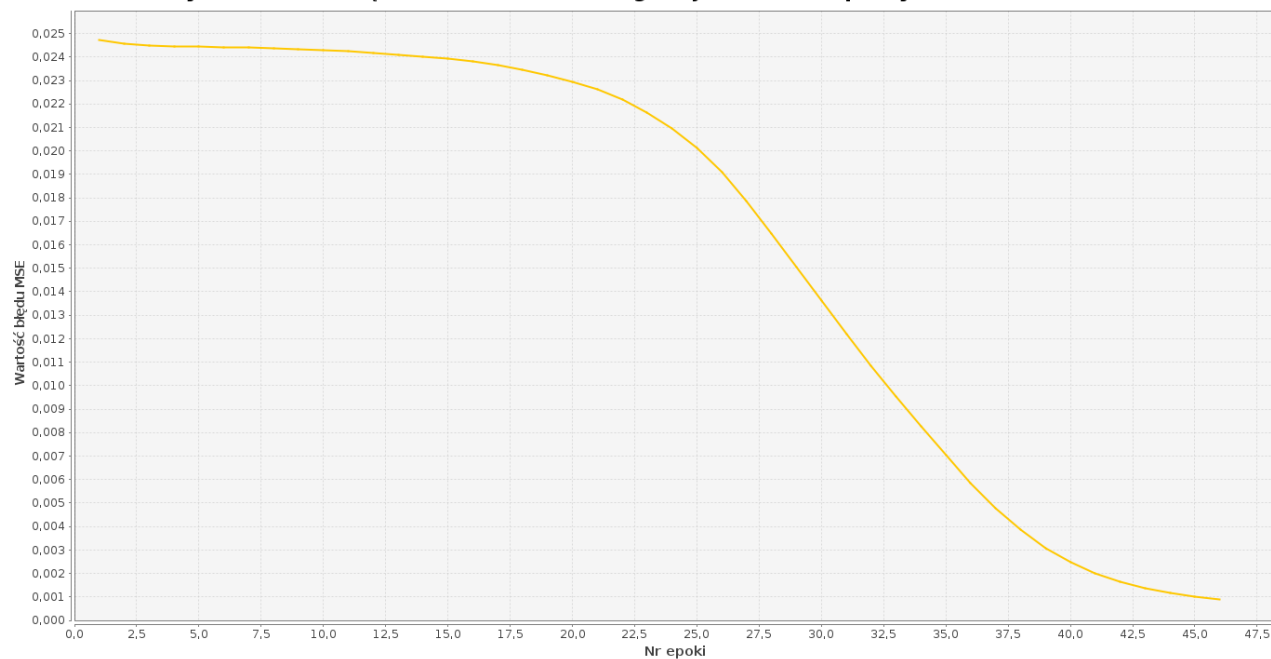


Wariant nr 2 – schemat 2-10-1

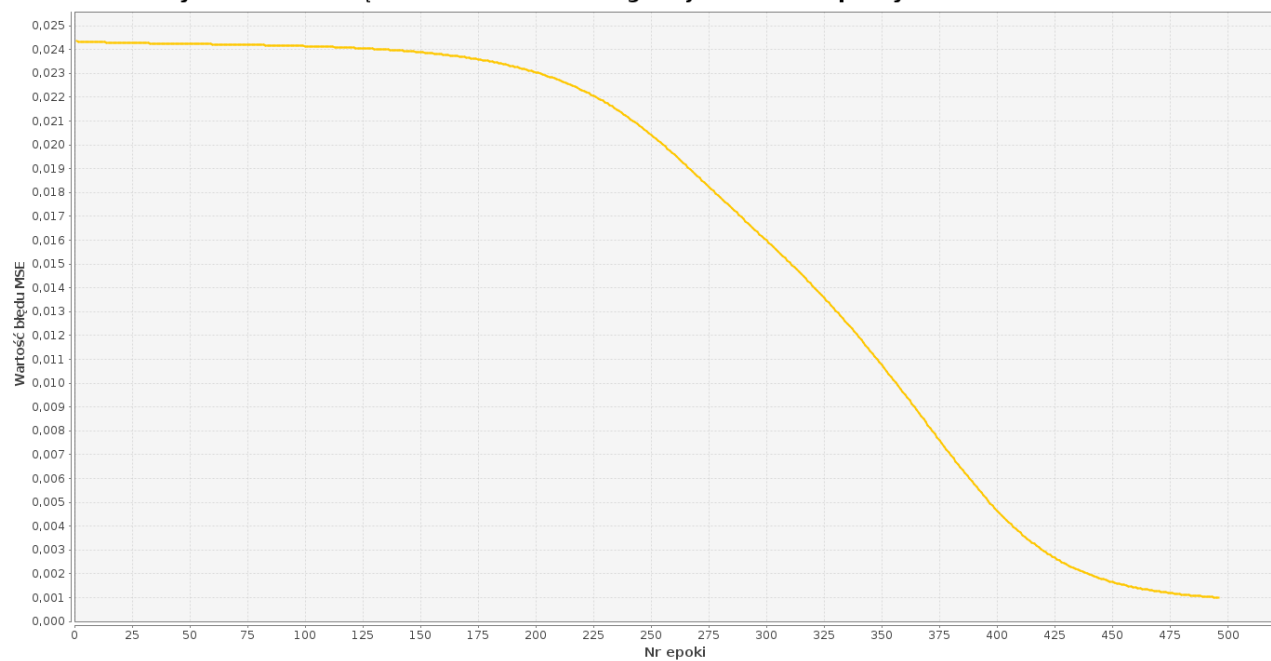
Wykres wartości błędu MSE dla sieci w konfiguracji 2-10-1 ze współczynnikiem uczenia 0.5



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

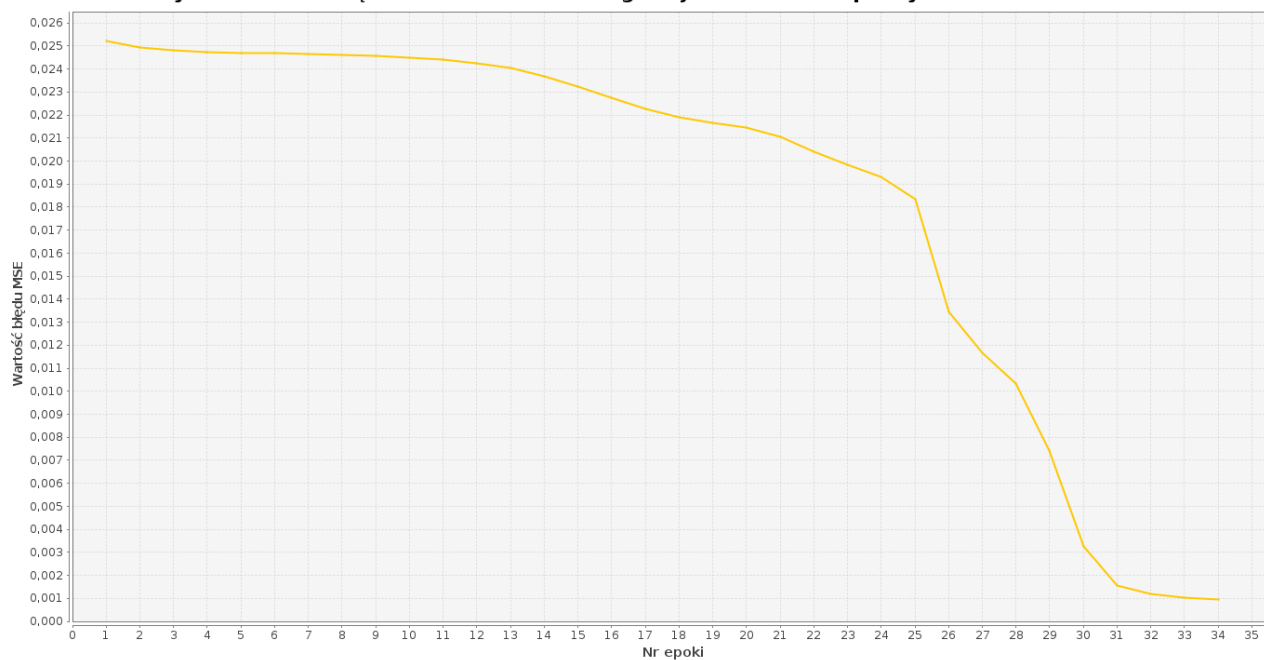


Wykres wartości błędu MSE dla sieci w konfiguracji 2-10-1 ze współczynnikiem uczenia 0.01

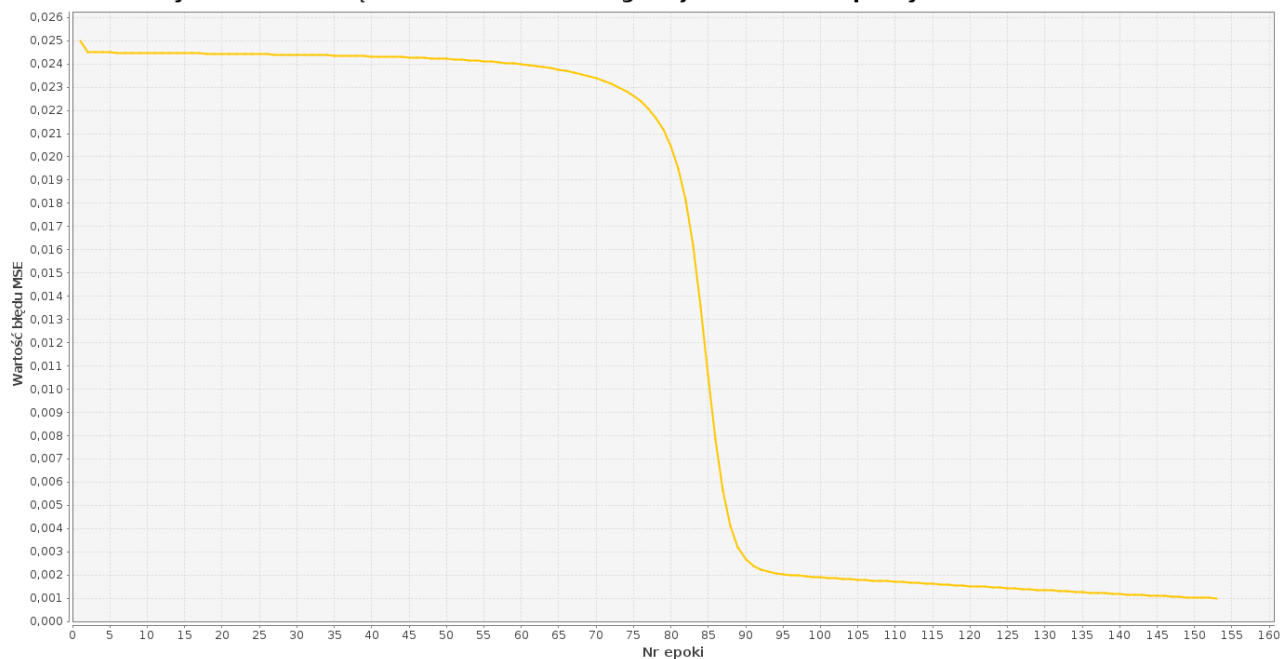


Wariant nr 3 – schemat 2-15-5-1

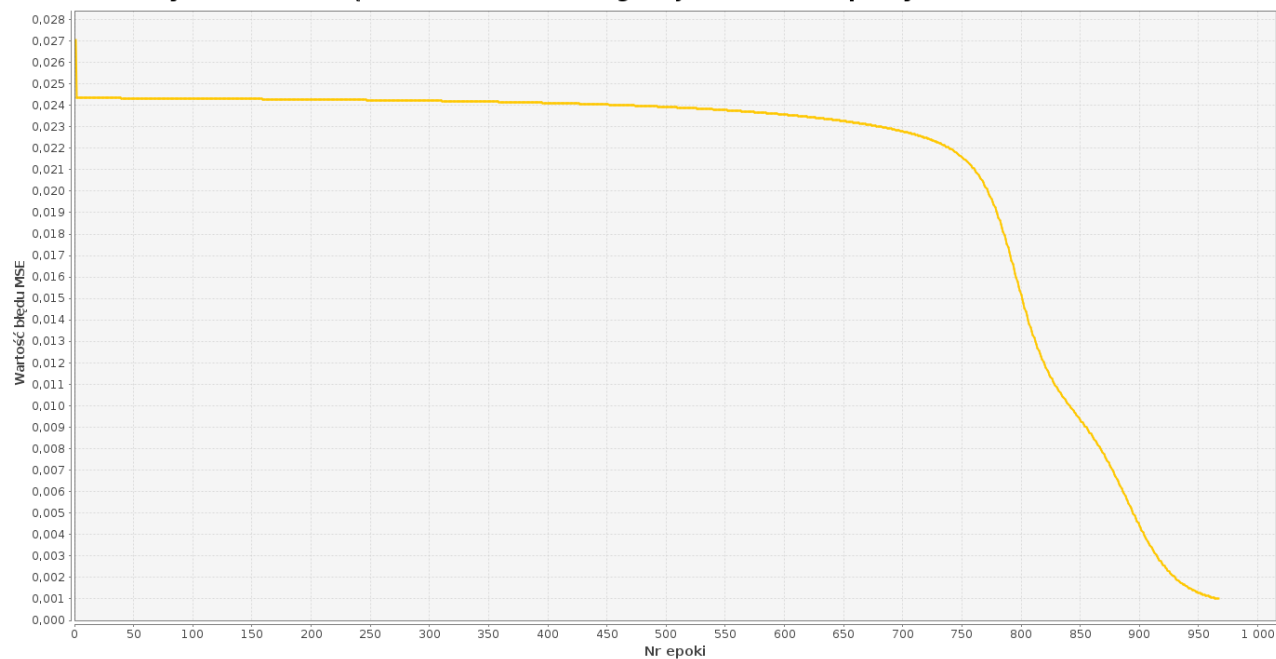
Wykres wartości błędu MSE dla sieci w konfiguracji 2-15-5-1 ze współczynnikiem uczenia 0.5



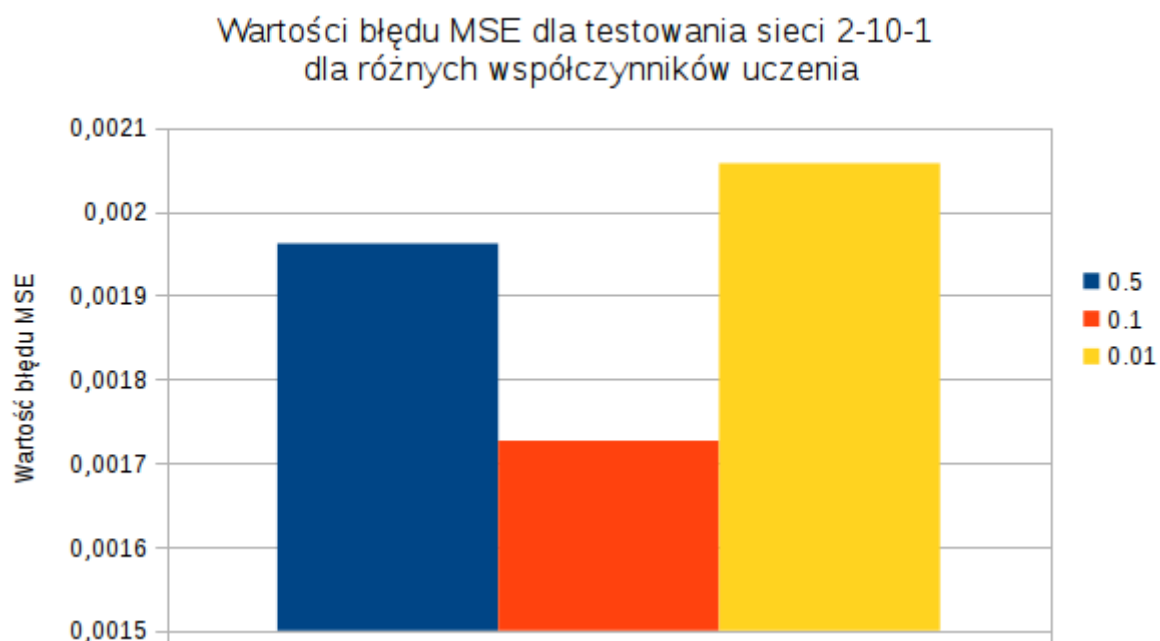
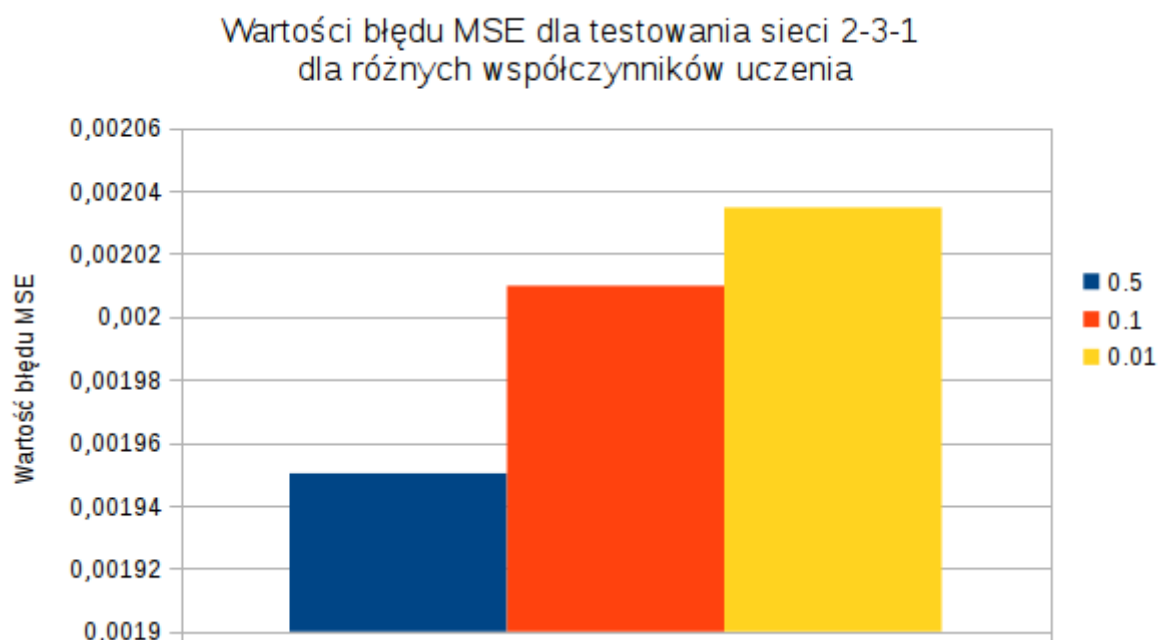
Wykres wartości błędów MSE dla sieci w konfiguracji 2-15-5-1 ze współczynnikiem uczenia 0.1



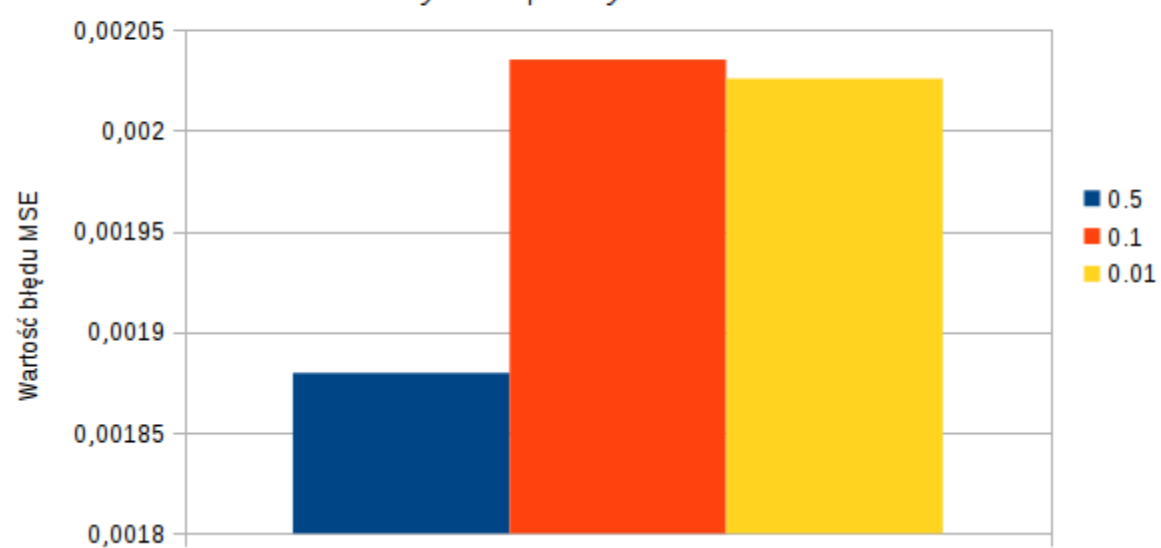
Wykres wartości błędu MSE dla sieci w konfiguracji 2-15-5-1 ze współczynnikiem uczenia 0.01



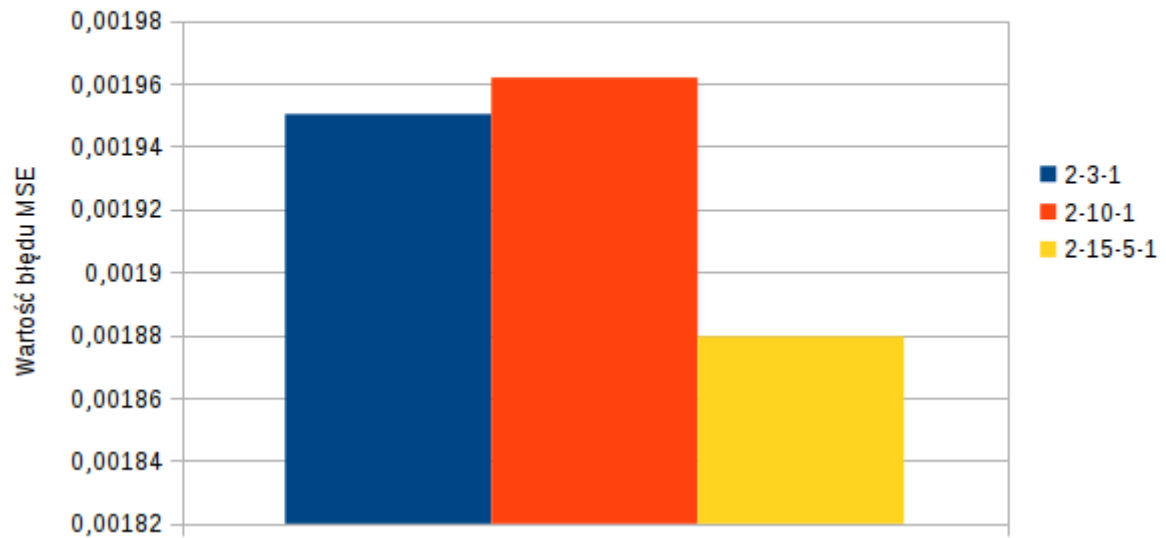
Po nauczaniu każdej sieci dla każdego wariantu współczynnika uczenia przystąpiłem do testowania tychże sieci dla pozostałych 1500 danych testujących. Poniżej przedstawiam histogramy dla porównania błędów MSE dla testowania sieci:



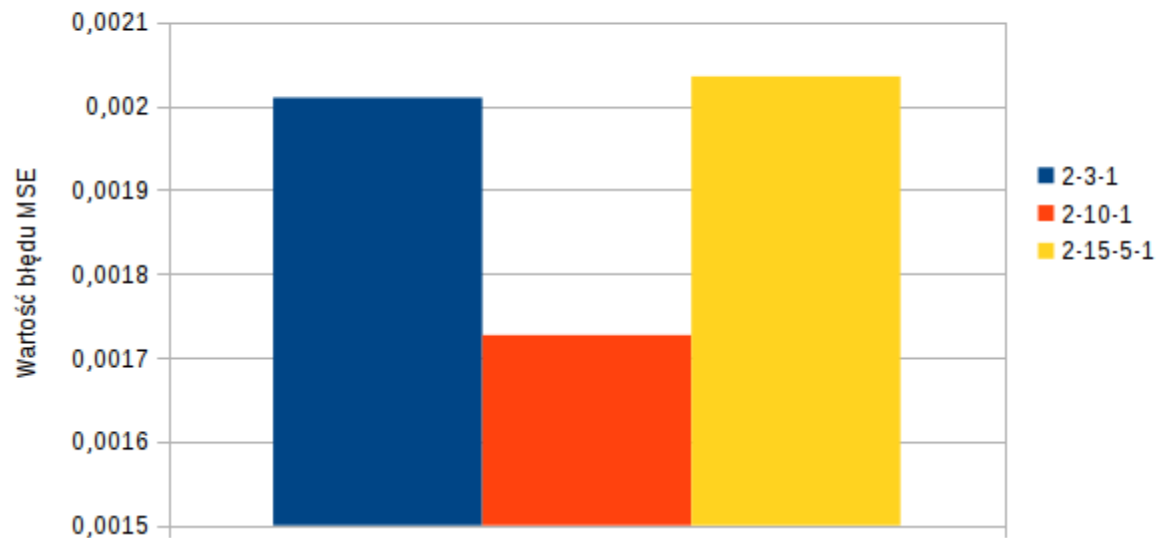
Wartości błędu MSE dla testowania sieci 2-15-5-1
dla różnych współczynników uczenia



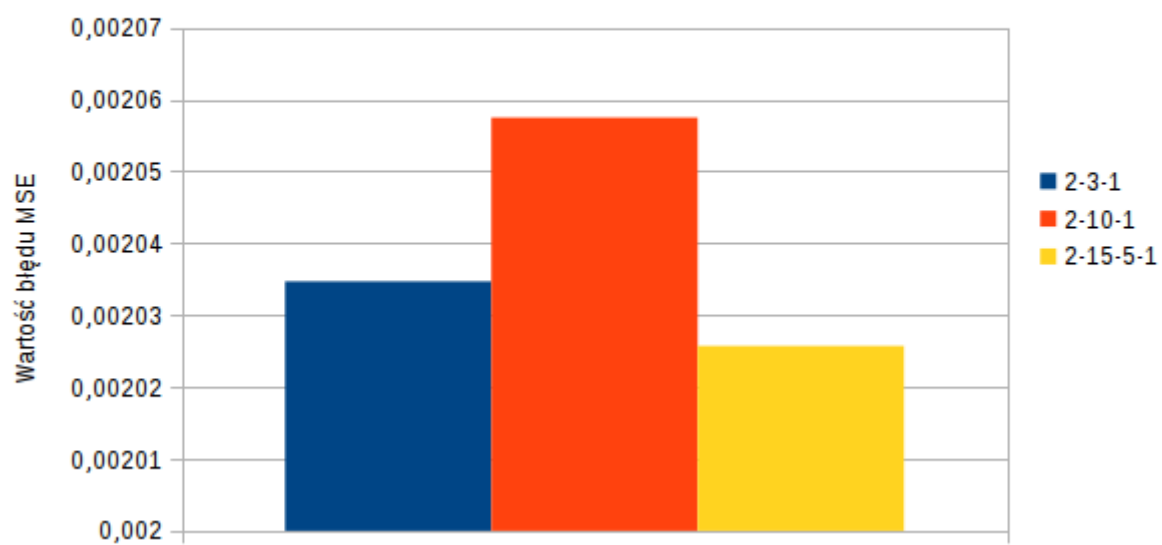
Wartości błędu MSE dla testowania
wszystkich typów sieci dla współczynnika uczenia 0.5



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



Wartości błędu MSE dla testowania
wszystkich typów sieci dla współczynnika uczenia 0.01



3) Analiza i dyskusja błędów uczenia i testowania opracowanej 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 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 sieć 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 kodu oraz zrzuty konfiguracji i wykorzystania programu z opisem.

```
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]
));
    }

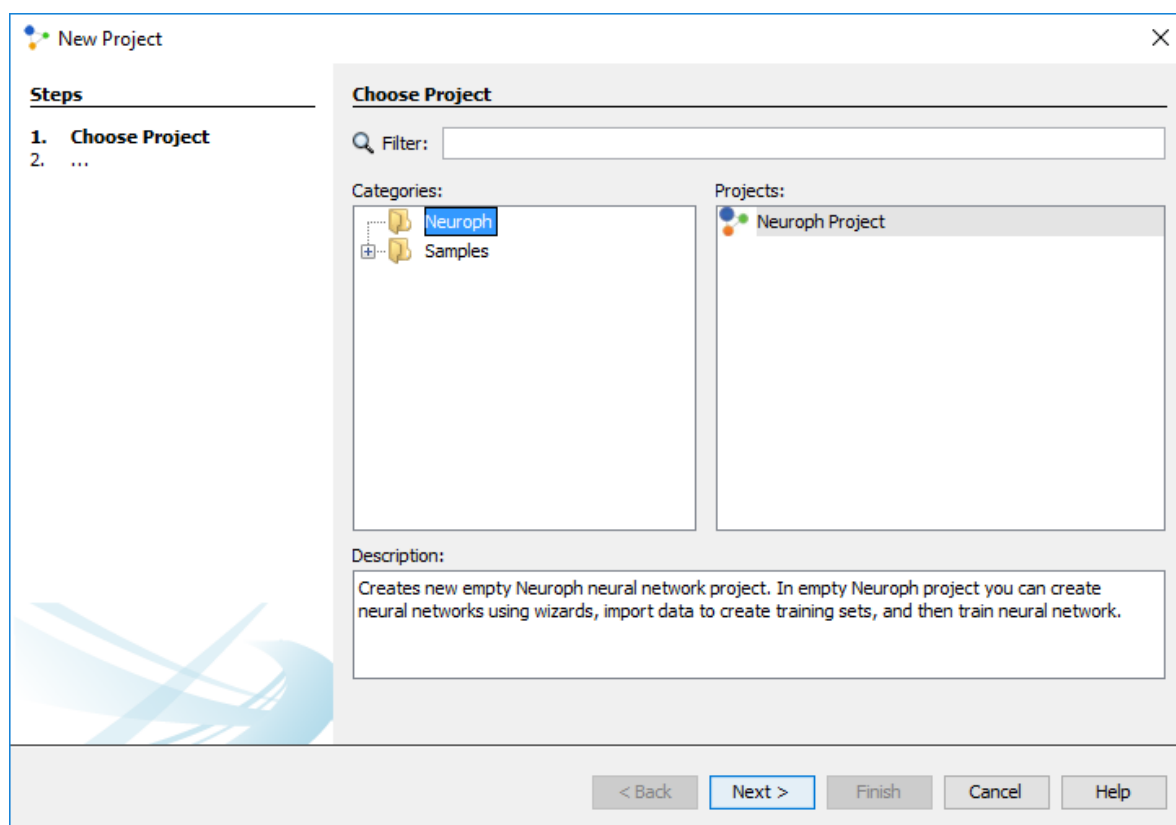
    for ( int i = 0; i < testingSize; i++ ) {
        testingData[i][0] = normalization( 4.0 * rand.nextDouble() - 2.0 );
        testingData[i][1] = normalization( 4.0 * rand.nextDouble() - 2.0 );
        testingY[i] = normalizationRastrigin( calculateRastrigin3D( testingData[i][0], testingData[i][1] )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

//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;
}

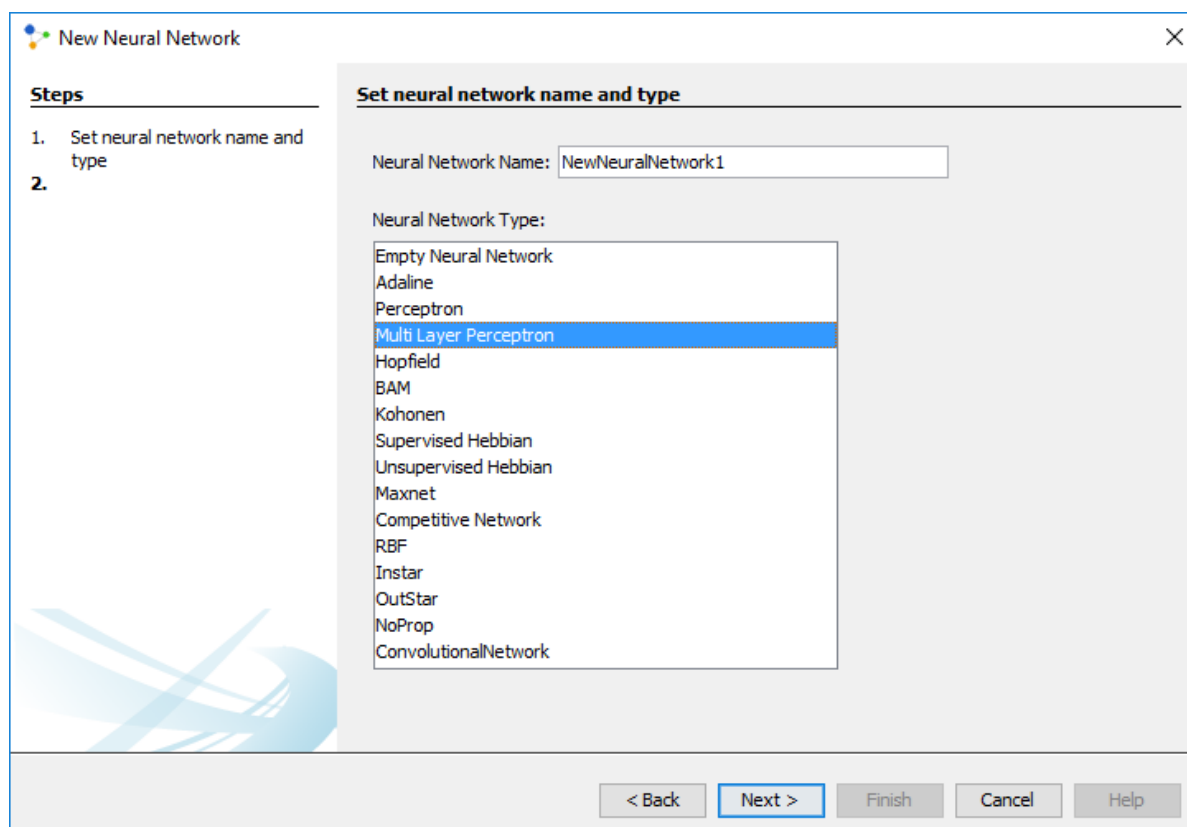
//funkcja normalizująca dane wejściowe "x" i "y" do funkcji Rastrigin
private static double normalization ( double x ) {
    double min = - 2, max = 2;
    double new_min = 0, new_max = 1;
    return ( ( x - min ) / ( max - min ) ) * ( new_max - new_min ) + new_min;
}
}

```

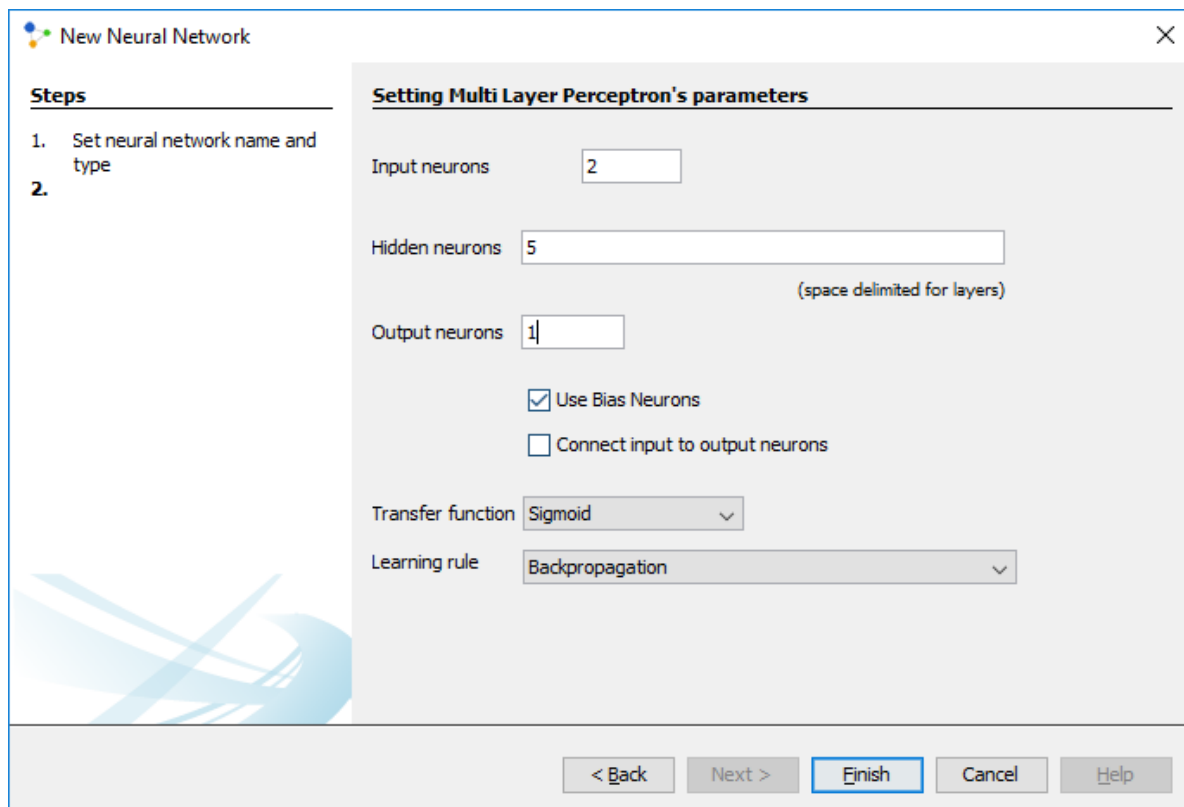

Po otwarciu programu NeurophStudio tworzymy nowy projekt:



Po utworzeniu projektu tworzymy nową sieć. Ważne jest aby wybrać z listy odpowiedni typ sieci.



Następnie wybieramy ilość warstw i neuronów oraz funkcję aktywacji i algorytm uczenia.

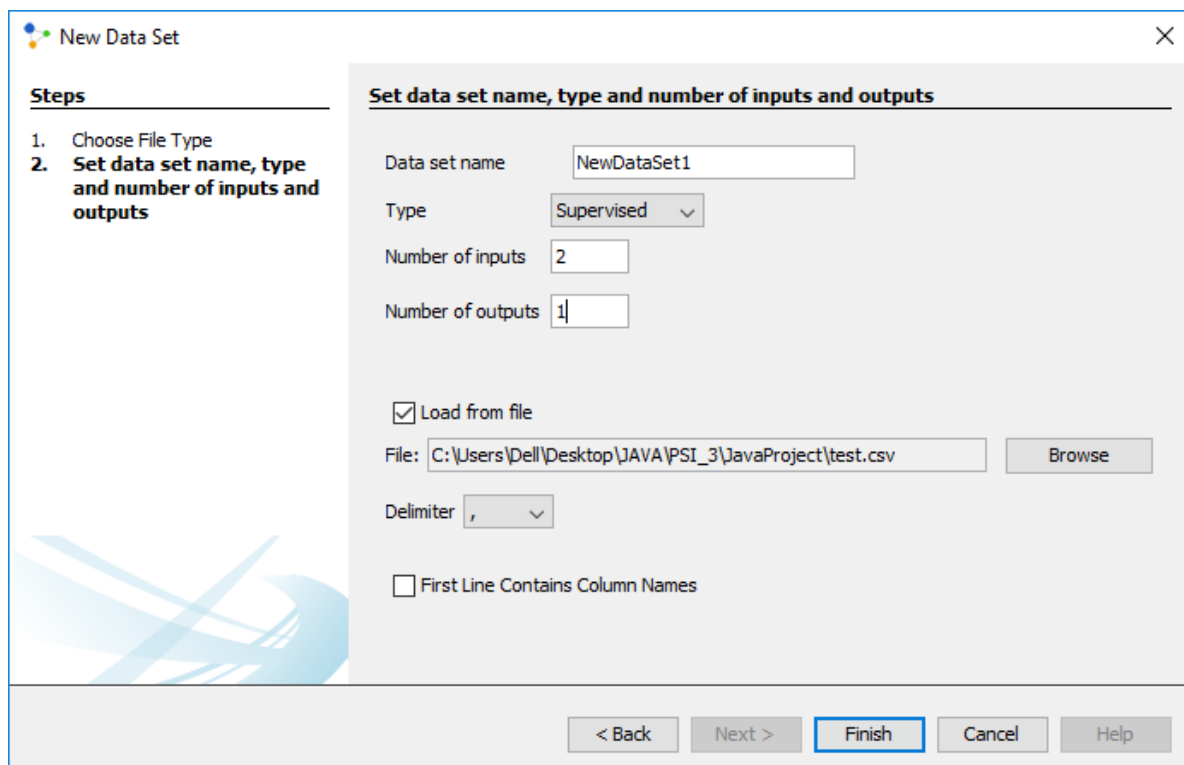


The screenshot shows the 'New Neural Network' dialog box. On the left, under 'Steps', step 2 is highlighted. The main area is titled 'Setting Multi Layer Perceptron's parameters'. It contains the following fields and options:

- Input neurons: 2
- Hidden neurons: 5 (with a note '(space delimited for layers)')
- Output neurons: 1
- ☒ Use Bias Neurons
- ☐ Connect input to output neurons
- Transfer function: Sigmoid (dropdown)
- Learning rule: Backpropagation (dropdown)

At the bottom, there are buttons: '< Back', 'Next >', 'Finish' (highlighted), 'Cancel', and 'Help'.

Pozostało nam jeszcze wczytać dane. W przypadku czytania z pliku jest to banalnie proste.

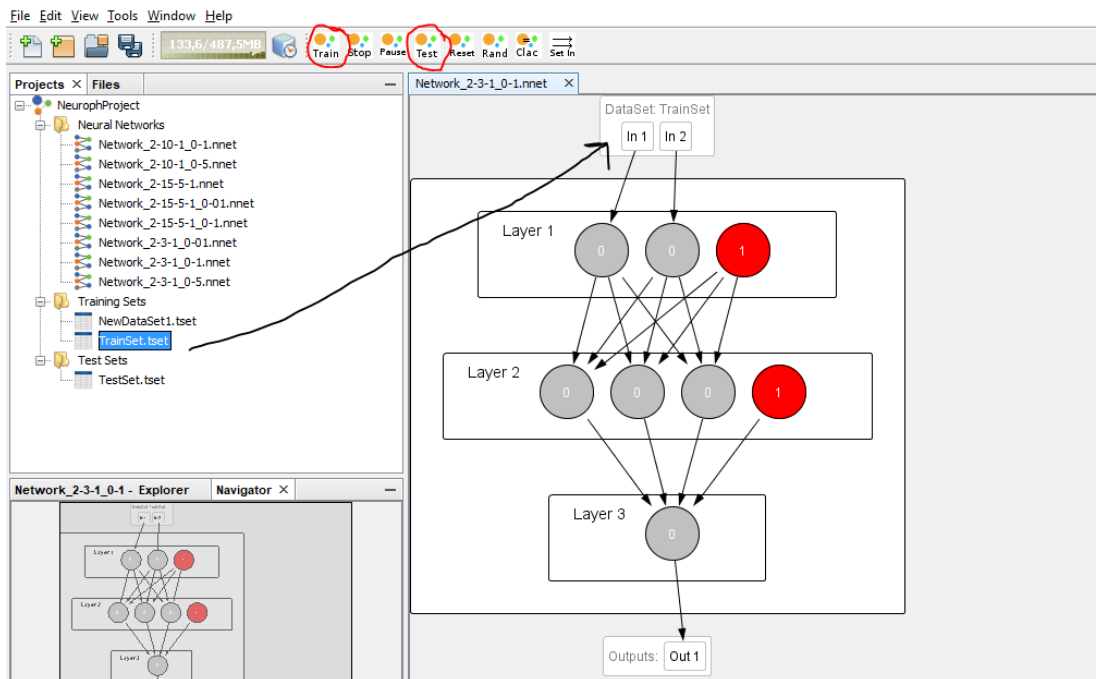


The screenshot shows the 'New Data Set' dialog box. On the left, under 'Steps', step 2 is highlighted. The main area is titled 'Set data set name, type and number of inputs and outputs'. It contains the following fields and options:

- Data set name: NewDataSet1
- Type: Supervised (dropdown)
- Number of inputs: 2
- Number of outputs: 1
- ☒ Load from file
- File: C:\Users\Dell\Desktop\JAVA\PSI_3\JavaProject\test.csv (with a 'Browse' button)
- Delimiter: , (dropdown)
- ☐ First Line Contains Column Names

At the bottom, there are buttons: '< Back', 'Next >', 'Finish' (highlighted), 'Cancel', and 'Help'.

Pozostało nam już tylko uczenie i testowanie. W tym celu najpierw musimy przeciągnąć odpowiednie dane na naszą sieć, którą chcemy testować. Później wybieramy na górnym panelu odpowiedni przycisk w zależności czy chcemy uczyć czy testować.



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.

The screenshot shows the 'Training Dialog' window. It has several sections: 'Stopping Criteria' with 'Max Error' set to 0.001 and an unchecked 'Limit Max Iterations' checkbox; 'Learning Parameters' with 'Learning Rate' set to 0.5 and 'Momentum' set to 0.7; 'Crossvalidation' with an unchecked 'Use Crossvalidation' checkbox, 'Subset count' set to 4, 'Subset distribution (%)' set to 60 20 20, and unchecked checkboxes for 'Allow samples repetition' and 'Save all trained networks'; and 'Options' with a checked 'Display Error Graph' checkbox and a 'Turn off for faster learning' checkbox. At the bottom are 'Train' and 'Close' buttons.

Bibliografia:

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

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

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