Scenariusz 3

Białek Tomasz, gr. 1

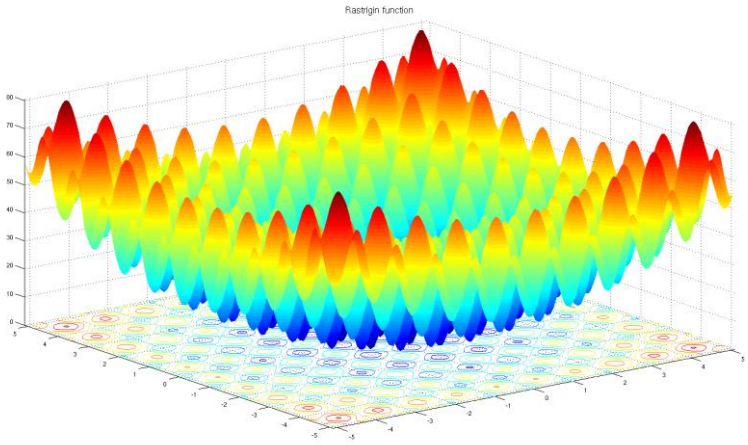
Temat ćwiczenia: Budowa i działanie sieci wielowarstwowej typu feedforward

1. Cel ćwiczenia

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

2. Opis budowy sieci i algorytmów uczenia.

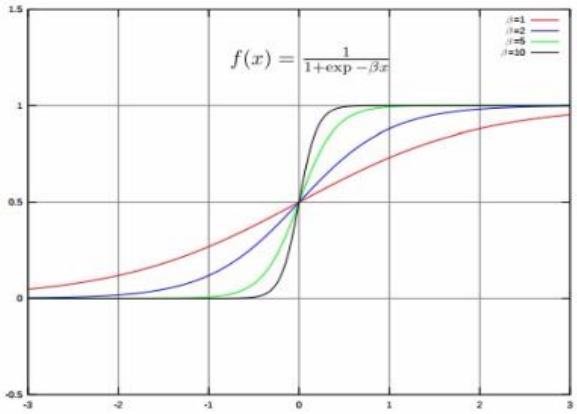
Celem budowanej sieci jest rozpoznawanie funkcji rastrigin. Funkcja ta w naszym przyjmuje za wejście współrzędne x, y z przedziału <-2; 2> i zwraca współrzędną z.
Wzór funkcji rastrigin: $f(x) = 10 * n + \sum_{i=1}^{n}[x_i^2 - 10 * \cos(2\pi x_i)$
Rastrigin ma minimum globalne w punkcie (0, 0, 0).



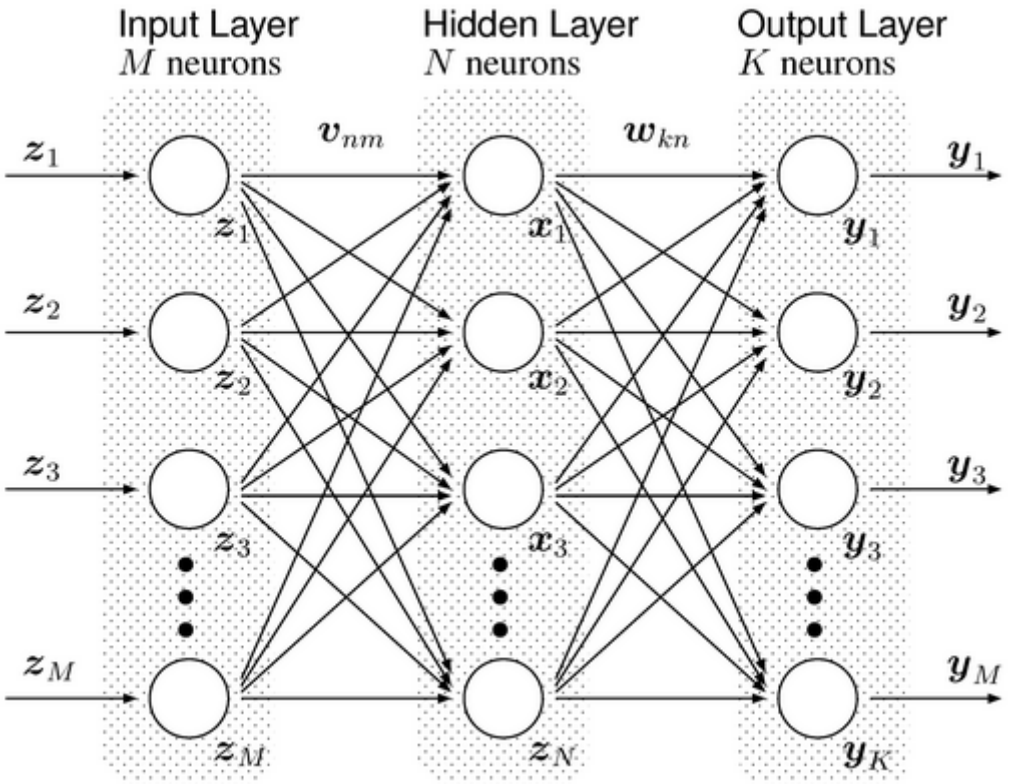
Rys. 1 Funkcja Rastrigin

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 użytwo funkcji sigmoidalnej.

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



Rys. 2 Funkcja sigmoidalna unipolarna z różnymi współczynnikami β



Rys. 3 Ogólny schemat sieci neuronowej wielowarstwowej

Sieć wielowarstwowa składa się z warstwy wejściowej (Input Layer), co najmniej jednej warstwy ukrytej (Hidden Layer) oraz warstwy wyjściowej (Output Layer). 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 seci). 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.

Schemat uczenia sieci wielowarstwowej

1. Aby było możliwe uczenie trzeba najpierw znormalizować dane uczące. Normalizację przeprowadza się według wzoru:

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

2. Wybór współczynnika uczenia $\eta > 0$ oraz $E_{max} > 0$.
3. Wybór początkowych wartości wag z zakresu <0; 1>
4. Ustawienie sumy błędu średniokwadratowego $E = 0$.
5. Aktualizacja wag:

Wzór ogólny:

$$w_{ij}(t+1) = w_{ij}(t) - \eta * \frac{\partial \mathrm{E}}{\partial \mathrm{w}} = w_{ij}(t) + \eta * \delta * \mathrm{x}$$

Zmiana wag z warstwy wejściowej i do warstwy ukrytej j:

$$w_{ij}(t+1) = w_{ij}(t) + \eta * \delta_j * x_i, gdzie\ \delta_j = o_j * (1 - o_j) * \sum_k w_{jk} * \delta_k$$

Zmiana wag z warstwy ukrytej j do warstwy wyjściowej k:

$$w_{ij}(t+1) = w_{ij}(t) + \eta * \delta_k * x_j, gdzie\ \delta_k = (y_{target,k} - y_k) * y_k * (1 - y_k)$$

gdzie:
$\eta - współczynnik\ uczenia$
$\delta_j - błąd\ neuronu\ j$
$o_i - wyjście\ neuronu\ i\ (będące\ wejściem\ neuron\ j)$
$\alpha - \text{momentum}$
6. Obliczenie łącznego błędu epoki:

$$E = E + \frac{1}{2}\sum (d_i - y_i)^2$$

7. Uczenie uważamy za zakończenie jeżeli $E < E_{max}$. W przeciwnym razie zwiększamy epokę uczenia i wracamy do punktu 4.

3. Otrzymane wyniki i ich analiza

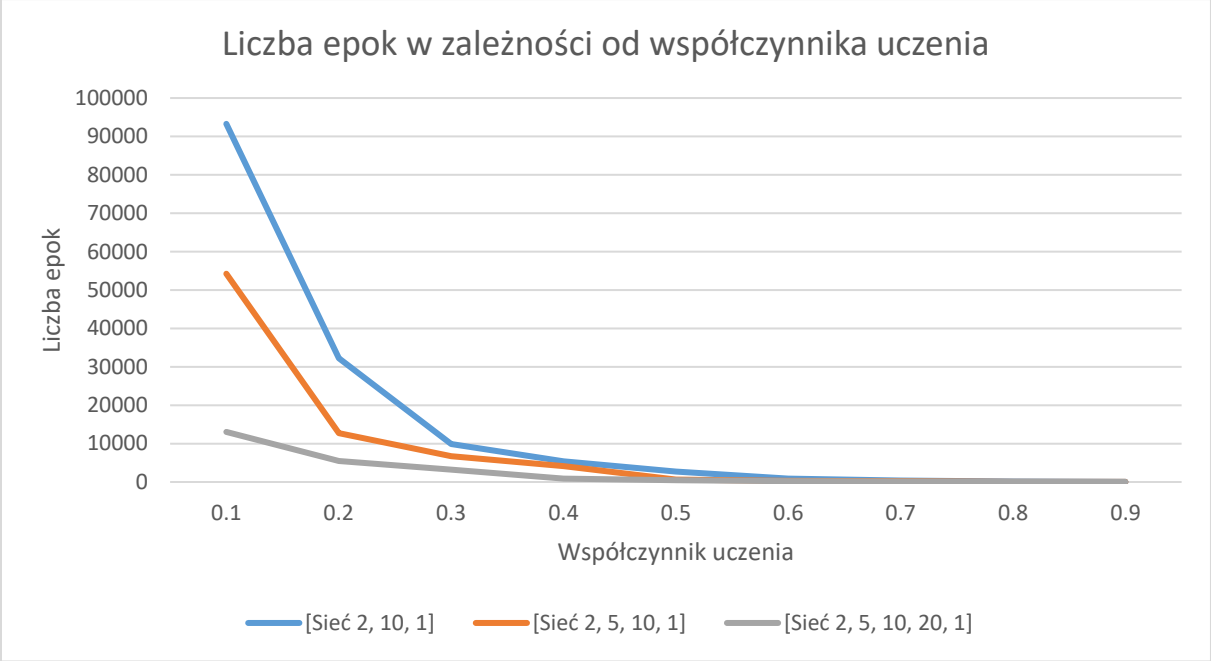
Przykładowe działanie programu:

```
Teaching: MultiLayerPerceptron, LearningData: TrainingData
Using algorithm BackPropagation with configuration:
Max Error: 0.014
Learning Rate: 0.1
Teaching is finished:
Number of iterations: 7470
Total error: 0.013999993632557807
Set 1: x = 0.0 y = 0.0
Desired output: 0.1797752808988764
Output: 0.5873245186766622

Set 2: x = 0.0 y = 0.025000000000000022
Desired output: 0.21392876530900065
Output: 0.5848009526202897

Set 3: x = 0.0 y = 0.050000000000000044
Desired output: 0.31797370912922573
Output: 0.5815332527236604
```

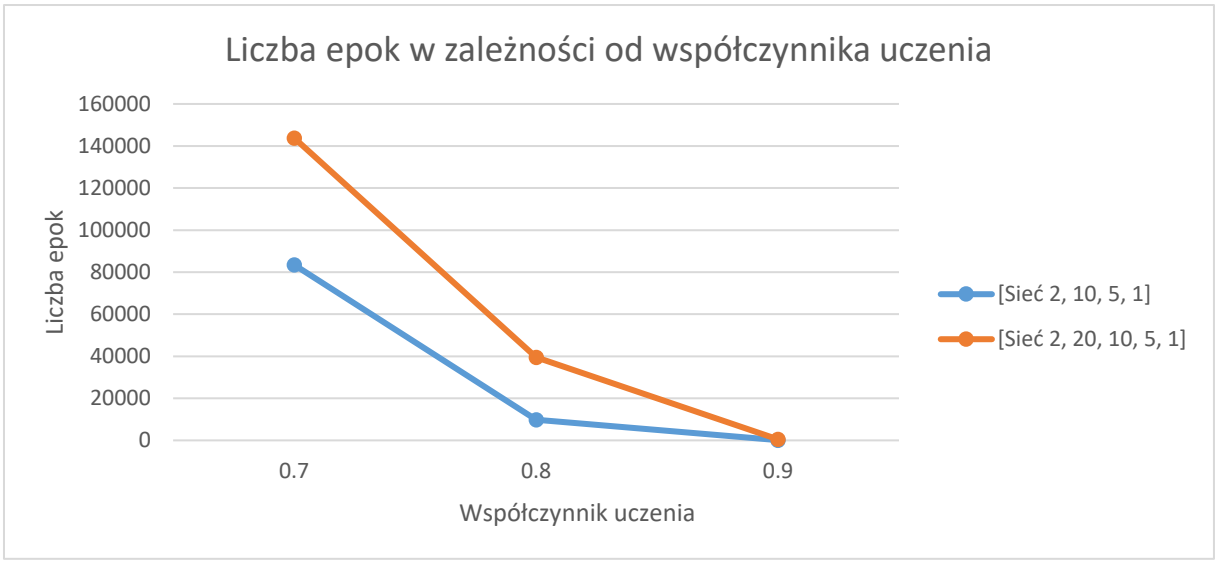
```
Teaching: MultiLayerPerceptron, LearningData: TrainingData
Using algorithm BackPropagation with configuration:
Max Error: 0.015
Learning Rate: 0.05
Teaching is finished:
Number of iterations: 2693
Total error: 0.014999660444118821
```

```
Teaching: MultiLayerPerceptron, LearningData: TrainingData
Using algorithm BackPropagation with configuration:
Max Error: 0.015
Learning Rate: 0.02
Teaching is finished:
Number of iterations: 32209
Total error: 0.01499883107359946
Set 1: x = 0.0 y = 0.0
Desired output: 0.1797752808988764
Output: 0.38115137069566146
```

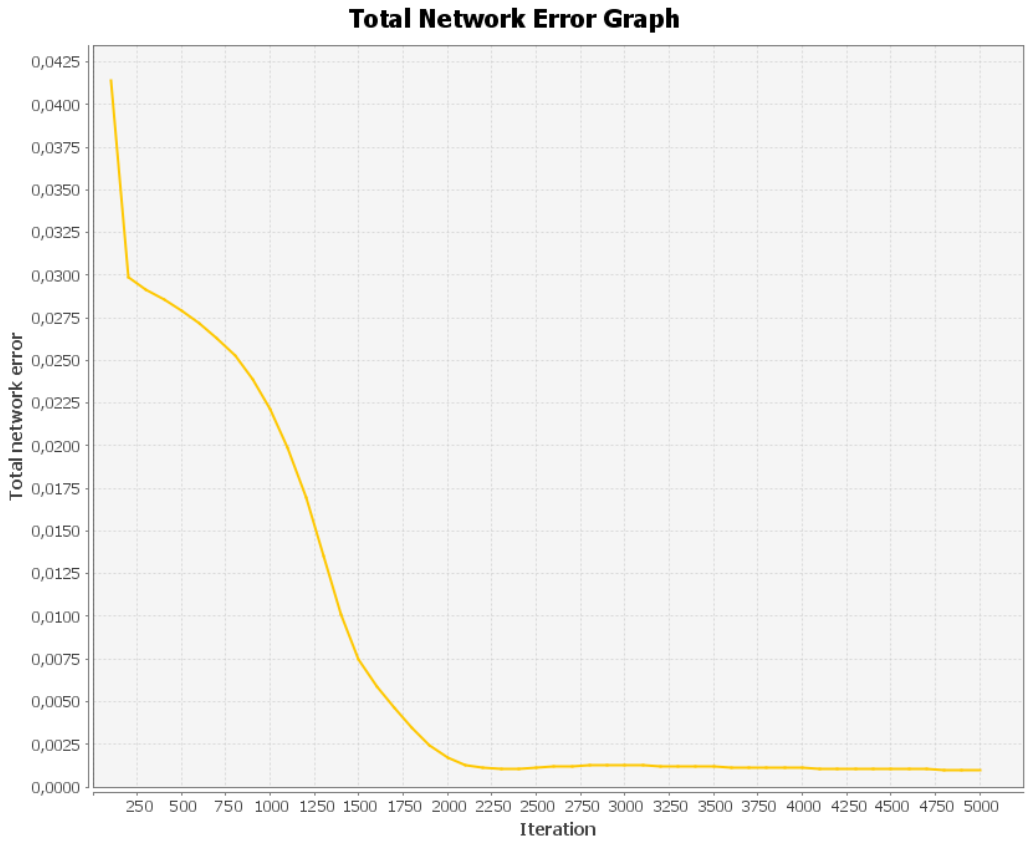
Analiza



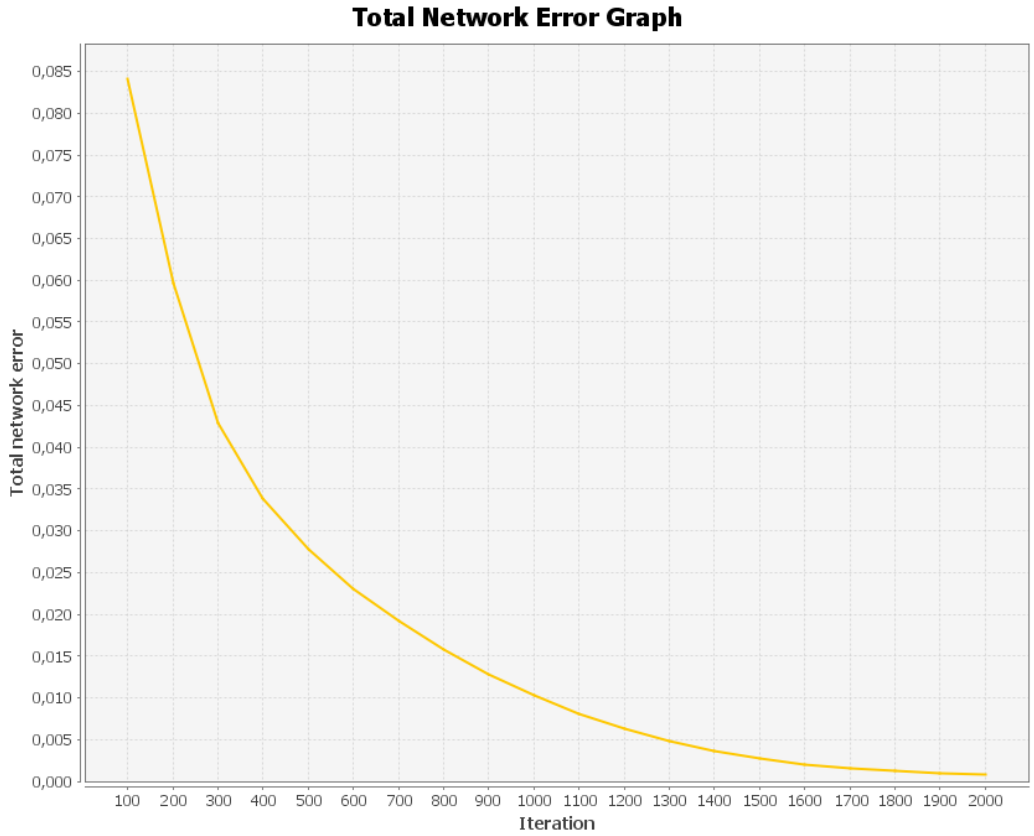
Z powyższego wykresu można odczytać, że na efektywność uczenia ma wpływ współczynnik uczenia. Ponadto sama struktura sieci wielowarstwowej ma wpływ na szybkość uczenia. Im więcej warstw, więcej neuronów, tym sieć uczyła się sprawniej. Widać to przy współczynniku uczenia równym 0.1, gdzie sieć z trzema warstwami ukrytymi potrzebowała dziewięć razy mniej epok aniżeli sieć z jedną warstwą ukrytą. Jednakże bez względu na strukturę sieci, można powiedzieć, że uczenie ze współczynnikiem uczenia mniejszym od 0.5 jest zbyt powolne, żeby było akceptowalne.



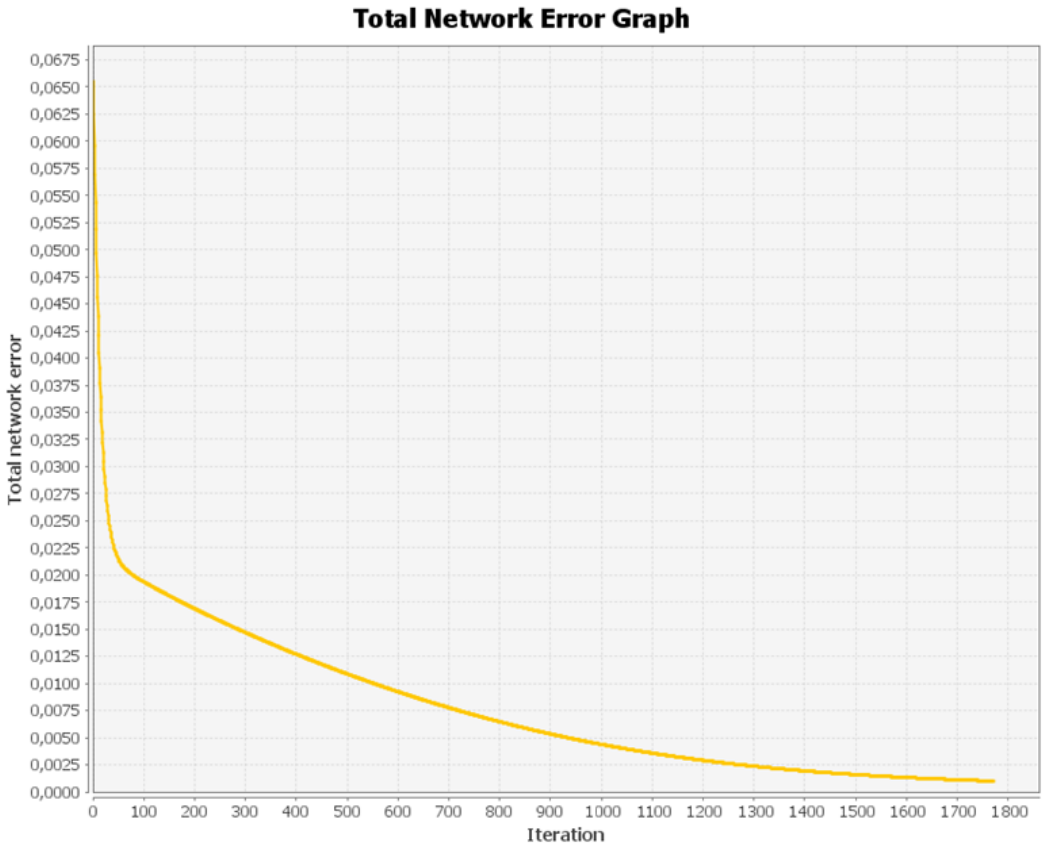
Wykres ten przedstawia liczbę epok potrzebnych, żeby sieć nauczyła się funkcji Rastrigin. Porównując przedstawione struktury: [Sieć 2, 10, 5, 1] oraz [Sieć 2, 20, 10, 5, 1] z ich odpowiednikami (odpowiednio [Sieć 2, 5, 10, 1] oraz [Sieć 2, 5, 10, 20, 1]) można zauważyć, że na efektywność uczenia ma wpływ nie tylko współczynnik uczenia, liczba warstw oraz neuronów, ale także ich rozmieszczenie w danych warstwach. Sieci, które posiadały w początkowych warstwach ukrytych więcej neuronów uczyły się znacznie gorzej aniżeli sieci, które posiadały ich mniejszą ilość. Testy dla mniejszych współczynników uczenia nie zostały przeprowadzone z powodu ogromnej ilości epok potrzebnych do nauczenia.



Błąd średniokwadratowy - Sieć [2, 10, 1]



Błąd średniokwadratowy – Sieć [2, 5, 10, 1]



Błąd średniokwadratowy – Sieć [2, 5, 10, 20, 1]

Powyższe wykresy przedstawiają wartości błędu średniokwadratowego podczas uczenia danych sieci. Bez względu na ich strukturę błąd średniokwadratowy szybko malał w początkowych epokach, szczególnie dla sieci o większej strukturze. Dla [Sieć 2, 5, 10, 1] oraz [Sieć 2, 5, 10, 20, 1] przebieg funkcji błędu średniokwadratowego przebiegał dosyć stabilnie, jednakże dla [Sieć 2, 10, 1] pojawił się nagły uskok przy zmniejszaniu błędu średniokwadratowego, co może być spowodowane małą strukturą tej sieci oraz mniejszą stabilnością przy aktualizacji wag – porównując do większych odpowiedników.

4. Podsumowanie

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. 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. W przypadku sieci wielowarstwowej jego wpływ jest wyraźny, więc lepiej dobierać większe współczynniki uczenia, które spowodują w miarę szybkie nauczenie sieci aniżeli oscylujące w granicy wartości 0.1, gdzie na wytrenowanie sieci trzeba było poczekać kilka, kilkanaście minut.

5. Kod programu

„Main.java”

```
package com.company;

import org.neuroph.core.NeuralNetwork;
import org.neuroph.core.data.DataSet;
import org.neuroph.core.data.DataSetRow;
import org.neuroph.nnet.MultiLayerPerceptron;
import org.neuroph.nnet.learning.BackPropagation;
import org.neuroph.util.TransferFunctionType;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;

public class Main {

    private static double zMAX = Double.MIN_VALUE;
    private static double zMIN = Double.MAX_VALUE;
    private static double xyMAX = 2.0;
    private static double xyMIN = -2.0;

    public static void main(String[] args) throws IOException {
        //uworzeine danych testowych
        DataSet trainingData = new DataSet(2, 1);
        trainingData.setLabel("TrainingData");

        //określenie minimum i maximum funkcji Rastrigin na przedziale <-
2.0; 2.0>
```

```
        double tmp;
        for (double i = -2.0; i <= 2.0; i += 0.1) {
            for (double j = -2.0; j <= 2.0; j += 0.1) {
                tmp = RastriginFunction.rastrigin(i, j);
                if (zMAX <= tmp)
                    zMAX = tmp;
                if (zMIN >= tmp)
                    zMIN = tmp;
            }
        }

        File newFile = new File("learningDataSet.txt");
        FileWriter fileWriter = new FileWriter(newFile);

        //wypisanie danych uczących (przed normalizacją) [40*40 danych]
        double x, y, z;
        for (double i = -2.0; i <= 2.0; i += 0.1)
            for (double j = -2.0; j <= 2.0; j += 0.1) {
                x = i;
                y = j;
                z = RastriginFunction.rastrigin(i, j);
                System.out.println("x = " + i + "\ty = " + j + "\tz = " +
z);
                fileWriter.write(x + ";" + y + ";" + z + "\n");
            }

        fileWriter.close();

        //wczytanie danych wejściowcyh z pliku
        Scanner in = new Scanner(newFile);
        String[] result;
        do {
            String line = in.nextLine();
            result = line.split(";");
            //dane do normalizacji danych wejściowych  i wyjściowych
            System.out.println("Normalization: x = " +
RastriginFunction.normalization(Double.parseDouble(result[0]), xyMIN,
xyMAX)
                    + " y = " +
RastriginFunction.normalization(Double.parseDouble(result[1]), xyMIN,
xyMAX)
                    + " z = " +
RastriginFunction.normalization(Double.parseDouble(result[2]), zMIN,
zMAX));
            //dodanie rzędu danych uczących
            trainingData.addRow(new DataSetRow(new double[]{

RastriginFunction.normalization(Double.parseDouble(result[0]), xyMIN,
xyMAX),

RastriginFunction.normalization(Double.parseDouble(result[1]), xyMIN,
xyMAX)
            }, new
double[]{RastriginFunction.normalization(Double.parseDouble(result[2]),
zMIN, zMAX)}));
        } while (in.hasNext());

        //wybranie metody backPropagation i ustawienie maksymalnego błędu i
współczynnika uczenia
        BackPropagation backPropagation = new BackPropagation();
        backPropagation.setMaxError(0.015);
```

```
        backPropagation.setLearningRate(0.04);

        //utworzenie sieci wielowarstwowej i ustalenie ilości warstw
ukrytych
        MultiLayerPerceptron multiLayerPerceptron = new
MultiLayerPerceptron(TransferFunctionType.SIGMOID, 2, 5, 10, 20, 1);
        multiLayerPerceptron.setLabel("MultiLayerPerceptron");
        multiLayerPerceptron.setLearningRule(backPropagation);

        System.out.println("Teaching: " + multiLayerPerceptron.getLabel() +
", LearningData: " + trainingData.getLabel());
        System.out.println("Using algorithm BackPropagation with
configuration:");
        System.out.println("Max Error: " + backPropagation.getMaxError());
        System.out.println("Learning Rate: " +
backPropagation.getLearningRate());

        //losowanie wstępnych wag i rozpoczęcie uczenia
        multiLayerPerceptron.randomizeWeights();
        multiLayerPerceptron.learn(trainingData);

        System.out.println("Teaching is finished:");
        //wyświetlenie liczby iteracji oraz całkowitego błędu uczenia w
sieci neuronowej
        System.out.println("Number of iterations: " +
backPropagation.getCurrentIteration());
        System.out.println("Total error: " +
backPropagation.getErrorFunction().getTotalError());
        //zapis sieci do pliku .nnet
        multiLayerPerceptron.save("mlp.nnet");

        //testowanie utworzonej sieci
        NeuralNetwork neuralNetwork =
NeuralNetwork.createFromFile("mlp.nnet");
        int iteration = 1;
        for (DataSetRow dataSetRow : trainingData.getRows()) {
            double[] input = dataSetRow.getInput();
            System.out.println("Set " + iteration + ": x = " + input[0] + "
y = " + input[1]);

            double[] desiredOutput = dataSetRow.getDesiredOutput();
            neuralNetwork.setInput(dataSetRow.getInput());
            neuralNetwork.calculate();
            double[] output = neuralNetwork.getOutput();
            System.out.println("Desired output: " + desiredOutput[0]);
            System.out.println("Output: " + output[0]);
            System.out.println();
            iteration++;
        }
    }
}
```

„RastriginFunction.java”

```
package com.company;

public class RastriginFunction {
    //obliczenie wartości funkcji rastrigin na osi z
    static double rastrigin(double x1, double x2) {
        return 10 * 2 +
                Math.pow(x1, 2) - 10 * Math.cos(2 * Math.PI * x1) +
                Math.pow(x2, 2) - 10 * Math.cos(2 * Math.PI * x2);
```

```
    }

    //normalizacja do przedziału <0; 1>
    static double normalization (double number, double rangeA, double
rangeB){
        return ((number - rangeA)/(rangeB - rangeA)) * (1.0 - (0.0)) +
(0.0);
    }
}
```