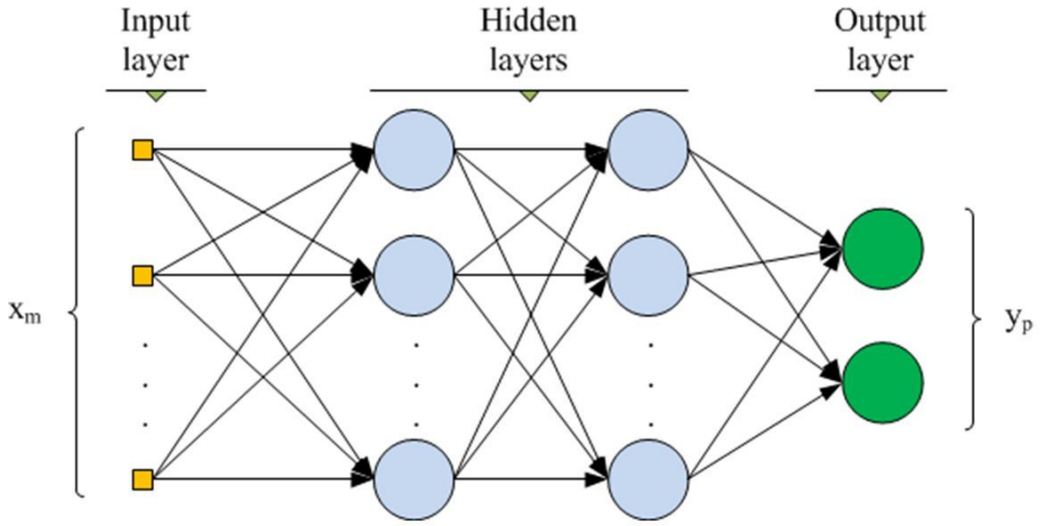
Imię Nazwisko Grupa Michał Słowikowski Gr 4	Temat Scenariusz 3	Data 24.11.2017r.

Celem ćwiczenia była budowa sieci wielowarstwowej z użyciem algorytmu wstecznej propagacji błędu oraz naukę aproksymacji funkcji Rastrigin.

Syntetyczny opis algorytmu uczenia

Do ćwiczenia wykorzystałem sieć typu feedforward zbudowaną z różnej ilości warstw i znajdujących się w niej neuronów z sigmoidalną funkcją aktywacji.



Jako funkcję aktywacji wykorzystałem unipolarną funkcję sigmoidalną:

$$f(x) = \frac{1}{1 + e^{-\beta x}}$$

Jako algorytmu użyłem Backpropagation - wstecznej propagacji błędu.
Błąd potrzebny do korekcji wag dla każdego neuronu w ostatniej warstwie obliczałem za pomocą wzoru:

$$\frac{\partial E}{\partial W_{jk}} = \mathcal{O}_j \delta_k$$

$$\delta_k = \mathcal{O}_k(1 - \mathcal{O}_k)(\mathcal{O}_k - t_k)$$

Natomiast każdą wcześniejszą

$$\frac{\partial E}{\partial W_{ij}} = \mathcal{O}_i \delta_j$$

$$\delta_j = \mathcal{O}_j(1 - \mathcal{O}_j) \sum_{k \in K} \delta_k W_{jk}$$

Korekcja wag ze wzoru:

$$\Delta W = -\eta \delta_\ell \mathcal{O}_{\ell-1}$$
$$\Delta \theta = \eta \delta_\ell$$

, gdzie η - learning rate, δ - błąd na neuronie, O wynik z poprzedniej warstwy

Łączny błąd liczyłem z MSE, czyli

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

Dane

Dane potrzebne do zadania wyliczałem z funkcji podanej przez prowadzącego, którą można znaleźć pod tym linkiem: https://en.wikipedia.org/wiki/Rastrigin_function jako przedział przyjąłem dane dla x i y należących do przedziału [-2,2] .

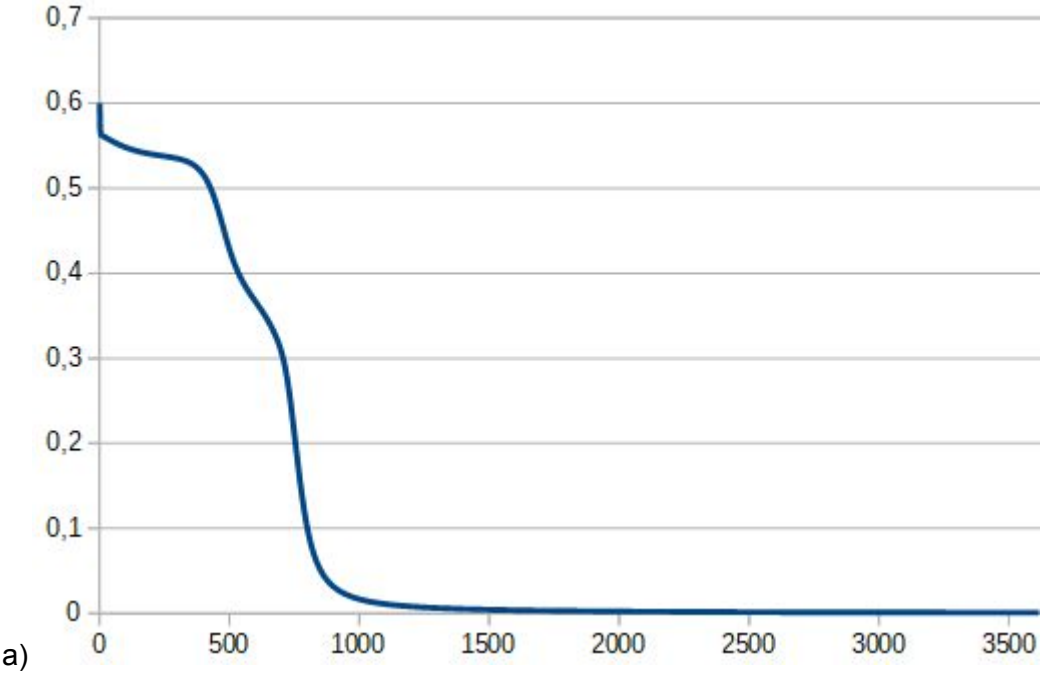
Wygenerowałem łącznie 1600 danych z czego 400 posłużyło mi do testowania sieci. Każdy wynik przed porównaniem przeskalowałem, żeby znajdowała się w przedziale od 0 do 1 (tak jak signum unipolarne). Przed skalowaniem obliczyłem, że największa wartość jaką przyjmuje funkcja to ~40 a najmniejsza to 0. Dla bezpieczeństwa przyjąłem pierwotną skalę od 0 do 50.

Sieci jakich użyłem to (ze względu na ilość neuronów znajdujących się w warstwie
a)10-3-1
b)2-5-1
c)4-3-3-1

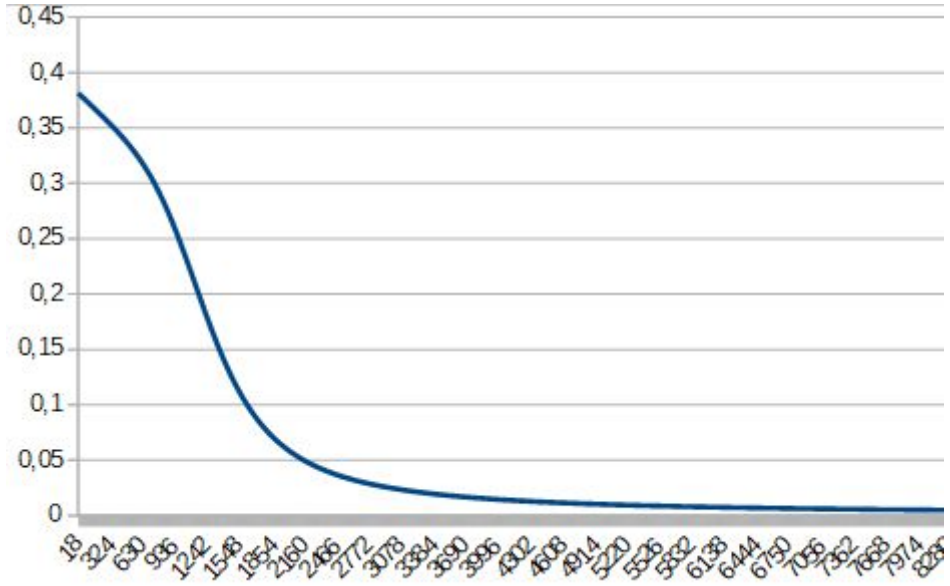
Do każdy zestaw przetestowałem z współczynnikami uczenia 0.1 i 0.01.

Wyniki

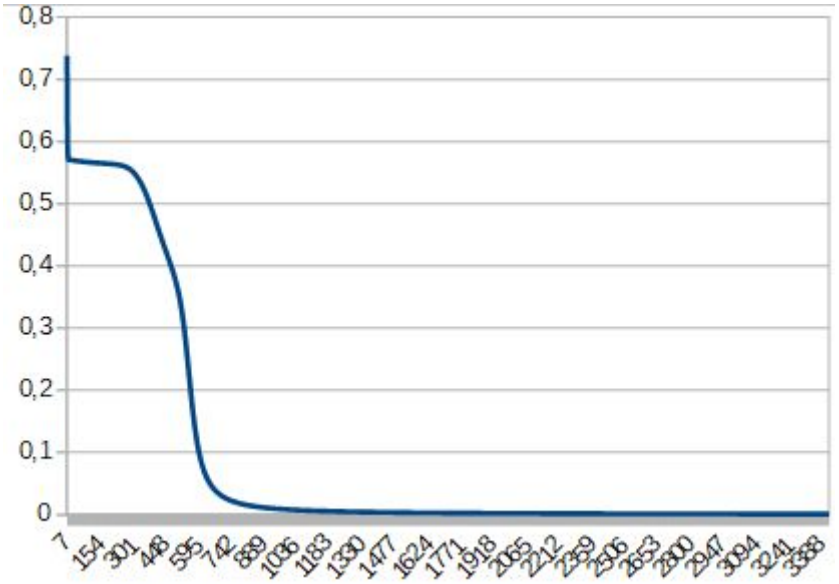
a)

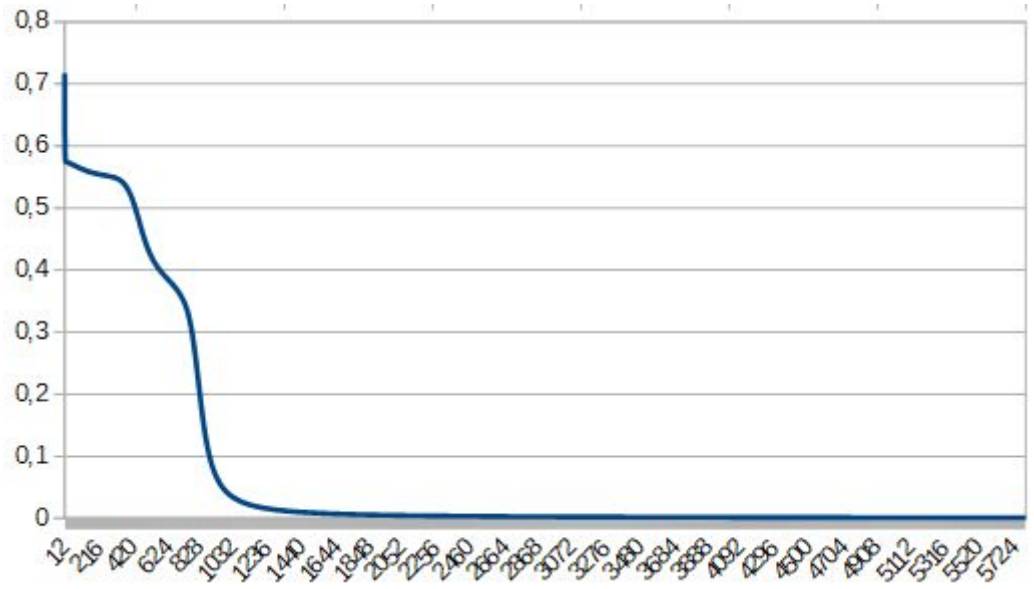

0,7
0,6
0,5
0,4
0,3
0,2
0,1
0
0
500
1000
1500
2000
2500
3000
3500

a)

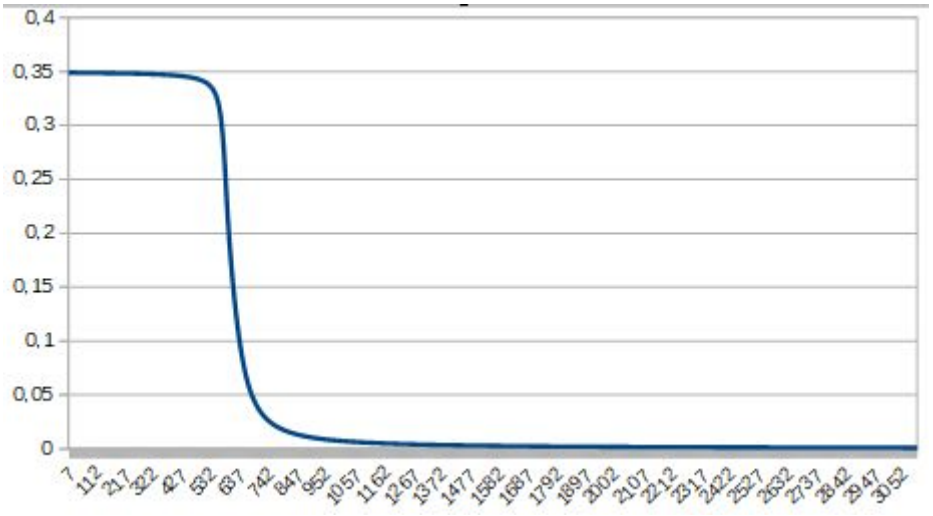

0,45
0,4
0,35
0,3
0,25
0,2
0,15
0,1
0,05
0
18
324
630
936
1242
1548
1854
2160
2466
2772
3078
3384
3690
3996
4302
4608
4914
5220
5526
5832
6138
6444
6750
7056
7362
7668
7974

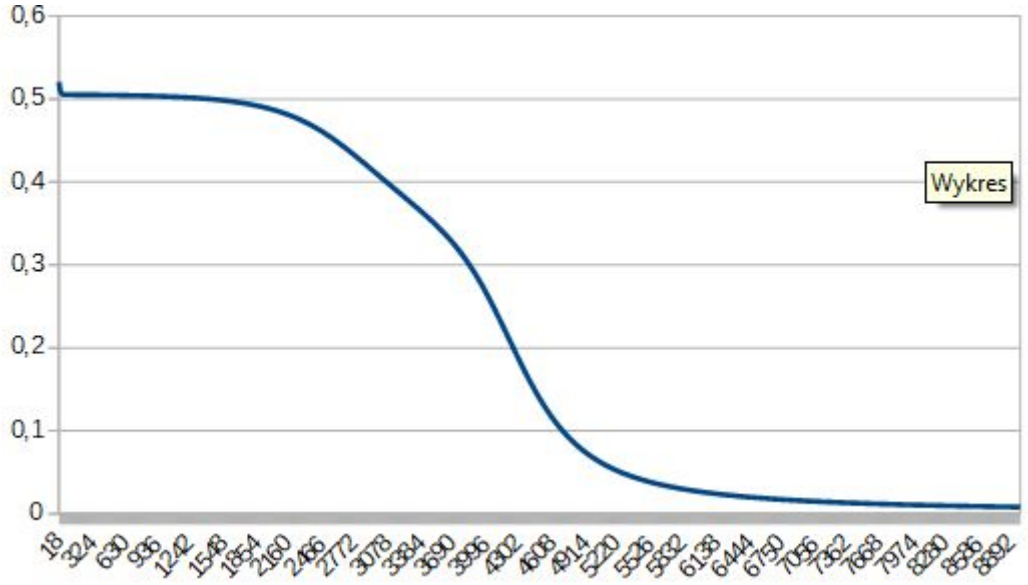
b)


0,8
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0
7
154
301
448
595
742
889
1036
1183
1330
1477
1624
1771
1918
2065
2212
2359
2506
2653
2800
2947
3094
3241
3388


0,8
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0
12
216
420
624
828
1032
1236
1440
1644
1848
2052
2256
2460
2664
2868
3072
3276
3480
3684
3888
4092
4296
4500
4704
4908
5112
5316
5520
5724

c)


0,4
0,35
0,3
0,25
0,2
0,15
0,1
0,05
0
7
112
217
322
427
532
637
742
847
952
1057
1162
1267
1372
1477
1582
1687
1792
1897
2002
2107
2212
2317
2422
2527
2632
2737
2842
2947
3052



Testowanie:

Podczas testowania wyliczałem średni błąd dla wartości przewidywanej i otrzymanej.

a) 0.256%
   0.245%
b) 0.373%
   0.285%
c) 0.265%
   0.254%

Analiza wyników:

Tak jak w poprzednich ćwiczeniach widać wpływ współczynnika uczenia na długość trwania tego procesu. Jak widać z wykresów wygląd funkcji zależności błędu od epoki zależy od wyglądu sieci (ilości warstw oraz ilości neuronów w tych warstwach). Ciężko określić, która z sieci wypadła najlepiej. Można jednak zauważyć zależność, że wraz z niższym współczynnikiem wykres staje się gładszy. W porównaniu do poprzednich ćwiczeń nauka sieci trwała dłużej, jednak bez tego nie bylibyśmy w stanie przewidywać skomplikowanych zjawisk.

Wnioski:

- Współczynnik uczenia ma wpływ na szybkość uczenia
- Stosunkowo wysoki błąd może wynikać z niewystarczającej ilości danych uczących lub źle obranej skali
- Wielowarstwowa sieć neuronowa pozwala aproksymować skomplikowane funkcje

Listening Kodu:

```
namespace Scen2ver2
{
    class Program
    {
        static void Main(string[] args)
        {
            int[,] struktura = new int[3, 2] { { 10, 2 }, { 3,10}, { 1, 3 } };
            double learningRate = 0.1;
            /*Console.WriteLine("Wygląd struktury: ");
            for(int i = 0; i < struktura.GetLength(0); i++)
            {
                Console.WriteLine("Warstwa nr: " + i + " neurony: " + struktura[i, 0] + " input: " + struktura[i, 1]);
            }*/
            double[] input = new double[2] { 1, 2 };
                 Trainer t = new Trainer(struktura, learningRate);
                 t.PrintNeuralNetwork();
                 t.Learn();
                 t.Test();



            Console.WriteLine(t.GetOutput(input));

            Console.ReadLine();
        }

    }
}
```

```
namespace Scen2ver2
{
    class Neuron
    {
        private double BIAS = 1;
        private double[] weights;

        public Neuron(double[] weights)
        {
            this.weights = new double[weights.Length];
            Array.Copy(weights, this.weights, weights.Length);
        }

        public double GetResult(double[] input)
        {
            double sum = InputSummary(input);
            return PerceptronActivation(sum);
        }
        public void Learn(double[] input, double error, double lr)
        {
            for (int i = 0; i < input.Length; i++)
                weights[i] += error * lr * input[i];

            weights[input.Length] += lr * error;
        }
        public double GetWeight(int numberOfInput)
        {
            return weights[numberOfInput];
        }
        public void PrintWeights()
        {
            for (int i = 0; i < weights.Length; i++)
            {
                if (i != weights.Length - 1)
                    Console.Write("Weight " + i);
                else
                    Console.Write("BIAS ");
                Console.WriteLine(" : " + weights[i]);
            }
        }
        public double InputSummary(double[] input)
        {
            double sum = 0;
            for (int i = 0; i < input.Length; i++)
            {
                sum += weights[i] * input[i];
            }

            sum += weights[input.Length];

            return sum;
        }
        protected double PerceptronActivation(double sum)
        {
            return 1 / (1 + Math.Exp(-sum));
        }
        protected double Derive(double x)
        {
            return (1 - PerceptronActivation(x)) * PerceptronActivation(x);
        }
    }
}
```

```
    public double[] CalculateOutput(double[] input)
    {
        double[] result = new double[neurons.Length];
        for(int i = 0; i < neurons.Length; i++)
        {
            result[i] = neurons[i].GetResult(input);
        }
        Array.Copy(result, lastOutput, lastOutput.Length);
        return result;
    }

    public double[] GetWeights(int neuronNumber)
    {
        double[] weights = new double[neurons.Length];
        for(int i = 0; i < weights.Length; i++)
        {
            weights[i] = neurons[i].GetWeight(neuronNumber);
        }

        return weights;
    }
    public void CalculateNewWeights(double[] input)
    {
        for(int i = 0; i < neurons.Length; i++)
        {
            neurons[i].Learn(input, errors[i], learningRate);
        }
        input = lastOutput;
    }
}
```

```
namespace Scen2ver2
{
    class Layer
    {
        private Neuron[] neurons;
        private double[] errors;
        private double learningRate;
        public double[] lastOutput;
        public Layer(int numberOfNeurons, int numberOfInputs, double learningRate)
        {
            this.learningRate = learningRate;
            lastOutput = new double[numberOfNeurons];
            neurons = new Neuron[numberOfNeurons];
            errors = new double[numberOfNeurons];
            for(int i = 0; i < numberOfNeurons; i++)
            {
                errors[i] = 0.0;
            }
            Random r = new Random();
            double[] weights = new double[numberOfInputs + 1];

            for (int i = 0; i < numberOfNeurons; i++)
            {
                for (int j = 0; j < weights.Length; j++)
                {
                    weights[j] = r.NextDouble();
                }
                neurons[i] = new Neuron(weights);
            }
        }

        public void PrintLayer()
        {
            for(int i = 0; i < neurons.Length; i++)
            {
                Console.WriteLine("Neuron " + i);
                neurons[i].PrintWeights();
            }
        }

        public double[] CalculateLastError(double[] outputError)
        {
            for (int i = 0; i < errors.Length; i++)
            {
                errors[i] = lastOutput[i]*(1-lastOutput[i])*outputError[i];
            }
            return errors;
        }
        public double[] CalculateError(double[] errorsFromNexLayer, Layer nextLayer)
        {
            for(int i = 0; i < errors.Length; i++)
            {
                errors[i] = 0.0;
                for(int j = 0; j < errorsFromNexLayer.Length; j++)
                {
                    errors[i] = errorsFromNexLayer[j] * nextLayer.GetWeights(i)[j];
                }

                errors[i] *= lastOutput[i]*(1-lastOutput[i]);
            }
            return errors;
        }
```

```
public void Learn()
{
    double error;
    double totalError;
    int counter = 0;
    double output;
    double expected;
    double pirvE;
    do
    {
        totalError = 0.0;
        for (double x = -2; x <= 2; x += 0.2)
        {
            for (double y = -2; y <= 2; y += 0.2)
            {
                output = 0.0;
                expected = (RastrignsProvider.CalculateResult(x, y) + 50) / 100;
                double[] input = new double[] { x, y };
                output = GetOutput(input);
                error = expected - output;
                // Console.WriteLine("True Expected: " + (expected*80-40) + " Expected: " + expected + " Got: " + output + " Error: " + error);
                totalError += Math.Pow(error, 2) / 2;
                BackPropagation(error, input);
            }
        }
        counter++;
        pirvE = totalError;
        Console.WriteLine(totalError + "\t" + counter);
        //  Console.WriteLine("-----------------------------------------------------------------------------------------");
    } while (totalError > 0.001 && counter < Max);
    Console.WriteLine(counter);
    Console.ReadLine();
}

public void Test()...

public void BackPropagation(double error, double[] input)
{
    double[] errorsFromNextLayer = new double[] { error };

    errorsFromNextLayer = network[network.Length - 1].CalculateLastError(errorsFromNextLayer);

    for (int i = network.Length - 2; i >= 0; i--)
    {
        errorsFromNextLayer = network[i].CalculateError(errorsFromNextLayer, network[i + 1]);
    }

    CalculateNewWeights(input);
}

private void CalculateNewWeights(double[] input)
{
    double[] result = new double[input.Length];
    Array.Copy(input, result, input.Length);
    for (int i = 0; i < network.Length; i++)
    {
        network[i].CalculateNewWeights(result);
        result = network[i].lastOutput;
    }
}
```

```
namespace Scen2ver2
{
    class Trainer
    {
        private Layer[] network;
        private const int Max = 100000;

        public Trainer(int[,] neuralStructure, double learningRate)
        {
            network = new Layer[neuralStructure.GetLength(0)];
            for (int i = 0; i < neuralStructure.GetLength(0); i++)
            {
                network[i] = new Layer(neuralStructure[i, 0], neuralStructure[i, 1], learningRate);
            }
        }

        public void PrintNeuralNetwork()
        {
            int i = 1;
            foreach (Layer l in network)
            {
                Console.WriteLine("------------------- Warstwa " + i + " -------------------");
                l.PrintLayer();
                i++;
            }
        }

        public double GetOutput(double[] input)
        {
            double[] result = new double[input.Length];
            Array.Copy(input, result, input.Length);
            double output = 0.0;
            for (int i = 0; i < network.Length; i++)
            {
                result = network[i].CalculateOutput(result);
            }
            for (int i = 0; i < result.Length; i++)
                output += result[i];

            return output;
        }

        public void Learn()
```

```
namespace Scen2ver2
{
    class RastrignsProvider
    {
        public static double CalculateResult(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));
        }

        private RastrignsProvider()
        {

        }
    }
}
```