Marcin Damek
Inżynieria obliczeniowa, gr 1
Nr. albumu 285952

Sprawozdanie numer 2

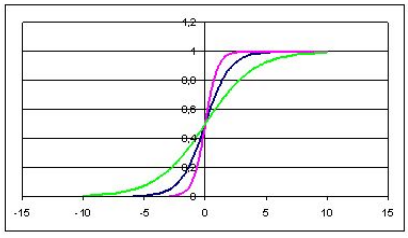
Budowa i działanie sieci jednowarstwowej

Celem ćwiczenia jest poznanie budowy i działania jednowarstwowych sieci neuronowych oraz uczenie rozpoznawania wielkości liter.

Sieć neuronowa (*sztuczna sieć neuronowa*) – ogólna nazwa struktur matematycznych i ich programowych lub sprzętowych modeli, realizujących obliczenia lub przetwarzanie sygnałów poprzez rzędy elementów, zwanych sztucznymi neuronami, wykonujących pewną podstawową operację na swoim wejściu. Oryginalną inspiracją takiej struktury była budowa naturalnych neuronów, łączących je synaps, oraz układów nerwowych, w szczególności mózgu.

Celem budowanej sieci jest rozpoznawanie wielkości liter. Do tego celu utworzony został zestaw liter (10 dużych i 10 małych), które są reprezentowane w postaci dwuwymiarowej tablicy 5x7 pikseli dla jednej litery.
Ćwiczenie wykonałem poprzez swoją implementację w języku C# przy pomocy uczenia nadzorowanego.
Wykorzystałem model sieci Adaline i DeltaRule.

Duże litery których użyłem w ćwiczeniu: ABCDEFGHIJ (Poniżej w pierwszej kolumnie są ich reprezentacje)
Małe litery których użyłem w ćwiczeniu: abcdefghij (Poniżej w drugiej kolumnie są ich reprezentacje)
Do nauki użyłem 20 tablic po 35 znaków każda, tzn w każdym znaku były dwie możliwości tj. 1 lub 0. Sieć DeltaRule używa funkcji sigmoidalnej unipolarnej ze stałym współczynnikiem β = 0,5 jako funkcji aktywacji.



$$f_\alpha(s) = \frac{1}{1+e^{-\alpha s}}$$

$\alpha = 2.0$

$\alpha = 1.0$

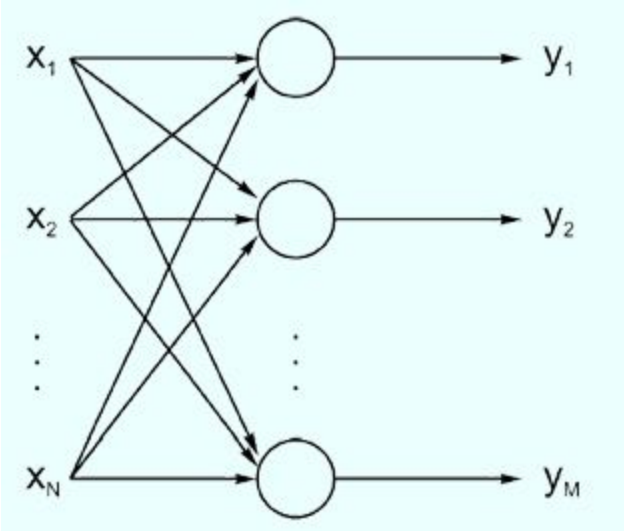
$\alpha = 0.5$

$$\lim_{\alpha \to 0} f_\alpha(s) = 0.5 \qquad \lim_{\alpha \to +\infty} f_\alpha(s) = \begin{cases} 1 & \Leftrightarrow \quad s > 0 \\ 0.5 & \Leftrightarrow \quad s = 0 \\ 0 & \Leftrightarrow \quad s < 0 \end{cases}$$

A	0 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 1 1 1 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1	a	0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 1 0 1 1 1 1 1 0 0 0 1 0 1 1 1 1
B	1 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 1 1 1 0	b	0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 1 1 0 0 1 0 0 1 0 1 1 1 0 0
C	0 1 1 1 0 1 0 0 0 1 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 1 1 1 0	c	0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 0 0 0 1 1 0 0 0 0 1 0 0 0 1 0 1 1 1 0
D	1 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 1 1 1 0	d	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 1 1 0 1 0 0 1 0 0 1 1 1
E	1 1 1 1 1 1 0 0 0 0 1 0 0 0 0 1 1 1 1 0 1 0 0 0 0 1 0 0 0 0 1 1 1 1 1	e	0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 0 0 0 1 1 1 1 1 0 1 0 0 0 0 0 1 1 1 0

F	1 1 1 1 1 1 0 0 0 0 1 0 0 0 0 1 1 1 1 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0	f	0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 0 0 0 1 1 1 0 0 0 1 0 0 0 0 1 0 0 0
G	1 1 1 1 1 1 0 0 0 1 1 0 0 0 0 1 0 1 1 1 1 0 0 0 1 1 0 0 0 1 0 1 1 1 0	g	0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 1 0 1 1 1 1 0 0 0 0 1 1 1 1 1 0
H	1 0 0 0 1 1 0 0 0 1 1 0 0 0 1 1 1 1 1 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1	h	0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 1 1 1 0 1 0 0 0 1 1 0 0 0 1
I	0 1 1 1 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 1 1 0	i	0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 1 0
J	1 1 1 1 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 1 0 0 0 1 0 1 1 1 0	j	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 0 0 0 0 1 0 1 0 0 1 0 0 1 1 0

Sieć neuronowa jednowarstwowa



Uczenie sieci neuronowej

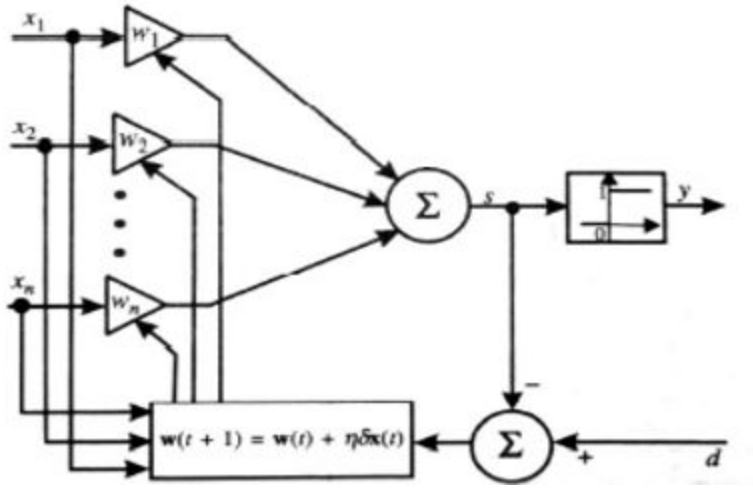
Sieci neuronowe mają zdolność do uczenia się, czyli zdolność do samodzielnego dostosowywania współczynników wagowych. czyli inaczej mówiąc uczenie sieci jest to wymuszeni na niej określonego zareagowania na sygnały wejściowe. Dzięki temu mówi sie ze mają one właśnie charakter AI, bo potrafią samodzielnie dostosowana sie do zmieniających się do warunków. Celem uczenia jest taki dobór wag w poszczególnych neuronach aby sieci mogła rozwiązywać stawiane przed nią problemy.

Reguła delta

Reguła delta jest regułą uczenia z nauczycielem. Polega ona na tym, że każdy neuron po otrzymaniu na swoich wejściach określone sygnały (z wejść sieci albo od innych neuronów, stanowiących wcześniejsze piętra przetwarzania informacji) wyznacza swój sygnał wyjściowy wykorzystując posiadaną wiedzę w postaci wcześniej ustalonych wartości współczynników wzmocnienia (wag) wszystkich wejść oraz (ewentualnie) progu. Sposoby wyznaczania przez neurony wartości sygnałów wyjściowych na podstawie sygnałów wejściowych omówione zostały dokładniej w poprzednim rozdziale. Wartość sygnału wyjściowego, wyznaczonego przez neuron na danym kroku procesu uczenia porównywana jest z odpowiedzią wzorcową podaną przez nauczyciela w ciągu uczącym. Jeśli występuje rozbieżność - neuron wyznacza różnicę pomiędzy swoim sygnałem wyjściowym a tą wartością sygnału, która była by - według nauczyciela prawidłowa. Ta różnica oznaczana jest zwykle symbolem greckiej litery d (delta) i stąd nazwa opisywanej metody.

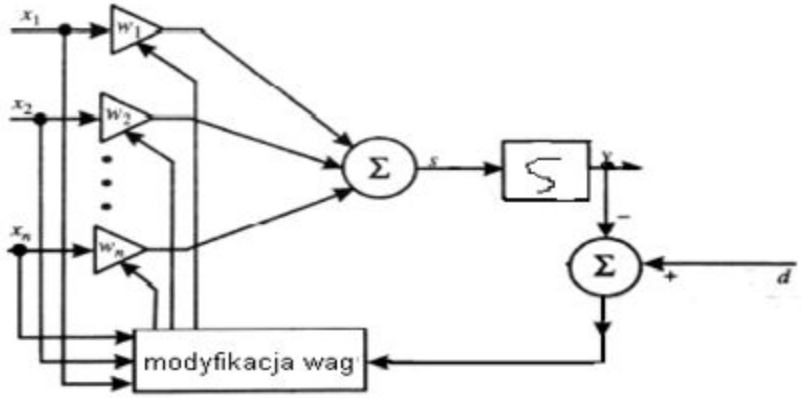
Schemat sieci Adaline

Model Adaline ma zbliżoną budowę do perceptronu. Różnią się one algorytmami uczenia. W modelu Adaline nie jest uwzględniana funkcja aktywacji przy porównywaniu sygnału wyjściowego z sygnałem wzorcowym.



Model sieci DeltaRule

Model DeltaRule ma analogiczną budowę do modelu Adaline, jednakże funkcją aktywacji jest funkcja sigmoidalna, a przy aktualizacji wag uwzględnia się pochodną tejże funkcji.



Wyniki

```
Learning rate: 0,01
Adaline
Epoch: 85
MSE error: 0,0999193970576812
Test letters:
Letter A is:
Big
Letter B is:
Big
Letter C is:
Big
Letter D is:
Big
Letter E is:
Big
Letter F is:
Big
Letter G is:
Big
Letter H is:
Big
Letter I is:
Big
Letter J is:
Big
Letter a is:
Small
Letter b is:
Small
Letter c is:
Small
Letter d is:
Small
Letter e is:
Small
Letter f is:
Small
Letter g is:
Small
Letter h is:
Small
Letter i is:
Small
Letter j is:
Small
```

```
Delta rule
Epoch: 668
MSE error: 0,0999166909600333
Test letters:
Letter A is:
Big
Letter B is:
Big
Letter C is:
Big
Letter D is:
Big
Letter E is:
Big
Letter F is:
Big
Letter G is:
Big
Letter H is:
Big
Letter I is:
Big
Letter J is:
Big
Letter a is:
Small
Letter b is:
Small
Letter c is:
Small
Letter d is:
Small
Letter e is:
Small
Letter f is:
Small
Letter g is:
Small
Letter h is:
Small
Letter i is:
Small
Letter j is:
Small
```

```
Learning rate: 0,05
Adaline
Epoch: 15
MSE error: 0,0904925850837542
Test letters:
Letter A is:
Big
Letter B is:
Big
Letter C is:
Big
Letter D is:
Big
Letter E is:
Big
Letter F is:
Big
Letter G is:
Big
Letter H is:
Big
Letter I is:
Big
Letter J is:
Big
Letter a is:
Small
Letter b is:
Small
Letter c is:
Small
Letter d is:
Small
Letter e is:
Small
Letter f is:
Small
Letter g is:
Small
Letter h is:
Small
Letter i is:
Small
Letter j is:
Small
```

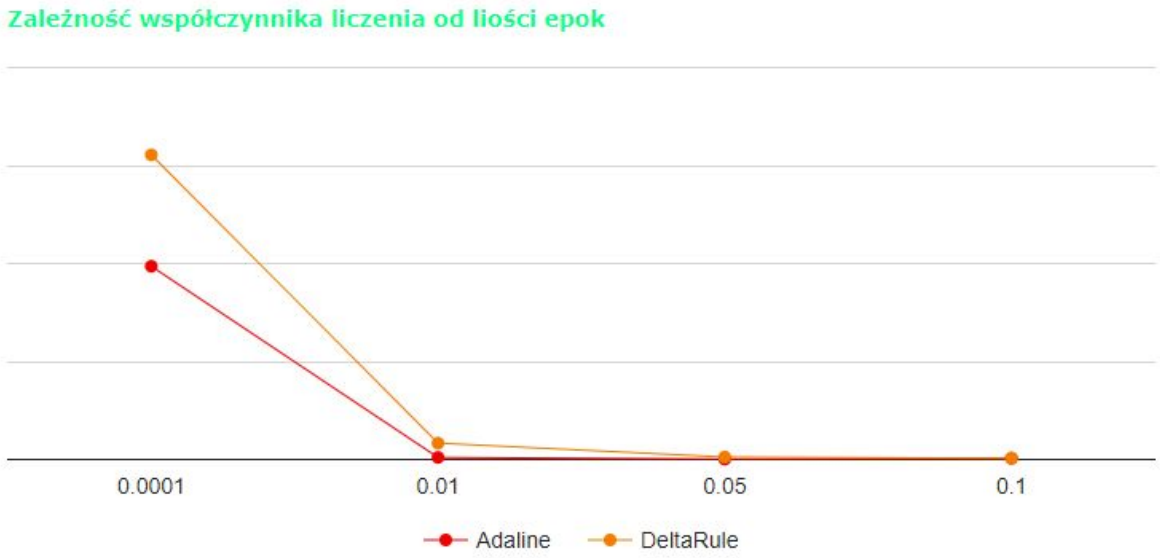
```
Delta rule
Epoch: 101
MSE error: 0,0997466098812923
Test letters:
Letter A is:
Big
Letter B is:
Big
Letter C is:
Big
Letter D is:
Big
Letter E is:
Big
Letter F is:
Big
Letter G is:
Big
Letter H is:
Big
Letter I is:
Big
Letter J is:
Big
Letter a is:
Small
Letter b is:
Small
Letter c is:
Small
Letter d is:
Small
Letter e is:
Small
Letter f is:
Small
Letter g is:
Small
Letter h is:
Small
Letter i is:
Small
Letter j is:
Small
```

```
Learning rate: 0,1
Adaline
Epoch: 51
MSE error: 0,0901803867750986
Test letters:
Letter A is:
Big
Letter B is:
Big
Letter C is:
Big
Letter D is:
Big
Letter E is:
Big
Letter F is:
Big
Letter G is:
Big
Letter H is:
Big
Letter I is:
Big
Letter J is:
Big
Letter a is:
Small
Letter b is:
Small
Letter c is:
Small
Letter d is:
Small
Letter e is:
Small
Letter f is:
Small
Letter g is:
Small
Letter h is:
Small
Letter i is:
Small
Letter j is:
Small
```

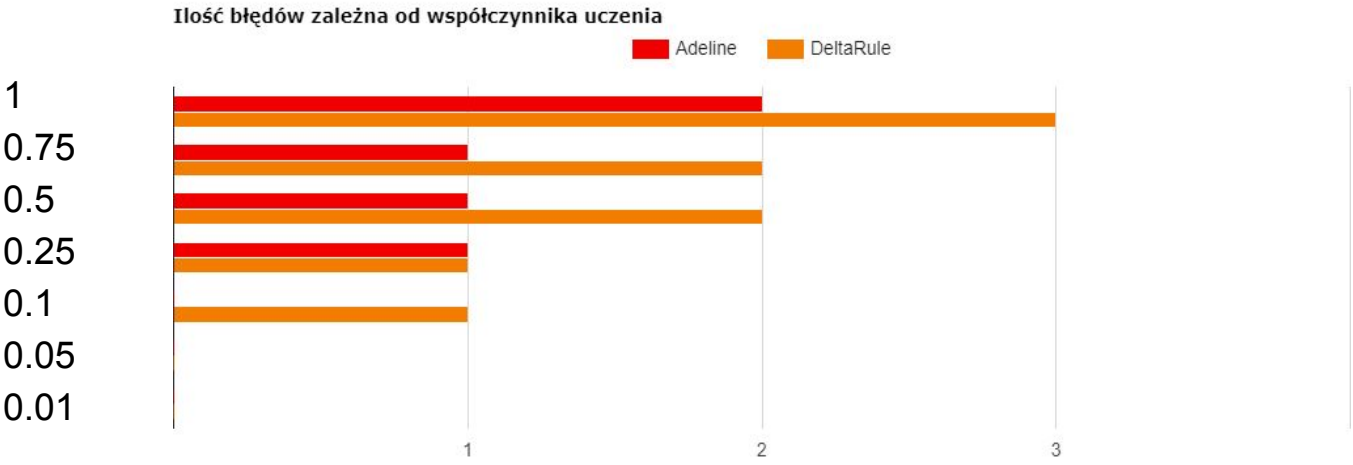
```
Delta rule
Epoch: 52
MSE error: 0,098698311071024
Test letters:
Letter A is:
Big
Letter B is:
Big
Letter C is:
Big
Letter D is:
Big
Letter E is:
Big
Letter F is:
Big
Letter G is:
Big
Letter H is:
Big
Letter I is:
Big
Letter J is:
Big
Letter a is:
Small
Letter b is:
Small
Letter c is:
Small
Letter d is:
Small
Letter e is:
Small
Letter f is:
Small
Letter g is:
Small
Letter h is:
Small
Letter i is:
Small
Letter j is:
Small
```

```
Learning rate: 0,0001
Adaline
Epoch: 7888
MSE error: 0,0999981093056813
Test letters:
Letter A is:
Big
Letter B is:
Big
Letter C is:
Big
Letter D is:
Big
Letter E is:
Big
Letter F is:
Big
Letter G is:
Big
Letter H is:
Big
Letter I is:
Big
Letter J is:
Big
Letter a is:
Small
Letter b is:
Small
Letter c is:
Small
Letter d is:
Small
Letter e is:
Small
Letter f is:
Small
Letter g is:
Small
Letter h is:
Small
Letter i is:
Small
Letter j is:
Small
```

Analiza



Na wykresie widać, że wraz ze wzrostem współczynnika uczenia zaobserwowałem wzrost wydajności uczenia przez zmniejszenie się ilości epok potrzebnych do nauczenia się sieci neuronowej. Dla współczynnika równego 0.0001 ilość epok potrzebnych do nauczenia modelu była większa niż 7000 w obu przypadkach. Już przy wartości 0.01 model miał dość dobrą wydajność, ponieważ Adaline potrzebował tylko 85 epok, jednak DeltraRule aż 668. Przy wartości 0.01 wydajność obu modeli była już bardzo zadowalająca i bardzo zbliżona, ponieważ wahała się w okolicach 50 epok.



Na wykresie powyżej możemy zaobserwować ilości błedów przy danym współczynniku uczenia. Metoda Adeline prawie zawsze wypada lepiej.

Wnioski

To jak bardzo szybko i wydajnie dana sieć neuronowa zostanie nauczona zależy przede wszystkim od wartości współczynnika uczenia. Dzięki temu, że jest zwiększany, również zwiększany jest przyrost wag, więc model szybciej się uczy. Z dwóch zaprezentowanych metod lepszą okazuje się metoda Adeline, ponieważ przy większości prób miała mniejszą ilość epok to popełniała mniej błędów.
Wraz ze wzrostem współczynnika nauki malała liczba epok potrzebnych do nauczenia sieci. Dzięki sytuacji, że dane wejściowe są pełne i nie ma w nich żadnych dziur, efekty uczenia są lepsze, ponieważ w przypadku niekompletnych danych wyniki mogłoby być błędne.

Kod programu

```
namespace OneLayerNetwork
{
   class Program
   {
      static void Main(string[] args)
      {
         var learningRate = 0.1;
         Console.WriteLine("Learning rate: " + learningRate);

         Console.WriteLine("Adaline");

         var adaline = new Adaline(learningRate);
         adaline.Learn();
         adaline.Test();

         Console.WriteLine("Delta rule");

         var deltaRule = new DeltaRule(learningRate);
         deltaRule.Learn();
         deltaRule.Test();

      }
   }
}
public class DeltaRule
   {
      private readonly List<double> _weightsContainer;
      private readonly double _learningRate;
      private readonly int[,] _inputData;
```

```
private readonly int[,] _testData;
private readonly int[] _expectedResults;
private readonly int _numberOfLetters;
private readonly int _numberOfFields;
private readonly char[] _setTestLetters;
private double _error;
private double _delta;
private double _output;

public DeltaRule(double learningRate)
{
    _setTestLetters = new[]
        {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j'};
    _learningRate = learningRate;
    _delta = 0;
    _numberOfFields = 35;
    _numberOfLetters = 20;
    _expectedResults = new int[20];
    _error = 0.0;
    _weightsContainer = new List<double>();
    var random = new Random();
    for (var i = 0; i < _numberOfFields; i++)
    {
        _weightsContainer.Add(random.NextDouble());
    }

    _inputData = ReadDataFromFile("C:\\repositories\\PSI\\Projekt2\\OneLayerNetwork\\data_for_learning.txt");
    _testData = ReadDataFromFile("C:\\repositories\\PSI\\Projekt2\\OneLayerNetwork\\data_for_test.txt");
    for (var i = 0; i < 20; i++)
    {
        if (i < 10)
            _expectedResults[i] = 1;
        else
        {
            _expectedResults[i] = 0;
        }
    }
}

private int[,] ReadDataFromFile(string url)
{
    var input = File.ReadAllText(url);

    var i = 0;
    var result = new int[_numberOfLetters, _numberOfFields];
    foreach (var row in input.Split('\n'))
    {
        var j = 0;
        foreach (var col in row.Trim().Split(' '))
        {
            result[i, j] = int.Parse(col.Trim());
            j++;
        }

        i++;
    }
```

```
    return result;
}

private int[] getOneRow(int[,] inputData, int whichRow)
{
    var tablica = new int[35];
    for (var i = 0; i < tablica.Length; i++)
    {
        tablica[i] = inputData[whichRow, i];
    }

    return tablica;
}

private double Sum(IReadOnlyList<int> letter, IReadOnlyList<double> weightsContainer)
{
    var sum = 0.0;
    for (var i = 0; i < _numberOfFields; i++)
        sum += letter[i] * weightsContainer[i];
    return sum;
}

public void Test()
{
    Console.WriteLine("Test letters: ");
    for (var i = 0; i < _numberOfLetters; i++)
    {
        Console.WriteLine("Letter " + _setTestLetters[i] + " is: ");
        Console.WriteLine(ActivationFunction(Sum(getOneRow(_testData, i), _weightsContainer)) > 0.5
            ? "Big"
            : "Small");
    }

    Console.ReadLine();
}

public void Learn()
{
    bool acceptableError;

    var epoch = 0;

    do
    {
        epoch++;
        _error = 0.0;
        for (var i = 0; i < _numberOfLetters; i++)
        {
            _output = ActivationFunction(Sum(getOneRow(_inputData, i), _weightsContainer));

            _delta = _expectedResults[i] - _output;

            for (var j = 0; j < _numberOfFields; j++)
                _weightsContainer[j] += _learningRate * _delta * _inputData[i, j] *
                                        DerivativeActivationFunction(Sum(getOneRow(_inputData, i),
                                            _weightsContainer));
```

```
                _error += _delta * _delta;
            }

            _error /= 2;

            acceptableError = !(_error > 0.1);
        } while (!acceptableError);

        Console.WriteLine("Epoch: " + epoch);
        Console.WriteLine("MSE error: " + _error);
    }

    private static double DerivativeActivationFunction(double sum)
    {
        return (1.0 * Math.Exp(-1.0 * sum)) / (Math.Pow(Math.Exp(-1.0 * sum) + 1, 2));
    }

    private double ActivationFunction(double sum)
    {
        return (1 / (1 + Math.Exp(-1.0 * sum)));
    }
  }
public class Adaline
  {
    private readonly List<double> _weightsContainer;
    private readonly double _learningRate;
    private readonly int[,] _inputData;
    private readonly int[,] _testData;
    private readonly int[] _expectedResults;
    private readonly int _numberOfLetters;
    private readonly int _numberOfFields;
    private readonly char[] _setTestLetters;
    private double _error;
    private double _delta;
    public Adaline(double learningRate)
    {
        _setTestLetters = new[]
            {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j'};
        _learningRate = learningRate;
        _delta = 0;
        _numberOfFields = 35;
        _numberOfLetters = 20;
        _expectedResults = new int [20];
        _error = 0.0;
        _weightsContainer = new List<double>();
        var random = new Random();
        for (var i = 0; i < _numberOfFields; i++)
        {
            _weightsContainer.Add(random.NextDouble());
        }

        _inputData = ReadDataFromFile("C:\\repositories\\PSI\\Projekt2\\OneLayerNetwork\\data_for_learning.txt");
        _testData = ReadDataFromFile("C:\\repositories\\PSI\\Projekt2\\OneLayerNetwork\\data_for_test.txt");
        for (var i = 0; i < 20; i++)
        {
            if (i < 10)
                _expectedResults[i] = 1;
```

```
        else
        {
            _expectedResults[i] = 0;
        }
    }
}

private int[,] ReadDataFromFile(string url)
{

    var input = File.ReadAllText(url);

    var i = 0;
    var result = new int[_numberOfLetters, _numberOfFields];
    foreach (var row in input.Split('\n'))
    {
        var j = 0;
        foreach (var col in row.Trim().Split(' '))
        {
            result[i, j] = int.Parse(col.Trim());
            j++;
        }
        i++;
    }
    return result;
}

private int[] getOneRow(int[,] inputData, int whichRow)
{
    var tablica = new int [35];
    for (var i = 0; i < tablica.Length; i++)
    {
        tablica[i] = inputData[whichRow, i];
    }

    return tablica;
}
public void Learn()
{

    bool acceptableError;

    var epoch = 0;

    do
    {
        epoch++;
        _error = 0.0;
        for (var i = 0; i < _numberOfLetters; i++)
        {

            _delta = _expectedResults[i] - Sum(getOneRow(_inputData, i), _weightsContainer);

            for (var j = 0; j < _numberOfFields; j++)
                _weightsContainer[j] += _learningRate * _delta * _inputData[i,j];

            _error += _delta * _delta;
```

```
            }
            _error /= 2;

            acceptableError = !(_error > 0.1);
        } while (!acceptableError);

        Console.WriteLine("Epoch: " + epoch);
        Console.WriteLine("MSE error: " + _error);
    }

    private double Sum(IReadOnlyList<int> letter, IReadOnlyList<double> weightsContainer)
    {
        var sum = 0.0;
        for (var i = 0; i < _numberOfFields; i++)
            sum += letter[i] * weightsContainer[i];
        return sum;
    }

    private static bool Active(double sum)
    {
        return sum > 0.5;
    }


    public void Test()
    {
        Console.WriteLine("Test letters: ");
        for (var i = 0; i < _numberOfLetters; i++)
        {
            Console.WriteLine("Letter " + _setTestLetters[i] + " is: ");
            Console.WriteLine(Active(Sum(getOneRow(_testData, i), _weightsContainer)) ? "Big" : "Small");
        }
        Console.ReadLine();
    }


}
```