Podstawy Sztucznej Inteligencji

Scenariusz 2: **Budowa​​ i​​ działanie​​ sieci​​ jednowarstwowej**

Niniejszy program nie uczy się czy litery są małe czy duże lecz rozpoznaje każdą literę od ‘A’ do ‘J’ oraz od ‘a’ do ‘j’ jest to modyfikacja scenariusza 2, która spowodowana jest moją błędną jego interpretacją lecz zaakceptowania po rozmowie z prowadząca jako prawidłowa. Dodatkowo przedstawiono tu jedynie jeden algorytm ponieważ na wykładach jeszcze nie zaczęliśmy omawiać danego tematu. Wszystkie projekty znajdują się w jednej solucji.

Wykonał:

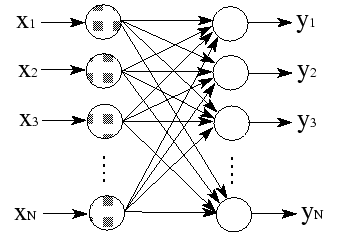
Bartłomiej Leja 254493

**1.Cel ćwiczenia**

Celem drugiego projektu było stworzenie na podstawie wcześniej zaimplementowanego perceptronu jednowarstwowej sieci neuronowej. Poznanie jej budowy i działania oraz nauczenie rozpoznawania liter.

**2.Sieć jednowarstwowa**

Sieć jednowarstwowa to najprostsza z sieci, polegająca na wykorzystaniu więcej niż jednego neuronu. Dane wejściowe przekazywane są do każdego neuronu należącego do sieci. Sieć taka zobrazowana jest na rysunku rys.1.



Rys.1

**3.Uczenie sieci jednowarstwowej**

Uczeni w sieci jednowarstwowej realizowane jest takim samym schematem jak uczenie perceptronu. Różnica polega na tym iż w przypadku sieci mamy do czynienia z wieloma neuronami. Każdy z neuronów dostaje na wejściu ten sam zestaw danych. Dodatkowo każdy neuron uczy się tylko jednej litery, której matrycą jest tablica 7x5. W projekcie rozpoznawane jest 10 dużych liter oraz 10 małych z czego wynika iż sieć składa się z 20 neuronów.

Uczenie neuronu przeprowadzane jest regułą delty. Regułą DELTA, zakłada że w raz z każdym wektorem wejściowym **X** do neuronu podawany jest sygnał **z**. Neuron odpowiada na sygnał **X** sygnałem 

**y = W \* X**

Przy czym jeśli neuron nie jest nauczony, sygnał ten jest inny niż wymagany (y≠z). Wewnątrz neuronu istnieje blok oceniający wielkość błędu

**δ = z - y**

Blok ten składa się inwertora oraz sumatora. Na podstawie sygnału błedu oraz wektora wejściowego **X** możliwe jest takie skorygowanie wektora wag **W**, by neuron lepiej realizował zadaną funkcję **y= f(X)**. Nowy wektor wag **W’** obliczany jest ze wzoru:

**W’ = W + ηδX**

gdzie **η** jest współczynnikiem liczbowym, decydującym o szybkości nauki.

**4.Uzyskane wyniki**

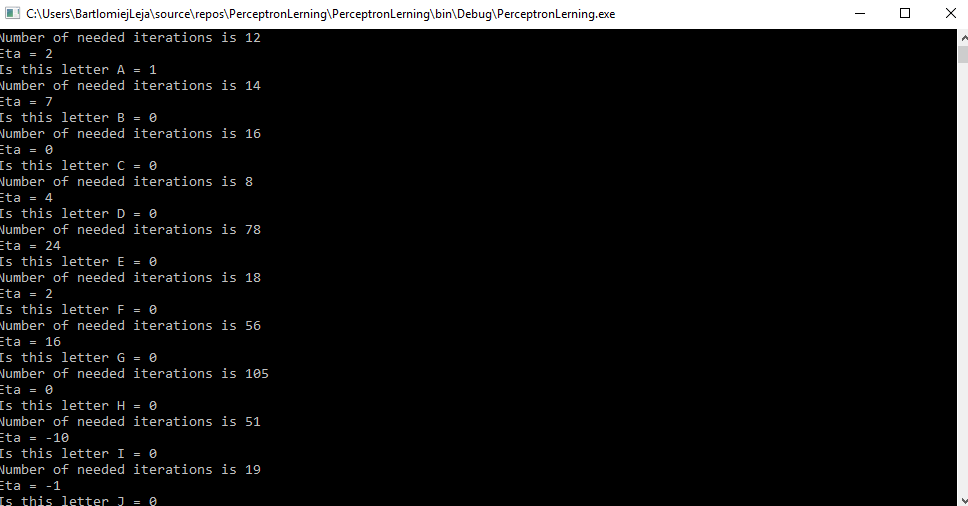
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Numer wariantu | Współczynnik uczenia η | Liczba danych uczących | Liczba potrzebnych iteracji do wyuczenia litery A | Liczba potrzebnych iteracji do wyuczenia litery B |
| 1 | 1 | 20 | 9 | 10 |
| 2 | 0,5 | 20 | 12 | 30 |
| 3 | 0,25 | 20 | 30 | 42 |
| 4 | 0,1 | 20 | 99 | 209 |

Tab. 1

Rys.2

Bardzo trudno przedstawić wyniki w postaci tabelarycznej ponieważ rozpoznajemy 20 liter z których każdą obsługuje inny neuron ze swoja liczbą iteracji i swoimi początkowymi losowo wybranymi wagami. Jak widać w tabeli 1 liczba iteracji jest inna dla każdej litery w tym przypadku mniejsza dla litery A. Dodatkowo jak można było się spodziewać czym większy współczynnik uczenia tym miej iteracji ponieważ algorytm staje się wtedy mniej dokładny. Dodatkowo zauważono, iż czym mniejszy współczynnik uczenia tym więcej razy algorytm rozpoznaje nawet zniekształconą literę A. Algorytm niestety nie jest przewidywalnym czasami literę ‘A’ rozpoznaję jako ‘H’ co jest związane z małą matryca i podobieństwem tych liter.

**6.Zrzut z ekranu**



**7.Listing kodu**

Klasa UniversalNeuron

using System;

using System.Collections.Generic;

namespace PerceptronLerning

{

class UniversalNeuron

{

private double \_eta;

private double \_learningCoefficient;

private double[] \_weight { get; set; }

private List<LetterData> \_letterInput;

private LetterDataTextFileReader dictionaryInput;

int countOfletterArray = 35;

int \_letterIndex;

char \_letter;

public UniversalNeuron(List<LetterData> letterInput, double learningCoefficien, int letterIndex, char letter )

{

\_letterInput = letterInput;

\_learningCoefficient = learningCoefficien;

dictionaryInput = new LetterDataTextFileReader();

\_letterIndex = letterIndex;

\_letter = letter;

}

public void PerceptronLearning()

{

drawWeight();

var errorCount = 0;

double error = 0;

var iterationCount = 2000;

for (int i = 0; i < iterationCount; i++)

{

errorCount = 0;

double y = 0;

foreach (var letter in \_letterInput)

{

for (int j = 0; j < 35; j++)

y += letter.LetterPattern[j] \* \_weight[j];

if (letter.LetterResult[\_letterIndex] != calculateValue(y))

{

error = letter.LetterResult[\_letterIndex] - calculateValue(y);

for (int k = 0; k < 35; k++)

\_weight[ k] = \_weight[ k] + \_learningCoefficient \* error \* letter.LetterPattern[ k];

\_eta = \_eta - error;

errorCount++;

}

}

if (errorCount == 0)

{

Console.WriteLine($"Number of needed iterations is {i + 1}");

for (int k = 0; k < 35; k++)

Console.WriteLine($"Weight {k} = {\_weight[k]}");

Console.WriteLine($"Eta = {\_eta}");

break;

}

}

testMethod(@"C:\Users\BartlomiejLeja\source\repos\PerceptronLerning\PerceptronLerning\letterTestData.txt");

}

private double calculateValue(double sum)

{

return (sum > \_eta) ? 1 : 0;

}

private void drawWeight()

{

\_weight = new double[countOfletterArray];

var randomNumber = new Random();

for (int i = 0; i < 35; i++)

\_weight[i] = -10 + randomNumber.NextDouble() \* 20;

for (int i = 0; i < 35; i++)

Console.WriteLine($"Weight {i} = {\_weight[i]}");

}

private void testMethod(string path)

{

double result = 0;

List<double[]> testLetter = dictionaryInput.ConverTextForTestDataArray(path);

for (int i = 0; i < testLetter.Count; i++)

{

for (int k = 0; k < 35; k++)

result += testLetter[i][k] \* \_weight[ k];

Console.WriteLine($"Is this letter {\_letter} = {calculateValue(result)}");

}

}

}

}

Klasa OneLayserNetwork

using System.Collections.Generic;

namespace PerceptronLerning

{

class OneLayerNetwork

{

private List<UniversalNeuron> NeuronLayer;

public OneLayerNetwork(List<LetterData> letterInput,double learningCoefficien)

{

char[] letters ={'A', 'B' ,'C','D','E','F','G','H','I','J','a','b','c','d','e','f','g','h','i','j'};

NeuronLayer = new List<UniversalNeuron>();

for (int i = 0; i < letterInput.Count; i++)

{

NeuronLayer.Add(new UniversalNeuron(letterInput, learningCoefficien, i,letters[i]));

}

CreateOneLayerNetwork();

}

private void CreateOneLayerNetwork()

{

foreach (var universalNeuron in NeuronLayer)

{

universalNeuron.PerceptronLearning();

}

}

}

}

Klasa pobierająca dane LetterDataTextFileReader.cs

using System;

using System.Collections.Generic;

using System.IO;

using System.Text;

namespace PerceptronLerning

{

//TODO Make it for static

class LetterDataTextFileReader

{

List<LetterData> letterList;

List<double[]> testLetterList;

double[] patternArray;

double[] resultArray;

double[] testArray;

string[] values;

int countOfLetterArray=35;

public List<double []> ConverTextForTestDataArray(string path)

{

testLetterList = new List<double[]>();

string[] lines = File.ReadAllLines(path, Encoding.UTF8);

for (int i = 0; i < lines.Length; i++)

{

values = lines[i].Split(' ');

testArray = new double[countOfLetterArray];

for (int j = 0; j < countOfLetterArray; j++)

{

testArray[j] = Convert.ToDouble(values[j]);

}

testLetterList.Add(testArray);

}

return testLetterList;

}

public List<LetterData> ConvertTextForArray(string path)

{

string[] lines = File.ReadAllLines(path, Encoding.UTF8);

letterList = new List<LetterData>();

for (int i = 0; i < lines.Length; i++)

{

values = lines[i].Split(' ');

if (values.Length == 35)

{

patternArray = new double[countOfLetterArray];

for (int j = 0; j < countOfLetterArray; j++)

{

patternArray[j] = Convert.ToDouble(values[j]);

}

}

else if (values.Length == 20)

{

resultArray = new double[20];

for (int j = 0; j < 20; j++)

{

resultArray[j] = Convert.ToDouble(values[j]);

}

letterList.Add(new LetterData(patternArray, resultArray));

}

}

return letterList;

}

}

}

Literatura:

<http://wwwold.ece.utep.edu/research/webfuzzy/docs/kk-thesis/kk-thesis-html/node15.html>

[**http://www.ai.c-labtech.net/sn/litery.html**](http://www.ai.c-labtech.net/sn/litery.html)

**http://galaxy.agh.edu.pl/~vlsi/AI/rdelta/**