Podstawy Sztucznej Inteligencji

Scenariusz 1: Budowa i działanie perceptronu

Poprawiona wersja po rozmowie z prowadzącą zajęcia oraz udostępnieniem wymagań co do sprawozdania z pierwszego projektu. Wszystkie projekty tworzone są w jednej solucji programu.

Wykonał:

Bartłomiej Leja 254493

**1.Cel ćwiczenia**

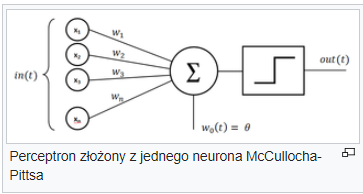
Celem pierwszego projektu było zapoznanie się z budową perceptronu oraz jego implementacja. Dodatkowo należało zaimplementować uczenia perceptronu realizującego uczenie jednej dowolnie wybranej funkcji logicznej z dwoma zmiennymi.

**2.Perceptron**

Perceptron – najprostsza sieć neuronowa, składająca się z jednego bądź wielu niezależnych neuronów McCullocha-Pittsa, implementująca algorytm uczenia nadzorowanego klasyfikatorów binarnych. Perceptron jest funkcją, która potrafi określić przynależność parametrów wejściowych do jednej z dwóch klas. Może być wykorzystywany tylko do klasyfikowania zbiorów liniowo separowalnych.

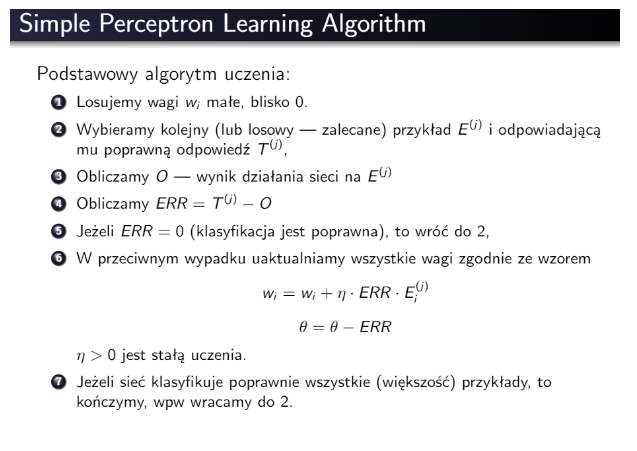
Perceptron składa się z:

* n wejść: x 1 , …, x n
* n wag: w 1 , … , w n
* progu θ
* wyjścia y.



**3.Uczenie perceptronu**

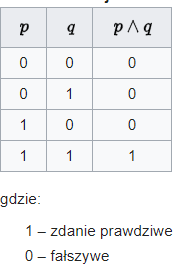
Proces uczenia polega na korygowaniu wartości współczynników wagowych. Początkowe wartości składowych wektora wag nie mają większego znaczenia i z reguły nadaje się im wartości losowe. W stworzonym algorytmie wykorzystano uczenie z nauczycielem (nadzorowane) polegające na pokazaniu na wejściu sieci wektora sygnałów wejściowych a na wyjściu pokazaniu oczekiwanej wartości. Zadaniem uczenia sieci jest dokonanie takiej korekty współczynników, aby zmniejszyć różnice pomiędzy oczekiwaną, a rzeczywistą odpowiedzią sieci.



**4.Uzyskane wyniki**

W wykonany projekcie zaimplementowano perceptron, który realizuję funkcję logiczną jaką

jest koniunkcja.



Przeprowadzając proces uczenia perceptronu zastosowano różne warianty różniące się między sobą współczynnikiem uczenia oraz licznością danych uczących.

|  |  |  |  |
| --- | --- | --- | --- |
| Numer wariantu | Współczynnik uczenia η | Liczba danych uczących | Liczba potrzebnych iteracji do wyuczenia |
| 1 | 1 | 30 | 4 |
| 2 | 0,5 | 30 | 6 |
| 3 | 0,25 | 30 | 7 |
| 4 | 0,1 | 30 | 11 |
| 5 | 1 | 15 | 4 |
| 6 | 0,5 | 15 | 10 |
| 7 | 0,25 | 15 | 13 |
| 8 | 0,1 | 15 | 21 |

Tab.1

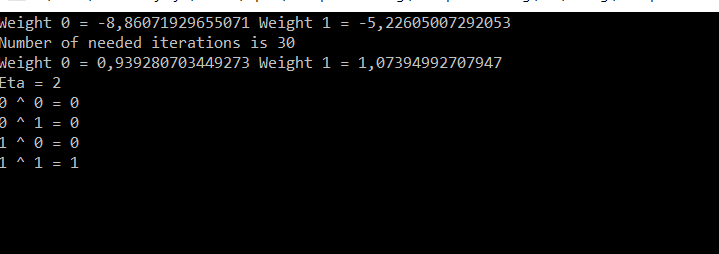
Rys.1

Rys. 2

**5. Wnioski**

Współczynnik uczenia (eta) wpływa na szybkość uczenia się poprzez bezpośrednie oddziaływanie na gradient zmiany. Wraz ze zmniejszaniem się współczynnika potrzeba coraz więcej iteracji na wykonanie algorytmu. Najczęściej stosowane wartości współczynnika uczenia zawierają się pomiędzy 0.01 a 0.6. Ilość danych uczących również wpływa na ilość iteracji czym miej danych uczących tym więcej iteracji co obrazują wykresy rys.1, rys.2 oraz tabela tab. 1.

**6. Zrzut ekranu z działania programu**



7. Listing kodu

Klasa neuron Neuron.cs

using System;

namespace PerceptronLerning

{

public class Neuron

{

private double \_eta;

private double \_learningCoefficient;

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

private double[,] \_input;

private int \_countOfInputs;

public Neuron(double[,] input, int countOfInputs,double learningCoefficien)

{

\_input = input;

\_countOfInputs = countOfInputs;

\_learningCoefficient = learningCoefficien;

}

public void PerceptronLearning()

{

drawWeight();

var errorCount = 0;

double error = 0;

var iterationCount = 2000;

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

{

errorCount = 0;

for(int j =0;j< \_input.Length/3;j++)

{

var y = \_input[j,0] \* \_weight[0] + \_input[j,1] \* \_weight[1];

if (\_input[j,2] != calculateValue(y))

{

error = \_input[j, 2] - calculateValue(y);

\_weight[0] = \_weight[0] + \_learningCoefficient \* error \* \_input[j,0];

\_weight[1] = \_weight[1] + \_learningCoefficient \* error \* \_input[j,1];

\_eta = \_eta - error;

errorCount++;

}

}

if(errorCount ==0)

{

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

Console.WriteLine($"Weight 0 = {\_weight[0]} Weight 1 = {\_weight[1]}");

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

break;

}

}

testMethod(0, 0);

testMethod(0, 1);

testMethod(1, 0);

testMethod(1, 1);

}

private void testMethod(int p, int q)

{

var result = p \* \_weight[0] + q \* \_weight[1];

Console.WriteLine($"{p} ^ {q} = {calculateValue(result)}");

}

private double calculateValue(double sum)

{

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

}

private void drawWeight()

{

\_weight = new double [2];

var randomNumber = new Random();

for (int i=0; i< \_countOfInputs; i++)

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

Console.WriteLine($"Weight 0 = {\_weight[0]} Weight 1 = {\_weight[1]}");

}

}

}

Klasa wczytująca dane z pliku **LogicalDataTextFileReader.cs**

|  |
| --- |
| using System; |
|  | using System.IO; |
|  | using System.Text; |
|  |  |
|  | namespace PerceptronLerning |
|  | { |
|  | class LogicalDataTextFileReader |
|  | { |
|  | double[,] resultArray; |
|  | public double[,] ConvertTextForArray(string path) |
|  | { |
|  | string[] lines = File.ReadAllLines(path, Encoding.UTF8); |
|  | resultArray = new double[lines.Length, 3]; |
|  |  |
|  | for(int i=0; i< lines.Length;i++) |
|  | { |
|  | string[] values = lines[i].Split(' '); |
|  | resultArray[i, 0] = Convert.ToDouble( values[0]); |
|  | resultArray[i, 1] = Convert.ToDouble(values[1]); |
|  | resultArray[i, 2] = Convert.ToDouble(values[2]); |
|  | } |
|  | return resultArray; |
|  | } |
|  | } |
|  | } |

Literatura:

* <https://pl.wikipedia.org/wiki/Perceptron>
* <https://github.com/nsadawi/perceptron/blob/master/Perceptron.java>
* <https://github.com/RichardKnop/ansi-c-perceptron/blob/master/perceptron.c>
* <http://edu.pjwstk.edu.pl/wyklady/nai/scb/wyklad3/w3.htm>