**Weronika Muszyńska , gr 05**

**Temat ćwiczenia : Budowa i działanie perceptronu**

**Cel ćwiczenia**

Celem ćwiczenia jest poznanie budowy i działanie perceptronu poprzez implementację oraz

uczenie perceptronu realizującego wybraną funkcję logiczną dwóch zmiennych.

**Sztucznym neuronem** nazywamy prosty system przetwarzający wartości sygnałów wprowadzanych na jego wejścia w pojedynczą wartość wyjściową, wysyłaną na jego jednym wyjściu (dokładny sposób funkcjonowania określony jest przez przyjęty model neuronu). Jest to podstawowy element sieci neuronowych, jego pierwowzorem był biologiczny neuron.

**Neuron McCullocha-Pittsa** matematyczny model neuronu. Posiada wiele wejść i jedno wyjście. Każdemu z wejść przyporządkowana jest waga wejścia (liczba rzeczywista). Neuron ten jest podstawowym budulcem sieci neuronowej perceptron. Wartość na wyjściu neuronu obliczana jest następująco:

1) obliczana jest suma iloczynów wartości 𝑥𝑖 podanych na wejścia i wag 𝑤𝑖 wejść:

𝑠=𝑤0+ 𝑥𝑖𝑤𝑖𝑛𝑖=1

2) na wyjście podawana jest wartość funkcji aktywacji f(s) dla obliczonej sumy.

**Funkcja aktywacji-** funkcja, według której obliczana jest wartość wyjścia neuronów sieci neuronowej.



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Funkcja OR | | Ilość iteracji | | | |
| A | B | 10 | 100 | 1000 | 10000 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0,466 | 0,444 | 0,741 | 0,845 |
| 1 | 0 | 0,53 | 0,666 | 0,547 | 0,5 |
| 0 | 0 | 0,996 | 1,11 | 1,287 | 1,345 |

**Wnioski :**

Sieć szybciej się uczy, gdy podamy jej więcej danych uczących.

Podczas testów okazało się, że dla 10 iteracji za każdym razem sieć dawała błędne odpowiedzi. Przy wzrastaniu danych uczących sieć uczyła się i poprawna była jej reakcja. Jednakże dla bardzo dużej liczby iteracji odpowiedzi sieci były różne, nie zawsze dobre. Sieć została przeuczona, trzeba mieć to na uwadze.

**Listing kodu pisanego w Java**

**Klasa Main**

public class Main {

public static void main(String[] args) {

Perceptron p = new Perceptron();

double inputs [][] ={{0,0},{0,1},{1,0},{1,1}};

int outputs[] = {0,1,1,1};

p.Train(inputs,outputs,0.5,0.001,10);

System.out.println(p.Output(new double[]{0,0}));

System.out.println(p.Output(new double[]{0,1}));

System.out.println(p.Output(new double[]{1,0}));

System.out.println(p.Output(new double[]{1,1}));

}

}

**Klasa Perceptron**

import java.util.Random;

public class Perceptron {

double [] weights;

double threshold;

public void Train(double [][] inputs, int [] outputs, double threshold,double lrate, int iterations)

{

this.threshold = threshold;

int n = inputs[0].length;

int p = outputs.length;

weights = new double[n];

Random r = new Random();

for(int i=0;i<n;i++){

weights[i]=r.nextDouble();

}

for(int i=0;i<iterations;i++){

int totalError = 0;

for(int j=0;j<p;j++){

int output = Output(inputs[j]);

int error = outputs[j] - output;

totalError+=error;

for(int k=0;k<n;k++){

double delta = lrate \* inputs[j][k] \* error;

weights[k]+=delta;;

}

}

if(totalError == 0) break;

}

}

public int Output(double [] input){

double sum = 0.0;

for(int i=0;i<input.length;i++){

sum+=weights[i]\*input[i];

}

System.out.println("SUM: " + sum);

if(sum>=threshold) return 1;

else return 0;

}

}