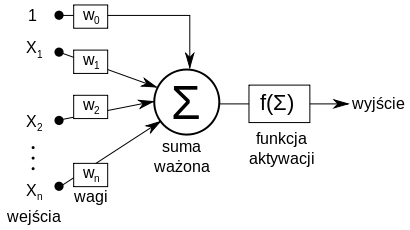
Maciej Słaboń

Celem ćwiczenia było poznanie budowy i działania perceptronu poprzez implementację oraz uczenie perceptronu realizującego wybraną funkcję logiczną dwóch zmiennych.

**1) Syntetyczny opis budowy wykorzystanego algorytmu uczenia:**



*Ilustracja 1: Model perceptronu McCullocha-Pitts*

Do budowy perceptronu wykorzystałem podany na wykładzie model McCullocha-Pittsa.

Wartość na wyjściu neuronu obliczana jest w następujący sposób[[1]](https://pl.wikipedia.org/wiki/Neuron_McCullocha-Pittsa#cite_note-1):

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

{\displaystyle s=w\_{0}+\sum \_{i=1}^{n}x\_{i}w\_{i}}



2. na wyjście podawana jest wartość [funkcji aktywacji](https://pl.wikipedia.org/wiki/Funkcja_aktywacji) f(s) dla obliczonej sumy

Neuron McCullocha-Pittsa jest podstawowym budulcem [sieci neuronowej](https://pl.wikipedia.org/wiki/Sie%C4%87_neuronowa) [perceptron](https://pl.wikipedia.org/wiki/Perceptron).

**2) Zestawienie otrzymanych wyników:**

Perceptron który zaprogramowałem uczy się bramki logicznej AND.

Wszystkie wagi początkowe ustawiłem na wartość równą 0.3. Z kolei współczynnik uczenia się ustawiłem na wartość 0.01. Proces uczenia wymagał 18 powtórzeń. Poniżej prezentacja na wykresie:

Sprawdziłem jak będzie wyglądał proces uczenia się dla różnych współczynników uczenia się. Test taki sam jak powyżej, wagi początkowe ustawiłem na 0.3. Wyniki prezentują się następująco:

Współczynnik: 0.01 =28

Współczynnik: 0.1 = 4

Współczynnik: 0.5 = 7

Współczynnik: 1= 6

Współczynnik: 1.5 = 6

Należy też sprawdzić jak zmieni się proces uczenia przy stałym współczynniku czyli dla 0.1 ponieważ przy nim potrzebował najmniejszej liczby iteracji, a zmiennych wagach. Wyniki tego eksperymentu to:

Waga: 0= 5

Waga: 0.2 = 3

Waga: 0.5 =4

Waga: 1 = 7

Waga: 3 = 18

**3) Analiza i dyskusja błędów uczenia i testowania opracowanego perceptronu w zależności od wartości współczynnika uczenia oraz liczby danych uczących:**

Nauka perceptronu jest zależna od trzech czynników którymi są: dane uczące, współczynnika uczenia oraz wag początkowych, gdzie można od razu zauważyć, że współczynnik uczenia ma największy wpływ na to jak szybko perceptron będzie wstanie się nauczyć

**4) Wnioski:**

Perceptron umożliwia implementacje funkcji matematycznych, która może być użyta na wiele sposobów np. tak jak w tym przykładzie do funkcji bramki AND, jest to dobry wstęp by pokazać możliwości jakie daję nam perceptron. Przy wykorzystaniu wielu perceptronów można będzie do rozwiązywania dożo bardziej skomplikowanych problemów

**5) Listing całego kodu**

#include<iostream>

#include<conio.h>

#include<time.h>

using namespace std;

int CalculateOutput(double w[], double x, double y);

int main(){

const int n = 4;

int maxiteration = 400;

double learning\_rate = 0.01;

srand(time(NULL));

double x[n], y[n], w[3], localerror, globalerror;

int iteration = 0, out[n], output;

w[0] = 0.1;//((double)rand() / (RAND\_MAX));

w[1] = 0.9;//((double)rand() / (RAND\_MAX));

w[2] = 0.03;//((double)rand() / (RAND\_MAX));

cout << w[0] << " " << w[1] << " " << w[2] << endl;

x[0] = 1;

x[1] = 1;

x[2] = 0;

x[3] = 0;

y[0] = 0;

y[1] = 1;

y[2] = 0;

y[3] = 1;

out[0] = 0;

out[1] = 0;

out[2] = 0;

out[3] = 1;

do {

iteration++;

globalerror = 0;

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

output = CalculateOutput(w, x[i], y[i]);

localerror = out[i] - output;

w[0] += localerror \* learning\_rate \* x[i];

w[1] += localerror \* learning\_rate \* y[i];

w[2] += learning\_rate \* localerror;

globalerror += (localerror\*localerror);

}

} while (globalerror != 0 && iteration < maxiteration);

cout << "Ilosc iteracji: " << iteration << endl;

cout << "Koncowe rownanie: " << w[0] << "\*x + " << w[1] << "\*y + " << w[2] << " = 0" << endl << endl;

system("pause");

return 0;

}

int CalculateOutput(double w[], double x, double y){

double sum = x\*w[0] + y\*w[1] + w[2];

return (sum >= 0) ? 1 : 0;

}