Krzysztof Sekuła

**SCENARIUSZ NR 1**

**Cel ćwiczenia:**

Celem ćwiczenia było poznanie budowy i działania perceptronu realizującego wybraną funkcje logiczną.

**Opis budowy:**

W programie dane wejściowe wraz ze swoimi wagami są przekazywane do perceptronu gdzie są sumowana w następujący sposób suma=∑x[i]\*w[i]

Wagi są losowane z przedziału od 0 do 1 a dane wejściowe wyglądają następująco:

x[0]=0 y[0]=0 out[0]=0

x[1]=0 y[1]=1 out[1]=1

x[2]=1 y[2]=0 out[2]=1

x[3]=1 y[3]=1 out[3]=1

Dane wyjściowe są obliczane na podstawie sumy, jeśli jest większa lub równa od 0 to dane wyjściowe wynoszą 1, w przeciwnym przypadku 0.

Następnie zmieniamy wagi w następujący sposób:

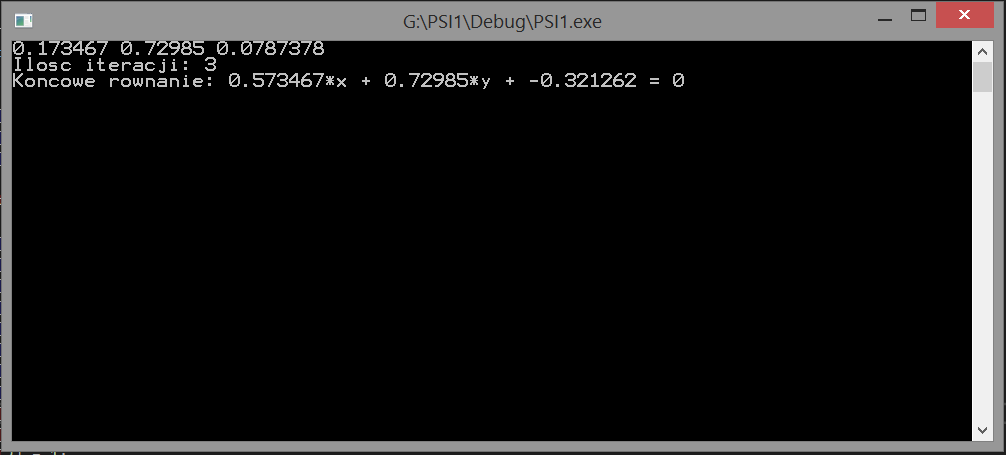
w[i]+=local\_error\*learning\_rate\*x[i]

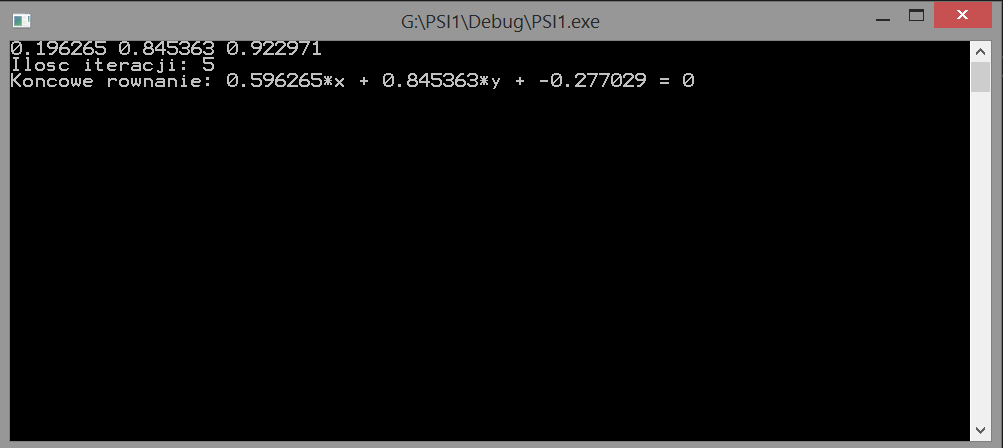
Global\_error +=(local\_error\*local\_error)

Jeśli Global\_error jest równy 0 lub ilość iteracji jest równa maksymalnej ilości iteracji to program kończy działanie.

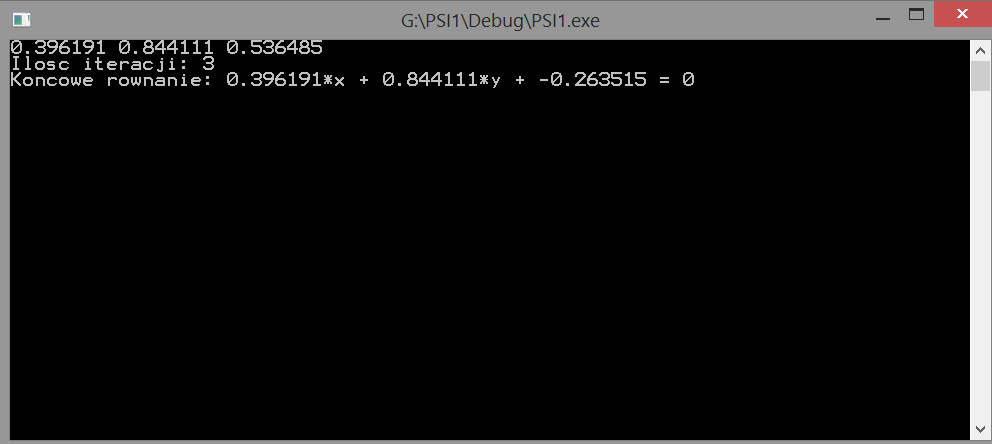
Program zwraca wzór prostej która stwierdza czy wartość zwracana będzie równa 0 czy 1.

**Wyniki:**









Jak możemy zauważyć program nie potrzebował zbyt wielu iteracji aby znaleźć wzór funkcji liniowej. Wpływ na tempo uczenia ma współczynnik learning\_rate który przyjmuje wartości od 0 do 1. Jeśli zwiększymy learning\_rate mogą się pojawić znaczące błędy.

**Wnioski:**

Na podstawie przeprowadzonego ćwiczenia możemy stwierdzić że nauka powinna trwać dopóki nie otrzymamy zadowalających wyników, od pewnego momentu błąd będzie wzrastał. Zwiększenie ilości danych uczących może również wpłynąć na dokładność uczenia.

**Listing kodu:**

#include<iostream>

#include<conio.h>

#include<time.h>

using namespace std;

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

{

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

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

}

int main()

{

const int n = 4;

int maxiteration = 400;

double learning\_rate = 0.4;

srand(time(NULL));

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

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

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

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

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

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

x[0] = 0;

x[1] = 0;

x[2] = 1;

x[3] = 1;

y[0] = 0;

y[1] = 1;

y[2] = 0;

y[3] = 1;

out[0] = 0;

out[1] = 1;

out[2] = 1;

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;

\_getch();

}