Jarosław Dutka

Inżynieria Obliczeniowa, rok III

WIMIiIP, AGH

**Sprawozdanie nr 1**

„Budowa i działanie perceptronu”

Celem ćwiczenia było poznanie działania perceptronu poprzez jego implementację oraz przetestowanie na różnych danych.

1. **Implementacja**

W ramach ćwiczenia zaimplementowano perceptron w języku C++, wzorując się na algorytmie opublikowanym na serwisie github.com pod linkiem: <https://github.com/RichardKnop/ansi-c-perceptron>.

Algorytm orzeka czy podany punkt leży nad czy pod wykresem funkcji liniowej, której wzór tworzy na podstawie punktów nad i pod tą prostą.

Dane wejściowe do algorytmu to pary współrzędnych oraz wynik:

* 1 – jeśli punkt leży nad wykresem funkcji;
* 0 – jeśli punkt leży pod wykresem funkcji.

Dane wejściowe zapisane są w pliku test1.txt, który został pobrany z podanego wyżej repozytorium.

Podstawowym elementem programu jest klasa Perceptron.

Obiekt tej klasy zawiera

1. zmienne

* dwie tablice zawierające dane wejściowe,
* tablice wag poszczególnych wejść,
* wynik swojego działania,
* tablice z wynikami danych uczących
* ilość – ilość danych do odczytania z pliku, podawana przez użytkownika
* ilość\_elementów – realna ilość odczytanych z pliku danych

1. funkcje

* losuj\_wagi() – losuje wartości początkowe wag
* sprawdz(float y, float x) – sprawdza położenie punktu o współrzędnych (x, y) względem prostej
* Perceptron() – konstruktor domyślny, który odczytuje dane uczące z pliku tekstowego
* ucz() –funkcja zawierająca algorytm wstecznej propagacji błędu, jej zadaniem jest takie dobieranie wag, aby błąd był możliwie najmniejszy

Klasa jest także zaprzyjaźniona z funkcją licz\_wyjscia(float wagi[], float x0, float x1),która użyta jest do obliczania danych wyjściowych z perceptronu.

1. **Wyniki**
2. Test nr 1

* Współczynnik nauki = 0,1
* Ilość danych wejściowych 208
* Wagi losowe

|  |  |  |
| --- | --- | --- |
| **Lp.** | **Równanie prostej** | **Ilość iteracji** |
| 1. | y = 0.870275\*x + 3.093051 | 16 |
| 2. | y = 0.916653\*x + 3.016291 | 5 |
| 3. | y = 0.890502\*x + 3.160631 | 9 |
| 4. | y = 0.934881\*x + 3.043536 | 9 |
| 5. | y = 0.856426\*x + 3.050766 | 12 |

Jak widać powyżej, wynikowe równania prostej po nauce o tych samych parametrach różnią się od siebie. Jest to efekt losowania wag przed etapem uczenia. Każdorazowe losowanie wag skutkuje różną ilością iteracji pętli, zanim średni błąd kwadratowy będzie wynosił 0.

**W celu zbadania wpływu zmiany parametrów do ilości iteracji, w kolejnych testach wartości początkowe wag ustawiono na 0.**

1. Test nr 2

* Współczynnik nauki = 0,1
* Ilość danych wejściowych 208
* Wagi stałe = 0

|  |  |  |
| --- | --- | --- |
| **Lp.** | **Równanie prostej** | **Ilość iteracji** |
| 1. | y = 0.891767\*x + 2.985586 | 14 |
| 2. | y = 0.891767\*x + 2.985586 | 14 |
| 3. | y = 0.891767\*x + 2.985586 | 14 |
| 4. | y = 0.891767\*x + 2.985586 | 14 |
| 5. | y = 0.891767\*x + 2.985586 | 14 |

Jak widać teraz za każdym razem wynik jest taki sam, ponieważ wagi początkowe mają w każdym pomiarze te same wartości.

1. Test nr 3

* Współczynnik nauki = 0,1

|  |  |  |  |
| --- | --- | --- | --- |
| **Lp.** | **Równanie prostej** | **Ilość danych wejściowych** | **Ilość iteracji** |
| 1. | y = 1.048876\*x + 3.470832 | 20 | 22 |
| 2. | y = 0.806143\*x + 2.873414 | 50 | 18 |
| 3. | y = 0.545505\*x + 1.176163 | 15 | 25 |
| 4. | y = 0.885036\*x + 2.915736 | 150 | 14 |
| 5. | y = 0.836767\*x + 3.258333 | 175 | 5 |

Ilość danych wejściowych w znacznym stopniu działa na ilość iteracji. Mała ilość danych wpływa także niekorzystnie na dokładność równania prostej.

1. Test nr 4

* Ilość danych wejściowych = 208

|  |  |  |  |
| --- | --- | --- | --- |
| **Lp.** | **Równanie prostej** | **Współczynnik nauki** | **Ilość iteracji** |
| 1. | y = 0.891766\*x + 2.985589 | 0,15 | 14 |
| 2. | y = 0.891767\*x + 2.985586 | 0,2 | 14 |
| 3. | y = 0.891767\*x + 2.985590 | 0,5 | 14 |
| 4. | y = 0.891767\*x + 2.985590 | 0,7 | 14 |
| 5. | y = 0.891767\*x + 2.985590 | 1 | 14 |

Manipulacja współczynnikiem nauki w nieznacznym stopniu wpływa na dokładność wyniku. Nie ma jednak wpływu na ilość wykonywanych przez program iteracji.

1. **Wnioski**

* Wyniki przeprowadzonych testów wykazały, że kluczowy w procesie nauki perceptrony jest odpowiedni dobór wag. W zależności od wylosowanych wag program dla tych samych danych wejściowych uczył się szybciej lub wolnej.
* Bardzo ważną kwestią jest także dostarczenie programowi odpowiednio dużej ilości danych wejściowych, aby wynik działania komórki nerwowej był jak najdokładniejszy.
* Współczynnik nauki wpływa na dokładność nauki w mniejszym stopniu.

1. **Kod źródłowy**

#include <cstdio>

#include <cstdlib>

#include <cmath>

#include <ctime>

#include <iostream>

#include <fstream>

#include <iomanip>

#define WSPOLCZYNNIK\_NAUKI 0.1

#define MAX\_ITERACJE 208

using namespace std;

class Perceptron

{

friend float licz\_wyjscia(float[], float, float);

int ilosc = 208; //wartość domyślna

float \*x0 = new float[ilosc];

float \*x1 = new float[ilosc];

float wagi[3];

int wynik;

int \*wynikOczekiwany = new int[ilosc];

int ilosc\_elementow;

public:

float losuj\_wagi()

{

return (float)rand() / (float)RAND\_MAX;

}

void ucz()

{

int iteracja = 0;

float blad\_lokalny;

float blad\_globalny = 0;

float sredni\_blad;

do {

blad\_globalny = 0;

iteracja++;

for (int p = 0; p < this->ilosc\_elementow; p++)

{

this->wynik = licz\_wyjscia(this->wagi, this->x0[p], this->x1[p]);

blad\_lokalny = this->wynikOczekiwany[p] - this->wynik;

this->wagi[0] += WSPOLCZYNNIK\_NAUKI \* blad\_lokalny \* this->x0[p];

this->wagi[1] += WSPOLCZYNNIK\_NAUKI \* blad\_lokalny \* this->x1[p];

this->wagi[2] += WSPOLCZYNNIK\_NAUKI \* blad\_lokalny;

blad\_globalny += (blad\_lokalny\*blad\_lokalny);

}

sredni\_blad = sqrt(blad\_globalny / this->ilosc\_elementow);

cout << "Iteracja #" << iteracja <<" |Średni błąd kwadratowy: "<< sredni\_blad<<fixed<< endl;

} while (blad\_globalny != 0 && iteracja <= MAX\_ITERACJE);

//cout << endl << "Równanie graniczne: " << this->wagi[0] << "\*x + " << this->wagi[1] << "\*y + " << this->wagi[2] << endl;

cout << endl << "Równanie graniczne: " << "y = " << this->wagi[0]/this->wagi[1] << "\*x + " << this->wagi[2] / this->wagi[1] << endl;

}

Perceptron()

{

//cout << "Podaj ilość danych uczących (max 208) >> "; cin >> ilosc;

//oczytanie danych wejsciowych z pliku

ifstream plik;

plik.open("test1.txt");

if (!plik.good())

{

cerr << "Nie można otworzyć pliku źródłowego!" << endl

<< "blad #" << plik.fail()<<endl<<endl;

return;

}

int i = 0;

while (!plik.eof() && i < ilosc -1) {

plik >> x0[i] >> x1[i] >> wynikOczekiwany[i];

if (wynikOczekiwany[i] == 0) {

wynikOczekiwany[i] = -1;

}

i++;

}

ilosc\_elementow = i;

plik.close();

//losowanie wag;

/\*wagi[0] = losuj\_wagi();

wagi[1] = losuj\_wagi();

wagi[2] = losuj\_wagi();\*/

wagi[0] = 0;

wagi[1] = 0;

wagi[2] = 0;

}

void sprawdz(float y, float x)

{

if ( y > (this->wagi[0]/ this->wagi[1]) \* x + (this->wagi[2]/this->wagi[1]))

cout << "Punkt leży nad prostą!" << endl;

else if((y < (this->wagi[0] / this->wagi[1]) \* x + (this->wagi[2] / this->wagi[1])))

cout << "Punkt leży pod prostą!" << endl;

else if ((y = (this->wagi[0] / this->wagi[1]) \* x + (this->wagi[2] / this->wagi[1])))

cout << "Punkt leży na prostą!" << endl;

}

};

float licz\_wyjscia(float wagi[], float x0, float x1)

{

float sum = x0 \* wagi[0] + x1 \* wagi[1] + wagi[2];

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

}

void main()

{

setlocale(LC\_ALL,"");

srand(time(NULL));

float y, x;

Perceptron p;

p.ucz();

cout << "Podaj y = "; cin >> y;

cout << "Podaj x ="; cin >> x;

p.sprawdz(y, x);

system("PAUSE");

}