**Hubert Bednarski**Podstawy Sztucznej Inteligencji  
IS, semestr V, gr. Proj. 1  
Data oddania: 20.10.2017r.

**Budowa i działanie perceptronu**sprawozdanie

Cel ćwiczenia:

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

Wykonanie:

Implementację wykonałem w języku C++, korzystając ze środowiska Microsoft Visual Studio 2015.   
Zaimplementowałem działanie perceptronu dla funkcji logicznej OR 2 zmiennych.  
Do uczenia perceptronu wykorzystałem algorytm szukania wag – regułę perceptronu. Początkowa wartość wag jest generowana losowo (w przedziale 0-1). Dane uczące także generowałem losowo, oczywiście w taki sposób, by perceptron działał poprawnie dla wybranej przeze mnie funkcji logicznej.

Przebieg najważniejszej fazy działania programu, a więc fazy uczenia, można zaprezentować w następujących punktach:

1.Obliczenie sumy na podstawie wartości danych wejściowych oraz wag i zwrócenie wartości wyjściowej – 0 lub 1.(funkcja\_sumujaca\_i\_aktywujaca)

2.Obliczenie różnicy między wynikiem rzeczywistym, a tym otrzymanym przez perceptron. (zmienna localError)

3.Skorygowanie wartości wag, tak by wyniki były możliwie jak najbardziej zbliżone do prawdziwych. (w funkcji nauczanie)

4.Zakończenie procesu uczenia perceptronu w momencie osiągnięcia założonego maksimum iteracji lub w przypadku, gdy zmienna globalError wykaże, że błąd obliczeń osiągnął wartość 0.

**Listing kodu programu:**

Plik perceptron.h:

#pragma once

#include <iostream>

#define LEARNING\_RATE 0.1

#define MAX\_ITERATION 16

**using** **namespace** std**;**

class Perceptron **{**

public**:**

static const int liczba\_zmiennych **=** 2**;**//liczba zmiennych, w tym przypadku - x oraz y

static const int liczba\_testow **=** 80**;**//liczba zestawow testujacych (w tym przypadku

private**:**

double x**[**liczba\_testow**];**

double y**[**liczba\_testow**];**

int wynik**[**liczba\_testow**];**

double localError**;**

double globalError**;**

double waga**[**liczba\_zmiennych**+**1**];**//[0]-waga x; [1]-waga y; [2] - bias

int iteracje **=** 0**;**

//??int klasa;

public**:**

double losuj\_punkty**();**

double losuj\_wagi**();**//w przedziale 0-1 do 3 miejsc po przecinku

//double funkcja\_sumujaca(int liczba\_zmiennych\_wejsciowych);

int funkcja\_sumujaca\_i\_aktywujaca**(**int**);**

int nauczanie**();**

int sprawdz\_wartosc**();**

**};**

Plik perceptron.cpp:

#include <ctime>

#include <cstdlib>

#include "Perceptron.h"

double Perceptron**::**losuj\_punkty**()** **{**

srand**(**time**(NULL));**

**for** **(**int i **=** 0**;** i **<** liczba\_testow**;** **++**i**)** **{**

**if** **(**i **%** 2 **==** 0**)** **{**

x**[**i**]** **=** **(**rand**()** **%** **(**50 **-** **-**0**)** **+** 1**);**

x**[**i**]** **/=** 10**;**

y**[**i**]** **=** **(**rand**()** **%** **(**50 **-** 0**)** **+** 1**);**// (rand() % (max-min)+1)+min

y**[**i**]** **/=** 10**;**

wynik**[**i**]** **=** 0**;**

cout **<<** x**[**i**]** **<<** "\t" **<<** y**[**i**]** **<<** "\t" **<<** wynik**[**i**]** **<<** endl**;**

**}**

**else** **{**

x**[**i**]** **=** **(**rand**()** **%** **(**100 **-** 50**)** **+** 1**)** **+** 50**;**

x**[**i**]** **/=** 10**;**

y**[**i**]** **=** **(**rand**()** **%** **(**100 **-** 50**)** **+** 1**)** **+** 50**;**

y**[**i**]** **/=** 10**;**

wynik**[**i**]** **=** 1**;**

cout **<<** x**[**i**]** **<<** "\t" **<<** y**[**i**]** **<<** "\t" **<<** wynik**[**i**]** **<<** endl**;**

**}**

**}**

**return** 0**;**

**}**

double Perceptron**::**losuj\_wagi**()** **{**

srand**(**time**(NULL));**

**for** **(**int i **=** 0**;** i **<=** liczba\_zmiennych**;** **++**i**)** **{**

waga**[**i**]** **=** **(**double**)(**rand**()** **%** 1000**)** **/** 1000**;**

//cout << waga[i] << endl;

**}**

**return** 0**;**

**}**

/\*double Perceptron::funkcja\_sumujaca(int liczba\_zmiennych\_wejsciowych) {//jako liczba zmiennych wejsciowych bedzie liczba testow

double suma = 0;

for (int i = 0; i < liczba\_zmiennych\_wejsciowych; ++i) {

suma += x[i] \* waga[0] + y[i] \* waga[1] + waga[2];

}

return suma;

}\*/

int Perceptron**::**funkcja\_sumujaca\_i\_aktywujaca**(**int index**){**

double suma **=** 0**;**

suma **+=** x**[**index**]** **\*** waga**[**0**]** **+** y**[**index**]** **\*** waga**[**1**]** **+** waga**[**2**];**

**if** **(**suma **>=** 0**)**

**return** 1**;**

**else**

**return** 0**;**

**}**

int Perceptron**::**nauczanie**()** **{**

int wynik**;**

**do** **{**

globalError **=** 0**;**

iteracje**++;**

**for** **(**int i **=** 0**;** i **<** MAX\_ITERATION**;** i**++)**

**{**

wynik **=** funkcja\_sumujaca\_i\_aktywujaca**(**i**);**

localError **=** **this->**wynik**[**i**]** **-** wynik**;**

waga**[**0**]** **+=** LEARNING\_RATE **\*** localError **\*** x**[**i**];**

waga**[**1**]** **+=** LEARNING\_RATE **\*** localError **\*** y**[**i**];**

waga**[**2**]** **+=** LEARNING\_RATE **\*** localError**;**

globalError **+=** **(**localError**\***localError**);**

**}**

cout **<<** "Liczba iteracji:\t" **<<** iteracje **<<** "\nBlad obliczen:\t\t" **<<** sqrt**(**globalError **/** MAX\_ITERATION**)** **<<** endl **<<** endl**;**

**}** **while** **(**globalError **!=**0 **&&** iteracje **<=** MAX\_ITERATION**);**

cout **<<** "Wagi koncowe, otrzymane po nauczaniu:\t" **<<** waga**[**0**]** **<<** "\t" **<<** waga**[**1**]** **<<** "\t" **<<** waga**[**2**]** **<<** endl**;**

**return** 0**;**

**}**

int Perceptron**::**sprawdz\_wartosc**()** **{**

int wartosc**;**

double x**,** y**;**

cout **<<** "Podaj x: " **<<** endl**;**

cin **>>** x**;**

//x \*= 10;

cout **<<** "Podaj y: " **<<** endl**;**

cin **>>** y**;**

//y \*= 10;

double suma **=** 0**;**

suma **+=** x **\*** waga**[**0**]** **+** y **\*** waga**[**1**]** **+** waga**[**2**];**

**if** **(**suma **>=** 0**)**

wartosc **=** 1**;**

**else**

wartosc **=** 0**;**

cout **<<** "Wartosc dla podanych punktow wyniesie: " **<<** wartosc **<<** endl**;**

**return** wartosc**;**

**}**

Plik main.cpp:

#include <iostream>

#include <ctime>

#include <cstdlib>

#include "Perceptron.h"

**using** **namespace** std**;**

int main**()** **{**

//srand(time(NULL));

Perceptron **\*** p **=** **new** Perceptron**();**

p**->**losuj\_punkty**();**

p**->**losuj\_wagi**();**

p**->**nauczanie**();**

p**->**sprawdz\_wartosc**();**

system**(**"pause"**);**

**return** 0**;**

**}**

Testy:

Przykładowe wyniki uczenia perceptronu:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Współ. Uczenia /nr wagi | 0.1 (6) | 0.1(9) | 0.1(11) | 0.04(8) | 0.04(12) | 0.04(16) | 0.3(6) | 0.3(4) | 0.3(7) |
| 1 | 0.186 | 0.025 | -0.384 | 0.039 | 0.163 | -0.344 | 0.677 | 0.09 | 0.391 |
| 2 | 0.19 | 0.333 | 0.943 | 0.084 | 0.019 | 0.745 | 0.469 | 0.679 | 0.968 |
| 3 | -1.701 | -2.003 | -2.475 | -0.571 | -0.808 | -1.476 | -5.318 | -3.583 | -5.531 |

W nawiasie przy współczynniku uczenia podana liczba iteracji.

Podczas testów przy użyciu dużych wartości współczynnika uczenia się (bliskich do 0.8) wartości wag otrzymywanych na końcu były dużo wyższe, co warto odnotować.

Wykres zależności liczby iteracji od współczynnika uczenia:

Wykres zależności liczby iteracji od liczby danych uczących:

Wnioski:

1. Na szybkość uczenia się perceptronu największy wpływ ma współczynnik uczenia się perceptronu. Im wyższy współczynnik zdecydujemy się ustawić tym szybciej będzie przebiegało uczenie perceptronu. Trzeba jednak uważać, by nie ustawić zbyt dużej wartości, bo wtedy uczenie perceptronu może stać się nieefektywne. Natomiast jeśli wartość będzie zbyt niska, to uczenie będzie przebiegać wolniej niż by mogło.
2. Poza współczynnikiem uczenia dużą rolę odgrywa też odpowiedni dobór danych uczących oraz ich ilość. Jeśli proces uczenia zostanie przeprowadzony na zbyt małej ilości danych, to nie osiągnie się zamierzonych skutków.
3. Ponieważ mieliśmy do czynienia z prostą funkcją logiczną ustalanie maksymalnej wartości iteracji okazało się być zbyteczne. Program zawsze był w stanie uzyskać poprawność działania na poziomie zbliżonym do 100%. Nie należy jednak wyciągać pochopnych wniosków i nie wolno ignorować tej wartości w przyszłości – dla trudniejszych problemów uzyskanie takiej poprawności może okazać się niemożliwe, a co więcej wskutek zbyt długo trwającego procesu uczenia może pojawić się tak zwane zjawisko przeuczenia sieci neuronowej.