

**Podstawy Sztucznej Inteligencji**

Scenariusz 1: Budowa i działanie perceptronu

Wykonała:

**Magdalena Migas**

**Wydział Inżynierii Metali i Informatyki Przemysłowej**

**Informatyka Stosowana**

1. **Cel ćwiczenia**

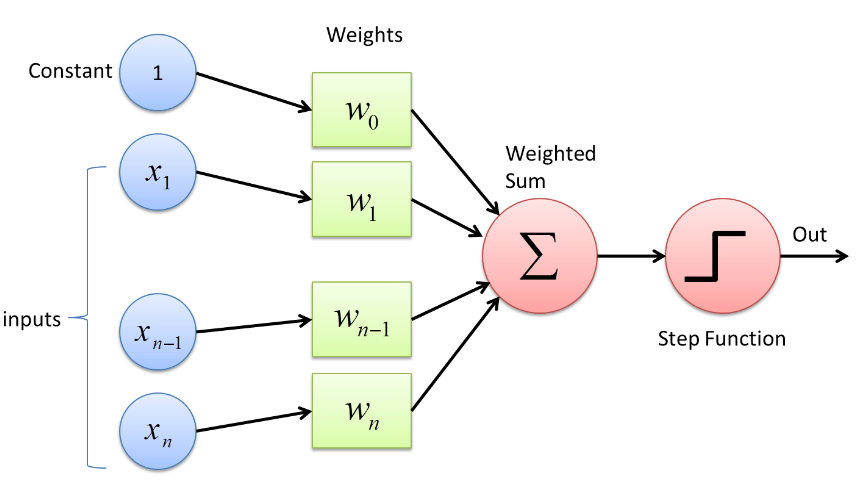
Celem wykonanego ćwiczenia było poznanie budowy i działania perceptronu poprzez implementację oraz uczenie perceptronu realizującego funkcję logiczną dwóch zmiennych.

1. **Definicja perceptronu**

Perceptron prosty jest najprostszą Sztuczną Siecią Neuronową, używaną do klasyfikacji binarnej. Perceptron składa się z pojedynczego neuronu z regulacją wag.

Perceptron składa się z:

* n wejść: x1, …, xn
* n wag: w1, … , wn
* progu θ
* wyjścia y.



Perceptron zwraca następującą wartość:

Gdzie jest funkcją aktywującą, która może przyjmować różne formy. Może to być na przykład funkcja progowa bipolarna:

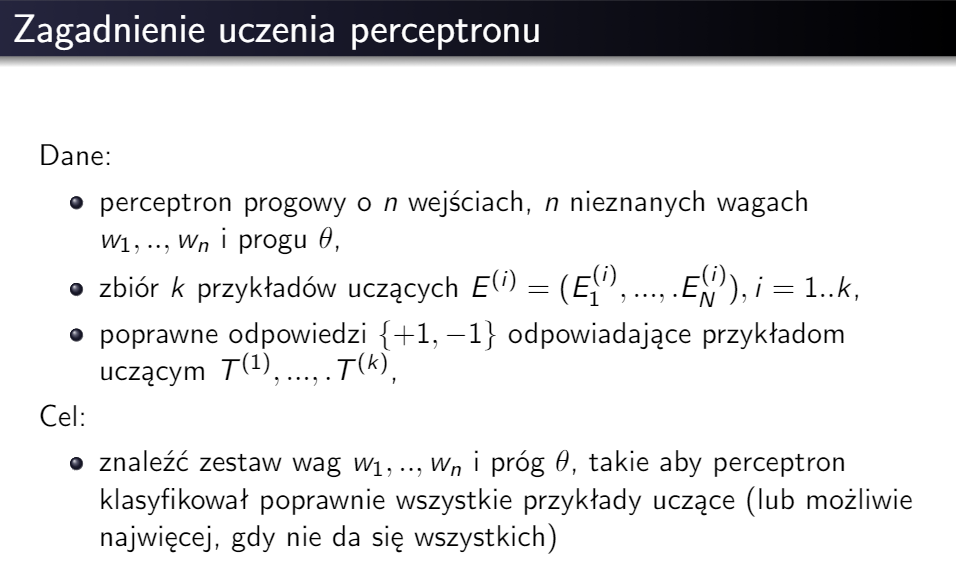
Wartość jest w tym przypadku dodatkowym parametrem opisującym perceptron. Ten typ funkcji modeluje wysyłanie impulsu po przekroczeniu pewnego progu, które to zachowanie z pewnym przybliżeniem charakteryzuje komórki neuronowe.

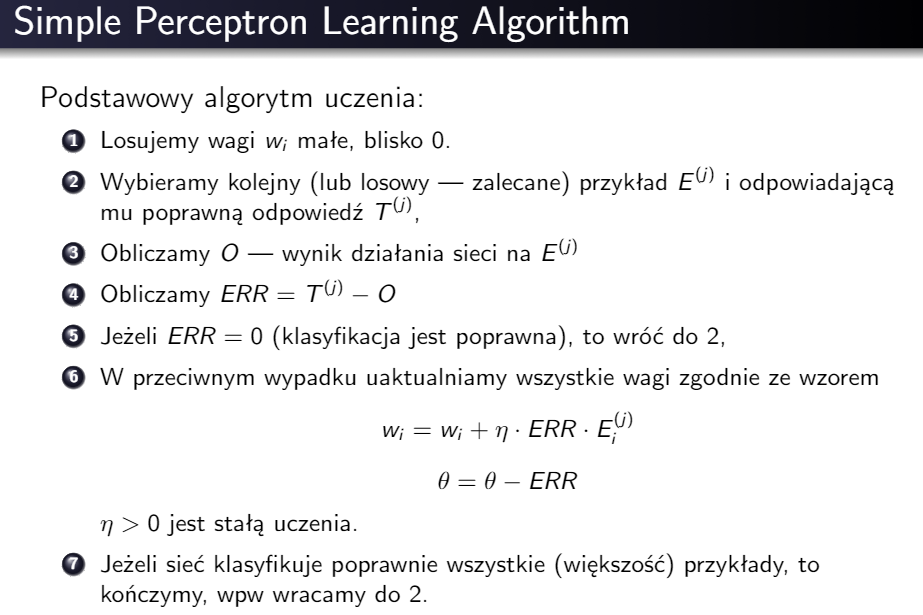
1. **Uczenie perceptronu**

Uczenie polega na automatycznym doborze wag w SSN, na podstawie zbioru przykładów nazwanych zbiorem uczącym. Zaczyna sią z losowymi małymi wagami i iteracyjnie zmienia się wagi, dopóki wszystkie przykłady uczące nie zostaną poprawnie sklasyfikowane.

Algorytm zaczerpnięto z następującego źródła:

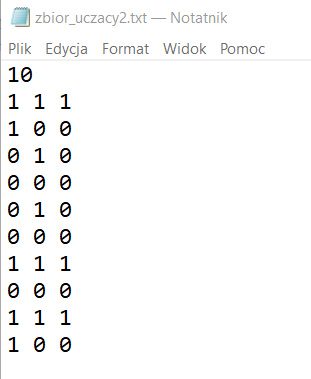
<http://www-users.mat.umk.pl/~rudy/wsn/wyk/wsn-wyklad-01-Perc.pdf>





1. **Zestaw danych uczących**

Przykładowe dane uczące znajdują się w pliku zbior\_uczacy1.txt. W pliku tym pierwsza linia zawiera liczbę oznaczającą liczność zbioru uczącego a kolejne linie zawierają dane, które mają posłużyć do wyuczenia perceptronu. Zawartość tego pliku przedstawia się następująco:



1. **Uzyskane wyniki**

W wykonany projekcie zaimplementowano perceptron, który realizuję funkcję logiczną jaką jest koniunkcja.



Przeprowadzając proces uczenia perceptronu zastosowano warianty różniące się między sobą współczynnikiem uczenia oraz licznością danych uczących. Dla każdego z wariantów przeprowadzono po 10 prób i w poniższej tabeli zestawiono średnie z uzyskanych wyników.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Numer wariantu** | **Współczynnik uczenia η** | **Liczba danych uczących** | **Liczba potrzebnych iteracji do wyuczenia** | **Liczba błędów przy testowaniu** |
| **1** | 1 | 30 | 2,8 | 0 |
| **2** | 0,5 | 30 | 3,2 | 0 |
| **3** | 0,25 | 30 | 4,1 | 0 |
| **4** | 0,1 | 30 | 8,2 | 0 |
| **5** | 1 | 10 | 3,9 | 0 |
| **6** | 0,5 | 10 | 5,3 | 0 |
| **7** | 0,25 | 10 | 7,9 | 0 |
| **8** | 0,1 | 10 | 17,1 | 0 |
| **9** | 1 | 5 | 11 | 0 |
| **10** | 0,5 | 5 | 19 | 0 |
| **11** | 0,25 | 5 | 39,4 | 0 |
| **12** | 0,1 | 5 | 103 | 0 |

Rysunek 1: Zależność potrzebnej liczby iteracji od współczynnika uczenia dla 3 wariantów różniących się licznością zbioru uczącego

Rysunek 2: Zależność potrzebnej liczby iteracji od liczności zbioru uczącego dla 3 wariantów różniących się współczynnikiem uczenia

1. **Analiza wyników i wnioski**

W wykonanym projekcie zastosowano perceptron do obliczania koniunkcji dwóch zmiennych logicznych. Zaimplementowany perceptron poddano uczeniu przy zastosowaniu różnych wariantów.

W pierwszej kolejności sprawdzono jak liczność zbioru uczącego wpływa na szybkość uczenia perceptronu. W tym celu przeprowadzono próby dla zbiorów uczących liczących kolejno 5, 10 i 30 przykładów. Jak należało się spodziewać im bardziej liczny jest zbiór uczący tym szybciej uczy się perceptron. Dla przykładu, kiedy przyjęto współczynnik uczenia 0.1 i zrobiono po 10 prób dla różniących się licznością zbiorów uczących to okazało się, że w takim przypadku do wyuczenia się perceptron potrzebuje średnio 8.2 iteracji dla zbioru 30 elementowego, 17.1 iteracji dla zbioru 10 elementowego i aż 103 iteracje dla zbioru 5 elementowego. Zależność tą można łatwo zaobserwować na rysunku numer 2.

W kolejnym etapie sprawdzono jak przyjęty współczynnik uczenia wpływa na szybkość uczenia się perceptronu. Współczynnik ten wpływa na szybkość uczenia poprzez bezpośrednie oddziaływanie na gradient zmiany. Zbyt duża jego wartość powoduje poruszanie się po wierzchołkach płaszczyzny błędu i pomijanie zagłębień z minimami. Zbyt mała wartość może spowodować utknięcie procesu uczenia w minimum lokalnym. Najczęściej stosowane wartości współczynnika uczenia zawierają się pomiędzy 0.01 a 0.6. Na podstawie przeprowadzonych prób stwierdzono, że im mniejsza jest wartość współczynnika uczenia tym więcej iteracji potrzebuje perceptron do wyuczenia się. Zależność tą ilustruje rysunek numer 1.

Jeśli chodzi o błędy testowania to podczas żadnej z prób nie stwierdzono ani jednego błędu. Wynika to głównie z niezbyt skomplikowanej natury problemu jakim była koniunkcja dwóch wartości logicznych.

1. **Listing kodu programu**

#include<iostream>

#include<cstdlib>

#include<fstream>

#include<string>

#include<ctime>

using namespace std;

class Perceptron {

public:

int wymiar;

int liczbaWejsc;

int liczbaIteracji;

double \*\*wejscia;

double \*wyniki;

double \*wagi;

double wspUczenia = 0.1;

double eta;

double obliczWartosc(double suma);

void losujWagi();

void uczPerceptron();

void test(int p, int q);

Perceptron();

};

Perceptron::Perceptron() {

fstream plik;

plik.open("zbior\_uczacy3.txt");

if (plik.good())

{

string linijka;

getline(plik, linijka);

wymiar = atoi(linijka.c\_str());

liczbaWejsc = 2;

wejscia = new double\*[liczbaWejsc];

for (int i = 0;i < liczbaWejsc;i++)

wejscia[i] = new double[wymiar];

wyniki = new double[wymiar];

wagi = new double[liczbaWejsc];

int licznik = 0;

while (getline(plik, linijka))

{

wejscia[0][licznik] = atoi(linijka.substr(0, linijka.find\_first\_of(' ')).c\_str());

linijka = linijka.substr(linijka.find\_first\_of(' ') + 1, linijka.length());

wejscia[1][licznik] = atoi(linijka.substr(0, linijka.find\_first\_of(' ')).c\_str());

linijka = linijka.substr(linijka.find\_first\_of(' ') + 1, linijka.length());

wyniki[licznik] = atoi(linijka.c\_str());

cout << wejscia[0][licznik] <<" "<< wejscia[1][licznik]<<" "<< wyniki[licznik]<<endl;

licznik++;

}

plik.close();

}

else

{

cout << "blad otwarcia pliku!";

}

}

double Perceptron::obliczWartosc(double suma) {

if (suma > eta)

return 1;

else

return 0;

}

void Perceptron::losujWagi() {

for(int i=0; i<liczbaWejsc; i++)

wagi[i]= -10 + (double)rand() /RAND\_MAX\* (20);

cout << wagi[0] << wagi[1] << endl;

}

void Perceptron::test(int p, int q)

{

double wynik= p \* wagi[0] + q \* wagi[1];

cout << p << " ^ " << q << " = " << obliczWartosc(wynik) << endl;

}

void Perceptron::uczPerceptron() {

losujWagi();

double licznikBledow = 0;

double y = 0;

double blad = 0;

liczbaIteracji = 1000000;

for (int i = 0; i < liczbaIteracji; i++)

{

licznikBledow = 0;

for (int j = 0; j < wymiar; j++)

{

y = wejscia[0][j]\*wagi[0]+wejscia[1][j]\*wagi[1];

if (wyniki[j] != obliczWartosc(y))

{

blad = wyniki[j] - obliczWartosc(y);

wagi[0] = wagi[0] + wspUczenia\*blad \* wejscia[0][j];

wagi[1]= wagi[1] + wspUczenia\*blad \* wejscia[1][j];

eta = eta - blad;

licznikBledow++;

}

}

if (licznikBledow == 0)

{

cout << "Liczba potrzebnych iteracji:" << i+1 << endl;

break;

}

}

cout << "Perceptron nauczony :-)" << endl;

cout<<"Uzyskane wagi to:"<<"w[0]:"<<wagi[0]<<"w[1]:"<<wagi[1]<<"eta:"<<eta<<endl;

test(1, 1);

test(0, 1);

test(1, 0);

test(0, 0);

}

int main()

{

srand(time(NULL));

Perceptron przykladowy;

przykladowy.uczPerceptron();

getchar();

}

1. **Literatura**

* <http://home.agh.edu.pl/~horzyk/lectures/biocyb/BIOCYB-SieciNeuronowe.pdf>
* <http://www-users.mat.umk.pl/~rudy/wsn/wyk/wsn-wyklad-01-Perc.pdf>
* <http://www.cs.put.poznan.pl/rklaus/assn/percep.htm>
* <https://appliedgo.net/perceptron/>
* <https://www.ii.uni.wroc.pl/~aba/teach/NN/w4.pdf>
* <http://www-users.mat.umk.pl/~piersaj/www/contents/teaching/wsn2013/wsn-notatki.pdf>
* <http://wikizmsi.zut.edu.pl/uploads/7/74/PSI_ZIP_S1_W5.pdf>