

**Podstawy Sztucznej Inteligencji**

Scenariusz 2: Budowa i działanie sieci jednowarstwowej

Wykonała:

**Magdalena Migas**

**Wydział Inżynierii Metali i Informatyki Przemysłowej**

**Informatyka Stosowana**

1. **Cel ćwiczenia**

Celem ćwiczenia było poznanie budowy i działania jednowarstwowych sieci neuronowych

oraz uczenie rozpoznawania wielkości liter.

1. **Przebieg ćwiczenia**

a) Wygenerowanie danych uczących i testujących, zawierających 10 dużych i 10

małych liter polskiego alfabetu w postaci dwuwymiarowej tablicy o wymiarach 5x7.

b) Implementacja dwóch jednowarstwowych sieci - każda wg. innego algorytmu.

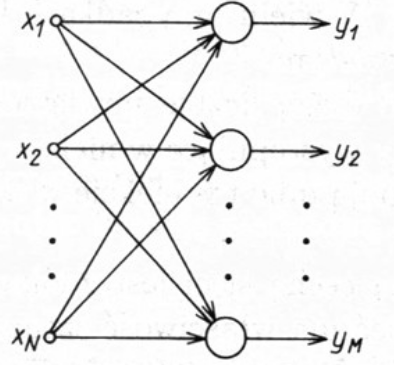
c) Uczenie sieci przy różnych współczynnikach uczenia.

d) Testowanie sieci.

1. **Jednowarstwowa sieć neuronowa**

Sieć jednokierunkowa jednowarstwowa - właściwości:

* neurony ułożone w jednej warstwie zasilanej z węzłów wejściowych,
* połączenie węzłów wejściowych i wyjściowych jest pełne (każdy węzeł jest połączony z każdym neuronem),
* przepływ sygnałów występuje w jednym kierunku, od wejścia do wyjścia,
* węzły wejściowe nie tworzą warstwy neuronów, gdyż nie zachodzi w nich żaden proces obliczeniowy,
* sieć tego rodzaju nazywa się perceptronem jednowarstwowym.



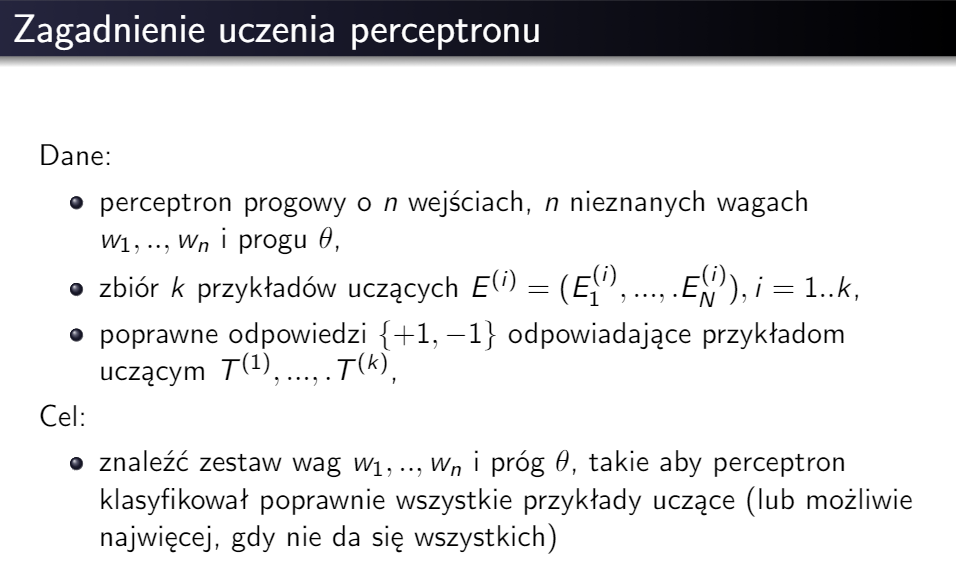
1. **Budowa i uczenie sieci neuronowej jednowarstwowej**

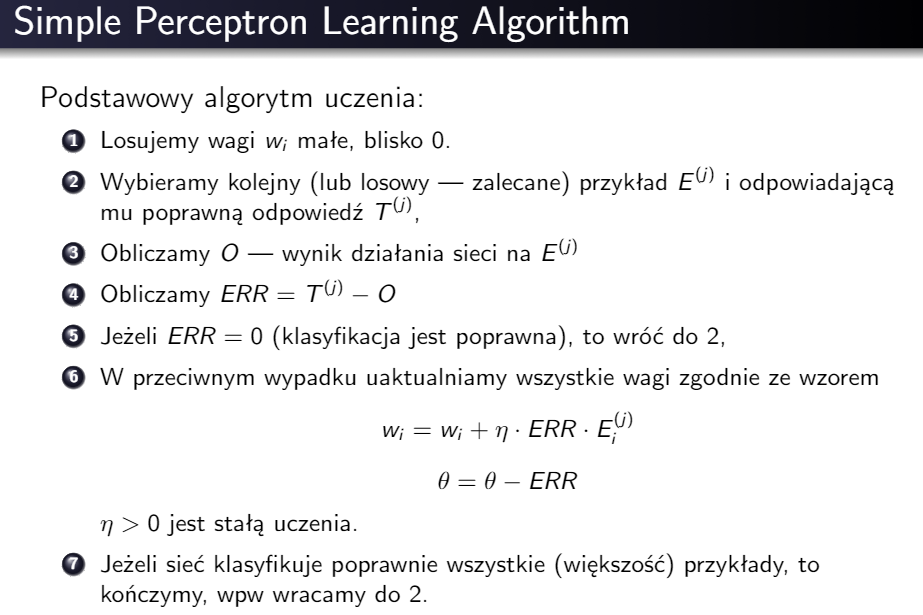
Sieć neuronowa miała na celu rozpoznanie wielkości liter polskiego alfabetu. Zbudowana została z 1 perceptronu zaimplementowanego podobnie jak w poprzednim zadaniu. Perceptron posiada 35 wejść i 35 wag gdyż każda litera została zapisana na matrycy 35-elementowej.

Pierwszy wykorzystany algorytm to Simple Perceptron Learning Algorithm, który wykorzystuje jako funkcję aktywacji funkcję progową unipolarną. Uczenie rozpoczyna się 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>





Drugi zastosowany algorytm to drobna modyfikacja algorytmu pierwszego. Jedyna zmiana polega na zastąpieniu funkcji progowej unipolarnej na ciągłą funkcję sigmoidalną. W tym algorytmie stosuje się następujące wzory:

* Funkcja aktywacji:
* Pochodna funkcji:
* Wzór na modyfikację wag:

1. **Zestaw danych uczących**

Przykładowe dane uczące znajdują się w pliku zbior\_uczacy1.txt. W pliku tym zawarto 20 liter polskiego alfabetu, które z matryc 35-elementowych rozwinięto na odpowiadające im wektory, dla przykładu litera A:

**0 1 1 1 0  
1 0 0 0 1  
1 0 0 0 1  
1 1 1 1 1 🡪 0 1 1 1 0 1 0 0 0 1 1 0 0 0 1 1 1 1 1 1 1 0 0 0 1 1 0 0 0 1 1 0 0 0 1  
1 0 0 0 1   
1 0 0 0 1  
1 0 0 0 1**

Ponadto każdej literze przyporządkowana jest cyfra według zasady: mała litera to cyfra 0, wielka litera to cyfra 1.

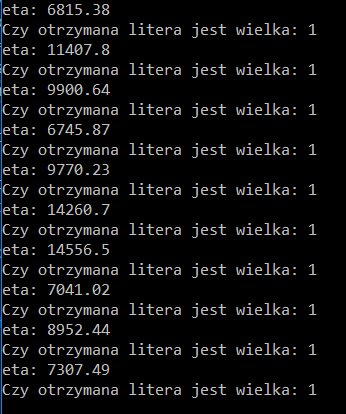
1. **Uzyskane wyniki**

Przeprowadzając proces uczenia sieci neuronowej jednowarstwowej zastosowano warianty różniące się między sobą współczynnikiem uczenia oraz licznością zbioru uczącego. Dla każdego z wariantów przeprowadzono po 20 prób i w poniższej tabeli zestawiono średnie z uzyskanych wyników.

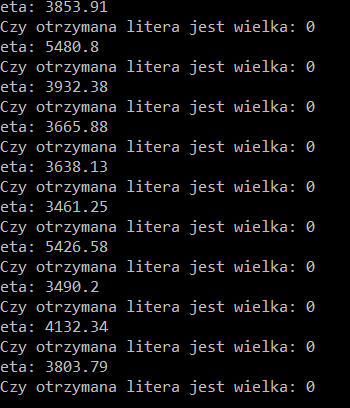
**Wariant I:**

* Zestaw uczący zawiera 20 liter: 10 wielkich i 10 małych;
* Zastosowano algorytm z funkcją aktywacji w postaci funkcji progowej unipolarnej
* Testy przeprowadzono dla 4 różnych współczynników uczenia: 1; 0.5; 0.25; 0.1
* W testach sprawdzano czy sieć poprawnie sklasyfikuje litery: M i s.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Numer wariantu** | **Współczynnik uczenia η** | **Liczba danych uczących** | **Średnia liczba potrzebnych iteracji do wyuczenia** | **Liczba błędów przy testowaniu** |
| **1** | 1 | 20 | >1000 | 0 |
| **2** | 0,5 | 20 | >1000 | 0 |
| **3** | 0,25 | 20 | >1000 | 0 |
| **4** | 0,1 | 20 | >1000 | 0 |



Rysunek 1: Test dla litery M i współczynnika uczenia równego 1



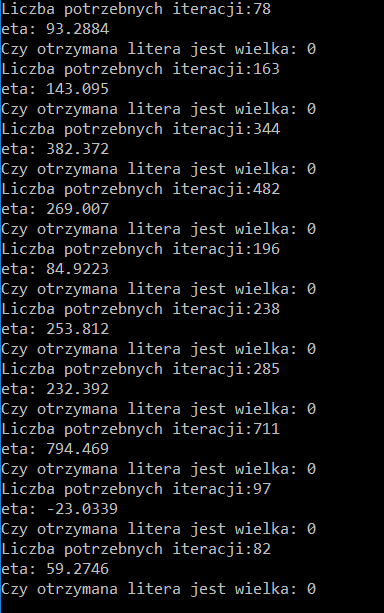
Rysunek 2: Test dla litery s i współczynnika uczenia równego 0.1

W wariancie numer 1 gdzie zastosowano funkcję progową unipolarną i zbiór uczący 20-elementowy możemy zaobserwować, że sieć potrzebuje dużej liczby iteracji aby poprawnie klasyfikować wszystkie elementy ze zbioru uczącego. W efekcie jednak uzyskane współczynniki wagowe pozwalają na poprawne określenie wielkości liter ze zbioru testującego gdyż podczas testów nie napotkano na ani jeden błędnie sklasyfikowany element.

**Wariant II:**

* Zestaw uczący zawiera 10 liter: 5 wielkich i 5 małych;
* Zastosowano algorytm z funkcją aktywacji w postaci funkcji progowej unipolarnej
* Testy przeprowadzono dla 4 różnych współczynników uczenia: 1; 0.5; 0.25; 0.1
* W testach sprawdzano czy sieć poprawnie sklasyfikuje litery: M i s.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Numer wariantu** | **Współczynnik uczenia η** | **Liczba danych uczących** | **Średnia liczba potrzebnych iteracji do wyuczenia** | **Liczba błędów przy testowaniu** |
| **1** | 1 | 10 | 354 | 0 |
| **2** | 0,5 | 10 | 301 | 0 |
| **3** | 0,25 | 10 | 292 | 0 |
| **4** | 0,1 | 10 | 268 | 0 |



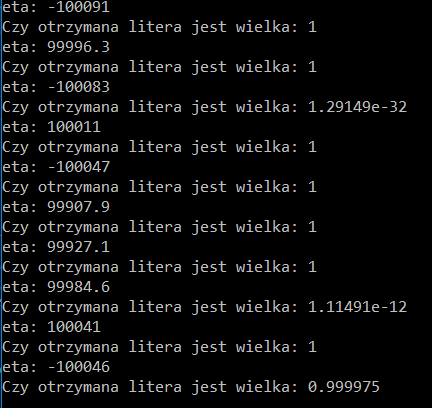
Rysunek 3: Test dla litery s i współczynnika uczenia równego 0.1

W wariancie numer 2 gdzie zastosowano funkcję progową unipolarną i zbiór uczący 10-elementowy możemy zaobserwować, że sieć potrzebuje zależnie od przyjętego współczynnika uczenia od 268 do 354 iteracji aby poprawnie klasyfikować wszystkie elementy ze zbioru uczącego. W efekcie jednak uzyskane współczynniki wagowe pozwalają na poprawne określenie wielkości liter ze zbioru testującego gdyż podczas testów nie napotkano na ani jeden błędnie sklasyfikowany element. Wykonany wykres pokazuje, że liczba potrzebnych iteracji rośnie wraz ze wzrostem współczynnika uczenia gdyż najmniej iteracji uzyskano przy współczynniku uczenia równym 0.1 a najwięcej przy współczynniku wynoszącym 1.

**Wariant III:**

* Zestaw uczący zawiera 20 liter: 10 wielkich i 10 małych;
* Zastosowano algorytm z funkcją aktywacji w postaci funkcji ciągłej sigmoidalnej
* Testy przeprowadzono dla 4 różnych współczynników uczenia: 1; 0.5; 0.25; 0.1
* W testach sprawdzano czy sieć poprawnie sklasyfikuje litery: M i s.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Numer wariantu** | **Współczynnik uczenia η** | **Liczba danych uczących** | **Średnia liczba potrzebnych iteracji do wyuczenia** | **Liczba błędów przy testowaniu** |
| **1** | 1 | 10 | >1000 | 4 |
| **2** | 0,5 | 10 | >1000 | 3 |
| **3** | 0,25 | 10 | >1000 | 3 |
| **4** | 0,1 | 10 | >1000 | 2 |



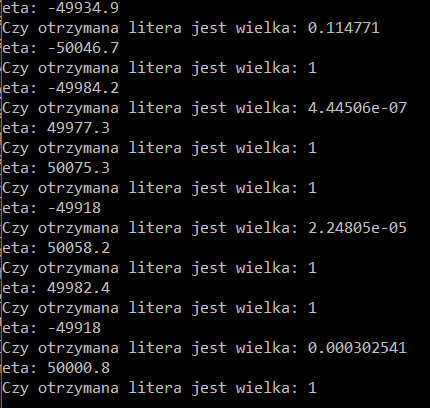
Rysunek 4: Test dla litery M i współczynnika ufności równego 0.1

W wariancie numer 3 gdzie zastosowano funkcję ciągłą sigmoidalną i zbiór uczący 20-elementowy możemy zaobserwować, że sieć znowu potrzebuje dużej liczby iteracji do wyuczenia i tak naprawdę nie osiąga takich współczynników wagowych aby błąd testowania wynosił 0. W efekcie przy testowaniu na 20 przykładach kilka z nich zostaje błędnie sklasyfikowanych. Na podstawie wykonanego wykresu i przedstawionej tabeli widzimy, że błędów jest tym więcej i większą wartość osiąga współczynnik uczenia.

**Wariant IV:**

* Zestaw uczący zawiera 10 liter: 5 wielkich i 5 małych;
* Zastosowano algorytm z funkcją aktywacji w postaci funkcji ciągłej sigmoidalnej
* Testy przeprowadzono dla 4 różnych współczynników uczenia: 1; 0.5; 0.25; 0.1
* W testach sprawdzano czy sieć poprawnie sklasyfikuje litery: M i s.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Numer wariantu** | **Współczynnik uczenia η** | **Liczba danych uczących** | **Średnia liczba potrzebnych iteracji do wyuczenia** | **Liczba błędów przy testowaniu** |
| **1** | 1 | 10 | >1000 | 7 |
| **2** | 0,5 | 10 | >1000 | 5 |
| **3** | 0,25 | 10 | >1000 | 4 |
| **4** | 0,1 | 10 | >1000 | 3 |



Rysunek 5: Test dla litery M i współczynnika ufności równego 0.25

Wariant numer 4, gdzie zastosowano funkcję ciągłą sigmoidalną i zbiór uczący 10-elementowy, jest bardzo podobny do wariantu numer 3. Jedyną różnicę można zaobserwować w ilości błędów podczas testowania, gdyż nieco się ona zwiększyła. Zależność tę przedstawiono na wykresie, z którego wynika, że dla tych samych współczynników uczenia ilość błędów podczas testowania jest zawsze większa w przypadku mniej licznego zbioru uczącego. Wynika to z bardzo prostej przyczyny, a mianowicie że zaimplementowany perceptron miał zbyt mało przykładów by móc się skutecznie wyuczyć.

1. **Wnioski**

* Opracowane zagadnienie polegało na wykorzystaniu jednowarstwowej sieci neuronowej do rozpoznawania wielkości liter polskiego alfabetu. Wykorzystano do tego celu dwa algorytmy uczenia sieci różniące się między sobą funkcjami aktywacji: jeden z nich korzystał z funkcji progowej unipolarnej, drugi natomiast z ciągłej funkcji sigmoidalnej. Porównując wyniki uzyskane podczas testów należy stwierdzić, że w przypadku omawianego problemu zdecydowanie lepsze wyniki osiągnięto wykorzystując algorytm pierwszy gdyż podczas jego testowania nie stwierdzono żadnych błędów. Podczas testów algorytmu z funkcją sigmoidalną natomiast zaobserwowano błędy w rozpoznawaniu wielkości liter ze zbioru testującego.
* Podczas testów zrealizowano szereg wariantów. Jednym z aspektów poddanym próbie był wpływ współczynnika uczenia na otrzymane rezultaty. Na podstawie uzyskanych wyników trzeba jasno stwierdzić, że im mniejszą wartość przyjmuje współczynnik uczenia tym mniej iteracji jest potrzebne do wyuczenia sieci i tym mniej błędów występuje podczas testowania.
* W dalszej kolejności testowano także jaki wpływ na proces uczenia ma liczność zbioru testującego. W tym przypadku wyciągnięto dwa główne wnioski. Po pierwsze mniejszy zbiór uczący powoduje, że sieć potrzebuje mniej iteracji do wyuczenia się. Po drugie zaś mniejszy zbiór uczący może sprawić, że sieć nie wyuczy się w pełni i spowoduje to występowanie błędów podczas testowania.

1. **Listing kodu**

#include<iostream>

#include<cstdlib>

#include<fstream>

#include<string>

#include<ctime>

#include<list>

#include<math.h>

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);

double obliczWartosc2(double suma);

void losujWagi();

void uczPerceptron();

void test(int \*testTable);

int indeksLitery;

char litera;

double wyjscie;

Perceptron();

};

Perceptron::Perceptron() {

fstream plik;

plik.open("zbior\_uczacy2.txt");

if (plik.good())

{

liczbaWejsc = 35;

wymiar = 10;

wejscia = new double\*[wymiar];

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

wejscia[i] = new double[liczbaWejsc];

wyniki = new double[wymiar];

wagi = new double[liczbaWejsc];

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

{

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

{

plik >> wejscia[j][i];

}

plik >> wyniki[j];

}

}

else

{

cout << "blad otwarcia pliku!";

}

}

double Perceptron::obliczWartosc(double suma) {

if (suma > eta)

return 1;

else

return 0;

}

double Perceptron::obliczWartosc2(double suma) {

return (1 / (1 + exp(-suma)));

}

void Perceptron::losujWagi() {

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

{

wagi[i] = -20 + (double)rand() / RAND\_MAX\* (40);

}

eta = -100 + (double)rand() / RAND\_MAX\* (200);

}

void Perceptron::test(int \*testTable)

{

double wynik = 0;

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

wynik += (testTable[i] \* wagi[i]);

cout << "Czy otrzymana litera jest wielka: " << obliczWartosc2(wynik) << endl;

}

void Perceptron::uczPerceptron() {

losujWagi();

double licznikBledow = 0;

double y = 0;

double blad = 0;

liczbaIteracji = 1000;

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

{

licznikBledow = 0;

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

{

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

y += (wejscia[j][k] \* wagi[k]);

if (wyniki[j] != obliczWartosc2(y))

{

blad = wyniki[j] - obliczWartosc2(y);

double pochodna = obliczWartosc2(y)\*(1-obliczWartosc2(y));

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

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

wagi[l]=wagi[l] wspUczenia\*blad\*wejscia[j][l]\*pochodna;

eta = eta - blad;

licznikBledow++;

}

}

if (licznikBledow == 0)

{

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

break;

}

}

cout << "eta: " << eta << endl;

int tab[35] = { 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1 };// K

int tab2[35] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1 };

test(tab);

}

int main()

{

srand(time(NULL));

Perceptron A, B, C, D, E, F, G, H, I, J;

A.uczPerceptron();

B.uczPerceptron();

C.uczPerceptron();

D.uczPerceptron();

E.uczPerceptron();

F.uczPerceptron();

G.uczPerceptron();

H.uczPerceptron();

I.uczPerceptron();

J.uczPerceptron();

getchar();

}

1. **Literatura**

* <http://pracownik.kul.pl/files/31717/public/Model_neuronu_sigmoidalnego.pdf>
* <http://zsi.tech.us.edu.pl/~nowak/si/SI_w4.pdf>
* <https://pl.wikipedia.org/wiki/Sie%C4%87_neuronowa>
* <http://iisi.pcz.pl/nn/arch.php?art=2>
* <http://www.cs.put.poznan.pl/jstefanowski/aed/TPDANN.pdf>