

**Podstawy Sztucznej Inteligencji**

Scenariusz 3: Budowa i działanie sieci wielowarstwowej

Wykonała:

**Magdalena Migas**

**Wydział Inżynierii Metali i Informatyki Przemysłowej**

**Informatyka Stosowana**

1. **Cel ćwiczenia**

Celem ćwiczenia jest poznanie budowy i działania wielowarstwowych sieci neuronowych poprzez uczenie z użyciem algorytmu wstecznej propagacji błędu rozpoznawania konkretnych liter alfabetu.

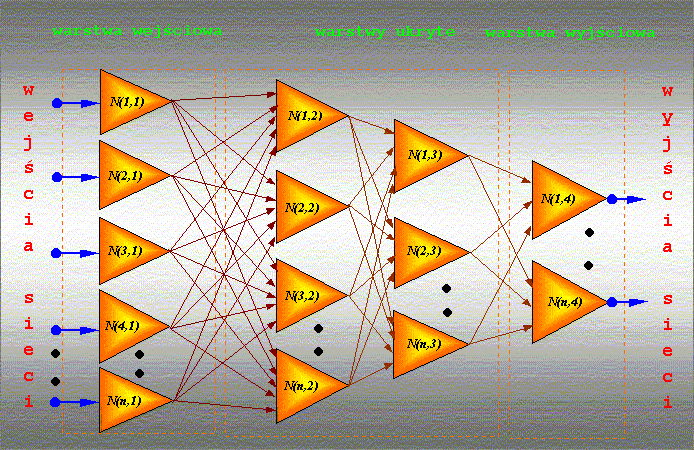
1. **Sieć wielowarstwowa**

Nazywana jest czasem perceptronem wielowarstwowym(MultiLayer Perceptron). Jej cechą charakterystyczną jest występowanie co najmniej jednej warstwy ukrytej neuronów, pośredniczącej w przekazywaniu sygnałów pomiędzy wejściami a wyjściami sieci.

W sieci warstwowej wszystkie perceptrony podzielone są na kolejno występujące po sobie warstwy rozłączne. Warstwa L+1-sza za wejścia przyjmuje wyniki z warstwy L-tej i tylko te. W sieciach MLP nie są dopuszczane bezpośrednie połączenia wewnątrz tej samej warstwy ani połączenia przeskakujące warstwę tj. z warstwy L do L+2, z L do L+3 itd.

Taka sieć składa się z trzech typów warstw:

* warstwa pierwsza - wejściowa (jednostki w tej warstwie są to jednostki wejściowe),
* warstwa ostatnia - wyjściowa (jednostki w tej warstwie są to jednostki wyjściowe),
* warstwy pomiędzy wejściową a wyjściową - ukryte (jednostki w tych warstwach są to jednostki ukryte).



1. **Algorytm wstecznej propagacji błędu**

Algorytm wstecznej propagacji błędu (backpropagation) jest obecnie podstawowym (i jednym z najskuteczniejszych) algorytmem nadzorowanego uczenia wielowarstwowych jednokierunkowych sieci neuronowych. Nazwa tego algorytmu wynika z kolejności obliczania sygnałów błędu d, która przebiega w kierunku odwrotnym niż przechodzenie sygnałów przez sieć, to znaczy od warstwy wyjściowej poprzez warstwy ukryte w kierunku warstwy wejściowej.

Ogólny schemat procesu trenowania sieci wygląda następująco:

1. Ustalamy topologię sieci, tzn. liczbę warstw, liczbę neuronów w warstwach.
2. Inicjujemy wagi losowo (na małe wartości).
3. Dla danego wektora uczącego obliczamy odpowiedź sieci (warstwa po warstwie).
4. Każdy neuron wyjściowy oblicza swój błąd, oparty na różnicy pomiędzy obliczoną odpowiedzią *y* oraz poprawną odpowiedzią *t*.
5. Błędy propagowane są do wcześniejszych warstw.
6. Każdy neuron (również w warstwach ukrytych) modyfikuje wagi na podstawie wartości błędu i wielkości przetwarzanych w tym kroku sygnałów.
7. Powtarzamy od punktu 3. dla kolejnych wektorów uczących. Gdy wszystkie wektory zostaną użyte, losowo zmieniamy ich kolejność i zaczynamy wykorzystywać powtórnie.
8. **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 w pliku tym do każdej z 20 liter przyporządkowany jest wektor składający się z 20 cyfr: dziewiętnastu zer i jednej jedynki. Wektor ten służy do rozpoznania jaka to litera: dla przykładu literze A odpowiada wektor:

**1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

literze B wektor:

**0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0**

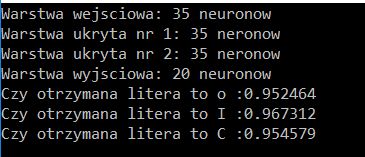
1. **Uzyskane wyniki**

Napisany program poddano testom z wykorzystaniem różnych współczynników uczenia sieci a także różnej ilości iteracji zadanej do wyuczenia sieci. Do zbioru testującego wybrano 3 litery: C, I, o. W każdym przypadku sprawdzano na ile stworzona sieć rozpoznaje daną literę.

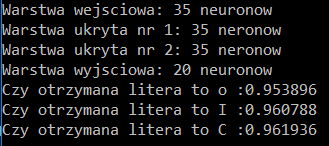
**Wariant I:**

* Zestaw uczący zawiera 20 liter: 10 wielkich i 10 małych;
* Zastosowano algorytm z funkcją aktywacji w postaci tangensa hiperbolicznego;
* Testy przeprowadzono dla 4 różnych współczynników uczenia: 0.01, 0.1, 0.2, 0.4;
* W testach sprawdzano czy sieć poprawnie sklasyfikuje litery: **C**, **I**, **o**.

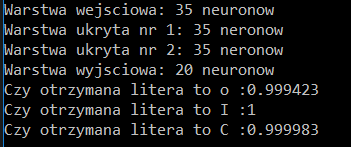
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Numer wariantu** | **Współczynnik uczenia η** | **Liczba danych uczących** | **Liczba potrzebnych iteracji do wyuczenia** | **Liczba błędów przy testowaniu** |
| **1** | 0.01 | 20 | 1000 | 0 |
| **2** | 0.1 | 20 | 1000 | 0 |
| **3** | 0.2 | 20 | 1000 | 0 |
| **4** | 0.4 | 20 | 1000 | 0 |



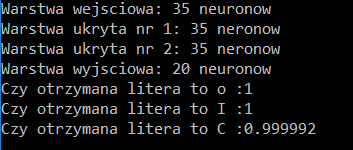
Rysunek 1: Test dla współczynnika uczenia równego 0.01



Rysunek 2: Test dla współczynnika uczenia równego 0.1



Rysunek 3: Test dla współczynnika uczenia równego 0.2



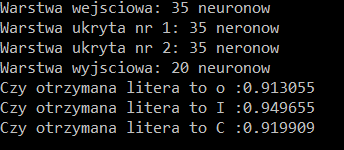
Rysunek 4:Test dla współczynnika uczenia równego 0.4

Wykres 1: Dokładność rozpoznania litery 'C' przez sieć w zależności od przyjętego współczynnika uczenia

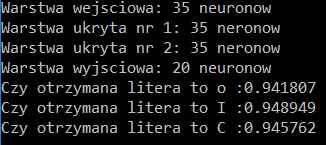
**Wariant II:**

* Zestaw uczący zawiera 20 liter: 10 wielkich i 10 małych;
* Zastosowano algorytm z funkcją aktywacji w postaci tangensa hiperbolicznego;
* Testy przeprowadzono dla 4 różnych współczynników uczenia: 0.01, 0.1, 0.2, 0.4
* W testach sprawdzano czy sieć poprawnie sklasyfikuje litery: **C**, **I**, **o**.

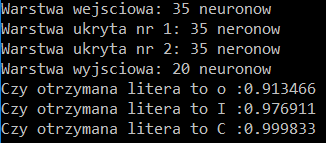
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Numer wariantu** | **Współczynnik uczenia η** | **Liczba danych uczących** | **Liczba potrzebnych iteracji do wyuczenia** | **Liczba błędów przy testowaniu** |
| **1** | 0.01 | 20 | 500 | 0 |
| **2** | 0.1 | 20 | 500 | 0 |
| **3** | 0.2 | 20 | 500 | 0 |
| **4** | 0.4 | 20 | 500 | 0 |



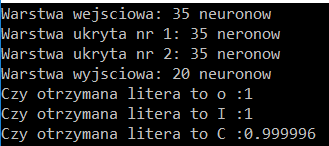
Rysunek 5: Test dla współczynnika uczenia równego 0.01



Rysunek 6: Test dla współczynnika uczenia równego 0.1



Rysunek 7: Test dla współczynnika uczenia równego 0.2



Rysunek 8: Test dla współczynnika uczenia równego 0.4

Wykres 2: Dokładność rozpoznania litery 'C' przez sieć w zależności od przyjętego współczynnika uczenia

1. **Wnioski**

* Wykonane ćwiczenie polegało na implementacji wielowarstwowej sieci neuronowej, której zadaniem było rozpoznawanie liter polskiego alfabetu. W procesie uczenia wykorzystano algorytm wstecznej propagacji błędu oparty na funkcji tangensa hiperbolicznego.
* Stworzona sieć składa się z 4 warstw: jednej wejściowej, jednej wyjściowej i dwóch warstw ukrytych. Liczba iteracji, w której sieć uczy się liter jest z góry zadana przez użytkownika.
* W trakcie testów stworzonej sieci wykorzystano dwa warianty, różniące się ilością iteracji w procesie uczenia. W pierwszym wariancie liczba iteracji wynosiła 1000 a w drugim wariancie 500. Ponadto w poszczególnych wariantach wprowadzono rozróżnienie ze względu na przyjęty współczynnik uczenia. W efekcie każdy wariant obejmuje po 4 przypadki testowe.
* Wariant numer jeden to proces uczenia trwający 1000 iteracji. Możemy zaobserwować, że niezależnie od przyjętego współczynnika uczenia sieć dobrze rozpoznaje daną literę gdyż w każdym z przypadków wartość wyjściowa uzyskana przez sieć jest bliska wartości 1 dla testowanej litery. Co należy podkreślić, zwiększając stopniowo współczynnik uczenia od wartości 0.01 do wartości 0.4, dokładność z jaką sieć rozpoznaje daną literę rośnie. Kiedy jednak przyjmiemy za współczynnik uczenia wartość 0.5 to dokładność już spada co oznacza, że optymalna wartość dla współczynnik uczenia oscyluje wokół 0.4. Zależność ta została przedstawiona na wykresie nr 1.
* W wariancie numer 2 przyjęto, że ilość iteracji w procesie uczenia wynosi 500. W tym przypadku zaobserwowano zależności analogiczne do wariantu numer 1, a mianowicie wzrost dokładności rozpoznawania liter przy rosnącym współczynniku uczenia aż do osiągnięcia ekstremum w punkcie 0.4 i następnie spadek dokładności. Opisana zależność przedstawiona jest na wykresie nr 2.
* Porównując między sobą warianty nr 1 i nr 2, różniące się między sobą ilością iteracji w procesie uczenia, możemy zaobserwować, że wzrost ilości iteracji powoduje wzrost dokładności rozpoznawania liter.

1. **Listing kodu programu**

#include<iostream>

#include<math.h>

#include<cstdlib>

#include<fstream>

using namespace std;

class SiecNeronowa {

public:

class Warstwa {

public:

int iloscWejsc; // neurony w warstwie poprzedniej

int iloscWyjsc; // neurony w warstwie obecnej

float \*wyjscia;

float \*wejscia;

float \*\*wagi;

float \*\*deltaWag;

float \*gamma;

float \*blad;

Warstwa(int inp, int out) {

this->iloscWejsc = inp;

this->iloscWyjsc = out;

wyjscia = new float[out];

wejscia = new float[inp];

wagi = new float \*[out];

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

wagi[i] = new float[inp];

deltaWag = new float \*[out];

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

deltaWag[i] = new float[inp];

gamma = new float[out];

blad = new float[out];

Inicjuj\_wagi();

}

Warstwa() {}

void Inicjuj\_wagi() {

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

{

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

{

wagi[i][j] = 0.5 + (double)rand() / RAND\_MAX\* (-1);

}

}

}

float\* Licz\_w\_przod(float \*in)

{

this->wejscia = in;

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

{

wyjscia[i] = 0;

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

{

wyjscia[i] += wejscia[j] \* wagi[i][j];

}

wyjscia[i] = tanh(wyjscia[i]);

}

return wyjscia;

}

float pochodnaTan(float value)

{

return (1 - (value\*value));

}

void propaguj\_wyjscie(float \*expected) {

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

blad[i] = wyjscia[i] - expected[i];

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

gamma[i] = blad[i] \* pochodnaTan(wyjscia[i]);

for (int i = 0; i < iloscWyjsc; i++) {

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

{

deltaWag[i][j] = gamma[i] \* wejscia[j];

}

}

}

void propaguj\_ukryte(float \*gammaForward, int size2, float\*\* weightsForward) {

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

{

gamma[i] = 0;

for (int j = 0; j < size2; j++)

{

gamma[i] += gammaForward[j] \* weightsForward[j][i];

}

gamma[i] \*= pochodnaTan(wyjscia[i]);

}

for (int i = 0; i < iloscWyjsc; i++) {

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

{

deltaWag[i][j] = gamma[i] \* wejscia[j];

}

}

}

void aktualizuj\_wagi() {

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

{

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

{

wagi[i][j] -= deltaWag[i][j] \* 0.4;

}

}

}

};

int \*warstwa;

int rozmiar;

Warstwa \*warstwy;

float\* LiczWPrzod(float \*inputs)

{

warstwy[0].Licz\_w\_przod(inputs);

for (int i = 1; i < rozmiar - 1; i++)

{

warstwy[i].Licz\_w\_przod(warstwy[i - 1].wyjscia);

}

return warstwy[rozmiar - 2].wyjscia;

}

void LiczWTyl(float \*expected) {

for (int i = rozmiar - 2; i >= 0; i--)

{

if (i == rozmiar - 2)

{

warstwy[i].propaguj\_wyjscie(expected);

}

else {

warstwy[i].propaguj\_ukryte(warstwy[i + 1].gamma, warstwy[i + 1].iloscWyjsc, warstwy[i + 1].wagi);

}

}

for (int i = 0; i < rozmiar - 1; i++)

{

warstwy[i].aktualizuj\_wagi();

}

}

SiecNeronowa(int \*war, int rozm) {

this->rozmiar = rozm;

this->warstwa = new int[rozm];

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

{

this->warstwa[i] = war[i];

}

this->warstwy = new Warstwa[rozm - 1];

cout << "Warstwa wejsciowa: 35 neuronow " << endl;

for (int i = 0; i < rozm-1; i++)

{

Warstwa \*A=new Warstwa(warstwa[i], warstwa[i + 1]);

warstwy[i] = \*A;

if (i == rozm - 2)

{

cout << "Warstwa wyjsciowa: 20 neuronow " << endl;

break;

}

cout << "Warstwa ukryta nr " << i+1 << ": " << A->iloscWyjsc <<" neronow" << endl;

}

}

void Testuj\_siec(float\* wejscia, char litera)

{

char literki\_tab[20] = { 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'a', 'b', 'c', 'd', 'n', 'f', 'h', 'i', 'k', 'o' };

int index = 0;

for (int i = 0;i<20;i++)

{

if (literki\_tab[i] == litera) index = i;

}

int n = 20;

float \*wynik = new float[n];

wynik = LiczWPrzod(wejscia);

cout << "Czy otrzymana litera to " << litera << " :" << wynik[index] << endl;

}

SiecNeronowa() {}

};

class ZestawDanych {

public:

int ilosc;

float \*Wzor;

float \*Wynik;

ZestawDanych() {

this->ilosc = 20;

int n = 35;

int m = 20;

Wzor = new float[n];

Wynik = new float[m];

}

};

int main()

{

int n = 4;

int\* tab = new int[n];

tab[0] = 35;

tab[1] = 35;

tab[2] = 35;

tab[3] = 20;

SiecNeronowa \*MLP=new SiecNeronowa(tab, 4);

ZestawDanych letters[20];

fstream plik;

plik.open("zbior\_uczacy1.txt");

if (plik.good())

{

for (int i = 0; i<20;i++)

{

for (int j = 0; j<35; j++)

{

plik >> letters[i].Wzor[j];

}

for (int k = 0; k<20; k++)

{

plik >> letters[i].Wynik[k];

}

}

}

else

{

cout << "blad otwarcia pliku!";

}

for (int i = 0; i < 500; i++)

{

for (int j = 0;j< 20;j++)

{

MLP->LiczWPrzod(letters[j].Wzor);

MLP->LiczWTyl(letters[j].Wynik);

}

}

float test[35]={ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0 }; // o

float test2[35] = {0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0 ,1 ,0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0 ,0 ,0 ,0, 0, 0, 0 }; // I

float test3[35] = { 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0 ,0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0 }; // C

MLP->Testuj\_siec(test,'o');

MLP->Testuj\_siec(test2, 'I');

MLP->Testuj\_siec(test3, 'C');

getchar();

}

1. **Literatura**

* <http://www.ai.c-labtech.net/sn/pod_prakt.html>
* <http://www-users.mat.umk.pl/~rudy/wsn/wyk/wsn-wyklad-05a-propag.pdf>
* <http://edu.pjwstk.edu.pl/wyklady/nai/scb/wyklad3/w3.htm>