Kamil Witek

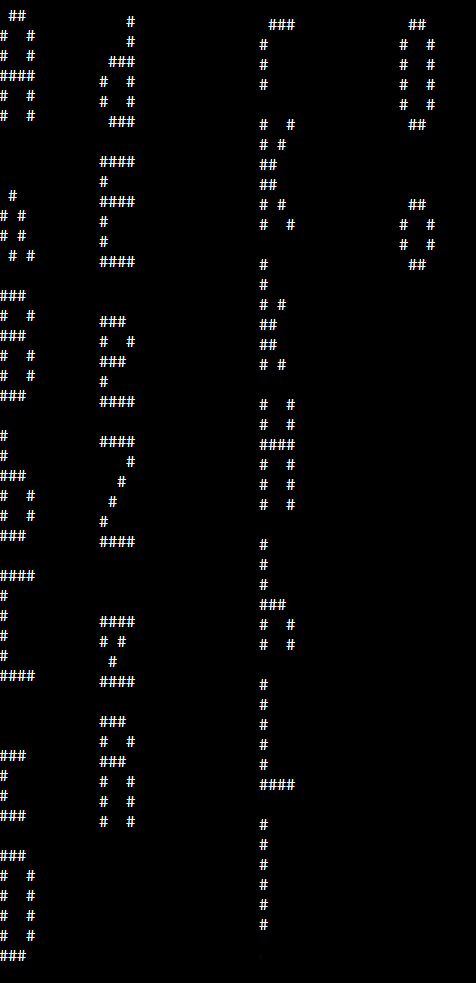
**Sprawozdanie - scenariusz​ ​2**

Temat​ ​ćwiczenia: Budowa i działanie sieci jednowarstwowej

**Cel​ ​ćwiczenia**

Celem ćwiczenia jest poznanie budowy i działania jednowarstwowych sieci neuronowych oraz uczenie rozpoznawania wielkości liter.

W ramach ćwiczeń wygenerowano 10 małych oraz 10 dużych liter alfabetu i podzielono je siatką w postaci dwuwymiarowej tablicy 4x6

****

Rysunek 1 Przedstawione graficznie dane uczące

Zaimplementowano uczenie się neuronu według algorytmu Widrowa – Hoffa. Zakłada on obliczanie średniego równego kwadratowi różnicy między wynikiem oczekiwanym w danych wejściowych a wynikiem otrzymanym w procedurze sumowania perceptronu. Błąd ten jest używany w warunku przerwania pętli uczącej.

Widrow\_Hoff():

1. wynik =
2. waga(i) += wspolczynnik\_nauki \* (wynik\_oczekiwany – wynik) \* wejście(i)
3. blad = (wynik\_oczekiwany – wynik)2
4. Kroki 1, 2 i 3 powtarza się, aż blad < prog

**Testowanie działania aplikacji**

Każdy test zakładał kilkukrotne uruchomienie programu z ustawieniem określonych parametrów.

1. Test I

Przy zestawie 20 danych uczących zmieniano współczynnik nauki zaczynając od 0,005 z krokiem 0,1. Wyniki przedstawiono na wykresie. Do testów użyto dwudziestu liter, z których dwie nie zawierały się w danych uczących. Wyniki testów przedstawiono w poniższej tabeli.

|  |  |  |  |
| --- | --- | --- | --- |
| **Lp.** | **Współczynnik nauki** | **Ilość epok** | **Skuteczność** |
| 1. | 0,0005 | 852 | 90% |
| 2. | 0,005 | 87 | 90% |
| 3. | 0,01 | 45 | 90% |
| 4. | 0,05 | 12 | 100% |
| 5. | 0,15 | 24 | 85% |
| 6. | 0,25 | 35 | 50% |
| 7. | 0,35 | 13 | 50% |
| 8. | 0,5 | 7 | 50% |

1. Test II

Ustawiono stały współczynnik nauki, zmieniano natomiast ilość danych uczących. Dane testujące to zbiór składający się z 20 liter. Współczynnik nauki ustawiono na poziomie 0,05, ponieważ w pozostałych testach dawał on największą dokładność.

Wyniki testów przedstawiono w poniższej tabeli.

|  |  |  |  |
| --- | --- | --- | --- |
| **Lp.** | **Ilość danych uczących** | **Ilość epok** | **Skuteczność** |
| 1. | 5 | 4 | 55% |
| 2. | 7 | 8 | 65% |
| 3. | 9 | 8 | 65% |
| 4. | 11 | 8 | 85% |
| 5. | 13 | 7 | 90% |
| 6. | 15 | 9 | 95% |
| 7. | 17 | 12 | 90% |
| 8. | 19 | 11 | 95% |
| 9. | 20 | 12 | 100% |

**Wnioski**

Pojedynczy perceptron jest w stanie odpowiadać na pytania, zwracając prawdę lub fałsz, co za tym idzie jest w stanie rozpoznać czy podana litera jest duża czy mała.

Z pierwszego testu wynika, że perceptron działa najskuteczniej dla współczynnika nauczania równego 0,05. Dla współczynników powyżej 0,25 nie da się zmierzyć dokładnej skuteczności. Wynika to z faktu, że perceptron może zwracać tylko dwa wyniki. Możemy więc zakładać, że wraz ze wzrostem wartości współczynnika nauczania powyżej wartości 0,25 skuteczność stopniowo spada, a otrzymane 50% skuteczności to tylko przypadek.

Wraz ze zmniejszaniem współczynnika nauczania, ilość epok nauki rośnie. Teoretycznie im więcej epok tym perceptron jest lepiej nauczony, lecz na wykresie możemy zauważyć, że zmniejszenie współczynnika nauczania poniżej 0,05 powoduje zmniejszenie skuteczności. Zjawisko to nazywa się przeuczeniem neuronu. Programista powinien dobrać tak współczynnik nauczania, aby wyeliminować to niekorzystne zjawisko.

Zwiększenie ilości danych uczących generalnie powoduje wydłużenie czasu uczenia. Ma to jednak wymierny wpływ na jakość nauczenia perceptronu, co wynika z wykresu nr 2 w teście drugim.

W analizowanym przypadku, najoptymalniejszym zestawem parametrów nauczania dla algorytmu Widrowa – Hoffa są:

* Współczynnik nauki = 0,05
* Ilość danych uczących = 20

**Listing kodu**

#pragma once

#include "Perceptron.h"

using namespace std;

void wczytaj\_wzornik(int tab[20][24]);

void wypisz(int tab[20][24]);

void testuj(int tab[20][24], Perceptron per);

void main()

{

setlocale(LC\_ALL, "");

Perceptron nowy;

int tab[20][24];

wczytaj\_wzornik(tab);

nowy.losuj\_wagi();

nowy.Widrow\_Hoff();

// wypisz(tab);

testuj(tab, nowy);

system("PAUSE");

}

/\* WZORNIK z danych uczacych

0,1,1,0,1,0,0,1,1,0,0,1,1,1,1,1,1,0,0,1,1,0,0,1 A

0,0,0,0,0,0,0,0,0,1,0,0,1,0,1,0,1,0,1,0,0,1,0,1 a

1,1,1,0,1,0,0,1,1,1,1,0,1,0,0,1,1,0,0,1,1,1,1,0 B

1,0,0,0,1,0,0,0,1,1,1,0,1,0,0,1,1,0,0,1,1,1,1,0 b

1,1,1,1,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,1,1,1 C

0,0,0,0,0,0,0,0,1,1,1,0,1,0,0,0,1,0,0,0,1,1,1,0 c

1,1,1,0,1,0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1,1,1,0 D

0,0,0,1,0,0,0,1,0,1,1,1,1,0,0,1,1,0,0,1,0,1,1,1 d

1,1,1,1,1,0,0,0,1,1,1,1,1,0,0,0,1,0,0,0,1,1,1,1 E

0,0,0,0,1,1,1,0,1,0,0,1,1,1,1,0,1,0,0,0,1,1,1,1 e

1,1,1,1,0,0,0,1,0,0,1,0,0,1,0,0,1,0,0,0,1,1,1,1 Z

0,0,0,0,0,0,0,0,1,1,1,1,1,0,1,0,0,1,0,0,1,1,1,1 z

1,1,1,0,1,0,0,1,1,1,1,0,1,0,0,1,1,0,0,1,1,0,0,1 R

0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,1,0,0,0,1,0,0,0 r

1,0,0,1,1,0,1,0,1,1,0,0,1,1,0,0,1,0,1,0,1,0,0,1 K

1,0,0,0,1,0,0,0,1,0,1,0,1,1,0,0,1,1,0,0,1,0,1,0 k

1,0,0,1,1,0,0,1,1,1,1,1,1,0,0,1,1,0,0,1,1,0,0,1 H

1,0,0,0,1,0,0,0,1,0,0,0,1,1,1,0,1,0,0,1,1,0,0,1 h

1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,1,1,1 L

1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0 l

0,1,1,0,1,0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,0,1,1,0 O

0,0,0,0,0,0,0,0,0,1,1,0,1,0,0,1,1,0,0,1,0,1,1,0 o

\*/

void wczytaj\_wzornik(int tab[20][24])

{

cout << "Wczytywanie wzornika do testów..." << endl;

ifstream odczyt; //utworzenie zmiennej ifstream w celu uzyskania dostepu do pliku

odczyt.open("wzornik.txt"); //otwiera plik w katalogu domyslnym o nazwie "test1.txt"

if (!odczyt.good()) //jesli nie mozna otworzyc pliku to wypisz blad

{

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

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

system("PAUSE");

exit(1); //jesli nie udalo sie otworzyc pliku to zakoncza dzialanie calego programu

}

while (!odczyt.eof()) // dopoki sa dane w pliku, wczytuj dane

{

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

{

for (int j = 0; j < 24; j++)

{

odczyt >> tab[i][j]; //wczytaj dane do tetów

}

}

cout << "Odczyt z pliku zakońcony powodzeniem!" << endl;

}

odczyt.close(); // zamknij plik

}

void wypisz(int tab[20][24])

{

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

{

for (int j = 0; j < 24; j++)

{

cout << tab[i][j] << ", ";

}

cout << endl;

}

}

void testuj(int tab[20][24], Perceptron per)

{

cout << "Testowanie danych..." << endl << endl;

int pom[24];

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

{

for (int j = 0; j < 24; j++)

{

pom[j] = tab[i][j];

}

per.sprawdz(pom);

}

}

#pragma once

#include <cstdio>

#include <cstdlib>

#include <cmath>

#include <ctime>

#include <iostream>

#include <fstream>

#include <iomanip>

#define WSPOLCZYNNIK\_NAUKI 0.1

#define MAX\_ITERACJE 20

using namespace std;

class Perceptron

{

int tab\_wej[20][24]; //tablica tab[n][m] gdzie n to ilosc danych uczacych a m to ilosc znakow w kazdej danej

float tab\_wag[24]; //tablica zawierajaca wagi dla kazdego wejscia do perceptronu

int tab\_wyn\_ocz[20]; //tablica zawiera oczekiwane wyniki (1 lub 0) dla danych uczacych

int ilosc\_wejsc; //ilosc wejsc w zaleznosci od podanj ilosci danych wejsciowych

bool wynik; //wynik po sprawdzeniu przez funkcje sprawdz(int tab[]) true albo false

float wspolczynnik\_nauki; //wspolczynnik nauki w funkcji ucz()

public:

Perceptron(); //konstruktor domyslny, zawiera ustawienie danych domyslnych oraz funkcje wczytania danych

void wczytanie\_danych(); //wczytanie danych z pliku

void losuj\_wagi(); // losowanie wag z przedziału od 0 do 1

float licz\_wyjscia(int i); //funkcja sumujšca perceptronu

void Widrow\_Hoff(); //funkcja uczaca na podstawie roznicy wyniku tymczasowego i wyniku oczekiwanego

void sprawdz(int tab[]); //funkcja sprawdzajaca po zakonczeniu procesu uczenia

void wypisz(); //wypisanie wyniku sprawdzania

~Perceptron(); //destruktor domyslny

};

#pragma once

#include "Perceptron.h"

Perceptron::Perceptron()

{

this->ilosc\_wejsc = 10; //ustawienie wynika z przygotowanych danych uczacych

this->wspolczynnik\_nauki = 0.05;

wczytanie\_danych(); //wczytanie danych z pliku tekstowego

}

void Perceptron::wczytanie\_danych()

{

ifstream plik; //utworzenie zmiennej ifstream w celu uzyskania dostepu do pliku

plik.open("test1.txt"); //otwiera plik w katalogu domyslnym o nazwie "test1.txt"

if (!plik.good()) //jesli nie mozna otworzyc pliku to wypisz blad

{

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

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

system("PAUSE");

exit(1); //jesli nie udalo sie otworzyc pliku to zakoncza dzialanie calego programu

}

while (!plik.eof()) // dopoki sa dane w pliku, wczytuj dane

{

for (int i = 0; i < ilosc\_wejsc; i++)

{

for (int j = 0; j < 24; j++)

{

plik >> this->tab\_wej[i][j]; //wczytaj dane do tablicy z wejsciami

}

plik >> this->tab\_wyn\_ocz[i]; //wczytaj wynik oczekiwany

}

}

plik.close(); // zamknij plik

}

void Perceptron::losuj\_wagi()

{

for (int i = 0; i < 24; i++)

{

this->tab\_wag[i] = (float)rand() / (float)RAND\_MAX; //dla kazdej wagi perceptronu losuje wartosc z zakresu od 0 do 1

}

}

float Perceptron::licz\_wyjscia(int i)

{

/\*funkcja sumujaca perceptronu

za argument przyjmuje indeks danej z tab\_wej[][], ktora jest teraz wykorzystywana do uczenia.

Nastepnie oblicza sume dla przyjetej danej(skladajacej sie z 24 elementow)

\*/

float suma = 0;

for (int j = 0; j < 24; j++)

{

suma += this->tab\_wej[i][j] \* this->tab\_wag[j];

}

return suma;

}

void Perceptron::Widrow\_Hoff()

{

float blad\_lokalny; //zmienna do obliczania bledu lokalnego

float blad\_globalny = 0; //zmienna do oblioczania bledu globalnego, musi byc zainicjowana!

int epoka = 0; //liczba epok nauczania

int przypadek; //indek elementu tab\_wej, ktory jest obecnie rozpatrywany

float wynik\_tmp; //przechowuje tymczasowy wynik, ktory jest obliczany w kazdej iteracji

float prog = 1.55; //zmienna, ktora okresla, kiedy algorytm jest juz wystaczajaco nauczony

do

{

blad\_globalny = 0;

for (przypadek = 0; przypadek < this->ilosc\_wejsc; przypadek++)

{

/\*OBLICZANIE WYNIKU W ITERACJI \*/

wynik\_tmp = licz\_wyjscia(przypadek);

blad\_lokalny = this->tab\_wyn\_ocz[przypadek] - wynik\_tmp; //obliczanie bledu lokalnego w tej iteracji petli for

/\*AKTALIZACJA WAG\*/

for (int i = 0; i < 24; i++)

{

this->tab\_wag[i] += this->wspolczynnik\_nauki \* blad\_lokalny \* this->tab\_wej[przypadek][i];

}

blad\_globalny += (blad\_lokalny\*blad\_lokalny); //obliczanie bledu globalnego, warunku wyjscia z petli do...while();

}

epoka++;

} while (blad\_globalny > prog); //powtarza dopoki blad globalny jest wiekszy od zadanego progu

cout << "Liczba wykonanych epok = " << epoka << endl;

}

void Perceptron::sprawdz(int tab[])

{

/\*Za argument przyjmuje tablice z jedna dana testujaca(zawierajaco 24 bity)\*/

float suma = 0, blad\_lokalny = 0, blad\_globalny = 0, prog\_aktywacji = 1.5;

for (int i = 0; i < 24; i++)

{

suma += tab[i] \* this->tab\_wag[i]; //oblicza sume dla zadanego elementu

}

blad\_lokalny = 1 - suma; //oblicza blad lokalny dla zadanego elementu

blad\_globalny = pow(2, blad\_lokalny); //blad globalny dla zadanego elementu

if (blad\_globalny < prog\_aktywacji) //jesli blad globalny jest mniejszy od naszego progu aktuwacji

this->wynik = true; //ustawia wynik na true

else

this->wynik = false; //ustawia wynik na false

wypisz(); //wypisz w konsoli wynik sprawdzenia

}

void Perceptron::wypisz()

{

if (this->wynik == true)

cout << "Podana jest mała litera!" << endl;

else

cout << "Podana jest duża litera!" << endl;

}

Perceptron::~Perceptron()

{

}