**Scenariusz 1**

**Temat ćwiczenia:** Budowa i działanie perceptronu

**Wykonała:** Burnat Magdalena, IS, gr. 3, 270652

1. **Cel ćwiczenia**

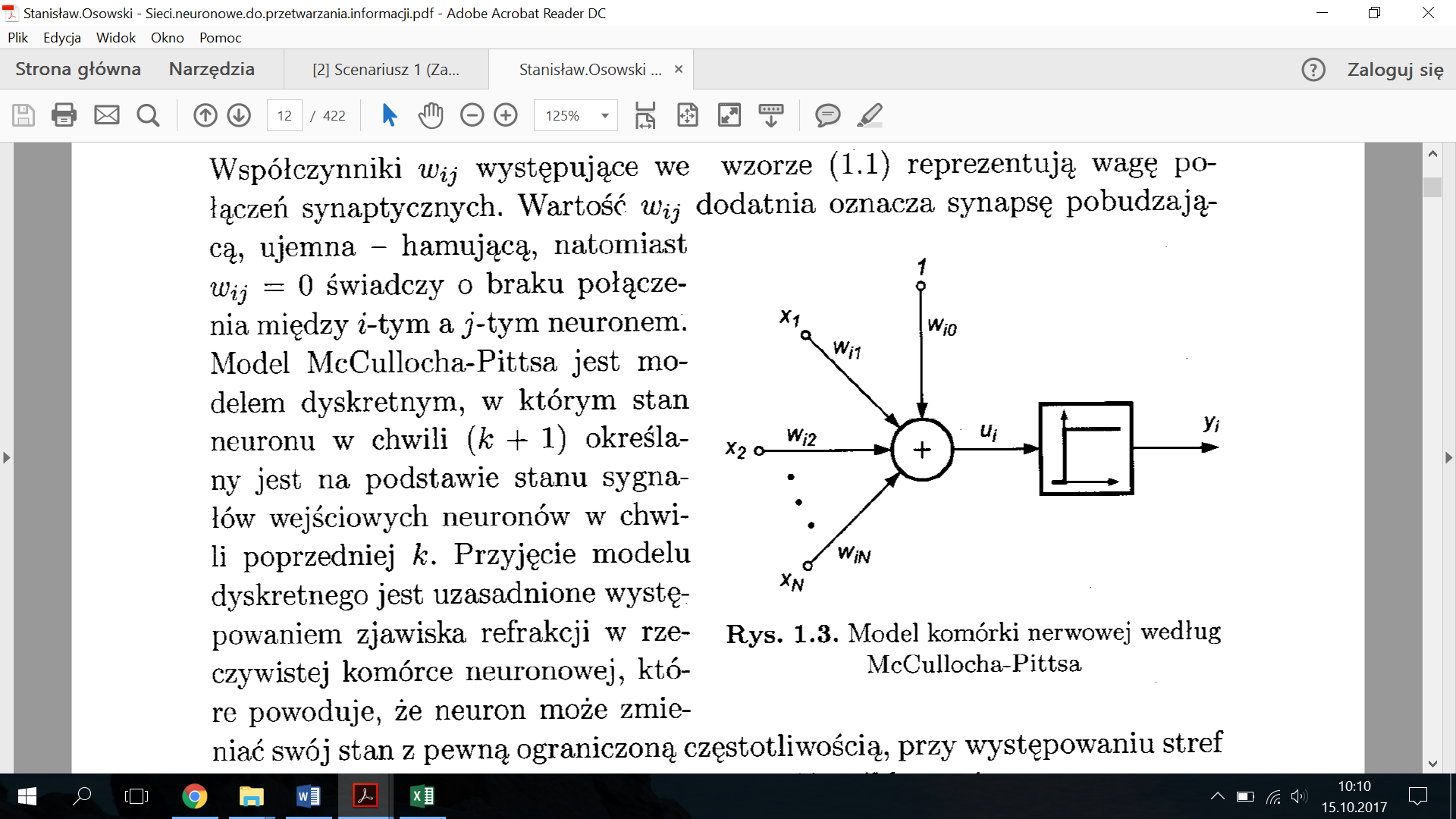
Celem ćwiczenia jest poznanie budowy i działanie perceptronu poprzez implementację oraz

uczenie perceptronu realizującego wybraną funkcję logiczną dwóch zmiennych.

1. **Opis budowy perceptronu i wykorzystanego algorytmu ucznia.**

Perceptron został zaimplementowany zgodnie z modelem McCullocha-Pittsa. Sygnały wejściowe xj (j = 1, 2, …, N) są sumowane z odpowiednimi wagami wij w sumatorze, a następnie jest dodawany jest sygnał polaryzacji x0 = 1 dla którego waga wynosiwio. Sygnał wyjściowy neuronu yi wyraża się, więc zależnością:

gdzie funkcje wyrażamy wzorem:



Rys. 1 Perceptron według modelu McCullocha-Pittsa[1]

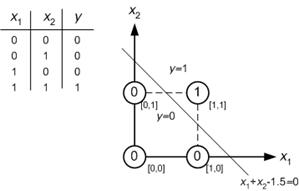
Model ten jest modelem dyskretnym, w którym stan neuronu w chwili (k+1) określony jest na podstawie stanu sygnałów wejściowych neuronów w chwili poprzedniej k.

Do uczenia perceptronu wykorzystano regułę Widrowa-Hoffa uzupełnioną o współczynnik uczenia . Na wstępie wagi dobrano w sposób losowy z zakresu 0-1. Dzięki nim obliczamy wartość sygnału wyjściowego yi. W wyniku porównania aktualnej wartości yi oraz wartości zadanej di dokonuje się aktualizacji wag. Jeżeli wartości otrzymana i zadana się nie różnią to wagi pozostają niezmienione. A więc zgodnie z tą regułą dobór wag odbywa się według zależności:

Podczas uczenia wykorzystuje się jedynie informacje o aktualnej wartości sygnału wyjściowego neuronu oraz wartości zadanej[1].

1. **Otrzymane wyniki**

Zaimplementowany perceptron składa się z jednej warstwy i jego zadaniem jest nauczyć się funkcji logicznej AND, a więc musi spełniać poniższe zależności.



Rys. 2 Funkcja logiczna AND

Początkowe wagi były ustawiane przez funkcję losującą, później w procesie uczenia były one odpowiednio modyfikowane, natomiast w trakcie testowania już nie następowały żadne zmiany. Przeprowadzono kilka testów uczenia różniących się od siebie liczbą danych uczących. Następnie w każdej z wersji perceptron był testowany za pomocą danych testujących w liczbie 50. Wyniki są przedstawione w poniższej tabeli.

Tab. 1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Wyniki procesu uczenia przy różnej liczbie danych uczących i stałym współczynniku uczenia równym: 0,01 | | | | | | |
|
| Uczenie: | | | | Testowanie: | | |
| Liczba danych uczących | Liczba dobrych odpowiedzi | Liczba złych odpowiedzi | Dobre odpowiedzi [%] | Liczba dobrych odpowiedzi | Liczba złych odpowiedzi | Dobre odpowiedzi [%] |
|
| 15 | 5 | 10 | 33,33 | 15 | 35 | 30 |
| 30 | 11 | 19 | 36,67 | 16 | 34 | 32 |
| 50 | 19 | 31 | 38,00 | 18 | 32 | 36 |
| 75 | 30 | 45 | 40,00 | 28 | 22 | 56 |
| 85 | 38 | 47 | 44,71 | 38 | 12 | 76 |
| 100 | 47 | 53 | 47,00 | 50 | 0 | 100 |

Drugim z przeprowadzonych testów była zmian współczynnika uczenia przy stałej liczbie danych uczących.

Tab. 2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Wyniki procesu uczenia przy stałej liczbie danych uczących wynoszących 100 i różnym współczynniku uczenia | | | | | | |
| Uczenie: | | | | Testowanie: | | |
| Współczynnik uczenia | Liczba dobrych odpowiedzi | Liczba złych odpowiedzi | Dobre odpowiedzi [%] | Liczba dobrych odpowiedzi | Liczba złych odpowiedzi | Dobre odpowiedzi [%] |
| 0,01 | 47 | 53 | 47 | 16 | 34 | 32 |
| 0,05 | 84 | 16 | 84 | 50 | 0 | 100 |
| 0,1 | 90 | 10 | 90 | 50 | 0 | 100 |
| 0,5 | 91 | 9 | 91 | 50 | 0 | 100 |

1. **Analiza wyników**

Z wykresu wynika, że wraz ze wzrostem liczby danych uczących procent dobrych odpowiedzi się zwiększa. Otrzymujemy także większy stosunek dobrych odpowiedzi podczas późniejszego testowania dla przypadków gdzie liczba danych uczenia była większa. I tak przy liczbie danych uczenia równych 100 mamy poprawnie działający perceptron, który nie popełnił już ani jednego błędu w późniejszym testowaniu.

Wykres nr.1

Wykres nr. 2

Na powyższym wykresie możemy obserwować wzrost procentowego udziału poprawnych odpowiedzi w zależności od współczynnika uczenia. Wraz z jego wzrostem skuteczność procesu uczenia rośnie i tak już dla wartości 0,05 otrzymujemy 100% poprawności w późniejszym procesie testowania.

1. **Wnioski**

* Skuteczność procesu uczenia zależy od liczby danych uczących. Wraz z jej wzrostem proces ten wykazuje większą skuteczność. Jeśli liczba ta będzie za mała perceptron nie będzie działał poprawnie. Dąży się do tego aby liczba niepoprawnie wykonany operacji była wartością najbliższą zeru
* Kolejnym aspektem wpływającym na skuteczność jest współczynnik uczenia. Wraz z jego wzrostem proces uczenia jest poprawniejszy. Należy tak dobrać jego wartość, aby przy zadanej liczbie danych uczących
* Jednowarstwowe sieci neuronowe można stosować do wykonywania prostych funkcji logicznych. Po odpowiednio przeprowadzonym procesie uczenia otrzymana bezbłędnie działający perceptron wykonujący funkcję logiczną AND dwóch zmiennych.

1. **Kod programu**

main.cpp

#include <iostream>

#include "Neuron.h"

#include <time.h>

#include <fstream>

using namespace std;

void ustawWartosciWejsc(Neuron& neuron, bool testowanie);

void ucz(Neuron& neuron, bool testowanie);

fstream PLIK\_WYNIK\_UCZENIA;

fstream PLIK\_WYNIK\_TESTOWANIA;

fstream PLIK\_DO\_NAUKI;

fstream PLIK\_DO\_TESTOWANIA;

int ILOSC\_ZLYCH\_ODPOWIEDZI;

int ILOSC\_DOBRYCH\_ODPOWIEDZI;

int PRAWIDLOWA\_ODPOWIEDZ;

int main()

{

srand(time(NULL));

Neuron neuron(3, 0.1);

PLIK\_WYNIK\_UCZENIA.open("wynik\_nauki.txt", ios::out);

PLIK\_WYNIK\_TESTOWANIA.open("wynik\_testowania.txt", ios::out);

PLIK\_DO\_NAUKI.open("do\_nauki.txt", ios::in);

PLIK\_DO\_TESTOWANIA.open("do\_testowania.txt", ios::in);

do

{

cout << "1. Ucz sie" << endl;

cout << "2. Testuj " << endl;

cout << "3. Wyjdz" << endl;

int wybor;

cin >> wybor;

switch (wybor)

{

case 1:

int liczbaDanych;

ILOSC\_ZLYCH\_ODPOWIEDZI = 0;

ILOSC\_DOBRYCH\_ODPOWIEDZI = 0;

cout << "Wpisz liczbe danych: " << endl;

cin >> liczbaDanych;

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

{

ucz(neuron, false);

}

PLIK\_WYNIK\_UCZENIA << "ILOSC ZLYCH ODPOWIEDZI: " << ILOSC\_ZLYCH\_ODPOWIEDZI << endl;

PLIK\_WYNIK\_UCZENIA << "ILOSC DOBRYCH ODPOWIEDZI: " << ILOSC\_DOBRYCH\_ODPOWIEDZI << endl;

break;

case 2:

ILOSC\_ZLYCH\_ODPOWIEDZI = 0;

ILOSC\_DOBRYCH\_ODPOWIEDZI = 0;

for (int i = 0; i < 50; i++)

{

ucz(neuron, true);

}

PLIK\_WYNIK\_TESTOWANIA << "ILOSC ZLYCH ODPOWIEDZI: " << ILOSC\_ZLYCH\_ODPOWIEDZI << endl;

PLIK\_WYNIK\_TESTOWANIA << "ILOSC DOBRYCH ODPOWIEDZI: " << ILOSC\_DOBRYCH\_ODPOWIEDZI << endl;

break;

case 3:

PLIK\_WYNIK\_UCZENIA.close();

PLIK\_WYNIK\_TESTOWANIA.close();

PLIK\_DO\_TESTOWANIA.close();

PLIK\_DO\_NAUKI.close();

return 0;

}

} while (true);

return 0;

}

void ustawWartosciWejsc(Neuron& neuron, bool testowanie)

{

if (testowanie == false)

{

int czyt1, czyt2, czyt3, odp;

PLIK\_DO\_NAUKI >> czyt1;

PLIK\_DO\_NAUKI >> czyt2;

PLIK\_DO\_NAUKI >> czyt3;

PLIK\_DO\_NAUKI >> odp;

neuron.wejscia[0] = czyt1;

neuron.wejscia[1] = czyt2;

neuron.wejscia[2] = czyt3;

PRAWIDLOWA\_ODPOWIEDZ = odp;

PLIK\_WYNIK\_UCZENIA << "Wej 0b: " << neuron.wejscia[0];

PLIK\_WYNIK\_UCZENIA << " Wej 1: " << neuron.wejscia[1];

PLIK\_WYNIK\_UCZENIA << " Wej 2: " << neuron.wejscia[2];

PLIK\_WYNIK\_UCZENIA << " POdp: " << PRAWIDLOWA\_ODPOWIEDZ;

}

else

{

int zczyt1, zczyt2, zczyt3, odpow;

PLIK\_DO\_TESTOWANIA >> zczyt1;

PLIK\_DO\_TESTOWANIA >> zczyt2;

PLIK\_DO\_TESTOWANIA >> zczyt3;

PLIK\_DO\_TESTOWANIA >> odpow;

neuron.wejscia[0] = zczyt1;

neuron.wejscia[1] = zczyt2;

neuron.wejscia[2] = zczyt3;

PRAWIDLOWA\_ODPOWIEDZ = odpow;

PLIK\_WYNIK\_TESTOWANIA << "Wej 0b: " << neuron.wejscia[0];

PLIK\_WYNIK\_TESTOWANIA << " Wej 1: " << neuron.wejscia[1];

PLIK\_WYNIK\_TESTOWANIA << " Wej 2: " << neuron.wejscia[2];

PLIK\_WYNIK\_TESTOWANIA << " POdp: " << PRAWIDLOWA\_ODPOWIEDZ;

}

}

void ucz(Neuron& neuron, bool testowanie)

{

ustawWartosciWejsc(neuron, testowanie);

neuron.sumujWszystkieWejscia();

int odpowiedzNeuronu = neuron.obliczWyjscieNeuronu();

int \*macierzWejsc = new int[2];

macierzWejsc[0] = neuron.wejscia[1];

macierzWejsc[1] = neuron.wejscia[2];

if (odpowiedzNeuronu)

{

if (PRAWIDLOWA\_ODPOWIEDZ == 0)

{

ILOSC\_ZLYCH\_ODPOWIEDZI++;

if (testowanie == false)

{

PLIK\_WYNIK\_UCZENIA << " Odp: 1";

neuron.obliczNowaWage(PRAWIDLOWA\_ODPOWIEDZ, odpowiedzNeuronu, macierzWejsc);

PLIK\_WYNIK\_UCZENIA << " -" << " ZMIANA WAG: " << neuron.wagi[0] << " " << neuron.wagi[1] << " " << neuron.wagi[2] << endl;

}

else

{

PLIK\_WYNIK\_TESTOWANIA << " Odp: 1 -" << endl;

}

}

else

{

ILOSC\_DOBRYCH\_ODPOWIEDZI++;

if (testowanie == false)

{

PLIK\_WYNIK\_UCZENIA << " Odp: 1";

PLIK\_WYNIK\_UCZENIA << " +" << " ZOSTAWIENIE WAG: " << neuron.wagi[0] << " " << neuron.wagi[1] << " " << neuron.wagi[2] << endl;

}

else

{

PLIK\_WYNIK\_TESTOWANIA << " Odp: 1 +" << endl;

}

}

}

else

{

if (PRAWIDLOWA\_ODPOWIEDZ == 1)

{

ILOSC\_ZLYCH\_ODPOWIEDZI++;

if (testowanie == false)

{

PLIK\_WYNIK\_UCZENIA << " Odp: 0";

neuron.obliczNowaWage(PRAWIDLOWA\_ODPOWIEDZ, odpowiedzNeuronu, macierzWejsc);

PLIK\_WYNIK\_UCZENIA << " -" << " ZMIANA WAG: " << neuron.wagi[0] << " " << neuron.wagi[1] << " " << neuron.wagi[2] << endl;

}

else

{

PLIK\_WYNIK\_TESTOWANIA << " Odp: 0 -" << endl;

}

}

else

{

ILOSC\_DOBRYCH\_ODPOWIEDZI++;

if (testowanie == false)

{

PLIK\_WYNIK\_UCZENIA << " Odp: 0";

PLIK\_WYNIK\_UCZENIA << " +" << " ZOSTAWIENIE WAG: " << neuron.wagi[0] << " " << neuron.wagi[1] << " " << neuron.wagi[2] << endl;

}

else

{

PLIK\_WYNIK\_TESTOWANIA << " Odp: 0 +" << endl;

}

}

}

}

Neuron.cpp

#include "Neuron.h"

#include <time.h>

void Neuron::sumujWszystkieWejscia()

{

double suma = 0;

for (int i = 0; i < 3; i++)

{

suma += obliczPojedynczeWejscie(i);

}

sumaWszystkichWejsc = suma;

}

int Neuron::obliczWyjscieNeuronu()

{

if (sumaWszystkichWejsc > 0)

{

wartoscWyjsciowa = 1;

return wartoscWyjsciowa;

}

else

{

wartoscWyjsciowa = 0;

return wartoscWyjsciowa;

}

}

void Neuron::obliczNowaWage(int prawidlowaOdpowiedz, int odpowiedzNeuronu, int\* macierzWejsc)

{

wagi[0] += wspolczynnikUczenia \* (prawidlowaOdpowiedz - odpowiedzNeuronu);

for (int i = 1; i < 3; i++)

{

wagi[i] += wspolczynnikUczenia \* (prawidlowaOdpowiedz - odpowiedzNeuronu) \* macierzWejsc[i - 1];

}

}

double Neuron::obliczPierwszaWage()

{

double waga = ((double)rand() / (RAND\_MAX));

return waga;

}

Neuron.h

#include <iostream>

using namespace std;

class Neuron

{

public:

double obliczPojedynczeWejscie(int indeks) { return wejscia[indeks] \* wagi[indeks]; }

void sumujWszystkieWejscia();

int obliczWyjscieNeuronu();

void obliczNowaWage(int prawidlowaOdpowiedz, int odpowiedzNeuronu, int \*macierzWejsc);

Neuron(int amount, double wsplUczenia)

{

wejscia[0] = 0;

wejscia[1] = 0;

wejscia[2] = 0;

wagi[0] = obliczPierwszaWage();

wagi[1] = obliczPierwszaWage();

wagi[2] = obliczPierwszaWage();

wspolczynnikUczenia = wsplUczenia;

}

double obliczPierwszaWage();

double wejscia[3];

double wagi[3];

double sumaWszystkichWejsc;

int wartoscWyjsciowa;

double wspolczynnikUczenia;

};

1. **Literatura**

[1] Osowski S., Sieci neuronowe do przetwarzania informacji, Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa 2006