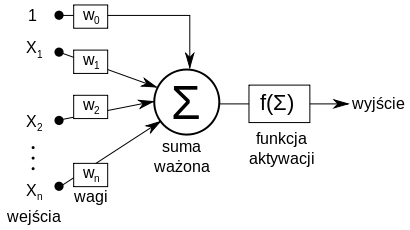
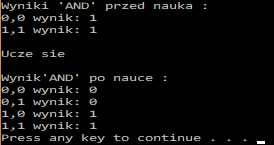
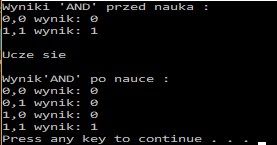
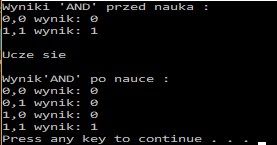
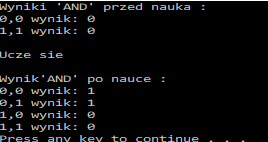
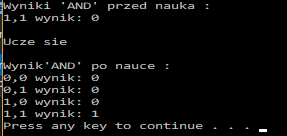
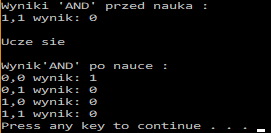
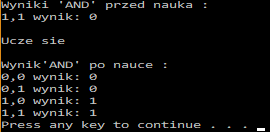
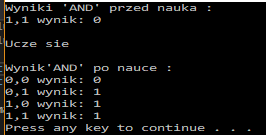
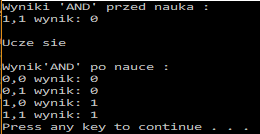
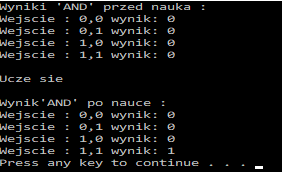
**Mateusz Bożek  
  
Scenariusz 1**  
  
**Temat :** Budowa i działanie perceptronu  
  
**Cel ćwiczenia :** Poznanie budowy i działania perceptronu poprzez implementację oraz uczenie perceptronu realizującego wybraną funkcję logiczną dwóch zmiennych.  
  
**1.** Do zrealizowania zadania wykorzystano perceptron zgodny z modelem McCulocha-Pittsa.W sumatorzeodbywa się sumowanie sygnałów wraz odpowiednimi wagami. Następnie dodawany jest sygnał polaryzacji xo = 1 dla którego waga wynosi wio. By wyliczyć sygnał wyjścia z neuronu należy skorzystać z zależności :  
  
 **2.** Perceptron użyty w podanym zadaniu ma za zadanie nauczyć się funkcji logicznej AND



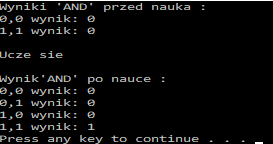
|  |  |  |
| --- | --- | --- |
| **X1** | **X2** | **y** |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 0 | 0 | 1 |

Na początku programu wagi zostały ustawione w wyniku użycia funkcji losującej, natomiast podczas procesu nauki przez program, wagi były odpowiednio modyfikowane. Nauczanie. Podczas testowania nie nastąpiły żadne zmiany. Po przeprowadzeniu 16 epok, program nauczył się prawidłowo. Przedstawia to poniższy zrzut ekranu :

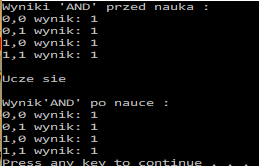
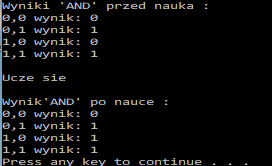
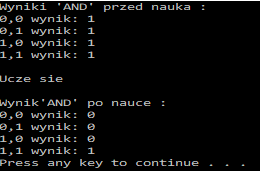
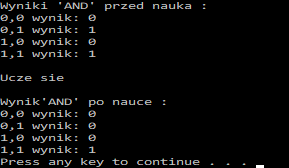
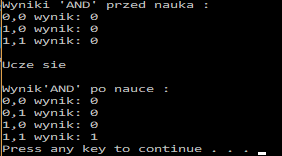
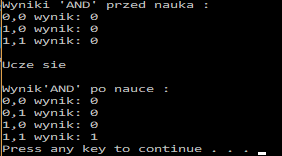
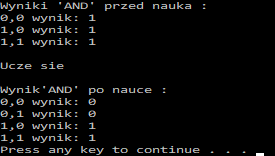
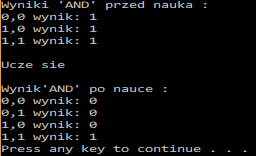
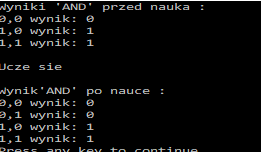
**3.**  
Podane dane : 1 dana ucząca, liczba epok = 5, współczynnik = 0.8  
  
  
Podane dane : 1 dana ucząca, liczba epok = 3, współczynnik = 0.9  
  
  
  
Podane dane : 1 dana ucząca, liczba epok = 4, współczynnik = 0.5  
  
Podane dane : 1 dana ucząca, liczba epok = 5, współczynnik = 0.1  
  
  
  
Podane dane : 1 dana ucząca, liczba epok = 6 , współczynnik = 0.2  
  
  
  
Podane dane : 2 dana ucząca, liczba epok = 5, współczynnik = 0.1  
  
  
  
Podane dane : 2 dana ucząca, liczba epok = 4, współczynnik = 0.3  
  
  
  
Podane dane : 2 dana ucząca, liczba epok = 6, współczynnik = 0.6  
  
  
Podane dane : 2 dana ucząca, liczba epok = 5, współczynnik = 0.1



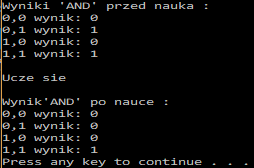
Podane dane : 3 dana ucząca, liczba epok = 3, współczynnik = 0.5



Podane dane : 3 dana ucząca, liczba epok = 3, współczynnik = 0.5  
  
  
Podane dane : 3 dana ucząca, liczba epok = 5, współczynnik = 0.3  
  
  
Podane dane : 3 dana ucząca, liczba epok = 6, współczynnik = 0.1  
  
  
Podane dane : 3 dana ucząca, liczba epok = 5, współczynnik = 0.1  
Podane dane : 3 dana ucząca, liczba epok = 4, współczynnik = 0.4  
Podane dane : 4 dana ucząca, liczba epok = 3, współczynnik = 0.9  
  
  
Podane dane : 4 dana ucząca, liczba epok = 5, współczynnik = 0.1  
  
  
Podane dane : 4 dana ucząca, liczba epok = 5, współczynnik = 0.1  
  
  
Podane dane : 4 dana ucząca, liczba epok = 5, współczynnik = 0.1



Podane dane : 4 dana ucząca, liczba epok = 6, współczynnik = 0.3



**Analiza:**Z wykresów oraz z powyższych wyników wynika fakt, że na szybkość uczenia się największy wpływ mają współczynnik uczenia oraz liczba danych. Skuteczność uczenia się jest większy dla dużej ilości danych oraz małego współczynnika danych. Kiedy podajemy mniej danych do programu, perceptron nie uczy się tak szybko. Najwolniej perceptron uczy się, gdy podana zostanie tylko jedna dana ucząca. Im więcej danych zostanie podanych do programu, tym większa jest szansa, że program nauczy się prawidłowo. Natomiast w przypadku współczynnika uczenia się, gdy zostają podane 4 dane, jego wpływ na szybkość nauki jest niewielki. Dla małych wartości współczynnika uczenia się, program szybciej uczy się prawidłowych wyników. **4.  
Wnioski:**  
 Szybkość procesu uczenia zależy od liczby danych uczących. Im jest ich więcej, tym perceptron szybciej się uczy. Lecz gdy jest ich mniej, uczy się wolniej i niedokładnie. Współczynnik uczenia się znajduje się w zakresie 0,1-1 i dzięki niemu nie zawsze potrzebna jest duża liczba danych by program szybciej się nauczył prawidłowych wyników.

**5.Kod programu :**  
source.c

#include <stdio.h>

#include "perceptron.h"

const int iteration = 16, numberOfImputs = 2;

const double learnRate = 0.2;

int main() {

const double ZERO\_ZERO[] = {0, 0};

const double ZERO\_ONE[] = {0, 1};

const double ONE\_ZERO[] = {1, 0};

const double ONE\_ONE[] = {1, 1};

int i;

Perceptron \*pOR = Perceptron\_new(numberOfImputs, learnRate);

printf("Wyniki 'AND' przed nauka :\n");

printf("Wejscie : 0,0 wynik: %d\n", uzyskajWynik(pOR, ZERO\_ZERO));

printf("Wejscie : 0,1 wynik: %d\n", uzyskajWynik(pOR, ZERO\_ONE));

printf("Wejscie : 1,0 wynik: %d\n", uzyskajWynik(pOR, ONE\_ZERO));

printf("Wejscie : 1,1 wynik: %d\n", uzyskajWynik(pOR, ONE\_ONE));

printf("\nUcze sie \n\n"); \\ rozpoczęcie procesu uczenia

for (i = 0 ; i < iteration; i++) {

ucz(pOR, ZERO\_ZERO, 0);

ucz(pOR, ZERO\_ONE, 0);

ucz(pOR, ONE\_ZERO, 0);

ucz(pOR, ONE\_ONE, 1);

}

printf("Wynik 'AND' po nauce :\n"); \\ wynik uczenia

printf("Wejscie : 0,0 wynik: %d\n", uzyskajWynik(pOR, ZERO\_ZERO));

printf("Wejscie : 0,1 wynik: %d\n", uzyskajWynik(pOR, ZERO\_ONE));

printf("Wejscie : 1,0 wynik: %d\n", uzyskajWynik(pOR, ONE\_ZERO));

printf("Wejscie : 1,1 wynik: %d\n", uzyskajWynik(pOR, ONE\_ONE));

system("pause");

return 0;

}  
  
  
perceptron.h   
  
#ifndef PERCEPTRON\_H

#define PERCEPTRON\_H

#ifdef \_\_cplusplus

extern "C" {

#endif

typedef struct Perceptron {

unsigned numInputs\_;

double \*weights\_;

double threshold\_, trainingRate\_;

} Perceptron;

Perceptron \*Perceptron\_new(unsigned numOfInputs, double trainingRate);

void ucz(Perceptron \*perceptron, const double inputs[], int expectedResult);

int uzyskajWynik(const Perceptron \*perceptron, const double inputs[])

double uzyskajWartosc(const Perceptron \*perceptron, const double inputs[]);

void ustawWage(Perceptron \*perceptron, const double \*weights);

const double \*uzyskajWage(const Perceptron \*perceptron);

double losowaWartosc();

void zmianaWagi(Perceptron \*perceptron, int actualResult, int desiredResult, const double inputs[]);

#ifdef \_\_cplusplus

}

#endif

#endif

perceptron.cpp  
  
#include <stdlib.h>

#include <time.h>

#include "perceptron.h"

#ifdef \_\_cplusplus

extern "C" {

#endif

// konstruktor

Perceptron \*Perceptron\_new(unsigned numOfInputs, double trainingRate) {

unsigned i;

static int initializedRandomization\_ = 0;

Perceptron \*perc = (Perceptron\*)malloc(sizeof(Perceptron));

perc->numInputs\_ = numOfInputs;

if (perc->numInputs\_ <= 0) {

perc->numInputs\_ = 1;

}

perc->trainingRate\_ = trainingRate;

perc->weights\_ = (double\*)malloc(perc->numInputs\_ \* sizeof(double));

if (! initializedRandomization\_) {

srand(time(NULL));

initializedRandomization\_ = 1;//

}

for (i = 0 ; i < perc->numInputs\_ ; i++) {

perc->weights\_[i] = losowaWartosc();

}

perc->threshold\_ = losowaWartosc();

return perc;

}

// funkcja odpowiedzialna za proces uczenia się

void ucz(Perceptron \*perceptron, const double inputs[], int expectedResult) {

int result = uzyskajWynik(perceptron, inputs);

if (result == expectedResult) {

return;

}

zmianaWagi(perceptron, result, expectedResult, inputs);

}

// funkcja odpowiedzialna za zmianę wagi wejść

void zmianaWagi(Perceptron \*perceptron, int actualResult, int desiredResult, const double inputs[]) {

unsigned i;

for (i = 0; i < perceptron->numInputs\_; i++) {

perceptron->weights\_[i] += perceptron->trainingRate\_ \* (desiredResult - actualResult) \* inputs[i];

}

perceptron->threshold\_ -= perceptron->trainingRate\_ \* (desiredResult - actualResult);

}

//funckaj do uzyskania wartość perceptronu

double uzyskajWartosc(const Perceptron \*perceptron, const double inputs[]) {

unsigned i;

double ans = 0;

for (i = 0 ; i < perceptron->numInputs\_ ; i++) {

ans += perceptron->weights\_[i] \* inputs[i];

}

return ans;

}

//funkcja do ustawiania wagi wejscia

void ustawWage(Perceptron \*perceptron, const double \*weights) {

unsigned i;

for (i = 0; i < perceptron->numInputs\_; i++) {

perceptron->weights\_[i] = weights[i];

}

}

// funkcja do uzyskania wyniku

int uzyskajWynik(const Perceptron \*perceptron, const double inputs[]) {

return (uzyskajWartosc(perceptron, inputs) >= perceptron->threshold\_);

}

// funkcja do ustawiania losowej wartości

double losowaWartosc() {

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

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

if (negativeRand < 0.5) {

randValue \*= -1.0;

}

return randValue;

}

//funkcja do uzyskania wagi

const double \*uzyskajWage(const Perceptron \*perceptron) {

return perceptron->weights\_;

}

#ifdef \_\_cplusplus

}

#endif