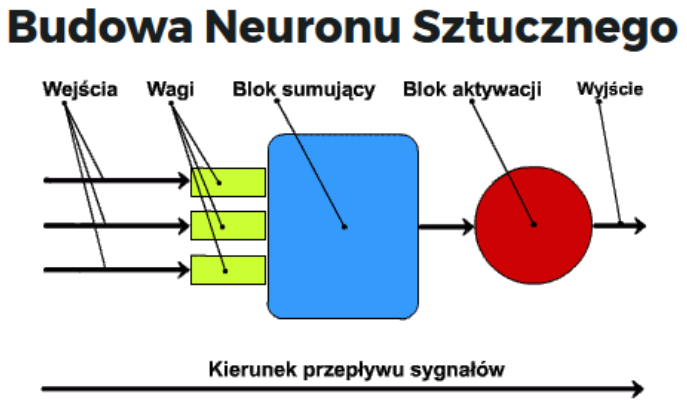
PODSTAWY SZTUCZNEJ INTELIGENCJI

Scenariusz 1
Maria Podkalicka, IS grupa 3

SZTUCZNY NEURON - to prosty system przetwarzający wartości sygnałów wprowadzanych na jego wejścia w pojedynczą wartość wyjściową, wysyłaną na jego jednym wyjściu (dokładny sposób funkcjonowania określony jest przez przyjęty model neuronu). Jest to podstawowy element sieci neuronowych, jego pierwowzorem był biologiczny neuron.



Neuron McCullocha-Pittsa– jeden z matematycznych modeli neuronu. Posiada wiele wejść i jedno wyjście. Każdemu z wejść przyporządkowana jest liczba rzeczywista - waga wejścia(liczba rzeczywista). Neuron ten jest podstawowym budulcem sieci neuronowej perceptron. Wartość na wyjściu neuronu obliczana jest w następujący sposób:

1. obliczana jest suma iloczynów wartości x_i podanych na wejścia i wag w_i wejść:

$$s = w_0 + \sum_{i=1}^{n} x_i w_i$$

2. na wyjście podawana jest wartość funkcji aktywacji f(s) dla obliczonej sumy

FUNKCJA AKTYWACJI - funkcja, według której obliczana jest wartość wyjścia neuronów sieci neuronowej.

Wykorzystałam neuron **McCullocha-Pittsa** z dwoma wejściami. Jako funkcję aktywacji wykorzystałam funkcję progową unipolarną w postaci:

$$y = \begin{cases} 0 \; dla \; s < 0 \\ 1 \; dla \; s \geq 0 \end{cases}$$

y-wyjście neuronu

Skorzystałam z następującego algorytmu uczenia:

- Początkowe wagi zostały wylosowane z zakresu <-0.5, 0.5>
- Sprawdziłam czy na podstawie tak przygotowanych danych wejściowych otrzymam oczekiwany wynik.

  Jeżeli nie uzyskałam oczekiwanego wyniku to wykonałam następujące czynności:

  - Obliczyłam błąd: e=uzyskany_wynik - oczekiwany_wynik
  - Modyfikujacja wagi: Waga=Waga+współczynnik_uczenia*e*dana_wejściowa, oraz b=b+współczynnik_uczenia *e

- Procedurę powtórzyłam dla wszystkich przygotowanych zestawów danych (kolejność użycia zestawów jest losowa) a następnie sprawdziłam błąd średniokwadratowy:

$$E=\frac{1}{2}\sum_{i=1}^{p}(d_i - y_i)^2$$

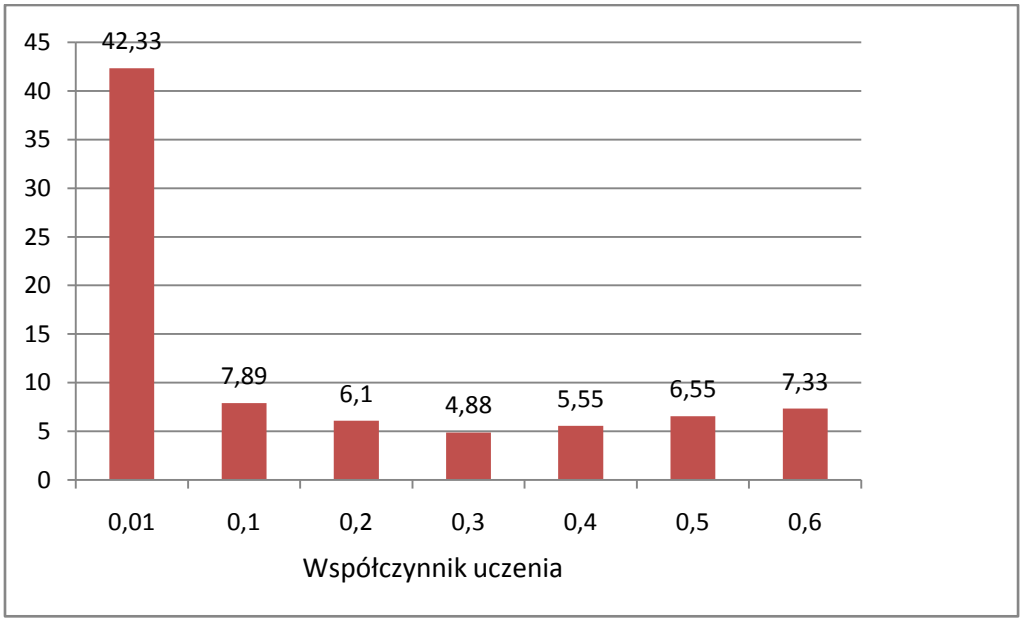
  p-liczba przykładów do nauki
  d_i-oczekiwana odpowiedź perceptronu
  y_i-uzyskana odpowiedź
- Jeżeli e>0 to powtarzam proces uczenia

W moim programie neuron miał wykonać operacje logiczną OR:
Operacja logiczna OR przyjmuje 2 parametry binarne i odpowiada wartością 0 lub 1.
Tabela przedstawia odpowiedzi funkcji XOR dla wszystkich możliwych danych wejściowych:

a	b	x
0	0	0
0	1	1
1	0	1
1	1	1

	liczba cykli uczenia									
ni	test 1	2	3	4	5	6	7	8	9	średnia
0,01	50	48	51	41	38	39	45	10	59	42,33
0,1	10	6	5	11	8	9	10	7	6	7,89
0,2	5	6	10	7	5	5	6	7	4	6,10
0,3	5	3	6	6	7	4	6	3	4	4,88
0,4	6	9	3	4	3	8	5	7	5	5,55
0,5	7	6	4	8	5	4	8	9	10	6,55
0,6	8	9	5	10	6	8	5	8	7	7,33

Tabela przedstawia zależność pomiędzy współczynnikiem uczenia a ilością potrzebnych cykli uczenia.



Wykres przedstawia zależność pomiędzy współczynnikiem uczenia a średnią ilością potrzebnych cykli uczenia.

Wnioski:
Dzięki wynikom zebranym w tabeli można zauważyć, że jeśli współczynnik uczenia jest większy to przy takiej samej wartości błędu jest wykonywana większa poprawka wag, więc dla małych wartości η potrzebne jest więcej iteracji. Jednak może się też zdarzyć, że poprawka wag może być za duża dla dużych wartości η i nie przybliży nas do rozwiązania. W moich testach najlepsza wartość współczynnika uczenia wyniosła 0,3.

Kod programu:

main.cpp:

```
#include <iostream>
#include <fstream>
#include "neuron.h"

using namespace std;

double funAktywacji(double x);
double* read_csv(string plik);

int main()
{
	double *wejscie1 = read_csv("../file1.csv");
	double *wejscie2 = read_csv("../file2.csv");
	neuron *neurone1 = new neuron(64);

	neurone1->setfunkcjaAktywacji(funAktywacji);

	cout << neurone1->start(wejscie2) << endl;

	neurone1->nauka(wejscie2, 0);
	cout << neurone1->start(wejscie2) << endl;
```

```
	cout << neurone1->start(wejscie1) << endl;

	neurone1->nauka(wejscie1, 1);
	cout << neurone1->start(wejscie1) << endl;
	cout << neurone1->start(wejscie2) << endl;

	system("pause");
	return 0;
}

double funAktywacji(double x)
{
	if (x > 0)
		return 1;
	else
		return 0;
}


double* read_csv(string file)
{

	ifstream is(file);

	double *in = new double[64];
	int i = 0;
	char  ch;

	while (is.get(ch))
	{
		if (ch != ',' && ch != '\n')
		{
			in[i] = ch - 48;
			i++;
		}
	}
	is.close();

	return in;
}
```

neuron.h:

```
#pragma once
typedef double(*fun1)(double x);

class neuron
{
public:
	neuron();
	neuron(int n);
	neuron(int n, double *wagi);
	~neuron();

	void set_wagi(double *m_wagi);
	double get_wagi(int i);
	void set_wagi(double x, int i);

	void set_n(int Mn);
```

```
        int get_n() { return n; }

        void setfunkcjaAktywacji(fun1 funAktywacji);
        void set_pochodna(fun1 Mpochodna);

        double start(double *input);
        void nauka(double *input, double output);
        fun1 getPochodna() { return pochodna; };


private:
        double *wagi;
        int n;
        fun1 funAktywacji;
        fun1 pochodna;

        double sum(double *m_input);
};
```

neuron.cpp:

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "neuron.h"


double f_rand(double f_min, double f_max)
{
        double f = (double)rand() / RAND_MAX;
        return f_min + f * (f_max - f_min);
}
neuron::neuron()
{
        wagi = NULL;
        funAktywacji = NULL;
        srand(time(NULL));
}
neuron::neuron(int Mn) :
        n(Mn)
{
        funAktywacji = NULL;
        srand(time(NULL));
        wagi = new double[n];
        for (int i = 0; i < n; i++)
        {
                wagi[i] = f_rand(0, 1);
        }
}
neuron::neuron(int Mn, double *m_wagi) :
        n(Mn),
        wagi(m_wagi)
{
        funAktywacji = NULL;
        srand(time(NULL));
}
void neuron::set_wagi(double *m_wagi)
{
```

```
	wagi = m_wagi;
}
double neuron::get_wagi(int i)
{
	if (wagi != NULL && i < n)
	{
		return wagi[i];
	}
	else
	{
		throw "error-getWagi";
	}
}
void neuron::set_wagi(double x, int i)
{
	if (wagi != NULL && i < n)
	{
		wagi[i] = x;
	}
	else
	{
		throw "error-set_wagi";
	}
}
void neuron::set_n(int Mn)
{
	n = Mn;

}
void neuron::setfunkcjaAktywacji(fun1 m_funkcjaAktywująca)
{
	funAktywacji = m_funkcjaAktywująca;
}
void neuron::set_pochodna(fun1 Mpochodna)
{
	pochodna = Mpochodna;
}
double neuron::start(double *inputs)
{
	if (funAktywacji == 0)
		throw "ustaw funkcje aktyacji!";
	double a = sum(inputs);
	return funAktywacji(a);

}
double neuron::sum(double *m_input)
{

	double suma = 0;
	for (int i = 0; i < n; i++)
	{
		suma += wagi[i] * m_input[i];
	}

	return suma;
}
void neuron::nauka(double * input, double output)
```

```
{
	if (funAktywacji == 0)
		throw "ustaw funkcje aktywacji!";

	double ni = 0.5;
	double result = this->start(input);
	if (result != output)
	{
		for (int i = 0; i < n; i++)
		{
			wagi[i] += ni*(output - result)*input[i];
		}
	}


}
neuron::~neuron()
{
	delete wagi;
}
```