Tomasz Tomala

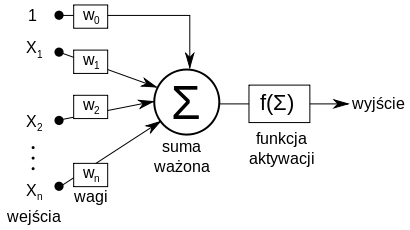
Podstawy Sztucznej Inteligencji

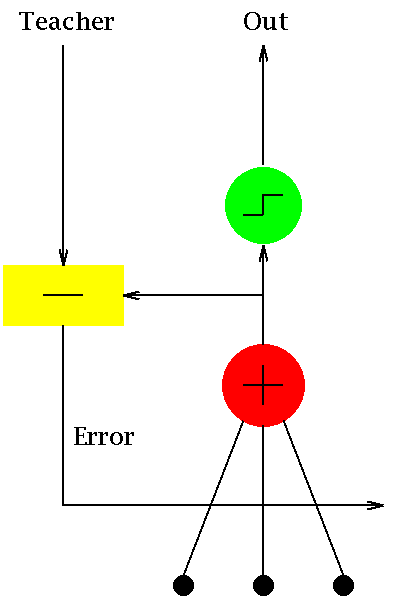
Sprawozdanie z projektu nr 2

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

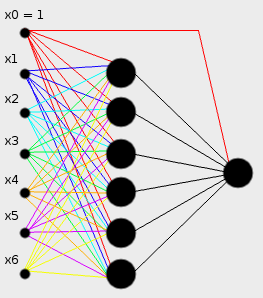
**1) Syntetyczny opis budowy wykorzystanego algorytmu uczenia:**

Aby wykonać ćwiczenie stworzyłem dwie jednowarstwowe sieci – każdą z wykorzystaniem innego algorytmu. Pierwsza z nich korzysta z modelu perceptronu McCullocha-Pittsa, druga z kolei wykorzystuje neuron typu Adaline.

  
Ilustracja 1: Model perceptronu McCullocha-Pittsa

  
Ilustracja 2: Model neuronu Adaline

Obie sieci składają się z siedmiu neuronów. Sześć pierwszych stanowi pierwszą warstwę i przesyła sygnały wyjściowe do siódmego wyjściowego neuronu. Każdy z neuronów otrzymuje po siedem sygnałów wejściowych. Sposób w jaki wszystko zostało połączone przedstawiam na poniższej grafice:

  
Ilustracja 3: Graficzne zilustrowanie połaczeń w sieci

Do nauki perceptronów oraz adaline wykorzystałem algorytm Widrow-Hoffa.

Do budowy perceptronu wykorzystałem podany na wykładzie model McCullocha-Pittsa. Zaimplementowana przeze mnie klasa Perceptron składa się z trzech metod: active, process oraz learn.

Metoda ***active*** wykorzystuje unipolarną funkcję progową która zwraca wynik 0 lub 1 dla podanego argumentu.



Metoda ***process*** sumuje iloczyn sygnałów wejściowych i odpowiadających im wag. Uruchamia metodę activate z otrzymaną sumą iloczynów jako parametrem, oraz zwraca wynik tej metody.



Metoda ***learn*** wywołuje metodę process dla otrzymanych wejść jako parametrów, po czym na podstawie otrzymanego wyniku modyfikuje wszystkie wagi dla odpowiednich wejść.

**Wi = Wi + ( y – y’ ) \* xi \* α**

Budowa adaline jest bardzo podobna do budowy perceptronu. Różni się tylko tym, że modyfikowanie wag wykonuje się z pominięciem funkcji aktywacji. Zaimplementowana przeze mnie klasa Adaline składa się z czterech metod: active, process, learn oraz test.

Metoda ***active*** wykorzystuje unipolarną funkcję progową która zwraca wynik -1 lub 1 dla podanego argumentu.



Metoda ***process*** sumuje iloczyn sygnałów wejściowych i odpowiadających im wag, oraz zwraca ten wynik.



Metoda ***learn*** wywołuje metodę process dla otrzymanych wejść jako parametrów, po czym na podstawie otrzymanego wyniku modyfikuje wszystkie wagi dla odpowiednich wejść.

**Wi = Wi + ( y – y’ ) \* xi \* α**

Metoda ***test*** uruchamia metodę active z wynikiem metody process jako parametrem.

**2) Zestawienie otrzymanych wyników:**

Jako dane uczące i testujące wykorzystałem własnoręcznie stworzone litery alfabetu. Są one przedstawione jako dwuwymiarowa tablica o rozmiarach 7 x 5. Przedstawiam je poniżej:











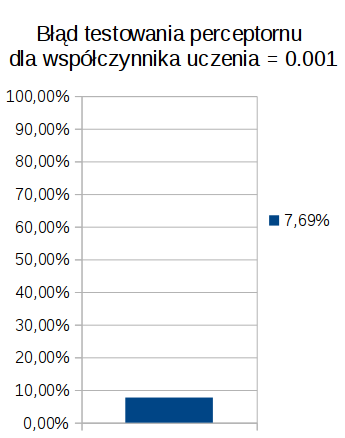


Tablicę 7 x 5 podzieliłem na 6 pod obszarów. Każdy z nich to jeden sygnał wejściowy do neuronów. Jeżeli w danym obszarze pojawił się choćby jeden piksel z litery to obszar zwracał sygnał 1, w przeciwnym wypadku zwracał sygnał 0. Poniżej sposób podziału tablicy na obszary, oraz przykład wyniku dla dużej i małej litery „M”:





Przygotowałem 26 dużych i małych liter alfabetu. Pierwszą połowę z nich użyłem jako dane uczące a drugą połowę jako dane testujące. Proces uczenia przeprowadziłem kilkukrotnie dla różnych współczynników uczenia. Wagi początkowe dobierane są w sposób losowy, co wpływa na wyniki uczenia się, dlatego przy każdym współczynniku uczenia wykonałem kilka testów i wybrałem najlepsze rezultaty. Poniżej przedstawiam wyniki:



**3) Analiza i dyskusja błędów uczenia i testowania opracowanych sieci w zależności od wartości współczynnika uczenia oraz wybranego algorytmu:**

Zarówno w przypadku neuronu typu perceptron oraz adaline widać, że im mniejszy współczynnik uczenia, tym więcej epok zajęło nauczenie się rozpoznawania wielkości liter. Jeśli jednak porównać oba te algorytmy, to widać sporą różnicę. Przy współczynniku uczenia:

* 0.1 - perceptron 6 epok, adaline 1 epoka
* 0.01 – perceptron 19 epok, adaline 3 epoki
* 0.001 – perceptron 89 epok, adaline 25 epok

W każdym przypadku uczenia wykonałem również testowanie na pozostałej połowie liter alfabetu. W prawie wszystkich przypadkach błąd przy testowaniu wynosił 0%. Jedynie w przypadku współczynnika uczenia 0.001 błąd przy testowaniu perceptronu wynosił 7,69%. Mogły to spowodować znacznie różniące się od innych litery takie jak np. litera „s”. Dane te zostały wybrane jako najlepsze ze wszystkich jakie uzyskałem w trakcie testów.

Dodatkowo jeżeli spojrzeć na wykresy, można zauważyć, głównie przy tych niższych współczynnikach uczenia, jak % błędów czasami skacze z niższych na wyższe wartości. Prawdopodobnie spowodowane było to podobieństwem między niektórymi wielkimi i małymi literami takimi jak np.: B b, F f, L l.

**4) Sformułowanie wniosków:**

Z powyższych danych jasno wynika, iż neuron typu adaline jest o wiele wydajniejszy niż perceptron. Niezależnie od współczynnika uczenia zawsze uczył się kilkukrotnie szybciej, dodatkowo podczas testowania jego błąd zawsze wynosił 0%. Wynika to z tego w jaki sposób uczy się adaline. Otóż modyfikowanie wag następuje przed funkcją aktywacji. Oznacza to, że jeżeli wartość oczekiwana będzie się znacznie różnić od wartości wyliczonej, to wagi również zostaną znacznie zmodyfikowane. Z kolei jeżeli popełniony błąd będzie niewielki, to wagi również zostaną niewiele zmienione. Co więcej, adaline różni się od perceptronu jeszcze w jednej kwestii. W przypadku perceptronu, gdzie sygnały wejściowe, tak jak i sygnały wyjściowe to 0 lub 1, modyfikacja wag będzie się odbywać tylko i wyłącznie poprzez zwiększanie ich wartości. Jeśli chodzi o adaline, to sygnały wejściowe jak i wyjściowe to -1 lub 1. Dzięki temu modyfikacja wag może odbywać się nie tylko poprzez zwiększanie ich wartości, ale także poprzez ich zmniejszanie, co znacznie poprawia dokładność.

**5) Listing całego kodu**

**package perceptron;**

import java.util.Random;

public class Perceptron {

private int noi; //ilość wejść

private double[] w; //wagi

public Perceptron ( int numbers\_of\_inputs ) {

noi = numbers\_of\_inputs;

w = new double[noi];

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

w[i] = new Random().nextDouble(); //wagi początkowe są losowane

}

//funkcja aktywująca

private int active ( double y\_p ) {

return y\_p < 0 ? 0 : 1;

}

//sumator

public int process ( int[] x ) {

double y\_p = 0;

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

y\_p += x[i] \* w[i];

return active( y\_p );

}

//uczenie

public void learn ( int[] x, double y, double lr ) {

double y\_p = process( x );

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

w[i] += ( y - y\_p ) \* lr \* x[i]; //modyfikacja wag

}

}

**package perceptron;**

import alphabet.Alphabet;

import java.util.Arrays;

public class Main\_perceptron {

public static void main ( String[] args ) {

int noi = 7; //ilość wejść

int nol = 13; //ilość liter MAX 26

int counter = 0; //licznik ilości epok uczenia się

double lr = 0.1; //krok uczenia się

Perceptron[] perc = new Perceptron[noi];

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

perc[i] = new Perceptron( noi );

int[] y = new int[nol \* 2]; //0 - duża litera, 1 - mała litera

Arrays.*fill*( y, 0, nol, 0 );

Arrays.*fill*( y, nol, nol \* 2, 1 );

int[] wyj = new int[nol \* 2]; //tablica przechowująca wyniki testowania perceptronu

Arrays.*fill*( wyj, 0, nol \* 2, 0 );

while ( ! Arrays.*equals*( y, wyj ) ) {

for ( int i = 0; i < 2; i++ ) //0 - wielkie litery, 1 - małe litery

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

*learn*( perc, noi, lr, i, j );

wyj = *test*( perc, nol, noi );

counter++;

}

System.*out*.println( "Ilość kroków do nauczenia się = " + counter );

}

public static void learn ( Perceptron[] perc, int noi, double lr, int i, int j ) {

int[] vector; //tablica przechowująca wektor sygnałów wejściowych do uczenia pierwszej warstwy sieci

vector = Alphabet.*getLetter*( i, j );

int[] vector\_p = new int[noi]; //tablica przechowująca wektor sygnałów wyjściowych pierwszej warstwy sieci

vector\_p[0] = 1; //bias

for ( int k = 0; k < noi - 1; k++ ) { //uczenie pierwszej warstwy

perc[k].learn( vector, i, lr );

vector\_p[k + 1] = perc[k].process( vector ); //pobranie sygnału wyjściowego

}

perc[noi - 1].learn( vector\_p, i, lr ); //uczenie perceptronu wynikowego

//na podstawie sygnałów wyjściowych pierwszej warstwy

}

public static int[] test ( Perceptron[] perc, int nol, int noi ) {

int[] wyj = new int[nol \* 2];

int[] vector; //tablica przechowująca wektor sygnałów wejściowych do testowania pierwszej warstwy sieci

int[] vector\_p = new int[noi]; //tablica przechowująca wektor sygnałów wyjściowych pierwszej warstwy sieci

vector\_p[0] = 1; //bias

for ( int i = 0; i < 2; i++ ) { //testowanie, celem upewnienia się, czy sieć już nauczona

for ( int j = 0; j < nol; j++ ) {

vector = Alphabet.*getLetter*( i, j );

for ( int k = 0; k < noi - 1; k++ )

vector\_p[k + 1] = perc[k].process( vector );

wyj[i \* nol + j] = perc[noi - 1].process( vector\_p );

}

}

return wyj;

}

}

**package adaline;**

import java.util.Random;

public class Adaline {

private int noi; //ilość wejść

private double[] w; //wagi

public Adaline ( int numbers\_of\_inputs ) {

noi = numbers\_of\_inputs;

w = new double[noi];

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

w[i] = new Random().nextDouble(); //wagi początkowe są losowane

}

//funkcja aktywująca

private int active ( double y\_p ) {

return y\_p <= 0 ? (-1) : 1;

}

//sumator

public double process ( int[] x ) {

double y\_p = 0;

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

y\_p += x[i] \* w[i];

return y\_p;

}

//uczenie

public void learn ( int[] x, double y, double lr ) {

double y\_p = process( x );

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

w[i] += ( y - y\_p ) \* lr \* x[i]; //modyfikacja wag

}

//testowanie

public int test ( int[] x )

{

return ( active( process( x ) ) );

}

}

**package adaline;**

import alphabet.Alphabet;

import java.util.Arrays;

public class Main\_adaline {

public static void main ( String[] args ) {

int noi = 7; //ilość wejść

int nol = 13; //ilość liter MAX 26

int counter = 0; //licznik ilości epok uczenia się

double lr = 0.1; //krok uczenia się

Adaline[] ada = new Adaline[noi];

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

ada[i] = new Adaline( noi );

int[] y = new int[nol \* 2]; //-1 - duża litera, 1 - mała litera

Arrays.*fill*( y, 0, nol, - 1 );

Arrays.*fill*( y, nol, nol \* 2, 1 );

int[] wyj = new int[nol \* 2]; //tablica przechowująca wyniki testowania adaline

Arrays.*fill*( wyj, 0, nol \* 2, 0 );

while ( ! Arrays.*equals*( y, wyj ) ) {

for ( int i = 0; i < 2; i++ ) { //-1 - wielkie litery, 1 - małe litery

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

*learn*( ada, noi, lr, i, j );

}

wyj = *test*( ada, nol, noi );

counter++;

}

System.*out*.println( "Ilość kroków do nauczenia się = " + counter );

}

private static void learn ( Adaline[] ada, int noi, double lr, int i, int j ) {

int[] vector; //tablica przechowująca wektor sygnałów wejściowych do uczenia pierwszej warstwy sieci

vector = Alphabet.*getLetter*( i, j );

*format*( vector );

int[] vector\_p = new int[noi]; //tablica przechowująca wektor sygnałów wyjściowych pierwszej warstwy sieci

vector\_p[0] = 1; //bias

int letter\_size;

if ( i == 0 ) letter\_size = - 1;

else letter\_size = 1;

for ( int k = 0; k < noi - 1; k++ ) { //uczenie pierwszej warstwy

ada[k].learn( vector, letter\_size, lr );

vector\_p[k + 1] = ada[k].test( vector ); //pobranie sygnału wyjściowego

}

ada[noi - 1].learn( vector\_p, letter\_size, lr ); //uczenie perceptronu wynikowego

//na podstawie sygnałów wyjściowych pierwszej warstwy

}

private static int[] test ( Adaline[] ada, int nol, int noi ) {

int[] wyj = new int[nol \* 2];

int[] vector; //tablica przechowująca wektor sygnałów wejściowych do testowania pierwszej warstwy sieci

int[] vector\_p = new int[noi]; //tablica przechowująca wektor sygnałów wyjściowych pierwszej warstwy sieci

vector\_p[0] = 1; //bias

for ( int i = 0; i < 2; i++ ) { //testowanie, celem upewnienia się, czy sieć już nauczona

for ( int j = 0; j < nol; j++ ) {

vector = Alphabet.*getLetter*( i, j );

*format*( vector );

for ( int k = 0; k < noi - 1; k++ )

vector\_p[k + 1] = ada[k].test( vector );

wyj[i \* nol + j] = ada[noi - 1].test( vector\_p );

}

}

return wyj;

}

//w przypadku adaline sygnały wejściowe = 0 muszą być zamienione na sygnały -1

private static void format( int[] vector ){

for ( int k = 0; k < vector.length; k++ )

if ( vector[k] == 0 ) vector[k] = -1;

}

}

**package alphabet;**

import java.util.Arrays;

public class Alphabet {

static int[][][][] *letters* = {

{//wielkie litery

{ { 0, 0, 1, 0, 0 }, { 0, 1, 0, 1, 0 }, { 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 } }, // A 0

{ { 1, 1, 1, 1, 0 }, { 1, 0, 0, 0, 1 }, { 1, 0, 0, 0, 1 }, { 1, 1, 1, 1, 0 }, { 1, 0, 0, 0, 1 }, { 1, 0, 0, 0, 1 }, { 1, 1, 1, 1, 0 } }, // B 1

{ { 0, 1, 1, 1, 1 }, { 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, 1, 1, 1, 1 } }, // C 2

{ { 1, 1, 1, 1, 0 }, { 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, 1, 1, 1, 0 } }, // D 3

{ { 1, 1, 1, 1, 1 }, { 1, 0, 0, 0, 0 }, { 1, 0, 0, 0, 0 }, { 1, 1, 1, 1, 1 }, { 1, 0, 0, 0, 0 }, { 1, 0, 0, 0, 0 }, { 1, 1, 1, 1, 1 } }, // E 4

{ { 1, 1, 1, 1, 1 }, { 1, 0, 0, 0, 0 }, { 1, 0, 0, 0, 0 }, { 1, 1, 1, 1, 1 }, { 1, 0, 0, 0, 0 }, { 1, 0, 0, 0, 0 }, { 1, 0, 0, 0, 0 } }, // F 5

{ { 0, 1, 1, 1, 1 }, { 1, 0, 0, 0, 0 }, { 1, 0, 0, 0, 0 }, { 1, 0, 0, 1, 1 }, { 1, 0, 0, 0, 1 }, { 1, 0, 0, 0, 1 }, { 0, 1, 1, 1, 1 } }, // G 6

{ { 1, 0, 0, 0, 1 }, { 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 } }, // H 7

{ { 0, 0, 1, 0, 1 }, { 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, 1, 0, 0 } }, // I 8

{ { 0, 1, 1, 1, 0 }, { 0, 0, 0, 1, 0 }, { 0, 0, 0, 1, 0 }, { 0, 0, 0, 1, 1 }, { 0, 0, 1, 1, 0 }, { 0, 1, 0, 1, 0 }, { 0, 0, 1, 0, 0 } }, // J 9

{ { 1, 0, 0, 0, 1 }, { 1, 0, 0, 1, 0 }, { 1, 0, 1, 0, 0 }, { 1, 1, 0, 0, 0 }, { 1, 0, 1, 0, 0 }, { 1, 0, 0, 1, 0 }, { 1, 0, 0, 0, 1 } }, // K 10

{ { 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, 1, 0, 0 }, { 0, 0, 1, 1, 1 } }, // L 11

{ { 0, 1, 0, 1, 0 }, { 1, 0, 1, 0, 1 }, { 1, 0, 1, 0, 1 }, { 1, 0, 0, 0, 1 }, { 1, 0, 0, 0, 1 }, { 1, 0, 0, 0, 1 }, { 1, 0, 0, 0, 1 } }, // M 12

{ { 1, 1, 0, 0, 1 }, { 1, 1, 0, 0, 1 }, { 1, 0, 1, 0, 1 }, { 1, 0, 1, 0, 1 }, { 1, 0, 1, 0, 1 }, { 1, 0, 0, 1, 1 }, { 1, 0, 0, 1, 1 } }, // N 13

{ { 0, 1, 1, 1, 0 }, { 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 }, { 0, 1, 1, 1, 0 } }, // O 14

{ { 1, 1, 1, 1, 0 }, { 1, 0, 0, 0, 1 }, { 1, 0, 0, 0, 1 }, { 1, 1, 1, 1, 0 }, { 1, 0, 0, 0, 0 }, { 1, 0, 0, 0, 0 }, { 1, 0, 0, 0, 0 } }, // P 15

{ { 0, 1, 1, 1, 0 }, { 1, 0, 0, 0, 1 }, { 1, 0, 0, 0, 1 }, { 1, 0, 0, 0, 1 }, { 1, 0, 0, 0, 1 }, { 1, 0, 0, 1, 0 }, { 0, 1, 1, 0, 1 } }, // Q 16

{ { 1, 1, 1, 1, 0 }, { 1, 0, 0, 0, 1 }, { 1, 0, 0, 0, 1 }, { 1, 1, 1, 1, 0 }, { 1, 0, 1, 0, 0 }, { 1, 0, 0, 1, 0 }, { 1, 0, 0, 0, 1 } }, // R 17

{ { 0, 1, 1, 1, 1 }, { 1, 0, 0, 0, 0 }, { 1, 0, 0, 0, 0 }, { 0, 1, 1, 1, 0 }, { 0, 0, 0, 0, 1 }, { 0, 0, 0, 0, 1 }, { 1, 1, 1, 1, 0 } }, // S 18

{ { 1, 1, 1, 1, 1 }, { 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, 1, 0, 0 } }, // T 19

{ { 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 }, { 0, 1, 1, 1, 0 } }, // U 20

{ { 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, 1, 0, 1 }, { 0, 1, 0, 1, 0 } }, // W 21

{ { 1, 0, 0, 0, 1 }, { 1, 0, 0, 0, 1 }, { 1, 0, 0, 0, 1 }, { 1, 0, 0, 0, 1 }, { 0, 1, 0, 1, 0 }, { 0, 1, 0, 1, 0 }, { 0, 0, 1, 0, 0 } }, // V 22

{ { 1, 0, 0, 0, 1 }, { 1, 0, 0, 0, 1 }, { 0, 1, 0, 1, 0 }, { 0, 0, 1, 0, 0 }, { 0, 1, 0, 1, 0 }, { 1, 0, 0, 0, 1 }, { 1, 0, 0, 0, 1 } }, // X 23

{ { 1, 0, 0, 0, 1 }, { 0, 1, 0, 1, 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 } }, // Y 24

{ { 1, 1, 1, 1, 1 }, { 0, 0, 0, 0, 1 }, { 0, 0, 0, 1, 0 }, { 0, 0, 1, 0, 0 }, { 0, 1, 0, 0, 0 }, { 1, 0, 0, 0, 0 }, { 1, 1, 1, 1, 1 } } // Z 25

},

{//małe litery

{ { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 1, 1, 1, 0 }, { 1, 0, 0, 1, 0 }, { 1, 0, 0, 1, 0 }, { 0, 1, 1, 1, 1 } }, // a 0

{ { 1, 0, 0, 0, 0 }, { 1, 0, 0, 0, 0 }, { 1, 0, 0, 0, 0 }, { 1, 1, 1, 0, 0 }, { 1, 0, 0, 1, 0 }, { 1, 0, 0, 1, 0 }, { 1, 1, 1, 0, 0 } }, // b 1

{ { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 1, 1, 1, 0 }, { 1, 0, 0, 0, 0 }, { 1, 0, 0, 0, 0 }, { 0, 1, 1, 1, 0 } }, // c 2

{ { 0, 0, 0, 1, 0 }, { 0, 0, 0, 1, 0 }, { 0, 0, 0, 1, 0 }, { 0, 1, 1, 1, 0 }, { 1, 0, 0, 1, 0 }, { 1, 0, 0, 1, 0 }, { 0, 1, 1, 1, 0 } }, // d 3

{ { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 1, 1, 0, 0 }, { 1, 0, 1, 0, 0 }, { 1, 1, 0, 0, 0 }, { 0, 1, 1, 1, 0 } }, // e 4

{ { 0, 0, 0, 0, 0 }, { 0, 1, 1, 0, 0 }, { 1, 0, 0, 0, 0 }, { 1, 1, 1, 0, 0 }, { 1, 0, 0, 0, 0 }, { 1, 0, 0, 0, 0 }, { 1, 0, 0, 0, 0 } }, // f 5

{ { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 0, 1, 0, 0 }, { 0, 1, 0, 1, 0 }, { 0, 0, 1, 1, 0 }, { 0, 0, 0, 1, 0 }, { 0, 1, 1, 0, 0 } }, // g 6

{ { 1, 0, 0, 0, 0 }, { 1, 0, 0, 0, 0 }, { 1, 0, 0, 0, 0 }, { 1, 1, 1, 0, 0 }, { 1, 0, 1, 0, 0 }, { 1, 0, 1, 0, 0 }, { 1, 0, 1, 0, 0 } }, // h 7

{ { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 0, 1, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 0, 1, 0, 0 }, { 0, 0, 1, 0, 0 } }, // i 8

{ { 0, 0, 0, 0, 0 }, { 0, 0, 1, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 0, 1, 0, 0 }, { 0, 0, 1, 0, 0 }, { 0, 0, 1, 0, 0 }, { 0, 1, 1, 0, 0 } }, // j 9

{ { 1, 0, 0, 0, 0 }, { 1, 0, 1, 0, 0 }, { 1, 0, 1, 0, 0 }, { 1, 1, 0, 0, 0 }, { 1, 0, 1, 0, 0 }, { 1, 0, 1, 0, 0 }, { 1, 0, 1, 0, 0 } }, // k 10

{ { 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 }, { 1, 0, 0, 0, 0 }, { 1, 1, 0, 0, 0 } }, // l 11

{ { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 1, 0, 1, 0 }, { 1, 0, 1, 0, 1 }, { 1, 0, 0, 0, 1 } }, // m 12

{ { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 1, 1, 1, 0 }, { 0, 1, 0, 1, 0 }, { 0, 1, 0, 1, 0 } }, // n 13

{ { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 0, 1, 0, 0 }, { 0, 1, 0, 1, 0 }, { 0, 0, 1, 0, 0 } }, // o 14

{ { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 1, 1, 0, 0, 0 }, { 1, 0, 1, 0, 0 }, { 1, 1, 0, 0, 0 }, { 1, 0, 0, 0, 0 }, { 1, 0, 0, 0, 0 } }, // p 15

{ { 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, 1, 0, 0 }, { 0, 1, 0, 1, 0 } }, // q 16

{ { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 1, 0, 1, 1, 0 }, { 0, 1, 0, 0, 0 }, { 0, 1, 0, 0, 0 } }, // r 17

{ { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 0, 1, 1, 0 }, { 0, 1, 0, 0, 0 }, { 0, 1, 1, 0, 0 }, { 0, 0, 1, 0, 0 }, { 1, 1, 0, 0, 0 } }, // s 18

{ { 0, 1, 0, 0, 0 }, { 1, 1, 1, 0, 0 }, { 0, 1, 0, 0, 0 }, { 0, 1, 0, 0, 0 }, { 0, 1, 0, 0, 0 }, { 0, 1, 0, 0, 0 }, { 0, 1, 1, 0, 0 } }, // t 19

{ { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 1, 0, 1, 0 }, { 0, 1, 0, 1, 0 }, { 0, 1, 1, 1, 0 } }, // u 20

{ { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 1, 0, 0, 0, 1 }, { 1, 0, 1, 0, 1 }, { 0, 1, 0, 1, 0 } }, // w 21

{ { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 1, 0, 1, 0 }, { 0, 1, 0, 1, 0 }, { 0, 0, 1, 0, 0 } }, // v 22

{ { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 1, 0, 1, 0 }, { 0, 0, 1, 1, 0 }, { 0, 1, 0, 1, 0 } }, // x 23

{ { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 1, 0, 1, 0 }, { 0, 0, 1, 0, 0 }, { 0, 1, 0, 0, 0 }, { 1, 0, 0, 0, 0 } }, // y 24

{ { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 0, 0, 0, 0, 0 }, { 1, 1, 1, 1, 0 }, { 0, 0, 1, 0, 0 }, { 0, 1, 0, 0, 0 }, { 1, 1, 1, 1, 0 } } // z 25

}

};

//zwraca wektor sygnałów wejściowych zależny od danej litery

public static int[] getLetter ( int size, int letter ) {

int[] ret = new int[7];

Arrays.*fill*( ret, 1, 6, 0 );

ret[0] = 1;

//sektor 1

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

for ( int j = 0; j < 2; j++ )

if ( *letters*[size][letter][i][j] == 1 )

ret[1] = 1;

//sektor 2

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

for ( int j = 2; j < 3; j++ )

if ( *letters*[size][letter][i][j] == 1 )

ret[2] = 1;

//sektor 3

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

for ( int j = 3; j < 5; j++ )

if ( *letters*[size][letter][i][j] == 1 )

ret[3] = 1;

//sektor 4

for ( int i = 3; i < 7; i++ )

for ( int j = 0; j < 2; j++ )

if ( *letters*[size][letter][i][j] == 1 )

ret[4] = 1;

//sektor 5

for ( int i = 3; i < 7; i++ )

for ( int j = 2; j < 3; j++ )

if ( *letters*[size][letter][i][j] == 1 )

ret[5] = 1;

//sektor 6

for ( int i = 3; i < 7; i++ )

for ( int j = 3; j < 5; j++ )

if ( *letters*[size][letter][i][j] == 1 )

ret[6] = 1;

return ret;

}

}

**Bibliografia:**

<https://pl.wikipedia.org/wiki/Neuron_McCullocha-Pittsa>

<https://en.wikipedia.org/wiki/ADALINE>

https://en.wikipedia.org/wiki/Perceptron