Konrad Tabiś

Laboratorium nr 2

Sprawozdanie - Podstawy Sztucznej Inteligencji – Scenariusz 2

Temat ćwiczenia: Budowa i działanie sieci jednowarstwowej.

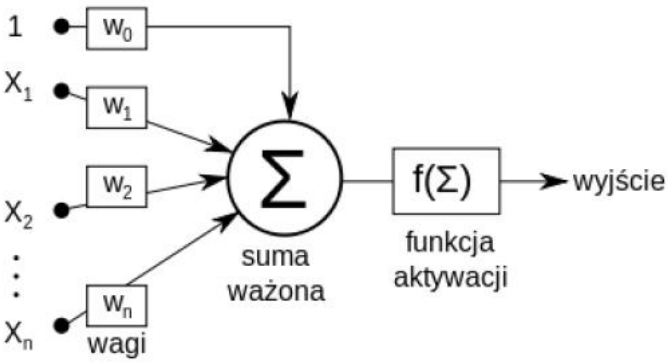
1.Cel ćwiczenia:

Poznanie budowy i działania jednowarstwowych sieci neuronowych oraz uczenie rozpoznawania wielkości liter.

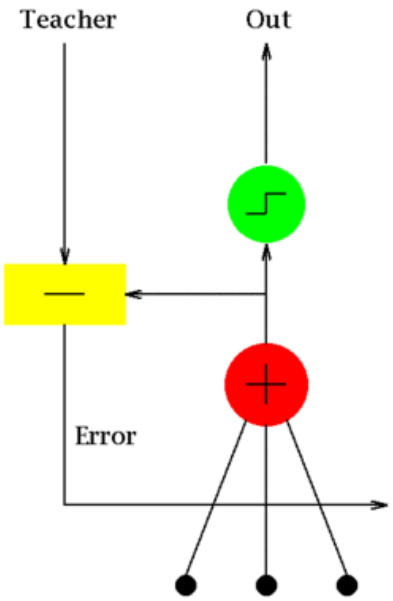
2.Syntetyczny opis budowy algorytmu uczenia się

W celu wykonania ćwiczenia stworzyłem dwie jednowarstwowe sieci, z wykorzystaniem różnych algorytmów. Pierwsza korzysta z perceptronu wg McCullocha-Pittsa, druga typu Adaline.

Perceptron McCullocha-Pittsa:



Adaline:



Do nauki perceptronów oraz adaline wykorzystałem algorytm Widrow-Hoffa.

Sieci składają się z siedmiu neuronów. Pierwsza warstwa składa się z sześciu neuronów, które przysyłają wynik do siódmego – neuronu wyjściowego. Każdy z neuronów otrzymuje 7 sygnałów wejściowych.

Budowa perceptronu

Do budowy perceptronu wykorzystałem podany na wykładzie model McCullocha-Pittsa.

Metody klasy Perceptron:
active funkcja progowa która zwraca wynik 0 lub 1 dla podanego argumentu
process sumuje iloczyn sygnałów wejściowych i odpowiadających im wag. Uruchamia metodę activate

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

Budowa Adaline

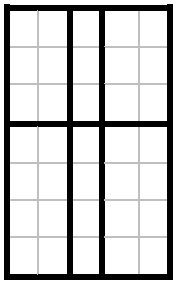
Budowa Adaline jest bardzo podobna do budowy perceptronu. Różni się tylko tym, że modyfikowanie wag wykonuje się z pominięciem funkcji aktywacji.

Metody klasy Adaline:
active funkcja progowa która zwraca wynik -1 lub 1 dla podanego argumentu
process sumuje iloczyn sygnałów wejściowych i odpowiadających im wag, oraz zwraca ten wynik.

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

test uruchamia metodę active z wynikiem metody process jako parametrem.

Litery alfabetu
Litery stworzyłem za pomocą dwuwymiarowej tablicy 7x5 którą podzieliłem na 6 obszarów. Jeżeli w danym obszarze pojawiła się choćby jedna komórka ustawiona na 1 odpowiedni sygnał wejściowy do neuronu zwracał sygnał 1. Kiedy cały obszar był wypełniony zerami zwracał sygnał 0.
Podział tablicy:



Przykład:

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

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

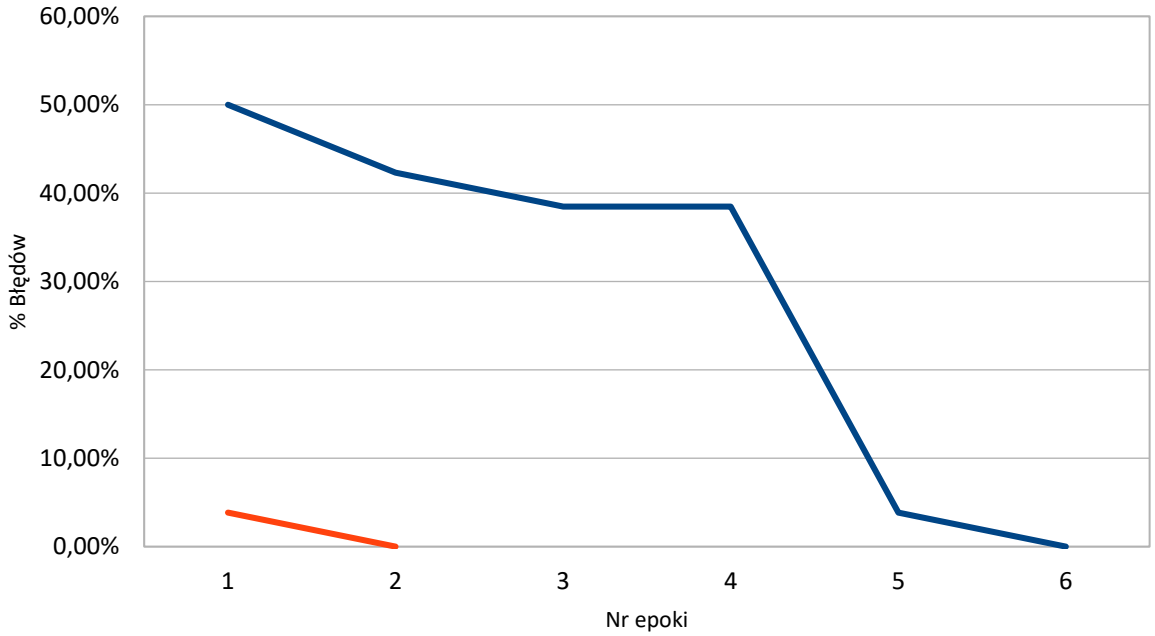
3.Proces uczenia i zebrane wyniki

W klasie Letters przygotowałem 26 liter małych i wielkich które wykorzystałem do uczenia rozpoznawania liter. Proces uczenia przeprowadziłem dla różnych współczynników uczenia.

Wyniki uczenia się różniły się z powodu losowych wag początkowych, dlatego dla każdego współczynnika uczenia się robiłem kilkukrotnie test i wybierałem najlepsze wyniki.

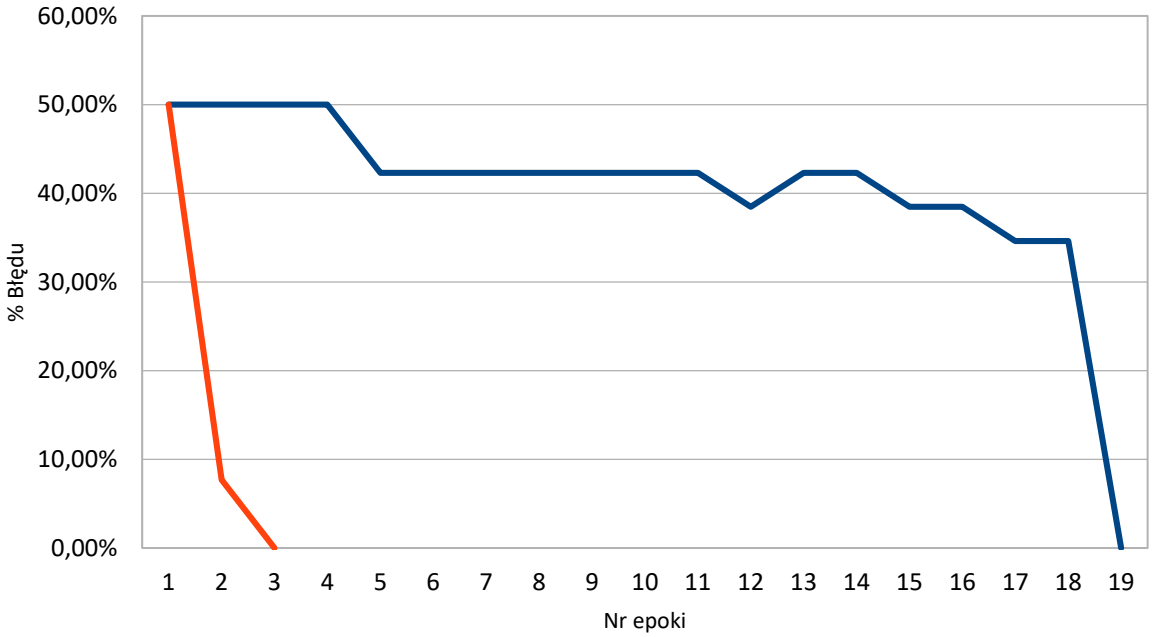
Współczynnik uczenia = 0.1

Nr Epoki	% Błędów Perceptron	% Błędów Adaline
1	50,00%	3,85%
2	42,31%	0,0%
3	38,46%	
4	38,46%	
5	3,85%	
6	0,00%	

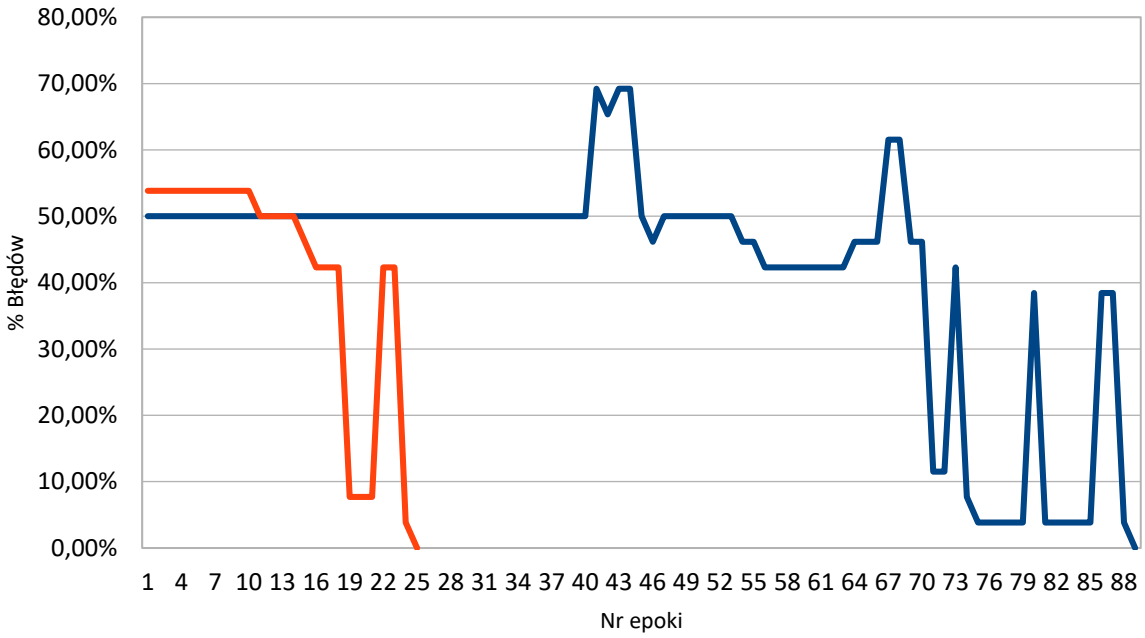


Współczynnik uczenia = 0.01

Nr Epoki	% Błędów Perceptron	% Błędów Adaline
1	50,00%	50,00%
2	50,00%	7,69%
3	50,00%	0,00%
4	50,00%	
5	42,31%	
6	42,31%	
7	42,31%	
8	42,31%	
9	42,31%	
10	42,31%	
11	42,31%	
12	38,46%	
13	42,31%	
14	42,31%	
15	38,46%	
16	38,46%	
17	34,62%	
18	34,62%	
19	0,00%	



Współczynnik uczenia = 0.001



4.Analiza wyników:

Patrząc na wykresy od razu rzuca nam się w oczy, że algorytm Adaline w każdym przypadku potrzebował mniej epok na nauczenie się. Kolejnym wnioskiem, który możemy z łatwością wyciągnąć jest to, że im mniejszy współczynnik uczenia tym więcej epok oba algorytmy potrzebowały na nauczenie się rozpoznawania liter.

Testując oba algorytmy 13 literami, którymi nie uczyłem, tylko w jednym przypadku miałem błąd wielkości 7,69% przy algorytmie Adaline i współczynniku uczenia 0.001. W pozostałych przypadkach oba algorytmy były bezbłędne.

5.Wnioski:

Neuron typu Adaline jest o wiele wydajniejszy niż perceptron. Niezależnie od współczynnika uczenia zawsze uczył się kilkukrotnie szybciej. 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.

6.Listing kodu:

```
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 sa 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
   }
}
```

```
import Letters.Letters;

import java.util.Arrays;

public class Main {

   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 );

      int proc;
      while ( ! Arrays.equals( y, wyj ) ) {

         proc=0;
         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 );
         for ( int i = 0; i < nol * 2; i++ ) //testowanie, do sprawozdania
            if ( wyj[i] != y[i] )
               proc++;
         counter++;
         System.out.format( "%.6f%n", ( double ) proc / ( nol * 2 ) );
      }

      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 = Letters.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 = Letters.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;
   }
}
```

```
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 sa 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 ) ) );
   }
}
```

```
import Letters.Letters;

import java.util.Arrays;

public class Main {

   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 );

      int proc;
      while ( ! Arrays.equals( y, wyj ) ) {

         proc=0;
         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 );

         for ( int i = 0; i < nol * 2; i++ ) //testowanie, do sprawozdania
            if ( wyj[i] != y[i] )
               proc++;

         counter++;
         System.out.format( "%.6f%n", ( double ) proc / ( nol * 2 ) );
      }

      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 = Letters.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 = Letters.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 Letters;

import java.util.Arrays;

public class Letters {
   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
    },

    {//male 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;
   }
}
```