Kamil Witek

**Sprawozdanie – scenariusz 2**

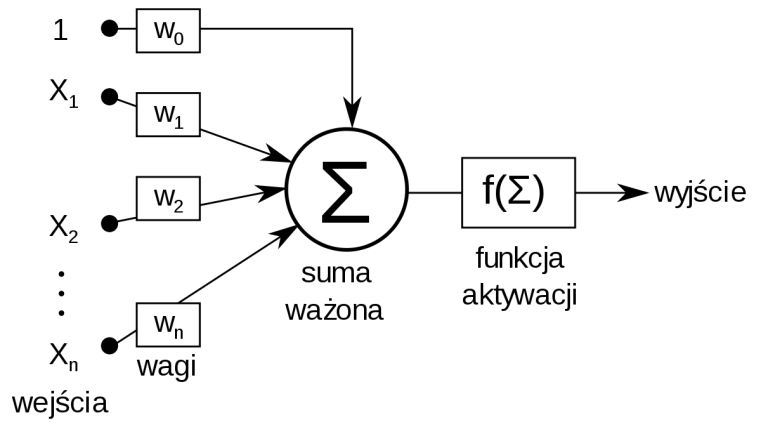
Temat: ćwiczenia: Budowa i działanie sieci jednowarstwowej

**Cel ćwiczenia**

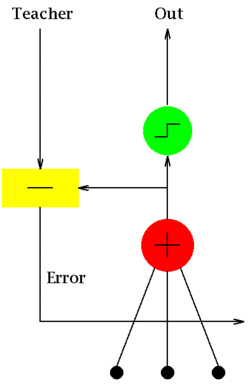
Celem ćwiczenia jest poznanie budowy i działania jednowarstwowych sieci neuronowych oraz uczenie rozpoznawania wielkości liter.

**Syntetyczny opis budowy wykorzystanego algorytmu uczenia:**

Aby wykonać ćwiczenie stworzono dwie jednowarstwowe sieci, przy użyciu modelu perceptronu McCullocha-Pitts oraz Adaline.



Rysunek 1 Schemat neuronu McCullocha-Pittsa



Rysunek 2 Schemat neuronu Adaline

Sieć składa się z 6 neuronów pierwszej warstwy oraz 1 neuronu wyjściowego, który daje ostateczny wynik.

Do nauki wygenerowano 13 małych oraz 13 dużych liter alfabetu i podzielono je siatką w postaci dwuwymiarowej tablicy 5x7, następnie każdą literę podzielono na 6 obszarów.

|  |  |  |
| --- | --- | --- |
| 1 1  1 0  1 0 | 1  0  0 | 1 0  0 1  0 1 |
| 1 1  1 0  1 0  1 0  1 0 | 1  0  0  0  0 | 1 0  0 0  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  1 1  1 0  1 0 | 0  1  0  0  0 | 0 0  0 0  0 0  0 0  0 0 |

Zaimplementowano uczenie się neuronu według algorytmu Widrowa – Hoffa.

**Perceptron:**

Konstruktor „Perceptron()” zapisuje przekazaną ilość wejść do obiektu oraz losuje wagi.

Metoda „activation()” wykorzystuje funkcję progową która zwraca wynik 0 lub 1

Metoda „suma()” sumuje iloczyn wag i danych uczących.

Metoda „learn()” wywołuje metodę proces dla otrzymanej tablicy wejść, następnie na podstawie otrzymanego wyniku modyfikuje wagi.

**Adaline:**

Konstruktor „Adaline()” zapisuje przekazaną ilość wejść do obiektu oraz losuje wagi.

Metoda „activation()” wykorzystuje funkcję progową która zwraca wynik -1 lub 1

Metoda „suma()” sumuje iloczyn wag i danych uczących.

Metoda „learn()” wywołuje metodę proces dla otrzymanej tablicy wejść, następnie na podstawie otrzymanego wyniku modyfikuje wagi.

Metoda „test()” uruchamia metodę funkcji aktywacji „activation()” z wartością którą zwróciła funkcja „suma()”

**Zestawienie otrzymanych wyników**

Rysunek 3 Błąd uczenia dla współczynnika nauki 0.1

Rysunek 4 Błąd uczenia dla współczynnika nauki 0.01

Rysunek 5 Błąd uczenia dla współczynnika nauki 0.001

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

W obu użytych sieciach neuronów typu perceptron jak i adaline, możemy zauważyć że im mniejszy współczynnik uczenia, tym więcej czasu potrzeba do nauki. Porównując oba algorytmy widać że adaline radził sobie znacznie lepiej i potrzebował mniej iteracji do nauki.

|  |  |  |
| --- | --- | --- |
| Współczynnik uczenia | Perceptron – ilość epok | Adaline – ilość epok |
| 0.1 | 11 | 2 |
| 0.01 | 25 | 8 |
| 0.001 | 135 | 63 |

**Wnioski**

Na podstawie otrzymanych wyników można stwierdzić, że sieć składająca się z neuronów typu adaline jest znacznie wydajniejsza od neuronów typu perceptron. Niezależnie od zadanego współczynnika uczenia adaline radził sobie znacznie lepiej. Jest to spowodowane sposobem w jaki adaline modyfikuje wagi. Modyfikacja wag odbywa się przed funkcją aktywacji, dzięki czemu jeśli otrzymana wartość odbiega znacznie od oczekiwanej, waga jest zmieniana w większym stopniu niż ma to miejsce w perceptronie.

**Listing kodu**

**package** Perceptron;  
  
**import** java.util.Arrays;  
  
**public class** Main{  
  
 **public static void** main ( String[] args ) {  
  
 **int** liczbaWejsc = 7; *//ilość wejść* **int** liczbaLiter = 13; *//ilość liter MAX 26* **int** epoka = 0; *//licznik ilości epok uczenia się* **double** wspolczynnikNauki = 0.1; *//krok uczenia się* Perceptron[] perceptron = **new** Perceptron[liczbaWejsc];  
 **for** ( **int** i = 0; i < liczbaWejsc; i++ )  
 perceptron[i] = **new** Perceptron( liczbaWejsc );  
  
 **int**[] y = **new int**[liczbaLiter \* 2]; *//0 - duża litera, 1 - mała litera* Arrays.*fill*( y, 0, liczbaLiter, 0 );  
 Arrays.*fill*( y, liczbaLiter, liczbaLiter \* 2, 1 );  
  
 **int**[] wyj = **new int**[liczbaLiter \* 2]; *//tablica przechowująca wyniki testowania perceptronu* Arrays.*fill*( wyj, 0, liczbaLiter \* 2, 0 );  
 **int** licznik;  
 **while** ( ! Arrays.*equals*( y, wyj ) ) {  
 licznik=0;  
 **for** ( **int** i = 0; i < 2; i++ ) *//0 - wielkie litery, 1 - małe litery* **for** ( **int** j = 0; j < liczbaLiter; j++ )  
 *learn*( perceptron, liczbaWejsc, wspolczynnikNauki, i, j );  
  
 wyj = *test*( perceptron, liczbaLiter, liczbaWejsc );  
  
 **for** (**int** i = 0; i < liczbaLiter \* 2; i++) {  
 **if** (wyj[i] != y[i])  
 licznik++;  
 }  
  
 System.***out***.println((licznik \* 100) / (2 \* liczbaLiter)+**",00%"**);  
 epoka++;  
 }  
  
 System.***out***.println( **"Ilość kroków do nauczenia się = "** + epoka );  
 }  
  
  
 **public static void** learn ( Perceptron[] perceptron, **int** liczbaWejsc, **double** wspolczynnikNauki, **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**[liczbaWejsc]; *//tablica przechowująca wektor sygnałów wyjściowych pierwszej warstwy sieci* vector\_p[0] = 1; *//bias* **for** ( **int** k = 0; k < liczbaWejsc - 1; k++ ) { *//uczenie pierwszej warstwy* perceptron[k].learn( vector, i, wspolczynnikNauki );  
 vector\_p[k + 1] = perceptron[k].suma( vector ); *//pobranie sygnału wyjściowego* }  
 perceptron[liczbaWejsc - 1].learn( vector\_p, i, wspolczynnikNauki ); *//uczenie perceptronu wynikowego na podstawie sygnałów wyjściowych pierwszej warstwy* }  
  
 **public static int**[] test ( Perceptron[] perceptron, **int** liczbaLiter, **int** liczbaWejsc ) {  
 **int**[] wyj = **new int**[liczbaLiter \* 2];  
 **int**[] vector; *//tablica przechowująca wektor sygnałów wejściowych do testowania pierwszej warstwy sieci* **int**[] vector\_p = **new int**[liczbaWejsc]; *//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 < liczbaLiter; j++ ) {  
 vector = Alphabet.*getLetter*( i, j );  
 **for** ( **int** k = 0; k < liczbaWejsc - 1; k++ )  
 vector\_p[k + 1] = perceptron[k].suma( vector );  
  
 wyj[i \* liczbaLiter + j] = perceptron[liczbaWejsc - 1].suma( vector\_p );  
 }  
 }  
 **return** wyj;  
 }

**package** Perceptron;  
  
**import** java.util.Random;  
  
**public class** Perceptron {  
  
 **private int liczbaWejsc**; *//ilość wejść* **private double**[] **w**; *//wagi* **public** Perceptron ( **int** iloscWejsc ) {  
 **liczbaWejsc** = iloscWejsc;  
 **w** = **new double**[**liczbaWejsc**];  
  
 **for** ( **int** i = 0; i < **liczbaWejsc**; i++ )  
 **w**[i] = **new** Random().nextDouble(); *//nadanie wag poczatkowych, losowo* }  
  
 *//Funkcja aktywacji* **public int** activation(**double** output) {  
 **if** (output < 0) **return** 0;  
 **else return** 1;  
 }  
  
  
 **public int** suma ( **int**[] x ) {  
 **double** y\_p = 0;  
 **for** ( **int** i = 0; i < **liczbaWejsc**; i++ )  
 y\_p += x[i] \* **w**[i];  
  
 **return** activation( y\_p );  
 }  
  
 *//nauka* **public void** learn ( **int**[] x, **double** y, **double** lr ) {  
 **double** y\_p = suma( x );  
  
 **for** ( **int** i = 0; i < **liczbaWejsc**; i++ )  
 **w**[i] += ( y - y\_p ) \* lr \* x[i]; *//modyfikacja wag* }

}

**package** Adaline;  
  
**import** java.util.Arrays;  
  
**public class** Main{  
  
 **public static void** main ( String[] args ) {  
  
 **int** liczbaWejsc = 7; *//ilość wejść* **int** liczbaLiter = 13; *//ilość liter MAX 26* **int** epoka = 0; *//licznik ilości epok uczenia się* **double** wspolczynnikNauki = 0.1; *//krok uczenia się* Adaline[] adaline = **new** Adaline[liczbaWejsc];  
 **for** ( **int** i = 0; i < liczbaWejsc; i++ )  
 adaline[i] = **new** Adaline( liczbaWejsc );  
  
 **int**[] y = **new int**[liczbaLiter \* 2]; *//-1 - duża litera, 1 - mała litera* Arrays.*fill*( y, 0, liczbaLiter, - 1 );  
 Arrays.*fill*( y, liczbaLiter, liczbaLiter \* 2, 1 );  
  
 **int**[] wyj = **new int**[liczbaLiter \* 2]; *//tablica przechowująca wyniki testowania adalineline* Arrays.*fill*( wyj, 0, liczbaLiter \* 2, 0 );  
 **int** licznik = 0;  
 **while** ( ! Arrays.*equals*( y, wyj ) ) {  
 licznik=0;  
 **for** ( **int** i = 0; i < 2; i++ ) { *//-1 - wielkie litery, 1 - małe litery* **for** ( **int** j = 0; j < liczbaLiter; j++ )  
 *learn*( adaline, liczbaWejsc, wspolczynnikNauki, i, j );  
 }  
  
 wyj = *test*( adaline, liczbaLiter, liczbaWejsc );  
 **for** (**int** i = 0; i < liczbaLiter \* 2; i++) {  
 **if** (wyj[i] != y[i])  
 licznik++;  
 }  
  
 System.***out***.println((licznik \* 100) / (2 \* liczbaLiter)+**",00%"**);  
 epoka++;  
 }  
  
 System.***out***.println( **"Ilość kroków do nauczenia się = "** + epoka );  
 }  
  
  
 **private static void** learn ( Adaline[] adaline, **int** liczbaWejsc, **double** wspolczynnikNauki, **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**[liczbaWejsc]; *//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 < liczbaWejsc - 1; k++ ) { *//uczenie pierwszej warstwy* adaline[k].learn( vector, letter\_size, wspolczynnikNauki );  
 vector\_p[k + 1] = adaline[k].test( vector ); *//pobranie sygnału wyjściowego* }  
 adaline[liczbaWejsc - 1].learn( vector\_p, letter\_size, wspolczynnikNauki ); *//uczenie perceptronu wynikowego na podstawie sygnałów wyjściowych pierwszej warstwy* }  
  
 **private static int**[] test ( Adaline[] adaline, **int** liczbaLiter, **int** liczbaWejsc ) {  
 **int**[] wyj = **new int**[liczbaLiter \* 2];  
 **int**[] vector; *//tablica przechowująca wektor sygnałów wejściowych do testowania pierwszej warstwy sieci* **int**[] vector\_p = **new int**[liczbaWejsc]; *//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 < liczbaLiter; j++ ) {  
 vector = Alphabet.*getLetter*( i, j );  
 *format*( vector );  
  
 **for** ( **int** k = 0; k < liczbaWejsc - 1; k++ )  
 vector\_p[k + 1] = adaline[k].test( vector );  
  
 wyj[i \* liczbaLiter + j] = adaline[liczbaWejsc - 1].test( vector\_p );  
 }  
 }  
 **return** wyj;  
 }  
  
 *//w przypadku adalineline 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** Adaline;  
  
**import** java.util.Random;  
  
**public class** Adaline {  
  
 **private int liczbaWejsc**; *//ilość wejść* **private double**[] **w**; *//wagi* **public** Adaline ( **int** iloscWejsc ) {  
 **liczbaWejsc** = iloscWejsc;  
 **w** = **new double**[**liczbaWejsc**];  
  
 **for** ( **int** i = 0; i < **liczbaWejsc**; i++ )  
 **w**[i] = **new** Random().nextDouble(); *//wagi początkowe sa losowane* }  
  
 *//Funkcja aktywacji* **public int** activation(**double** output) {  
 **if** (output < 0) **return** -1;  
 **else return** 1;  
 }  
  
 *//suma* **public double** suma ( **int**[] x ) {  
 **double** y\_p = 0;  
 **for** ( **int** i = 0; i < **liczbaWejsc**; i++ )  
 y\_p += x[i] \* **w**[i];  
  
 **return** y\_p;  
 }  
  
 *//nauka* **public void** learn ( **int**[] x, **double** y, **double** lr ) {  
 **double** y\_p = suma( x );  
  
 **for** ( **int** i = 0; i < **liczbaWejsc**; i++ )  
 **w**[i] += ( y - y\_p ) \* lr \* x[i]; *//modyfikacja wag* }  
  
 *//testowanie* **public int** test ( **int**[] x )  
 {  
 **return** ( activation( suma( x ) ) );  
 }  
}

**package** Adaline;  
  
**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* },  
  
 {*//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;  
 }  
}