

Celem ćwiczenia było poznanie budowy i działania sieci Kohonena przy wykorzystaniu reguły WTM do odwzorowania istotnych cech liter alfabetu.

1) Syntetyczny opis budowy użytej sieci i algorytmów uczenia

Reguła Kohonena opiera się na mechanizmie współzawodnictwa między neuronami.

Wagi każdego neuronu tworzą wektor $\mathbf{w}_i = [w_{i1}, w_{i2}, \dots, w_{iN}]^T$. Przy założeniu normalizacji wektorów wejściowych, we współzawodnictwie wygrywa neuron, którego wagi najmniej różnią się od odpowiednich składowych tego wektora. Zwycięski neuron spełnia relację:

$$d(\mathbf{x}, \mathbf{w}_w) = \min_{1 \leq i \leq n} d(\mathbf{x}, \mathbf{w}_i)$$

Gdzie $d(\mathbf{x}, \mathbf{w})$ oznacza odległość w sensie wybranej metryki między wektorem \mathbf{x} i wektorem \mathbf{w} a n to ilość neuronów. Podczas ćwiczenia do obliczania odległości między wektorami użyłem miary według normy L_1 (Mahnattan):

$$d(\mathbf{x}, \mathbf{w}_i) = \sqrt{\sum_{j=1}^N |x_j - w_{ij}|}$$

Zmiana wag zachodzi wg zależności:

$$\mathbf{w}_i(k+1) = \mathbf{w}_i(k) + \eta_i(k)[\mathbf{x} - \mathbf{w}_i(k)]$$

W strategii WTM poza neuronem zwycięzcą swoje wagi modyfikują także neurony sąsiednie. Jako neurony sąsiednie określa się te, które znajdują się w zakresie określonego promienia od neuronu zwycięzcy. Wartość tego promienia definiujemy sami. Podczas ćwiczenia użyłem sąsiedztwa typu prostokątnego, tak więc jeśli jakiś neuron został określony jako sąsiedni do zwycięzcy, to jego wagi były modyfikowane w takim samym stopniu jak u zwycięzcy. Pozostałe neurony, które nie zostały zakwalifikowane jako sąsiednie nie zmieniają swoich wag:

$$G(i, \mathbf{x}) = \begin{cases} 1 & \text{dla } d(i, \mathbf{w}) \leq \lambda \\ 0 & \text{dla pozostałych} \end{cases}$$

Dodatkowo podczas ćwiczenia użyłem sporej nadmiarowości jeśli chodzi o ilość neuronów. Było to konieczne, ponieważ inicjalizacja wag sieci jest losowa, tak więc część neuronów można znaleźć się w strefie, w której nie ma danych lub ich liczba jest znikoma. Neurony takie mają niewielkie szanse na zwycięstwo i zwane są neuronami martwymi.

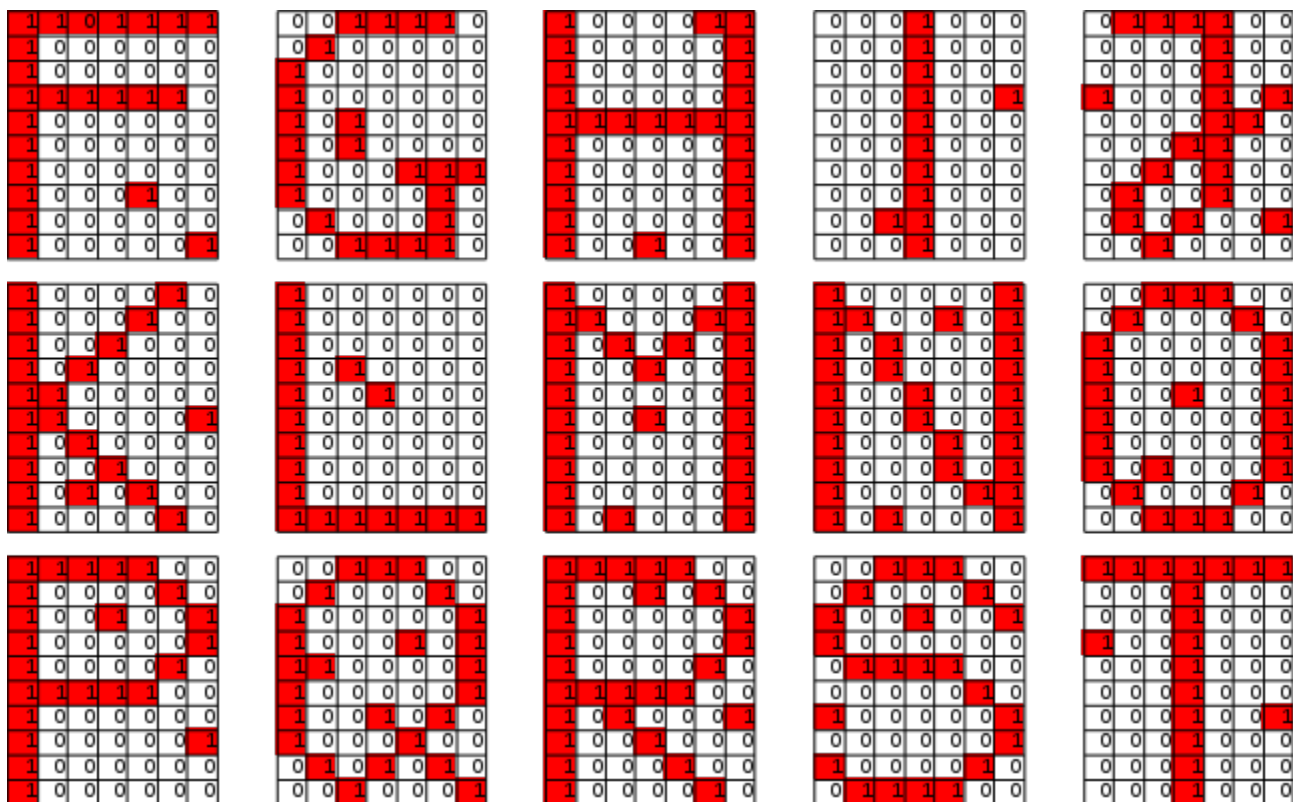
2) Zestawienie otrzymanych wyników

Jako dane uczące i testujące wykorzystałem własnoręcznie stworzone 20 liter alfabetu o rozmiarach 7x10, tak więc wektor wejściowy składał się z 70 składowych. Poniżej przedstawiam dane uczące:

0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 1 0 0 0 1 0 0 1 0 0 0 1 0 1 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1	1 1 1 1 1 1 0 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 1 1 1 1 1 0 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 1 1 1 1 1 0	0 0 0 1 1 1 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 1 1 0	1 1 1 1 1 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 1 0 1 1 1 1 1 0 0	1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 1 1 1 1 1
1 1 0 1 1 1 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 1 1 1 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0	0 0 1 1 1 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 1 1 1 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 1 1 1 0	1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1	0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0	0 1 1 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 0 1 0 0 0 1 0 0 1 0 0 0 1 0 1 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0
1 0 0 0 0 1 0 1 0 0 0 1 0 0 1 0 0 1 0 0 0 1 0 1 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 1 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 0 0 1 0 0 0 1 0 0	1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 1 1 1 1 1	1 0 0 0 0 0 1 1 1 0 0 0 1 1 1 0 1 0 1 0 1 1 0 0 1 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1	1 0 0 0 0 0 1 1 1 0 0 0 0 1 1 0 1 0 0 0 1 1 0 1 0 0 0 1 1 0 0 1 0 0 1 1 0 0 1 0 0 1 1 0 0 0 1 0 1 1 0 0 0 1 0 1 1 0 0 0 1 0 1 1 0 0 0 0 1 1 1 0 0 0 0 0 1	0 0 1 1 1 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 0 1 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 1 1 0 0
1 1 1 1 1 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0	0 0 1 1 1 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 0 1 0 1 0 1 0 0 1 0 1 0 1 0 0 0 1 0 0 1 1	1 1 1 1 1 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 0 1 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 0 0	0 0 1 1 1 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 1 1 1 1 0 0 0 1 1 1 1 0 0	1 1 1 1 1 1 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0

Z kolei jako dane testujące użyłem te same litery, jednak dodatkowo zaszumione:

0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 1 0 0 0 1 0 0 1 0 0 0 1 0 1 0 0 0 1 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 1 0 0 0 1 1 0 0 0 0 0 1	1 1 1 1 1 1 0 1 0 0 0 0 0 1 1 0 0 1 0 0 1 1 0 0 0 0 0 1 1 1 1 1 1 1 0 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 1 0 0 0 0 1 1 1 1 1 1 1 0	0 0 0 1 1 1 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 1 1 0 0 0 0 1 1 1 0	1 1 1 1 1 0 0 1 0 0 0 0 1 0 1 0 0 1 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 1 0 1 1 1 1 1 0 0 1 1 1 1 1 0 0	1 1 1 1 1 1 1 1 0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 1 1 1 1 1 1 1
0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 1 0 0 0 1 0 0 1 0 0 0 1 0 1 0 0 0 1 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 1 0 0 0 1 1 0 0 0 0 0 1	1 1 1 1 1 1 0 1 0 0 0 0 0 1 1 0 0 1 0 0 1 1 0 0 0 0 0 1 1 1 1 1 1 1 0 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 1 0 0 0 0 1 1 1 1 1 1 1 0	0 0 0 1 1 1 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 1 1 0 0 0 0 1 1 1 0	1 1 1 1 1 0 0 1 0 0 0 0 1 0 1 0 0 1 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 1 0 1 1 1 1 1 0 0 1 1 1 1 1 0 0	1 1 1 1 1 1 1 1 0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 1 1 1 1 1 1 1

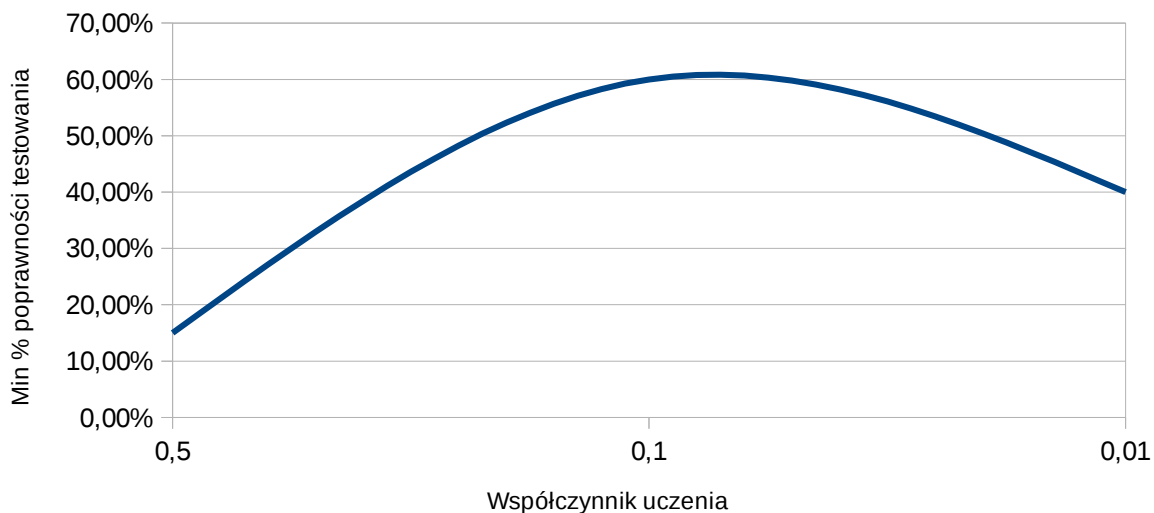


Uczenie przeprowadziłem po 10 razy dla 3 różnych współczynników uczenia: 0.5, 0.1 oraz 0.01. Dla każdego współczynnika uczenia użyłem po dwa różne promienie sąsiedztwa: 4,2 oraz 5.0. W procesie uczenia brało udział 50000 neuronów. Poniżej przedstawiam wyniki:

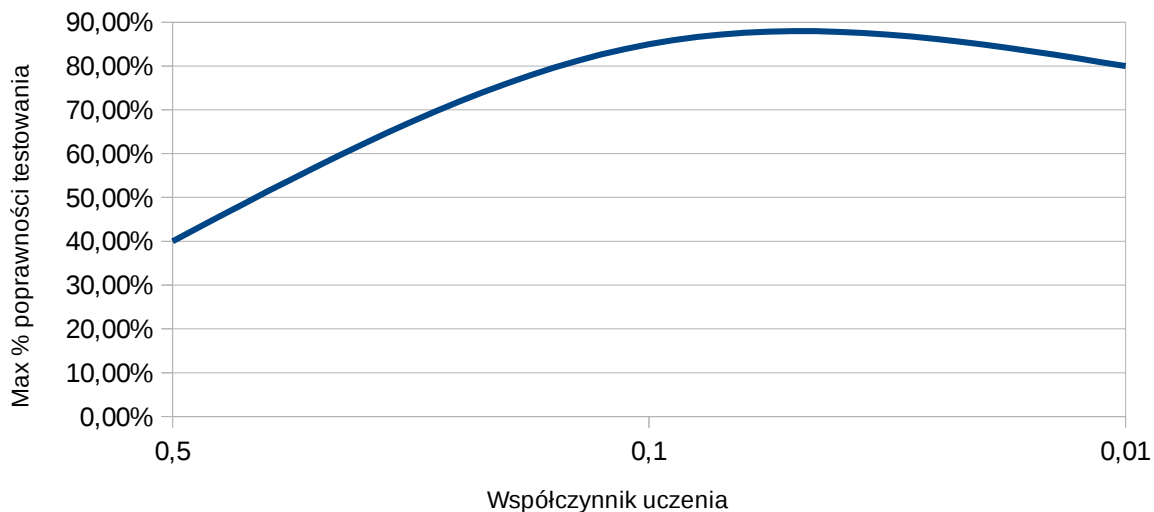
Promień sąsiedztwa		4,2	5,0	4,2	5,0	4,2	5,0
LP	Współczynnik uczenia	0,5		0,1		0,01	
1	% Poprawności testowania	25,00%	15,00%	85,00%	50,00%	80,00%	15,00%
	Liczba epok	7	2	6	1	42	166
2	% Poprawności testowania	25,00%	20,00%	75,00%	50,00%	50,00%	40,00%
	Liczba epok	2	2	4	1	1	160
3	% Poprawności testowania	30,00%	20,00%	75,00%	25,00%	60,00%	45,00%
	Liczba epok	5	2	12	1	1	165
4	% Poprawności testowania	40,00%	35,00%	80,00%	35,00%	45,00%	40,00%
	Liczba epok	1	2	3	1	1	162
5	% Poprawności testowania	20,00%	35,00%	70,00%	35,00%	50,00%	30,00%
	Liczba epok	10	2	7	1	1	1
6	% Poprawności testowania	30,00%	45,00%	80,00%	35,00%	40,00%	45,00%
	Liczba epok	6	1	4	15	1	157
7	% Poprawności testowania	15,00%	40,00%	60,00%	35,00%	40,00%	30,00%
	Liczba epok	3	2	10	16	1	168
8	% Poprawności testowania	40,00%	35,00%	65,00%	50,00%	80,00%	35,00%
	Liczba epok	7	2	2	16	38	1
9	% Poprawności testowania	30,00%	20,00%	65,00%	50,00%	45,00%	40,00%
	Liczba epok	6	2	11	35	1	1
10	% Poprawności testowania	35,00%	20,00%	80,00%	30,00%	40,00%	30,00%
	Liczba epok	8	2	2	1	1	158

Współczynnik uczenia	0,5		0,1		0,01	
Min % poprawności	15,00%	15,00%	60,00%	25,00%	40,00%	15,00%
Max % poprawności	40,00%	45,00%	85,00%	50,00%	80,00%	45,00%
Średni % poprawności	29,00%	28,50%	73,50%	39,50%	53,00%	35,00%
Min liczba epok	1	1	2	1	1	1
Max liczba epok	10	2	12	35	42	168
Średnia liczba epok	5,5	1,9	6,1	8,8	8,8	113,9

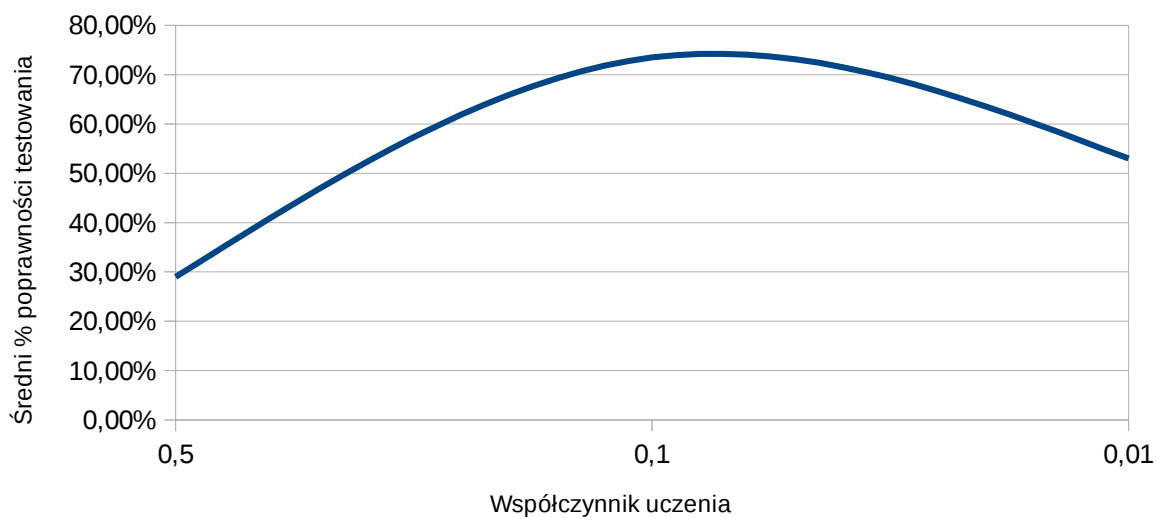
Wykres zależności min % poprawności testowania w zależności od współczynnika uczenia przy promieniu sąsiedztwa = 4,2



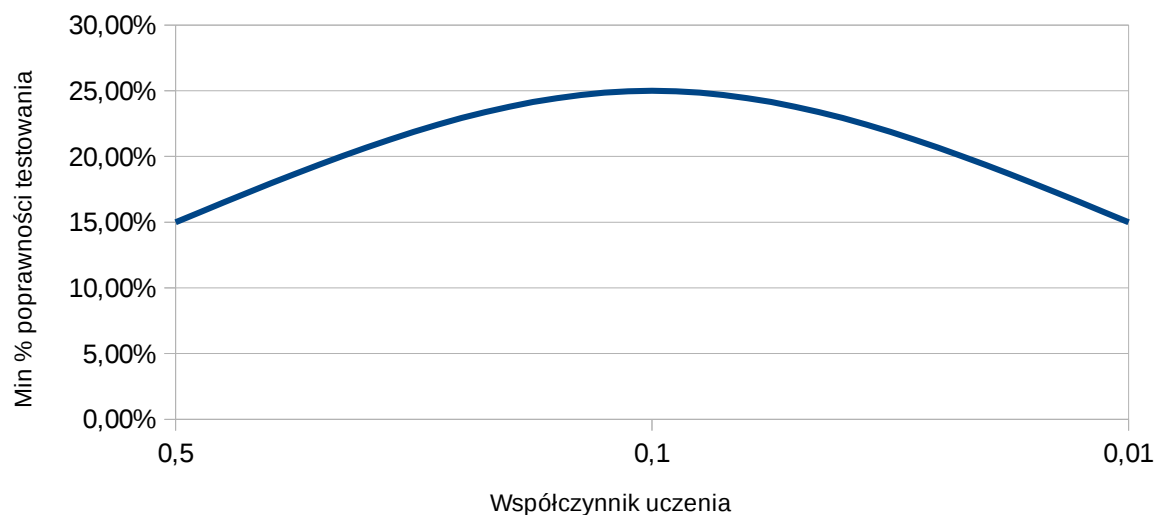
Wykres zależności max % poprawności testowania w zależności od współczynnika uczenia przy promieniu sąsiedztwa = 4,2



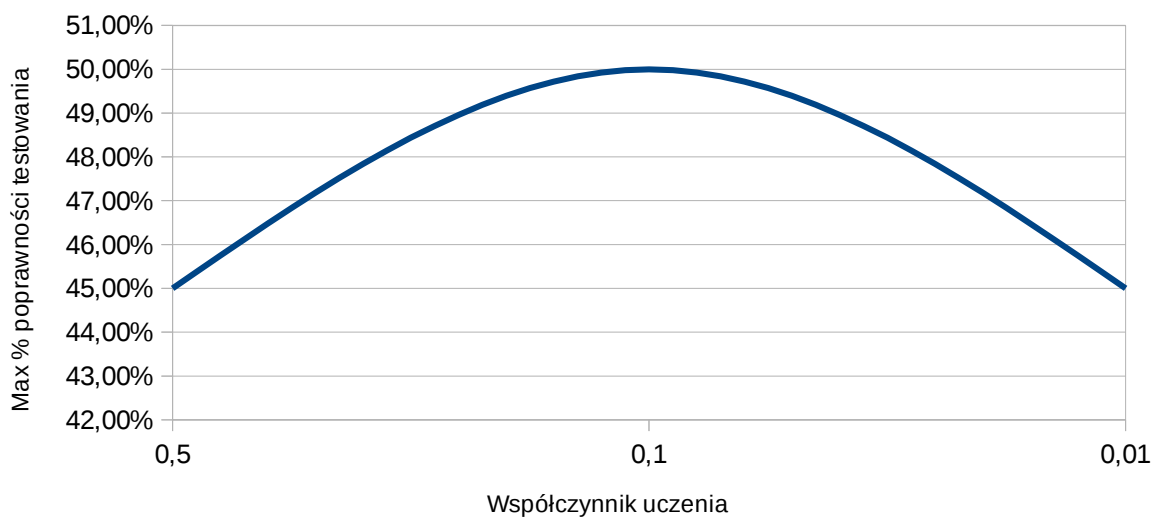
Wykres zależności średniej % poprawności testowania w zależności od współczynnika uczenia przy promieniu sąsiedztwa = 4,2



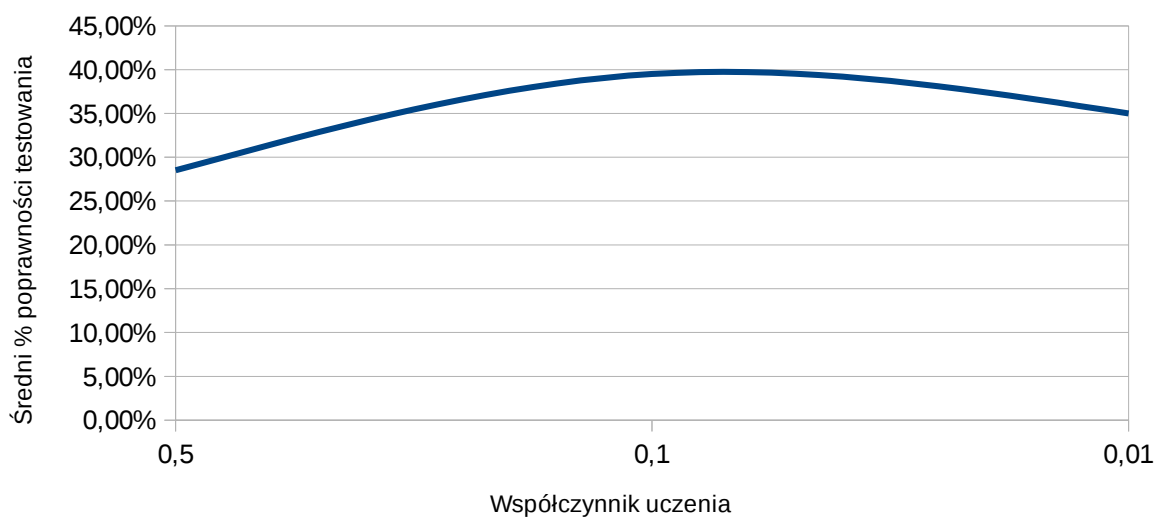
Wykres zależności min % poprawności testowania w zależności od współczynnika uczenia przy promieniu sąsiedztwa = 5,0



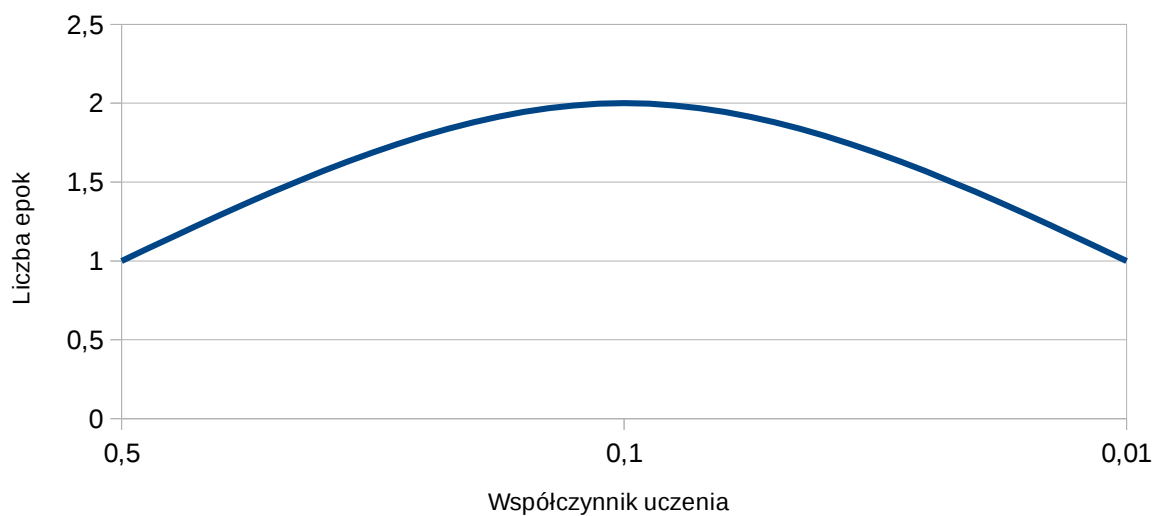
Wykres zależności max % poprawności testowania w zależności od współczynnika uczenia przy promieniu sąsiedztwa = 5,0



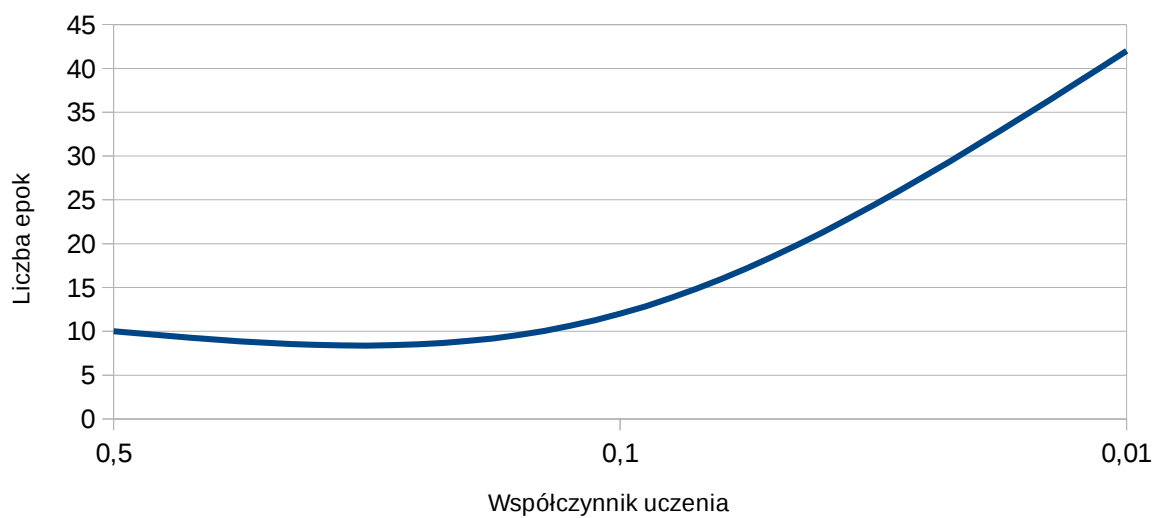
Wykres zależności średniej % poprawności testowania w zależności od współczynnika uczenia przy promieniu sąsiedztwa = 5,0



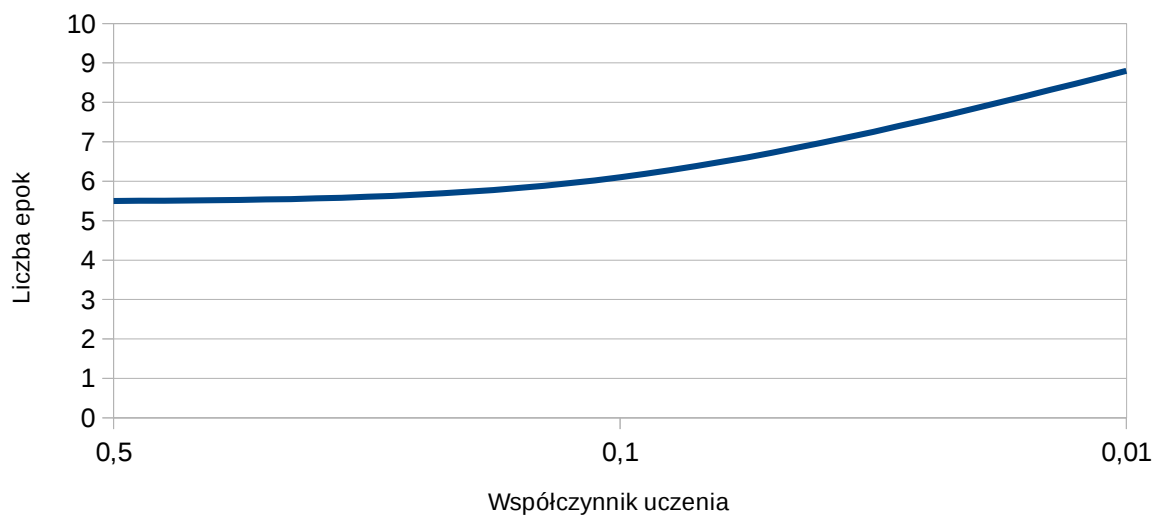
Wykres min liczby epok w zależności
od współczynnika uczenia przy promieniu sąsiedztwa = 4,2



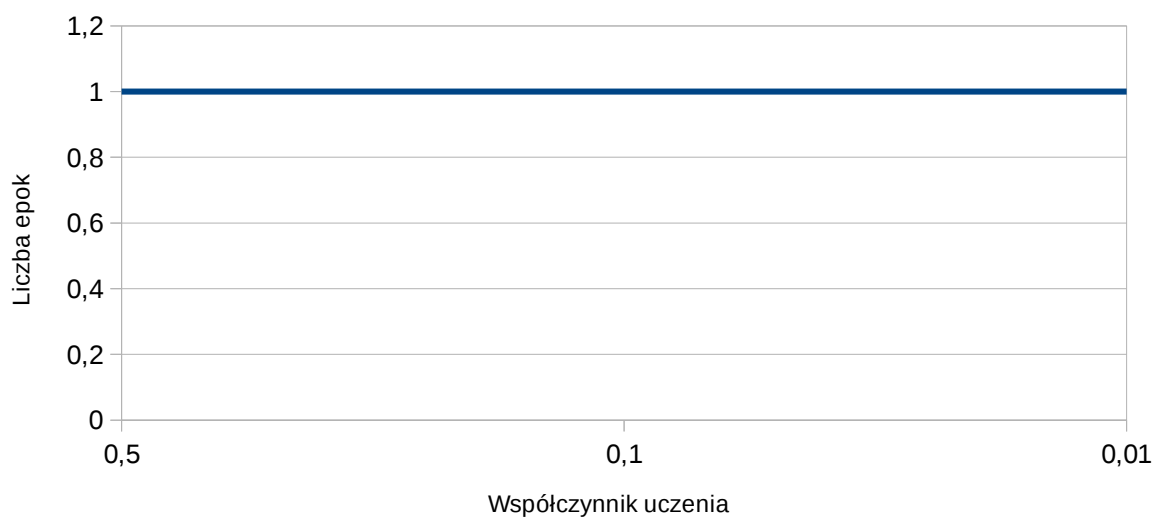
Wykres max liczby epok w zależności
od współczynnika uczenia przy promieniu sąsiedztwa = 4,2



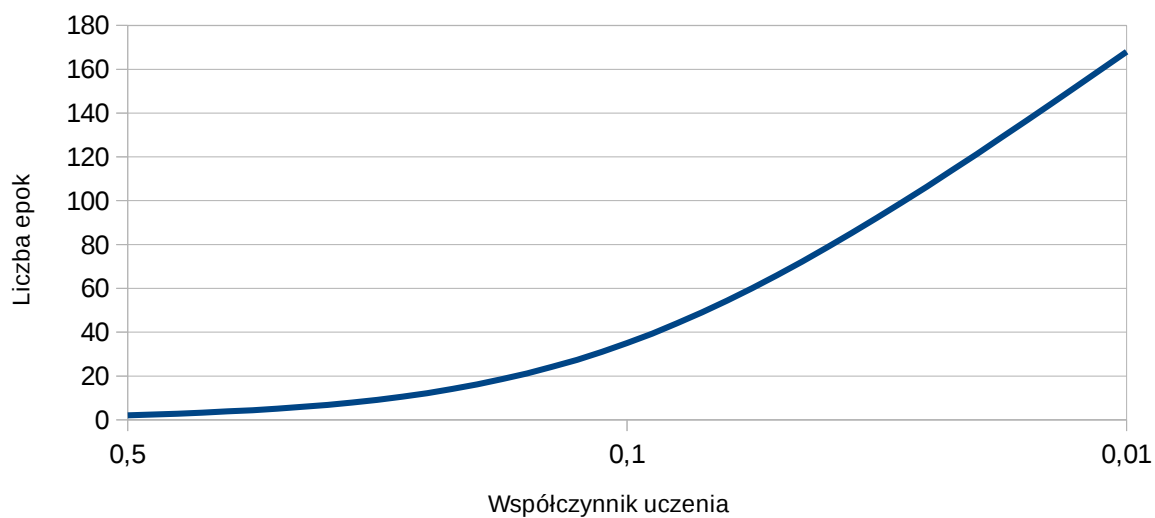
Wykres średniej liczby epok w zależności
od współczynnika uczenia przy promieniu sąsiedztwa = 4,2



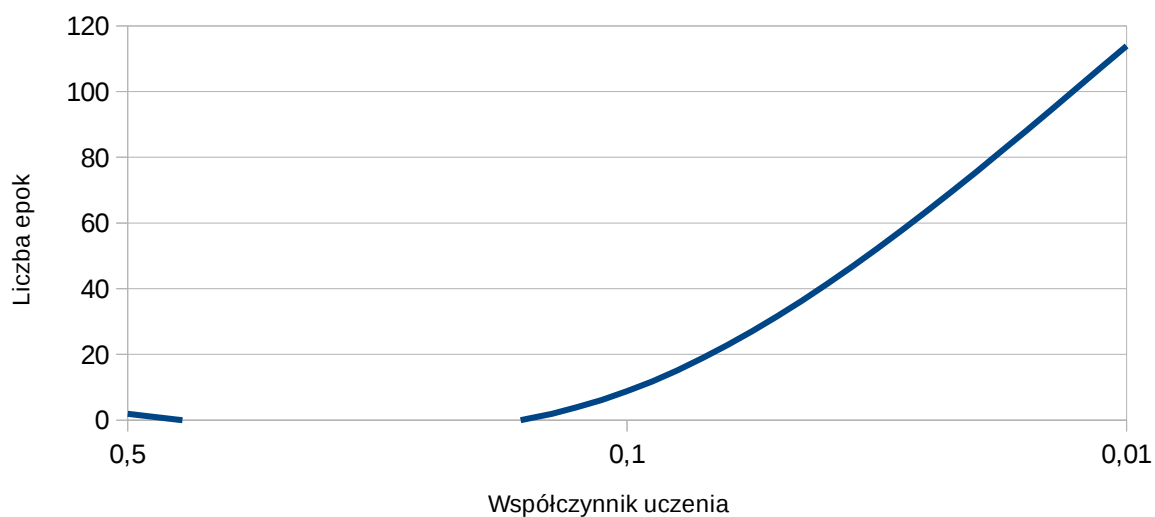
Wykres min liczby epok w zależności
od współczynnika uczenia przy promieniu sąsiedztwa = 5,0



Wykres max liczby epok w zależności
od współczynnika uczenia przy promieniu sąsiedztwa = 5,0



Wykres średni liczby epok w zależności
od współczynnika uczenia przy promieniu sąsiedztwa = 5,0



3) Analiza i dyskusja błędów uczenia i testowania oraz wyłonionych cech dla wyników opracowanej sieci w zależności od wartości współczynnika uczenia

Jak widać na powyższych wynikach, ilość epok jaka była potrzebna do nauczania sieci znacząco różniła się w poszczególnych przypadkach. Wyniki te różniły się o wiele rzędów wielkości. Tak więc jeśli chodzi o szybkość uczenia, widać, że nie jest to dobra miara ocenienia jakości uczenia sieci, gdyż jest to spowodowane wyłącznie początkowymi wartościami wag neuronów, a te są losowe. Jeśli jednak spojrzeć na wykresy, to widać, że zarówno jeśli chodzi o wartości maksymalne, minimalne czy też średnie, wraz ze wzrostem współczynnika uczenia, wartość liczby epok potrzebnych do nauczania sieci stale spada.

Jeśli chodzi o dobór współczynnika uczenia oraz promienia sąsiedztwa, to mają one jednak znaczenie. Widać, że najlepsze wyniki uzyskane zostały przy współczynniku uczenia równym 0.1. Szczególnie widoczne jest to na wykresach średniej poprawności testowania danych w zależności od współczynnika uczenia oraz promienia sąsiedztwa.

4) Sformułowanie wniosków

Na podstawie powyższych wyników można wnioskować, iż ostrożnie trzeba dobrać współczynnik uczenia oraz promień sąsiedztwa. Najlepsze wyniki zostały uzyskane przy współczynniku uczenia równym 0.1 oraz przy promieniu sąsiedztwa równym 4.2. Stało się to dlatego, że zbyt mały współczynnik uczenia powodował zbyt małe zmiany wektora wag a z kolei zbyt duży współczynnik uczenia powodował zbyt duże zmiany.

Kluczowy jednak okazał się sam promień sąsiedztwa. Wyniki testowania dla wartości równej 4.2 były o wiele lepsze niż dla wartości równej 5.0. Większy promień powodował, że zbyt wielka liczba sąsiednich neuronów podlegała modyfikacji. Dopiero odpowiednio niższy promień dał bardziej obiecujące wyniki – były one prawie dwa razy lepsze jeśli chodzi o wyniki testowania poprawności. Jednak trzeba pamiętać, że promień nie może być również zbyt mały.

W odniesieniu do powyższego, ważna również była sama ilość neuronów. Zbyt mała liczba mogłaby doprowadzić do tego, że tylko zbyt mała liczba neuronów byłaby zwycięzcą, a reszta neuronów pozostałaby martwa.

Jeśli chodzi o sam proces testowania, to widać, że dosyć często pojawiały się błędy. Było to spowodowane zbyt małą liczbą danych uczących. Jeśli uczyć by sieć na odpowiednio dużym zbiorze liter, również takich, które odbiegają trochę od normy, to wyniki byłyby duże lepsze.

5) Listing z komentarzami całego kodu programu

[illegible]

[illegible]

```

package main;

import java.util.Random;

public class KohonenWTM {

    private int noi;           //ilość wejść
    private double[] w;        //wagi

    public KohonenWTM ( 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 w zakresie od 0 do 1
    }

    //uczenie poprzez zmniejszenie odległości między wektorem wag a zadany wektorem
    public void learn ( double[] x, double lr ) {
        for ( int i = 0; i < noi; i++ )
            w[i] += lr * ( x[i] - w[i] );
    }

    //zwraca wektor wag
    public double[] getW () {
        return w;
    }
}

```

```

package main;

public class MainKohonenWTM {

    private static double learningRate = 0.1; //współczynnik uczenia się
    private static int numberOfInputs = 70; //ilość wejść
    private static int numberOfNeurons = 500; //liczba neuronów
    private static int numberOfLearnSamples = 20; //liczba danych uczących dla każdego kwiatu
    private static int numberOfTestSamples = 20; //liczba danych testujących dla każdego kwiatu
    private static int learnLimit = 50; //maksymalny próg epok uczenia
    private static double neighborhoodRadius = 5.0; //początkowa wartość promienia sąsiedztwa

    public static void main ( String[] args ) {

        KohonenWTM[] kohonens = new KohonenWTM[numberOfNeurons];
        for ( int i = 0; i < numberOfNeurons; i++ )
            kohonens[i] = new KohonenWTM( numberOfInputs );

        int ages = learn( kohonens );

        int[] winnerLearn = new int[numberOfLearnSamples], winnerTest = new int[numberOfTestSamples];
        int percent = 0;

        //wyniki uczenia
        for ( int i = 0; i < numberOfLearnSamples; i++ )
            winnerLearn[i] = getWinner( kohonens, Letters.lettersLearn[i] );

        //wyniki testowania
        for ( int i = 0; i < numberOfLearnSamples; i++ )
            winnerTest[i] = getWinner( kohonens, Letters.lettersTest[i] );

        for ( int i = 0; i < numberOfTestSamples; i++ )
            if ( winnerLearn[i] == winnerTest[i] )
                percent++;

        System.out.println( "Ilość epok = " + ages );
        System.out.println( "Poprawność testowania = " + ( ( percent * 100 ) / numberOfTestSamples ) + "%" );
    }

    //uczenie sieci
    private static int learn ( KohonenWTM[] kohonens ) {

        int counter = 0;
        int winner;
        int[] winners = new int[numberOfLearnSamples];
        for ( int i = 0; i < numberOfLearnSamples; i++ )
            winners[i] = - 1;
    }
}

```

```

while ( ! isUnique( winners ) ) { //dopóki sieć się nauczy
    //uczymy sieć po kolei każdy kwiat z każdego gatunku

    for ( int i = 0; i < numberOfLearnSamples; i++ ) {
        winner = getWinner( kohonens, Letters.lettersLearn[i] );
        kohonens[winner].learn( Letters.lettersLearn[i], learningRate );

        //uczenie sąsiednich neuronów
        for ( int j = 0; j < numberOfNeurons; j++ )
            if ( j != winner )
                if ( distanceBetweenVectors( kohonens[winner].getW(), kohonens[j].getW() ) <= neighborhoodRadius )
                    kohonens[j].learn( Letters.lettersLearn[i], learningRate );
    }

    //po zakończeniu epoki pobieramy zwycięzców
    for ( int i = 0; i < numberOfLearnSamples; i++ )
        winners[i] = getWinner( kohonens, Letters.lettersLearn[i] );

    //jeśli ilość prób nauczania osiągnie limit to uczenie uznajemy za nieudane i kończymy
    if ( ++ counter == learnLimit )
        break;
}
return counter;
}

//sprawdza czy sieć jest już nauczona
private static boolean isUnique ( int[] winners ) {

    //czy zwycięzca każdego z gatunków różni się od zwycięzców pozostałych gatunków
    for ( int i = 0; i < numberOfLearnSamples; i++ )
        for ( int j = i; j < numberOfLearnSamples; j++ )
            if ( i != j )
                if ( winners[i] == winners[j] )
                    return false;

    return true;
}

//zwraca zwycięzcę dla danego kwiatu
private static int getWinner ( KohonenWTM[] kohonens, double[] vector ) {

    int winner = 0;
    double minDistance = distanceBetweenVectors( kohonens[0].getW(), vector );

    //sprawdza który neuron jest zwycięzcą
    //miarą zwycięstwa jest odległość między wektorem wag neuronu a wektorem cech kwiatu
    for ( int i = 0; i < numberOfNeurons; i++ ) {
        if ( distanceBetweenVectors( kohonens[i].getW(), vector ) < minDistance ) {
            winner = i;
            minDistance = distanceBetweenVectors( kohonens[i].getW(), vector );
        }
    }

    return winner;
}

//zwraca odległość między zadanymi wektorami
private static double distanceBetweenVectors ( double[] vector1, double[] vector2 ) {

    double suma = 0.0;

    for ( int i = 0; i < vector1.length; i++ )
        suma += Math.abs( vector1[i] - vector2[i] ); //miara Manhattan

    return Math.sqrt( suma );
}
}

```

Bibliografia:

- S. Osowski – „Sieci neuronowe do przetwarzania informacji”