Scenariusz 5

Temat ćwiczenia: Budowa i działanie sieci Kohonena dla WTA
Wykonała: Burnat Magdalena, IS, gr. 3, 270652

1. Cel ćwiczenia

Celem ćwiczenia jest poznanie budowy i działania sieci Kohonena przy wykorzystaniu reguły WTA do odwzorowania istotnych cech kwiatków.

2. Opis budowy sieci i algorytmów uczenia.

Celem budowanej sieci jest podział wzorców uczących na klasy obrazów zbliżonych do siebie i przyporządkowanie każdej klasie osobnego elementu wyjściowego. Podział na grupy odbywa się w takich sposób, by elementy w tej samej grupie były do siebie podobne a jednocześnie jak najbardziej odmienne od elementów z pozostałych grup.

Tab. 1. Zestaw danych

Długość działki kielicha	Szerokość działki kielicha	Długość płatka	Szerokość płatka	*Rodzaj kwiatu*
5.8	4.0	1.2	0.2	*I. setosa*
5.7	4.4	1.5	0.4	*I. setosa*
5.7	3.8	1.7	0.3	*I. setosa*
5.5	4.2	1.4	0.2	*I. setosa*
5.5	3.5	1.3	0.2	*I. setosa*
5.4	3.9	1.7	0.4	*I. setosa*
5.4	3.7	1.5	0.2	*I. setosa*
5.4	3.9	1.3	0.4	*I. setosa*
5.4	3.4	1.7	0.2	*I. setosa*
5.4	3.4	1.5	0.4	*I. setosa*
5.3	3.7	1.5	0.2	*I. setosa*
5.2	3.5	1.4	0.2	*I. setosa*
5.2	3.5	1.5	0.2	*I. setosa*
5.2	3.4	1.4	0.2	*I. setosa*
5.2	4.1	1.5	0.1	*I. setosa*
5.1	3.5	1.4	0.3	*I. setosa*
5.1	3.8	1.5	0.3	*I. setosa*
5.1	3.7	1.5	0.4	*I. setosa*
5.1	3.3	1.7	0.5	*I. setosa*
5.1	3.4	1.5	0.2	*I. setosa*
5.1	3.8	1.9	0.4	*I. setosa*
5.1	3.8	1.6	0.2	*I. setosa*
5.0	3.6	1.4	0.3	*I. setosa*
5.0	3.4	1.5	0.2	*I. setosa*
5.0	3.0	1.6	0.2	*I. setosa*
5.0	3.4	1.6	0.4	*I. setosa*
5.0	3.2	1.2	0.2	*I. setosa*
5.0	3.5	1.3	0.3	*I. setosa*
5.0	3.5	1.6	0.6	*I. setosa*
5.0	3.3	1.4	0.2	*I. setosa*
4.9	3.0	1.4	0.2	*I. setosa*
4.9	3.1	1.5	0.1	*I. setosa*
4.9	3.1	1.5	0.2	*I. setosa*
4.9	3.6	1.4	0.1	*I. setosa*

4.8	3.4	1.6	0.2	*I. setosa*
4.8	3.0	1.4	0.1	*I. setosa*
4.8	3.4	1.9	0.2	*I. setosa*
4.8	3.1	1.6	0.2	*I. setosa*
4.8	3.0	1.4	0.3	*I. setosa*
4.7	3.2	1.3	0.2	*I. setosa*
4.7	3.2	1.6	0.2	*I. setosa*
4.6	3.1	1.5	0.2	*I. setosa*
4.6	3.4	1.4	0.3	*I. setosa*
4.6	3.6	1.0	0.2	*I. setosa*
4.6	3.2	1.4	0.2	*I. setosa*
4.5	2.3	1.3	0.3	*I. setosa*
4.4	2.9	1.4	0.2	*I. setosa*
4.4	3.0	1.3	0.2	*I. setosa*
4.4	3.2	1.3	0.2	*I. setosa*
4.3	3.0	1.1	0.1	*I. setosa*
7.0	3.2	4.7	1.4	*I. versicolor*
6.9	3.1	4.9	1.5	*I. versicolor*
6.8	2.8	4.8	1.4	*I. versicolor*
6.7	3.1	4.4	1.4	*I. versicolor*
6.7	3.0	5.0	1.7	*I. versicolor*
6.7	3.1	4.7	1.5	*I. versicolor*
6.6	2.9	4.6	1.3	*I. versicolor*
6.6	3.0	4.4	1.4	*I. versicolor*
6.5	2.8	4.6	1.5	*I. versicolor*
6.4	3.2	4.5	1.5	*I. versicolor*
6.4	2.9	4.3	1.3	*I. versicolor*
6.3	3.3	4.7	1.6	*I. versicolor*
6.3	2.5	4.9	1.5	*I. versicolor*
6.3	2.3	4.4	1.3	*I. versicolor*
6.2	2.2	4.5	1.5	*I. versicolor*
6.2	2.9	4.3	1.3	*I. versicolor*
6.1	2.9	4.7	1.4	*I. versicolor*
6.1	2.8	4.0	1.3	*I. versicolor*
6.1	2.8	4.7	1.2	*I. versicolor*
6.1	3.0	4.6	1.4	*I. versicolor*
6.0	2.2	4.0	1.0	*I. versicolor*
6.0	2.9	4.5	1.5	*I. versicolor*
6.0	2.7	5.1	1.6	*I. versicolor*
6.0	3.4	4.5	1.6	*I. versicolor*
5.9	3.0	4.2	1.5	*I. versicolor*
5.9	3.2	4.8	1.8	*I. versicolor*
5.8	2.7	4.1	1.0	*I. versicolor*
5.8	2.7	3.9	1.2	*I. versicolor*
5.8	2.6	4.0	1.2	*I. versicolor*
5.7	2.8	4.5	1.3	*I. versicolor*
5.7	2.6	3.5	1.0	*I. versicolor*
5.7	3.0	4.2	1.2	*I. versicolor*
5.7	2.9	4.2	1.3	*I. versicolor*
5.7	2.8	4.1	1.3	*I. versicolor*
5.6	2.9	3.6	1.3	*I. versicolor*
5.6	3.0	4.5	1.5	*I. versicolor*
5.6	2.5	3.9	1.1	*I. versicolor*
5.6	3.0	4.1	1.3	*I. versicolor*
5.6	2.7	4.2	1.3	*I. versicolor*
5.5	2.3	4.0	1.3	*I. versicolor*
5.5	2.4	3.8	1.1	*I. versicolor*
5.5	2.4	3.7	1.0	*I. versicolor*

5.5	2.5	4.0	1.3	*I. versicolor*
5.5	2.6	4.4	1.2	*I. versicolor*
5.4	3.0	4.5	1.5	*I. versicolor*
5.2	2.7	3.9	1.4	*I. versicolor*
5.1	2.5	3.0	1.1	*I. versicolor*
5.0	2.0	3.5	1.0	*I. versicolor*
5.0	2.3	3.3	1.0	*I. versicolor*
4.9	2.4	3.3	1.0	*I. versicolor*
7.9	3.8	6.4	2.0	*I. virginica*
7.7	3.8	6.7	2.2	*I. virginica*
7.7	2.6	6.9	2.3	*I. virginica*
7.7	2.8	6.7	2.0	*I. virginica*
7.6	3.0	6.6	2.1	*I. virginica*
7.4	2.8	6.1	1.9	*I. virginica*
7.3	2.9	6.3	1.8	*I. virginica*
7.2	3.6	6.1	2.5	*I. virginica*
7.2	3.2	6.0	1.8	*I. virginica*
7.2	3.0	5.8	1.6	*I. virginica*
7.1	3.0	5.9	2.1	*I. virginica*
6.9	3.2	5.7	2.3	*I. virginica*
6.9	3.1	5.4	2.1	*I. virginica*
6.9	3.1	5.1	2.3	*I. virginica*
6.8	3.0	5.5	2.1	*I. virginica*
6.8	3.2	5.9	2.3	*I. virginica*
6.7	2.5	5.8	1.8	*I. virginica*
6.7	3.3	5.7	2.1	*I. virginica*
6.7	3.1	5.6	2.4	*I. virginica*
6.7	3.3	5.7	2.5	*I. virginica*
6.7	3.0	5.2	2.3	*I. virginica*
6.5	3.0	5.8	2.2	*I. virginica*
6.5	3.2	5.1	2.0	*I. virginica*
6.5	3.0	5.5	1.8	*I. virginica*
6.5	3.0	5.2	2.0	*I. virginica*
6.4	2.7	5.3	1.9	*I. virginica*
6.4	3.2	5.3	2.3	*I. virginica*
6.4	2.8	5.6	2.1	*I. virginica*
6.4	2.8	5.6	2.2	*I. virginica*
6.4	3.1	5.5	1.8	*I. virginica*
6.3	3.3	6.0	2.5	*I. virginica*
6.3	2.9	5.6	1.8	*I. virginica*
6.3	2.7	4.9	1.8	*I. virginica*
6.3	2.8	5.1	1.5	*I. virginica*
6.3	3.4	5.6	2.4	*I. virginica*
6.3	2.5	5.0	1.9	*I. virginica*
6.2	2.8	4.8	1.8	*I. virginica*
6.2	3.4	5.4	2.3	*I. virginica*
6.1	3.0	4.9	1.8	*I. virginica*
6.1	2.6	5.6	1.4	*I. virginica*
6.0	2.2	5.0	1.5	*I. virginica*
6.0	3.0	4.8	1.8	*I. virginica*
5.9	3.0	5.1	1.8	*I. virginica*
5.8	2.7	5.1	1.9	*I. virginica*
5.8	2.8	5.1	2.4	*I. virginica*
5.8	2.7	5.1	1.9	*I. virginica*
5.7	2.5	5.0	2.0	*I. virginica*
5.6	2.8	4.9	2.0	*I. virginica*
4.9	2.5	4.5	1.7	*I. virginica*

Proces uczenia odbywa się przy pomocy uczenia rywalizującego, które jest metodą uczenia sieci samoorganizujących. Podczas uczenia neurony uczą się rozpoznawać dane i zbliżają się odpowiednio do obszarów zajmowanych przez te dane. Po wejściu każdego wektora uczącego wybierany jest tylko jeden neuron, najbliższy prezentowanemu wzorcowi. Neurony rywalizują między sobą i zwycięża ten neuron, którego wartość jest największa. Zwycięski neuron przyjmuje na wyjściu wartość 1, a pozostałe neurony 0. Jest to uczenie bez nauczyciela.

Neuron zwycięski ma prawo uaktualnić swoje wagi wg jednej z dwóch zasad: WTA(Winner Takes All), WTM(Winner Takes Most). W tym projekcie została wykorzystana pierwsza zasada.

Stworzona sieć to sieć jednowarstwowa składająca się z 7 neuronów.

Proces uczenia przebiega według następującego schematu w danej epoce uczenia:
1. Wybór η za zakresu 0 do 1
2. Normalizacja wektorów danych wejściowych
3. Wybór początkowych wartości wag, jako niewielkich liczb losowych z zakresu 0 do 1
4. Dla danego wektora uczącego obliczmy odpowiedź sieci. A więc dla pojedynczego neuronu obliczana jest suma ilorazów sygnałów wejściowych i wag - u^k_i.
5. Wybierany jest neuron, dla którego obliczona suma jest największa i zaktualizowanie jego wag według wzoru:

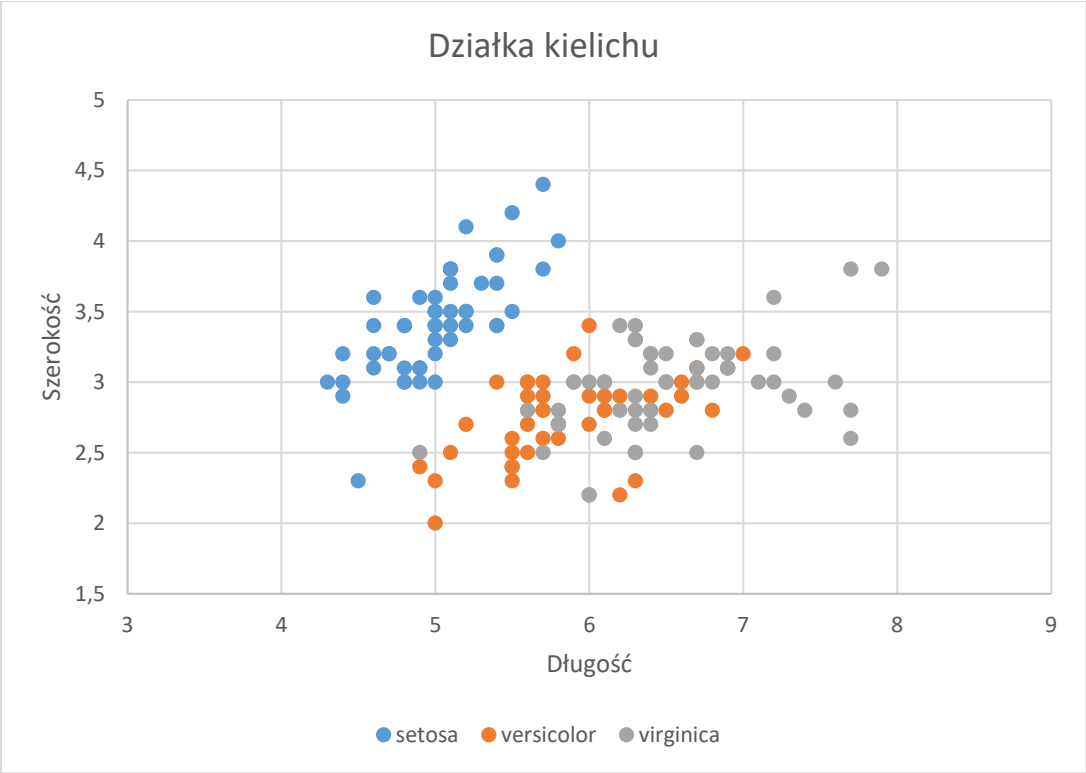
$$w_{i,j}(t+1) \;=\; w_{ij}(t) + \eta \cdot (x_i \cdot w_{ij}(t))$$

η to współczynnik uczenia wybierany z zakresu od 0 do 1
6. Znormalizowanie wartości nowego wektora wag.
7. Zwycięski neuron daje odpowiedź na swoim wyjściu 1, a pozostałe 0.
8. Wczytanie kolejnego wektora uczącego

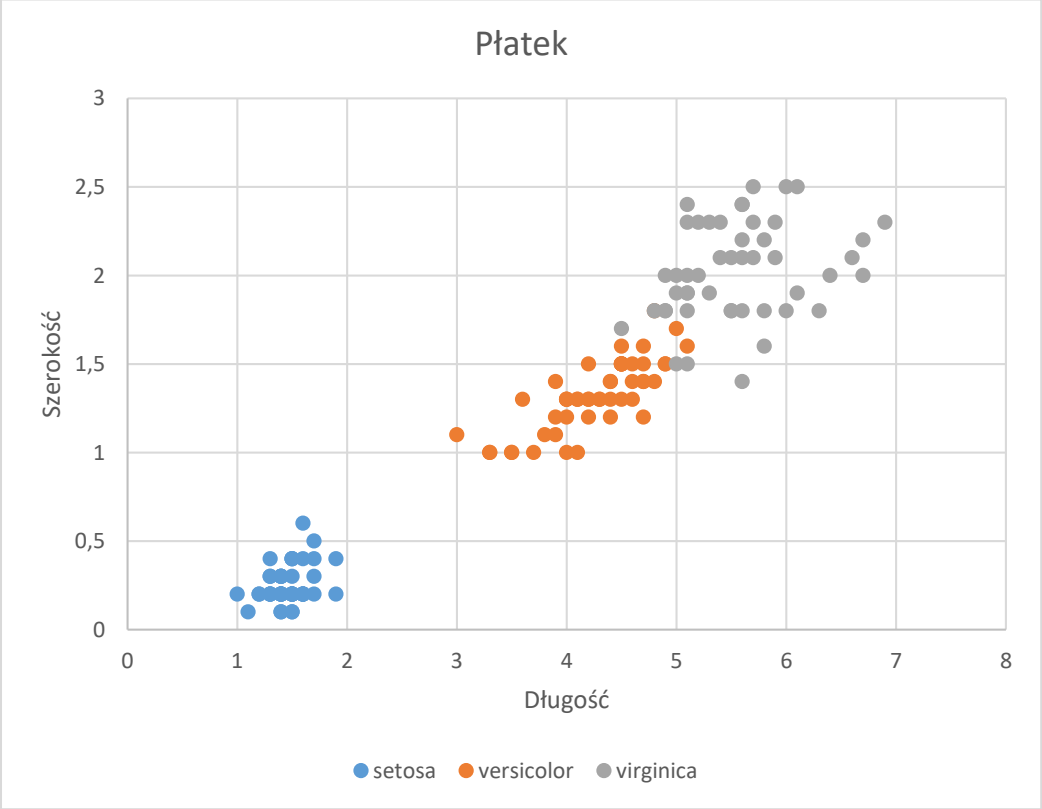
3. Otrzymane wyniki

Zestaw danych składa się ze 150 wektorów dla każdego rodzaju kwiatka po 50. Dane te zostały podzielone na 70% danych uczących i 30% danych testujących. Wykonano trzy rożne badania:

a) Z wykorzystanie cech dotyczących tylko działki kielicha (długość i szerokość)
b) Z wykorzystanie cech dotyczących tylko płatka (długość i szerokość)
c) Z wykorzystaniem wszystkich cech kwiatów



Wykres 1. Przedstawienie rozkładu cech działki kielichu poszczególnych próbek danych kwiatów.



Tab. 2.Wyniki testowania według cech działki kielicha.

Tab. 3.Wyniki testowania według cech płatka

Długość działki kielicha	Szerokość działki kielicha	*Rodzaj kwiatu*	*Odpowiedź sieci*
5,8	4	*I. setosa*	*1000000*
5,7	4,4	*I. setosa*	*1000000*
5,7	3,8	*I. setosa*	*1000000*
5,5	4,2	*I. setosa*	*1000000*
5,5	3,5	*I. setosa*	*1000000*
5,4	3,9	*I. setosa*	*1000000*
5,4	3,7	*I. setosa*	*1000000*
5,4	3,9	*I. setosa*	*1000000*
5,4	3,4	*I. setosa*	*1000000*
5,4	3,4	*I. setosa*	*1000000*
5,3	3,7	*I. setosa*	*1000000*
5,2	3,5	*I. setosa*	*1000000*
5,2	3,5	*I. setosa*	*1000000*
5,2	3,4	*I. setosa*	*1000000*
5,2	4,1	*I. setosa*	*1000000*
7	3,2	*I. versicolor*	*0100000*
6,9	3,1	*I. versicolor*	*0100000*
6,8	2,8	*I. versicolor*	*0100000*
6,7	3,1	*I. versicolor*	*0100000*
6,7	3	*I. versicolor*	*0100000*
6,7	3,1	*I. versicolor*	*0100000*
6,6	2,9	*I. versicolor*	*0100000*
6,6	3	*I. versicolor*	*0100000*
6,5	2,8	*I. versicolor*	*0100000*
6,4	3,2	*I. versicolor*	*0100000*
6,4	2,9	*I. versicolor*	*0100000*
6,3	3,3	*I. versicolor*	*0100000*
6,3	2,5	*I. versicolor*	*0100000*
6,3	2,3	*I. versicolor*	*0100000*
6,2	2,2	*I. versicolor*	*0100000*
6,4	2,8	*I. virginica*	*0100000*
6,4	2,8	*I. virginica*	*0100000*
6,4	3,1	*I. virginica*	*0100000*
6,3	3,3	*I. virginica*	*0100000*
6,3	2,9	*I. virginica*	*0100000*
6,3	2,7	*I. virginica*	*0100000*
6,3	2,8	*I. virginica*	*0100000*
6,3	3,4	*I. virginica*	*0100000*
6,3	2,5	*I. virginica*	*0100000*
6,2	2,8	*I. virginica*	*0100000*
6,2	3,4	*I. virginica*	*0100000*
6,1	3	*I. virginica*	*0100000*
6,1	2,6	*I. virginica*	*0100000*

Długość płatka	Szerokość płatka	*Rodzaj kwiatu*	*Odpowiedź sieci*
1,2	0,2	*I. setosa*	*1000000*
1,5	0,4	*I. setosa*	*1000000*
1,7	0,3	*I. setosa*	*1000000*
1,4	0,2	*I. setosa*	*1000000*
1,3	0,2	*I. setosa*	*1000000*
1,7	0,4	*I. setosa*	*1000000*
1,5	0,2	*I. setosa*	*1000000*
1,3	0,4	*I. setosa*	*1000000*
1,7	0,2	*I. setosa*	*1000000*
1,5	0,4	*I. setosa*	*1000000*
1,5	0,2	*I. setosa*	*1000000*
1,4	0,2	*I. setosa*	*1000000*
1,5	0,2	*I. setosa*	*1000000*
1,4	0,2	*I. setosa*	*1000000*
1,5	0,1	*I. setosa*	*1000000*
4,7	1,4	*I. versicolor*	*0100000*
4,9	1,5	*I. versicolor*	*0100000*
4,8	1,4	*I. versicolor*	*0100000*
4,4	1,4	*I. versicolor*	*0100000*
5	1,7	*I. versicolor*	*0100000*
4,7	1,5	*I. versicolor*	*0100000*
4,6	1,3	*I. versicolor*	*0100000*
4,4	1,4	*I. versicolor*	*0100000*
4,6	1,5	*I. versicolor*	*0100000*
4,5	1,5	*I. versicolor*	*0100000*
4,3	1,3	*I. versicolor*	*0100000*
4,7	1,6	*I. versicolor*	*0100000*
4,9	1,5	*I. versicolor*	*0100000*
4,4	1,3	*I. versicolor*	*0100000*
4,5	1,5	*I. versicolor*	*0100000*
5,6	2,1	*I. virginica*	*0010000*
5,6	2,2	*I. virginica*	*0010000*
5,5	1,8	*I. virginica*	*0010000*
6	2,5	*I. virginica*	*0010000*
5,6	1,8	*I. virginica*	*0010000*
4,9	1,8	*I. virginica*	*0010000*
5,1	1,5	*I. virginica*	*0010000*
5,6	2,4	*I. virginica*	*0010000*
5	1,9	*I. virginica*	*0010000*
4,8	1,8	*I. virginica*	*0010000*
5,4	2,3	*I. virginica*	*0010000*
4,9	1,8	*I. virginica*	*0010000*
5,6	1,4	*I. virginica*	*0010000*

6	2,2	*I. virginica*	*0100000*	5	1,5	*I. virginica*	*0010000*
6	3	*I. virginica*	*0100000*	4,8	1,8	*I. virginica*	*0010000*

Tab. 4. Wyniki testowania wszystkich cech kwiatów.

Długość działki kielicha	Szerokość działki kielicha	Długość płatka	Szerokość płatka	*Rodzaj kwiatu*	*Odpowiedź sieci*
5,8	4	1,2	0,2	*I. setosa*	*1000000*
5,7	4,4	1,5	0,4	*I. setosa*	*1000000*
5,7	3,8	1,7	0,3	*I. setosa*	*1000000*
5,5	4,2	1,4	0,2	*I. setosa*	*1000000*
5,5	3,5	1,3	0,2	*I. setosa*	*1000000*
5,4	3,9	1,7	0,4	*I. setosa*	*1000000*
5,4	3,7	1,5	0,2	*I. setosa*	*1000000*
5,4	3,9	1,3	0,4	*I. setosa*	*1000000*
5,4	3,4	1,7	0,2	*I. setosa*	*1000000*
5,4	3,4	1,5	0,4	*I. setosa*	*1000000*
5,3	3,7	1,5	0,2	*I. setosa*	*1000000*
5,2	3,5	1,4	0,2	*I. setosa*	*1000000*
5,2	3,5	1,5	0,2	*I. setosa*	*1000000*
5,2	3,4	1,4	0,2	*I. setosa*	*1000000*
5,2	4,1	1,5	0,1	*I. setosa*	*1000000*
7	3,2	4,7	1,4	*I. versicolor*	*0100000*
6,9	3,1	4,9	1,5	*I. versicolor*	*0100000*
6,8	2,8	4,8	1,4	*I. versicolor*	*0100000*
6,7	3,1	4,4	1,4	*I. versicolor*	*0100000*
6,7	3	5	1,7	*I. versicolor*	*0100000*
6,7	3,1	4,7	1,5	*I. versicolor*	*0100000*
6,6	2,9	4,6	1,3	*I. versicolor*	*0100000*
6,6	3	4,4	1,4	*I. versicolor*	*0100000*
6,5	2,8	4,6	1,5	*I. versicolor*	*0100000*
6,4	3,2	4,5	1,5	*I. versicolor*	*0100000*
6,4	2,9	4,3	1,3	*I. versicolor*	*0100000*
6,3	3,3	4,7	1,6	*I. versicolor*	*0100000*
6,3	2,5	4,9	1,5	*I. versicolor*	*0100000*
6,3	2,3	4,4	1,3	*I. versicolor*	*0100000*
6,2	2,2	4,5	1,5	*I. versicolor*	*0100000*
6,4	2,8	5,6	2,1	*I. virginica*	*0100000*
6,4	2,8	5,6	2,2	*I. virginica*	*0100000*

6,4	3,1	5,5	1,8	*I. virginica*	*0100000*
6,3	3,3	6	2,5	*I. virginica*	*0100000*
6,3	2,9	5,6	1,8	*I. virginica*	*0100000*
6,3	2,7	4,9	1,8	*I. virginica*	*0100000*
6,3	2,8	5,1	1,5	*I. virginica*	*0100000*
6,3	3,4	5,6	2,4	*I. virginica*	*0100000*
6,3	2,5	5	1,9	*I. virginica*	*0100000*
6,2	2,8	4,8	1,8	*I. virginica*	*0100000*
6,2	3,4	5,4	2,3	*I. virginica*	*0100000*
6,1	3	4,9	1,8	*I. virginica*	*0100000*
6,1	2,6	5,6	1,4	*I. virginica*	*0100000*
6	2,2	5	1,5	*I. virginica*	*0100000*
6	3	4,8	1,8	*I. virginica*	*0100000*

Tab.5. Wyniki testowania wpływu współczynnika uczenia na proces

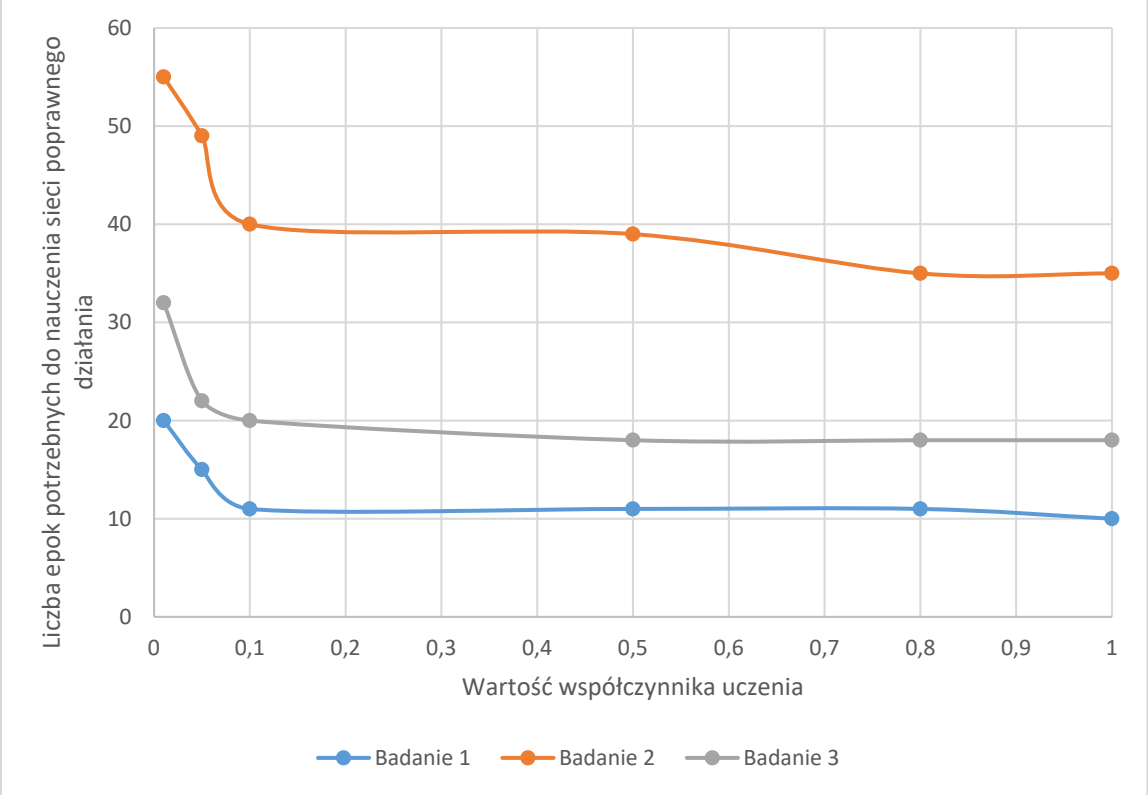
Wartość współczynnika uczenia	Liczba epok potrzebnych do nauczenia sieci w poszczególnych rodzajach badań		
	Badanie ze względu na cechu działki kielicha	Badanie ze względu na cechy płatka	Badanie ze względu na wszystkie cechy
0,01	20	55	32
0,05	15	49	22
0,1	11	40	20
0,5	11	39	18
0,8	11	35	18
1	10	35	18

4. Analiza wyników

Analizując wyniki otrzymane podczas pierwszego z badań dotyczącego cech działki kielicha (tab.2.) widzimy, że sieć daję odpowiedź pierwszym albo drugim neuronem. W więc dane zostały podzielone na dwa klastry. Kwiat rodzaju l. setosa do jednego. Natomiast l. virginica oraz l. versicolor razem do drugiego. Porównując to z wykresem nr 1 jest to jak najbardziej pożądany podział. Po rozkładzie punktów poszczególnych kwiatów widać, że pierwszy różni się od pozostałych dwóch. Natomiast między kolejnymi dwoma różnica nie jest tak zauważalna, dlatego też po procesie nauczania zostały przydzielone do tego samego klastra.

W kolejnym badaniu były brane tylko pod uwagę cechy dotyczące płatka kwiatu. Analizując wyniki umieszczone w tab.3. zauważono, że sieć odpowiada pierwszym, drugim, albo trzecim neuronem. Oznacza to, że badane cechy w każdym z trzech rodzaju kwiatu były na tyle różne, że sieć podzieliła je na trzy różne klastry. Ten wynik jest również zgodny z wykresem prezentującym dane wejściowe (wykres 2). Widoczna na nim jest bardzo duża różnica pierwszego kwiatu od pozostałych dwóch, oraz dość zauważalna różnica pomiędzy kolejnymi dwoma, mimo że leżą dość blisko siebie.

W ostatnim badaniu wzięto pod uwagę wszystkie cechy na raz i na tej podstawie sieć miała zadecydować o ilości i poszczególnym podziale na klastry. W wynikach zamieszczonych w tabeli nr 4 widzimy, że sieć uznała, że cechy kwiatów rodzaju I. virginica oraz I. versicolor są na tyle podobne, że można je umieścić w wspólnym klastrze, odpowiadając na nie drugim neuronem. Granica między nimi jest ulega zatarciu. Natomiast w cechy kwiatów rodzaju I. setosa były na tyle odmienne, że zostały przydzielone od osobnego klastra, na który odpowiadał neuron pierwszy.



Wykres 3. Zależność liczby epok potrzebnych do nauczenia neuronu od wartości współczynnika uczenia

Na podstawie powyższego wykresu stwierdzono, że liczba epok potrzebnych do nauczenia poprawnego działania sieci spada wraz z wzrostem wartości współczynnika uczenia. Po osiągnięciu pewnego poziomu danego współczynnika dalsze zmniejszanie nie daje już większego pożytku i liczba epok utrzymuje się na stałym poziomie. W przypadku badania, w którym następował podział na większą liczbę klastrów zakres zmian liczby epok jest znacznie

wyżej położony niż w pozostałych dwóch badaniach, gdzie następował podział tylko na dwa klastry.

5. Wnioski

- Sieć Kohonena jest siecią samoorganizującą pozwalającą podzielić dane posiadające różne wartości poszczególnych cech na odpowiednie klastry nie wymagając przy tym wartości oczekiwanych podczas procesu uczenia (nauczanie bez nauczyciela).
- Sieć po odpowiednim procesie uczenia daje 100% poprawnych odpowiedzi w późniejszej fazie testowania. Każda wektor wejściowy dał odpowiedź, która była zgodna z oczekiwaną. A więc ten rodzaj sieci sprawdziłby się doskonale przy podziale i wykrywaniu wspólnych cech przy bardzo dużej ilości różnych obiektów o których wiedza jest bardzo ograniczona.
- Procesu uczenia zależy od współczynnika uczenia. Wraz z jego wzrostem proces uczenia jest szybszy. Jest to wytłumaczalne z tego względu, że im ta wartość jest większa tym przyrost wag, które na samym początku są niewielkie jest szybszy, a więc i proces uczenia przebiega szybciej. Jednak po pewnej jego wartości następuje stabilizacja i już nie zauważa się dalszych wymiernych korzyści z stosowania dużych wartości współczynnika
- Porównując z wcześniej wykonanymi projektami w tym przypadku potrzebna jest dużo mniejsza liczba epoko potrzebnych do nauczenia poprawnego działania sieci. Dzieje się tak, ponieważ wprowadzamy wiele próbek tego samego rodzaju kwiatu, którego wartości poszczególnych cech różnią się między sobą w bardzo niewielkim stopniu, a więc niejako możemy uznać to za kolejną epokę.
- W przypadku zastosowaniu iloczynu skalarnego bez wcześniejszej normalizacji wektorów danych uczących sieć nie działa poprawnie, ponieważ dochodzi do niespójnego podziału przestrzeni cech.

6. Kod programu

main.cpp

```
#include <iostream>
#include <vector>
#include "Warstwa.h"
#include <time.h>
#include <fstream>
using namespace std;

void ustawDaneWejsciowe(Neuron& neuron, vector< vector<double> > inputData, int numberOfEntrances, int
inputDataRow);
void ucz(Warstwa& layer, vector< vector<double> > inputData);
void testuj(Warstwa& layer, vector< vector<double> > inputData);
void wczytajDaneUczace(vector< vector<double> > &learningInputData, int numberOfEntrances, int
amountOfOutputs);
void wczytajDaneTestujace(vector< vector<double> > &testingInputData, int numberOfEntrances, int
amountOfOutputs);
```

```
fstream PLIK_WYJSCIOWY_UCZENIE;
fstream PLIK_WYJSCIOWY_TESTOWANIE;
fstream PLIK_WYJSCIOWY_TEST;
fstream DANE_UCZACE;
fstream DANE_TESTUJACE;

int main()
{
	srand(time(NULL));
	vector< vector<double> > daneUczace;
	vector< vector<double> > daneTestujace;
	int liczbaNeuronow = 7;
	int liczabaWejsc = 4;
	int liczbaWyjsc = 1;
	double wspolczynnikUczenia = 0.01;
	Warstwa kohonenNetwork(liczbaNeuronow, liczabaWejsc, liczbaWyjsc, wspolczynnikUczenia);
	wczytajDaneUczace(daneUczace, liczabaWejsc, liczbaWyjsc);
	wczytajDaneTestujace(daneTestujace, liczabaWejsc, liczbaWyjsc);

	do
	{
		cout << "1. Ucz" << endl;
		cout << "2. Testuj" << endl;
		cout << "3. Wyscie" << endl;

		int choice;
		cin >> choice;
		switch (choice)
		{
		case 1:
			PLIK_WYJSCIOWY_UCZENIE.open("output_learning_data.txt", ios::out);

			for (int liczbaEpok = 1, i = 0; i < 50; i++, liczbaEpok++)
			{
				ucz(kohonenNetwork, daneUczace);
				PLIK_WYJSCIOWY_UCZENIE << "EPOCH NUMBER: " << liczbaEpok << endl;
			}
			break;

		case 2:
			PLIK_WYJSCIOWY_TESTOWANIE.open("output_testing_data.txt", ios::out);
			PLIK_WYJSCIOWY_TEST.open("output_testing_neuron.txt", ios::out);
			testuj(kohonenNetwork, daneTestujace);
			break;

		case 3:
			PLIK_WYJSCIOWY_UCZENIE.close();
			PLIK_WYJSCIOWY_TESTOWANIE.close();
			return 0;
		}

	} while (true);
```

```
	return 0;
}

void ustawDaneWejsciowe(Neuron& neuron, vector< vector<double> > inputData, int liczbaWejsc, int wiersz)
{
	for (int i = 0; i < liczbaWejsc; i++)
		neuron.ustawWejscie(i, inputData[wiersz][i]);
}

void ucz(Warstwa& layer, vector< vector<double> > inputData)
{
	static int licznik = 0;
	for (int wierszDanych = 0; wierszDanych < inputData.size(); wierszDanych++)
	{
		for (int i = 0; i < layer.wezLiczbeNeuronow(); i++)
		{
			ustawDaneWejsciowe(layer.neurony[i], inputData,
layer.neurony[i].wezLiczbeWejsc(), wierszDanych);
			layer.neurony[i].liczSume();
		}
		layer.zmienWagi(false);
		if (licznik == 35)
		{
			licznik = 0;
			PLIK_WYJSCIOWY_UCZENIE << "Next flower" << endl;
		}
		PLIK_WYJSCIOWY_UCZENIE << layer.wezIndeksZwyciezcy() << endl;
		licznik++;
	}
}

void testuj(Warstwa& layer, vector< vector<double> > danaWejsciowa)
{
	static int licznik = 0;
	for (int wierszDanych = 0; wierszDanych < danaWejsciowa.size(); wierszDanych++)
	{
		for (int i = 0; i < layer.wezLiczbeNeuronow(); i++)
		{
			ustawDaneWejsciowe(layer.neurony[i], danaWejsciowa,
layer.neurony[i].wezLiczbeWejsc(), wierszDanych);
			layer.neurony[i].liczSume();
		}

		if (licznik == 15)
		{
			licznik = 0;
			PLIK_WYJSCIOWY_TEST << "Next flower" << endl;
		}

		layer.zmienWagi(true);
		PLIK_WYJSCIOWY_TESTOWANIE <<
layer.neurony[layer.wezIndeksZwyciezcy()].wezSumeWszystkich() << endl;
		PLIK_WYJSCIOWY_TEST << layer.wezIndeksZwyciezcy() << endl;
```

```
			licznik++;
		}
}

void wczytajDaneUczace(vector< vector<double> > &daneUczace, int liczbaWejsc, int liczbaWyjsc)
{
		DANE_UCZACE.open("learning_data_2l.txt", ios::in);
		vector<double> wiersz;

		do
		{
			wiersz.clear();

			for (int i = 0; i < liczbaWejsc; i++)
			{
				double inputTmp = 0.0;
				DANE_UCZACE >> inputTmp;
				wiersz.push_back(inputTmp);
			}

		  double vectorLength = 0.0;

			for (int i = 0; i < liczbaWejsc; i++)
				vectorLength += pow(wiersz[i], 2);

			vectorLength = sqrt(vectorLength);

			for (int i = 0; i < liczbaWejsc; i++)
				wiersz[i] /= vectorLength;

			daneUczace.push_back(wiersz);

		} while (!DANE_UCZACE.eof());

		DANE_UCZACE.close();
}

void wczytajDaneTestujace(vector< vector<double> > &daneTestujace, int liczbaWejsc, int amountOfOutputs)
{
		DANE_TESTUJACE.open("testing_data_2l.txt", ios::in);
		vector<double> row;

		while (!DANE_TESTUJACE.eof())
		{
			row.clear();

			for (int i = 0; i < liczbaWejsc; i++)
			{
				double inputTmp = 0.0;
				DANE_TESTUJACE >> inputTmp;
				row.push_back(inputTmp);
			}
```

```
			double dlugosc = 0.0;

			for (int i = 0; i < liczbaWejsc; i++)
				dlugosc += pow(row[i], 2);

			dlugosc = sqrt(dlugosc);

			for (int i = 0; i < liczbaWejsc; i++)
				row[i] /= dlugosc;

			daneTestujace.push_back(row);
		}
		DANE_TESTUJACE.close();
}
```

Neuron.h

```
#pragma once
#include <iostream>
#include <vector>
using namespace std;

class Neuron
{
public:
		void stworzWejscia(int amountOfDendrites, int amountOfSynapses);
		void stworzWejscie() { _wejscia.push_back(0); }
		void stworzWagi(int index) { _wagi.push_back(0); }
		int wezLiczbeWejsc() { return _wejscia.size(); }
		int wezLiczbeWag() { return _wagi.size(); }
		double wezWage(int index) { return _wejscia[index]; }
		void ustawWejscie(int index, double value) { _wejscia[index] = value; }
		double wezSynapse(int index) { return  _wagi[index]; }
		void ustawSynapse(int index, double value) { _wagi[index] = value; }
		double wezSumeWszystkich() { return _sumaWejsc; }
		double wezWyjscie() { return _wartoscWyjscia; }
		double utworzWejscie(int index) { return _wejscia[index] * _wagi[index]; }
		void utworzWyjscie();
		void liczNowaWage();
		double liczSume();
		Neuron();
		Neuron(int amountOfDendrites, int amountOfOutputs, double learningCoefficient);

private:
		double pierwszaWaga();
		void znormalizowanaWaga();
		vector<double> _wejscia;
		vector<double> _wagi;
		double _sumaWejsc;
		double _wartoscWyjscia;
		double _wspolczynnikUczenia;
};
```

Neuron.cpp

```
#include "Neuron.h"
#include <time.h>
#include <math.h>

Neuron::Neuron()
{
	_wejscia.resize(0);
	_wagi.resize(0);
	_sumaWejsc = 0.0;
	_wartoscWyjscia = 0.0;
	_wspolczynnikUczenia = 0.0;
}

Neuron::Neuron(int amountOfDendrites, int amountOfOutputs, double learningCoefficient)
{
	stworzWejscia(amountOfDendrites, amountOfOutputs);
	znormalizowanaWaga();
	_wspolczynnikUczenia = learningCoefficient;
	_sumaWejsc = 0.0;
	_wartoscWyjscia = 0.0;
}

void Neuron::stworzWejscia(int amountOfDendrites, int amountOfOutputs)
{
	for (int j = 0; j < amountOfDendrites; j++)
	{
		_wejscia.push_back(0);
		_wagi.push_back(pierwszaWaga());
	}
}

double Neuron::liczSume()
{
	_sumaWejsc = 0.0;
	for (int i = 0; i < wezLiczbeWejsc(); i++)
		//_sumOfAllInputs += (pow(_inputs[i] - _weights[i], 2));
		_sumaWejsc += _wejscia[i] * _wagi[i];
	return _sumaWejsc;
}

void Neuron::utworzWyjscie()
{
	double beta = 1.0;
	_wartoscWyjscia = (1.0 / (1.0 + (exp(-beta * _sumaWejsc))));
}

void Neuron::liczNowaWage()
{
	for (int i = 0; i < wezLiczbeWag(); i++)
		_wagi[i] = _wagi[i] + (_wspolczynnikUczenia * (_wejscia[i] - _wagi[i]));
```

```
	znormalizowanaWaga();
}

double Neuron::pierwszaWaga()
{
	double max = 1.0;
	double min = 0.0;
	double weight = ((double(rand()) / double(RAND_MAX)) * (max - min)) + min;
	return weight;
}

void Neuron::znormalizowanaWaga()
{
	double vectorLength = 0.0;

	for (int i = 0; i < wezLiczbeWag(); i++)
		vectorLength += pow(_wagi[i], 2);
	vectorLength = sqrt(vectorLength);
	for (int i = 0; i < wezLiczbeWag(); i++)
		_wagi[i] /= vectorLength;
}
```

Warstwa.h

```
#pragma once
#include<vector>
#include"Neuron.h"
using namespace std;

class Warstwa
{
public:
	Warstwa(int numberOfNeurons, int amountOfDendrites, int amountOfOutputs, double
learningCoefficient);
	vector<Neuron> neurony;
	int wezLiczbeNeuronow() { return _liczbaNeuronow; }
	void zmienWagi(bool testing);
	double wezSumy(int index) { return _ssumy[index]; }
	void zbierzSumy();
	int wezIndeksZwyciezcy(){ return _indeksZwyciezcy; }

private:
	void znajdzMaxSume(bool testing);
	int _liczbaNeuronow;
	vector<double> _ssumy;
	int _indeksZwyciezcy;
};
```

Warstwa.cpp

```
#include"Warstwa.h"

Warstwa::Warstwa(int numberOfNeurons, int amountOfDendrites, int amountOfOutputs, double
learningCoefficient)
```

```
{
	_liczbaNeuronow = numberOfNeurons;
	neurony.resize(numberOfNeurons);
	for (int i = 0; i < numberOfNeurons; i++)
		neurony[i].Neuron::Neuron(amountOfDendrites, amountOfOutputs, learningCoefficient);
}

void Warstwa::zmienWagi(bool testing)
{
	zbierzSumy();
	znajdzMaxSume(testing);
}

void Warstwa::zbierzSumy()
{
	_ssumy.clear();
	for (int i = 0; i < _liczbaNeuronow; i++)
		_ssumy.push_back(neurony[i].liczSume());
}

void Warstwa::znajdzMaxSume(bool testing)
{
	double tmp = _ssumy[0];
	_indeksZwyciezcy = 0;
	for (int i = 1; i < _ssumy.size(); i++)
	{
		if (tmp < _ssumy[i])
		{
			_indeksZwyciezcy = i;
			tmp = _ssumy[i];
		}
	}
	neurony[_indeksZwyciezcy].utworzWyjscie();
	if (testing == false)
		neurony[_indeksZwyciezcy].liczNowaWage();
}
```