11.02.2018 r.	Izabela Musztyfaga	Podstawy Sztucznej Inteligencji
Scenariusz 5 - Budowa i działanie sieci Kohonena dla WTA		

Cel ćwiczenia

Celem ćwiczenia jest poznanie budowy i działania sieci Kohonena przy wykorzystaniu reguły WTA do odwzorowywania istotnych cech kwiatów

Dane uczące sieć – numeryczny opis cech kwiatów[1]

W plikach z danymi założono, że:

- *I. setosa → 1*
- *I. versicolor → 2*
- *I. virginica → 3*

Dataset Order	Sepal length	Sepal width	Petal length	Petal width	Species
1	5.1	3.5	1.4	0.2	*I. setosa*
2	4.9	3.0	1.4	0.2	*I. setosa*
3	4.7	3.2	1.3	0.2	*I. setosa*
4	4.6	3.1	1.5	0.2	*I. setosa*
5	5.0	3.6	1.4	0.3	*I. setosa*
6	5.4	3.9	1.7	0.4	*I. setosa*
7	4.6	3.4	1.4	0.3	*I. setosa*
8	5.0	3.4	1.5	0.2	*I. setosa*
9	4.4	2.9	1.4	0.2	*I. setosa*
10	4.9	3.1	1.5	0.1	*I. setosa*
11	5.4	3.7	1.5	0.2	*I. setosa*
12	4.8	3.4	1.6	0.2	*I. setosa*
13	4.8	3.0	1.4	0.1	*I. setosa*
14	4.3	3.0	1.1	0.1	*I. setosa*
15	5.8	4.0	1.2	0.2	*I. setosa*
16	5.7	4.4	1.5	0.4	*I. setosa*
17	5.4	3.9	1.3	0.4	*I. setosa*
18	5.1	3.5	1.4	0.3	*I. setosa*
19	5.7	3.8	1.7	0.3	*I. setosa*
20	5.1	3.8	1.5	0.3	*I. setosa*
21	5.4	3.4	1.7	0.2	*I. setosa*
22	5.1	3.7	1.5	0.4	*I. setosa*
23	4.6	3.6	1.0	0.2	*I. setosa*
24	5.1	3.3	1.7	0.5	*I. setosa*
25	4.8	3.4	1.9	0.2	*I. setosa*

1 https://en.wikipedia.org/wiki/Iris_flower_data_set

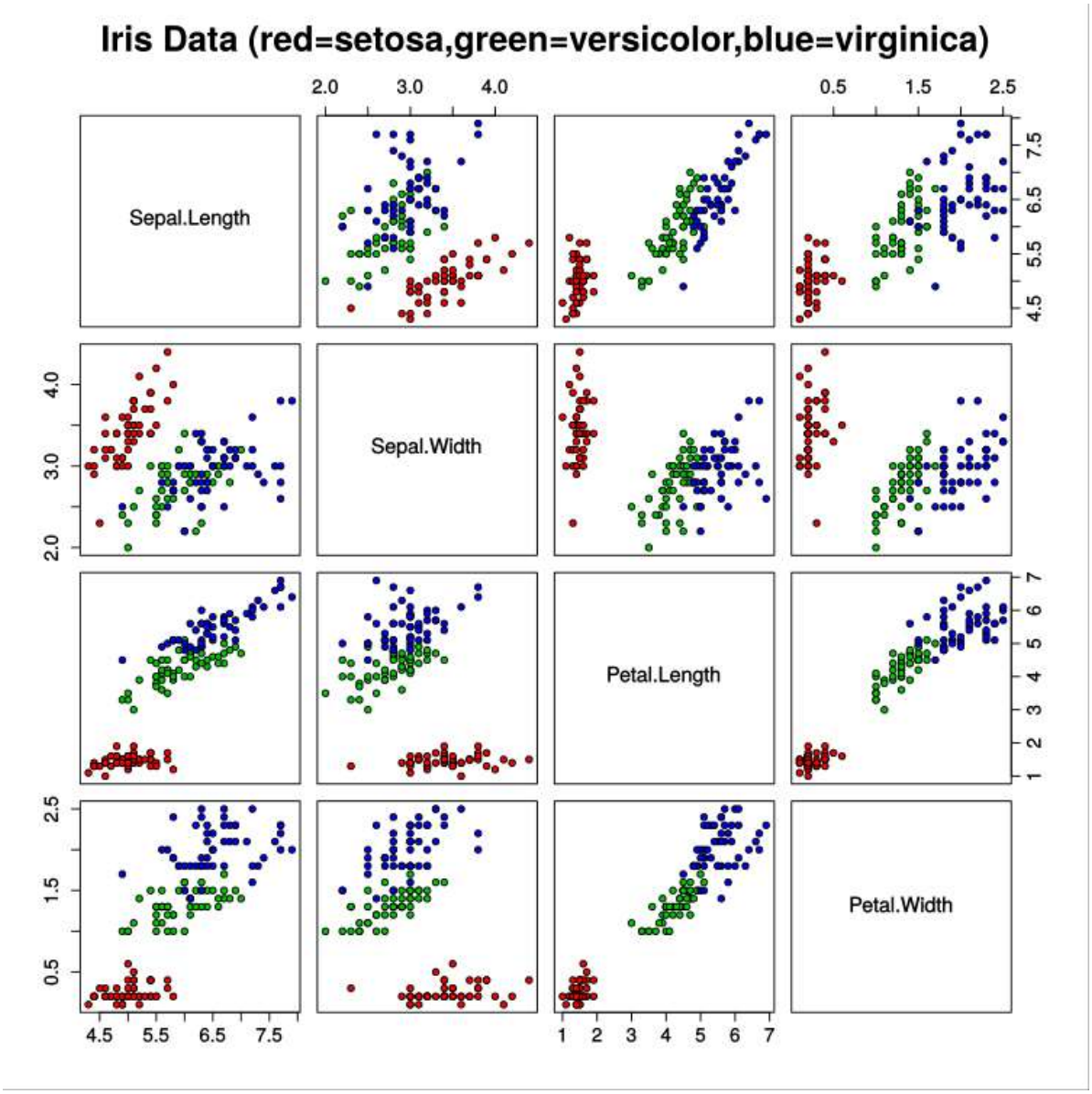
Dataset Order	Sepal length	Sepal width	Petal length	Petal width	Species
26	5.0	3.0	1.6	0.2	*I. setosa*
27	5.0	3.4	1.6	0.4	*I. setosa*
28	5.2	3.5	1.5	0.2	*I. setosa*
29	5.2	3.4	1.4	0.2	*I. setosa*
30	4.7	3.2	1.6	0.2	*I. setosa*
31	4.8	3.1	1.6	0.2	*I. setosa*
32	5.4	3.4	1.5	0.4	*I. setosa*
33	5.2	4.1	1.5	0.1	*I. setosa*
34	5.5	4.2	1.4	0.2	*I. setosa*
35	4.9	3.1	1.5	0.2	*I. setosa*
36	5.0	3.2	1.2	0.2	*I. setosa*
37	5.5	3.5	1.3	0.2	*I. setosa*
38	4.9	3.6	1.4	0.1	*I. setosa*
39	4.4	3.0	1.3	0.2	*I. setosa*
40	5.1	3.4	1.5	0.2	*I. setosa*
41	5.0	3.5	1.3	0.3	*I. setosa*
42	4.5	2.3	1.3	0.3	*I. setosa*
43	4.4	3.2	1.3	0.2	*I. setosa*
44	5.0	3.5	1.6	0.6	*I. setosa*
45	5.1	3.8	1.9	0.4	*I. setosa*
46	4.8	3.0	1.4	0.3	*I. setosa*
47	5.1	3.8	1.6	0.2	*I. setosa*
48	4.6	3.2	1.4	0.2	*I. setosa*
49	5.3	3.7	1.5	0.2	*I. setosa*
50	5.0	3.3	1.4	0.2	*I. setosa*
51	7.0	3.2	4.7	1.4	*I. versicolor*
52	6.4	3.2	4.5	1.5	*I. versicolor*
53	6.9	3.1	4.9	1.5	*I. versicolor*
54	5.5	2.3	4.0	1.3	*I. versicolor*
55	6.5	2.8	4.6	1.5	*I. versicolor*
56	5.7	2.8	4.5	1.3	*I. versicolor*
57	6.3	3.3	4.7	1.6	*I. versicolor*
58	4.9	2.4	3.3	1.0	*I. versicolor*
59	6.6	2.9	4.6	1.3	*I. versicolor*
60	5.2	2.7	3.9	1.4	*I. versicolor*
61	5.0	2.0	3.5	1.0	*I. versicolor*
62	5.9	3.0	4.2	1.5	*I. versicolor*
63	6.0	2.2	4.0	1.0	*I. versicolor*
64	6.1	2.9	4.7	1.4	*I. versicolor*
65	5.6	2.9	3.6	1.3	*I. versicolor*

Dataset Order	Sepal length	Sepal width	Petal length	Petal width	Species
66	6.7	3.1	4.4	1.4	*I. versicolor*
67	5.6	3.0	4.5	1.5	*I. versicolor*
68	5.8	2.7	4.1	1.0	*I. versicolor*
69	6.2	2.2	4.5	1.5	*I. versicolor*
70	5.6	2.5	3.9	1.1	*I. versicolor*
71	5.9	3.2	4.8	1.8	*I. versicolor*
72	6.1	2.8	4.0	1.3	*I. versicolor*
73	6.3	2.5	4.9	1.5	*I. versicolor*
74	6.1	2.8	4.7	1.2	*I. versicolor*
75	6.4	2.9	4.3	1.3	*I. versicolor*
76	6.6	3.0	4.4	1.4	*I. versicolor*
77	6.8	2.8	4.8	1.4	*I. versicolor*
78	6.7	3.0	5.0	1.7	*I. versicolor*
79	6.0	2.9	4.5	1.5	*I. versicolor*
80	5.7	2.6	3.5	1.0	*I. versicolor*
81	5.5	2.4	3.8	1.1	*I. versicolor*
82	5.5	2.4	3.7	1.0	*I. versicolor*
83	5.8	2.7	3.9	1.2	*I. versicolor*
84	6.0	2.7	5.1	1.6	*I. versicolor*
85	5.4	3.0	4.5	1.5	*I. versicolor*
86	6.0	3.4	4.5	1.6	*I. versicolor*
87	6.7	3.1	4.7	1.5	*I. versicolor*
88	6.3	2.3	4.4	1.3	*I. versicolor*
89	5.6	3.0	4.1	1.3	*I. versicolor*
90	5.5	2.5	4.0	1.3	*I. versicolor*
91	5.5	2.6	4.4	1.2	*I. versicolor*
92	6.1	3.0	4.6	1.4	*I. versicolor*
93	5.8	2.6	4.0	1.2	*I. versicolor*
94	5.0	2.3	3.3	1.0	*I. versicolor*
95	5.6	2.7	4.2	1.3	*I. versicolor*
96	5.7	3.0	4.2	1.2	*I. versicolor*
97	5.7	2.9	4.2	1.3	*I. versicolor*
98	6.2	2.9	4.3	1.3	*I. versicolor*
99	5.1	2.5	3.0	1.1	*I. versicolor*
100	5.7	2.8	4.1	1.3	*I. versicolor*
101	6.3	3.3	6.0	2.5	*I. virginica*
102	5.8	2.7	5.1	1.9	*I. virginica*
103	7.1	3.0	5.9	2.1	*I. virginica*
104	6.3	2.9	5.6	1.8	*I. virginica*
105	6.5	3.0	5.8	2.2	*I. virginica*

Dataset Order	Sepal length	Sepal width	Petal length	Petal width	Species
106	7.6	3.0	6.6	2.1	*I. virginica*
107	4.9	2.5	4.5	1.7	*I. virginica*
108	7.3	2.9	6.3	1.8	*I. virginica*
109	6.7	2.5	5.8	1.8	*I. virginica*
110	7.2	3.6	6.1	2.5	*I. virginica*
111	6.5	3.2	5.1	2.0	*I. virginica*
112	6.4	2.7	5.3	1.9	*I. virginica*
113	6.8	3.0	5.5	2.1	*I. virginica*
114	5.7	2.5	5.0	2.0	*I. virginica*
115	5.8	2.8	5.1	2.4	*I. virginica*
116	6.4	3.2	5.3	2.3	*I. virginica*
117	6.5	3.0	5.5	1.8	*I. virginica*
118	7.7	3.8	6.7	2.2	*I. virginica*
119	7.7	2.6	6.9	2.3	*I. virginica*
120	6.0	2.2	5.0	1.5	*I. virginica*
121	6.9	3.2	5.7	2.3	*I. virginica*
122	5.6	2.8	4.9	2.0	*I. virginica*
123	7.7	2.8	6.7	2.0	*I. virginica*
124	6.3	2.7	4.9	1.8	*I. virginica*
125	6.7	3.3	5.7	2.1	*I. virginica*
126	7.2	3.2	6.0	1.8	*I. virginica*
127	6.2	2.8	4.8	1.8	*I. virginica*
128	6.1	3.0	4.9	1.8	*I. virginica*
129	6.4	2.8	5.6	2.1	*I. virginica*
130	7.2	3.0	5.8	1.6	*I. virginica*
131	7.4	2.8	6.1	1.9	*I. virginica*
132	7.9	3.8	6.4	2.0	*I. virginica*
133	6.4	2.8	5.6	2.2	*I. virginica*
134	6.3	2.8	5.1	1.5	*I. virginica*
135	6.1	2.6	5.6	1.4	*I. virginica*
136	7.7	3.0	6.1	2.3	*I. virginica*
137	6.3	3.4	5.6	2.4	*I. virginica*
138	6.4	3.1	5.5	1.8	*I. virginica*
139	6.0	3.0	4.8	1.8	*I. virginica*
140	6.9	3.1	5.4	2.1	*I. virginica*
141	6.7	3.1	5.6	2.4	*I. virginica*
142	6.9	3.1	5.1	2.3	*I. virginica*
143	5.8	2.7	5.1	1.9	*I. virginica*
144	6.8	3.2	5.9	2.3	*I. virginica*
145	6.7	3.3	5.7	2.5	*I. virginica*

Dataset Order	Sepal length	Sepal width	Petal length	Petal width	Species
146	6.7	3.0	5.2	2.3	*I. virginica*
147	6.3	2.5	5.0	1.9	*I. virginica*
148	6.5	3.0	5.2	2.0	*I. virginica*
149	6.2	3.4	5.4	2.3	*I. virginica*
150	5.9	3.0	5.1	1.8	*I. virginica*

Rys. 1 –



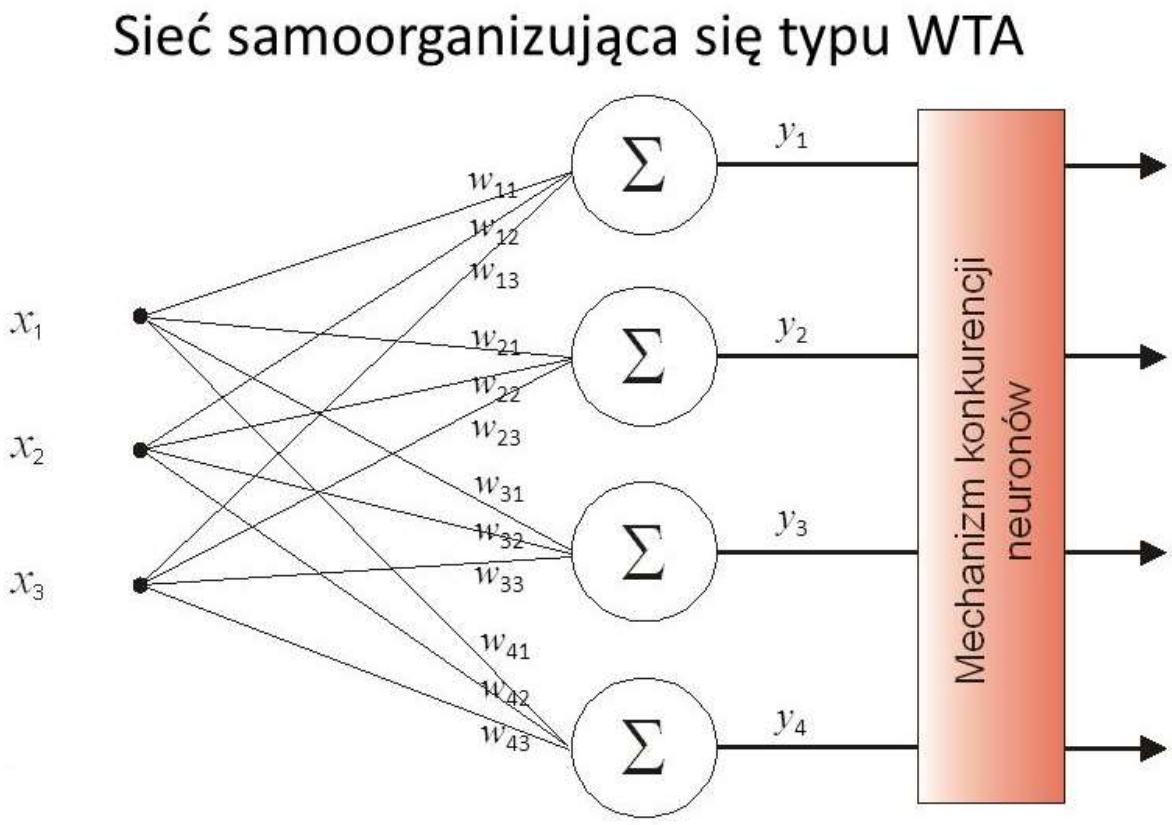
Graficzne przedstawienie danych

Dane testujące sieć – numeryczny opis cech kwiatów

Dataset Order	Sepal length	Sepal width	Petal length	Petal width	Species
1	4.7	3.1	1.2	0.2	*I. setosa*
2	5.5	3.5	1.3	0.2	*I. setosa*
3	4.7	3.6	1.4	0.1	*I. setosa*
4	4.4	3.0	1.3	0.2	*I. setosa*
5	5.1	3.4	1.5	0.2	*I. setosa*
6	5.0	3.5	1.3	0.3	*I. setosa*
7	4.5	2.3	1.3	0.3	*I. setosa*
8	4.4	3.2	1.3	0.2	*I. setosa*
9	5.0	2.5	1.5	0.6	*I. setosa*
10	5.1	3.8	1.9	0.4	*I. setosa*
11	4.8	3.0	1.8	0.3	*I. setosa*
12	4.1	3.8	1.6	0.2	*I. setosa*
13	4.6	2.2	1.4	0.2	*I. setosa*
14	5.3	3.7	1.5	0.4	*I. setosa*
15	5.0	3.3	1.8	0.5	*I. setosa*
16	7.0	2.8	3.9	1.4	*I. versicolor*
17	6.4	3.2	4.4	1.5	*I. versicolor*
18	6.9	3.1	4.7	1.5	*I. versicolor*
19	5.5	2.3	4.0	1.2	*I. versicolor*
20	6.5	2.8	4.6	1.5	*I. versicolor*
21	5.7	2.8	4.5	1.3	*I. versicolor*
22	6.3	3.3	4.7	1.7	*I. versicolor*
23	4.9	2.4	3.3	1.0	*I. versicolor*
24	6.6	2.9	5.1	1.3	*I. versicolor*
25	5.2	2.7	3.9	1.4	*I. versicolor*
26	5.0	2.0	3.5	1.0	*I. versicolor*
27	6.0	3.0	4.0	1.7	*I. versicolor*
28	6.7	2.2	4.0	1.0	*I. versicolor*
29	6.1	2.9	4.7	1.5	*I. versicolor*
30	5.6	2.9	3.6	1.3	*I. versicolor*
31	6.3	3.3	6.0	2.5	*I. virginica*
32	6.5	3.0	5.5	1.3	*I. virginica*
33	5.7	3.8	6.7	2.2	*I. virginica*
34	7.7	2.6	6.9	2.3	*I. virginica*
35	6.0	2.2	5.0	1.5	*I. virginica*
36	8.9	3.2	5.7	2.3	*I. virginica*
37	5.6	2.8	4.9	2.0	*I. virginica*

Dataset Order	Sepal length	Sepal width	Petal length	Petal width	Species
38	7.7	2.8	6.7	2.0	*I. virginica*
39	6.3	3.1	4.9	1.8	*I. virginica*
40	6.7	3.3	5.7	2.1	*I. virginica*
41	7.2	3.2	6.0	2.8	*I. virginica*
42	7.2	2.8	2.8	1.8	*I. virginica*
43	6.1	3.0	5.1	1.8	*I. virginica*
44	6.4	2.7	5.6	2.1	*I. virginica*
45	7.2	3.4	5.8	1.6	*I. virginica*

Syntetyczny opis sieci



Rys. 2 – Schemat sieci samoorganizującej się WTA

Nauka sieci odbywa się za pomocą uczenia rywalizującego (metoda uczenia sieci samoorganizujących). Podczas procesu uczenia neurony są nauczane rozpoznawania danych i zbliżają się do obszarów zajmowanych przez te dane. Po wejściu każdego wektora uczącego wybierany jest tylko jeden neuron (neuron będący najbliższemu prezentowanemu wzorcowi). Wszystkie neurony rywalizują między sobą, gdzie zwycięża ten neuron, którego wartość jest największa. Zwycięski neuron przyjmuje na wyjściu wartość 1, pozostałe 0.

Sieć dzieli dane na grupy, biorąc pod uwagę to, aby elementy były do siebie jak najbardziej podobne, a jednocześnie zupełnie inne dla różnych grup.

Najpopularniejszą metodą uczenia sieci samoorganizujących się jest uczenie rywalizujące. Zaimplementowane neurony uczą się rozpoznawania obszarów, w których są jakieś dane. Po 'wejściu' wektora w obszar z danymi wybierany jest dokładnie jeden neuron, który

Algorytm uczenia

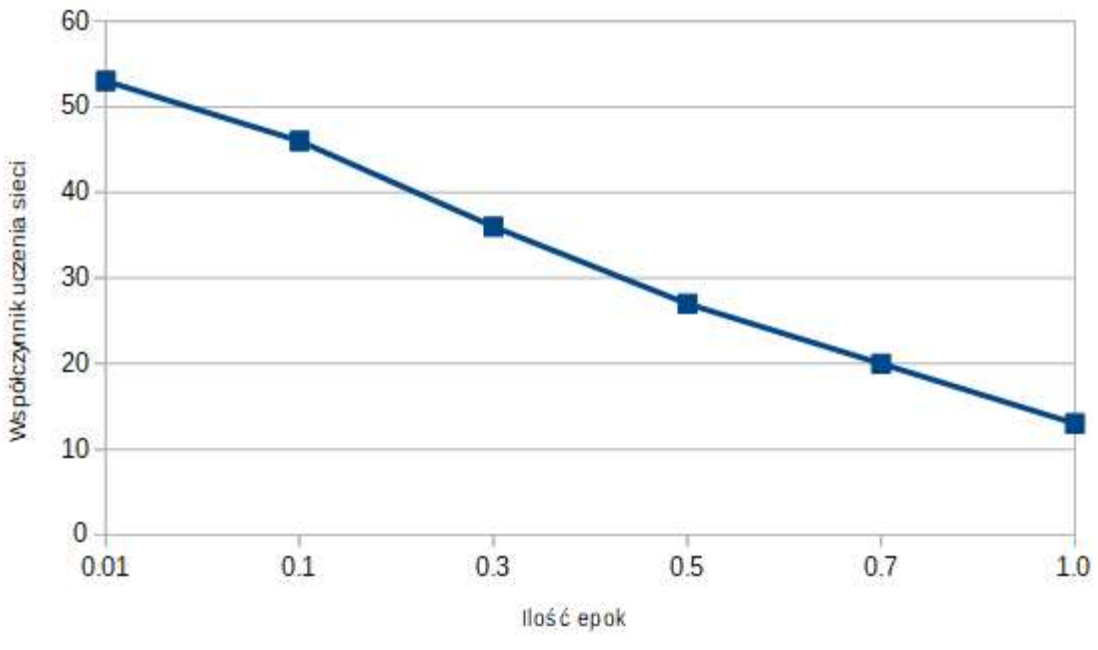
1. Normalizacja danych uczących i danych testowych
2. Wybór współczynnika uczenia η z przedziału (0; 1>
3. Losowy wybór początkowych wartości wag z przedziału <0; 1)
4. Dla każdego zbioru danych uczących obliczamy odpowiedź sieci – obliczane dla każdego pojedynczego neuronu.
5. Wybierany jest neuron, dla którego obliczona suma ilorazów sygnałów wejściowych oraz wag (punkt 5) jest największa. Tylko dla tego neuronu następuje aktualizacja wag, według poniższego wzoru:

$$w_{i,j}(t+1) = w_{i,j}(t) + \eta * \left(x_i * w_{i,j}(t)\right)$$

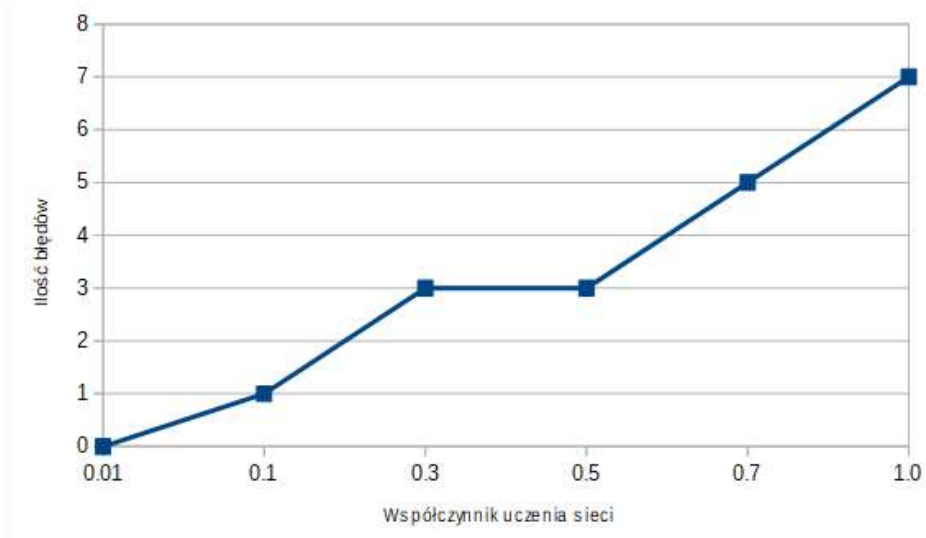
6. Normalizacja wartości nowego wektora wag
7. Zwycięski neuron daje odpowiedź na swoim wyjściu równą 1, a pozostałe 0.
8. Wczytanie kolejnego wektora uczącego – aż nie dostaną wczytane wszystkie wektory uczące.

Zestawienie otrzymanych wyników

Współczynnik uczenia	Ilość epok uczenia
0.01	53
0.1	46



	sieć
0.01	0
0.1	1
0.3	3
0.5	3
0.7	5
1.0	7



Wnioski

Sieć Kohonena jest samoorganizująca się – to znaczy tyle, że potrafi sama dzielić dane ze względu na ich różne wartości. Co więcej, można uczyć ją bez nauczyciela, tzn bez podawania pewnych wartości oczekiwanych. Sama efektywność procesu uczenia jest ściśle związana ze współczynnikiem uczenia się sieci. Im jest on większy, tym szybciej sieć uczy się efektywnie. Niestety, odwrotnie jest w przypadku ilości popełnianych przez sieć błędów.
Dane pobierane przez sieć – w tym przypadku wektory z danymi – muszą być poddane procesowi normalizacji.

Listing kodu wraz z komentarzami (nie własny kod, znaleziony w internecie)

main.cpp

```
#include <iostream>
#include <ctime>
#include <fstream>
#include <vector>

#include "Network.h"

using namespace std;
```

```
//wczytanie do tablic danych wejsciowych
void setInputData(Neuron& neuron, vector<vector<double>> inputData, int numberOfInputs, int
inputDataRow);
//uczenie sieci
void learn(Network& layer, vector<vector<double>> inputData);
//testowanie sieci
void test(Network& layer, vector<vector<double>> inputData);
//wczytanie danych uczacych
void loadTrainingData(vector<vector<double>>&learnData, int numberOfInputs);
//wczytaniee danych testowych
void loadTestingData(vector<vector<double>>&testData, int numberOfInpus);

//strumienie do plikow sluzace do wczytania danych uczacych oraz zapisu wynikow
fstream OUTPUT_FILE_LEARNING, OUTPUT_FILE_TESTING_SUM, OUTPUT_FILE_TESTING_WINNER;
fstream TRAINING_DATA, TESTING_DATA;

int main()
{
	srand(time(NULL));

	//wektory z danymi uczacymi oraz testujacymi
	vector<vector<double>> trainData;
	vector<vector<double>> testData;
	int numberOfNeurons = 10;
	int numberOfInputs = 4;
	double learningRate = 0.01;
	//stworzenie sieci Kohonena
	Network kohonenNetwork(numberOfNeurons, numberOfInputs, learningRate);
	//wczytanie danych uczacych
	loadTrainingData(trainData, numberOfInputs);
	//wczytaniee danych testowych
	loadTestingData(testData, numberOfInputs);


	do {
		cout << "1. Learn" << endl;
		cout << "2. Test" << endl;
		cout << "3. Exit" << endl;

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

			for (int epoch = 1, i = 0; i < 5; i++, epoch++)
			{
				//uczenie
				OUTPUT_FILE_LEARNING << "Epoka: " << epoch << endl;
				cout << "Epoka: " << epoch << endl;
				learn(kohonenNetwork, trainData);
			}
			OUTPUT_FILE_LEARNING.close();
			break;
		case 2:
			OUTPUT_FILE_TESTING_SUM.open("Dane_testujace_wyjscie.txt", ios::out);
			//testowanie
			test(kohonenNetwork, testData);
			break;
		case 3:
			OUTPUT_FILE_LEARNING.close();
			OUTPUT_FILE_TESTING_SUM.close();
```

```
                        OUTPUT_FILE_TESTING_WINNER.close();
                        return 0;
                default:
                        cout << "Blad" << endl;
                }
        } while (true);

        return 0;
}

//wczytanie do tablic danych wejsciowych
void setInputData(Neuron& neuron, vector<vector<double>> inputData, int numberOfInputs, int
row)
{
        for (int i = 0; i < numberOfInputs; i++) {
                neuron.inputs[i] = inputData[row][i];
        }
}

//uczenie
void learn(Network& layer, vector<vector<double>> inputData)
{
        int counter = 0;
        for (int rowOfData = 0; rowOfData < inputData.size(); rowOfData++)
        {
                for (int i = 0; i < layer.numberOfNeurons; i++)
                {
                        //wczytanie danych do tablic
                        setInputData(layer.neurons[i], inputData,
layer.neurons[i].getNumberOfInputs(), rowOfData);
                        //wyliczenie sumy wejscia
                        layer.neurons[i].calculateSumOfAllInputs();
                }
                //zmiana wag
                layer.changeWeights(true);
                //przeskoczenie do kolejnego rodzaju kwiatka
                if (counter == 50)
                {
                        counter = 0;
                        OUTPUT_FILE_LEARNING << "Nastepny kawiat" << endl;
                        cout << "Nastepny kawiatr" << endl;
                }

                OUTPUT_FILE_LEARNING << layer.winnerIndex << endl;
                cout << "Zwyciezca: " << layer.winnerIndex << endl;
                counter++;
        }
}

//testowanie
void test(Network& layer, vector<vector<double>> inputData) {
        int counter = 0;
        for (int wierszDanych = 0; wierszDanych < inputData.size(); wierszDanych++) {
                for (int i = 0; i < layer.numberOfNeurons; i++) {
                        //wczytanie danych do tablic
                        setInputData(layer.neurons[i], inputData,
layer.neurons[i].getNumberOfInputs(), wierszDanych);
                        //wyliczenia sumy wejscia
                        layer.neurons[i].calculateSumOfAllInputs();
                }
                //przeskoczenie na kolejny rodzaj kwiatka (wyzerowanie licznika)
                if (counter == 15)
```

```
		{
			counter = 0;
			OUTPUT_FILE_TESTING_WINNER << "Nastepny kawiat" << endl;
			cout << "Nastepny kawiat" << endl;
		}

		//wagi nie beda zaktualiowane dla zwyciezcy
		layer.changeWeights(false);
		OUTPUT_FILE_TESTING_SUM << layer.neurons[layer.winnerIndex].sumOfAllInputs <<
endl;
		OUTPUT_FILE_TESTING_WINNER << layer.winnerIndex << endl;
		cout << "Zwyciezki neuron: " << layer.winnerIndex << endl;
		counter++;
	}
}

//wczytanie danych uczacych z pliku
void loadTrainingData(vector<vector<double>> &trainData, int numberOfInputs) {
	TRAINING_DATA.open("dane_uczace.txt", ios::in);
	vector<double> row;

	do {
		row.clear();
		for (int i = 0; i < numberOfInputs; i++) {
			double inputTmp = 0.0;
			TRAINING_DATA >> inputTmp;
			row.push_back(inputTmp);
			if (i == numberOfInputs - 1) {
				TRAINING_DATA >> inputTmp;
				//row.push_back(inputTmp);
			}
		}

		//znormalizowanie danych uczacych
		double length = 0.0;

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

		length = sqrt(length);

		for (int i = 0; i < numberOfInputs; i++)
			row[i] /= length;

		trainData.push_back(row);

	} while (!TRAINING_DATA.eof());

	TRAINING_DATA.close();
}

//wczytanie danych testujacych z pliku
void loadTestingData(vector<vector<double>> &testData, int numberOfInputs) {
	TESTING_DATA.open("dane_testujace.txt", ios::in);
	vector<double> row;

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

		for (int i = 0; i < numberOfInputs; i++) {
			double inputTmp = 0.0;
			TESTING_DATA >> inputTmp;
```

```
                        row.push_back(inputTmp);
                        if (i == numberOfInputs - 1) {
                                TRAINING_DATA >> inputTmp;
                                //row.push_back(inputTmp);
                        }
                }

                //znormalizowanie danych uczacych
                double length = 0.0;

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

                length = sqrt(length);

                for (int i = 0; i < numberOfInputs; i++)
                        row[i] /= length;

                testData.push_back(row);
        }

        TESTING_DATA.close();
}
```

Neuron.h

```
#include <iostream>
#include <vector>

using namespace std;

class Neuron
{
public:

        vector<double> inputs; //wejscia
        vector<double> weights; //wagi
        double sumOfAllInputs; //suma wszystkich wejsc
        double outputValue; //wartosc wyjsciowa
        double learningRate; //wspolczynnik uczenia

        Neuron(); //konstruktor
        Neuron(int numberOfInputs, double learningRate); //konstruktor
        ~Neuron();

        double firstWeight(); //wylosowanie poczatkowych wag z zakresu <0;1)
        void normalizeWeight(); //znormalizowanie wag (podczas procesu uczenia)
                                                //stworzenie poczatkowych wejsc (ustawienie
wejsc na 0, wykorzystanie metody firstWeight())
        void createInputs(int numberOfInputs);
        void activationFunction(); //funkcja sigmoidalna obliczajaca wyjscie
        void calculateNewWeight(); //obliczenie nowej wagi dla zwycieskiego neuronu
        double calculateSumOfAllInputs(); //obliczenie sumy wszystkich wejsc

        int getNumberOfInputs() { //zwraca rozmiar wejsc
                return inputs.size();
        }

        int getNumberOfWeights() { //zwraca rozmiar wag
                return weights.size();
        }
};
```

Neuron.cpp

```
#include "Neuron.h"
#include <ctime>
#include <cmath>

//konstruktor
Neuron::Neuron() {
	this->inputs.resize(0);
	this->weights.resize(0);
	this->sumOfAllInputs = 0.0;
	this->outputValue = 0.0;
	this->learningRate = 0.0;
}

//konstruktor
Neuron::Neuron(int numberOfInputs, double learningRate) {
	createInputs(numberOfInputs);
	normalizeWeight();
	this->learningRate = learningRate;
	this->sumOfAllInputs = 0.0;
	this->outputValue = 0.0;
}

Neuron::~Neuron()
{

}

//stworzenie poczatkowych wejsc (ustawienie wejsc na 0, wykorzystanie metody firstWeight())
void Neuron::createInputs(int numberOfInputs)
{
	for (int j = 0; j < numberOfInputs; j++) {
		this->inputs.push_back(0);
		this->weights.push_back(firstWeight());
	}
}

//obliczenie sumy wszystkich wejsc
double Neuron::calculateSumOfAllInputs()
{
	this->sumOfAllInputs = 0.0;
	for (int i = 0; i < getNumberOfInputs(); i++)
		this->sumOfAllInputs += inputs[i] * weights[i];
	return sumOfAllInputs;
}

//funkcja sigmoidalna obliczajaca wyjscie
void Neuron::activationFunction() {
	double beta = 1.0;
	this->outputValue = (1.0 / (1.0 + (exp(-beta * this->sumOfAllInputs))));
}

//obliczenie nowych wag
void Neuron::calculateNewWeight() {
	for (int i = 0; i < getNumberOfWeights(); i++)
		this->weights[i] += this->learningRate*(this->inputs[i] - this->weights[i]);
```

```
	normalizeWeight();
}

//ustalenie poczatkowych wag dla wszystkich wejsc - zakres <0;1)
double Neuron::firstWeight() {
	double max = 1.0;
	double min = 0.0;
	double weight = ((double(rand()) / double(RAND_MAX))*(max - min)) + min;
	return weight;
}

//znormalizowanie nowo obliczonej wagi zwycieskiego neuronu
void Neuron::normalizeWeight() {
	double vectorLength = 0.0;

	for (int i = 0; i < getNumberOfWeights(); i++)
		vectorLength += pow(this->weights[i], 2);

	vectorLength = sqrt(vectorLength);
	for (int i = 0; i < getNumberOfWeights(); i++)
		this->weights[i] /= vectorLength;
}
```

Network.h

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

class Network
{
public:

	vector<Neuron> neurons; //wektor neuronow
	vector<double> sums; //wektor sum wejsc
	int numberOfNeurons; //liczba neuronow
	int winnerIndex; //indeks zwyciezcy

	Network();				//konstruktor
	Network(int numberOfNeurons, int numberOfInputs, double learningRate);
	~Network();

	//zmiana wag (learning = true dla procesu uczenia, = false dla procesu testowania)
	void changeWeights(bool learning);
	void findTheLargestSum(bool learning); //szukanie zwycieskiego neuronu
	void sumOfTheLayer(); //obliczenie sumy wszystkich wejsc
};
```

Network.cpp

```
#include "Network.h"


Network::Network()
{

}

Network::Network(int numberOfNeurons, int numberOfInputs, double learningRate)
	//konstruktor
```

```
{
	this->numberOfNeurons = numberOfNeurons;
	this->neurons.resize(numberOfNeurons);
	for (int i = 0; i < numberOfNeurons; i++)
		this->neurons[i].Neuron::Neuron(numberOfInputs, learningRate);
}

Network::~Network()		//destruktor
{

}

//obliczenie sum wszystkich wejsc, poszukiwanie tego o najwiekszej sumie i aktualizacja jego
wag
void Network::changeWeights(bool learning) {
	sumOfTheLayer();
	findTheLargestSum(learning);
}

//obliczenie sumy wszystkich wejsc
void Network::sumOfTheLayer() {
	this->sums.clear();
	for (int i = 0; i < this->numberOfNeurons; i++)
		this->sums.push_back(neurons[i].calculateSumOfAllInputs());
}

//poszukiwanie wejscia o najwiekszej sumie
void Network::findTheLargestSum(bool learning) {
	double tmp = this->sums[0];
	this->winnerIndex = 0;
	for (int i = 1; i < this->sums.size(); i++) {
		if (tmp < this->sums[i]) {
			this->winnerIndex = i;
			tmp = this->sums[i];
		}
	}
	this->neurons[this->winnerIndex].activationFunction();
	if (learning) //aktualizacja wag
		this->neurons[this->winnerIndex].calculateNewWeight();
}
```