Rafał Szyszka

Sprawozdanie 5 –Budowa i działanie sieci Kohonena dla WTA

1.Cel ćwiczenia:

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

2. Przebieg wykonania ćwiczenia:

a) Przygotowanie danych uczących i testujących zawierających numeryczny opis cech kwiatów. Zestaw został przygotowany na podstawie Wikipedii.
b) Implementacja sieci Kohonena i algorytmu uczenia o regułę Winner TakesAll
c) Uczenie sieci dla różnych współczynników uczenia
d) Testowanie sieci

3. Syntetyczny opis budowy użytej sieci i algorytmów uczenia:

Sieć Kohonena jest to sieć która klasyfikuje wejściowe wektory w jedną z określonej liczby *m* kategorii, zgodnie z klastrami wykrytymi w zbiorze treningowym $\{x^1,\ldots, x^K\}$

Algorytm uczący traktuje set z *m* wektorów wag jako zmienne wektory który muszą być nauczone. Wszystko losowo wybrane wektory wag muszą zostać znormalizowane. Normalizacja odbywa się za pomocą wzoru:

$$w_r = \frac{w_r}{||w_r||}$$

Wagi każdego neuronu tworzą wektor $w_i = [w_{i1}, w_{i2}, \ldots, w_{iN}]^T$. Aktualizacja wag polega na wybraniu takiego w_r który spełnia relację:

$$d_m(x, w_r) = \min_{i=1,\ldots,m} d_m(x, w_i)$$

Gdzie indeks *r* oznacza odpowiedni numer zwycięskiego neuronu do wektora w_r , który jest najbliższą aproksymacją danej wejściowej *x*.
$d_m(x,w_i)$ oznacza odległość w sensie wybranej metryki między wektorem *x* i wektorem *w*. Wybrałem metrykę typu miejskiego (Manhattan):

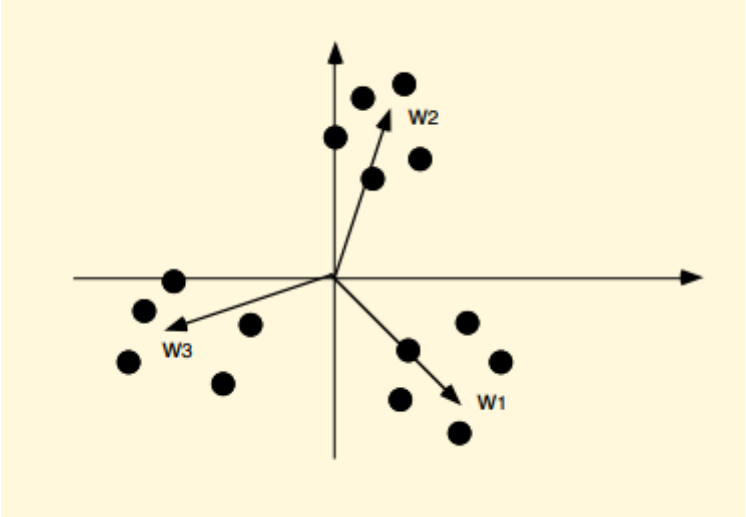
$$d_m(x, w_i) = \sum_{k=1}^{n} |x_k - w_{ik}|$$

Po zidentyfikowaniu zwycięskiego neuronu, jego wagi muszą być zaktuazowane. Odbywa się to według wzoru:

$$w_r = w_r + \eta(x - w_r)$$

Gdzie η to odpowiednio mały krok uczenia wybierany zazwyczaj z przedziału 0.1 i 0.7. Wagi pozostałych neuronów pozostają bez zmian.

Na końcu procesu uczenia ostateczne wektory wag wskazują na środek grawitacji klas.



Rys 1. Finalne wartości wektorów wag.

Warto zauważyć, że sieć będzie trenowalna tylko wtedy jeśli klasy są od siebie separowalne. Aby zapewnić separowalność klas konieczne było użycie nadmiarowej ilości neuronów, ponieważ inicjalizacja wag sieci jest losowa, tak więc część neuronów może 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.

4. Zestawienie i analiza otrzymanych wyników:

Tabela 1. Zbiorcze zestawienie błędów uczenia.

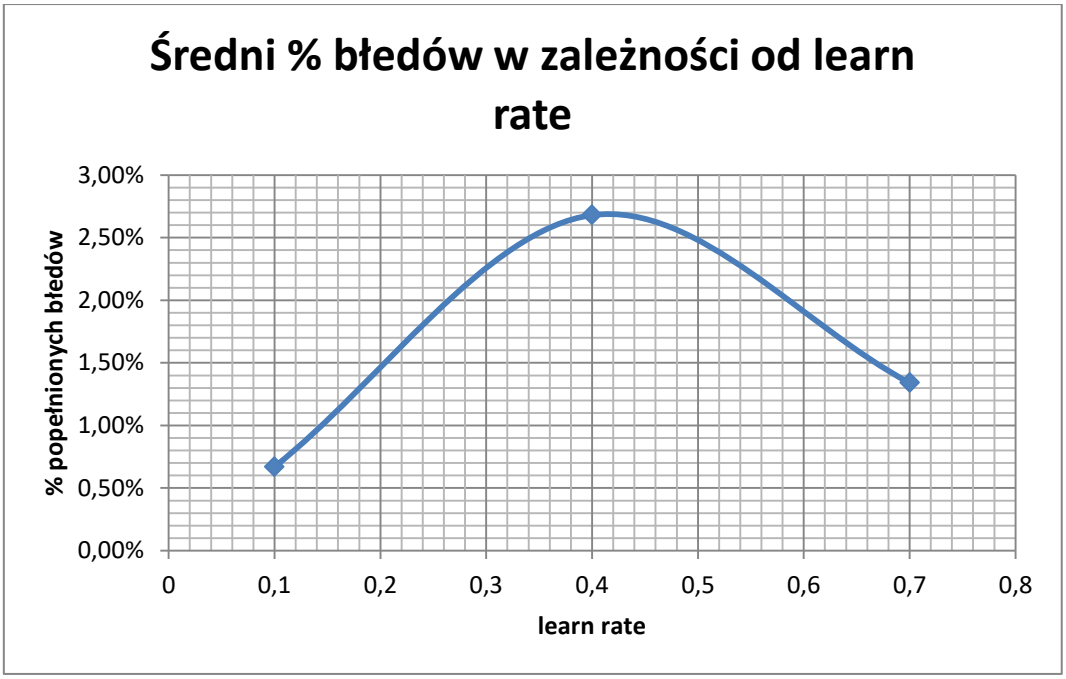
	%bledow w zaleznosci od learn rate		
nr testu	0,1	0,4	0,7
1	0,00%	0,00%	6,70%
2	0,00%	6,70%	0,00%
3	6,70%	6,70%	0,00%
4	0,00%	0,00%	0,00%
5	0,00%	0,00%	0,00%
6	0,00%	6,70%	0,00%
7	0,00%	0,00%	0,00%
8	0,00%	0,00%	0,00%
9	0,00%	0,00%	0,00%
10	0,00%	6,70%	6,70%
ŚREDNIO	0,67%	2,68%	1,34%

Na tabeli 1 mamy zestawione błędy uczenia dla różnych współczynników uczenia dla każdego testu oraz obliczoną średnią. Możemy zauważyć, że najmniejszy % błędów otrzymano dla kroku uczenia 0,1.

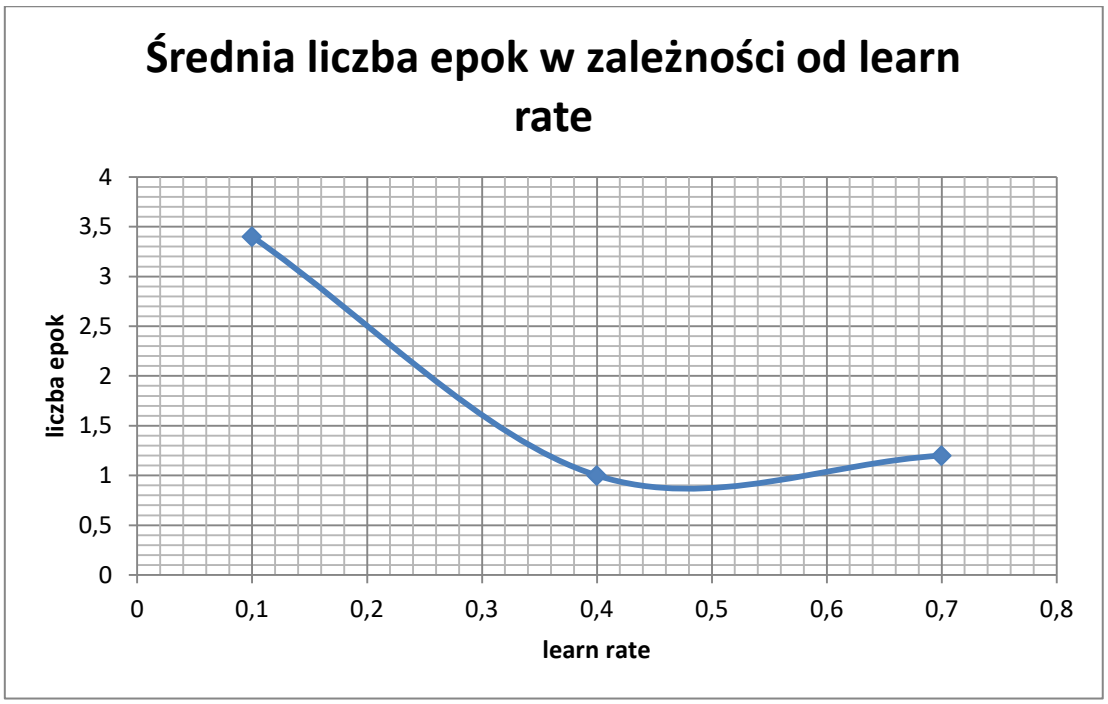
Tabela 2. Zbiorcze zestawienie liczby epok.

	liczba epok w zaleznosci od learn rate		
nr testu	0,1	0,4	0,7
1	12	1	1
2	3	1	1
3	2	1	1
4	1	1	1
5	1	1	1
6	5	1	1
7	1	1	1
8	1	1	1
9	4	1	2
10	4	1	2
ŚREDNIO	3,4	1	1,2

Na tabeli 2 mamy zebrane wyniki liczby epok w zależności od learn rate. Dostrzec można, że sieć uczyla się dłużej dla learn rate 0,1 w porównaniu z 0,4 i 0,7.

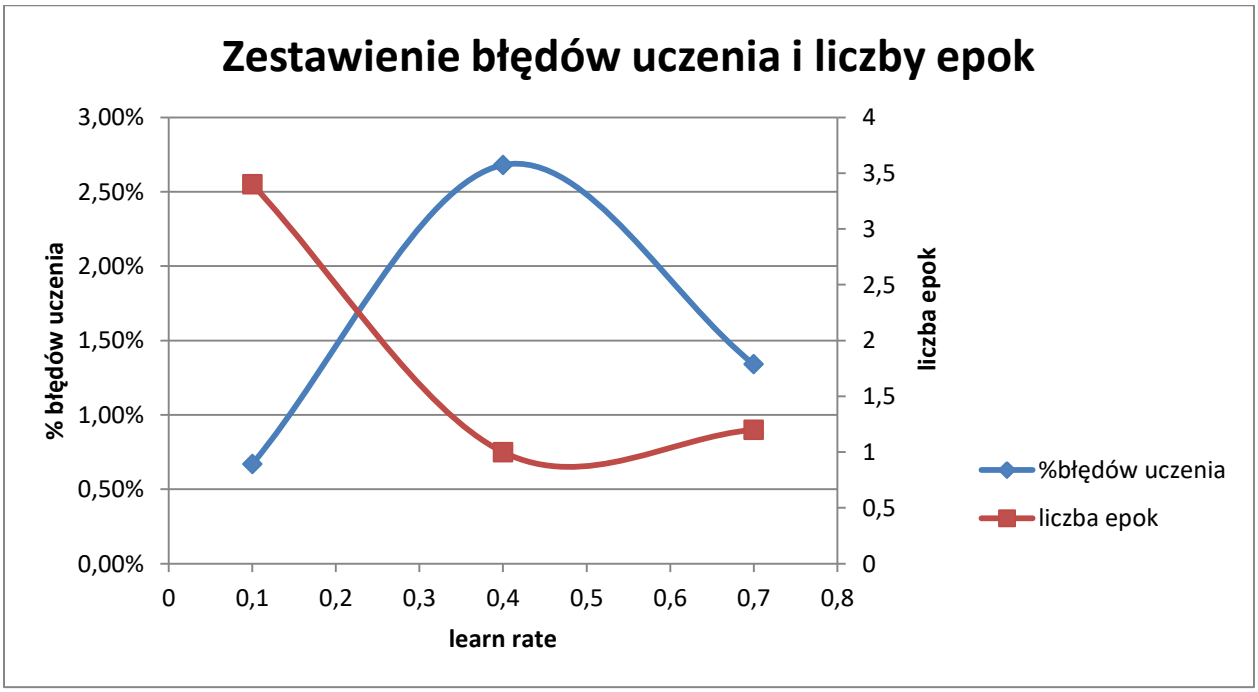


Na powyższym wykresie widzimy, że najmniejszy średni % błędów osiągany był przy learn rate 0,1. Następnie ilość błędów znacznie wzrosła przy 0,4. Spowodowane może być to faktem, iż sieć uczyła się mniej dokładnie.



Na powyższym wykresie widać jak wraz ze wzrostem learn rate średnia liczba epok spadła. Ma to związek oczywiście z tym, że przy wyższym kroku uczenia sieć jest w stanie szybciej dojść do rozwiązania.

Spoglądając na dwa powyższe wykresy zauważyć można pewną zależność. Przy learn rate 0,4 nasza sieć nauczyła się najszybciej jednak mylila się ona częściej niż w innych przypadkach. Zaś przy learn rate 0,1 sieć uczyła się najdłużej lecz średni % popełnianych błędów był najniższy. Może mieć to związek z faktem, że przy większej liczbie epok sieć była wstanie nauczyć się bardziej szczegółowo. Zależność tą przedstawiono na poniższym wykresie.



Widać odwrotną proporcjonalność liczby epok do % błędów uczenia.

5. Sformułowanie wniosków.

Na podstawie uzyskanych wyników, można wnioskować, że sieć Kohonena z algorytmem uczenia WTA świetnie nadaje się do problemu odróżniania kwiatów. Sieć szybko się uczyła a błędy przez nią popełniane były na bardzo niskim poziomie. Najlepsze wyniki osiągane były przy learn rate równym 0,1 co może oznaczać, że była to optymalna wartość. Dla wyższych współczynników osiągane były gorsze rezultaty.

Ważne było aby dane uczące były znormalizowane, ponieważ w przeciwnym razie zwycięzcą prawie zawsze w kolejnych epokach byłby ten sam neuron z powodu faktu, że po pierwszej modyfikacji wag, wektor wagowy zwycięzcy byłby dużo bliżej rozwiązania niż reszty neuronów, których to wektory początkowo znajdowały się w dużej odległości od wektorów uczących. Dzięki normalizacji wektory wszystkich wag znajdowały się w otoczeniu danych uczących od początku.

6. Listing kodu

```java
import java.util.Random;

/**
 *
 * @author Szysz
 */
public class WinnerTakesAllNeuron {

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

    public WinnerTakesAllNeuron(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 w zakresie od 0 do 1
        }
    }

    //uczenie poprzez zmniejszenie odległości między wektorem wag a zadanym wektorem
    public void learn(double[] x, double lr) {

        for (int i = 0; i < noi; i++) {
            w[i] += lr * (x[i] - w[i]); //modyfikacja wag
        }
    }

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

}
```

```
 *
 * @author Szysz
 */
public class LearningDataSet {

    //tablica z danymi uczącymi
    public static double[][][] flowerLearn = {
        {
            {0.809246635, 0.5446852351, 0.217874094, 0.0311248706},
            {0.8281328734, 0.5070201266, 0.2366093924, 0.0338013418},
            {0.8053330754, 0.5483118811, 0.2227517017, 0.0342694926},
            {0.8000302475, 0.5391508189, 0.2608794285, 0.0347839238},
            {0.7904706124, 0.5691388409, 0.2213317715, 0.0474282367},
            {0.7841749863, 0.5663486012, 0.2468699031, 0.058087036},
            {0.7801093557, 0.5766025673, 0.2374245865, 0.0508766971},
            {0.8021849185, 0.5454857446, 0.2406554756, 0.0320873967},
            {0.8064236562, 0.5315065006, 0.2565893451, 0.0366556207},
            {0.81803119, 0.5175299366, 0.2504177112, 0.0166945141},
            {0.8037351881, 0.5507074437, 0.2232597745, 0.0297679699},
            {0.7869910029, 0.5574519604, 0.2623303343, 0.0327912918},
            {0.8230721776, 0.514420111, 0.2400627185, 0.017147337},
            {0.802512599, 0.559892511, 0.2052939207, 0.0186630837},
            {0.8112086464, 0.5594542389, 0.1678362717, 0.0279727119},
            {0.7738111103, 0.5973278746, 0.2036345027, 0.0543025341},
            {0.794289441, 0.5736534852, 0.1912178284, 0.0588362549},
            {0.8032741237, 0.5512665555, 0.2205066222, 0.047251419},
            {0.806828203, 0.5378854687, 0.2406329728, 0.0424646423},
            {0.7796488324, 0.5809148163, 0.2293084801, 0.045861696},
            {0.8173378965, 0.5146201571, 0.2573100785, 0.0302717739},
            {0.7859185787, 0.5701762238, 0.2311525231, 0.0616406728},
            {0.775770746, 0.6071249316, 0.1686458143, 0.0337291629},
            {0.8059779151, 0.5215151215, 0.268659305, 0.0790174427},
            {0.7761140001, 0.5497474167, 0.3072117917, 0.0323380833},
            {0.8264745061, 0.4958847037, 0.264471842, 0.0330589802},
            {0.7977820578, 0.5424917993, 0.2552902585, 0.0638225646},
            {0.806419649, 0.5427824561, 0.2326210526, 0.0310161403},
            {0.8160942667, 0.5336000975, 0.2197176872, 0.031388241},
            {0.7952406381, 0.5414404345, 0.2707202172, 0.0338400272},
            {0.8084658442, 0.5221341911, 0.2694886147, 0.0336860768},
            {0.8222502813, 0.5177131401, 0.2284028559, 0.0609074282},
            {0.7657831085, 0.6037905278, 0.2208989736, 0.0147265982},
            {0.7786744728, 0.5946241429, 0.1982080476, 0.0283154354},
            {0.8176894181, 0.5173137135, 0.2503130872, 0.0333750783},
            {0.8251229525, 0.5280786896, 0.1980295086, 0.0330049181},
            {0.826997544, 0.5262711644, 0.1954721468, 0.030072638},
            {0.7852322109, 0.5769052978, 0.2243520603, 0.0160251472},
            {0.8021241325, 0.5469028176, 0.236991221, 0.0364601878}
        }, //setosa
```

```
    //tablica z danymi testujacymi
    public static double[][][] flowerTest = {
        {
            {0.8077956849, 0.5385304566, 0.2375869661, 0.0316782622},
            {0.8003330078, 0.5602331055, 0.208086582, 0.0480199805},
            {0.8609385733, 0.4400352708, 0.2487155878, 0.0573959049},
            {0.7860903755, 0.5717020913, 0.2322539746, 0.0357313807},
            {0.788894791, 0.5522263537, 0.2524463331, 0.0946673749},
            {0.766938972, 0.5714447242, 0.2857223621, 0.0601520762},
            {0.8221058465, 0.5138161541, 0.2397808719, 0.0513816154},
            {0.7772909267, 0.5791579454, 0.243855977, 0.0304819971},
            {0.7959478212, 0.5537028322, 0.2422449891, 0.034606427},
            {0.7983702483, 0.5573528148, 0.2259538439, 0.0301271792}
        }, //setosa

        {
            {0.747141937, 0.3396099714, 0.5433759542, 0.1765971851},
            {0.7326039145, 0.3602970072, 0.552455411, 0.1681386033},
            {0.7626299404, 0.341868594, 0.525951683, 0.1577855049},
            {0.7698687947, 0.3541396456, 0.5081134045, 0.1539737589},
            {0.7354428354, 0.3545885099, 0.5515821266, 0.1707278011},
            {0.7323961773, 0.3854716722, 0.5396603411, 0.1541886689},
            {0.7344604664, 0.3736728689, 0.5411813963, 0.1675085274},
            {0.7572810335, 0.3542120963, 0.5252110393, 0.1587847328},
            {0.7233711848, 0.3419572873, 0.5786969478, 0.1578264403},
            {0.7825805423, 0.3836179129, 0.4603414955, 0.1687918817}
        }, //versicolor

        {
            {0.6999703739, 0.3238668894, 0.5850498648, 0.2507356563},
            {0.690525124, 0.3214513508, 0.6071858849, 0.2262065061},
            {0.691935021, 0.3256164805, 0.6003553859, 0.2340368453},
            {0.6891487079, 0.3394314531, 0.5862906918, 0.2571450403},
            {0.7215572479, 0.3230853349, 0.5600145805, 0.2476987567},
            {0.7296535933, 0.2895450767, 0.5790901534, 0.2200542583},
            {0.7165389871, 0.3307103017, 0.5732311897, 0.2204735345},
            {0.6746707199, 0.3699807173, 0.5876164334, 0.2502810735},
            {0.7333788618, 0.3294890538, 0.542062637, 0.2444596206},
            {0.6902591586, 0.3509792332, 0.5966646964, 0.2105875399}
        } //virginica
    };

}
```

```java
/*
 * Copyright (C) 2018 Szysz
 */
package kohonen;

/**
 *
 * @author Szysz
 */
public class Main {

    private static double learningRate = 0.7;           //współczynnik uczenia się
    private static int numberOfInputs = 4;                      //ilość wejść
    private static int numberOfNeurons = 200;           //liczba neuronów
    private static int numberOfFlowers = 3;                     //liczba kwiatów
    private static int numberOfLearnSamples = 15;       //liczba danych uczących dla każdego kwiatu
    private static int numberOfTestSamples = 5;         //liczba danych testujących dla każdego kwiatu
    private static int learnLimit = 10000;                      //maksymalny próg epok uczenia

    public static void main(String[] args) {

        int successCounter = 0;         //licznik prób uczenia zakończonych powodzeniem
        int unsuccesCounter = 0;        //licznik prób uczenia zakończonych niepowodzeniem

        while (successCounter != 10 && unsuccesCounter != 100) {

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

            int ages = learn(kohonens);

            if (ages != learnLimit) {
                successCounter++;

                int winner;

                System.out.println("PO UCZENIU");
                for (int i = 0; i < numberOfFlowers; i++) {
                    winner = getWinner(kohonens, LearningDataSet.flowerLearn[i][0]);
                    System.out.println("Flower[" + i + "] winner = " + winner);
                }
                System.out.println();

                System.out.println("PO TESTOWANIU");
                for (int i = 0; i < numberOfFlowers; i++) {
                    for (int j = 0; j < numberOfTestSamples; j++) {
                        winner = getWinner(kohonens, LearningDataSet.flowerTest[i][j]);
                        System.out.println("Flower[" + i + "][" + j + "] test winner = " + winner);
                    }
                    System.out.println();
                }
                System.out.println();

                System.out.println("Ilość epok = " + ages + "\n\n\n");
            } else {
                unsuccesCounter++;
            }
        }
        System.out.println("\nIlość niepowodzeń = " + unsuccesCounter);
    }

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

        int counter = 0;
        int winner;

        int[][] winners = new int[numberOfFlowers][numberOfLearnSamples];
        for (int i = 0; i < numberOfFlowers; i++) {
            for (int j = 0; j < numberOfLearnSamples; j++) {
                winners[i][j] = - 1;
            }
        }

        while (!isUnique(winners)) {    //dopóki sieć się nauczy

            //uczymy sieć po kolei każdy kwiat z każdego gatunku
            for (int i = 0; i < numberOfFlowers; i++) {
                for (int j = 0; j < numberOfLearnSamples; j++) {
                    winner = getWinner(kohonens, LearningDataSet.flowerLearn[i][j]);
                    kohonens[winner].learn(LearningDataSet.flowerLearn[i][j], learningRate);
                }
            }
```

```
            //po zakończeniu epoki pobieramy zwycięzców
            for (int i = 0; i < numberOfFlowers; i++) {
                for (int j = 0; j < numberOfLearnSamples; j++) {
                    winners[i][j] = getWinner(kohonens, LearningDataSet.flowerLearn[i][j]);
                }
            }

            //jeśli ilość prób nauczenia 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 kwiaty danego gatunku mają tylko jednego zwycięzcę
        for (int i = 0; i < numberOfFlowers; i++) {
            for (int j = 1; j < numberOfLearnSamples; j++) {
                if (winners[i][0] != winners[i][j]) {
                    return false;
                }
            }
        }

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

        return true;
    }

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

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

        //sprawdza który neuron jest zwycięzcą
        //miarą zwycięztwa 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
    public static double distanceBetweenVectors(double[] vector1, double[] vector2) {

        double suma = 0.0;

        for (int i = 0; i < vector1.length; i++) {
            //suma += Math.pow( vector1[i] - vector2[i], 2 ); //miara Euklidesowa
            suma += Math.abs(vector1[i] - vector2[i]); //miara Manhattan
        }

        return Math.sqrt(suma);
    }

}
```

7. Bibliografia:

Stanisław Osowski – Sieci neuronowe do przetwarzania informacji, ISBN 83-7207-615-4

http://uni-obuda.hu/users/fuller.robert/winner.pdf

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

http://www.michalbereta.pl/dydaktyka/WdoSI/lab_neuronowe_II/Sieci_Neuronowe_2%20Sieci%20Kohonena.pdf