Scenariusz 5

Maciej Słaboń
Gr 4

Celem ćwiczenia było napisanie zapoznanie się z działaniem sieci Kohonena przy wykorzystaniu reguły WTA do odwzorowania istotnych cech kwiatów.

Syntetyczny opis algorytmu uczenia:

Zbudowałem SOM (self organizing map) z użyciem algorymtmu WTA. Sieć składa się ze zmiennej ilości neuronów, przebierające losowe wagi. Do uczenia wykorzystałem 105 (po 35 z każdego gatunku) natomiast do testowania wyników 45 (po 15 z każdego gatunku). Mapa składała się z siatki o różnej ilości neuronów, jednak do przedstawiania danych zawsze wypisywana była jako kwadrat.

Sieć Kohonena pomaga reprezentować wielowymiarowe dane w przestrzeni o mniejszym wymiarze. Algorytm polega na tym, że najpierw z zestawu danych losujemy losowy rekord. Następnie szukamy neuronu, który znajduje się “najbliżej” jego, będzie on tzw. “Zwycięzcą” . Obliczanie odległości odbywało się wg. Wzoru:

$$Dist = \sqrt{\sum_{i=0}^{i=n} (V_i - W_i)^2}$$

Equation 1

Następnie należało wyznaczyć nowe wagi tego neuronu wg wzoru.

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

Dane do nauki:

Dane do nauki znalazłem na stronie:
https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data

Przykładowe dane:

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
```

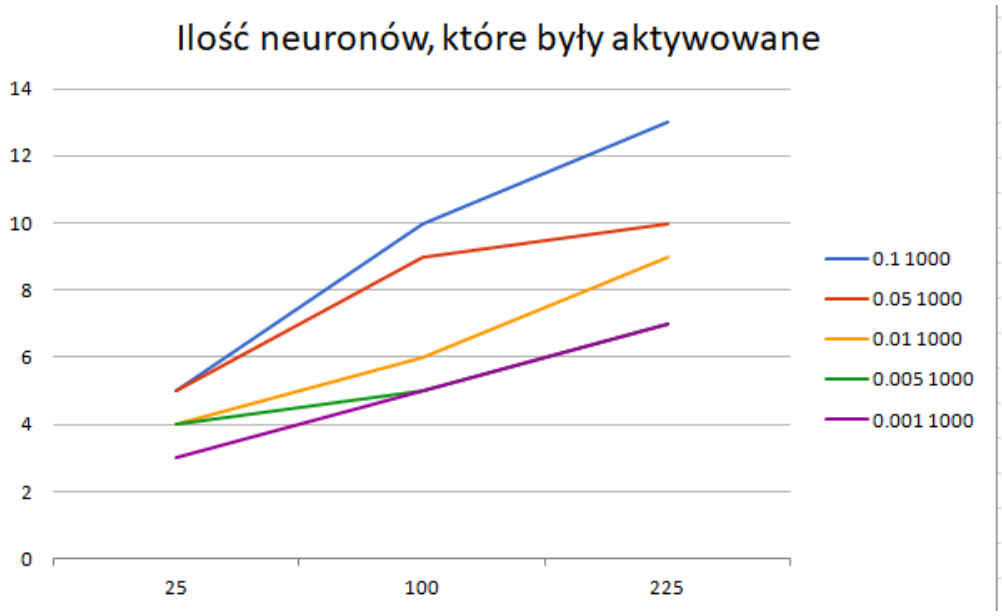
potrzebna była normalizacja danych

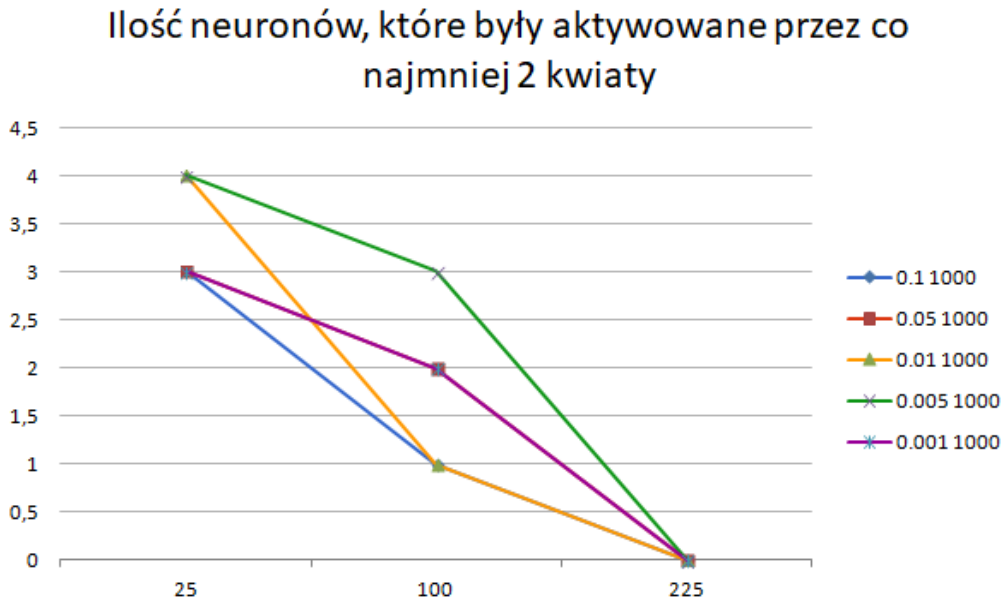
Przykładowy output:

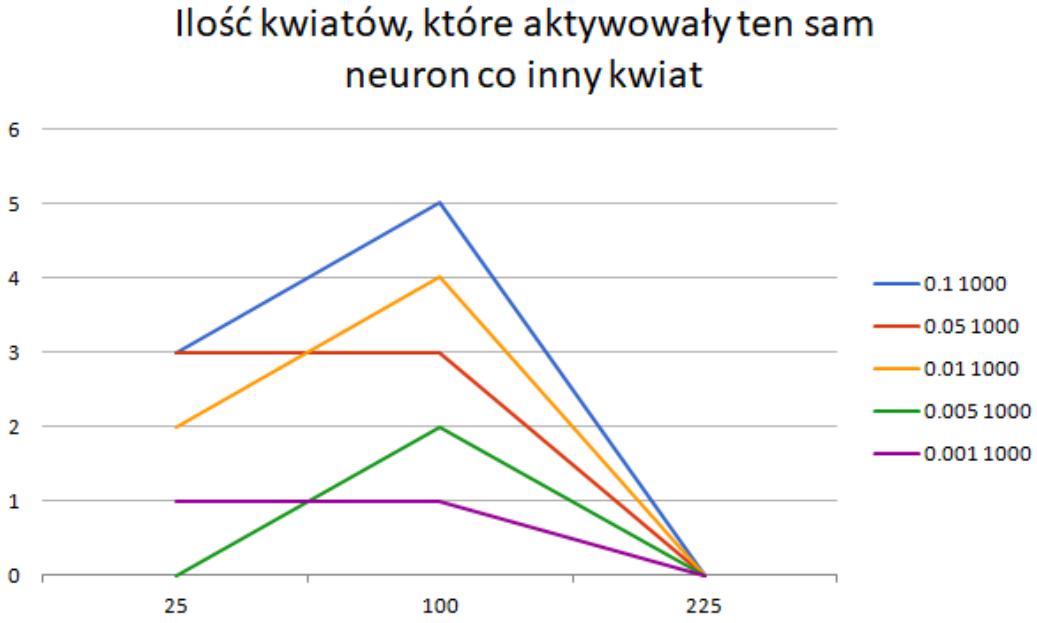
```
Neuron: 0 Count: 12
Neuron: 1 Count: 15
Neuron: 2 Count: 8
Neuron: 15 Count: 5
Neuron: 16 Count: 5
222222222333        111111111111111     33333333        #       #
#       #       #       #       #
#       #       #       #       #
23333       22222       #       #       #
#       #       #       #       #

Process finished with exit code 0
```

Lr = 0.1 Liczba neuronów = 25, Ilość epok uczenia = 1000
1-Iris- versicolor
2-Iris- setosa
3-Iris-virginica

Wyniki Testów:




Ilość neuronów, które były aktywowane przez co najmniej 2 kwiaty
4,5
4
3,5
3
2,5
2
1,5
1
0,5
0
25
100
225
0.1 1000
0.05 1000
0.01 1000
0.005 1000
0.001 1000


Ilość kwiatów, które aktywowały ten sam neuron co inny kwiat
6
5
4
3
2
1
0
25
100
225
0.1 1000
0.05 1000
0.01 1000
0.005 1000
0.001 1000

Analiza wyników:

Wyniki wskazują na to, że wielkość siatki ma bardzo duży wpływ, im większa siatka tym więcej neuronów było aktywowanych, można zauważyć, że przy mniejszych sieciach tworzą się skupiska neuronów, w których znajduje się większość kwiatów danego gatunku, oraz neurony przejściowe, mniejsze i zawierające czasami po 2 różne garunki.

Wpływy ustawień sieci na naukę:

- Mniejszy współczynnik uczenia lepiej sprawdził się od większego. Miał on również największy wpływ na jakość nauki oraz powodował najmniejszą ilość neuronów przejściowych.
- Sposób korygowania wartości współczynnika uczenia w kolejnych epokach nie miał aż tak dużego wpływu. Jednak można zaobserwować, że powodował poprawę uczenia.

Ważne jest by wagi były losowe co minimalizuje pojawianie się tych samych kwiatów w neuronach

Dla dużych siatek o ile ilość aktywowanych neuronów była większa to ilość kwiatów z różnych gatunków, które aktywowały ten sam neuron była mniejsza.

Wnioski:

- Zmiana współczynnika uczenia podczas nauki zwiększała jakość uczenia
- Ważne dla jakości nauczania było losowe dobieranie danych wejściowych z setu
- Im większa sieci tym lepsze wyniki się pojawiały
- Ilość aktywowanych neuronów nie świadczy o jakości uczenia
- Kiedy neuron był aktywowany przez 2 gatunki kwiatów oznaczało to, ze są do siebie bardzo podobne co pokazuje przykład kwiatów 2 i 3
- Mapa sprowadziła wielowymiarowy problem do mapy 2d

Kod

```java
public class Main {

    public static void main(String[] args) {
        Map m = new Map( learningRate: 0.1,  numberOfNeurons: 25,  iter: 1000);      //stworzenie mapy (learning rate, liczba neuronów, liczba cykli)
        m.learn();                          //nauczanie
        m.test(DataProvider.testInput);     //testowanie
        m.print(DataProvider.testInput); //wyświetlanie mapy dając dane testowe
    }
}
```

```java
import java.util.Random;

public class DataProvider {
    public static double[][] input = new double[][]{
            {0.6375, 0.4375, 0.175, 0.025},
            {0.6125, 0.375, 0.175, 0.025},
            {0.5875, 0.4, 0.1625, 0.025},
            {0.575, 0.3875, 0.1875, 0.025},
            {0.625, 0.45, 0.175, 0.025},
            {0.675, 0.4875, 0.2125, 0.05},
            {0.575, 0.425, 0.175, 0.0375},
            {0.625, 0.425, 0.1875, 0.025},
            {0.55, 0.3625, 0.175, 0.025},
            {0.6125, 0.3875, 0.1875, 0.0125},
            {0.675, 0.4625, 0.1875, 0.025},
            {0.6, 0.425, 0.2, 0.025},
            {0.6, 0.375, 0.175, 0.0125},
            {0.5375, 0.375, 0.1375, 0.0125},
            {0.725, 0.5, 0.15, 0.025},
            {0.7125, 0.55, 0.1875, 0.05},
            {0.675, 0.4875, 0.1625, 0.05},
            {0.6375, 0.4375, 0.175, 0.0375},
            {0.7125, 0.475, 0.2125, 0.0375},
            {0.6375, 0.475, 0.1875, 0.0375},
            {0.675, 0.425, 0.2125, 0.025},
            {0.6375, 0.4625, 0.1875, 0.05},
            {0.575, 0.45, 0.125, 0.025},
            {0.6375, 0.4125, 0.2125, 0.0625},
            {0.6, 0.425, 0.2375, 0.025},
            {0.625, 0.375, 0.2, 0.025},
            {0.625, 0.425, 0.2, 0.05},
            {0.65, 0.4375, 0.1875, 0.025},
```

```
        {0.8, 0.3875, 0.6875, 0.225},
        {0.75, 0.375, 0.6, 0.225},
        {0.8625, 0.3875, 0.675, 0.2625},
        {0.8375, 0.3875, 0.7, 0.3},
        {0.8625, 0.3875, 0.6375, 0.2875},
        {0.725, 0.3375, 0.6375, 0.2375},
        {0.85, 0.4, 0.7375, 0.2875},
        {0.8375, 0.4125, 0.7125, 0.3125},
        {0.8375, 0.375, 0.65, 0.2875},
        {0.7875, 0.3125, 0.625, 0.2375},
        {0.8125, 0.375, 0.65, 0.25},
        {0.775, 0.425, 0.675, 0.2875},
        {0.7375, 0.375, 0.6375, 0.225}
};

public static double[] getInput(int number, double[][] data) {
    double[] result = new double[input[0].length];

    for (int i = 0; i < result.length; i++) {
        result[i] = data[number][i];
    }
    return result;
}

public static void restoreInputToCorrectOrder() {
    double[][] correct = new double[][]{
            {0.6375, 0.4375, 0.175, 0.025},
            {0.6125, 0.375, 0.175, 0.025},
            {0.5875, 0.4, 0.1625, 0.025},
            {0.575, 0.3875, 0.1875, 0.025},
            {0.625, 0.45, 0.175, 0.025},
            {0.675, 0.4875, 0.2125, 0.05},
            {0.575, 0.425, 0.175, 0.0375},
            {0.625, 0.425, 0.1875, 0.025}
```

```
            {0.8, 0.35, 0.7, 0.2625},
            {0.9, 0.375, 0.725, 0.2},
            {0.925, 0.35, 0.7625, 0.2375},
            {0.9875, 0.475, 0.8, 0.25},
            {0.8, 0.35, 0.7, 0.275},
            {0.7875, 0.35, 0.6375, 0.1875},
            {0.7625, 0.325, 0.7, 0.175}
    };
    System.arraycopy(correct, srcPos: 0, DataProvider.input, destPos: 0, correct.length);
}

public static void randomise() {    //pomieszaj dane wejsciowe
    int inputDataCount = input.length;
    int inputDataAttributionsCount = input[0].length;
    Random r = new Random();
    int place;

    double[] tmp = new double[inputDataAttributionsCount];
    for (int i = 0; i < inputDataCount; i++) {

        for (int j = 0; j < inputDataAttributionsCount; j++) {
            tmp[j] = input[i][j];
        }

        place = r.nextInt( bound: inputDataCount - i) + i;


        for (int j = 0; j < inputDataAttributionsCount; j++) {
            input[i][j] = input[place][j];
        }

        for (int j = 0; j < inputDataAttributionsCount; j++) {
            input[place][j] = tmp[j];
        }
    }
}
```

```java
public class Neuron {
    double[] weights;

    public Neuron(double[] weights)                    //kopiuje dostarczona tablice wag
    {
        this.weights = new double[weights.length];
        System.arraycopy(weights,  srcPos: 0, this.weights,  destPos: 0, weights.length);
    }

    public double calculateDistance(double[] input)    //liczy dystans neuronu od danych wejsciowych
    {
        double sum = 0.0;
        for (int i = 0; i < input.length; i++) {
            sum += Math.pow((input[i] - weights[i]), 2);
        }
        sum += Math.pow((1 - weights[input.length]), 2);
        return Math.sqrt(sum);
    }

    public void updateWeights(double[] input, double lr)      //oblicza nowe wagi dla zwycięzcy
    {
        for (int i = 0; i < input.length; i++) {
            weights[i] += lr * (input[i] - weights[i]);
        }

        weights[input.length] += lr * (1 - weights[input.length]);
    }
}
```

```java
import java.util.Arrays;
import java.util.Random;

public class Map {
    private Neuron[] neuronMap;
    private int numberOfNeurons;
    private double lr;
    private double Max;

    public Map(double learningRate, int numberOfNeurons, int iter)      //tworzy tablice neuronów i losuje początkowe wagi
    {
        this.Max = iter;
        this.numberOfNeurons = numberOfNeurons;
        lr = learningRate;
        neuronMap = new Neuron[this.numberOfNeurons];
        Random r = new Random();
        double[] weights = new double[DataProvider.input[0].length + 1];
        for (int j = 0; j < neuronMap.length; j++) {
            for (int i = 0; i < weights.length; i++) {
                weights[i] = r.nextDouble();
            }

            neuronMap[j] = new Neuron(weights);
        }
    }

    public void learn() {

        double lenght;
        double min;
        int counter = 0;
        int neuronNumber = 0;
        do {
            DataProvider.randomise();                               //miesza dane wejściowe
            for (int i = 0; i < DataProvider.input.length; i++)       //dla każdego rekordu
            {
                min = 0.0;
                for (int j = 0; j < neuronMap.length; j++)                  //szuka najbliższego neuronu
                {
                    double[] input = DataProvider.getInput(i, DataProvider.input);
                    lenght = neuronMap[j].calculateDistance(input);
                    if (lenght < min || j == 0) {
                        min = lenght;
                        neuronNumber = j;
                    }
                    neuronMap[neuronNumber].updateWeights(input, lr); //oblicza nowe wagi dla zwycięzcy
                }
            }
            counter++;
        } while (Max > counter);
        DataProvider.restoreInputToCorrectOrder();
    }
```

```java
public void test(double[][] inputArray)                                 //testowanie, wypisuje który neuron ile razy został aktywowany
{
    int neuronNumber;
    int[] responseCounter = new int[neuronMap.length];
    Arrays.fill(responseCounter,  val: 0);
    for (int i = 0; i < inputArray.length; i++) {
        double[] input = DataProvider.getInput(i, inputArray);
        neuronNumber = setClass(input);
        responseCounter[neuronNumber]++;
    }
    int n;
    for (int i = 0; i < responseCounter.length; i++) {
        n = responseCounter[i];
        if (n != 0)
            System.out.println("Neuron: " + i + " Count: " + n);
    }
}

public void print(double[][] inputArray) {
    int size = inputArray.length / 3;
    int mapS = (int) Math.sqrt(numberOfNeurons);
    int beg = 0;
    int end = size;
    String[] map = new String[numberOfNeurons];
    for (int i = 0; i < map.length; i++) {
        map[i] = "";
    }

    map = calculate(map, beg, end,  type: "A", inputArray);
    beg += size;
    end += size;
    map = calculate(map, beg, end,  type: "B", inputArray);
    beg += size;
    end += size;
    map = calculate(map, beg, end,  type: "C", inputArray);

    for (int i = 0; i < mapS; i++) {
        for (int j = 0; j < mapS; j++) {
            int k = i * mapS + j;
            if (map[k] == "") {
                System.out.print(".\t\t");
            } else {
                System.out.print(map[k] + "\t\t");
            }
        }
        System.out.println();
    }
}
```

```java
private String[] calculate(String[] map, int beg, int end, String type, double[][] inputArray) //metoda do jakie dane aktywuje neuron
{
    int nn;
    for (int i = beg; i < end; i++) {
        double[] input = DataProvider.getInput(i, inputArray);
        nn = setClass(input);
        map[nn] += type;
    }
    return map;
}


private int setClass(double[] input)       //który neuron jest aktywowany dla danego inputu
{
    int classification = 0;
    double lenght;
    double min = 0.0;
    for (int j = 0; j < neuronMap.length; j++) {
        lenght = neuronMap[j].calculateDistance(input);
        if (lenght < min || j == 0) {
            min = lenght;
            classification = j;
        }
    }
    return classification;
}
```