Michał Radziejowski – Scenariusz 5 Sprawozdanie

Budowa i działanie sieci Kohonena dla WTA.

Cel:

Celem ćwiczenia jest poznanie budowy i działania sieci Kohonena przy wykorzystaniu reguły Winner-Takes-All do odwzorowywania istotnych cech kwiatów.

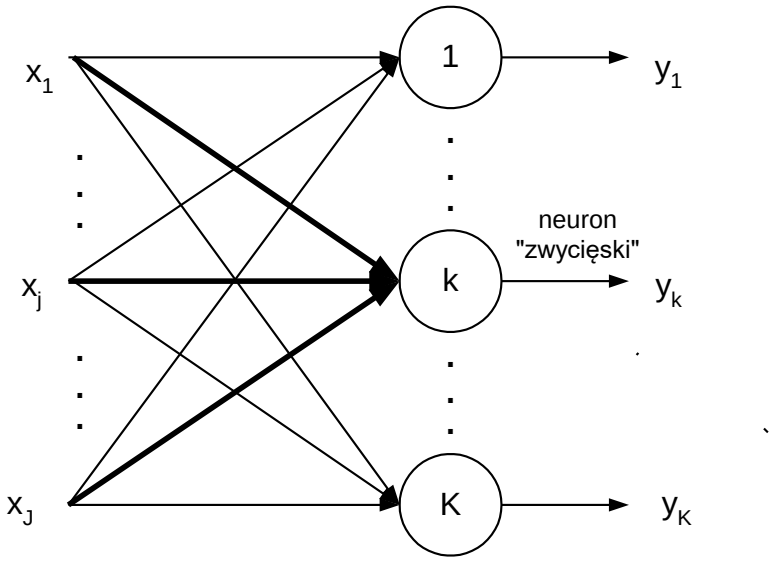
Wykonanie zadania:

1) Stworzono pliki .txt zawierające dane uczące i testujące w postaci zaprezentowanej nam w instrukcji.
2) Sieć Kohonena i algorytm Winner-Takes-All został zaimplementowana w języku JAVA.
3) Uczono sieć w oparciu o różne współczynniki uczenia
4) Przetestowano poprawne działanie sieci, bazując na zastosowaniu algorytmu Winner-Takes-All.

Opis ćwiczenia i algorytmu:

Sieci Kohonena są szczególnym przypadkiem algorytmu realizującego uczenie się bez nadzoru. Ich głównym zadaniem jest organizacja wielowymiarowej informacji (takiej jak na przykład obiekty opisane 10 parametrami) w taki sposób, aby można było ją prezentować i analizować w przestrzeni o znacznie mniejszej liczbie wymiarów, czyli mapie (np. na dwuwymiarowym ekranie). Warunkiem jest 'podobieństwo' danych wejściowych do mapy. Topologia sieci Kohonena odpowiada topologii docelowej przestrzeni. Jeśli np. chcemy prezentować wynik na ekranie, rozsądnym modelem jest prostokątna siatka węzłów.

W skład warstwy sieci jednowarstwowej, wchodzi 30 neuronów, z których każdy posiada 4 wejścia i jedno wyjście. Dzięki temu neurony bez problemu pracują na rekordach składających się z czterech liczb zmiennoprzecinkowych. Stosując sieć Kohonena, wiemy iż używamy algorytmu uczenia bez nauczyciela. W konkurencyjnej metodzie uczenia sieci, tylko jeden element wyjściowy może znajdować się w stanie aktywnym. Nazywany jest on zwycięzcą, a schemat takiej reguły aktywacji neuronów określany jest mianem Winner-Takes-All (WTA). Neurony w pewnym sensie rywalizują o to, który będzie poddawany nauce dla zestawu uczącego. Każda komórka jest połączona ze wszystkimi elementami wzorca wejściowego za pomocą wag. Neuron zwycięski warstwy wyjściowej sieci otrzymuje najsilniejszy sygnał pobudzenia.



W zależności od aktualnych wartości wag, sygnały wyjściowe różnią się między sobą. Wartości tych sygnałów są porównywane i zwycięża ten neuron, którego wartość jest największa. Neuron „zwycięzca” przyjmuje na swoim wyjściu stan 1, a pozostałe neurony (neurony „przegrywające”) stan 0.

Pliki tekstowe zawierające dane testowe oraz uczące, zawierają odpowiednio 45 oraz 150 rekordów opisujących 3 rodzaje kwiatów.

Przykłady działania programu:

Na samym początku, program wczytuje dane z pliku input.txt. Po pozytywnym wczytaniu (sprawdzeniu czy plik istnieje) program wyświetla listę rekordów wraz z komentarzem.

```
Dane zaladowano pomyslnie
5.1 3.5 1.4 0.2 setosa
4.9 3.0 1.4 0.2 setosa
4.7 3.2 1.3 0.2 setosa
4.6 3.1 1.5 0.2 setosa
5.0 3.6 1.4 0.2 setosa
5.4 3.9 1.7 0.4 setosa
```

Po poprawnym wczytaniu danych, program rozpoczyna uczenie sieci neuronowej na zadanej wcześnie liczbie epok. Wyświetla ich wartość, oraz komunikat o zakończeniu nauki.

```
6.7 3.3 5.7 2.5 virginica
6.7 3.0 5.2 2.3 virginica
6.3 2.5 5.0 1.9 virginica
6.5 3.0 5.2 2.0 virginica
6.2 3.4 5.4 2.3 virginica
5.9 3.0 5.1 1.8 virginica
Uczenie zakonczone.
Liczba epok uczenia = 10000
```

Następnie, program testuje oraz wypisuje konkretną nazwę kwiatu, wraz z przyznaną mu poprzez algorytm grupom.

```
ID #10. Rodzaj setosa, grupa 24
ID #11. Rodzaj setosa, grupa 24
ID #12. Rodzaj setosa, grupa 24
ID #13. Rodzaj setosa, grupa 24
ID #14. Rodzaj setosa, grupa 24
ID #15. Rodzaj virginica, grupa 6
ID #16. Rodzaj virginica, grupa 6
ID #17. Rodzaj virginica, grupa 6
ID #18. Rodzaj virginica, grupa 6
```

Na końcu, wyświetla się output określający ilość oraz ID zwycięskich grup.

```
Lista zwycieskich grup
24
6
16
```

Wyniki i wnioski:

Współczynnik nauki	Pomyłki w przypisaniu do grup
0.1	0/45
0.2	15/45
0.4	15/45
0.6	4/45
0.8	15/45
0.9	7/45
0.05	0/45

Na podstawie powyższej tabeli można śmiało stwierdzić, iż wysoka wartość współczynnika nauki wpływa negatywnie na naukę neuronów metodą Winner Takes All. Dla każdej wartości zostało wykonane pięć pomiarów, które miały za zadanie wykluczyć potencjalne błędy grube mogące się trafić przy korzystaniu z algorytmu nauki bez nadzoru nauczyciela. Jednakże ilość pomyłek była na tyle podobna, że z dokładnością do liczb całkowitych można było określić ich ilość pomyłek. Bardzo częstym błędem było przyznanie jednakowej grupy rodzajowi Virginica oraz Versicolor (grupy posiadające 15/45 nieprawidłowych przypisań).

Przy użyciu współczynników 0.6 oraz 0.9 została utworzona dodatkowa grupa neuronów dla rodzaju Versicolor, skutkiem czego zestaw testowy został podzielony pomiędzy 4 grupy neuronów (dla poprawnej liczby równej 3).

Sieci kohonena znajdują szerokie zastosowanie. Jako iż są to sieci nie wymagające obecności wzorca sprawdzającego (nauczyciela) można je wykorzystywać w sytuacjach gdy wynik końcowy testu nie jest znany. Aby dążyć do stworzenia wzorca mapy, zgodnego ze strukturą analizowanych danych, sieci w pewnym sensie rozpinają się wokół zbiorów,

dopasowując do nich swój kształt i właśnie strukturę. Przez te właściwości, sieci kohonena są stosowane przykładowo w klasyfikacji wzorców tekstu czy muzyki, oraz innych zagadnień których rekordy zawierają więcej niż jeden parametr.

Listing kodu:

Główna klasa programu:

```
public class NeuralNetwork
{
    final static double learningRate = 0.05; // wspolczynnik uczenia
    private final static int Epochs = 10000;

    private final static int inputNumber = 150;
    private final static int testNumber = 45;
    final static int input_tabSize = 4;

    private Data[] learning_tabData;
    private Data[] testing_tabData;

    private Layer layer;

    NeuralNetwork(int neuronsCount)
    {
        try {
            Scanner scanner = new Scanner(new File("input.txt"));
            Scanner scanner2 = new Scanner(new File("testinput.txt"));

            layer = new Layer(neuronsCount);
            learning_tabData = new Data[inputNumber];
            testing_tabData = new Data[testNumber];
            loadData(scanner, false);
            loadData(scanner2, true);
            scanner.close();
            showData();

        } catch (FileNotFoundException e) {
            System.out.println("Nie znaleziono plikow");
        }
    }

    private void loadData(Scanner s, boolean isTest) {
        int inputsCount = isTest ? testNumber : inputNumber;
        Data[] loader = new Data[inputsCount];
        for(int i = 0; i < inputsCount; i++) {
            loader[i] = new Data(input_tabSize);
            for(int j = 0; j < input_tabSize; j++) {
                loader[i].setXi(j, Double.parseDouble(s.next()));
            }
            loader[i].setY(s.next());
        }
        if (isTest) {
            testing_tabData = loader;
        } else {
            learning_tabData = loader;
        }
    }


    private void showData() {
        System.out.println("Dane zaladowano pomyslnie");
        for (int i = 0; i < inputNumber; i++) {
            for(int j = 0; j < input_tabSize; j++) {
                System.out.print(learning_tabData[i].getXi(j) + " ");
            }
```

```
            System.out.println(learning_tabData[i].getY());
        }

    }

    public void learn() {
        normalize(learning_tabData);
        normalize(testing_tabData);

        int iterations = 0;
        ArrayList<Double> result;

        while (iterations < Epochs) {
            for (int i = 0; i < inputNumber; i++) {
                result = layer.compute(learning_tabData[i]);
                layer.modify(result.indexOf(Collections.max(result)));
                result.clear();
            }
            ++iterations;
        }

        System.out.println("Uczenie zakonczone. \nLiczba epok uczenia = " + iterations +
"\n\n");
    }

    public void test(){
        ArrayList<Double> result;
        ArrayList<Integer> group = new ArrayList<>();
        int winner;
        for(int i = 0; i < testNumber; i++) {

            result = layer.compute(testing_tabData[i]);
            winner = result.indexOf(Collections.max(result));

            if(!group.contains(winner)){
                group.add(result.indexOf(Collections.max(result)));
            }
            System.out.println("ID #" + i + ". Rodzaj " + testing_tabData[i].getY() + ", grupa
" + winner);

        }

        System.out.println("\nLista zwycieskich grup");
        for(Integer el: group)
            System.out.println(el.toString());
    }

    private void normalize(Data[] data){
        for (Data aData : data) {
            double lenght = aData.getXi(0) * aData.getXi(0) +
                    aData.getXi(1) * aData.getXi(1) +
                    aData.getXi(2) * aData.getXi(2) +
                    aData.getXi(3) * aData.getXi(3);
            lenght = Math.sqrt(lenght);

            aData.setXi(0, aData.getXi(0) / lenght);
            aData.setXi(1, aData.getXi(1) / lenght);
            aData.setXi(2, aData.getXi(2) / lenght);
            aData.setXi(3, aData.getXi(3) / lenght);
        }
    }
}
```

Klasa Neuron

```
public class Neuron {
    private double[] weight;
```

```
    Neuron() {
        weight = new double[NeuralNetwork.input_tabSize];
        for (int i = 0; i < NeuralNetwork.input_tabSize; i++) {
            weight[i] = ThreadLocalRandom.current().nextDouble(0.01, 0.1);
        }
        normalizeWeights();
    }

    public double compute(Data data) {

        double signal = signalF(data);
        return activationFunction(signal);
    }

    private double activationFunction(double signal) {
        return signal;
    }

    public void modifyWeights(Data data) {
        for (int i = 0; i < NeuralNetwork.input_tabSize; i++) {
            weight[i] = weight[i] + NeuralNetwork.learningRate * (data.getXi(i) - weight[i]);
        }
        normalizeWeights();
    }

    private double signalF(Data data) {
        double signal = 0;
        for (int i = 0; i < NeuralNetwork.input_tabSize; i++) {
            signal += data.getXi(i) * weight[i];
        }

        return signal;
    }

    private void normalizeWeights() {
        double lenghtSquared = weight[0] * weight[0] + weight[1] * weight[1] + weight[2] *
weight[2] + weight[3] * weight[3];
        double lenght = Math.sqrt(lenghtSquared);

        weight[0] /= lenght;
        weight[1] /= lenght;
        weight[2] /= lenght;
        weight[3] /= lenght;
    }
}
```

Klasa Data

```
public class Data {
    private double x [];
    private int xCount;
    private String y;

    public Data(double[] x, String y, int xCount) {
        this.x = x;
        this.y = y;
        this.xCount = xCount;
    }

    public Data(int xCount){
        this.x = new double[xCount];
        this.xCount = xCount;

    }

    public void setXi(int i, double x){
```

```
        if (i < xCount)
            this.x[i] = x;
    }

    public double getXi(int i){
        if(i < xCount)
            return this.x[i];
        else return 0;
    }

    public String getY() {
        return y;
    }

    public void setY(String y) {
        this.y = y;
    }

}
```

Klasa Layer

```
public class Layer {

    public Neuron[] neurons;
    private int neuronCount;
    private Data data;


    public Layer(int neuronCount){

        this.neuronCount = neuronCount;

        neurons = new Neuron[neuronCount];
        for(int i = 0; i < neuronCount; i++){
            neurons[i] = new Neuron();
        }
    }


    public ArrayList<Double> compute(Data input){
        ArrayList<Double> results = new ArrayList<>();
        this.data = input;

        for (int i = 0; i < neurons.length; i++){
            results.add(neurons[i].compute(input));
        }

        return results;
    }

    public void modify(int id) {
        neurons[id].modifyWeights(data);
    }
}
```

Klasa Main

```
public class Main {

    public static void main(String[] args) {
        NeuralNetwork network = new NeuralNetwork(30);
        network.learn();
        network.test();     }
}
```