Michał Radziejowski – Scenariusz 6 Sprawozdanie

Budowa i działanie sieci Kohonena dla WTM

Cel:

Celem ćwiczenia jest poznanie budowy i działania sieci Kohonena przy wykorzystaniu reguły WTM do odwzorowywania istotnych cech liter alfabetu.

Wykonanie zadania:

1) Stworzono pliki .txt zawierające dane uczące i testujące w postaci zaproponowanej w instrukcji
2) Sieć Kohonena i algorytm Winner-Takes-Most został zaimplementowany 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:

Tak jak w przypadku scenariusza 5, korzystamy w tym zadaniu z sieci Kohonena. Realizują one sposób uczenia bez nadzoru nauczyciela, a ich głównym zadaniem jest organizacja wielowymiarowej informacji w taki sposób, aby była ona możliwa do zaprezentowania i przeanalizowania w przestrzeni o mniejszej liczbie wymiarów, czyli mapie. Warunkiem podstawowym jest swego rodzaju podobieństwo danych wejściowych do mapy. Jedną z ważniejszych cech sieci Kohonena, jest brak bezpośredniego powiązania ze sobą kolejnych neuronów. Połączenia między nimi służą jedynie ukazaniu odległości pomiędzy konkretnymi neuronami wyjściowej warstwy.

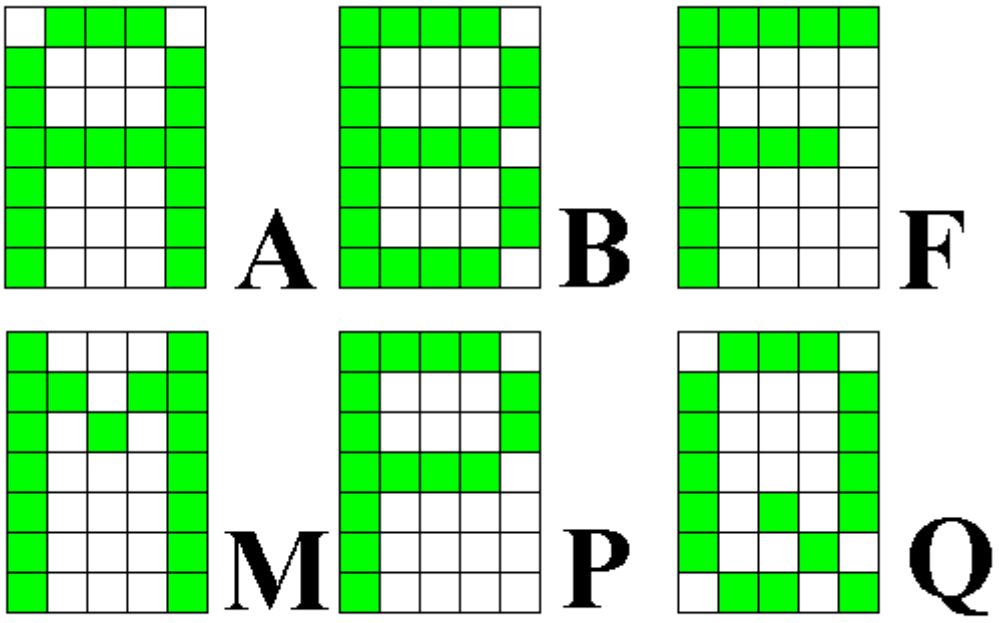
Algorytm WTM – Winner-Takes-Most wykazuje pewne podobieństwo do algorytmu WTA użytego w poprzednim scenariuszu, aczkolwiek z pewnymi zmianami. W tym przypadku, poza zwycięskim neuronem również neurony które z nim sąsiadują zmieniają wagi. W każdej iteracji zostaje zmniejszany początkowy współczynnik uczenia oraz promień sąsiedztwa.

Dane zostają odwzorowane na mapie za pomocą wcześniej zadeklarowanej liczby neuronów. Oblicza się ją, jako iloczyn pierwiastka wartości oznaczającego liczbę wektorów danych, oraz piątki. W naszym przypadku wygląda to następująco:

$$M = 5\sqrt{35}$$

co w przybliżeniu daje nam wartość M = 30~, taka ilość neuronów tworzy siatkę sieci.

Dane wejściowe oraz testowe znajdują się w plikach .txt. Ich reprezentacją są wektory stworzone odpowiednio z cyfr -1 oraz 1, tworzących obraz litery na siatce 5x7 tak jak sugerowano w instrukcji zadania.



rysunek matryc 5 x 7 przykładowych liter A, B, F, M, Q

Współczynnik uczenia jest obliczany każdorazowo przy iteracji programu z poniższego wzoru:

$$L(t) = L_0 \, e^{-\frac{t}{\lambda}}$$

Gdzie **λ** określa liczbę całkowitą epok uczenia, T jest numerem epoki natomiast wartość L_0 opisuje początkową wartość współczynnika uczenia.

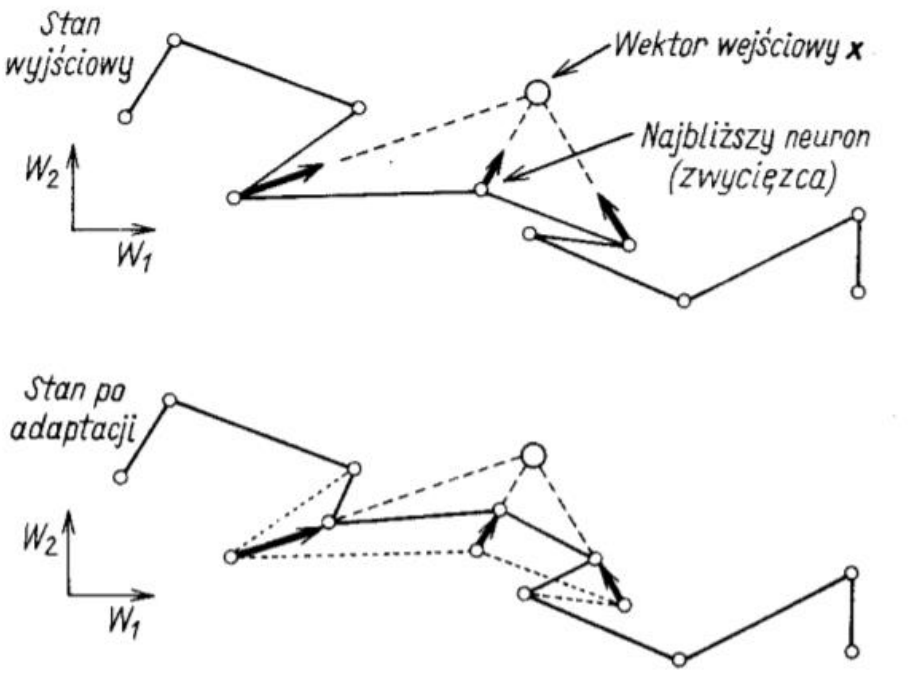
W każdej iteracji jest również wyłaniany neuron zwycięski, wraz z sąsiedztwem. Wagi zostają zmodyfikowane według następującego wzoru:

$$W(t+1) = W(t) + \Theta(t) \times L(t) \times (X(t) - W(t))$$

Gdzie symbol **θ** jest opisywany jako funkcja theta, mająca postać:

$$\Theta = e^{\left(\frac{dist^2}{2\times(\sigma(t))^2}\right)}$$

Parametr dist, jest to odległość pomiędzy neuronem zwycięskim a neuronem dla którego obliczamy zmianę wag.



Ilustracja procesu adaptacji wag

Przykłady działania programu:

Program prezentuje nam dane wczytane z pliku input.txt

```
Wczytano 20 liter

A
01110
10001
10001
11111
10001
10001
10001
```

Po przeprowadzeniu nauki, program wypisuje na ekranie liczbę epok przez które przeszła sieć, promień sąsiedztwa oraz wartość współczynnika uczenia po zakończeniu trenowania sieci:

```
Liczba epok = 10000
Promień  = 11
Współczynnik uczenia po zakończeniu = 0.25754136166851255
```

Następnie, program wypisuje poszczególne litery z inputu testowego, wraz z przypisanym im numerem zwycięskiego neuronu:

```
Litera: A Numer neuronu 12
Litera: B Numer neuronu 12
Litera: C Numer neuronu 29
Litera: D Numer neuronu 29
Litera: E Numer neuronu 12
Litera: F Numer neuronu 11
Litera: G Numer neuronu 29
Litera: H Numer neuronu 12
Litera: I Numer neuronu 1
Litera: J Numer neuronu 29
Litera: K Numer neuronu 11
Litera: L Numer neuronu 13
Litera: M Numer neuronu 12
Litera: N Numer neuronu 12
Litera: O Numer neuronu 29
Litera: P Numer neuronu 12
Litera: Q Numer neuronu 29
Litera: R Numer neuronu 12
Litera: S Numer neuronu 23
Litera: T Numer neuronu 1
```

Na końcu, program pokazuje nam pogrupowane litery, wraz z przyporządkowanymi im numerami zwycięskich neuronów:

```
Neuron o numerze 12 Litery: A, B, E, H, M, N, P, R,
Neuron o numerze 29 Litery: C, D, G, J, O, Q,
Neuron o numerze 11 Litery: F, K,
Neuron o numerze 1 Litery: I, T,
Neuron o numerze 13 Litery: L,
Neuron o numerze 23 Litery: S,
```

Wyniki i wnioski:

Learning rate	0.1	0.3	0.5	0.7	0.9
Grupy liter	Nr 23 Litery: A, Nr 15 Litery: B, Nr 11 Litery: C, Nr 16 Litery: D, Nr 24 Litery: E, Nr 28 Litery: F, Nr 13 Litery: G, Nr 29 Litery: H, K, R, Nr 0 Litery: I, T, Nr 7 Litery: J, Nr 17 Litery: L, Nr 27 Litery: M, N, Nr 12 Litery: O, Q, Nr 26 Litery: P, Nr 4 Litery: S,	Nr 0 Litery: A, H, M, N, Nr 16 Litery: B, Nr 17 Litery: C, D, G, J, O, Q, Nr 12 Litery: E, Nr 5 Litery: F, K, P, R, Nr 29 Litery: I, T, Nr 15 Litery: L, Nr 20 Litery: S,	Nr 0 Litery: A, H, K, M, N, Nr 16 Litery: B, Nr 17 Litery: C, D, G, L, O, Q, Nr 12 Litery: E, Nr 5 Litery: F, R, Nr 29 Litery: I, T, Nr 18 Litery: J, S, Nr 6 Litery: P,	Nr 12 Litery: A, B, E, H, M, N, P, R, Nr 29 Litery: C, D, G, J, O, Q, Nr 11 Litery: F, K, Nr 1 Litery: I, T, Nr 13 Litery: L, Nr 23 Litery: S,	Nr 0 Litery: A, F, H, M, N, Nr 28 Litery: B, C, L, S, Nr 29 Litery: D, G, J, O, Q, Nr 1 Litery: E, P, Nr 24 Litery: I, Nr 4 Litery: K, R, Nr 16 Litery: T,

Dzięki sieciom Kohonena oraz przy użyciu algorytmu Winner-Takes-Most jesteśmy w stanie rozpoznać charakterystyczne cechy danych wejściowych (w tym przypadku wektorowych obrazów liter) oraz starać się je pogrupować względem ich podobieństwa. Widzimy iż przy małej wartości współczynnika nauki, powstało najwięcej grup zawierających pojedynczą literę. Świadczy to o braku rozpoznania cech wspólnych w danych testowych, jak również o poprawnym rozpoznaniu konkretnych liter poprzez przyporządkowanie im zwycięskiego neuronu. Sam algorytm WTM bardzo dobrze sprawdza się na zestawie danych wielowymiarowych, które następnie jesteśmy w stanie reprezentować w postaci wymiarów mniejszych.

Listing kodu:

Layers:

```
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> computeLayer(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, double alfa, double theta) {
        neurons[id].modifyWeights(data, alfa, theta);
    }

    public int getNeuronCount() {
        return neuronCount;
    }
}
```

NeuralNetwork:

```
public class NeuralNetwork {
    final static double learning_rate = 0.1; // wspolczynnik uczenia
    final static double promien_sigma = 30; // poczatkowy promien sasiedztwa
    private final static int epochs = 10000;

    private int neuron_number;
    private final static int input_number = 20;
    private final static int testinput_number = 20;
```

```
    final static int record_vectorsize = 35;

    private Data[] learningtab_data;
    private Data[] testingtab_data;

    private final char letter_tab[] = { 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K',
'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T'};

    private Layer layer;

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

            layer = new Layer(neuron_number);
            learningtab_data = new Data[input_number];
            testingtab_data = new Data[testinput_number];
            loadData(scanner, false);
            loadData(scanner2, true);
            scanner.close();
            showData();

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

    private void loadData(Scanner s, boolean isTest) {
        int inputsCount = isTest ? testinput_number : input_number;
        Data[] cyrk = new Data[inputsCount];
        for (int i = 0; i < inputsCount; i++) {
            cyrk[i] = new Data(record_vectorsize);
            cyrk[i].setY(s.next());
            for (int j = 0; j < record_vectorsize; j++) {
                cyrk[i].setXi(j, Double.parseDouble(s.next()));
            }
        }
        if (isTest) {
            testingtab_data = cyrk;
        } else {
            learningtab_data = cyrk;
        }
    }


    private void showData() {
        System.out.println("Wczytano 20 liter");
        System.out.println();
        for (int i = 0; i < 20; i++) {
            System.out.println(letter_tab[i]);
            for (int j = 1; j <= 35; j++) {
                if ((int) learningtab_data[i].getXi(j-1) > 0) {
                    System.out.print(1);
                } else {
                    System.out.print(0);
                }
                if (j % 5 == 0) {
                    System.out.println();
                }
            }
            System.out.println();
        }
    }

    public void learn() {
        normalize(learningtab_data);

        int iterations = 0;
        int winner;
```

```
        double sigma, theta, alfa = 0;
        int sigmaRounded;

        ArrayList<Double> result;

        do {
            alfa = alfaFunction(iterations);
            sigma = sigmaFunction(iterations);
            sigmaRounded = (int) Math.round(sigma);

            for (int i = 0; i < input_number; i++) {
                result = layer.computeLayer(learningtab_data[i]);
                winner = result.indexOf(Collections.min(result));

                for (int j = (-sigmaRounded); j <= sigmaRounded; ++j) {
                    if (((winner + j) >= 0) && ((winner + j) < neuron_number)) {
                        theta = thetaFunction(sigma, j);
                        layer.modify((winner + j), alfa, theta);
                    }
                }
            }
            ++iterations;
        } while (iterations < epochs);

        System.out.println("Uczenie zakończone. \nLiczba epok = " + iterations + "\n" +
                "Promień  = " + sigmaRounded + "\n" +
                "Współczynnik uczenia po zakończeniu = " +alfa + "\n\n");
    }

    public void test() {
        double globalError = 0.0;
        normalize(testingtab_data);
        ArrayList<Double> result;
        ArrayList<Double> errors = new ArrayList<Double>();
        ArrayList<LetterMap> groups = new ArrayList<LetterMap>();
        int winner;
        for (int i = 0; i < testinput_number; i++) {

            result = layer.computeLayer(testingtab_data[i]);

            winner = result.indexOf(Collections.min( result ));
            errors.add(Collections.min( result ));

            boolean flag = true;
            for(LetterMap el : groups){
                if(el.id == winner) {
                    el.letters += testingtab_data[i].getY() + ", ";
                    flag = false;
                }
            }

            if(flag){
                groups.add(new LetterMap(winner, (testingtab_data[i].getY() + ", " )));
            }

            System.out.println(" Litera: " + testingtab_data[i].getY() + " Numer neuronu " +
winner);

        }

        System.out.print("\n\nWyszczególnione grupy\n");

        for(LetterMap el: groups)
            System.out.println("Nr " + el.id + " Litery: " + el.letters);

        for(Double el: errors)
            globalError += el;

        globalError = (1.0 / (double) testinput_number) * globalError;
```

```
        System.out.println("\nGlobal Error = " +globalError);

    }

    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);
        }
    }

    private double alfaFunction(int counter) {
        return (learning_rate * Math.exp(-((double) counter) / ((double) epochs)));
    }

    private double sigmaFunction(int counter) {
        return (promien_sigma * Math.exp(-((double) counter) / ((double) epochs)));
    }

    private double thetaFunction(double sigma, int r) {
        return Math.exp(-(Math.pow(r, 2)) / (2 * Math.pow(sigma, 2)));
    }

    private class LetterMap{
        public int id;
        public String letters;
        public LetterMap(int id, String letters){this.id = id; this.letters = letters;}
    }
```

Neuron:

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

    Neuron()
    {
        weight = new double[NeuralNetwork.record_vectorsize];
        for(int i = 0; i < NeuralNetwork.record_vectorsize; 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){
        double result;

        result = 1.0 / (Math.pow(signal, 0.5));

        return result;
    }

    public void modifyWeights(Data data, double alfa, double theta) {
        for (int i = 0; i < NeuralNetwork.record_vectorsize; i++) {
            weight[i] = weight[i] + alfa * theta *  (data.getXi(i) - weight[i]);
```

```
        }
        normalizeWeights();
    }

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

        return signal;
    }

    private void normalizeWeights(){
        double lenghtSquared = 0.0;
        double lenght;

        for(int i = 0; i < weight.length; i++) {
            lenghtSquared += Math.pow(weight[i], 2);
        }

        lenght = Math.sqrt(lenghtSquared);

        for(int i=0; i<weight.length; i++) {
            weight[i] /= lenght;
        }
    }

    public double getWeightI(int i) {
        return weight[i];
    }
}
```

Main:

```
public class Main {

    public static void main(String[] args) {
        System.out.println("Scenariusz 6 - Budowa i dzialanie sieci Kohonena dla WTM");

        NeuralNetwork network = new NeuralNetwork(30);

        network.learn();
        network.test();
    }
}
```

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 double[] getX() {
        return x;
    }

    public void setX(double[] x) {
        this.x = x;
    }

    public String getY() {
        return y;
    }

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

    public int getxCount() {
        return xCount;
    }

    public void setxCount(int xCount) {
        this.xCount = xCount;
    }
}
```