Scenariusz 4

Maciej Słaboń
Gr 4

Celem ćwiczenia :
Napisanie programu rozpoznającego emotikony używając reguły Hebba w kilku wersjach.

Syntetyczny opis algorytmu uczenia:

Do wykonania ćwiczenia wykorzystałem jednowarstwową sieć składającą się z czterech neuronów. Pojedynczy neuron dobrze sprawował się w zadaniach, w których należało przyporządkować dany obiekt zero - jedynkowo.

Użyłem reguły Hebba w dwóch wersjach: bez i z nauczycielem, oraz implementują współczynnik zapominania. W obydwu funkcją aktywacji była unipolarna funkcja sigmoidalna. Decyzja, którą emotikonę przedstawiają dane na wejściu, była podejmowana na podstawie, który neuron uzyskał największy wynik.

Modyfikacja wag dla wersji bez nauczyciela:

$$w_i(t+1) = (1-\gamma)w_i(t) + \eta y x_i$$

- i - numer wagi neuronu,
- t - numer iteracji w epoce,
- y - sygnał wyjściowy neuronu,
- x - wartość wejściowa neuronu,
- η - współczynnik uczenia (0,1>,
- γ - współczynnik zapominania <0,1):

Należy zauważyć, że jeżeli γ jest równa 0 otrzymujemy wersję uczenia bez zapominania.

Dla wersji z nauczycielem wystarczy zamienić y - sygnał wyjściowy z neuronu na d - sygnał wzorcowy.

Podczas testów można było zauważyć pewną wadę. Wagi potrafiły wzrosnąć do bardzo wysokich wag (co znacznie wpłynęło na jakość uczenia, proces uczenia był rozbieżny). Aby temu zapobiec zastosowałem regułę Oji.

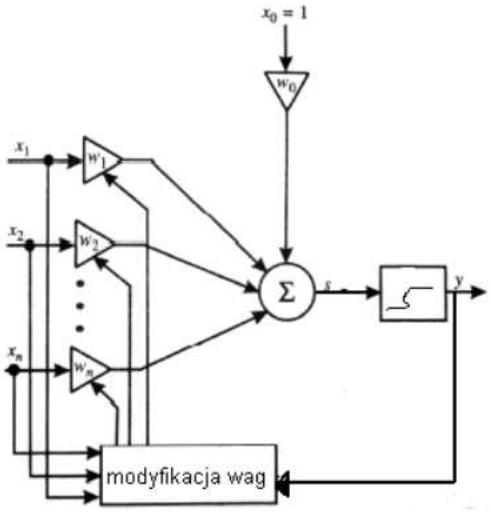
$$w_j(t+1) = \frac{w_j(t) + \alpha y(t) x_j(t)}{\{\sum_k (w_k(t) + \alpha y(t) x_k(t))^2\}^{\frac{1}{2}}}$$

gdzie α - współczynnik uczenia

Funkcja aktywacji

$$f(x) = \frac{1}{1 + e^{-\beta x}}$$

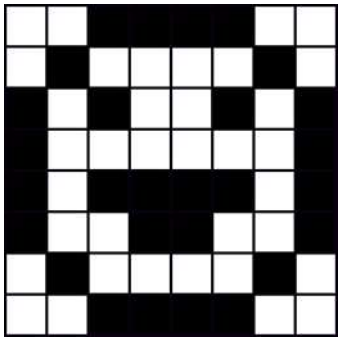
Rysunek przedstawia model neuronu Hebba

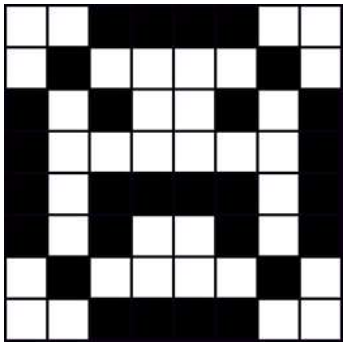


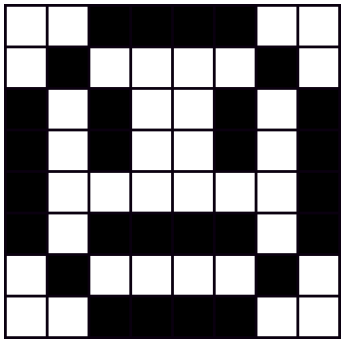
Dane:

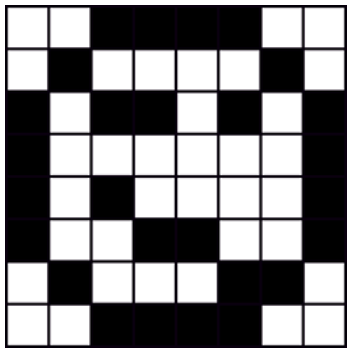
Dane potrzebne do ćwiczenia wykonałem sam.

Jako dane uczące przygotowałem 4 emotikony:

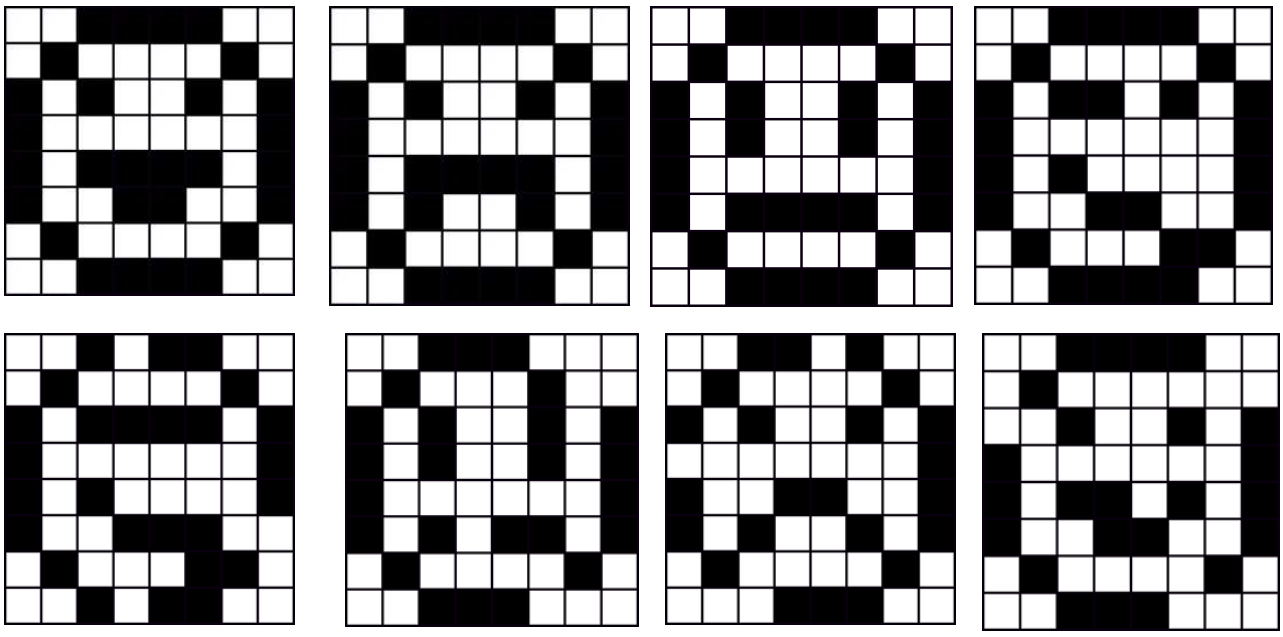
 :D

 :(

 :|

 ;/

Jako dane testujące przygotowałem 8 wariacji ze wzrastającym stopniem zniekształcenia.



Nauczanie przeprowadziłem dla kilku wersji:
Z nauczycielem, Bez nauczyciela
Dla współczynnika uczenia: 0.5; 0.1; 0.01;
Dla współczynnika zapominania: 0.001; 0.01; 0.1; 0.3

Wyniki:

Wykres ilość epok dla poszczególnych wariantów i ilość błędów

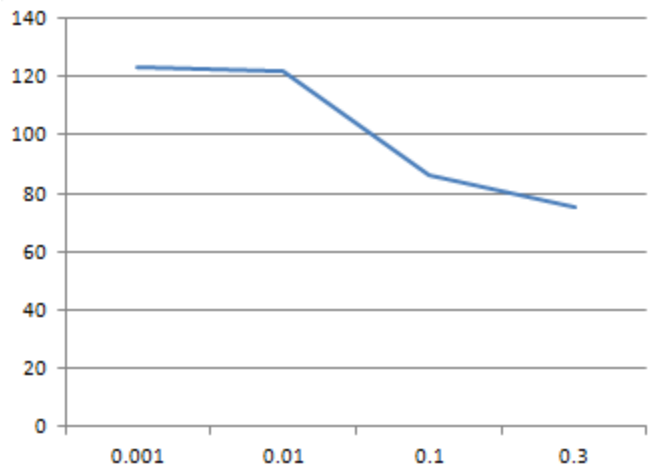
Wersja	Współczynnik Uczenia	Współczynnik Zapominania	Ilość epok	Błąd
Z nauczycielem	0.5	0.001	123	0
Z nauczycielem	0.5	0.01	122	1
Z nauczycielem	0.5	0.1	86	0
Z nauczycielem	0.5	0.3	75	0
Z nauczycielem	0.1	0.001	23	0
Z nauczycielem	0.1	0.01	13	0
Z nauczycielem	0.1	0.1	5	0
Z nauczycielem	0.1	0.3	7	0
Z nauczycielem	0.01	0.001	140	1
Z nauczycielem	0.01	0.01	63	0
Z nauczycielem	0.01	0.1	11	0
Z nauczycielem	0.01	0.3	30	0
Bez nauczyciela	0.5	0.001	132	-
Bez nauczyciela	0.5	0.01	80	-
Bez nauczyciela	0.5	0.1	80	-
Bez nauczyciela	0.5	0.3	79	-
Bez nauczyciela	0.1	0.001	54	0
Bez nauczyciela	0.1	0.01	64	1
Bez nauczyciela	0.1	0.1	42	1

Bez nauczyciela	0.1	0.3	113	0
Bez nauczyciela	0.01	0.001	199	1
Bez nauczyciela	0.01	0.01	79	1
Bez nauczyciela	0.01	0.1	4	2
Bez nauczyciela	0.01	0.3	9	2

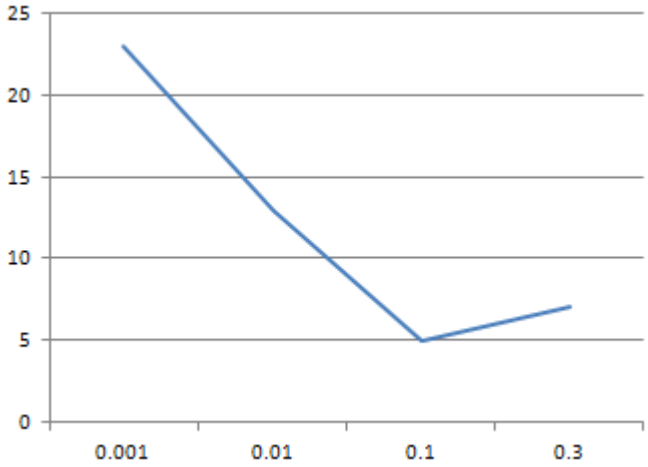
Wykresy:

A) Z nauczycielem

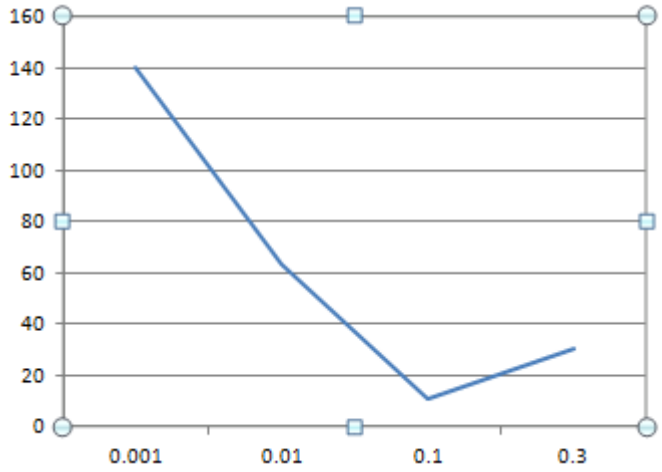
- Współczynnik uczenia 0.5
- ilość epok do współczynnika zapominania



- Współczynnik uczenia 0.1
- ilość epok do współczynnika zapominania



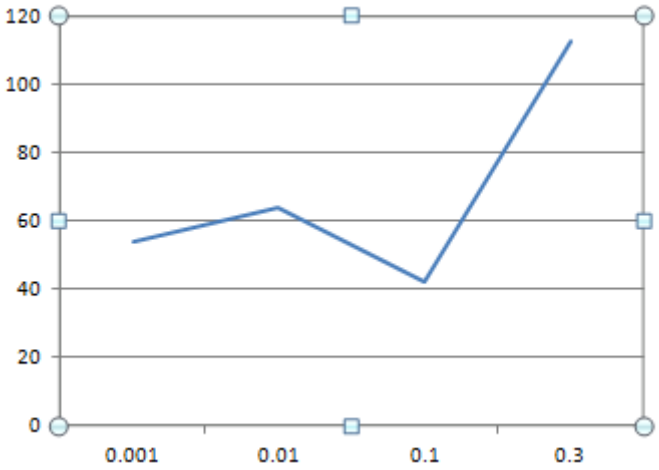
- Współczynnik uczenia 0.01
- ilość epok do współczynnika zapominania



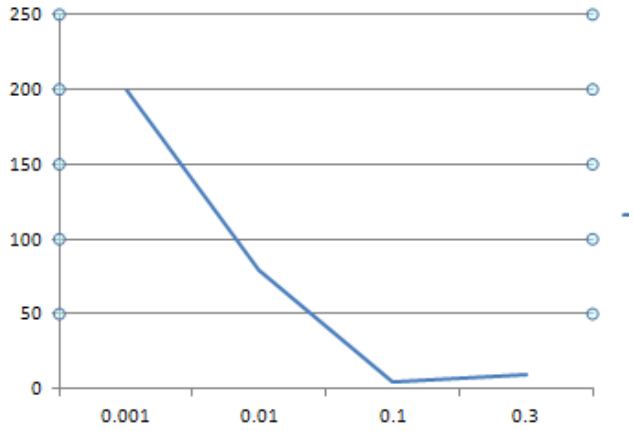
B) Bez nauczyciela

- Współczynnik uczenia 0.5
- ilość epok do współczynnika zapominania
  Przy tym współczynniku i powyżej nauczanie nie powiodło się

- Współczynnik uczenia 0.1
- ilość epok do współczynnika zapominania



- Współczynnik uczenia 0.01
- ilość epok do współczynnika zapominania



Analiza wyników:

Wyniki w wersjach nauczania z i bez nauczyciela są podobnie natomiast należy zwrócic uwagę, że z nauczycielem nauka wypadała dużo lepiej. Dodatkowo należy wspomnieć o fakcie, że dla wersji bez nauczyciela sieć nie potrafiła się nauczyć emotikon przy współczynniku 0.5
Największy wpływ na wyniki miały wylosowane wagi początkowe na co wskazuje dość losowe uczenie, które potrafiło zakończyć się sukcesem już po pierwszej epoce.
Bardzo ważny jest współczynnik zapominania . Dla wersji z nauczycielem i współczynnika uczenia 0.01 Najlepiej wypadł współczynnik zapominania równy 0.1 z ok 11 epokami. Dla porównania warto zwrócić uwagę, że dla współczynnika równego 0.001 potrzeba było 140 epok.

Na proces uczenia wpływała wartość współczynnika uczenia. Bardzo dobrze działa współczynnik o wartości 0.1.

Im więcej danych do nauki tym mniejsze prawdopodobieństwo losowości wyników oraz rozpoznawania zaszumionych emotikon.

Wnioski:

Analizując wyniki można dojść do następujących wniosków:

- Wersja z nauczycielem jest lepsza, zawsze była w stanie nauczyć się zadania, robiła to minimalnie szybciej, miała mniejsze błędy.
- W przypadku wersji bez nauczyciela nie wiemy, który neuron nauczył się rozpoznawać którą emotikonę. Trzeba do tego przeprowadzić dodatkowe testy.
- Wersja bez nauczyciela często nie potrafiła nauczyć się rozpoznawać emotikon, szczególności przy współczynniku uczenia powyżej 0.5, możliwe rozwiązania to zwiększenie ilości danych uczących oraz zwiększenie limitu epok.
- Aby uzyskać optymalną ilość epok uczenia dla sieci należało zastosować parametry: współczynnik uczenia: 0.1, współczynnik zapominania 0.1 (wskazane nauczanie z

nauczycielem). Ponieważ przy współczynniku zapominania mniejszym od 0.1 proces nauki bardzo się przeciągał.

Kod

```
public class Main {

    public static void main(String[] args) {
        double learningRate = 0.1;
        double decay = 0.9; //współczynnik zapominania
        int numberOfNeurons = 4;
        Trainer trainer = new Trainer(numberOfNeurons, learningRate, decay, shouldLearningBeSupervised: false);
        if (trainer.train()) {
            System.out.println("Testowanie nauki na danych do nauki");
            trainer.test(DataProvider.correct);
            System.out.println("Testowanie nauki na danych zaszumionych");
            trainer.test(DataProvider.corrupted);
        }
    }
}
```

```
public class DataProvider {
    public static int nFields = 64;
    public static int nFieldsX = 8;
    public static int nFieldsY = 8;

    public static String[] emoticons = {":D", ":(", ":|", ";/"};
    public static double[][][] correct = new double[][][]{
            {
                    {0, 0, 1, 1, 1, 1, 0, 0},
                    {0, 1, 0, 0, 0, 0, 1, 0},
                    {1, 0, 1, 0, 0, 1, 0, 1},
                    {1, 0, 0, 0, 0, 0, 0, 1},
                    {1, 0, 1, 1, 1, 1, 0, 1},
                    {1, 0, 0, 1, 1, 0, 0, 1},
                    {0, 1, 0, 0, 0, 0, 1, 0},
                    {0, 0, 1, 1, 1, 1, 0, 0}
            },

            {
                    {0, 0, 1, 1, 1, 1, 0, 0},
                    {0, 1, 0, 0, 0, 0, 1, 0},
                    {1, 0, 1, 0, 0, 1, 0, 1},
                    {1, 0, 0, 0, 0, 0, 0, 1},
                    {1, 0, 1, 1, 1, 1, 0, 1},
                    {1, 0, 1, 0, 0, 1, 0, 1},
                    {0, 1, 0, 0, 0, 0, 1, 0},
                    {0, 0, 1, 1, 1, 1, 0, 0}
            },
            {
                    {0, 0, 1, 1, 1, 1, 0, 0},
                    {0, 1, 0, 0, 0, 0, 1, 0},
                    {1, 0, 1, 0, 0, 1, 0, 1},
                    {1, 0, 1, 0, 0, 1, 0, 1},
                    {1, 0, 0, 0, 0, 0, 0, 1},
                    {1, 0, 1, 1, 1, 1, 0, 1},
                    {0, 1, 0, 0, 0, 0, 1, 0},
                    {0, 0, 1, 1, 1, 1, 0, 0}
            },
            {
                    {0, 0, 1, 1, 1, 1, 0, 0},
                    {0, 1, 0, 0, 0, 0, 1, 0},
                    {1, 0, 1, 1, 0, 1, 0, 1},
                    {1, 0, 0, 0, 0, 0, 0, 1},
                    {1, 0, 1, 0, 0, 0, 0, 1},
                    {1, 0, 0, 1, 1, 0, 0, 1},
                    {0, 1, 0, 0, 0, 1, 1, 0},
                    {0, 0, 1, 1, 1, 1, 0, 0}
            }
    };
```

```
public static double[] expected = {0, 1, 2, 3};
public static double[][][] corrupted = new double[][][]{
        {
                {0, 0, 1, 1, 1, 1, 0, 0},
                {0, 1, 0, 0, 0, 0, 0, 0},
                {0, 0, 1, 0, 0, 1, 0, 1},
                {1, 0, 0, 0, 0, 0, 0, 1},
                {1, 0, 1, 1, 0, 1, 0, 1},
                {1, 0, 0, 1, 1, 0, 0, 1},
                {0, 1, 0, 0, 0, 0, 1, 0},
                {0, 0, 1, 1, 1, 0, 0, 0}
        },
        {
                {0, 0, 1, 1, 0, 1, 0, 0},
                {0, 1, 0, 0, 0, 0, 1, 0},
                {1, 0, 1, 0, 0, 1, 0, 1},
                {0, 0, 0, 0, 0, 0, 0, 1},
                {1, 0, 0, 1, 1, 0, 0, 1},
                {1, 0, 1, 0, 0, 1, 0, 1},
                {0, 1, 0, 0, 0, 0, 1, 0},
                {0, 0, 0, 1, 1, 1, 0, 0}
        },
        {
                {0, 0, 1, 1, 1, 0, 0, 0},
                {0, 1, 0, 0, 0, 1, 0, 0},
                {1, 0, 1, 0, 0, 1, 0, 1},
                {1, 0, 1, 0, 0, 1, 0, 1},
                {1, 0, 0, 0, 0, 0, 0, 1},
                {1, 0, 1, 0, 1, 1, 0, 1},
                {0, 1, 0, 0, 0, 0, 1, 0},
                {0, 0, 1, 1, 1, 0, 0, 0}
        },
        {
                {0, 0, 1, 0, 1, 1, 0, 0},
                {0, 1, 0, 0, 0, 0, 1, 0},
                {1, 0, 1, 1, 1, 1, 0, 1},
                {1, 0, 0, 0, 0, 0, 0, 1},
                {1, 0, 1, 0, 0, 0, 0, 1},
                {1, 0, 0, 1, 1, 1, 0, 0},
                {0, 1, 0, 0, 0, 1, 1, 0},
                {0, 0, 1, 0, 1, 1, 0, 0}
        }
};

public static double[] getData(int number, double[][][] data) {
    double[] result = new double[nFields];
    for (int i = 0; i < nFieldsY; i++) {
        for (int j = 0; j < nFieldsX; j++) {
            result[i * nFieldsX + j] = data[number][i][j];
        }
    }
```

```java
public class Neuron {
    private double[] weights;
    private double bias = 0.5;

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

    public double getResult(double[] input) {   //wyniki
        double sum = sum(input);
        return actFun(sum);
    }

    public void learn(double[] input, double output, double lr, double decay) { //algorytm uczenia zgodnie ze sprawozdaniem
        for (int i = 0; i < input.length; i++) {
            weights[i] = ((1 - decay) * weights[i] + lr * output * input[i]);

        }
        bias = ((1 - decay) * bias + lr * output);
        normalize();
    }

    private double sum(double[] input) {    //sumowanie wag*dane wejściowe
        double sum = 0;
        for (int i = 0; i < input.length; i++) {
            sum += weights[i] * input[i];
        }

        return sum + bias;
    }

    private double actFun(double sum) { return 1 / (1 + Math.exp(-sum)); }   //funkcja aktywacji

    private void normalize() {  //normalizowanie wag (tak aby nie były zbyt duże)
        double dl = 0.0;
        for (int i = 0; i < weights.length; i++)
            dl += Math.pow(weights[i], 2);

        dl = Math.sqrt(dl);

        for (int i = 0; i < weights.length; i++)
            if (weights[i] > 0 && dl != 0)
                weights[i] = weights[i] / dl;
        if (dl != 0)
            bias = bias / dl;
    }
}
```

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

public class Trainer {
    private Neuron[] layer;
    private double learningRate;
    private int Max = 1000;
    private double decay;
    private boolean teacherFlag;
    int counter;

    public Trainer(int numberOfNeurons, double learningRate, double decay, boolean shouldLearningBeSupervised) {
        this.learningRate = learningRate;
        this.decay = decay;
        this.teacherFlag = shouldLearningBeSupervised;
        layer = new Neuron[numberOfNeurons];

        Random r = new Random();
        double[] weights = new double[DataProvider.nFields];

        for (int i = 0; i < numberOfNeurons; i++) {
            for (int j = 0; j < weights.length; j++) {
                weights[j] = r.nextDouble();
            }
            layer[i] = new Neuron(weights);
        }
    }

    public boolean train() {    //uczenie zależne od tego czy z nauczycielem czy bez
        this.counter = 0;

        if (this.teacherFlag) {
            supervised();
        } else {
            unsupervisedLearning();
        }
        if (counter == Max) {
            System.out.println("Nie udało się nauczyć");
            return false;
        }
        System.out.println("Epoki: " + this.counter);
        return true;
    }
```

```java
    public int act(double[] input) {    //sprawdzanie który neuron zareagował na którą emotikonę
        int winner = 0;
        double result;
        double winnerResult = layer[0].getResult(input);
        for (int i = 1; i < layer.length; i++) {
            result = layer[i].getResult(input);
            if (winnerResult < result) {
                winner = i;
                winnerResult = result;
            }
        }
        return winner;
    }

    private void supervised() { //uczenie z nauczycielem zgodnie ze sprawozdaniem
        this.counter = 0;
        double[] input;
        double expected;
        double[] winners = new double[layer.length];
        do {
            for (int i = 0; i < DataProvider.expected.length; i++) { //dla wszystkich emotikon
                input = DataProvider.getData(i, DataProvider.correct);  //pobiera dane wejściowe z providera
                for (int j = 0; j < layer.length; j++) {    //dla każdego neuronu
                    if (i == j) {   //jeżeli id neuronu == id emotikony neuron powinien na nią zareagować, w przeciwnym razie powinien nie zostać wzbudzony
                        expected = 1;
                    } else {
                        expected = 0;
                    }
                    layer[j].learn(input, expected, this.learningRate, this.decay); //uczenie neuronu
                }
                for (int j = 0; j < layer.length; j++) {
                    input = DataProvider.getData(j, DataProvider.correct);
                    winners[j] = act(input);    //sprawdzanie czy na daną emotikonę zareagował tylko jeden neuron
                }
            }
            this.counter++;
        } while (this.counter < Max && Arrays.stream(winners).distinct().count() != winners.length);    //drugi warunek sprawdza czy każdy neuron reaguje na różną emotikonę
    }
```

```
private void unsupervisedLearning() {   //uczenie bez nauczyciela zgodnie ze sprawozdaniem
    this.counter = 0;
    double[] input;
    double result;
    double[] winners = new double[layer.length];
    do {
        for (int i = 0; i < DataProvider.expected.length; i++) {
            input = DataProvider.getData(i, DataProvider.correct);
            for (int j = 0; j < layer.length; j++) {
                result = layer[j].getResult(input);
                layer[j].learn(input, result, this.learningRate, this.decay);
            }
            for (int j = 0; j < layer.length; j++) {
                input = DataProvider.getData(j, DataProvider.correct);
                winners[j] = act(input);
            }
        }
        this.counter++;
    } while (this.counter < Max && Arrays.stream(winners).distinct().count() != winners.length);
    for (int j = 0; j < layer.length; j++) {
        input = DataProvider.getData(j, DataProvider.correct);
        winners[j] = act(input);
    }
}

public void test(double[][][] dataSet) {    //testowanie neuronów czy zwracają wartości oczekiwane na dane emotikony (czy odpowiednie neurony zostały aktywowane)
    double[] input;
    int[] winners = new int[layer.length];
    int errors = 0;
    String expected, got;
    for (int j = 0; j < layer.length; j++) {
        input = DataProvider.getData(j, DataProvider.correct);
        winners[j] = act(input);
    }
    for (int i = 0; i < dataSet.length; i++) {
        input = DataProvider.getData(i, dataSet);
        expected = DataProvider.emoticons[winners[i % DataProvider.emoticons.length]];
        got = DataProvider.emoticons[act(input)];
        System.out.println("Emotikona nr " + (i + 1) + ". Exp: " + expected + " Got: " + got);
        if (!DataProvider.emoticons[act(input)].equals(DataProvider.emoticons[winners[i % DataProvider.emoticons.length]])) {
            errors++;
        }
    }
    System.out.println("Errors: " + errors);
}
```