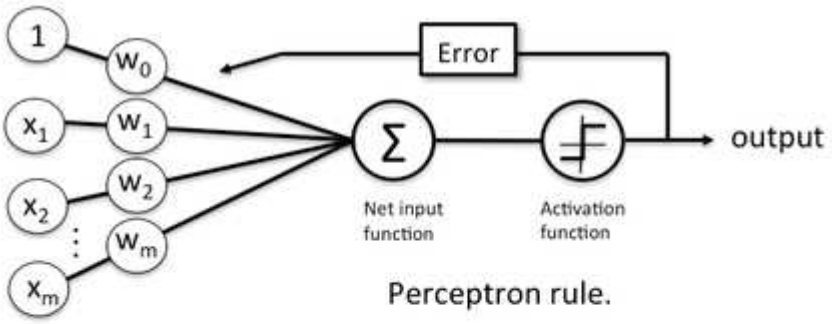
Scenariusz 2

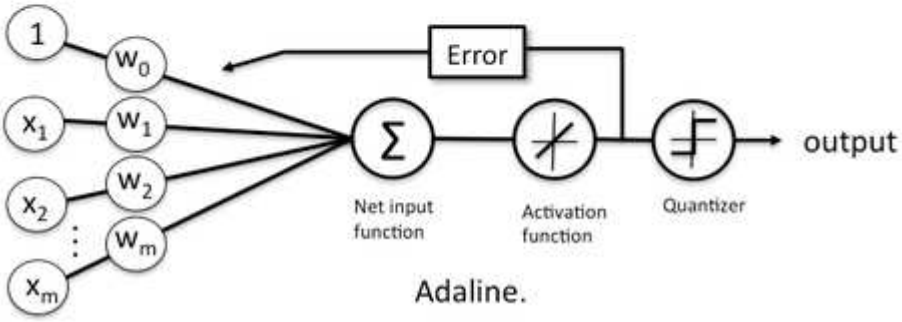
Maciej Słaboń
Gr 4

Cel ćwiczenia:
Nauczenie jednowarstwowej sieci neuronowej do rozpoznawania dużych i małych liter

Do ćwiczenia wykorzystałem perceptron oraz Adaline.

Syntetyczny opis algorytmu uczenia:





Do wykonania ćwiczenia wykorzystałem sieć neuronową składającą się z dwóch neuronów sigmoidalnych. Taki typ neuronu, uznałem za stosowny ponieważ posiadam ciągłe wartości w przedziale [0,1], ze względu na metodykę decydowania o wielkości litery.

Unipolarną funkcję sigmoidalną możemy zapisać wzorem:

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

Każdy z nich badał jakiej wielkości jest litera. Pierwszy szukał liter dużych, drugi liter małych. Decyzja czy litera jest duża czy mała była podejmowana na podstawie który neuron zwrócił większą wartość. Im bliżej 1 tym większe prawdopodobieństwo, że litera jest danej wielkości.

Nowe wagi obliczane są przy pomocy funkcji:

$$w = w + \eta * (o - y) * f'(z) * x$$

Gdzie
η to learning rate
o-y to różnica wyjścia i oczekiwanej
f'(z) to pochodna funkcji aktywacji równa y*(1-y) a
x to wejście
W drugim przypadku podobnie wykorzystałem sieć składającą się z dwóch neuronów Adaline. Jako funkcję aktywacji przyjąłem signum.
Był on uczony zgodnie z regułą:

$$w(k+1) = w(k) + \eta\, x(k)[d(k) - w^T(k)x(k)]$$

gdzie człon [d(k) - w(k)x(k)] odpowiada błędowi neuronu

Łączny błąd liczyłem z MSE, czyli

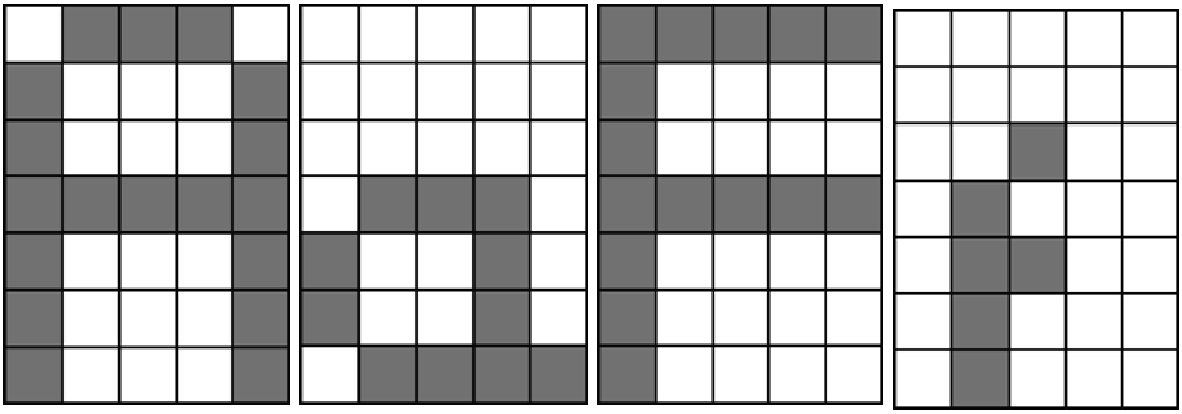
$$E_{total} = \sum \tfrac{1}{2}(target - output)^2$$

Perceptron:
Zbudowany na podstawie modelu podanym na wykładzie oraz wg. książki Stanisława Osowskiego "Sieci neuronowe do przetwarzania informacji".
Metody publiczne:
-GetResult: Przyjmuje dane wejściowe, zwraca wyliczoną wartość
-Learn: Zajmuje się nauką neuronu. Modyfikuje wagi.

Adaline:
Jest podobny do perceptron, tylko błąd jest obliczany przed właściwą funkcją aktywacji (wcześniej jest tylko funkcja liniowa). Błąd jest obliczamy na rzeczywistych wynikach. Został zbudowany na podstawie powyższej książki.
Metody publiczne:
-GetResult: Przyjmuje dane wejściowe, zwraca faktyczny wynik (przed funkcją aktywacji)
-Test: Zwraca dane, które przeszły przez funkcję aktywacji.
-Learn: Zajmuje się nauką neuronu. Modyfikuje wagi.

W obydwu przypadkach zastosowano progową funkcję aktywacji (albo litera jest mała, albo duża).

Zestaw danych testowych:

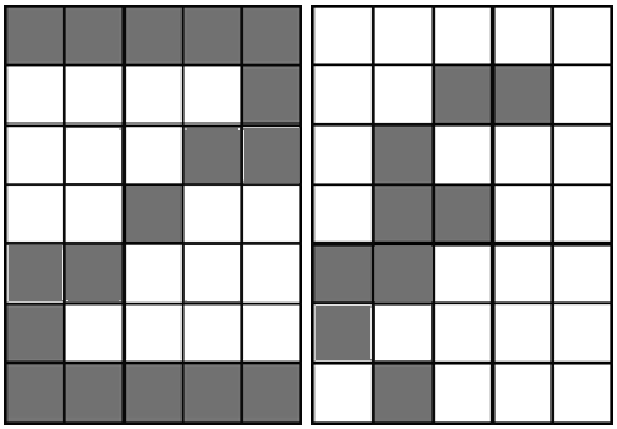


(przykładowe litery)

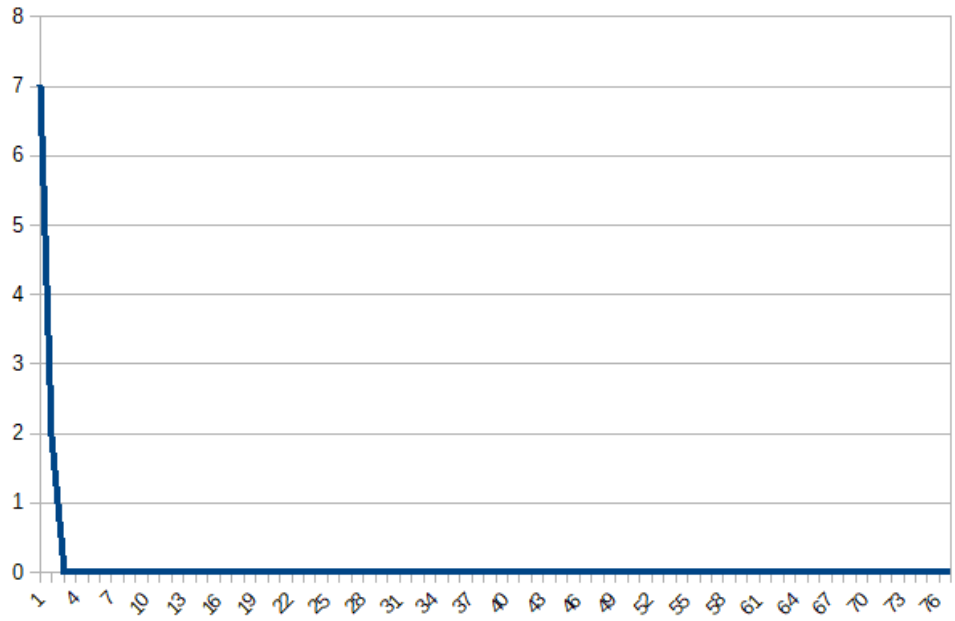
Testy:

Testy przeprowadziłem dla neuronów.
Testy zostały przeprowadzane ze względu na 2 kryteria:
-współczynnik uczenia się
-ilość obszarów

Na koniec (po zakończeniu uczenia) neurony były poddawane jeszcze testowi, w którym litery były zaszumiane (niektóre piksele miały złe wartości). Ten zestaw danych nie był wykorzystywany podczas uczenia.

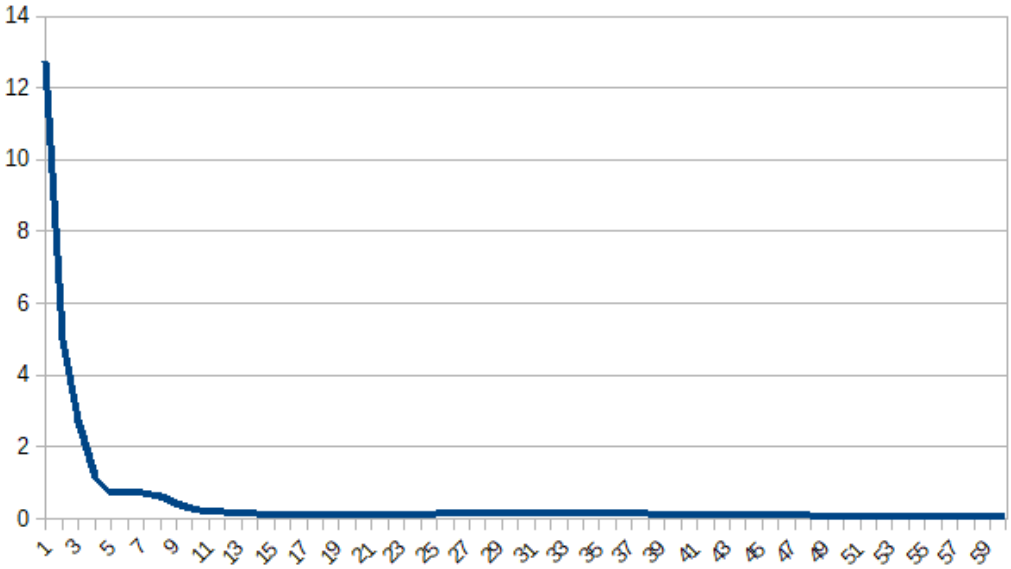


Użyłem 2 perceptronów. Jeden uczył się rozpoznawać duże litery, drugi natomiast próbował rozpoznawać małe. Postanowiłem również nie dzielić obszaru na fragmenty ale wykorzystać wszystkie 35 wejść.
Współczynnik uczenia 0.1;
Perceptron:



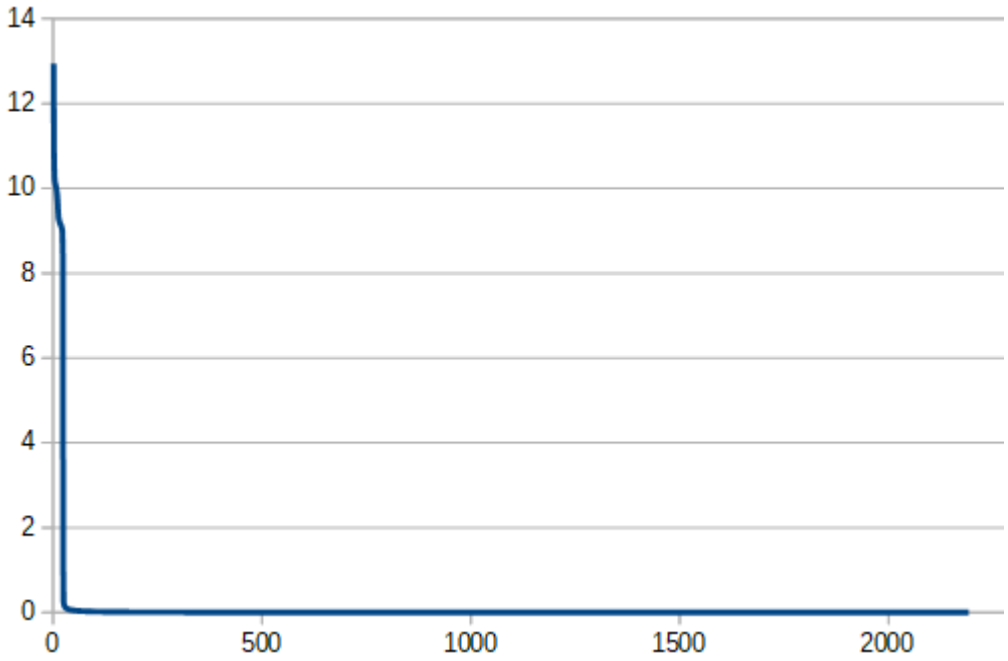
Wynik Testu - 100%

Adaline:



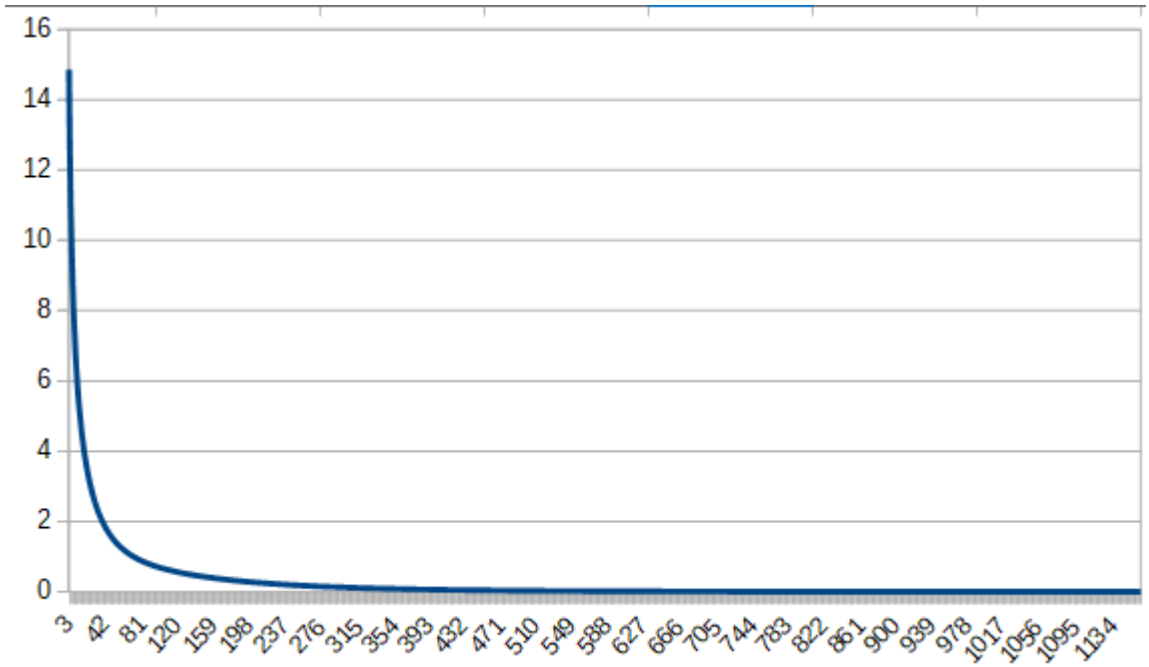
Wynik Testu - 95%

Współczynnik uczenia: 0.01
Perceptron:



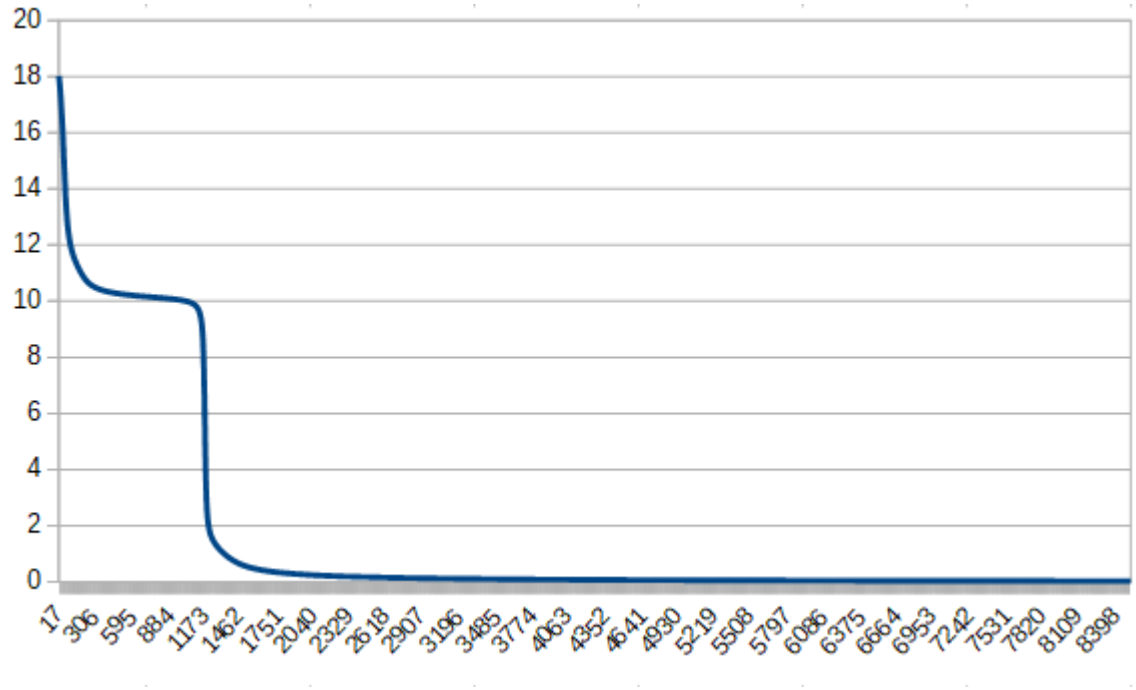
Wynik Testu - 100%

Adaline:



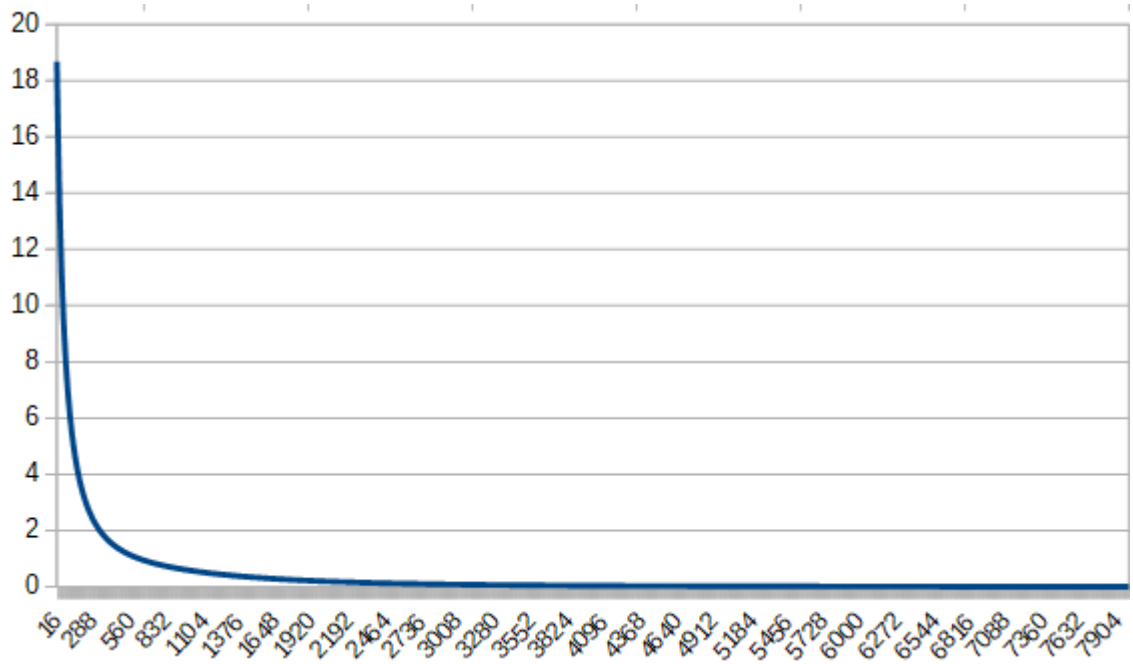
Wynik Testu - 100%

Współczynnik uczenia: 0.001
Perceptron:



Wynik testu - 95%

Adaline



Wynik testu - 95%

Analiza wyników:

Można również zwrócić uwagę na fakt, że sieć złożona z neuronów Adaline uczyła się znacznie szybciej od drugiej wersji.
Sieć zbudowana z Adaline uczyła się szybciej od perceptronów

Dwa neurony zawsze były w stanie nauczyć się rozpoznawać litery oraz mocno zaszumione litery. Sieć miała problemy z mała litera f, często rozpoznawał ją jako dużą literę

Wnioski:

- Złożone problemy wymagają większej ilości neuronów w warstwie.
- Adaline szybciej się uczył od Perceptronu
- Błędne przewidywanie zaszumionej małej litery ‘f’ mogło być spowodowane podobieństwem jej do dużej litery.
- Błąd z zaszumioną literą można by zniwelować dodając zaszumione litery do nauki
- Ważne jest aby danych było dużo i uwzględniały różne przypadki.
- Najlepszym współczynnikiem uczenia był zdecydowanie 0.1. Mniejsze nie zwiększały dokładności, zaś wydłużały okres nauki.

```
public class Main {

    public static void main(String[] args) {
        double learningRate = 0.1;

        System.out.println("Perceptron");
        PerceptronTrainer trainer = new PerceptronTrainer( numberOfPerceptrons: 2, learningRate);     //tworzy trenera z 2 perceptronami
        trainer.train();    //trenuje
        System.out.println("Testowanie poprawnych");
        trainer.test(Letters.correct);      //testowanie na danych wejściowych
        System.out.println("Testowanie zaszumionych");
        trainer.test(Letters.corrupted);        //testowanie danych zaszumionych

        System.out.println("Adaline");
        AdalineTrainer aTrainer = new AdalineTrainer(learningRate); //tworzy trenera adaline z 2 neuronami adaline
        aTrainer.train();   //trenuje
        System.out.println("Testowanie poprawnych");
        aTrainer.test(Letters.correct); //testowanie na danych wejściowych
        System.out.println("Testowanie zaszumionych");
        aTrainer.test(Letters.corrupted);       //na danych zaszumionych

    }
}
```

```
public class Letters {
        static int numberX = 5;         //ilość pól na szerokość
        static int numberY = 7;         //ilość pól na wysokość
        static int nFields = numberX * numberY; //ilość pól
        static int[] expected = new int[] { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }; //1 = duża, 0 - mała
        static char[] letter = new char[] { 'A', 'C', 'F', 'G', 'I', 'N', 'O', 'R', 'S', 'Z', 'a', 'c', 'f', 'g', 'i', 'n', 'o', 'r', 's', 'z' };

        static int[][][] correct = new int[][][]{
                {
                        {0, 1, 1, 1, 0},
                        {1, 0, 0, 0, 1},
                        {1, 0, 0, 0, 1},
                        {1, 1, 1, 1, 1},
                        {1, 0, 0, 0, 1},
                        {1, 0, 0, 0, 1},
                        {1, 0, 0, 0, 1}
                },                  //A
                {
                        {0, 1, 1, 1, 1},
                        {1, 0, 0, 0, 0},
                        {1, 0, 0, 0, 0},
                        {1, 0, 0, 0, 0},
                        {1, 0, 0, 0, 0},
                        {1, 0, 0, 0, 0},
                        {0, 1, 1, 1, 1}
                },                  //C
                {
```

```
                {0, 0, 1, 0, 0},
                {0, 1, 0, 0, 0},
                {0, 0, 1, 1, 1}
        }                   //z
        //Male
};
public static int[] getLetter(int number, int[][][] data) { //funkcja zwarca pojedyńcza litere z zestawu danych jako tablicę
    int[] result = new int[nFields];
    for (int i = 0; i < numberY; i++) {
        for (int j = 0; j < numberX; j++) {
            result[i * numberX + j] = data[number][i][j];
        }
    }
    return result;
}
```

```
public class Perceptron {
    private double[] weights;

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

    public double getResult(int[] input) { //zwraca wynik perceptronu funkcjaAktywacji(suma)
        double sum = sum(input);
        return actFun(sum);
    }

    public void learn(int[] input, double expected, double lr) { //nauka perceptronu zgodnie ze sprawozdaniem
        double result = getResult(input);   //wynik perceptronu
        double delta = expected - result;   //różnica między oczekiwana - wynik

        double error = delta * div(sum(input)); //obliczanie błędu (różnica*funkcja aktywacji)

        for (int i = 0; i < input.length; i++)
            weights[i] += error * lr * input[i];    //aktualizacja wag

        weights[Letters.nFields] += lr * error; //aktualizacja BIAS-u
    }

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

        double bias = 1;
        return sum + weights[input.length] * bias;  //dodanie biasu
    }

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

    private double div(double x) {
        return Math.exp(-x) / Math.pow((Math.exp(-x) + 1), 2);
    }   //pochodna funkcji aktywacji
}
```

```java
import java.util.Random;

public class PerceptronTrainer {
    private Perceptron[] layer; //ilość neuronów (każdy neuron powinien rozpoznawać jeden typ [duża,mała])
    private double learningRate;
    private final int maxEpoch = 1000000; //maksymalna liczba epok

    public PerceptronTrainer(int numberOfPerceptrons, double learningRate) {    //konstruktor(liczba neuronów, lr)
        this.learningRate = learningRate;
        layer = new Perceptron[numberOfPerceptrons];

        Random r = new Random();                                //losowanie wag i inicjalizacja neuronów
        double[] weights = new double[Letters.nFields + 1];

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

    public void train() {   //funckcja ucząca neurony
        int counter = 0;    //licznik epok
        int[] input;    //wejście
        int expected;   //wartość oczekiwana
        double result;  //wynik neuronu
        double mseError;    //błąd średniokwadratowy
        do {
            mseError = 0.0;
            for (int i = 0; i < Letters.expected.length; i++) { //dla każdej litery
                for (int j = 0; j < layer.length; j++) {   //dla każdego neuronu
                    input = Letters.getLetter(i, Letters.correct); //litera z providera jako tablica
                    expected = getExpected(Letters.expected[i], j); //sprawdza czego powinien oczekiwać dla którego neuronu (neuron 1 powinien oczekiwać odwrotnej wartości niż neuron 2)
                    layer[j].learn(input, expected, learningRate); //uczenie
                    result = layer[j].getResult(input); //weź wynik z uczenia
                    mseError += Math.pow((expected - result), 2.0); //oblicz błąd średniokwadratowy
                }
            }
            counter++;  //zwiększ licznik epok
        } while (counter < maxEpoch && mseError > 0.001); //warunek zakończenia uczenia
    }
```

```java
    public void test(int[][][] data) {   //testowanie data - litery poprawne lub zaszumione
        double correctAnswers = Letters.expected.length;  //ilość poprawnych wyników (na początku max)
        int[] input;
        int expected;
        for (int i = 0; i < Letters.expected.length; i++) { //dla wszystkich liter
            input = Letters.getLetter(i, data);
            expected = Letters.expected[i];

            if (act(input) != expected)     //sprawdza czy otrzymał wartość oczekiwaną
                correctAnswers--; //jeśli nie to zmniejsza ilość poprawnych wyników

            if (act(input) == 1) {
                System.out.print("d ");
            } else {
                System.out.print("m ");
            }
            System.out.println(Letters.letter[i]);
        }
        System.out.println("Procent: " + correctAnswers / 20 * 100 + "%");
    }

    private int getExpected(int expected, int nr) { //zwraca wartość oczekiwaną
        if (nr == 0) {
            return expected;
        } else if (expected > 0) {
            return 0;
        }

        return 1;
    }

    public int act(int[] input) {   //zwraca wynik, który neuron powinien być aktywowany
        if (layer[0].getResult(input) > layer[1].getResult(input)) {
            return 1;
        } else {
            return 0;
        }
    }
}
```

```java
public class Adaline {
    private double[] weights;
    private double bias = 1;

    public Adaline(double[] weights) {  //konstruktor kopiuje wagi neuronu
        this.weights = new double[weights.length];
        System.arraycopy(weights,  srcPos: 0, this.weights,  destPos: 0, weights.length);
    }

    public double getResult(int[] input) { return sum(input); }   //zwraca wynik

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

        return sum + weights[input.length] * bias;
    }

    private double actFun(double sum) {
        return 1 / (1 + Math.exp(sum));
    } //funkcja aktywacji zgodna ze wzorem w sprawozdaniu

    public void learn(int[] input, double expected, double lr) { //nauka zgodnie ze wzorem w sprawozdaniu
        double result = getResult(input);

        for (int i = 0; i < input.length; i++)
            weights[i] += (expected - result) * lr * input[i];

        weights[Letters.nFields] += lr * (expected - result);
    }
}
```

```java
import java.util.Random;

public class AdalineTrainer {
    private Adaline[] layer;
    private double learningRate;

    public AdalineTrainer(double learningRate) {
        this.learningRate = learningRate;
        int biasID = Letters.nFields;
        int numberOfNeurons = 2;
        layer = new Adaline[numberOfNeurons];
        double[] results = new double[numberOfNeurons];

        Random r = new Random();
        double[] weights = new double[Letters.nFields + 1];

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

    public void train() {
        int counter = 0;
        int[] input;
        int expected;
        double result;
        double mseError;
        int max = 100000000;
        do
        {
            mseError = 0.0;
            for (int i = 0; i < Letters.expected.length; i++)
            {
                for (int j = 0; j < 2; j++)
                {
                    input = Letters.getLetter(i, Letters.correct);
                    expected = getExpected(Letters.expected[i], j);
                    layer[j].learn(input, expected, learningRate);
                    result = layer[j].getResult(input);
                    mseError += Math.pow((expected - result), 2.0);
                }
            }
            counter++;
        } while (counter < max && mseError > 0.001);
        System.out.println(counter);
    }
```

```java
public void test(int[][][] letters) {
    double rate = Letters.expected.length;
    int[] input;
    int expected;
    for (int i = 0; i < Letters.expected.length; i++)
    {
        input = Letters.getLetter(i, letters);
        expected = Letters.expected[i];

        if (act(input) != expected)
            rate--;

        if (act(input) == 1)
        {
            System.out.print("d ");
        }
        else
        {
            System.out.print("m ");
        }

        System.out.println(Letters.letter[i]);
    }
    System.out.println("Poprawne odpowiedzi: " + rate / 20 * 100 + "%");
}

private int getExpected(int expected, int nr) {
    if (nr == 0)
    {
        if (expected == 0)
            return -1;
        return expected;
    }
    else if (expected > 0)
    {
        return -1;
    }

    return 1;
}

public int act(int[] input) {
    if (layer[0].getResult(input) > layer[1].getResult(input))
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
```