Daniel Wenecki

Podstawy Sztucznej Inteligencji

Sprawozdanie – Projekt nr 2

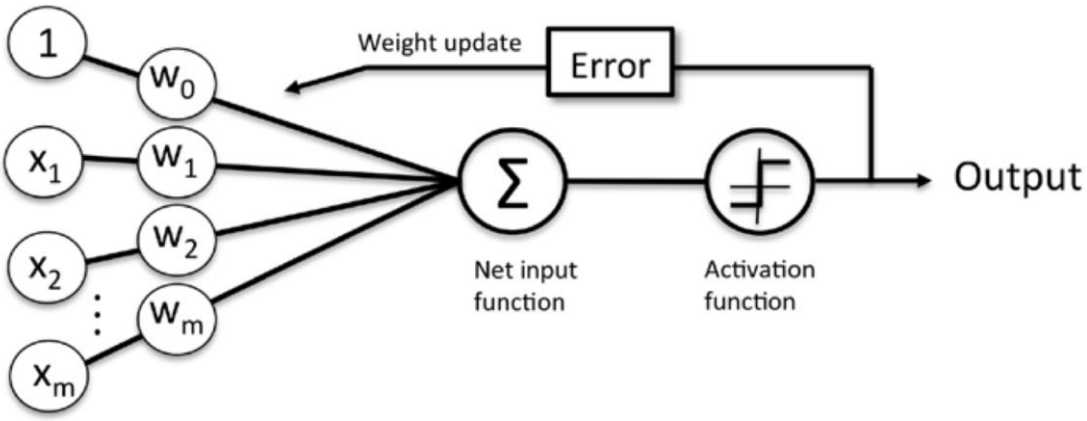
Celem ćwiczenia jest poznanie budowy i działania jednowarstwowych sieci neuronowych

oraz uczenie rozpoznawania wielkości liter.

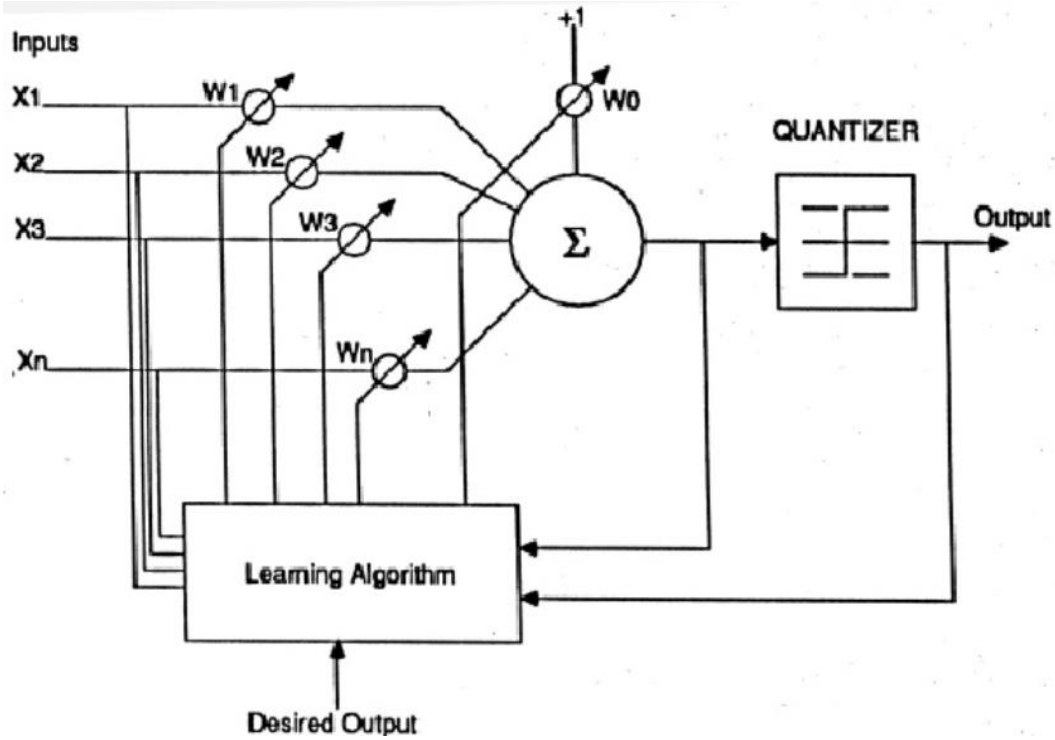
**1) Syntetyczny opis budowy oraz wykorzystania sieci i algorytmów uczenia**

Do wykonania ćwiczenia stworzyłem dwa modele jednowarstwowych sieci. Jedną złożoną z perceptronów, drugą wykorzystującą model MADALINE.

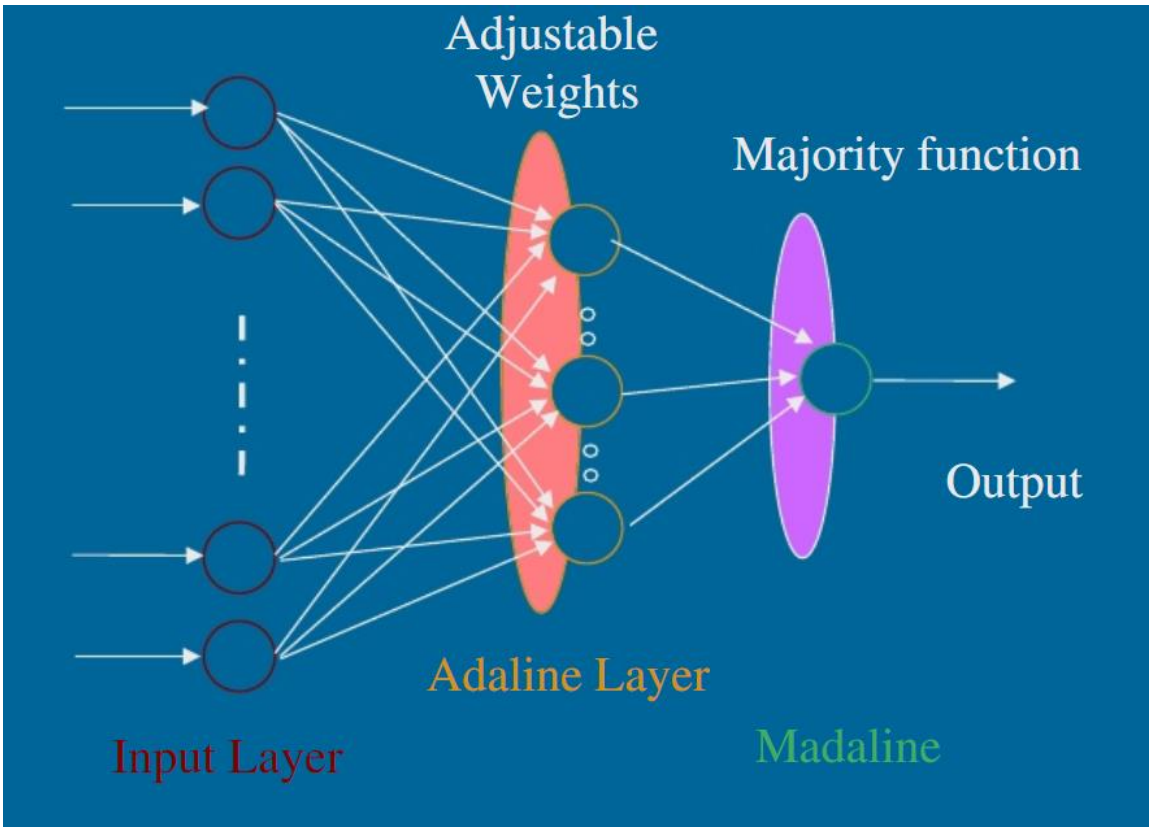
Budowa perceptronu:




Budowa ADALINE:



Struktura sieci MADALINE:




W drugim przypadku sieć wyglądała analogicznie, z tą różnicą, że do jej budowy użyłem perceptronów zamiast neuronów ADALINE i dostosowałem algorytm do obsługi .

Schemat nauki sieci MADALINE:

1. Inicjalizacja losowych wag i progu

2. Wprowadzenie wektora wejściowego  i oczekiwanej wartości

3. Obliczenie aktualnej wartości wyliczanej przez każdy neuron

$$y_k(t) = F_h[\sum_{i=0} w_{ki}(t) * x_{ki}(t)]$$

where $F_h(e) = 1$ when $e > 0$, and

$= -1$ when $e <= 0$

$y_k(t)$ is the output from adaline unit k.

4.Określenie wyjściowego sygnału z sieci M(t)

$$M(t) = \text{Majority}\ (y_k(t))$$

5.Określenie

Jeśli M(t) jest równe wartości oczekiwanej, nie ma potrzeby by zmieniać wagi.

W innym przypadku:

Wybieram jeden wzbudzony neuron, z suma wag najbliższą zero, ale z nieodpowiednim wyjściem. Tylko ten neuron jest modyfikowany.

$$W_{ci}(t+1) = W_{ci}(t) + \eta\left[d(t) - \sum_{i=1}^{n} W_{ci}(t)X_i(t)\right]X_i(t)$$

Gdzie:

0<i<n

eta to krok nauczania (zazwyczaj eta <=1/n)

c to wybrany wygrany neuron

6. Powtarzaj krok od 2 do 5 dopóki oczekiwane wyniki nie odbiegają od rzeczywistych

W przypadku sieci perceptronów algorytm postępowanie jest analogiczny do MADALINE, z tą różnicą, że do modyfikacji wag użyłem wartości wyjściowej zamiast sumy iloczynu wag i wejść.

Reguła uczenia perceptronu (Widrowa-Hoffa): $\Delta w = \eta * (y_{oczekiwane} - y) * x$

**2) Zestawienie otrzymanych wyników:**

Krok uczenia: 0.01

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| MADALINE | 96 | 100 | 79 | 102 | 77 | 78 | 89 | 69 | 108 | 113 |
| Percpetrony | 34 | 98 | 44 | 29 | 48 | 59 | 38 | 49 | 57 | 63 |

Krok uczenia:0.05

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| MADALINE | 102 | 151 | 105 | 163 | 132 | 81 | 169 | 90 | 176 | 79 |
| Percpetrony | 76 | 187 | 57 | 106 | 78 | 89 | 47 | 69 | 79 | 57 |

Krok uczenia:0.01

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| MADALINE | 349 | 342 | 183 | 734 | 916 | 272 | 282 | 439 | 135 | 169 |
| Percpetrony | 388 | 576 | 494 | 333 | 509 | 143 | 388 | 380 | 645 | 275 |

Krok uczenia:0.05

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| MADALINE | 764 | 942 | 1180 | 739 | 1335 | 701 | 880 | 891 | 372 | 930 |
| Percpetrony | 520 | 881 | 523 | 679 | 704 | 1148 | 1007 | 424 | 1184 | 845 |

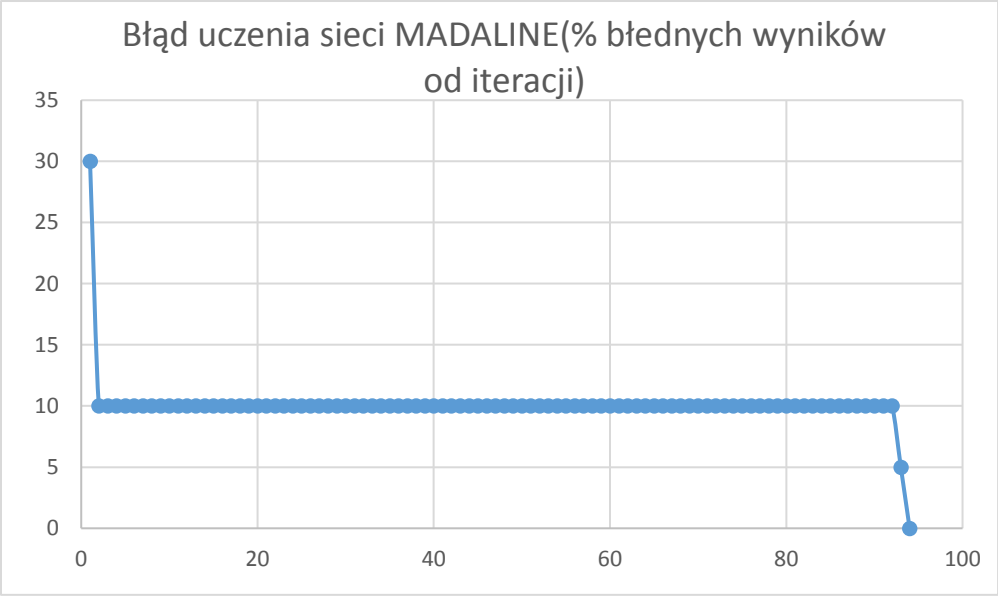
Krok uczenia:0.001

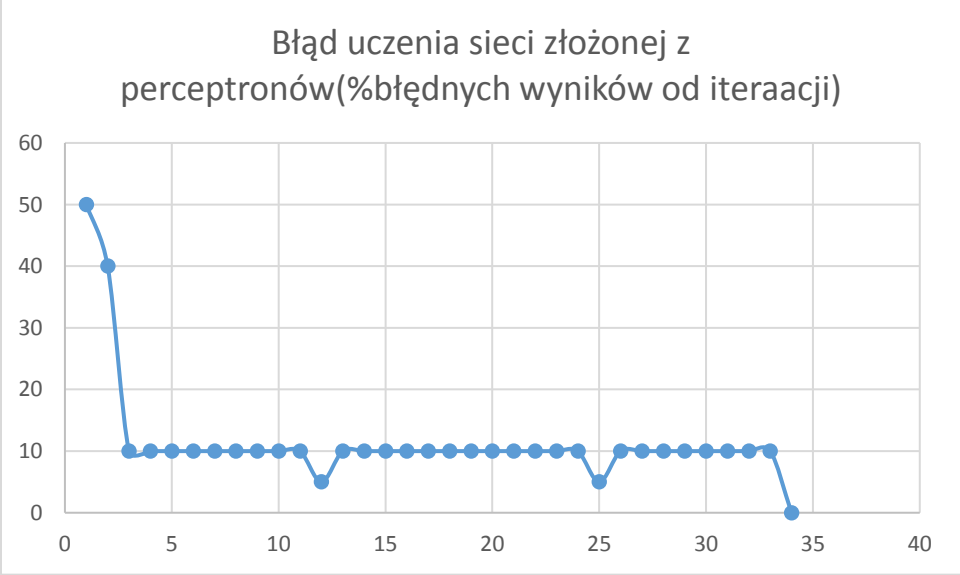
| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| MADALINE | 3837 | 5316 | 4950 | 3584 | 9226 | 3300 | 1860 | 3951 | 4251 | 9081 |
| Percpetrony | 893 | 2123 | 4732 | 5927 | 3782 | 2980 | 1042 | 617 | 3581 | 850 |

Średnie wartości:

| lr | MADALINE | Perceptrony |
|---|---|---|
| 0.01 | 91,1 | 51,9 |
| 0.005 | 124,8 | 84,5 |
| 0.001 | 382,1 | 413,1 |
| 0.0005 | 873,4 | 791,5 |
| 0.0001 | 4935,6 | 2652,7 |

Przykładowe błędy popełniane w czasie uczenia:




**3) Analiza i dyskusja błędów uczenia i testowania opracowanych sieci w zależności od wartości współczynnika uczenia oraz wybranego algorytmu**

Jeśli chodzi o błędy uczenia popełniane w kolejnych epokach, to widzimy podobną tendencje dla MADALINE i dla sieci złożonej z perceptronów. Błąd malał w kolejnych iteracjach skokowo.

Średnia ilość iteracji potrzebna by się nauczyć rozpoznawania wielkości liter była dla mojego przykładu mniejsza w przypadku sieci perceptronów niż dla sieci MADALINE. Jest to związane z różną regułą uczenia. Zmiana wag w neuronach ADALINE dostosowuje się w zależności od wielkości wag i wejść, natomiast w perceptronie są tylko dwie wartości modyfikujące wagi. W naszym przykładzie okazało się, że stałe bardziej drastyczne zmiany wag doprowadziły nas do wyniku szybciej, choć w wielu innych wypadkach, reguła modyfikacji ADALINE, okazuje się być szybsza.

**4. Wnioski**

Odpowiednie dobranie współczynnika nauczania i rodzaju sieci dla naszego problemu, może znacznie przyspieszyć osiąganie przez nas pożądanego wyniku. Dobranie zbyt dużego kroku uczenia może spowodować, żę w ogóle nie znajdziemy rozwiązania, natomiast dobranie zbyt małego kroku znacząco wydłuży poszukiwanie rozwiązania. Rodzaj sieci, jak widzimy w sprawozdaniu też ma znaczenie na szybkość wyszukania wyniku i warto doświadczalnie wybrać odpowiedni jego typ.

**5. Listing kodu**

Main:

```
public class Main {

    public static void main(String[] args) {
        System.out.println("Alfabet: ");

        Alfabet alfabet = new Alfabet();

        for(int i=0;i<alfabet.litery.size();i++)//przetworzenie liter na 6
danych wejsciowych
            alfabet.litery.get(i).update();

        double[][] xinputs = new double[20][6];
        int[] yexpected = new int[20];

        for(int i=0;i<20;i++)//spisywanie odpowiednio przetworzonych danych
do tablicy wejsc
        {
            for(int j=0;j<6;j++)
                xinputs[i][j] = alfabet.litery.get(i).input[j];
        }
        for(int i=0;i<20;i++)
            yexpected[i] = alfabet.litery.get(i).czyDuza;

        for(int i=0;i<20;i++) {
            alfabet.litery.get(i).wyswietl();
            System.out.println(" - ");
            for (int j = 0; j < 6; j++)
                System.out.println(xinputs[i][j]);
            System.out.println("\n"+alfabet.litery.get(i).czyDuza + "\n");
        }

        //tworzenie odpowiedniej sieci i uczenie jej
        SiecAdaline siec = new SiecAdaline(xinputs,yexpected);
        //SiecPerceptronow siec = new SiecPerceptronow(xinputs,yexpected);
        siec.train();
        System.out.println();
    }
```

```
}
```

Adaline:

```
import java.util.Random;

/**
 * Created by Daniel on 2017-10-19.
 */
public class Adaline {
    double learningRate = 0.01;
    double bias = 1;
    int nrOfBranches = 6;
    double[] weights = new double[6];
    double wb;

    public Adaline()//inicializacja losowych wag
    {
        Random r = new Random();
        for(int i=0;i<nrOfBranches;i++) {
            weights[i] = r.nextDouble();
        }
        wb = 1;
    }
    public void learn(double inputs[],int y1)//metoda nauki wywolywana
przez klase SiecAdaline
    {
        double yo = calculate(inputs);
        for (int j = 0; j < nrOfBranches; j++) {
            weights[j] += learningRate * ((double)y1 - yo) * inputs[j];
        }
        wb = wb + learningRate * ((double)y1 - yo);
    }
    public int treshhold(double sum)
    {
        if(sum>0) return 1;
        else return -1;
    }
    public double calculate(double inputs[])//obliczanie aktualnej sumy wag
razy ich wejscie
    {
        double sum = 0;
        for(int i=0;i<nrOfBranches;i++)
            sum += inputs[i]*weights[i];
        return sum + bias*wb;
    }
}
```

SiecAdaline:

```
public class SiecAdaline {
    double inputs[][];
    int y[];
    int nrOfNeurons = 6;
    Adaline[] adalines;
    int calculatedOutputs[];
    int nrOfEpochs = 10000;
    double epochError = 0;
```

```
    SiecAdaline(double inputs[][], int y[])//inicializacji odpowiednich
tablic i neuronow
    {
        this.inputs = inputs;
        this.y = y;
        this.adalines = new Adaline[nrOfNeurons];
        for(int i=0;i<nrOfNeurons;i++)
            adalines[i] = new Adaline();
        this.calculatedOutputs = new int[nrOfNeurons];
    }

    void train()
    {
        int majority;
        int idMin;
        double sumaWagxWejsc;
        double tmp;
        boolean czyJestUstalone = false;
        boolean wasModified = false;
        for(int k=0;k<nrOfEpochs;k++)
        {
            epochError = 0;
            wasModified = false;
        for(int j=0;j<20;j++) {
            majority = 0;
            for (int i = 0; i < nrOfNeurons; i++) {//obliczenie aktualnego
wyjscia przez neuron
                calculatedOutputs[i] =
adalines[i].treshhold(adalines[i].calculate(inputs[j]));
                majority += calculatedOutputs[i];
            }
            if (majority >= 0) majority = 1;
            else majority = -1;

            epochError += (majority - y[j])*(majority - y[j]);

            if (majority == y[j])//sprawdzenie czy obecne wyjsie jest
poprawne
                continue;
            else {//modyfikowanie wag odpowiedniego neuronu dla obecnego
wyjscia
                wasModified = true;
                idMin = 0;
                czyJestUstalone = false;
                sumaWagxWejsc = 0.;
                for (int i = 0; i < nrOfNeurons; i++) {//poszukiwanie
neuronu do edycji
                    if (czyJestUstalone == false) {
                        tmp = Math.abs(adalines[i].calculate(inputs[j]));
                        if
(adalines[i].treshhold(adalines[i].calculate(inputs[j])) != y[j]) {
                            idMin = i;
                            sumaWagxWejsc = tmp;
                        }
                    } else {
                        tmp = Math.abs(adalines[i].calculate(inputs[j]));
                        if (tmp < sumaWagxWejsc) {
                            idMin = i;
                            sumaWagxWejsc = tmp;
                        }
                    }
```

```
                }
                adalines[idMin].learn(inputs[j], y[j]);//wywolanie funkcji
aktualizujacej wagi
                System.out.println("Dla epoki "+k+" dla danych " + j + "
modyfikacja neuronu " + idMin);
            }
        }
        //System.out.println(epochError);
        if(wasModified == false) break;//gdy wszystko zostalo nauczone
konczymy dzialanie metody

        }
    }

    int oblicz(double inputs[])//wylicznie wyjscia dla danego wektora
wejsciowego
    {
        int majority = 0;
        for (int i = 0; i < nrOfNeurons; i++) {
            calculatedOutputs[i] =
adalines[i].treshhold(adalines[i].calculate(inputs));
            majority += calculatedOutputs[i];
        }
        if(majority >= 0) return 1;
        else  return -1;
    }


}
```

Perceptron:

```
import java.util.Random;

/**
 * Created by Daniel on 2017-10-19.
 */
public class Perceptron {
    double learningRate = 0.0001;
    double bias = 1;
    int nrOfData = 20;
    int nrOfBranches = 6;
    double[] weights = new double[6];
    double wb;

    public Perceptron(double inputs[][], int y[])
    {
        Random r = new Random();
        for(int i=0;i<nrOfBranches;i++)
            weights[i] = r.nextDouble();
        wb = 1;
    }
    public void learn(double inputs[],int y1)
    {
        double yo = treshhold(calculate(inputs));
        for (int j = 0; j < nrOfBranches; j++) {
            weights[j] += learningRate * ((double)y1 - yo) * inputs[j];
        }
        wb = wb + learningRate * ((double)y1 - yo);
    }
    public int treshhold(double sum)
    {
```

```
        if(sum>0) return 1;
        else return -1;
    }
    public double calculate(double inputs[])
    {
        double sum = 0;
        for(int i=0;i<nrOfBranches;i++)
            sum += inputs[i]*weights[i];
        return sum + bias*wb;
    }
}
```

SiecPerceptronow:

```
public class SiecPerceptronow {
    double inputs[][];
    int y[];
    int nrOfNeurons = 6;
    Perceptron[] perceptrons;
    int calculatedOutputs[];
    int nrOfEpochs = 10000;
    double epochError = 0;

    SiecPerceptronow(double inputs[][], int y[])
    {
        this.inputs = inputs;
        this.y = y;
        this.perceptrons = new Perceptron[nrOfNeurons];
        for(int i=0;i<nrOfNeurons;i++)
            perceptrons[i] = new Perceptron(inputs,y);
        this.calculatedOutputs = new int[nrOfNeurons];
    }

    void train()
    {
        int majority;
        int idMin;
        double sumaWagxWejsc;
        double tmp;
        boolean czyJestUstalone = false;
        boolean wasModified = false;
        for(int k=0;k<nrOfEpochs;k++)
        {
            epochError = 0;
            wasModified = false;
            for(int j=0;j<20;j++) {
                majority = 0;
                for (int i = 0; i < nrOfNeurons; i++) {
                    calculatedOutputs[i] =
perceptrons[i].treshhold(perceptrons[i].calculate(inputs[j]));
                    majority += calculatedOutputs[i];
                }
                if (majority >= 0) majority = 1;
                else majority = -1;
                epochError += (majority - y[j])*(majority - y[j]);

                if (majority == y[j])
                    continue;
                else {
```

```
                    wasModified = true;
                    idMin = 0;
                    czyJestUstalone = false;
                    sumaWagxWejsc = 0.;
                    for (int i = 0; i < nrOfNeurons; i++) {
                        if (czyJestUstalone == false) {
                            tmp =
Math.abs(perceptrons[i].calculate(inputs[j]));
                            if
(perceptrons[i].treshhold(perceptrons[i].calculate(inputs[j])) != y[j]) {
                                idMin = i;
                                sumaWagxWejsc = tmp;
                            }
                        } else {
                            tmp =
Math.abs(perceptrons[i].calculate(inputs[j]));
                            if (tmp < sumaWagxWejsc) {
                                idMin = i;
                                sumaWagxWejsc = tmp;
                            }
                        }
                    }
                    perceptrons[idMin].learn(inputs[j], y[j]);
                    System.out.println("Dla epoki "+k+" dla danych " + j +
" modyfikacja neuronu " + idMin);
                }
            }
            //System.out.println(epochError);
            if(wasModified == false) break;
        }
    }

    int oblicz(double inputs[])
    {
        int majority = 0;
        for (int i = 0; i < nrOfNeurons; i++) {
            calculatedOutputs[i] =
perceptrons[i].treshhold(perceptrons[i].calculate(inputs));
            majority += calculatedOutputs[i];
        }
        if(majority >= 0) return 1;
        else  return -1;
    }


}
```

Litera:

```
public class SiecPerceptronow {
    double inputs[][];
    int y[];
    int nrOfNeurons = 6;
    Perceptron[] perceptrons;
    int calculatedOutputs[];
    int nrOfEpochs = 10000;
    double epochError = 0;

    SiecPerceptronow(double inputs[][], int y[])
```

```
    {
        this.inputs = inputs;
        this.y = y;
        this.perceptrons = new Perceptron[nrOfNeurons];
        for(int i=0;i<nrOfNeurons;i++)
            perceptrons[i] = new Perceptron(inputs,y);
        this.calculatedOutputs = new int[nrOfNeurons];
    }

    void train()
    {
        int majority;
        int idMin;
        double sumaWagxWejsc;
        double tmp;
        boolean czyJestUstalone = false;
        boolean wasModified = false;
        for(int k=0;k<nrOfEpochs;k++)
        {
            epochError = 0;
            wasModified = false;
            for(int j=0;j<20;j++) {
                majority = 0;
                for (int i = 0; i < nrOfNeurons; i++) {
                    calculatedOutputs[i] =
perceptrons[i].treshhold(perceptrons[i].calculate(inputs[j]));
                    majority += calculatedOutputs[i];
                }
                if (majority >= 0) majority = 1;
                else majority = -1;
                epochError += (majority - y[j])*(majority - y[j]);

                if (majority == y[j])
                    continue;
                else {
                    wasModified = true;
                    idMin = 0;
                    czyJestUstalone = false;
                    sumaWagxWejsc = 0.;
                    for (int i = 0; i < nrOfNeurons; i++) {
                        if (czyJestUstalone == false) {
                            tmp =
Math.abs(perceptrons[i].calculate(inputs[j]));
                            if
(perceptrons[i].treshhold(perceptrons[i].calculate(inputs[j])) != y[j]) {
                                idMin = i;
                                sumaWagxWejsc = tmp;
                            }
                        } else {
                            tmp =
Math.abs(perceptrons[i].calculate(inputs[j]));
                            if (tmp < sumaWagxWejsc) {
                                idMin = i;
                                sumaWagxWejsc = tmp;
                            }
                        }
                    }
                    perceptrons[idMin].learn(inputs[j], y[j]);
                    System.out.println("Dla epoki "+k+" dla danych " + j +
" modyfikacja neuronu " + idMin);
                }
```

```
            }
            //System.out.println(epochError);
            if(wasModified == false) break;
        }
    }

    int oblicz(double inputs[])
    {
        int majority = 0;
        for (int i = 0; i < nrOfNeurons; i++) {
            calculatedOutputs[i] =
perceptrons[i].treshhold(perceptrons[i].calculate(inputs));
            majority += calculatedOutputs[i];
        }
        if(majority >= 0) return 1;
        else  return -1;
    }


}
```

Alfabet:

```
import java.util.ArrayList;
import java.util.List;

/**
 * Created by Daniel on 2017-10-03.
 */
public class Alfabet {//tworzenie danych wejsciowych w postaci liter
    Litera lita = new Litera();
    Litera litb = new Litera();
    Litera litc = new Litera();
    Litera litd = new Litera();
    Litera lite = new Litera();
    Litera litf = new Litera();
    Litera lith = new Litera();
    Litera liti = new Litera();
    Litera litl = new Litera();
    Litera lito = new Litera();

    Litera litA = new Litera();
    Litera litB = new Litera();
    Litera litC = new Litera();
    Litera litD = new Litera();
    Litera litE = new Litera();
    Litera litF = new Litera();
    Litera litH = new Litera();
    Litera litI = new Litera();
    Litera litL = new Litera();
    Litera litO = new Litera();

    List<Litera> litery = new ArrayList<Litera>();

    Alfabet()
    {
        //a
        lita.tab[8][3] = 1;
        lita.tab[8][4] = 1;
        lita.tab[8][5] = 1;
```

```
        lita.tab[8][6] = 1;
        lita.tab[9][2] = 1;
        lita.tab[10][2] = 1;
        lita.tab[11][2] = 1;
        lita.tab[12][2] = 1;
        lita.tab[8][7] = 1;
        lita.tab[9][7] = 1;
        lita.tab[10][7] = 1;
        lita.tab[11][7] = 1;
        lita.tab[12][7] = 1;
        lita.tab[13][3] = 1;
        lita.tab[13][4] = 1;
        lita.tab[13][5] = 1;
        lita.tab[13][6] = 1;
        lita.tab[13][8] = 1;

        //b
        litb.tab[8][3] = 1;
        litb.tab[8][4] = 1;
        litb.tab[8][5] = 1;
        litb.tab[8][6] = 1;
        litb.tab[9][2] = 1;
        litb.tab[10][2] = 1;
        litb.tab[11][2] = 1;
        litb.tab[12][2] = 1;
        litb.tab[9][7] = 1;
        litb.tab[10][7] = 1;
        litb.tab[11][7] = 1;
        litb.tab[12][7] = 1;
        litb.tab[13][3] = 1;
        litb.tab[13][4] = 1;
        litb.tab[13][5] = 1;
        litb.tab[13][6] = 1;
        for(int i = 1;i<14;i++)
            litb.tab[i][2] = 1;

        //c
        litc.tab[8][3] = 1;
        litc.tab[8][4] = 1;
        litc.tab[8][5] = 1;
        litc.tab[8][6] = 1;
        litc.tab[9][2] = 1;
        litc.tab[10][2] = 1;
        litc.tab[11][2] = 1;
        litc.tab[12][2] = 1;
        litc.tab[13][3] = 1;
        litc.tab[13][4] = 1;
        litc.tab[13][5] = 1;
        litc.tab[13][6] = 1;

        //d
        litd.tab[8][3] = 1;
        litd.tab[8][4] = 1;
        litd.tab[8][5] = 1;
        litd.tab[8][6] = 1;
        litd.tab[9][2] = 1;
        litd.tab[10][2] = 1;
        litd.tab[11][2] = 1;
        litd.tab[12][2] = 1;
        litd.tab[9][7] = 1;
        litd.tab[10][7] = 1;
```

```
        litd.tab[11][7] = 1;
        litd.tab[12][7] = 1;
        litd.tab[13][3] = 1;
        litd.tab[13][4] = 1;
        litd.tab[13][5] = 1;
        litd.tab[13][6] = 1;
        for(int i = 1;i<14;i++)
            litd.tab[i][7] = 1;

        //e
        lite.tab[8][3] = 1;
        lite.tab[8][4] = 1;
        lite.tab[8][5] = 1;
        lite.tab[8][6] = 1;
        lite.tab[9][2] = 1;
        lite.tab[10][2] = 1;
        lite.tab[11][2] = 1;
        lite.tab[12][2] = 1;
        lite.tab[9][7] = 1;
        lite.tab[10][7] = 1;
        lite.tab[13][3] = 1;
        lite.tab[13][4] = 1;
        lite.tab[13][5] = 1;
        lite.tab[13][6] = 1;
        lite.tab[13][7] = 1;
        lite.tab[10][3] = 1;
        lite.tab[10][4] = 1;
        lite.tab[10][5] = 1;
        lite.tab[10][6] = 1;

        //f
        for(int i = 4;i<14;i++)
            litf.tab[i][2] = 1;
        litf.tab[3][3] = 1;
        litf.tab[2][4] = 1;
        litf.tab[2][5] = 1;
        litf.tab[6][1] = 1;
        litf.tab[6][3] = 1;
        litf.tab[6][4] =1;


        //h
        for(int i = 1;i<14;i++)
            lith.tab[i][2] = 1;
        lith.tab[8][3] = 1;
        lith.tab[8][4] = 1;
        lith.tab[8][5] = 1;
        lith.tab[8][6] = 1;
        lith.tab[9][2] = 1;
        lith.tab[9][7] = 1;
        lith.tab[10][7] = 1;
        lith.tab[11][7] = 1;
        lith.tab[12][7] = 1;
        lith.tab[13][7] = 1;

        //i
        for(int i = 8;i<14;i++)
            liti.tab[i][4] = 1;
        liti.tab[6][4] =1;

        //l
```

```
        for(int i = 1;i<13;i++)
            litl.tab[i][2] = 1;
        litl.tab[13][3] = 1;
        litl.tab[13][4] = 1;
        litl.tab[13][5] = 1;

        //o
        lito.tab[8][3] = 1;
        lito.tab[8][4] = 1;
        lito.tab[8][5] = 1;
        lito.tab[8][6] = 1;
        lito.tab[9][2] = 1;
        lito.tab[10][2] = 1;
        lito.tab[11][2] = 1;
        lito.tab[12][2] = 1;
        lito.tab[9][7] = 1;
        lito.tab[10][7] = 1;
        lito.tab[11][7] = 1;
        lito.tab[12][7] = 1;
        lito.tab[13][3] = 1;
        lito.tab[13][4] = 1;
        lito.tab[13][5] = 1;
        lito.tab[13][6] = 1;


        //A
        litA.czyDuza = 1;
        for(int i=1;i<14;i++)
            litA.tab[i][0] = 1;
        for(int i=1;i<14;i++)
            litA.tab[i][9] = 1;
        for(int i=1;i<9;i++)
            litA.tab[0][i] = 1;
        for(int i=1;i<9;i++)
            litA.tab[8][i] =1;

        //B
        litB.czyDuza = 1;
        for(int i=1;i<14;i++)
            litB.tab[i][0] = 1;
        for(int i=1;i<14;i++)
            litB.tab[i][9] = 1;
        for(int i=1;i<9;i++)
            litB.tab[0][i] = 1;
        for(int i=1;i<9;i++)
            litB.tab[7][i] =1;
        for(int i=1;i<9;i++)
            litB.tab[13][i] =1;
        litB.tab[7][9] =0;

        //C
        litC.czyDuza = 1;
        for(int i=1;i<13;i++)
            litC.tab[i][0] = 1;
        for(int i=1;i<9;i++)
            litC.tab[0][i] = 1;
        for(int i=1;i<9;i++)
            litC.tab[13][i] = 1;

        //D
        litD.czyDuza =1;
```

```
        for(int i=0;i<14;i++)
            litD.tab[i][0] = 1;
        for(int i=1;i<9;i++)
            litD.tab[0][i] = 1;
        for(int i=1;i<9;i++)
            litD.tab[13][i] = 1;
        for(int i=1;i<13;i++)
            litD.tab[i][9] = 1;

        //E
        litE.czyDuza = 1;
        for(int i=0;i<14;i++)
            litE.tab[i][0] = 1;
        for(int i=1;i<10;i++)
            litE.tab[0][i] = 1;
        for(int i=1;i<10;i++)
            litE.tab[13][i] = 1;
        for(int i=1;i<10;i++)
            litE.tab[7][i] = 1;

        //F
        litF.czyDuza =1;
        for(int i=0;i<14;i++)
            litF.tab[i][0] = 1;
        for(int i=1;i<10;i++)
            litF.tab[0][i] = 1;
        for(int i=1;i<9;i++)
            litF.tab[7][i] = 1;

        //H
        litH.czyDuza = 1;
        for(int i=0;i<14;i++)
            litH.tab[i][0] = 1;
        for(int i=0;i<14;i++)
            litH.tab[i][9] = 1;
        for(int i=1;i<10;i++)
            litH.tab[8][i] = 1;

        //I
        litI.czyDuza = 1;
        for(int i=0;i<14;i++)
            litI.tab[i][4] = 1;

        //L
        litL.czyDuza = 1;
        for(int i=0;i<14;i++)
            litL.tab[i][0] = 1;
        for(int i=1;i<10;i++)
            litL.tab[13][i] = 1;

        //O
        litO.czyDuza = 1;
        for(int i=1;i<13;i++)
            litO.tab[i][0] = 1;
        for(int i=1;i<13;i++)
            litO.tab[i][9] = 1;
        for(int i=1;i<9;i++)
            litO.tab[13][i] = 1;
        for(int i=1;i<9;i++)
            litO.tab[0][i] = 1;
```

```
        litery.add(lita);
        litery.add(litb);
        litery.add(litc);
        litery.add(litd);
        litery.add(lite);
        litery.add(litf);
        litery.add(lith);
        litery.add(liti);
        litery.add(litl);
        litery.add(lito);

        litery.add(litA);
        litery.add(litB);
        litery.add(litC);
        litery.add(litD);
        litery.add(litE);
        litery.add(litF);
        litery.add(litH);
        litery.add(litI);
        litery.add(litL);
        litery.add(litO);
    }
}
```

Alfabet:

0000000000
0000000000
0000000000
0000000000
0000000000
0000000000
0000000000
0000000000
0001111100
0010000100
0010000100
0010000100
0010000100
0001111010

0000000000
0010000000
0010000000
0010000000
0010000000
0010000000
0010000000
0010000000
0011111000
0010000100
0010000100
0010000100

0010000100
0011111000

0000000000
0000000000
0000000000
0000000000
0000000000
0000000000
0000000000
0000000000
0001111000
0010000000
0010000000
0010000000
0010000000
0001111000

0000000000
0000000100
0000000100
0000000100
0000000100
0000000100
0000000100
0000000100
0001111100
0010000100
0010000100
0010000100
0010000100
0001111100

0000000000
0000000000
0000000000
0000000000
0000000000
0000000000
0000000000
0000000000
0001111000
0010000100
0011111100
0010000000
0010000000
0001111100

0000000000
0000000000
0000110000
0001000000
0010000000
0010000000
0111100000
0010000000
0010000000
0010000000
0010000000
0010000000
0010000000
0010000000

0000000000
0010000000
0010000000
0010000000
0010000000
0010000000
0010000000
0010000000
0010000000
0011111000
0010000100
0010000100
0010000100
0010000100
0010000100

0000000000
0000000000
0000000000
0000000000
0000000000
0000000000
0000100000
0000000000
0000100000
0000100000
0000100000
0000100000
0000100000
0000100000

0000000000
0010000000
0010000000

0010000000
0010000000
0010000000
0010000000
0010000000
0010000000
0010000000
0010000000
0010000000
0010000000
0001110000

0000000000
0000000000
0000000000
0000000000
0000000000
0000000000
0000000000
0000000000
0001111000
0010000100
0010000100
0010000100
0010000100
0001111000

0111111110
1000000001
1000000001
1000000001
1000000001
1000000001
1000000001
1000000001
1111111111
1000000001
1000000001
1000000001
1000000001
1000000001

0111111110
1000000001
1000000001
1000000001
1000000001
1000000001

1000000001
1111111110
1000000001
1000000001
1000000001
1000000001
1000000001
1111111111

0111111110
1000000000
1000000000
1000000000
1000000000
1000000000
1000000000
1000000000
1000000000
1000000000
1000000000
1000000000
0111111110

1111111110
1000000001
1000000001
1000000001
1000000001
1000000001
1000000001
1000000001
1000000001
1000000001
1000000001
1000000001
1111111110

1111111111
1000000000
1000000000
1000000000
1000000000
1000000000
1000000000
1111111111
1000000000

1000000000
1000000000
1000000000
1000000000
1111111111

1111111111
1000000000
1000000000
1000000000
1000000000
1000000000
1000000000
1111111110
1000000000
1000000000
1000000000
1000000000
1000000000
1000000000

1000000001
1000000001
1000000001
1000000001
1000000001
1000000001
1000000001
1000000001
1111111111
1000000001
1000000001
1000000001
1000000001
1000000001

0000100000
0000100000
0000100000
0000100000
0000100000
0000100000
0000100000
0000100000
0000100000
0000100000
0000100000
0000100000
0000100000

0000100000
0000100000

1000000000
1000000000
1000000000
1000000000
1000000000
1000000000
1000000000
1000000000
1000000000
1000000000
1000000000
1000000000
1111111111

0111111110
1000000001
1000000001
1000000001
1000000001
1000000001
1000000001
1000000001
1000000001
1000000001
1000000001
1000000001
0111111110

Źródła:
wikipedia.org
https://www.scribd.com/document/77995052/Adaline-Madaline