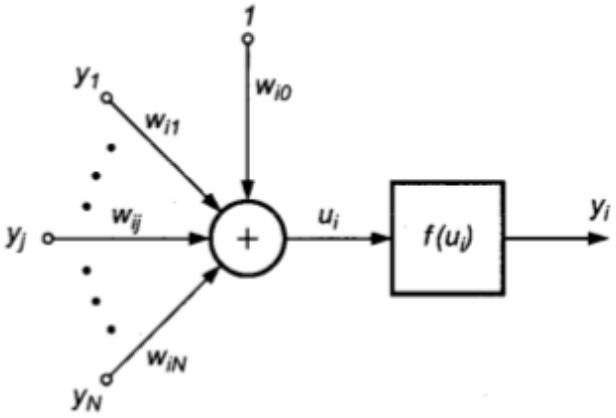
Daniel Wenecki
Podstawy Sztucznej Inteligencji
Sprawozdanie – Projekt nr 4

1.Cel ćwiczenia
Celem ćwiczenia jest poznanie działania reguły Hebba na przykładzie rozpoznawania emotikon.

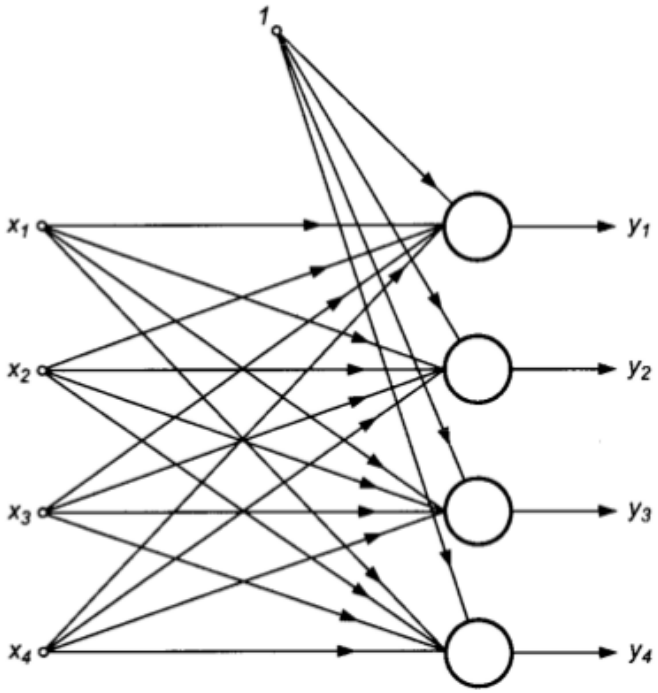
2.Syntetyczny opis budowy użytej sieci i algorytmów uczenia.

Do ćwiczenia użyłem jednowarstwowej sieci Hebba, bez nauczyciela, dla różnych wariantów współczynnika zapominania.



Rys. 2.13. Ogólny model neuronu Hebba

Schemat sieci Hebba:



Wzór modyfikacji wag:

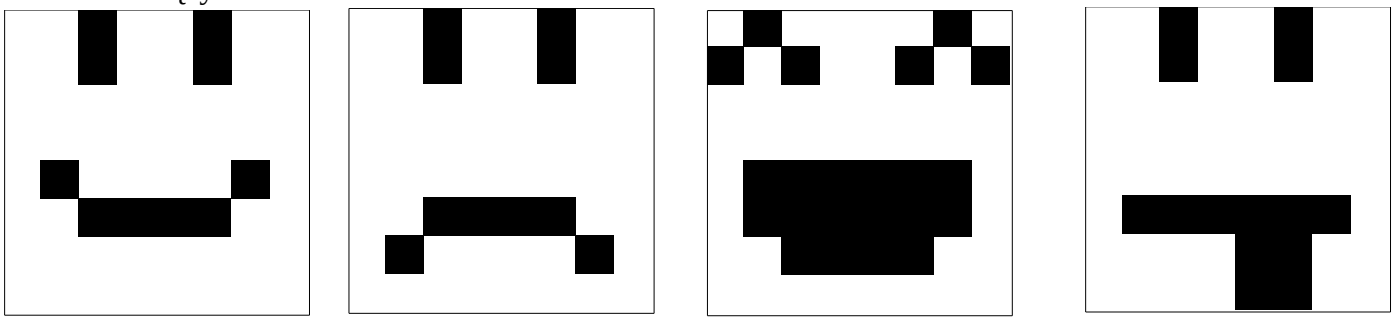
$$w_{ij}(k+1) = (1-\gamma)w_{ij}(k) + \Delta w_{ij}$$

Jako funkcje aktywacji użyłem unipolarną sigmoidalną funkcję:

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

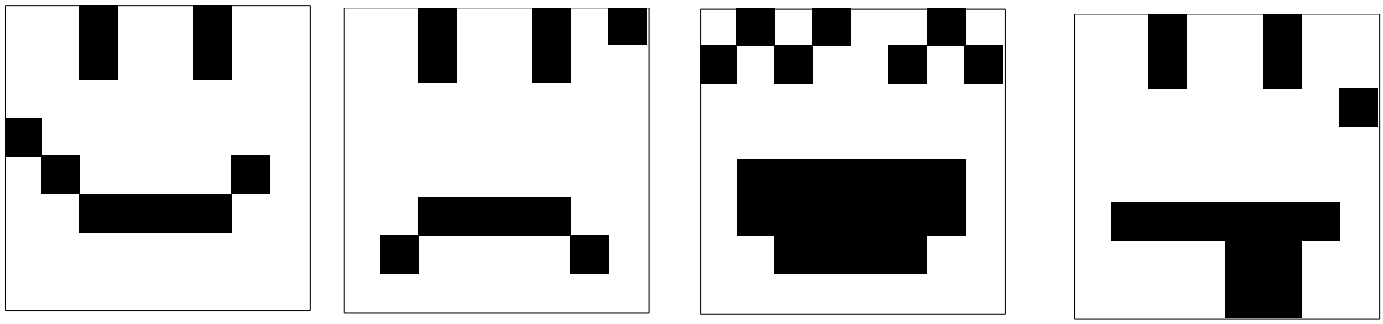
Użyte emotikony:
Emotikony stworzony na tablicach 8x8, ich odwzorowanie w programie to tablica jedynek i zer.
(1 oznacza czarne pole, 0 białe).

Zestaw uczący:



Zestaw testujący wygenerowano poprzed dodanie losowego piksela na wolnej przestrzeni.

Zestaw testujący:



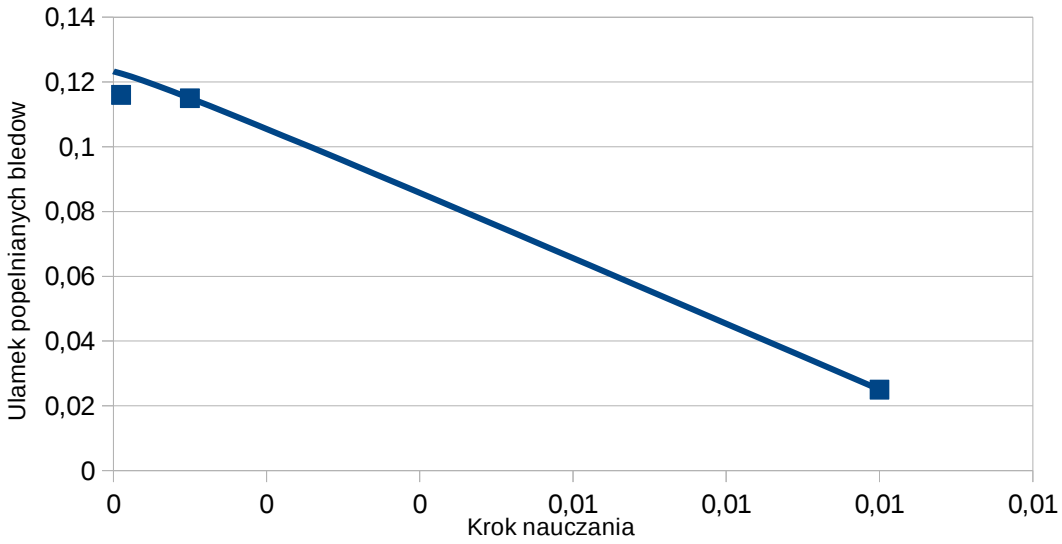
3.Zestawienie otrzymanych wyników.

Początkowo sieci uczyłem dla różych kroków uczenia i zapominania, a za warunek zakończenia nauczania przyjąłem moment w którym dla każdego zestawu z danych uczących reagował inny neuron.
Średnie wyniki dla 100 000 przypadków różnych sieci:

fr\lr	0,01	0,001	0,0001
0,01	1,35	7,72	71,38
0,001	1,36	7,77	71,515
0,0001	1,36	7,8	71,95
0	1,36	7,79	72,12

Zmierzyłem także błąd popełniany przez sieci nauczone w w.w. sposób dla zaszumionych danych.

lr	0,01	0,001	0,0001
Ułamek błędów	0,025	0,115	0,116



Następnie dokonałem modyfikacji programu, tak aby sieć uczyła się stałą liczbę iteracji, większą niż liczbe iteracji potrzebną by się nauczyć. W moim przypadku sprawiłem, aby sieć zawsze wykonywała 1000 iteracji. W takim przypadku ułamek błędów popełnianych przez sieć dla naszych zaszumionych danych zawsze wynosił 0, niezależnie od współczynnika uczenia i zapominania.

Dla stałej liczby wykonanych iteracji (1000).

lr	0,01	0,001	0,0001
Ułamek błędów	0	0	0

4.Sformułowanie wniosków

Z pierwszej tabeli możemy zauważyć wyraźny wpływ współczynnik uczenia od ilości potrzebnych iteracji by sieć się nauczyła. Jest tak, ponieważ krok nauczania określa wielkość o jaką modyfikujemy wagę. Ustawienie większej wartości przyspieszy rozwiązanie, ale ustawienie mniejszych wartości może dać nam dokładniejszy wynik.

Z tabeli możemy zauważyć też, że krok zapomianania ma wpływ na ilość potrzebnych iteracji. Oznacza to, że ustawienie jego odpowieniej wartości, może przyspieszyć znalezienie przez nas poszukiwanego wyniku.

Z tabeli z współczynnikiem nauczania i błędów popełaniancych przez sieć przy zaszumionych danych możemmy wywnioskować, że w przypadku sieci bez nauczyciela, dalsze nauczanie sieci poprawia jej zdolność do klasyfikacji podanych jej danych.

5. Listing kodu z komentarzami

```
import java.util.Random;
public class Hebb {
    double krokNauczania = 0.0001;
    double krokZapominania = 0.01;
    int iloscWejsc;
    double[] wagi;
    Siec siec;
```

```
    public Hebb(int iloscWejsc,Siec siec)//wprowadzam ilosc wejsc i losuje
wagi poczatkowe
    {
        this.siec = siec;
        this.iloscWejsc = iloscWejsc;
        wagi = new double[iloscWejsc];
        Random r = new Random();
        for(int i = 0; i< iloscWejsc; i++)
            wagi[i] = r.nextDouble();
    }
    public void learn(double inputs[],double yo)// w tej metodzie modyfikuje
wagi neuronu, a pozniej je normalizuje
    {
        for (int j = 0; j < iloscWejsc; j++) {
            wagi[j] = (1 - krokZapominania)*wagi[j] + krokNauczania * yo *
inputs[j];
        }
        normalizujWagi();
    }
    public double funkcjaAktywacji(double sum) // sigmoidalna funkcja
aktywacji
    {
       return (1.0/(1.0 + Math.pow(Math.E, - sum)));
        //return sum;
    }
    public double oblicz(double inputs[]) // zwraca sume wag razy podane
wejscie
    {
        double sum = 0;
        for(int i = 0; i< iloscWejsc; i++)
            sum += inputs[i]* wagi[i];
        return sum ;
    }
    public void normalizujWagi() // normalizacja w celu unikniecia zbyt
duzych wartosci wag
    {
        double mianownik = 0.;
        for(int i=0;i<iloscWejsc;i++)
            mianownik += wagi[i]*wagi[i];
        mianownik = Math.sqrt(mianownik);
        for(int i=0;i<iloscWejsc;i++)
            wagi[i] = wagi[i]/mianownik;
    }
}

public class Siec {
    int iloscWejsc = 65;
    int iloscZestawow = 4;
    int iloscNeuronow = 4;
    int[] wygrani = new int[iloscNeuronow];
    Hebb[] neuronyHebba = new Hebb[iloscNeuronow];
    ZestawUczacy zestawUczacy = new ZestawUczacy();
    Siec()// tworzenie sieci
    {
        int wygrany;
        for(int i=0;i<iloscNeuronow;i++)
            neuronyHebba[i] = new Hebb(iloscWejsc,this);
```

```
        for( int i=0;i<iloscNeuronow;i++)
        wygrani[i] = -1;
    }
    int naucz()// algorytm uczacy siec i zwracajacy ilosc iteracji
potrzebnych do jej nauczenia, zwraca -1 jesli nie nauczy sie w podanym
limicie
    {
        double yo[] = new double[iloscZestawow];
        int licznik = 1;
        int limit = 1000;
        //while(true)
        while(!wygraniSaRozni())
        {
            for(int i=0;i<iloscZestawow;i++)
            {
                double y1 = getNeuralNetOutput(i);
                for(int j=0;j<iloscNeuronow;j++)
                    neuronyHebba[j].learn(zestawUczacy.emotikony[j],y1);
                    wygrani[i] = -1;
            }
            for(int i=0;i<iloscZestawow;i++)
                wygrani[i] = znajdzZwyciezce(i);
            //System.out.println("Licznik w metodzie naucz: " + licznik);
            licznik++;
            if(licznik>limit)
            {
                //System.out.println("Licznik przekroczył maksymalną wartość
sieć nie jest nauczona");
                licznik = -1;
                break;
            }
        }
        return licznik;
    }
    boolean wygraniSaRozni()// sprawdza czy nie ma aktywowanego tego samego
neuronow dla roznych danych
    {
        for(int i=0;i<iloscNeuronow;i++)
            for(int j=i;j<iloscNeuronow;j++)
                if(i!=j)
                    if(wygrani[i] == wygrani[j])
                        return false;
        return true;
    }
    int znajdzZwyciezce(int nrZestawu)// szuka najbardziej aktywnego neuronu
    {
        double max =
neuronyHebba[0].funkcjaAktywacji(neuronyHebba[0].oblicz(zestawUczacy.emotiko
ny[nrZestawu]));
        int zwyciezca = 0;
        for(int i=1;i<iloscNeuronow;i++)

if(neuronyHebba[i].funkcjaAktywacji(neuronyHebba[i].oblicz(zestawUczacy.emot
ikony[nrZestawu])) > max)
            {
```

```
                max =
neuronyHebba[i].funkcjaAktywacji(neuronyHebba[i].oblicz(zestawUczacy.emotiko
ny[nrZestawu]));
                zwyciezca = i;
            }
        return zwyciezca;
    }
    int znajdzZwyciezceDlaTestu(double[] zestaw)
    {
        double max =
neuronyHebba[0].funkcjaAktywacji(neuronyHebba[0].oblicz(zestaw));
        int zwyciezca = 0;
        for(int i=1;i<iloscNeuronow;i++)

if(neuronyHebba[i].funkcjaAktywacji(neuronyHebba[i].oblicz(zestaw)) > max)
            {
                max =
neuronyHebba[i].funkcjaAktywacji(neuronyHebba[i].oblicz(zestaw));
                zwyciezca = i;
            }
        return zwyciezca;
    }
    void wyswietlAktualnyStan()
    {
        int wygrany;
        for(int i=0;i<iloscNeuronow;i++)
        {
            wygrany = znajdzZwyciezce(i);
            System.out.println("Dla zestawu " + i + " zwyciezca jest neuron
nr: " + wygrany);
        }
    }
    void wyswietlDlaPodanychDanych(double[][] zestawTestujacy)
    {
        int wygrany;
        for(int i=0;i<iloscNeuronow;i++)
        {
            wygrany = znajdzZwyciezceDlaTestu(zestawTestujacy[i]);
            System.out.println("Dla zestawu " + i + " zwyciezca jest neuron
nr: " + wygrany);
        }
    }
    double procentNiepoprawnychDlaPodanegoZestawu(double[][]
zestawTestujacych)
    {
        int[] tab = new int[iloscZestawow];
        for(int i=0;i<iloscZestawow;i++)
            tab[i] = znajdzZwyciezceDlaTestu(zestawTestujacych[i]);
        int licznik = 0;
        for(int i=0;i<iloscNeuronow;i++)
            for(int j=i;j<iloscNeuronow;j++)
                if(i!=j)
                    if(tab[i] == tab[j])
                        licznik++;
        double ulamekUdzialu = (double)licznik/iloscZestawow;
        return ulamekUdzialu;
    }
```

```
    double getNeuralNetOutput(int nrZestawu)
    {
        double output = 0;
        for(int i=0;i<iloscNeuronow;i++){

output+=neuronyHebba[i].funkcjaAktywacji(neuronyHebba[i].oblicz(zestawUczacy
.emotikony[nrZestawu]));
        }
        return output;
    }
}

public class ZestawUczacy {
    //w tej klasie tworze zestaw uczacy i testujacy
    public double [][] emotikony
={{1,0,0,1,0,0,1,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,
0,0,0,1,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

{1,0,0,1,0,0,1,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,1,1,1,1,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0},

{1,0,1,0,0,0,0,1,0,1,0,1,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,
1,1,1,0,0,1,1,1,1,1,1,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0},

{1,0,0,1,0,0,1,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,1,1,0,0,0,0,0,0,1,1,0,0}}
            ;
    public double [][] zaszumionyZestaw =
{{1,0,0,1,0,0,1,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0
,0,0,1,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

{1,0,0,1,0,0,1,0,1,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,1,1,1,1,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0},

{1,0,1,0,1,0,0,1,0,1,0,1,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,
1,1,1,0,0,1,1,1,1,1,1,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0},

{1,0,0,1,0,0,1,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,1,1,0,0,0,0,0,0,1,1,0,0}}
            ;
    void wyswietl(double[][] zestaw)
    {
        for(int i=0;i<4;i++)
            for(int k=0;k<8;k++) {
                for (int l = 0; l < 8; l++)
                    System.out.print(zestaw[i][(k * 8 + l) +1] + " ");
                System.out.println();
                if(k==7) System.out.println("\n");
            }
    }
}

public class Main {
    public static void main(String[] args) {
        ZestawUczacy zestawUczacy = new ZestawUczacy();
        int iloscSieci = 1000;
        Siec[] sieci = new Siec[iloscSieci];//tworzenie wiekszej ilosci
sieci w celu sredniego pomiaru ich wlasciwosci
```

```
        double suma = 0.0;
        int wyjscie = 0;
        for(int i=0;i<iloscSieci;i++)
        {
            sieci[i] = new Siec();
            wyjscie = sieci[i].naucz();
            if(wyjscie != -1)
                suma += (double)wyjscie;
            else
                System.out.println("Siec "+i+" nie nauczyla sie");
        }
        suma = suma/(double)iloscSieci;
        System.out.println("Średnia ilość iteracji wyniosła: " + suma);
        double sredniUlamekNiepoprawnych = 0.;//sprawdzanie bledu dla
zaszumionych danych
        for(int i=0;i<iloscSieci;i++)
        {
            sredniUlamekNiepoprawnych+=
sieci[i].procentNiepoprawnychDlaPodanegoZestawu(zestawUczacy.zaszumionyZesta
w);
        }
        sredniUlamekNiepoprawnych = sredniUlamekNiepoprawnych/
(double)iloscSieci;
        System.out.println("Średni procent niepoprawnych odpowiedzi wyniosl:
" + sredniUlamekNiepoprawnych);
        System.out.println("Test daych przy użyciu danych zaszumionych");
        sieci[0].wyswietlDlaPodanychDanych(zestawUczacy.zaszumionyZestaw);
    }
}
```

Bibliografia:
Stanisław Osowski : Sieci neuronowe do przetwarzania informacji.
https://pl.wikipedia.org