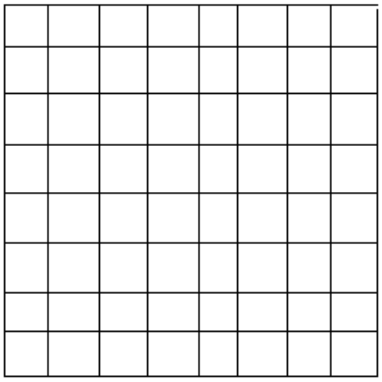
Rafał Szyszka

Sprawozdanie 4 – Uczenie sieci regułą Hebba.

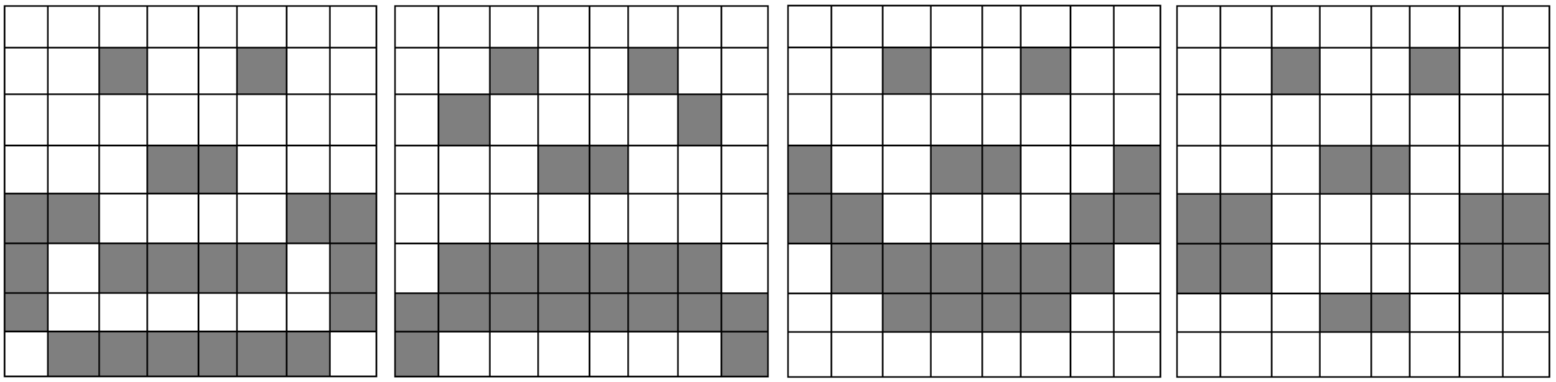
1. **Cel ćwiczenia:**

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

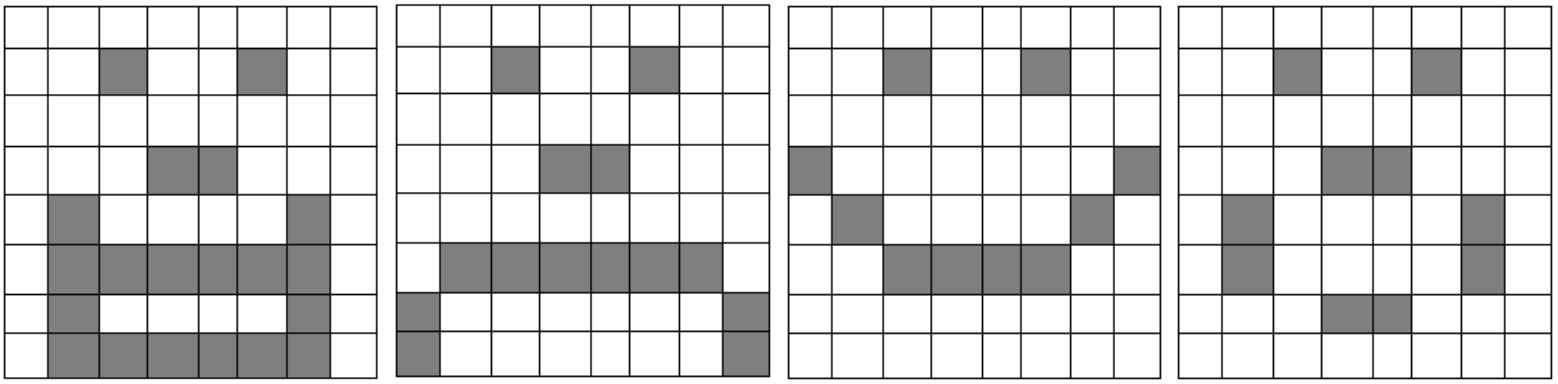
2. **Przebieg wykonania ćwiczenia**
   a) Przygotowanie danych uczących i testujących dla 4 różnych emotikonek na siatce o wymiarze 8x8



Dane uczące:

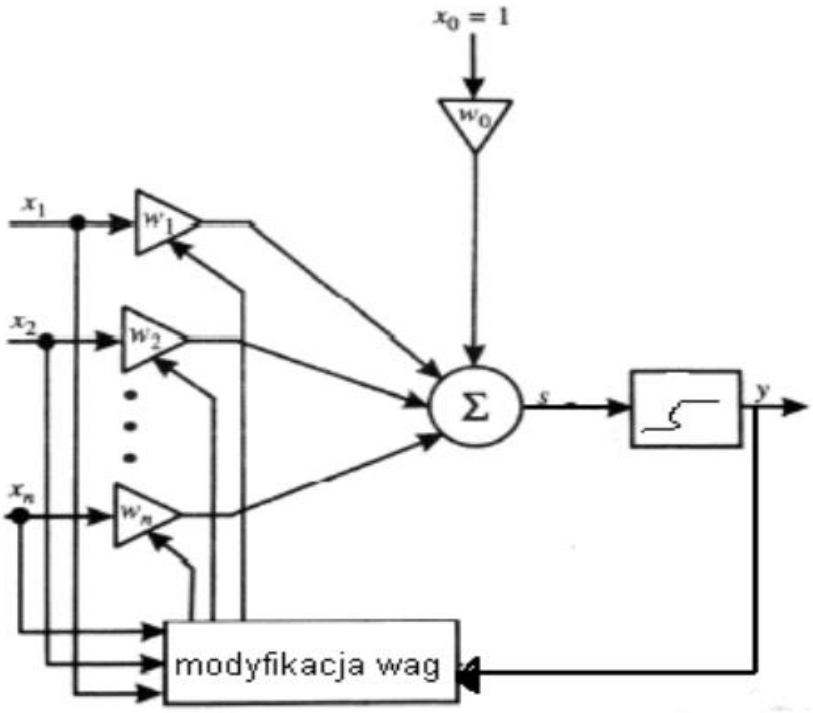


Dane testujące:



   b) Implementacja sieci oraz reguły Hebba w wersji z i bez współczynnika zapominania
   c) Uczenie sieci dla różnych współczynników uczenia i zapominania
   d) Testowanie sieci

3. Syntetyczny opis budowy użytej sieci i algorytmów uczenia:



Rys. 1 – Model neuronu Hebba

Ogólny model neuronu Hebba, przedstawiony na rys1. odpowiada standardowej postaci modelu neuronu. Waga w_{ij} włączona jest między sygnałem wejściowym y_j a węzłem sumacyjnym i-tego neuronu o sygnale wyjściowym y_j. Uczenie neuronu z zastosowaniem reguły Hebba w trybie bez nauczyciela polega na używaniu aktualnej wartości y_i sygnału wyjściowego neuronu.

Jeśli j-ta komórka o sygnale wyjściowym y_j powiązana jest z i-tą o sygnale wyjściowym y_i przez wagę w_{ij} to na stan powiązań tych komórek wpływają wartości sygnałów wyjściowych y_j oraz y_i. Zmiana wagi w_{ij} odbywa się proporcjonalnie do iloczynu jego sygnału wejściowego oraz wyjściowego.

$$\Delta w_{ij} = \eta y_j y_i$$

gdzie η jest stałą uczenia z przedziału 0 – 1. y_j – sygnał wyjściowy neuronu y_i – wartość wejściowa.

Reguła Hebba charakteryzuje się tym, że w jej wyniku wagi mogą przybierać wartości dowolnie duże, gdyż w każdym cyklu uczącym następuje proces sumowania aktualnych wartości wag i skończonego przyrostu Δw_{ij}

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

Jedną z metod poprawy stabilności procesu uczenia wg reguły Hebba jest przyjęcie przy aktualizacji wag nie ostatnie wartości w_{ij}, ale wartości zmniejszonej o tak zwany współczynnik zapominania γ. Wówczas regułę Hebba można zapisać w postaci:

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

Współczynnik zapominania γ zawiera się zwykle w przedziale 0 – 1.

4. Zestawienie i analiza otrzymanych wyników.

a) Wyniki dla przypadku z współczynnikiem zapominania

Tabela 1 – zbiorcze zestawienie wyników.

	lr	0,01		lr	0,05		lr	0,005		lr	0,0075		lr	0,025
	fr	0,003		fr	0,016		fr	0,0016		fr	0,0025		fr	0,0083
nr testu	l. epok	l. popr.		l. epok	l. popr.		l. epok	l. popr.		l. epok	l. popr.		l. epok	l. popr.
1	193	75%		34	25%		10	50%		9	75%		8	50%
2	206	25%		1	50%		12	0%		276	50%		90	75%
3	1	50%		6	25%		19	25%		17	0%		1	25%
4	10	50%		3	75%		13	25%		288	50%		5	50%
5	10	75%		5	50%		434	50%		225	25%		4	0%
6	14	25%		37	50%		162	25%		142	25%		4	50%
7	208	75%		1	0%		322	25%		12	75%		5	25%
8	12	100%		1	25%		178	0%		345	25%		38	50%
9	17	25%		2	50%		249	75%		275	50%		6	25%
10	212	50%		2	0%		414	25%		23	50%		27	25%
ŚREDNIO	**88,3**	**55%**		**9,2**	**35%**		**181,3**	**30%**		**161,2**	**43%**		**18,8**	**38%**

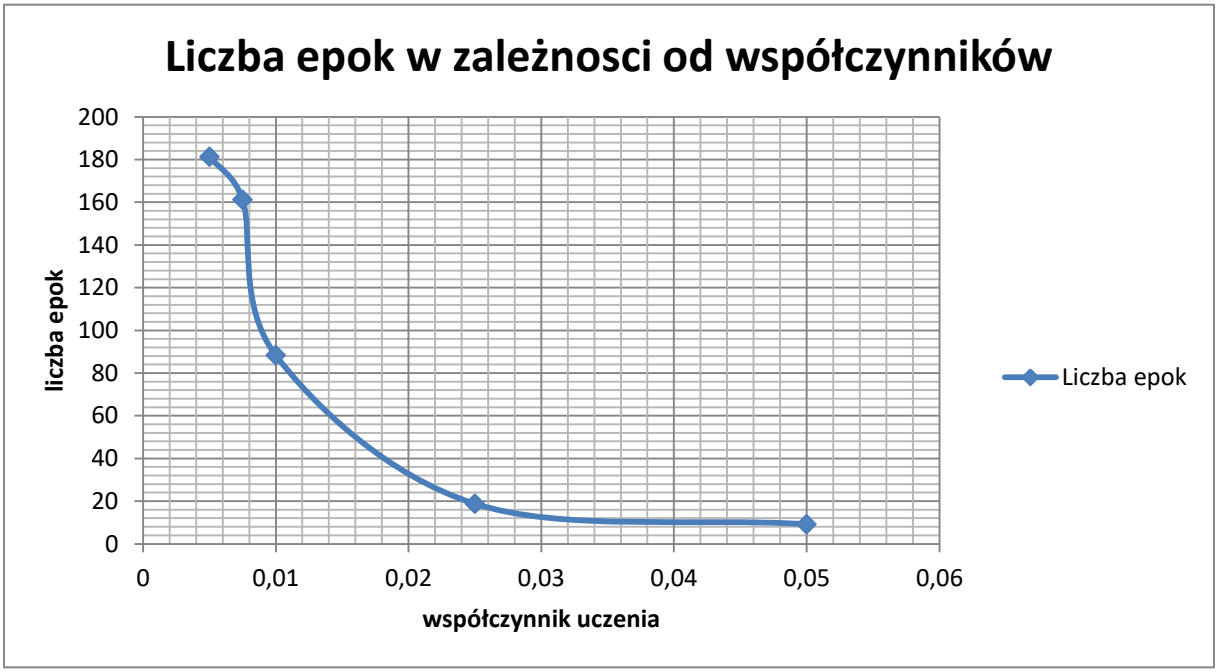
Tabela 1 przestawia zbiorcze zestawienie wyników dla różnych współczynników uczenia i zapominania. Dla każdego zestawu współczynników zostało wykonane 10 testów, w których zmierzone zostały liczba epok potrzebnych do nauczenia sieci, oraz % poprawnych odpowiedzi dla danych testujących.

Tabela 2 – średnie wyniki dla danych współczynników uczenia i zapominania

lr	fr	l. epok	%poprawnych
0,005	0,0016	181,3	30%
0,0075	0,0025	161,2	43%
0,01	0,003	88,3	55%
0,025	0,0083	18,8	38%
0,05	0,016	9,2	35%

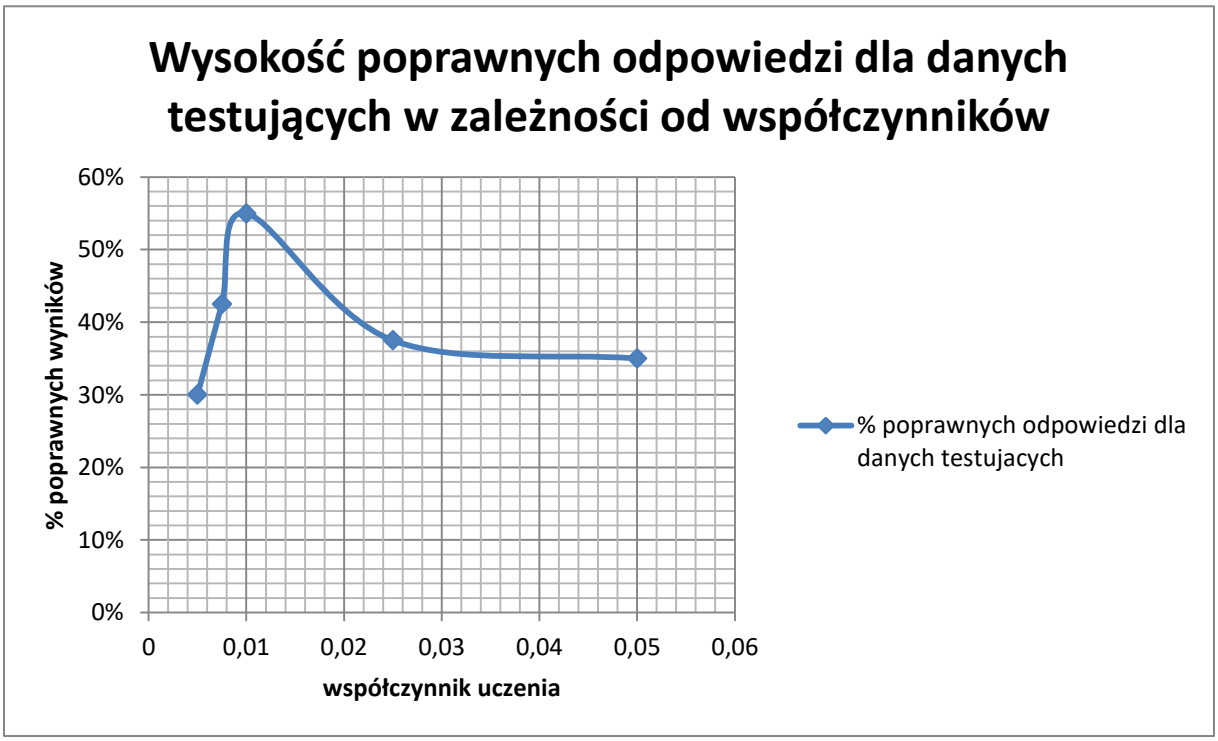
W opracowaniu wyników zostało przyjęte, że współczynnik zapominania stanowi $\frac{1}{3}$ współczynnika uczenia. Na powyższej tabelce widzimy, że średnia epok wraz ze wzrostem learning rate i forget rate maleje. Zaś % poprawnych wyników dla danych testujących wskazuje, że optymalny współczynnik uczenia oscyluje w granicach 0,01. Powyższe wyniki przedstawiono także na wykresie 1 i 2.

Wykres 1.



Na powyższym wykresie jasno widać, że liczba epok wraz ze wzrostem współczynników uczenia i zapominania maleje.

Wykres 2.



Tutaj zauważamy, jak znalezione zostało optymalne ustawienie współczynników uczenia i zapominania dla naszej sieci i danych uczących i testujących.

Analizując powyższe wyniki widzimy, że mimo ciągłego spadku liczby epok potrzebnych do nauczenia naszej sieci, liczba popełnianych błędów była najmniejsza dla współczynnika uczenia 0,01 i jego odpowiednika współczynnika zapominania 0,0033.

b) Wyniki dla przypadku bez współczynnika zapominania.

Tabela 3. – wyniki zbiorcze.

	lr	0,005		lr	0,0075		lr	0,01		lr	0,025		lr	0,05
nr testu	l. epok	% popr.		l. epok	% popr.		l. epok	% porp.		l epok	% popr.		l. epok	% popr.
1	9	25%		19	50%		20	50%		3	50%		3	25%
2	434	0%		12	75%		213	75%		85	50%		45	25%
3	49	25%		13	50%		13	50%		1	50%		2	50%
4	17	75%		280	50%		8	50%		3	75%		3	50%
5	29	25%		6	0%		145	25%		5	25%		6	25%
6	256	100%		19	25%		10	25%		11	0%		45	50%
7	9	50%		270	75%		216	25%		88	50%		2	25%
8	428	75%		98	50%		202	25%		6	25%		3	50%
9	1	50%		294	50%		12	0%		4	25%		46	75%
10	48	0%		2	25%		1	25%		92	25%		2	50%
SREDNIO	**128**	**43%**		**101,3**	**45%**		**84**	**35%**		**29,8**	**38%**		**15,7**	**43%**

Tabela 4 – średnie wyniki dla danego współczynnika uczenia

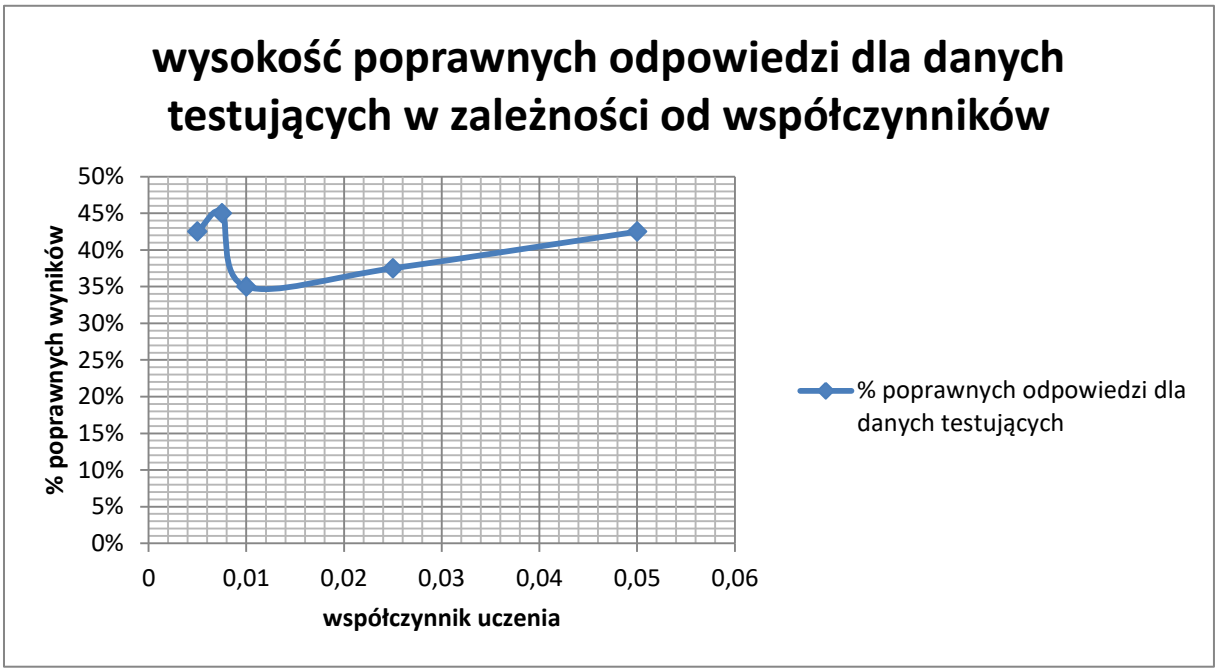
lr	l. epok	%poprawnych
0,005	128	43%
0,0075	101,3	45%
0,01	84	35%
0,025	29,8	38%
0,05	15,7	43%

Na podstawie powyższej tabeli zdecydowanie możemy stwierdzić, że wysokość poprawnych wyników sieci jest niższa niż dla wariantu z współczynnikiem zapominania. W tym przypadku % poprawnych wyników nie przekroczył 50%.

Wykres 3.

Tutaj podobnie jak w wariancie z współczynnikiem zapominania liczba epok potrzebnych do nauczenia sieci ciągle spada.

Wykres 4.



Na powyższym wykresie widzimy, że % poprawnych wyników w wariancie bez współczynnika zapominania dla danych testujących nie przekracza 50%, co wyraźnie sugeruje, że wariant z współczynnikiem zapominania pozwala osiągnąc lepsze wyniki.

5. Wnioski.

Na podstawie otrzymanych wyników pierwszy wniosek jaki się nasuwa to taki, że współczynnik zapominania pozwala osiągnąć wyniki na lepszym poziomie. Dla moich danych testujących sieć potrafiła osiągnąć 55% poprawnych wyników gdzie dla wariantu bez współczynnika zapominania było to 45%. Ważne jest jednak aby współczynnik zapominania nie był zbyt duży gdyż możemy doprowadzić do sytuacji gdzie sieć będzie zbyt szybko zapominać tego czego się nauczyła.

Wraz ze wzrostem współczynnika uczenia liczba epok potrzebnych do nauczenia sieci malała co wynika z faktu, że sieć szybciej dochodzi do rozwiązania dla danych uczących. Jednak testując ją podobnymi emotikonami myliła się ona co raz bardziej dla większych współczynników uczenia.

Błędy przy testowaniu sieci mogły wynikać z faktu, że emotikony testujące były różne od uczących w dosyć dużym stopniu, co mogło wpływać na to, że były one klasyfikowane do innych emotek niż powinny.

W uzyskaniu zadowalających wyników, ważne było znalezienie optymalnej wartości współczynnika uczenia i zapominania, które dla mojej sieci wynosiły odpowiednio 0,01 i 0,0033.

6. Listing kodu.

```
2
3   public class Main {
4
5       static int numberOfInputs = 64 + 1;                                  //ilość wejść
6       static double learnRate = 0.05;                                      //współczynnik uczenia się
7       static double forgetRate = 0.000033;                                 //współczynnik zapominania
8       static int numberOfEmotes = 4;                                       //liczba emotikonów
9       static int numberOfNeurons = 5;                                      //liczba nauronów
10
11      public static void main(String[] args) {
12
13          int winner;
14          Hebb[] hebbs = new Hebb[numberOfNeurons]; // tworzymy siec o okreslonym rozmiarze
15          for (int i = 0; i < numberOfNeurons; i++) {
16              hebbs[i] = new Hebb(numberOfInputs);
17          }
18
19          System.out.println("PRZED UCZENIEM"); // emotikony przypisane do neuronow przed uczeniem
20          for (int i = 0; i < numberOfEmotes; i++) {
21              winner = testHebb(hebbs, Emotes.learnData[i]);
22              System.out.println("Winner Hebb = " + winner);
23          }
24
25          int ages = learn(hebbs);
26
27          System.out.println("\n\nPO UCZENIU"); //emotikony przypsane do neuronow po uczeniu
28          for (int i = 0; i < numberOfEmotes; i++) {
29              winner = testHebb(hebbs, Emotes.learnData[i]);
30              System.out.println("Winner Hebb = " + winner);
31          }
32
33          System.out.println("\n\nIlość epok = " + ages);
34
35          System.out.println("\n\nTESTOWANIE"); //wypisanie wynikow testowania
36
37              for (int i = 0; i < numberOfEmotes; i++) {
38                  winner = testHebb(hebbs, Emotes.testingData[i]);
39                  System.out.println("Winner Hebb = " + winner);
40              }
41
42      }
43
44      //uczenie neuronów
45      public static int learn(Hebb[] hebbs) {
46
47          int counter = 0; //licznik iteracji
48          int limit = 10000; // limit iteracji
49          int[] winners = new int[numberOfNeurons];
50          for (int i = 0; i < numberOfNeurons; i++) {
51              winners[i] = - 1;
52          }
53
54          while (!isUnique(winners)) { // uczymy dopoki jedna emotka nie bedzie miala przypisanego jednego neuronu
55
56              for (int i = 0; i < numberOfNeurons; i++) {
57
58                  //uczenie neuronów każdej emotikony
59                  for (int j = 0; j < numberOfEmotes; j++) {
60                      hebbs[i].learn(Emotes.learnData[j], learnRate, forgetRate, Hebb.HEBB_WITH_FORGETTIN);
61                  }
62
63                  //tesotowanie sieci celem sprawdzenia, czy sieć jest już nauczona
64                  for (int k = 0; k < numberOfEmotes; k++) {
65                      winners[k] = testHebb(hebbs, Emotes.learnData[k]);
66                  }
67              }
68
69              if (++counter == limit) { // jesli licznik osiagnie limit - koniec
70                  break;
71              }
72          }
73
74          return counter; //zwracamy w ktorej epoce siec zostala wyuczona
75      }
76
77      //funkcja pomocnicza w procesie uczenie
78      //zwraca true jeśli każdy element w tablicy jest unikalny
79      public static boolean isUnique(int[] winners) {
80
81          for (int i = 0; i < numberOfNeurons; i++) {
82              for (int j = 0; j < numberOfNeurons; j++) {
83                  if (i != j) {
84                      if (winners[i] == winners[j]) {
85                          return false;
86                      }
87                  }
88              }
89          }
90
91          return true;
```

```java
        //zwraca wartość zwycięzkiego neuronu dla podanej emotikony
        public static int testHebb(Hebb[] hebbs, double[] emoji) {

            double max = hebbs[0].test(emoji);
            int winner = 0;

            for (int i = 1; i < numberOfNeurons; i++) {
                if (hebbs[i].test(emoji) > max) {
                    max = hebbs[i].test(emoji);
                    winner = i;
                }
            }

            return winner;
        }

    }

```

```java
public class Emotes {

        public static double[][] learnData = {
                    //pierwszy bias
                    { 1,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,
                      1,1,0,0,0,0,1,1,1,0,1,1,1,1,0,1,1,0,0,0,0,0,0,1,0,1,1,1,1,1,1,0}, // smiech

                    {1,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,0,0,0,0,
                      0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,1,1,1,1,1,1,1,1,0,0,0,0,0,0,1},    // smutek

                    { 1,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,1,0,0,1,1,
                      1,0,0,0,0,1,1,0,1,1,1,1,1,1,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0},    // usmiech

                    { 1,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,
                      1,0,0,0,0,1,1,1,1,0,0,0,0,1,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0},    // zawstydzony
        };

        public static double[][] testingData = {
                    //pierwszy bias
                    { 1,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,
                      1,0,0,0,0,1,0,0,1,1,1,1,1,1,0,0,1,0,0,0,0,1,0,0,1,1,1,1,1,1,0}, // smiech

                    { 1,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,
                      0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,1,0,0,0,0,0,0,1,1,0,0,0,0,0,0,1},    // smutek

                    { 1, 0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,
                      1,0,0,0,0,1,1,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},    // usmiech

                    { 1,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,1,
                      0,0,0,0,1,0,0,1,0,0,0,0,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0},    // zawstydzony
        };

}

```

```java
public class Hebb {

    private int noi;    //ilość wejść
    private double[] w; //wagi
    public static boolean HEBB_WITH_FORGETTIN = true; //flagi okreslajace jaki typ modyfikacji wag wybieramy
    public static boolean HEBB_WITHOUT_FORGETTIN = false;

    public Hebb ( int numbers_of_inputs ) {
        noi = numbers_of_inputs;
        w = new double[noi];

        for ( int i = 0; i < noi; i++ )
            w[i] = new Random().nextDouble(); //wagi początkowe sa losowane

        normalizeWeights();
    }

    //funkcja aktywacji
    private double activate ( double y_p ) {
        //return y_p; //funkcja liniowa
        return ( 1.0 / ( 1 + Math.pow( Math.E, - y_p ) ) );             //unipolarna sigmoidalna
    }

    //zwraca sumę iloczynów wag i sygnałów wejściowych
    private double sumator ( double[] x ) {
        double y_p = 0.0;
        for ( int i = 0; i < noi; i++ )
            y_p += x[i] * w[i];

        return y_p;
    }

    //uczenie
    public double learn ( double[] x, double lr, double fr, boolean version ) {
        double y = activate( sumator( x ) );

        //w zależności od podanej wersji, nauka będzie z lub bez współczynnika zapominania
        for ( int i = 0; i < noi; i++ )
            if ( version ) w[i] = ( 1 - fr ) * w[i] + lr * x[i] * y;        //ze współczynnikiem zapominania
            else w[i] += lr * x[i] * y;                                     //bez współczynnika zapominania

        normalizeWeights(); // normalizujemy wagi

        return activate( sumator( x ) );
    }

    //zwraca output neuronu
    public double test ( double[] x ) {
        return activate( sumator( x ) );
    }

    //normalizuje wagi
    private void normalizeWeights () {
        double dl = 0.0;
        for ( int i = 0; i < w.length; i++ )
            dl += Math.pow( w[i], 2 );

        dl = Math.sqrt( dl );

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

}
```

7. Bibliografia

Stanisław Osowski – Sieci neuronowe do przetwarzania informacji, ISBN 83-7207-615-4

http://ecee.colorado.edu/~ecen4831/Demuth/Ch7_pres.pdf

http://pracownik.kul.pl/files/31717/public/Model_neuronu_Hebba.pdf