**Kamil Wasilonek**

Podstawy Sztucznej Inteligencji

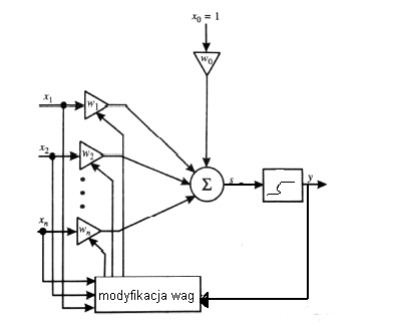
Sprawozdanie z projektu nr 4

**1.​ ​Cel​ ​ćwiczenia**

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

**2. Syntetyczny opis budowy wykorzystanego algorytmu:**

W ćwiczeniu wykorzystałem jednakowe neurony oparte na modelu sigmoidalnym oraz regułę Hebba w wersji bez nauczyciela.



Rys 2. Ogólny schemat działania sieci Hebba

Ogólny model neuronu Hebba, przedstawiony powyżej odpowiada standardowej postaci modelu neuronu. Waga wij włączona jest między sygnałem wejściowym yj a węzłem sumacyjnym i-tego neuronu o sygnale wyjściowym yi. W przypadku pojedynczego neuronu w trakcie uczenia będziemy modyfikować wartość wag proporcjonalnie zarówno do wartości sygnału podanego na i-te wejście, jak i sygnału wyjściowego y z uwzględnieniem współczynnika uczenia. Ważne , że w przypadku tym nie podajemy wzorcowej wartości wyjściowej. Uczenie neuronu z zastosowaniem reguły Hebba może odbywać się w trybie bez nauczyciela i z nauczycielem. W zastosowanym przez nas trybie bez nauczyciela używa się aktualnej wartości yi sygnału wyjściowego. Poniżej przedstawiam dwie metody modyfikacji wag:



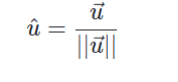
Ze współczynnikiem zapominania:



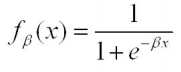
Bez współczynnika zapominania:



Normalizacja polega na podzieleniu każdej ze składowej wektora przez długość tego wektora:

****

Metoda aktywacji, wykorzystuje funkcje unipolarną sigmoidalną:

****

**3. Zestawienie i analiza otrzymanych wyników:**

Ćwiczenie rozpocząłem od wygenerowania danych uczących oraz danych do testowania w postaci macierzy 8x8.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |  | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |  |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 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 | 0 | 0 | 0 | 0 | 0 | 1 |  |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |  | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |  |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |  | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |  |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |  | 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 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |  | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |  |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 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 | 0 | 0 | 0 | 0 | 0 | 1 |  |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |  | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |  |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |  | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |  |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |  | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |  |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |  | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |  |

Rys 3. Graficzne przedstawienie wygenerowanych danych uczących:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |  | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 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 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |  | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |  | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |  | 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 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |  | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 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 | 1 | 0 | 0 | 1 | 0 | 1 |  | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |  | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |  | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |  | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |  | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |

Rys 4. Graficzne przedstawienie wygenerowanych danych testujących:

Dane do testów wykonane zostały na podstawie danych uczących , jednak dodane zostało zaszumienie dla maksymalnie dwóch pikseli aby odpowiednio dobrać rozmiar zaburzeń do wielkości macierzy.

Działanie programu polega na uczeniu sieci a następnie testowaniu otrzymanych wyników. Licznik epok uczenia został ustawiony na 1000. Poniżej przedstawiam po jednym wydruku z programu dla każdej opcji.

**Dla współczynnik ucznia = 0.1 oraz współczynnika zapominania = 0.1/3.0:**

Zwycięzca po uczeniu = 4

Zwycięzca po uczeniu = 1

Zwycięzca po uczeniu = 0

Zwycięzca po uczeniu = 3

Ilość epok = 3

Zwycięzca po testowaniu = 3

Zwycięzca po testowaniu = 1

Zwycięzca po testowaniu = 1

Zwycięzca po testowaniu = 4

Ilość błędu: 3

Wykres 1. Ilość wystąpień danego błędu dla współczynnik uczenia = 0.1 oraz współczynnika zapominania = 0.1/3.0

W przypadku tej opcji udało się uzyskać wynik w którym nauczona siec po przeprowadzeniu testu dawała całkowicie poprawny wynik.

**Dla współczynnik ucznia = 0.01 oraz współczynnika zapominania = 0.01/3.0:**

Zwycięzca po uczeniu = 0

Zwycięzca po uczeniu = 3

Zwycięzca po uczeniu = 4

Zwycięzca po uczeniu = 2

Ilość epok = 132

Zwycięzca po testowaniu = 0

Zwycięzca po testowaniu = 0

Zwycięzca po testowaniu = 1

Zwycięzca po testowaniu = 3

Ilość błędu: 3

Wykres 2. Ilość wystąpień danego błędu dla współczynnik uczenia = 0.01 oraz współczynnika zapominania = 0.01/3.0

W tym przypadku wyniki są o wiele gorsze od poprzednich. Po przeprowadzaniu testów nie udało się osiągnąć całkowicie poprawnego wyniku, dodatkowo ani raz nie otrzymano również jednego błędu. Można zaobserwować znaczny spadek poprawność z powodu zmniejszenia współczynnika uczenia.

**Dla współczynnik ucznia = 0.01 oraz brak współczynnika zapominania:**

Zwycięzca po uczeniu = 4

Zwycięzca po uczeniu = 2

Zwycięzca po uczeniu = 1

Zwycięzca po uczeniu = 0

Ilość epok = 12

Zwycięzca po testowaniu = 2

Zwycięzca po testowaniu = 2

Zwycięzca po testowaniu = 1

Zwycięzca po testowaniu = 0

Ilość błędu: 2

Wykres 3. Ilość wystąpień danego błędu dla współczynnik uczenia = 0.01 oraz brak współczynnika zapominania

**Dla współczynnik ucznia = 0.1 oraz brak współczynnika zapominania:**

Zwycięzca po uczeniu = 2

Zwycięzca po uczeniu = 4

Zwycięzca po uczeniu = 0

Zwycięzca po uczeniu = 3

Ilość epok = 3

Zwycięzca po testowaniu = 1

Zwycięzca po testowaniu = 4

Zwycięzca po testowaniu = 1

Zwycięzca po testowaniu = 3

Ilość błędu: 3

Wykres 4. Ilość wystąpień danego błędu dla współczynnik uczenia = 0.1 oraz brak współczynnika zapominania

Tabela 1. Zestawienie liczby epok potrzebnych do nauczenia

|  |  |  |  |
| --- | --- | --- | --- |
| **Współczynnik uczenia = 0,1** | **Współczynnik uczenia = 0,01** | **Współczynnik uczenia = 0,1** | **Współczynnik uczenia = 0,01** |
| **Współczynnik zapominania = 0.1/3.0** | **Współczynnik zapominania = 0.01/3.0** | **Brak współczynnika zapominania** | **Brak współczynnika zapominania** |
| Liczba epok | | | |
| 42 | 56 | 77 | 110 |
| 76 | 11 | 28 | 125 |
| 70 | 16 | 36 | 20 |
| 23 | 91 | 17 | 119 |
| 15 | 47 | 24 | 8 |
| 88 | 44 | 25 | 30 |
| 68 | 91 | 23 | 17 |
| 17 | 101 | 25 | 116 |
| 1 | 47 | 84 | 96 |
| 14 | 97 | 36 | 61 |
| 50 | 57 | 39 | 132 |
| 31 | 58 | 80 | 80 |
| 57 | 139 | 37 | 146 |
| 36 | 97 | 32 | 130 |
| 0 | 93 | 21 | 52 |
| 65 | 110 | 24 | 121 |
| 50 | 88 | 12 | 107 |
| 10 | 63 | 2 | 93 |
| 13 | 43 | 10 | 120 |
| 42 | 18 | 47 | 139 |
| 9 | 69 | 44 | 24 |
| 23 | 125 | 20 | 44 |
| 73 | 141 | 70 | 34 |
| 37 | 89 | 60 | 83 |
| 65 | 6 | 30 | 17 |
| 43 | 113 | 73 | 105 |
| 85 | 122 | 13 | 76 |
| 6 | 130 | 56 | 148 |
| 26 | 121 | 72 | 85 |
| 32 | 148 | 63 | 22 |
| 39 | 15 | 69 | 38 |

Wykres 5. Średnia ilość epok potrzebna do nauki

Dla każdej konfiguracji przeprowadziłem 30 pomiarów aby obserwować zachowanie sieci.

Na podstawie otrzymanych wyników zauważyłem dużą różnice w ilości epok które były potrzebne do nauczenia sieci. Podczas testów zdarzały się przypadki w których sieć nie była w stanie się nauczyć zaś w następnym kroku sieć była nauczona w kilku lub kilkunastu krokach. Jest to spowodowane

losowości wag początkowych ponieważ może zdarzyć się sytuacja ze wagi będą modyfikowane w taki sposób ze nieustannie będzie wygrywał jeden neuron. Wynikiem takiego działania programu jest problem w których nie osiągniemy stanu w którym każdy neuron odpowiada za konkretną emotkę.

Należy też zwrócić uwagę na duży wpływ współczynnika uczenia na wyniki ponieważ niezależnie od opcji ( z ustawionym bądź nieustawionym współczynnikiem zapominania ) jego zmniejszenie pogarsza otrzymywane wyniki. Wpływało to na większą ilość popełnianych błędów. W moim przypadku różnice z powodu współczynnika zapominania nie są znaczące jednak zauważalne.

**4.Wnioski**

Na podstawie przeprowadzonego uczenia sieci Hebba zaobserwowałem kilka różnic w otrzymanych wynikach dla odpowiednich współczynników uczenia przy zastosowaniu współczynnika zapominania lub jego braku. Opierając się na szczegółowej analizie stwierdzam ze nieznacznie lepsze wyniki dawał opcja z ustawionym współczynnikiem zapominania. Jednak największy wpływ na iość epok niezbędnych do nauki miał współczynnik uczenia.

Proces przeprowadzania testów bardzo często dawał błędne lub niedokładne wyniki. Jest to skutek dużego zaszumienia w stosunku do rozmiaru używanej macierzy. Siec ma trudność z prawidłowym rozpoznaniem emotikon ponieważ na obszarze 8x8 zachodzą zmiany które ciężko odróżnić. W przypadku zwiększenia ilości danych przy odpowiednio małym zaszumieniu wyniki na pewno byłby bardziej zadowalające.

Podczas testów należy również pamiętać o normalizacji wag ponieważ proces ten ma wpływ na działanie sieci. Brak tego mechanizmu znacznie pogarsza a w niektórych przypadkach uniemożliwia uzyskanie poprawnego wyniku.

**Listing Kodu:**

*/\*\*  
 \* Created by Kamil on 2017-12-14.  
 \*/*public class Emoticon {  
  
 // Tablica danych uczących , pierwszy element to bias  
 public static double[][] *emoticonToLearn* = {  
 { 1,0,0,1,1,1,1,0,0,  
 0,1,0,0,0,0,1,0,  
 1,0,1,0,0,1,0,1,  
 1,0,0,0,0,0,0,1,  
 1,0,1,0,0,1,0,1,  
 1,0,0,1,1,0,0,1,  
 0,1,0,0,0,0,1,0,  
 0,0,1,1,1,1,0,0 }, // :)  
  
 { 1,0,0,1,1,1,1,0,0,  
 0,1,0,0,0,0,1,0,  
 1,0,1,0,0,1,0,1,  
 1,0,0,0,0,0,0,1,  
 1,0,0,1,1,0,0,1,  
 1,0,0,0,0,0,0,1,  
 0,1,0,0,0,0,1,0,  
 0,0,1,1,1,1,0,0 }, // :| (małe)  
  
 { 1,0,0,1,1,1,1,0,0,  
 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,  
 1,0,0,0,0,0,0,1,  
 0,1,0,0,0,0,1,0,  
 0,0,1,1,1,1,0,0 }, // :| (duże)  
  
 { 1,0,0,1,1,1,1,0,0,  
 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 }, // :P (język na środku)  
  
  
 };  
  
 // Tablica danych uczących , pierwszy element to bias  
 public static double[][] EmoticonToTest = {  
 { 1,0,0,1,1,1,1,0,0,  
 0,1,0,0,0,0,1,0,  
 1,0,1,0,0,1,0,1,  
 1,0,0,0,0,0,0,1,  
 1,0,0,1,1,0,0,1,  
 1,0,1,0,0,1,0,1,  
 0,1,0,0,0,0,1,0,  
 0,0,1,1,1,1,0,0 }, // :)  
  
  
 { 1,0,0,1,1,1,1,0,0,  
 0,1,0,0,0,0,1,0,  
 1,0,1,0,0,1,0,1,  
 1,0,1,0,0,1,0,1,  
 1,0,0,1,1,0,0,1,  
 1,0,0,0,0,0,0,1,  
 0,1,0,0,0,0,1,0,  
 0,0,1,1,1,1,0,0 }, // :| (Małe)  
  
  
 { 1,0,0,1,1,1,1,0,0,  
 0,1,0,0,0,0,1,0,  
 1,0,1,0,0,1,0,1,  
 1,0,0,0,0,0,0,1,  
 1,0,0,1,1,1,0,1,  
 1,0,1,0,0,0,0,1,  
 0,1,0,0,0,0,1,0,  
 0,0,1,1,1,1,0,0 }, // :| (Duże)  
  
 { 1,0,0,1,1,1,1,0,0,  
 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,  
 1,0,0,0,1,0,0,1,  
 0,1,0,0,0,0,1,0,  
 0,0,1,1,1,1,0,0 }, // :P  
  
  
 };  
  
}

import java.util.Random;  
  
*/\*\*  
 \* Created by Kamil on 2017-12-14.  
 \*/*public class HebbMethod {  
 private int numberOfInputs; //ilość wejść  
 private double[] wages; //wagi  
 public static boolean *HebbWithForget* = true;  
 public static boolean *HebbWithoutForget* = false;  
  
 public HebbMethod(int numbers\_of\_inputs ) {  
 numberOfInputs = numbers\_of\_inputs;  
 wages = new double[numberOfInputs];  
  
 for (int i = 0; i < numberOfInputs; i++ )  
 wages[i] = new Random().nextDouble(); //wagi początkowe sa losowane  
  
 normalizeWeights();  
 }  
  
 //funkcja aktywacji  
 private double activationFunction(double y\_p ) {  
 return ( 1.0 / ( 1 + Math.*pow*( Math.*E*, - y\_p ) ) ); //unipolarna sigmoidalna  
 }  
  
 //zwraca sumę iloczynu wag i sygnałów wejściowych  
 private double sumMethod(double[] x ) {  
 double y\_p = 0.0;  
 for (int i = 0; i < numberOfInputs; i++ )  
 y\_p += x[i] \* wages[i];  
  
 return y\_p;  
 }  
  
 //uczenie  
 public double learnMethod(double[] x, double lr, double fr, boolean isTeacher ) {  
 double y\_p = activationFunction( sumMethod( x ) );  
  
 //wages zależności od podanej wersji, nauka będzie z lub bez współczynnika zapominania  
 for (int i = 0; i < numberOfInputs; i++ )  
 if ( isTeacher ) wages[i] = ( 1 - fr ) \* wages[i] + lr \* x[i] \* y\_p; //ze współczynnikiem zapominania  
 else wages[i] += lr \* x[i] \* y\_p; //bez współczynnika zapominania  
  
 normalizeWeights();  
  
 return activationFunction( sumMethod( x ) );  
 }  
  
 //zwraca output neuronu  
 public double test ( double[] x ) {  
 return activationFunction( sumMethod( x ) );  
 }  
  
 //normalizuje wagi  
 private void normalizeWeights () {  
 double dl = 0.0;  
 for (int i = 0; i < wages.length; i++ )  
 dl += Math.*pow*( wages[i], 2 );  
  
 dl = Math.*sqrt*( dl );  
  
 for (int i = 0; i < wages.length; i++ )  
 if ( wages[i] > 0 && dl != 0 )  
 wages[i] = wages[i] / dl;  
 }  
}

*/\*\*  
 \* Created by Kamil on 2017-12-14.  
 \*/*public class Main {  
 static int *numberOfInputs* = 64 + 1; //ilość wejść  
 static double *learningRate* = 0.1; //współczynnik uczenia się  
 static double *forgettingRate* = *learningRate* / 3.0; //współczynnik zapominania  
 static int *numberOfEmoticon* = 4; //liczba emotikonów  
 static int *numberOfNeurons* = 5; //liczba nauronów  
  
 public static void main ( String[] args ) {  
  
 int winner;  
 HebbMethod[] hebbMethods = new HebbMethod[*numberOfNeurons*];  
 for ( int i = 0; i < *numberOfNeurons*; i++ )  
 hebbMethods[i] = new HebbMethod( *numberOfInputs* );  
 System.*out*.println("\n");  
//  
 for (int i = 0; i < *numberOfEmoticon*; i++ ) {  
 winner = *testHebb*(hebbMethods, Emoticon.*emoticonToLearn*[i] );  
 System.*out*.println( "Zwycięsca przed uczeniem = " + winner );  
 }  
  
 int ages = *learn*(hebbMethods);  
 System.*out*.println("\n");  
//  
 for (int i = 0; i < *numberOfEmoticon*; i++ ) {  
 winner = *testHebb*(hebbMethods, Emoticon.*emoticonToLearn*[i] );  
 System.*out*.println( "Zwycięsca po uczeniu = " + winner );  
 }  
  
 System.*out*.println( "\n\nIlość epok = " + ages );  
 System.*out*.println("\n");  
//  
 for (int i = 0; i < *numberOfEmoticon*; i++ ) {  
 winner = *testHebb*(hebbMethods, Emoticon.*EmoticonToTest*[i] );  
 System.*out*.println( "Zwycięsca po testowaniu = " + winner );  
 }  
  
 }  
  
  
 //uczenie neuronów  
 public static int learn ( HebbMethod[] hebbMethods) {  
  
 int counter = 0;  
 int limit = 1000;  
 int[] winners = new int[*numberOfNeurons*];  
 for ( int i = 0; i < *numberOfNeurons*; i++ )  
 winners[i] = - 1;  
  
 while ( ! *isUnique*( winners ) ) {  
  
 for ( int j = 0; j < *numberOfNeurons*; j++ ) {  
  
 //uczenie neuronów każdej emotikony  
 for (int k = 0; k < *numberOfEmoticon*; k++ )  
 hebbMethods[j].learnMethod( Emoticon.*emoticonToLearn*[k], *learningRate*, *forgettingRate*, HebbMethod.*HebbWithoutForget*);  
  
 //tesotowanie sieci  
 for (int l = 0; l < *numberOfEmoticon*; l++ )  
 winners[l] = *testHebb*(hebbMethods, Emoticon.*emoticonToLearn*[l] );  
 }  
  
 if ( ++ counter == limit )  
 break;  
 }  
  
 return counter;  
 }  
  
 //funkcja pomocnicza w procesie uczenie  
 //zwraca true jeśli każdy element w tablicy jest unikalny  
 public static boolean isUnique ( int[] winners ) {  
  
 for ( int i = 0; i < *numberOfNeurons*; i++ )  
 for ( int j = 0; j < *numberOfNeurons*; j++ )  
 if ( i != j )  
 if ( winners[i] == winners[j] )  
 return false;  
  
 return true;  
 }  
  
 //zwraca wartość zwycięzkiego neuronu dla podanej emotikony  
 public static int testHebb (HebbMethod[] hebbMethods, double[] emoticon ) {  
  
 double max = hebbMethods[0].test( emoticon );  
 int winner = 0;  
  
 for ( int i = 1; i < *numberOfNeurons*; i++ ) {  
 if ( hebbMethods[i].test( emoticon ) > max ) {  
 max = hebbMethods[i].test( emoticon );  
 winner = i;  
 }  
 }  
  
 return winner;  
 }  
}