Konrad Tabiś

Laboratorium nr 4

Sprawozdanie - Podstawy Sztucznej Inteligencji – Scenariusz 4

Temat ćwiczenia: Uczenie sieci regułą Hebba.

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

**1) Syntetyczny opis budowy użytej sieci i algorytmów uczenia.**

W ćwiczeniu wykorzystałem neurony o identycznej strukturze jak w przypadku modelu sigmoidalnego, jednak charakteryzujące się inną metodą uczenia, znaną pod nazwą *reguły Hebba*. Metoda ta występuje w dwóch wersjach: z nauczycielem oraz bez nauczyciela. Do wykonania ćwiczenia wykorzystałem wersję bez nauczyciela. Występują również dwa rodzaje sposobu modyfikacji wag:

gdzie:  
η – współczynnik uczenia  
yj – sygnał wejściowy  
yi – sygnał wyjściowy

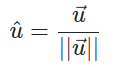
Ze współczynnikiem zapominania:

gdzie:  
γ – współczynnik zapominania

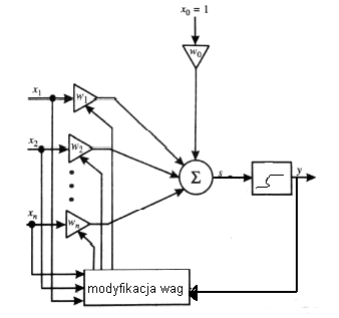
Bez współczynnika zapominania:



Dodatkowo podczas uczenia normalizuję wagi, aby zapobiec nieustającemu ich wzrostowi. Normalizacja polega na podzieleniu każdej ze składowej wektora przez długość tego wektora:

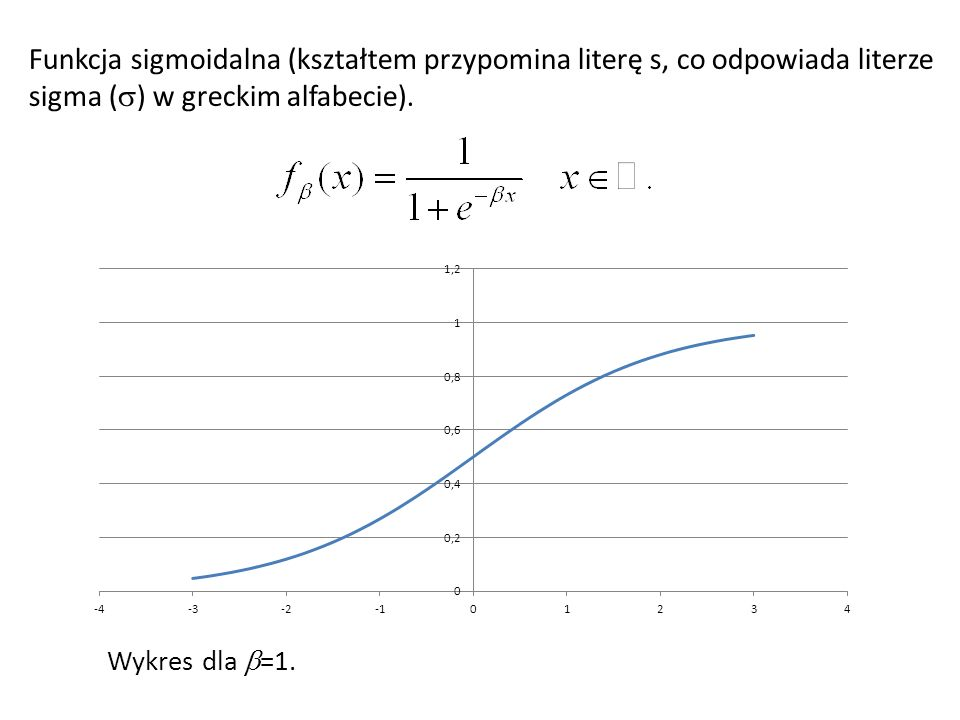


Tak więc w trakcie uczenia modyfikacja wagi zależna jest zarówno od sygnału podanego na wejście jak i sygnału wyjściowego. Poniżej schemat pojedynczego neuronu:

  
Schemat neuronu Hebba

Zaimplementowany przeze mnie klasa Hebb składa się z następujących metod:

***Active,*** wykorzystuje unipolarną sigmoidalną funkcję aktywacji:



***Sumator,*** zwraca sumę iloczynów wag oraz sygnałów wejściowych:



***LearnUnsupervised,*** uczenie poprzez modyfikację wag neuronu, zarówno w wersji ze współczynnikiem zapominania, jak i bez niego. Korzysta ona ze wzorów podanych już wcześniej.

***Test,*** zwraca sygnał wyjściowy neuronu.

***NormalizeWeights,*** normalizuje wagi neuronu.

**2) Zestawienie otrzymanych wyników**

Do uczenia wykorzystałem stworzone przeze mnie emotikony o wymiarach 8x8 pikseli.





Jako dane testujące wykorzystałem te same emotikony, jednak lekko zmodyfikowane, poprzez dodanie do nich szumu w postaci piksela w losowym miejscu:





Każdy piksel stanowi jedno wejście, tak więc każdy neuron otrzymuje po 64 sygnałów wejściowych.

Wyniki dla wersji modyfikacji wag ze współczynnikiem zapominania:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Learning rate | 0,1 | 0,1 | 0,01 | 0,01 | 0,001 | 0,001 |
|  | Forgetting rate | 0,03333 | 0,01667 | 0,00333 | 0,00167 | 0,00033 | 0,00017 |
| 1 | Poprawność testowania [%] | 50,00% | 50,00% | 100,00% | 50,00% | 50,00% | 50,00% |
| Ilość epok | 1 | 2 | 143 | 10 | 58 | 86 |
| 2 | Poprawność testowania [%] | 100,00% | 25,00% | 50,00% | 100,00% | 50,00% | 75,00% |
| Ilość epok | 16 | 1 | 17 | 15 | 174 | 119 |
| 3 | Poprawność testowania [%] | 50,00% | 25,00% | 100,00% | 50,00% | 50,00% | 25,00% |
| Ilość epok | 1 | 17 | 11 | 27 | 67 | 116 |
| 4 | Poprawność testowania [%] | 25,00% | 75,00% | 50,00% | 50,00% | 50,00% | 25,00% |
| Ilość epok | 4 | 3 | 143 | 19 | 137 | 27 |
| 5 | Poprawność testowania [%] | 25,00% | 50,00% | 100,00% | 100,00% | 25,00% | 75,00% |
| Ilość epok | 3 | 1 | 8 | 10 | 208 | 171 |
| 6 | Poprawność testowania [%] | 75,00% | 75,00% | 75,00% | 50,00% | 50,00% | 50,00% |
| Ilość epok | 2 | 1 | 148 | 30 | 68 | 99 |
| 7 | Poprawność testowania [%] | 50,00% | 75,00% | 50,00% | 100,00% | 50,00% | 100,00% |
| Ilość epok | 3 | 1 | 145 | 5 | 147 | 115 |
| 8 | Poprawność testowania [%] | 25,00% | 25,00% | 50,00% | 75,00% | 75,00% | 50,00% |
| Ilość epok | 3 | 2 | 141 | 17 | 71 | 141 |
| 9 | Poprawność testowania [%] | 75,00% | 75,00% | 50,00% | 100,00% | 25,00% | 75,00% |
| Ilość epok | 2 | 1 | 11 | 16 | 159 | 126 |
| 10 | Poprawność testowania [%] | 75,00% | 100,00% | 50,00% | 75,00% | 75,00% | 75,00% |
| Ilość epok | 13 | 4 | 15 | 12 | 69 | 58 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Learning rate | 0,1 | 0,1 | 0,01 | 0,01 | 0,001 | 0,001 |
| Forgetting rate | 0,03333 | 0,01667 | 0,00333 | 0,00167 | 0,00033 | 0,00017 |
| Max epok | 16 | 17 | 148 | 30 | 208 | 171 |
| Min epok | 1 | 1 | 8 | 5 | 58 | 27 |
| Max poprawność [%] | 100,00% | 100,00% | 100,00% | 100,00% | 75,00% | 100,00% |
| Min poprawność [%] | 25,00% | 25,00% | 50,00% | 50,00% | 25,00% | 25,00% |
| średnia ilość epok | 4,8 | 3,3 | 78,2 | 16,1 | 115,8 | 105,8 |
| średnia poprawność | 55,00% | 57,50% | 67,50% | 75,00% | 50,00% | 60,00% |

Wyniki dla wersji modyfikacji wag bez współczynnika zapominania:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Learning rate | 0,1 | 0,01 | 0,001 |
| 1 | Poprawność testowania [%] | 50,00% | 75,00% | 50,00% |
| Ilość epok | 8 | 7 | 75 |
| 2 | Poprawność testowania [%] | 75,00% | 75,00% | 50,00% |
| Ilość epok | 2 | 15 | 97 |
| 3 | Poprawność testowania [%] | 50,00% | 50,00% | 100,00% |
| Ilość epok | 17 | 18 | 98 |
| 4 | Poprawność testowania [%] | 50,00% | 100,00% | 50,00% |
| Ilość epok | 1 | 8 | 94 |
| 5 | Poprawność testowania [%] | 50,00% | 75,00% | 50,00% |
| Ilość epok | 2 | 1 | 84 |
| 6 | Poprawność testowania [%] | 50,00% | 50,00% | 75,00% |
| Ilość epok | 4 | 11 | 167 |
| 7 | Poprawność testowania [%] | 50,00% | 75,00% | 75,00% |
| Ilość epok | 3 | 15 | 2 |
| 8 | Poprawność testowania [%] | 25,00% | 50,00% | 50,00% |
| Ilość epok | 1 | 15 | 80 |
| 9 | Poprawność testowania [%] | 50,00% | 100,00% | 100,00% |
| Ilość epok | 3 | 147 | 65 |
| 10 | Poprawność testowania [%] | 75,00% | 50,00% | 25,00% |
| Ilość epok | 2 | 8 | 215 |

|  |  |  |  |
| --- | --- | --- | --- |
| Learning rate | 0,1 | 0,01 | 0,001 |
| Max epok | 17 | 147 | 215 |
| Min epok | 1 | 1 | 2 |
| Max poprawność [%] | 75,00% | 100,00% | 100,00% |
| Min poprawność [%] | 10,00% | 1,00% | 0,10% |
| średnia ilość epok | 4,3 | 24,5 | 97,7 |
| średnia poprawność | 52,50% | 70,00% | 62,50% |

**3) Analiza i dyskusja błędów uczenia i testowania oraz wyników rozpoznawania dla opracowanej sieci w zależności od wartości współczynnika uczenia i zapominania**

Jak widać na powyższych wynikach, ilość epok jaka była potrzebna do nauczenia sieci znacząco różniła się w poszczególnych przypadkach. Niekiedy wynik wynosił tylko 1 epokę, a niekiedy było to ponad 200 epok. Tak więc jeśli chodzi o szybkość uczenia, widać, że nie jest to dobra miara ocenienia jakości uczenia sieci, gdyż jest to spowodowane wyłącznie początkowymi wartościami wag neuronów, a te są losowe.

Jeśli jednak spojrzeć na wykresy, to widać, że zarówno jeśli chodzi o wartości maksymalne, minimalne czy też średnie, wraz ze wzrostem współczynnika uczenia, wartość ilości epok potrzebnych do nauczenia sieci stale spada.

Natomiast w przypadku doboru współczynników uczenia oraz współczynników zapominania widać dosyć wyraźne zależności. Dla wersji ze współczynnikiem zapominania najlepiej spojrzeć na wykresy średniej procentowej poprawności uczenia się w zależności od współczynnika uczenia oraz współczynnika zapominania. Widać na nim, że zwycięzcą zarówno w wersji uczenia ze współczynnikiem zapominania jak i bez niego, jest współczynnik uczenia równy 0,1. Dodatkowo zwycięzcą spośród wszystkich trzech wersji jest kombinacja: współczynnik uczenia równy 0,1 oraz współczynnik zapominania równy 1/6 wartości współczynnika uczenia.

**4) Sformułowanie wniosków**

Na podstawie powyższych wyników można wnioskować, iż najlepsze wyniki można uzyskać stosując metodę modyfikacji wag ze współczynnikiem zapominania. Najlepszy wynik osiągnęła sieć o współczynniku uczenia równym 0,1 oraz o współczynniku zapominania równym 1/6 wartości współczynnika uczenia. Sieć ucząca się bez współczynnika zapominania również osiągała dobre wyniki, jednak nie były one aż tak obiecujące jak w powyższym przypadku.

Jeśli chodzi o sam proces testowania zaszumionych emotikonów, widać, iż podczas testów prawie zawsze pojawiał się jakiś błąd. Spowodowane jest to tym, że emotikony są do siebie bardzo podobne – różnią się tylko praktycznie kilkoma pikselami. Dodatkowo sam rozmiar emotikonów pozostawia wiele do życzenia. Jest to bowiem rozmiar jedynie 8x8 pikseli, tak więc nie pozwala to na zbyt dużą różnorodność emotikonów. Gdyby emotikony miały większą rozdzielczość to podczas uczenia można by zastosować więcej wzorców jednej emotikony, które byłyby do siebie zbliżone. Wtedy sieć byłaby w sanie lepiej sklasyfikować emotikony, a i podczas testowania wyniki byłyby dokładniejsze.

**5) Listing kodu**

public class Main {  
  
 static int *numberOfInputs* = 64 + 1; //ilość wejść (+1 bo bias)  
 static double *learningRate* = 0.01; //współczynnik uczenia się  
 static double *forgettingRate* = *learningRate* / 6.0; //współczynnik zapominania  
 static int *numberOfEmoji* = 4; //liczba emotikonów  
 static int *numberOfNeurons* = 5; //liczba neuronów  
  
 public static void main ( String[] args ) {  
  
 int winner;  
 Hebb[] hebbs = new Hebb[*numberOfNeurons*];  
 for ( int i = 0; i < *numberOfNeurons*; i++ )  
 hebbs[i] = new Hebb( *numberOfInputs* );  
  
 int ages = *learn*( hebbs );  
  
 System.*out*.println( "PO UCZENIU" );  
 for ( int i = 0; i < *numberOfEmoji*; i++ ) {  
 winner = *testHebb*( hebbs, Emoji.*emoji*[i] );  
 System.*out*.println( "Emoji " + Emoji.*emojiType*[i] + " - winner neuron = " + winner );  
 }  
  
 System.*out*.println( "\nTESTOWANIE" );  
 for ( int i = 0; i < *numberOfEmoji*; i++ ) {  
 winner = *testHebb*( hebbs, Emoji.*emojiNoised*[i] );  
 System.*out*.println( "Emoji " + Emoji.*emojiType*[i] + " - winner neuron = " + winner );  
 }  
  
 System.*out*.println( "\nIlość epok = " + ages );  
  
 }  
  
 //uczenie neuronów  
 public static int learn ( Hebb[] hebbs ) {  
  
 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 < *numberOfEmoji*; k++ )  
 hebbs[j].learnUnsupervised( Emoji.*emoji*[k], *learningRate*, *forgettingRate*, Hebb.*HEBB\_WITH\_FORGETTING* );  
  
 //tesotowanie sieci celem sprawdzenia, czy sieć jest już nauczona  
 for ( int l = 0; l < *numberOfEmoji*; l++ )  
 winners[l] = *testHebb*( hebbs, Emoji.*emoji*[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 ( 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;  
 }  
  
}

import java.util.Random;  
  
public class Hebb {  
  
 private int noi; //ilość wejść  
 private double[] w; //wagi  
 public static boolean *HEBB\_WITH\_FORGETTING* = true; //flaga do uczenia ze współczynnikiem zapominania  
 public static boolean *HEBB\_WITHOUT\_FORGETTING* = false; //flaga do uczenia bez współczynnika zapominania  
  
 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 active ( double y\_p ) {  
 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 learnUnsupervised ( double[] x, double lr, double fr, boolean version ) {  
 double y\_p = active( 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\_p; //ze współczynnikiem zapominania  
 else w[i] += lr \* x[i] \* y\_p; //bez współczynnika zapominania  
  
 normalizeWeights();  
  
 return active( sumator( x ) );  
 }  
  
 //zwraca output neuronu  
 public double test ( double[] x ) {  
 return active( 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;  
 }  
  
}

public class Emoji {  
  
 public static double[][] *emoji* = {  
 //pierwszy input to bias  
 { 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, 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,  
 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 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, 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 }, // :D  
  
 };  
  
 public static double[][] *emojiNoised* = {  
 //pierwszy input to bias  
 { 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 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, 1, 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, 1, 0, 1, 0, 1, 1, 0, 0,  
 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 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, 1, 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 }, // :D  
 };  
  
 public static String[] *emojiType* = { ":)", ":(", ":|", ":D" };  
  
}