Red de Hopfield

García Prado, Sergio

23 de diciembre de 2016

Resumen

Implementación de un ejemplo simple de una red de Hopfield diseñada para obtener el punto más cercano de entre dos dados previamente a una estructura en forma de forma de cubo.

I. Introducción

Las redes de Hopfield son un tipo de redes neuronales recurrente inicialmente desarrolladas por John Hopfield. Estas redes son usadas como sistemas de Memoria asociativa con unidades binarias y están diseñadas para converger a un mínimo local. A pesar de ello, la convergencia a alguno de los patrones almacenados no está garantizada.

Las unidades de las redes Hopfield son binarias, es decir, solo tienen dos valores posibles para sus estados (Esto se consigue a partir del uso de una función signo). El valor se determina si las unidades superan o no un determinado umbral. Los valores posibles pueden ser 1 ó -1,

Como caso de prueba se ha realizado lo siguiente: Utilizando como modelo mental un cubo, se pretende introducir dos vértices opuestos del mismo, que serán utilizados como los patrones que se quieren reconocer a partir de pequeñas variaciones de los mismos. Estas pequeñas variaciones son el resto de vértices del cubo. La red de Hopfield obtendrá como resultado en cada caso, el patrón más próximo por cada entrada dada.

II. Implementación

La implementación del algoritmo, así como su prueba se ha realizado en el lenguage Octave (debido a las facilidad y simplicidad que ofrece cuando se codifican cálculos matemáticos). A continuación se describen cada una de las 3 partes en las cuales se ha dividido la implementación: La primera de ellas consiste en un fichero de control de entrada y ejecución (Main), mientras que la segunda y la tercera formar el propio algoritmo, correspondiendose la primera fase del mismo al periodo de aprendizaje de patrones y la segunda a la clasificación de entradas.

```
function main()
  clear all;

  x = [-1, -1, 1;   1, -1, -1;   1, 1, 1;   -1, 1, 1;   -1, -1, -1;   1, 1, -1]
  x_p = [1, -1, 1;   -1, 1, -1]

  w = hopfield_learning(x_p)
  s = hopfield_working(w, x)
end;
```

Figura 1: *Octave: main.m*

Tal y como se muestra en el código de la figura 1, primero inicializamos las matrices encargadas de alojar tanto los vectores que posteriormente serán utilizados como patrones como los que son utilizados como casos de prueba. Posteriormente se inicializa la matriz W encargada de almacenar los patrones tal y como se explicará a continuación. Finalmente se calcula el resultado (vector más cercano entre los patrones por cada entrada).

$$\overline{X}(i) = [x_1(i), x_2(i), ..., x_j(i), ..., x_n(i)], x_j(i) \in \{1, -1\} i \in (1, 2, .., r), \tag{1}$$

$$\overline{XP}(k) = [xp_1(k), xp_2(k), ..., xp_j(k), ..., xp_n(k)], xp_j(j) \in \{1, -1\}, k \in (1, 2, ...p), \tag{2}$$

```
function w = hopfield_learning(x_p)
  w = zeros(size(x_p,2),size(x_p,2));
  for i = 1:size(w,1);
    for j = 1:size(w,2);
      if(i ~= j);
        w(i,j) = sum(x_p(:,i) .* x_p(:,j));
      endif;
    endfor;
  endfor;
end
```

Figura 2: *Octave: hopfield_ learning.m*

El proceso de rellenado de la matrix W en la cual se almacenan los patrones enseñados en la fase de aprencizaje se puede modelizar matemáticamente de la siguiente manera:

$$w_{ij} = \sum_{k=1}^{p} x_i(k) x_j(k), \forall i \neq j; w_{ii} = 0 \tag{3}$$

```
function s = hopfield_working(w,s)
  for index = 1:size(s,1);
    do;
      for i = 1:size(s,2);
        s_old = s(index,:);
        temp = sum(w(i,:) .* s_old);
        if (temp > 0);
          s(index,i) = 1;
        elseif(temp < 0);
          s(index,i) = -1;
        endif;
      endfor;
    until(s_old == s(index,:));
  endfor;
end;
```

Figura 3: *Octave: hopfield_ working.m*

El proceso de reconocimiento de patrones se realiza de manera recurrente aplicando la funcion s definida de la siguiente manera:

$$\begin{cases} s_i(t) = x_i & t = 0 \\ s_i(t+1) = sgn(\sum_{j=1}^{n} w_{ij} x_j(k) - u_i) & t \neq 0 \end{cases} \tag{4}$$

$$sgn(x) = \begin{cases} +1 & x > 0 \\ -1 & x < 0 \end{cases} \tag{5}$$

Este proceso se repite indefinidamente hasta que el resultado se mantenga constante, es decir, $s_i(t+1) = s_i(t)$. Entonces se ha encontrado el patrón más cercano a dicha entrada.

III. Resultados

Los resultados obtenidos para una ejecucción concreta con las entradas x y x_p se muestran en la figura 4.

```
--------------------------------------------------------------------------------
Title: Hopfield Network: Cube Aproximation
Subject: Data Mining
Author: Sergio García Prado (garciparedes.me)
--------------------------------------------------------------------------------
Octave:

x =

  -1  -1   1
   1  -1  -1
   1   1   1
  -1   1   1
  -1  -1  -1
   1   1  -1

x_p =

   1  -1   1
  -1   1  -1

w =

   0  -2   2
  -2   0  -2
   2  -2   0

s =

   1  -1   1
   1  -1   1
   1  -1   1
  -1   1  -1
  -1   1  -1
  -1   1  -1

--------------------------------------------------------------------------------
```

Figura 4: *Resultados obtenidos*