



*Instituto Politécnico Nacional.  
Escuela Superior de Cómputo.*

# **Práctica 2: “Red de Hamming”**

Asignatura:  
Neural Networks.

Profesor: Marco Antonio Moreno Armendáriz.

Grupo: 3CM1.

Fecha de Entrega: 11/03/19

Alumno:

• Morales Flores Víctor Leonel.





# Neural Networks

## Práctica 2: Red de Hamming.

### Introducción:

El cerebro humano es capaz de clasificar una gran cantidad de objetos a la vez sin mayor complejidad, podemos decir fácilmente si un objeto es una fruta o un juguete pero eso no se detiene así. Nosotros somos capaces de hacer esa clasificación de forma automática y sin tardar tanto tiempo en el ejercicio.

Para una computadora realiza esta labor no es algo trivial, la computadora requiere convertir las características de los objetos en valores con los que pueda operar y entre más características se necesiten para clasificar un conjunto de objetos, más será el costo computacional asociado a la tarea.

Las redes neuronales en modo clasificador buscan lograr estas tareas, en el caso de la red de Hamming esto lo hace a través de la distancia de Hamming y operaciones en distintas épocas con una arquitectura de dos capas para lograr su cometido.

A lo largo de la presente práctica se mostrará cuáles son las características de esta arquitectura, cuál es su modelo matemático, funciones de activación y cómo distintos vectores de entrada( $p$ ) y vectores prototipo( $W1$ ) generan una evolución distinta a lo largo de la época en función de valores como es  $\epsilon$  (del cual se hablará más adelante).

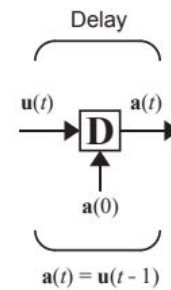
## Marco Teórico:

En ocasiones, la arquitectura de una red neuronal para resolver un problema contempla más de una capa para cumplir con su objetivo. En el caso de la red de Hamming, la arquitectura propuesta contempla dos capas, una capa feed-forward dónde llegan las entradas del exterior y una segunda capa recurrente que recibe como condición inicial la salida de nuestra primera capa.

Se dice que tenemos una capa feed- forward cuando la propagación a lo largo de ella es hacia un solo sentido y no hay una reconexión de la salida hacia la entrada de la capa. En el caso de las capas recurrentes, estas capas se caracterizan porque la salida de la red se conecta a la entrada de la misma. Para el caso de esta red neuronal el modelo contempla una capa recurrente en tiempo discreto.

Una red recurrente de tiempo discreto incorpora un *delay* en su interior que recibe nuestra condición inicial y su función es retrasar el valor de la entrada en  $t-1$ . Este comportamiento está modelado por la siguiente función:

$$a(t) = u(t - 1)$$

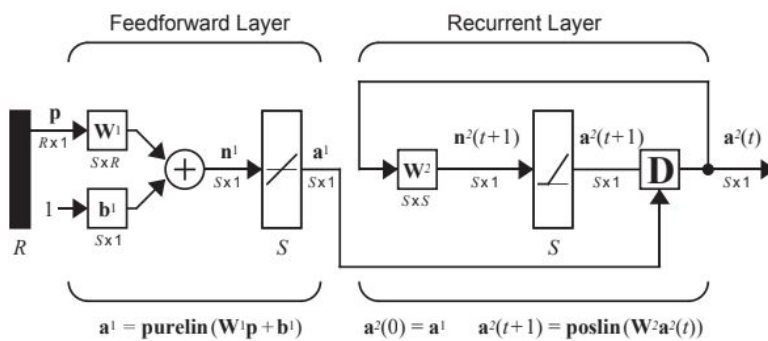


Esta red es una red competitiva, esto quiere decir que en este tipo de redes solo habrá una neurona ganadora. En otro tipo de redes todos los valores trabajan en conjunto para resolver la problemática pero en las redes competitivas esto no es así. Cada neurona tendrá un patrón distinto y el objetivo será resolver el problema de cuál de ellas representa de mejor manera al valor de entrada. La ganadora se llevará todo el crédito y el resto de neuronas serán desactivadas.

Las redes neuronales competitivas suelen ser bicapa y tanto sus valores de entrada como salida son bipolares o binarios. La salida tendrá el mismo formato que los valores a la entrada de nuestra red.

La red de Hamming fue diseñada para resolver problemas de reconocimiento de patrones, donde los valores de nuestro vector de entrada( $p$ ) pueden tener sólo dos valores.

El diagrama de la red de Hamming es el siguiente:



De este diagrama podemos extraer información importante para su implementación. La red hace uso de vectores prototipo que servirán para que pueda determinar a cuál de ellos (vectores prototipo) se aproxima más la entrada ( $p$ ). Estos vectores prototipos se usan para construir nuestra matriz de pesos  $W^1$  (perteneciente a la capa feed-forward) de la siguiente manera:

$$W^1 = \begin{pmatrix} V_{p1}^T \\ V_{p2}^T \\ \vdots \\ V_{pk}^T \end{pmatrix}$$

donde:

$V_{pk}^T$  es el k-ésimo vector prototipo proporcionado a la red

Dada esta ecuación podemos observar rápidamente que el número de neuronas de nuestra primera y segunda capa estará determinada por el número de vectores prototipos que se tengan.

El elemento que nos hace falta para poder hacer las operaciones de la primera capa es el *bias*. Con el fin de que la salida de la primera capa ( $a^1$ ) sea positivo, el bias será construido con el valor del tamaño del vector de entrada ( $R$ ) y su tamaño será de  $S$ .

$$b = \begin{pmatrix} R \\ R \\ \vdots \\ R \end{pmatrix}$$

$S \times 1$

Con estos valores ya somos capaces de obtener el resultado de nuestra primera capa de la red. Esta capa tiene una función de activación **purelin** y la ecuación que representa nuestra primera capa se coloca a continuación:

$$a^1 = \text{purelin}(W^1 p + b) \rightarrow a^1 = W^1 p + b$$

Como se puede apreciar en la ecuación superior, debido al comportamiento de la función de activación **purelin** ( $a=n$ ) podemos simplificar la expresión para dejarla sólo como  **$W^1 p + b$** .

Una vez que se realizaron las operaciones de nuestra capa feed-forward pasamos a las operaciones de nuestra capa recurrente. Para ello necesitamos saber cómo se generan las variables involucradas en este proceso. Es importante notar que nuestra capa recurrente no hace uso de un bias.

Lo primero que tenemos que establecer es nuestra condición inicial. Gracias al diagrama podemos apreciar que nuestra condición inicial(  $\mathbf{a2(0)}$  ) es igual a la salida de nuestra capa feed-forward( $\mathbf{a1}$ ). Para construir nuestra matriz de pesos  $W2$  requerimos calcular antes el valor de nuestro  $\epsilon$  con la inecuación:

$$\epsilon < \frac{1}{S - 1}$$

Una vez que se decide un valor de  $\epsilon$  que cumpla con la desigualdad mencionada arriba tenemos la información suficiente para poder construir nuestra matriz  $W2$ . Esta matriz tendrá un tamaño de  $S \times S$  y en la diagonal principal contendrá 1's. En el resto de la matriz se colocará el valor "menos  $\epsilon$ ":

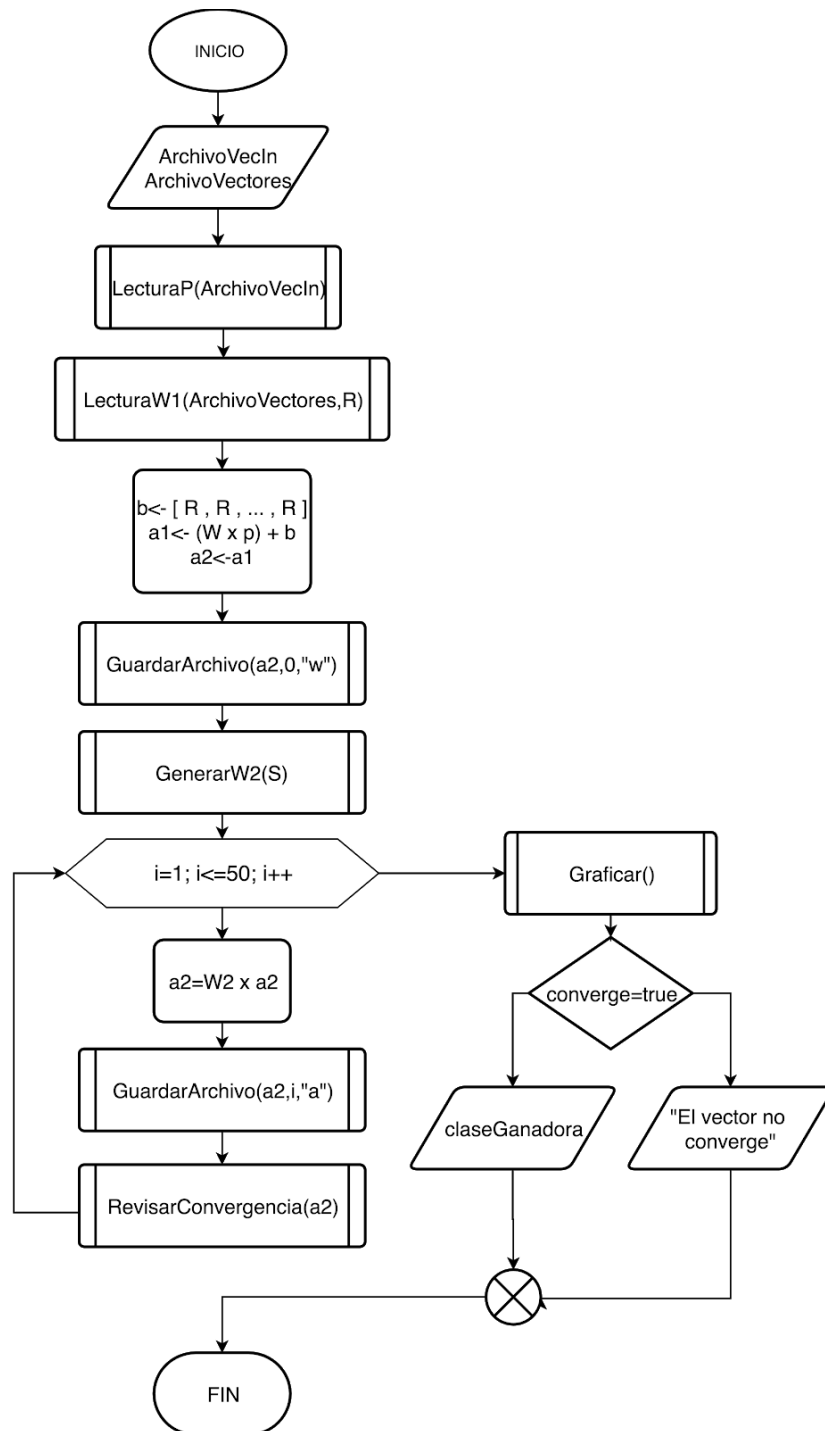
$$W^2 = \begin{pmatrix} 1 & -\epsilon & -\epsilon & \dots & -\epsilon \\ -\epsilon & 1 & -\epsilon & \dots & -\epsilon \\ -\epsilon & -\epsilon & 1 & \dots & -\epsilon \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -\epsilon & -\epsilon & -\epsilon & -\epsilon & 1 \end{pmatrix}_{S \times S} :$$

Ahora estamos listos para comenzar a realizar las iteraciones en esta capa. Nuestra red dejará de iterar cuando todos los valores de salida excepto uno valgan 0. Cuando esto suceda podremos asegurar que la capa convergió. Una vez que la capa converge podremos saber cuál es la clase que ganó a partir del valor dentro de la salida  $\mathbf{a2}$  que es distinto a '0' y con el apoyo de nuestros vectores prototipo. Para ello influye la forma en que nuestros vectores fueron acomodados dentro de la matriz de pesos  $\mathbf{W1}$ . Por ejemplo, si en nuestra matriz de pesos colocamos el vector prototipo que describe a un *lápiz* en la fila '3', entonces el valor de nuestro vector  $\mathbf{a2(3)}$  nos indicará si el vector de entrada  $\mathbf{p}$  corresponde a un lápiz o no. En caso de que en  $\mathbf{a2(3)}$  el valor sea mayor a '0', entonces la red habrá determinado que es un lápiz pero si por el contrario, el valor  $\mathbf{a2(3)}$  es igual a '0' se entenderá que el vector de entrada  $\mathbf{p}$  no corresponde a un *lápiz*.

Esto se debe a que la función de activación de esta capa es la **poslin**. Esta capa se representa a través de la siguiente ecuación:

$$a^2(t+1) = \text{poslin}(W^2 \times a^1(t))$$

## Diagrama de Flujo:



## Experimentos:

Nuestro primer bloque de experimentos se basó en el ejercicio propuesto en el libro que es la clasificación de una naranja y una manzana. Primero describiremos nuestros vectores prototipo de la siguiente forma(archivo **vectores.txt**):

$$Manzana = \begin{bmatrix} +1 \\ +1 \\ -1 \end{bmatrix} \quad Naranja = \begin{bmatrix} +1 \\ -1 \\ -1 \end{bmatrix}$$

El vector que de entrada que ingresaremos a la red será el siguiente que se encuentra en nuestro archivo **p.txt**:

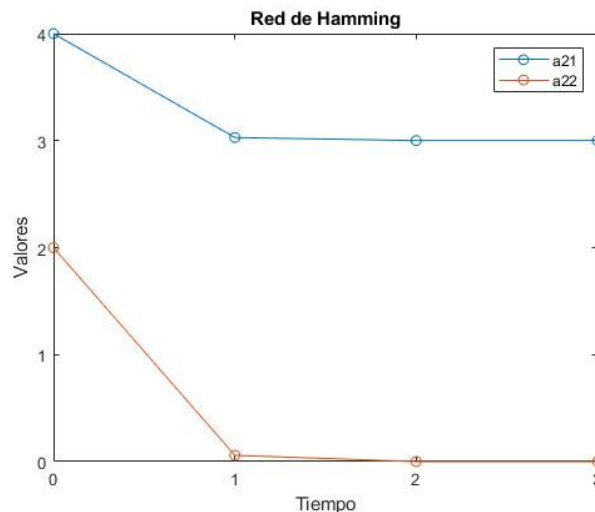
$$p = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}$$

Debido al orden en la que colocamos los vectores prototipo en nuestro archivo, la matriz de pesos quedará como se muestra a continuación:

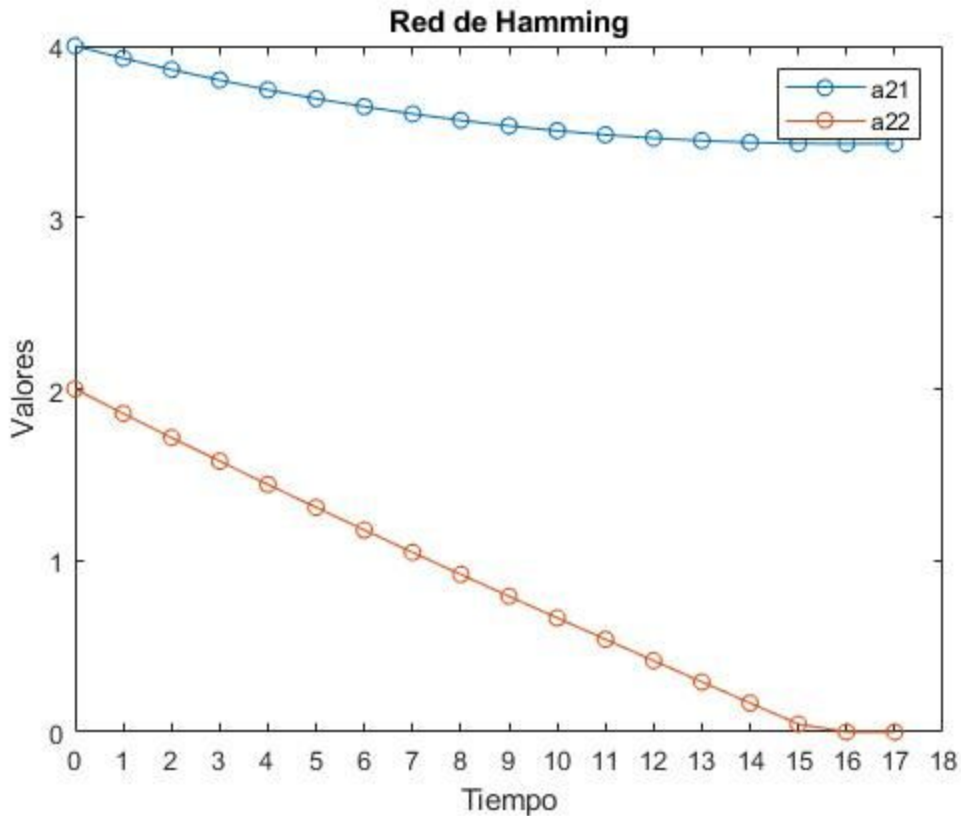
$$W^1 = \begin{pmatrix} +1 & -1 & -1 \\ +1 & +1 & -1 \end{pmatrix} \Rightarrow W^1 = \begin{pmatrix} Naranja(Clase a_{21}) \\ Manzana(Clase a_{22}) \end{pmatrix}$$

Con estos datos ya podemos interpretar la salida de nuestra red así que procedemos a ejecutar el programa y ver cuál es el valor obtenido por la red tras la clasificación de nuestro vector p.

Recordando que el valor de épsilon es aleatorio podremos observar cosas importantes en estas pruebas pero para ello debemos tener en cuenta los valores de épsilon utilizados para generar cada una de las gráficas. En el caso de la primer gráfica el valor de épsilon fue de 4.853756e-01 y la clase ganadora fue la a21(Naranja).

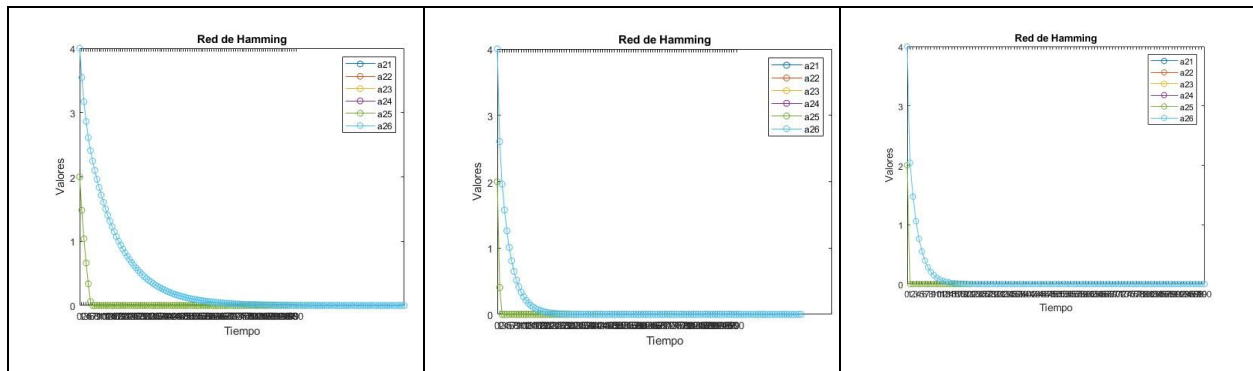


Con los mismos vectores ejecutamos el programa otra vez para apreciar cómo influye  $\epsilon$  en el número de épocas que la red requiere para converger. Para el caso de la siguiente gráfica el valor de  $\epsilon$  utilizado por el programa fue de  $571168e-02$  y el resultado fue el siguiente:



La última prueba con este vector  $p$  es aumentando el número de vectores prototipo a un total de 6-vectores(archivo **vectores2.txt**) prototipo. Con esto la distancia de Hamming entre nuestro vector  $p$  y todos nuestros vectores prototipo estará más cerca. Aquí veremos lo importante del valor  $\epsilon$  debido a que la red a veces convergerá a una clase (a21) y en otras ocasiones las 150 épocas(máximo definido en código) no serán suficientes para que logre converger la red. Para el caso de  $\epsilon$  con valor de  $3.252235e-02$  la red no logra converger(gráfica de la izquierda) pero con valor de  $\epsilon$   $9.967281e-02$  la red converge a la clase a21(gráfica del centro) y con un  $\epsilon$  igual a  $1.398153e-01$  la red converge a la clase a26(gráfica de la izquierda) tras 150 épocas.



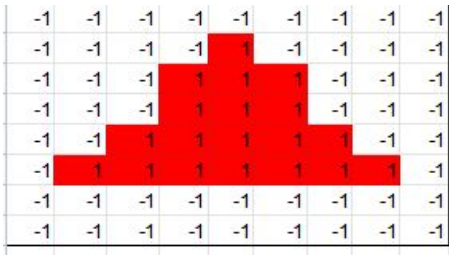
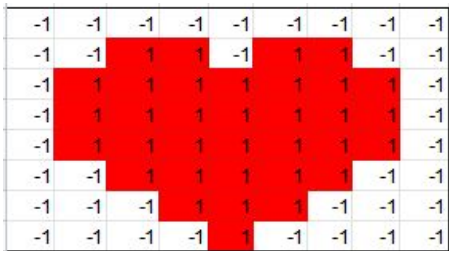
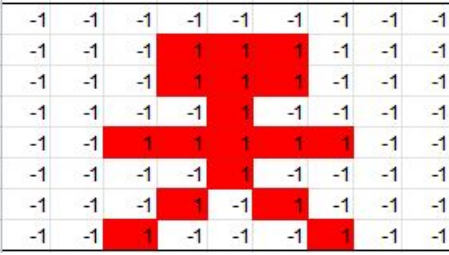


→ Con figuras:

Para la siguiente prueba primero “dibujamos” 5 con una dimensión de 8 filas por nueve columnas. En cada posición de esta matriz colocábamos un ‘1’ en caso de que estuviera coloreada y un ‘-1’ si era un espacio en blanco. Estos valores después los acomodamos en una sola fila y con esto generábamos un vector prototipo.

Como tenemos 5 figuras, al final obtendremos 5 vectores prototipo y cada vector prototipo contiene 72 valores. En conclusión, nuestra matriz W1 queda con un tamaño total de 5 filas(neuronas) por 64 columnas(entradas).

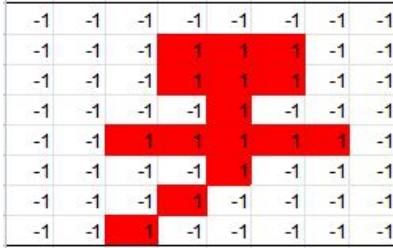
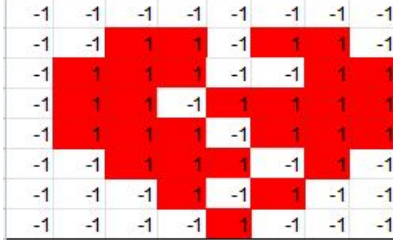
Figura	Vector Prototipo
<p>Rectángulo(Clase 1)</p>	-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 1 1 1 1 1 1 -1 1 1 1 1 1 1 1 1 -1 1 1 1 1 1 1 1 1 -1 1 1 1 1 1 1 1 1 -1 1 1 1 1 1 1 1 1 -1 1 1 1 1 1 1 1 1 -1 -1 -1 -1 -1 -1 -1 -1 -1
<p>Rombo(Clase 2)</p>	-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 -1 1 1 1 -1 -1 -1 -1 -1 1 1 1 1 1 -1 -1 -1 1 1 1 1 1 1 1 -1 -1 -1 1 1 1 1 1 -1 -1 -1 -1 -1 1 1 1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

 <p>Triángulo(Clase 3)</p>	<pre>-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 1 -1 -1 -1 -1 -1 -1 1 1 1 1 1 -1 -1 -1 -1 1 1 1 1 1 1 1 -1 -1 -1 1 1 1 1 1 1 1 1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1</pre>
 <p>Corazón(Clase 4)</p>	<pre>-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 1 -1 1 1 1 -1 -1 1 1 1 1 1 1 1 1 -1 -1 1 1 1 1 1 1 1 1 -1 -1 1 1 1 1 1 1 1 1 -1 -1 1 1 1 1 1 1 1 1 -1 -1 -1 -1 1 1 1 1 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1</pre>
 <p>Persona(Clase 5)</p>	<pre>-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 1 -1 -1 -1 -1 -1 -1 -1 1 1 1 1 -1 -1 -1 -1 -1 -1 1 1 1 1 1 -1 -1 -1 1 1 1 1 1 1 1 1 -1 -1 1 1 1 1 1 1 1 1 -1 -1 -1 -1 -1 1 1 1 -1 -1 -1 -1 1 -1 -1</pre>

Todos estos vectores están definidos en el archivo prototipoFiguras.txt en el mismo orden. Por lo tanto la salida la interpretaremos de la siguiente forma:

Figura	Clase correspondiente	Valor a la salida del programa
Rectángulo	Clase 1	a21
Rombo	Clase 2	a22
Triángulo	Clase 3	a23
Corazón	Clase 4	a24
Persona	Clase 5	a25

Ahora tendremos nuestros vectores de entrada a la red. Para su funcionamiento generamos 2-vectores de entrada distintos que se muestran a continuación:

Figura	Vector de entrada	Archivo
 <p>Persona sin un pie</p>	-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 1 -1 -1 -1 -1 -1 1 1 1 -1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 1 1 1 1 1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1	figuraPersona.txt
 <p>Corazón roto</p>	-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 -1 1 1 -1 -1 -1 1 1 -1 -1 1 1 -1 -1 1 1 -1 1 1 1 -1 -1 1 1 -1 1 1 1 -1 -1 -1 1 1 -1 -1 -1 -1 -1 -1 1 -1 1 -1 -1 -1 -1 -1 -1 1 -1 -1 -1 -1	vectorCorazon.txt

### Con el vector de la persona sin un pie:

Para hacer uso de este vector indicaremos “figuraPersona” en el programa cuando solicite el nombre del archivo que contiene al vector de entrada. Para el nombre del archivo que contiene los vectores prototipo introduciremos la cadena “prototipoFiguras”. Para esto es necesario que ambos archivos se encuentren en la misma ruta que el programa. La forma de introducir las cadenas en el programa es la siguiente:

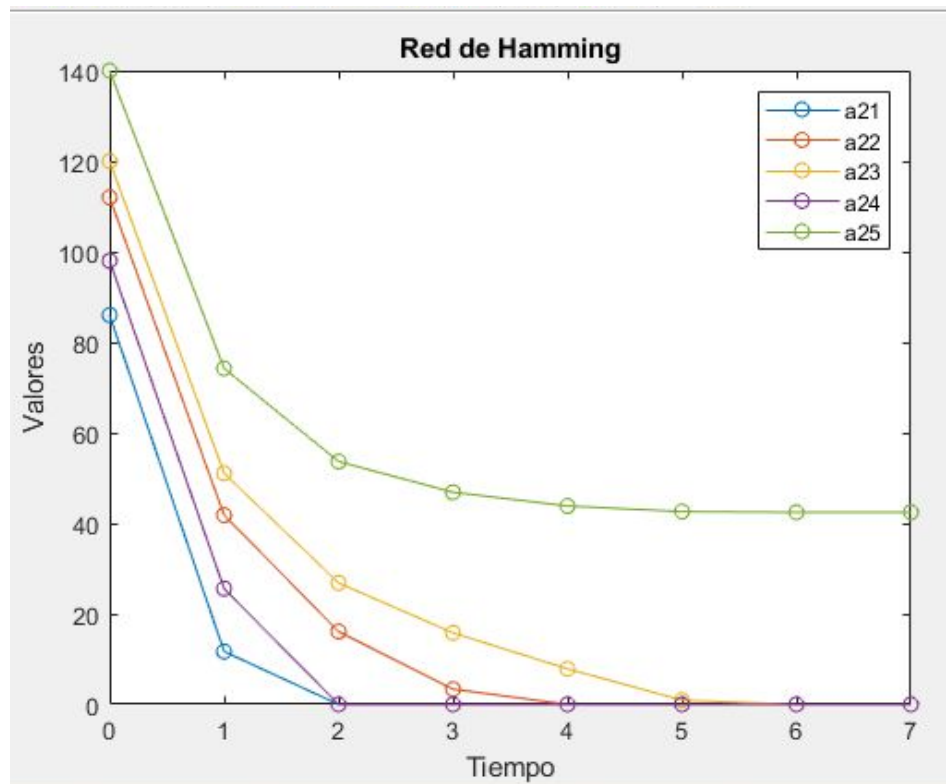
Ingrese el nombre del archivo que contiene el vector de entrada(sin extension .txt):

figuraPersona

Ingrese el nombre del archivo que contiene los vectores prototipo(sin extension .txt):

prototipoFiguras

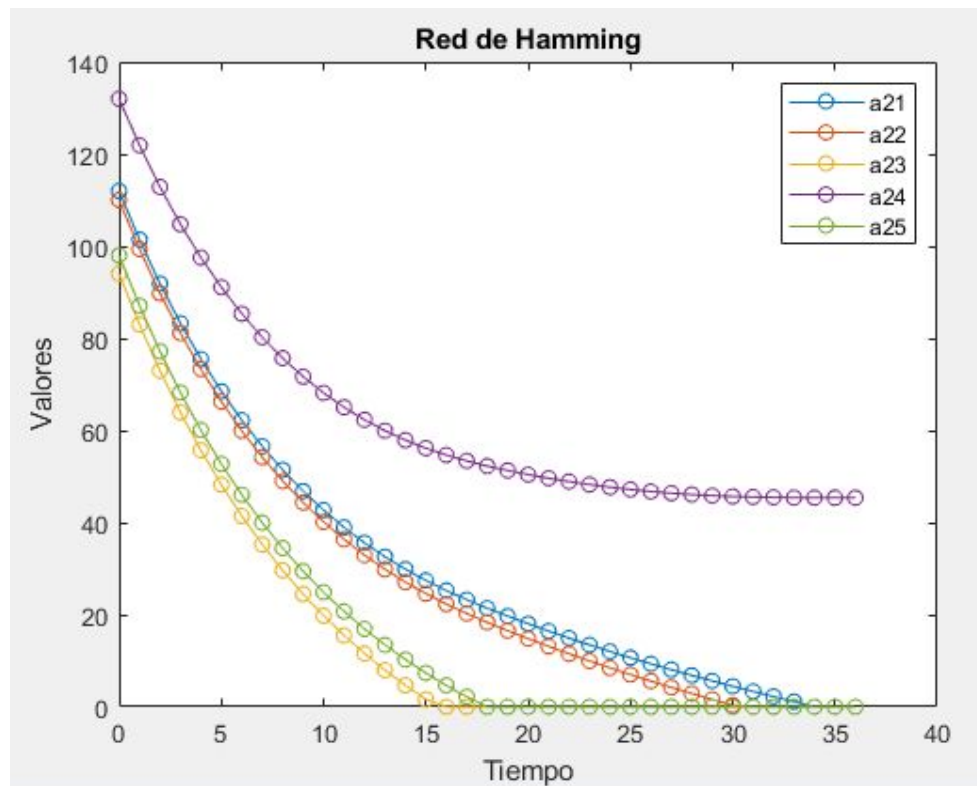
Cuando el programa termine de ejecutar las operaciones correspondientes nos mostrará en la consola que la ganadora es la clase 5(“**La clase ganadora fue: a25**”) y se desplegará una ventana emergente con la gráfica de los valores a lo largo de todas las iteraciones. La gráfica no siempre será la misma debido a que épsilon se genera de forma aleatoria y como ya vimos en las pruebas anteriores, el valor de épsilon influye en el número de iteraciones. La gráfica que nos generó para esta prueba fue la siguiente:



En el caso de la segunda prueba el archivo de vectores prototipo será el mismo, lo que cambiará será el nombre del archivo que contiene al vector de entrada. Parapara este caso el nombre de archivo que introduciremos será “vectorCorazon”. A continuación se da el ejemplo:

Ingrese el nombre del archivo que contiene el vector de entrada(sin extension .txt):  
vectorCorazon

Como era de esperarse, esta vez la gráfica y la clase ganadora es otra. Ahora la consola nos indica que la clase ganadora es la clase 4 (“**La clase ganadora fue: a24**”) y la gráfica que nos despliega es la siguiente:



De esta prueba podemos observar cosas relevantes. La primera es que esta red es muy versátil, dándole valores adecuados tanto a los vectores prototipos como a los vectores de entrada podremos cumplir con la labor de clasificación sin mayor problema. Como se aprecia en ambas gráficas, en el primer caso gana a25 y en el segundo caso gana a24 que coincide con la relación que hay entre la salida del programa y la clase a la que corresponde dicha salida para este ejercicio.

épsilon 1: 1.580898e-01

épsilon 2: 2.438510e-02

### Discusión:

La red de Hamming tiene como base la distancia de Hamming, cada vector prototipo se considera un atractor y tras cada época el vector de entrada se irá acercando a alguno de los atractores. La interpretación del resultado es muy fácil debido a que al ser una red competitiva todas las neuronas excepto una serán deshabilitadas y la interpretación es basándonos en cómo se acomodaron nuestros vectores prototipo en nuestra matriz de pesos  $W1$ .

Una de las desventajas de este modelo es que el valor de épsilon puede influir mucho en el tiempo(épocas) que tardará la red en converger. Otra desventaja es que esta red no siempre

converge por lo que al plantear los vectores prototipo hay que tener cuidado para que la red funcione de forma adecuada.

## Conclusiones:

La red de Hamming es una opción para poder reconocer patrones; sin embargo, dependiendo del número de entradas a la red y los vectores prototipo que se tienen y el valor de  $\epsilon$  que se proponga, el número de operaciones necesarias para generar la convergencia puede crecer demasiado, teniendo como consecuencia un costo computacional alto.

Esta red está constituida por dos capas. La primera capa es una capa feed-forward que es la encargada del cálculo de la distancia de Hamming y usa un bias para asegurar que los valores de la salida de la primera capa no sean negativos. La segunda capa es una capa recurrente y una de las características más peculiares es que en esta capa no hacemos uso de un bias.

Con este modelo no se puede asegurar que la red siempre convergerá pero si converge el resultado es “certero”. Existen otros modelos que siempre convergen y que se ahorran algunas operaciones (1 sola capa) como lo es la red de Hopfield; no obstante, aunque esta red siempre convergerá no siempre será el resultado correcto.

Por último se vio que con los mismos vectores prototipo y vector de entrada, con distintos valores de  $\epsilon$  podemos tener cosas distintas. Uno de los casos más peculiares de las pruebas fue cuando se hizo uso de 6- vectores prototipo y que en base al valor de nuestro  $\epsilon$  la red podía no converger o converger incluso a 2-clases. Esto se podría explicar porque  $\epsilon$  afecta en el tamaño de los “saltos” de nuestro vector de entrada a lo largo de las épocas. Por esa razón, con un  $\epsilon$  distinto (más grande o pequeño), la red podía quedar más cerca a una clase que a otra dentro del proceso de las épocas.

## Referencias:

- Ballesteros, A.** (2019). *“Redes Neuronales en java. Herramienta para redes Neuronales \*\* JavaBrain \*\*”*. Recuperado desde:  
<http://www.redes-neuronales.com.es/tutorial-redes-neuronales/red-neuronal-competitiva-simple.htm>
- Hagan, M., Demuth, H., Beale, M., & De Jesús, O.** (2016). *Neural network design* (2nd ed.). s. n.].



## Anexos:

### Archivo: P02\_Hamming\_2014081038.m

```
function [] = P02_Hamming_2014081038()  
%P2: Red de Hamming  
% Implementacion de La red de Hamming con lectura de vectores  
% prototipo y vector de entrada desde archivo txt; asi como la  
% graficacion de los resultados en las iteraciones  
%Fecha de elaboracion: 2019/03/07  
%Autor: Morales Flores Victor Leonel  
%Asignatura: Neural Networks  
%Escuela: ESCOM-IPN(MX)  
  
converge=false;  
numCeros=0;  
etiquetas="";  
bandera=0;  
claseGanadora=""  
iteracionesMaximas=50;  
  
archivoVecIn=input('Ingrese el nombre del archivo que contiene el vector de  
entrada(sin extension .txt): ','s');  
p=LecturaP(archivoVecIn);  
  
[R,n]=size(p);  
  
archivoVectores=input('Ingrese el nombre del archivo que contiene los  
vectores prototipo(sin extension .txt): ','s');  
W1=LecturaW1(archivoVectores,R);  
[S,R]=size(W1);  
  
b=zeros(S,1)+R;  
%Dado que para la capa feedforward la funcion de transferencia es  
%purelin(n) y purelin esta definida como purelin(n)=n, entonces  
%evitamos la variable n y asignamos directamente a 'a1'  
a1=W1*p+b;  
%Asignamos a1 a a2 ya que esta es nuestra condicion inicial  
a2=a1;  
GuardarArchivo(a2,0,"w");  
W2=GenerarW2(S);  
for i=1:iteracionesMaximas  
%Generacion de etiquetas para la grafica  
cadAux=strcat("a2",string(i)," ");
```



```
etiquetas=strcat(etiquetas,cadAux);
% Multiplicacion de la matriz de pesos W2 por a2(t) y asignacion a
% a2(t+1). Debido a la forma en que se opera(primeramente hace operacion
% y despues la asignacion a la variable) podemos hacer uso de una sola
% variable a2
a2=W2*a2;

%Funcion que revisa la convergencia de a2
[a2,converge,claseGanadora]=RevisarConvergencia(a2,S);
%Guarda en nuestro archivo los resultados con argumento 'a' para
%que escriba al final del documento.
GuardarArchivo(a2,i,"a");
%Bloque de codigo para hacer una iteración mas despues de que
%converge la primera vez. Esto es para confirmar la convergencia
if(converge)
    if(bandera==0)
        bandera=1;
        claseGanadora=cadAux;
        continue;
    else
        break;
    end

end

end

if converge
    fprintf("La clase ganadora fue: %s",claseGanadora);
else
    fprintf("No se logró convergencia tras %d iteraciones",iteracionesMaximas);
end
Graficar(S,etiquetas);

end
```

#### **Archivo: LecturaP.m**

```
function p = LecturaP(ArchivoVecIn)
% Con el fin de un código mas legible se crea esta función que se
% encargará de leer nuestro vector de entrada a la red a través del
% archivo 'ArchivoVecIn'. Esta variable es una cadena a la que el código
% mismo le concatenará la extensión txt(.txt) por lo que al ingresar por
% teclado el nombre del archivo, este debe ser un archivo TXT y no se le
% debe colocar la extensión del archivo.
%Fecha de elaboración: 2019/03/07
%Autor: Morales Flores Victor Leonel
%Asignatura: Neural Networks
```





```
%Escuela: ESCOM-IPN(MX)
```

```
fileVecIn = fopen(strcat(ArchivoVecIn, '.txt'), 'r');  
p=fscanf(fileVecIn, "%d");  
fclose(fileVecIn);
```

```
end
```

#### **Archivo: LecturaW1.m**

```
function W1 = LecturaW1(archivoVectores,R)  
% Con el fin de un código mas legible se crea esta función que se  
% encargará de leer nuestros vectores prototipo que generarán la matriz de  
% pesos W1 de la red a través del  
% archivo 'archivoVectores'. Esta variable es una cadena a la que el código  
% mismo le concatenará la extensión txt(.txt) por lo que al ingresar por  
% teclado el nombre del archivo, este debe ser un archivo TXT y no se le  
% debe colocar la extensión del archivo.  
% También recibe el número de entradas a la red(R) para darle el formato  
% adecuado a  
% la matriz de pesos. Esto es con el objetivo de hacer el código más  
% versátil para  
% que soporte procesamiento de distintos vectores de entrada siempre y  
% cuando las  
% columnas de W1 y filas de nuestro vector de entrada(p) coincidan en los  
% archivos TXT  
%Fecha de elaboración: 2019/03/07  
%Autor: Morales Flores Victor Leonel  
%Asignatura: Neural Networks  
%Escuela: ESCOM-IPN(MX)  
    sizeW1=[R Inf];  
    fileVectores = fopen(strcat(archivoVectores, '.txt'), 'r');  
    W1=fscanf(fileVectores, "%d", sizeW1);  
    fclose(fileVectores);  
    W1=W1.';  
  
end
```

#### **Archivo: GuardarArchivo.m**

```
function GuardarArchivo(a2,i,opc)  
% Con el fin de un código mas legible se crea esta función encargará  
% de escribir en un archivo de texto 'results.txt' todos los valores  
% a2 junto con 'i' que corresponde al número de interacción en la que  
% se generaron esos valores. opc permite que decidamos entre 'w' para  
% reescribir el archivo en caso que exista o 'a' para escribir al final  
% del mismo si es que existe.  
%Fecha de elaboración: 2019/03/07
```



*%Autor: Morales Flores Victor Leonel*

*%Asignatura: Neural Networks*

*%Escuela: ESCOM-IPN(MX)*

```
[S,]=size(a2);
results=fopen("results.txt",opc);
fprintf(results,"%d ",i);
for j=1:S
fprintf(results,"%d ",a2(j));
end
fprintf(results,"\n");
fclose(results);

end
```

#### **Archivo: GenerarW2 .m**

```
function W2 = GenerarW2(S)
% Con el fin de un código mas legible se crea esta función encargará
% de generar nuestra matriz de pesos W2 a partir de un épsilon aleatorio
% y 1's en la diagonal principal. Tal y como está especificada la
% construcción de esta matriz por el modelo de la red de Hamming
%Fecha de elaboración: 2019/03/07
%Autor: Morales Flores Victor Leonel
%Asignatura: Neural Networks
%Escuela: ESCOM-IPN(MX)
epsilon=0;
while epsilon==0
epsilon=(1/(S-1))*rand;
end
fprintf("Epsilon:%d",epsilon);
%Creamos la matriz W2 con ceros menos epsilon
W2=zeros(S,S)-epsilon;
%Llena la diagonal de la matriz con 1's.
for i=1:S
W2(i,i)=1;
end

end
```

#### **Archivo: RevisarConvergencia.m**

```
function [a2,converge,claseGanadora] = RevisarConvergencia(a2,S)
% Con el fin de un código mas legible se crea esta función encargará
% de revisar que en nuestro vector de salida a2 se haya generado la
% convergencia(todas las neuronas deshabilitadas, excepto 1) y también
% implementa el comportamiento de la función de activación poslin
% (si n<0 ->a=0;
% si n>=0 -> a=n).
%Fecha de elaboración: 2019/03/07
```



```
%Autor: Morales Flores Victor Leonel  
%Asignatura: Neural Networks  
%Escuela: ESCOM-IPN(MX)
```

```
numCeros=0;  
converge=false;  
claseGanadora="";  
    for j=1:S  
        if(a2(j)<0)  
            a2(j)=0;  
            numCeros=numCeros+1;  
        else  
            claseGanadora=strcat("a2",string(j)," ");  
        end  
    end  
    if numCeros== S-1 && converge==false  
        converge=true;  
    end  
end
```

#### **Archivo: Graficar.m**

```
function Graficar (S,etiquetas)  
    %Funcion que permite graficar todos los valores obtenidos. Recibe S  
    %para poder leer de forma adecuada el archivo que contiene los datos y  
    %'etiquetas' contiene los elementos que se mostrarn en legend de la  
    %grafica  
    %Fecha de elaboración: 2019/03/07  
    %Autor: Morales Flores Victor Leonel  
    %Asignatura: Neural Networks  
    %Escuela: ESCOM-IPN(MX)  
  
    %Abrimos el archivo archivo results todos los valores que se crearon en  
    %las n iteraciones  
    results = fopen('results.txt','r');  
    %Formato de lectura del archivo  
    sizeRes=[S+1 Inf];  
    %Se recupera en Res la matriz de resultados y se cierra el archivo  
    %results.txt  
    Res=fscanf(results,"%f",sizeRes);  
    fclose(results);  
    %En t se almacena la primer fila del archivo que corresponde a la  
    %iteracion en la que se generaron los resultados de esa columna  
    t=Res(1,:);  
    %Se guarda en Aux solo una parte de la matriz leida que contiene solo  
    %los valores generados y elimina la primer fila que era del numero de
```



```
%iteracion
Aux=Res(2:end,:);
%Configuracion de La grafica a mostrar
plot(t,Aux,"o-");
title('Red de Hamming');
xlabel('Tiempo');
ylabel('Valores');
et=strsplit(etiquetas);
legend(et);

end
```