Práctica 6: Memorias Asociativas Dinámicas
Modelos de la Computación

Emilio Gomez Esteban

1. Enunciado de la prática

1. Diseña un script denominado asocLineal.m para calcular la salida de un asociador lineal con 5 componentes de entrada y 2 de salida.
   - ¿Qué cuatro patrones memorizarías para que la salida sea reconocida correctamente ante el patrón de entrada?
   - ¿Qué cuatro patrones memorizarías para que la salida NO sea reconocida correctamente ante el patrón de entrada? ¿Por qué ha devuelto incorrectamente la salida?
2. Diseña un script, asocNolineal.m, para computar la salida de un asociador no lineal simple que memorice el siguiente patrón: [-1 -1 1 1 1]
   - Si iniciamos la red en el estado [-1 -1 1 -1 1] ¿En qué estado se estabiliza la red?
   - Si iniciamos la red en el estado [1 1 -1 -1 1] ¿En qué estado se estabiliza la red? ¿Ha reconocido exactamente el patrón memorizado? ¿Por qué?
   - ¿Existe alguna entrada que haga que la red se estabilice en un estado diferente del patrón memorizado o de su opuesto? ¿por qué?
   - ¿De qué depende que la red se estabilice en su opuesto?
   - Muestra el script utilizado para las pruebas. Explica brevemente mediante comentarios qué son o para qué usas cada una de las variables.
3. Crea un script denominado asocLetras.m a partir del script letras.m que memorice los cinco patrones que hay en la matriz "d" en un asociador no lineal simple. Obtén la salida cuando le introducimos como entrada cada uno de ellos (ver Nota 1).
   - ¿Cuántas unidades de proceso has necesitado?
   - ¿En qué patrón o patrones comete error?
   - ¿Qué ocurre al introducir el patrón C?, ¿por qué crees que se estabiliza en ese estado?
   - Prueba a memorizar solo los patrones A y B, introdúcelos después como entrada ¿comete error? ¿a qué crees que se debe la diferencia con el caso en el que memoriza las 5 letras?

Sube los script asocLineal.m, asocNolineal.m y asocLetras.m.
Nota 1: Utiliza la función reshape para convertir de vector a matriz o viceversa. Por ejemplo: B=Reshape(A, 1,9*7) convierte la matriz A de 9x7 a un vector de 1x(9*7); C=Reshape(B, 9, 7) convierte el vector B a una matriz de 9 por 7.

2. Resolución de la práctica

Ejercicio 1

Implementación de la asocLineal.m:

```
clear all;

X = rand(4,5);
Y = rand(4,2);
X = orth(X.')';
Y = orth(Y);
%Si quitamos la ortonormalizacion, Y y X*W no coinciden

W = X'*Y;
Y
X*W
```

Obtenemos un ejemplo de salida:

```
Y =

   -0.5138   -0.0967
   -0.2924   -0.8422
   -0.6156    0.0325
   -0.5211    0.5294


ans =

   -0.5138   -0.0967
   -0.2924   -0.8422
   -0.6156    0.0325
   -0.5211    0.5294
```

Figura 1: Ejemplo

El objetivo es encontrar patrones de entrada que, al ser procesados por el asociador lineal, produzcan las salidas deseadas de manera reconocible y sin ambigüedades. La matriz W en el script representa los pesos del asociador lineal, calculados como $W = X^T Y$. Este cálculo tiene sentido cuando los patrones de entrada X y las salidas deseadas Y son ortonormales, ya que se puede garantizar que el producto XW genera las salidas deseadas Y de forma precisa.

Por tanto, para responder a la pregunta de qué patrones memorizaríamos para que la salida sea reconocida, debemos tener que los patrones de entrada sean ortonormales (linealmente independientes y unitarios)y que haya consistencia con la salida deseada, que significa que Y debe ser también ortonormal.. Por ejemplo, $x_1 = (1,0,0,0,0), x_2 = (0,1,0,0,0), x_3 = (0,0,1,0,0), x_4 = (0,0,0,1,0)$. Las correspondientes salidas podrían ser: $y_1 = (1,0), y_2 = (0,1), y_3 = (1,1), y_4 = (0,-1)$.

Para que la salida no sea reconocida correctamente ante un patrón de entrada en un asociador lineal como el definido en tu script, los patrones de entrada y salida seleccionados deben generar ambigüedades o inconsistencias en el cálculo. Por ejemplo, $x_1 = (1,0,0,0,0), x_2 = (0,1,0,0,0), x_3 = (1,1,0,0,0), x_4 = (0,0,1,1,0)$. Las correspondientes salidas podrían ser: $y_1 = (1,0), y_2 = (0,1), y_3 = (1,0), y_4 = (0,-1)$. En este caso, el tercer patrón de X, genera una salida inconsistente $(1,0)$, que ya se asocia al primer patrón. El cuarto patrón de X tiene un patrón de salida no alineado con los anteriores, lo que introduce ruido.

En este caso, el reconocimiento falla porque hay ambigüedad en la asociación, por la falta de independencia lineal y por la incosistencia de los datos (si los patrones de salida no reflejan una transformación única de los patrones de entrada, el modelo no puede generalizar correctamente).

Ejercicio 2

- Dado que s = [-1 -1 1 1 1] es el único patrón memorizado, y la red Hopfield tiene un comportamiento de atracción hacia los patrones almacenados, la red debería estabilizarse en este patrón, a menos que existan estados alternativos estables (mínimos locales). Ejecutando el programa, vemos que la red se estabiliza en el estado s = [-1 -1 1 1 1]. Esto es debido a que es un mínimo energético del sistema, y la dinámica iterativa de Hopfield minimiza la energía. Al iniciarse cerca del patrón memorizado, la red converge hacia él porque la energía disminuye con cada iteración.

- Vemos que la red se estabiliza en el estado s = [1 1 -1 -1 -1]. En redes Hopfield, el estado inicial puede influir en el resultado, dependiendo de si este está suficientemente cerca del patrón memorizado en términos de similitud. En este caso el estado inicial no tiene una gran similitud con el patrón memorizado ya que difiere en cuatro de las cinco posiciones.

- En una red de tipo Hopfield clásica, como la definida en tu código, la dinámica de la red y su estabilización dependen de la estructura de la matriz de pesos, la cual está diseñada para memorizar un único patrón. Además, el patrón memorizado s, y su opuesto $-s$ son siempre mínimos de energía. Esto significa que si se inicia la red en un estado cercano a cualquiera de ellos, se estabilizará en ese estado.

  Sí, existe la posibilidad de que una entrada inicial haga que la red se estabilice en un estado diferente al patrón memorizado o su opuesto, aunque en este caso particular, dado que solo hay un patrón almacenado, estos mínimos espurios son menos probables.

- La red se estabiliza en el opuesto del patrón memorizado $-s$ si el estado inicial está más cerca de $-s$ que de s y la simetría de la matriz de pesos garantiza que $-s$ sea un atractor válido.

Aquí tenemos la implementación del script asocNoLineal, con comentarios explicativos:

```
clear all;

% Definimos el patron que queremos memorizar en la red
s = [-1 -1 1 1 1];    %Patron que memorizamos

% Inicializamos la matriz de pesos
w = zeros(size(s,2),size(s,2));  % Matriz de pesos sinapticos de la red (n x n)
w = (1/size(s,2))*(s'*s);        % Calculamos la matriz de pesos segun la regla de Hebb
w = w - diag(diag(w));           % Eliminamos la contribucion de los pesos autoconectados (diagonal)

% Inicializamos el conjunto de estados a lo largo del tiempo
S = zeros(size(s,2),21);    % Matriz para guardar los estados de las neuronas en cada iteracion
S(:,1) = [1 1 -1 -1 1];     % Estado inicial de la red (patron de entrada)

% Comienza la iteracion para actualizar los estados de las neuronas
for t=2:21
    cambio=false;       % Bandera para detectar si hubo cambios en los estados
    S(:,t) = S(:,t-1);  % Inicializamos el estado actual con el estado anterior

    % Iteramos sobre cada neurona para actualizar su estado
    for i=1:size(s,2)
        h = w(i,:)*S(:,t);      % Calculamos el potencial local (suma ponderada de entradas)
        S(i,t)=(h>0)*2-1;          % Actualizamos el estado de la neurona: 1 si h > 0, -1 si h <= 0
        cambio = cambio || S(i,t) ~= S(i,t-1);    % Verificamos si hubo algun cambio
    end

    % Si no hubo cambios en ningun estado, la red se estabiliza y terminamos
    if ~cambio
```

```
29          S(:,t-1)    % Mostramos el estado final estabilizado
30          return     % Salimos del bucle porque la red ya esta
               estable
31      end
32  end
```

Ejercicio 3

Implementamos el siguiente script denominado asocLetras:

```
1  % Inicializacion de la matriz de pesos y vectores de patrones
2  W = zeros(9*7,9*7);           % Matriz de pesos sinapticos de
      la red (63 x 63, porque 9*7=63)
3  dVect = zeros(5,9*7);         % Matriz donde se almacenaran
      los patrones aplanados (5 patrones de tamano 63)
4
5  % Transformamos los patrones en vectores y calculamos la
      matriz de pesos
6  for i = 1:5
7      dVect(i,:) = reshape(d(:,:,i), 1, 9*7); % Aplana el
          patron (d(:,:,i)) a un vector de 63 elementos
8      W = W + dVect(i,:)' * dVect(i,:);       % Actualiza la
          matriz de pesos usando la regla de Hebb
9  end
10
11 % Normalizacion y eliminacion de pesos autoconectados
12 W = (1/size(W,1)) * W;        % Normalizamos los pesos para
      evitar que crezcan demasiado
13 W = W - diag(diag(W));        % Eliminamos las conexiones
      autoconectadas (diagonal de la matriz)
14
15 % Inicializacion de los estados y patron inicial
16 S = zeros(size(dVect,2), 21);              % Matriz para
      almacenar los estados de las neuronas (63 x 21)
17 S(:,1) = reshape(d(:,:,4), 9*7, 1);        % Establecemos el
       estado inicial como el patron (d(:,:,4))
18
19 % Proceso iterativo para actualizar los estados
20 for t = 2:21
21     cambio = false;                       % Bandera para
          detectar si hubo cambios en los estados
22     S(:,t) = S(:,t-1);                    % Inicializamos
          el estado actual con el estado anterior
23
24     % Actualizamos cada neurona
25     for i = 1:size(dVect,2)
```

```
26          h = W(i,:) * S(:,t);                    % Calculamos el
               potencial local para la neurona (i)
27          S(i,t) = (h > 0) * 2 - 1;               % Actualizamos el
                estado de la neurona ((1) si (h > 0), (-1) si (h
               <= 0))
28          cambio = cambio || S(i,t) ~= S(i,t-1); % Verificamos
               si hubo algun cambio en la neurona (i)
29      end
30
31      % Si no hubo cambios en los estados, la red se estabilizo
32      if ~cambio
33          reshape(S(:,t-1), 9, 7)                 % Reconstruimos y
                mostramos el estado final estabilizado como una
               matriz (9*7)
34          return                                  % Terminamos la
               ejecucion, ya que la red es estable
35      end
36  end
```

- En este script, los patrones memorizados están representados como matrices 9×7, y cada uno se aplanó (transformado en un vector) para poder almacenarse en la red. Esto implica que cada patrón tiene 63 unidades de procesamiento, que son las que ha necesitado la red.

- La salida reconoce correctamente el patrón $d(:,:,2)$, que coincide completamente con la salida estabilizada. Por tanto comete errores en todos los demás patrones.

- Al introducir C, la red no necesariamente recupera C debido a la interferencia cruzada en la matriz de pesos. Se estabiliza en un estado relacionado con patrones memorizados, posiblemente el más cercano en términos de conexiones sinápticas. Hemos visto que la red se estabiliza en un estado que difiera de C en un dígito. Esto es debido a la similitud de C con patrones memorizados, la interferencia cruzada de W y la tendencia de la red a converger a un mínimo local.

- No comete errores. Si memorizamos solo los patrones A y B e introducimos como entrada A, la red se estabiliza al estado $d(:,:,1)$, que es justamente A. Si introducimos B como entrada, ocurre algo análogo, la red se estabiliza al estado $d(:,:,2)$ que es B. La diferencia con el caso en el que memoriza las 5 letras es que la red muestra un comportamiento más fiable con menos patrones porque la matriz de pesos está menos saturada y los patrones son más fácilmente separables. Al introducir más patrones, aumenta la complejidad y las probabilidades de error.