Práctica 5: Redes de Hopfield - Optimización a través de redes neuronales
Modelos de la Computación

Emilio Gomez Esteban

1. Enunciado de la prática

1. Escribe un script denominado '3torresSec.m' que resuelva el problema de las N-torres para $N = 3$ con los siguientes requisitos:
   - El estado inicial será [0 1 1; 1 1 0; 0 1 0].
   - La dinámica de la red será asíncrona (secuencial), recorriendo primero la fila 1, luego la 2 y así sucesivamente.
   - Usa la regla de actualización que devuelve 1 cuando el potencial es mayor o igual que el umbral y cero en caso contrario.
   - Almacena en una variable denominada H, el potencial sináptico de la neurona que se esté actualizando en cada momento.
   - Explica brevemente mediante comentarios en el código para qué usas cada una de las variables que aparecen en el script.
2. Muestra el estado en el que se estabiliza la red.
   - ¿Es un mínimo global o local de la función de energía?
   - Atendiendo a tu código, ¿qué sentencia pondrías en Matlab para calcular el diferencial de energía que se produce después de actualizar una neurona?
   - Introduce una sentencia "return" en tu código para que, si el diferencial de energía es mayor que 0, termine la ejecución ¿Se ejecutará alguna vez?¿Por qué?
3. Usando como estado inicial [0 0 1; 0 1 1; 0 0 0]
   - Cuando se actualiza la neurona de la fila 1 columna 1 ¿se activa la neurona?¿Aumenta la energía?¿Cómo es posible que no aumente la energía si ahora hay dos neuronas activas en la primera fila?
   - ¿En qué estado se estabiliza finalmente?
   - ¿Es un mínimo global o local? Razona tu respuesta.
   - ¿Qué cambios harías en la regla de actualización para que converja a una solución óptima del problema de las 3-torres?

4. Modifica el script 3torresSec.m para que resuelva el problema con cualquier número de torres, denomina al nuevo script NtorresSec.m. El estado inicial se calculará aleatoriamente. Añade los comandos ”tic” y ”toc” para medir el tiempo de ejecución del script.
   - Calcula el tiempo medio de ejecución para 3 torres (usa varias ejecuciones).
   - Calcula el tiempo medio de ejecución para 30 torres (usa varias ejecuciones).
   - ¿Se mantiene la proporcionalidad en el tiempo de ejecución respecto del número de torres?¿por qué crees que ocurre eso?

Entrega las respuestas a las preguntas en un documento de texto y los script 3torresSec.m y NtorresSec.m.

2. Resolución de la práctica

Ejercicio 1

Aquí mostramos la implementación del script NtorresSec siendo $N = 3$ y siendo el estado inicial [0 1 1; 1 1 0; 0 1 0] con comentarios explicativos:

```
clear all; % Limpia el espacio de trabajo.

% Parametros iniciales
N = 3; % Tamano del tablero NxN (3x3).

% Inicializacion de la matriz de pesos sinapticos 4D.
w = zeros(N, N, N, N); % Pesos entre cada par de neuronas (i, j) y (l,k).

% Umbrales de activacion de las neuronas.
Theta = -ones(N, N); % Umbrales iniciales fijados a -1 para todas las neuronas.

% Configuracion de la matriz de pesos
for i = 1:N
    for j = 1:N
        % Penalizacion (peso -2) para la fila de la neurona actual (i,j).
        w(i, j, i, 1:N) = -2;

        % Penalizacion (peso -2) para la columna de la neurona actual (i,j).
        w(i, j, 1:N, j) = -2;

        % Sin autoconexion para la neurona actual (peso 0).
        w(i, j, i, j) = 0;
```

```matlab
    end
end

% Numero de iteraciones maximas permitidas.
epoc = 20;

% Historial de estados de la red.
Shist = zeros(N, N, epoc); % Tensor que almacena el estado del tablero en cada epoca.
Shist(:, :, 1) = [0, 1, 1; 1, 1, 0; 0, 1, 0]; % Estado inicial predefinido.
% Alternativa: inicializacion aleatoria del tablero.
% Shist(:, :, 1) = round(rand(N, N), 0);

% Inicio de las iteraciones de actualizacion.
for e = 2:epoc
    cambio = false; % Variable para verificar si hay cambios en el tablero.
    Shist(:, :, e) = Shist(:, :, e-1); % Inicializamos con el estado anterior.

    % Recorrido de todas las neuronas del tablero.
    for i = 1:N
        for j = 1:N
            h = 0; % Potencial sinaptico acumulado para la neurona (i, j).

            % Calculo del potencial sinaptico como suma ponderada de entradas.
            for l = 1:N
                for k = 1:N
                    h = h + Shist(l, k, e) * w(i, j, l, k); % Acumula contribuciones de (l,k).
                end
            end

            % Actualizacion del estado de la neurona segun el umbral.
            Shist(i, j, e) = int16(h >= Theta(i, j)); % Activada si h >= Theta(i,j).

            % Detectar si hubo cambios en el estado de esta neurona.
            cambio = cambio || Shist(i, j, e) ~= Shist(i, j, e-1);
        end
```

```
    end

    % Verificar convergencia: si no hay cambios, la red se estabilizo.
    if ~cambio
        % Mostrar el estado final estabilizado.
        Shist(:, :, e)
        return; % Salir del bucle si no hay mas cambios.
    end
end
```

Ejercicio 2

Mostramos el estado en el que se estabiliza la red:

$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

- La red se estabiliza cuando no hay más cambios en los estados de las neuronas, lo que implica que cualquier actualización no puede reducir más la energía. Por tanto, el estado final es un mínimo local. No se puede garantizar sin comparar explícitamente la energía de este estado con todas las configuraciones posibles. Sin embargo, si las restricciones del problema (una sola activación por fila y columna) son suficientes para excluir otros mínimos, este podría coincidir con el mínimo global.

- Añadimos el siguiente bloque de código justo debajo del final del for $l = 1{:}N$:

```
    h = h + Theta(i, j); % Anadir el umbral

    % Determinar el cambio de estado (Delta S)
    Delta_S = int16(h >= Theta(i, j)) - Shist(i, j, e); % Estado nuevo - estado anterior

    % Calcular el diferencial de energia (Delta E)
    Delta_E = -h * Delta_S;

    % Mostrar el diferencial de energia para esta neurona
    fprintf('Delta_E para la neurona (%d, %d): %f\n', i, j, Delta_E);
```

- Colocamos las siguientes líneas de código justo después de las líneas que hemos añadido en el apartado anterior:

```
    % Si el diferencial de energia es mayor que 0, terminar ejecucion
    if Delta_E > 0
```

```
        fprintf('Se detecto Delta_E > 0. Terminando
          ejecucion.\n');
        return;
    end
```

La sentencia return para el caso en que el diferencial de energía es mayor que 0 es teóricamente innecesaria, porque la dinámica de la red garantiza que el diferencial de la energía es menor ó igual que 0 en cada paso. Por lo tanto, el código nunca terminará debido a esta condición.

Ejercicio 3

- La neurona (1,1) no se activa con el estado inicial dado. La energía no aumenta, ya que aunque ahora hay dos neuronas activas en la fila 1, las activaciones cumplen la regla de estabilización (minimización de energía) y las interacciones entre neuronas activas disminuyen la energía total. La energía de la red no depende directamente de la cantidad de neuronas activas, sino de las interacciones entre neuronas y los pesos sinápticos.

- Mostramos el estado en el que se estabiliza la red:

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

- Es un mínimo local porque ninguna neurona puede cambiar de estado sin aumentar la energía. Este estado es estable frente a perturbaciones individuales. También es un mínimo global porque la energía en este estado es igual a la de cualquier otra configuración válida, y todas las configuraciones válidas tienen la misma energía. No existe ningún estado con menor energía que este.

- Incorporar penalizaciones explícitas por configuraciones inválidas, reforzar las restricciones en la regla de activación y utilizar enfoques probabilísticos o de normalización son estrategias que pueden guiar la red hacia configuraciones óptimas del problema de las 3 torres. Estos cambios garantizan que la energía decrezca de manera consistente, evitando mínimos locales que no sean válidos para el problema.

Ejercicio 4

Hemos añadido esta línea para que el estado inicial se calcule aleatoriamente:

```
% Alternativa: inicializacion aleatoria del tablero.
Shist(:, :, 1) = round(rand(N, N), 0);
```

- Tiempo de ejecución para 3 torres en 3 ejecuciones distintas: [0.0096 s, 0.0082 s, 0.0104 s]

- Tiempo de ejecución para 30 torres en 3 ejecuciones distintas: [0.2353 s, 0.2217 s, 0.1897 s]

- El aumento del tiempo de ejecución no es proporcional de manera directa al número de neuronas (N) debido a que la cantidad de conexiones y las interacciones entre las neuronas crecen cuadráticamente a medida que N aumenta. Esto provoca un incremento en la complejidad computacional del algoritmo y, por lo tanto, un aumento no lineal en el tiempo de ejecución.