Sistemas Inteligentes 1

Práctica 5
Redes Neuronales.
Redes no supervisadas

Héctor Garbisu Arocha
Curso 2015-2016
Grado en Ingeniería Informática
Universidad de las Palmas de Gran Canaria

Índice

1. Presentación del problema

Se debe hacer un mapa auto-organizativo (mapa de Kohonen) que diferencie un conjunto de muestras. En mi caso las entradas serán los dígitos manuscritos del MNIST. El mapa estará compuesto por cien nodos dispuestos en una retícula de 10 por 10. El resultado debería mostrar que los centroides de la distribución para cada dígito diferente están dispersos a lo largo del mapa según su parecido.
Como la estructura es de una sola capa, espero que los parecidos que se obtengan de los diversos dígitos sean superficiales y no se tengan en cuenta características complejas como el número de trazos o las curvas.

2. Esquema general

En un mapa de Kohonen se comparan los pesos de cada neurona de la capa oculta con las entradas y se refuerza la neurona que menor distancia tenga entre sus pesos y las entradas. Además de esta neurona, llamada best matching unit (BMU), también se refuerzan las neuronas (o nodos) colindantes, según un esquema que dependerá del sentido geográfico que se le quiera dar a la red.

En el código, esta distancia respecto a la entrada se calcula mediante la función dist (que ya viene en Matlab). El BMU se localiza buscando el mínimo de entre todas las distancias.

Tras localizar la BMU se actualizan los pesos de dicho nodo y de los cercanos, para aproximarse más a la entrada a la que han de parecerse. El factor de aprendizaje es η, y el factor de vecindad es σ. Ambos factores decrecen con el tiempo para que el mapa vaya teniendo mejor precisión a medida que se aprenden las diferencias entre los dígitos.

3. Implementación

La implementación básica no tiene en cuenta la convergencia de las variables η y σ. La razón para ello es que a primera vista no se puede saber cual es el margen a partir del que los pesos se actualicen demasiado poco. En consecuencia, cada época tendrá un periodo donde se produce realmente el aprendizaje y otro en el que la variación es demasiado pequeña.
A pesar de ésto, la cantidad de muestras de entrada es suficientemente grande como para suponer que sus características estarán homogéneamente distribuidas, lo que resultaría en que las partes de mayor aprendizaje estarán afectadas de forma similar entre todas las categorías.

Las variables inicialmente tienen un valor arbitrario. Tras varias ejecuciones he llegado a los siguientes:

```
sigma_ = 10; %Tamaño del vecindario en el tiempo
tau1 = 10000000000/log(sigma_); %Constante de actualizacion de sigma
eta = 1; %Learning rate
tau2 = 1000000; %Constante de actualización de eta
```

Inicialización de la red:

```
X = loadMNISTImages('train-images.idx3-ubyte');
[ninputs,sujetos] = size(X);
nnodos = 100;
W = rand(nnodos,ninputs);
```

En la variable 'sujetos' queda el numero de entradas por las que iterar (60k)

El entrenamiento consiste en bucle de épocas que envuelve a otro de 'batch', aunque en este caso los 'batch' son todas las imágenes del conjunto de entrenamiento. En el primer paso se calcula la distancia euclídea de los pesos de cada neurona con respecto a la entrada.

```
for iter=1:iteraciones
    fprintf('  %3d|',iter)
    for ci=1:sujetos % current individual
        distX = dist(W,X(:,ci));
```

Luego, se busca la BMU

```
        [minVal,minIdx] = min(distX);
```

Y se actualizan los pesos en función del tiempo transcurrido y la cercanía de cada nodo a la BMU, utilizando las funciones descritas en el guión de prácticas.

```
        % relevancia segun distancia
        h_ = exp(-(distBMU.^2)/(2*(sigma_^2)));
        % vecindad ponderada,
        v_p = eta.*(h_*unos(1,:)).*((unos(:,1)*(X(:,ci)')-W));
        W = W + v_p;
```

Por último, se actualizan las variables de aprendizaje

```
        eta = eta*exp(-(ci/tau2));
        sigma_ = sigma_*exp(-(ci/tau1));
    end
end
```

4. Evaluación

Para comprobar el estado de la red, utilicé otro script Matlab. El script divide el set de entrada en los 10 subcojuntos utilizando las etiquetas como guía. Luego, escribe el centroide de cada uno de los sets, indicándolo con texto. Adicionalmente se obtiene una matriz con la suma de cuántas veces cada neurona ha sido BMU para cada dígito y se guarda en 'winner' a qué dígito está más fuertemente asociado cada neurona.

```
la = loadMNISTLabels('train-labels.idx1-ubyte');
s_0 = find(la==0);
s_1 = find(la==1);
s_2 = find(la==2);
s_3 = find(la==3);
s_4 = find(la==4);
s_5 = find(la==5);
s_6 = find(la==6);
s_7 = find(la==7);
s_8 = find(la==8);
s_9 = find(la==9);

mapas = zeros(10,10,10);
mapas(:,:,1) = escribeCentroide(W,X(:,s_0),'CERO',10,10);
mapas(:,:,2) = escribeCentroide(W,X(:,s_1),'UNO',10,10);
mapas(:,:,3) = escribeCentroide(W,X(:,s_2),'DOS',10,10);
mapas(:,:,4) = escribeCentroide(W,X(:,s_3),'TRES',10,10);
mapas(:,:,5) = escribeCentroide(W,X(:,s_4),'CUATRO',10,10);
mapas(:,:,6) = escribeCentroide(W,X(:,s_5),'CINCO',10,10);
mapas(:,:,7) = escribeCentroide(W,X(:,s_6),'SEIS',10,10);
mapas(:,:,8) = escribeCentroide(W,X(:,s_7),'SIETE',10,10);
mapas(:,:,9) = escribeCentroide(W,X(:,s_8),'OCHO',10,10);
mapas(:,:,10) = escribeCentroide(W,X(:,s_9),'NUEVE',10,10);
axis([-1 11 -1 11]);

winner = zeros(10,10);
for i=1:10
for j=1:10
    [~,winner(i,j)] = max(mapas(i,j,:));
end
end
winner = winner - 1;
```

La función escribeCentroide se encarga de dibujar el centroide de cada mapa.

```
function mapa = escribeCentroide(W,conjunto,nombre,dim1,dim2)
    [x,y,mapa] = calculaCentroide(W,conjunto,dim1,dim2);
    text(x,y,nombre);
end
```

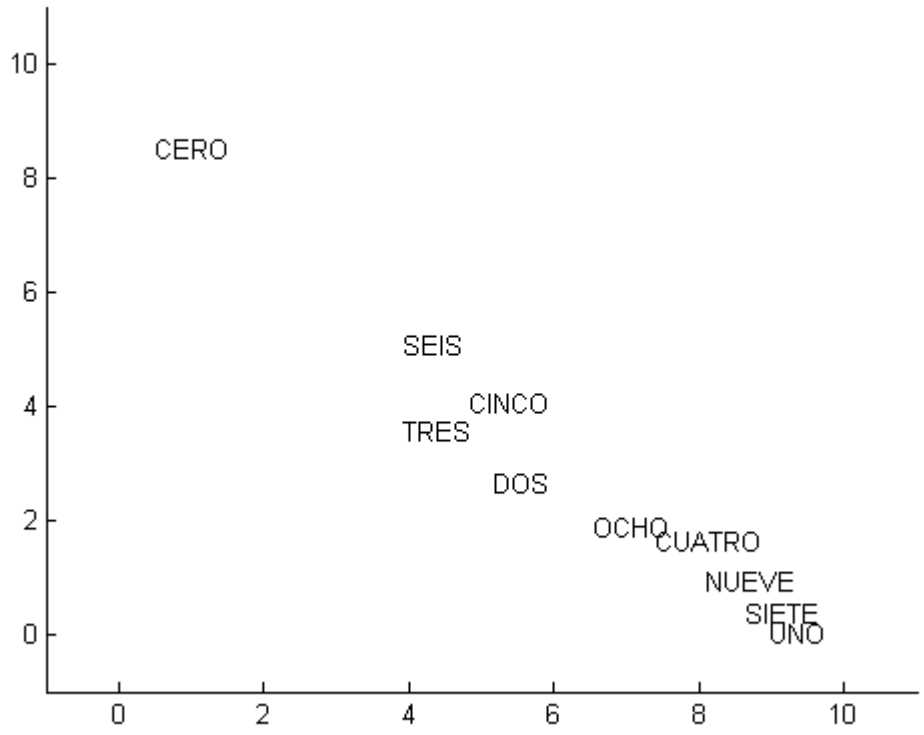
Los centroides se calculan como la media de sus dos componentes. Al mapa de cada distribución se le suma 1 en la posición donde se ha encontrado el BMU.

```
function [x,y,mapa] = calculaCentroide(W,distro,dim1,dim2)
    xs = zeros(1,size(distro,2));
    ys = xs;
    mapa = zeros(dim1,dim2);
    for i=1:size(xs,2)
        distX = dist(W,distro(:,i));
        [~,minIdx] = min(distX);
        [xtemp,ytemp] = getXY(dim1,dim2,minIdx);
        xs(i) = xtemp;
        ys(i) = ytemp;
        mapa(xtemp+1,ytemp+1) = mapa(xtemp+1,ytemp+1)+1;
    end
    x = mean(xs);
    y = mean(ys);
end
```

Ejemplo de resultado

```
winner =

     0     2     0     3     0     0     0     0     0     0
     0     2     3     0     3     0     0     0     0     0
     0     2     0     0     0     0     0     0     0     0
     0     2     0     0     0     0     0     0     0     0
     0     3     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     1     0     0     0     0     0     0     0     0     0
```

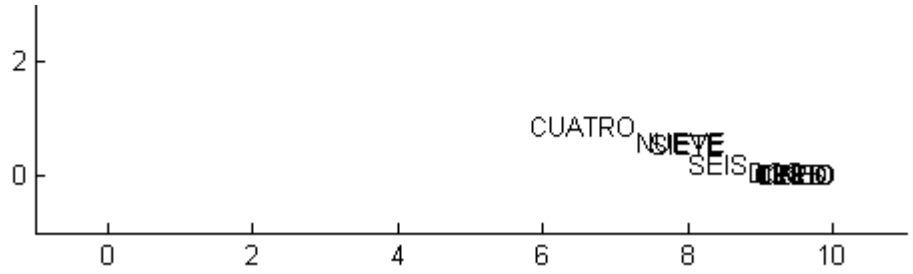


5. Resultados y ajuste de parámetros

Hay dos formas en que la red podría estar indebidamente configurada. De una forma podría ser que un mismo nodo esté pluriempleado y sea muy difícil diferenciar los valores que lo activan. El otro error en que puede caer la red es representar de forma muy poco diferenciada los nodos que activados por una categoría.
El primer caso de error está potenciado por la posibilidad de que varias entradas de diferente categoría sean muy similares entre sí, mientras que el segundo caso de error sería más probable ante entradas de la misma categoría que son muy diferentes.

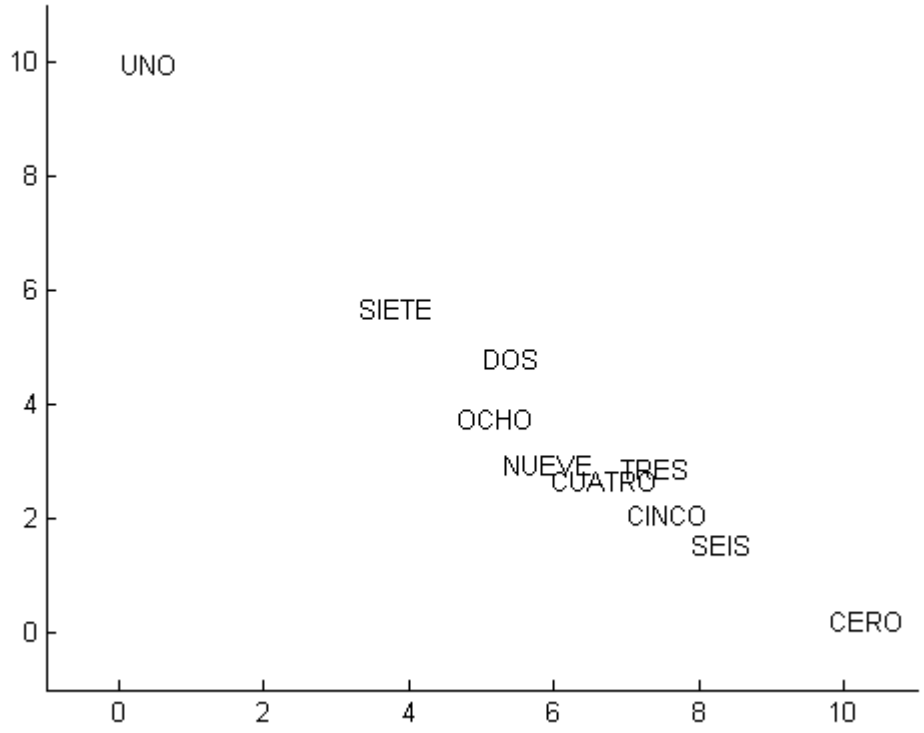
A falta de un algoritmo mejor, la táctica seguida para ajustar valores consiste en poner unas η y σ con valores naturales (entre 1 y 10). Luego, ajustar las τ para que las variables no se desvanezcan hacia valores demasiado pequeños muy rápidamente (considerando pequeño algo inferior a 1e-3 y demasiado rápido a menos de 5000 muestras). La arbitrariedad de los criterios es bastante grande y no sé si unos criterios mejores habrían llevado a obtener unas variables mejor ajustadas.

Ejemplo de variables mal ajustadas



Ejemplo con

```
sigma_ = 10; %Tamaño del vecindario en el tiempo
tau1 = 10000000000/log(sigma_); %Constante de actualizacion de sigma
% tau1 = 10000000000/log(sigma_); %Constante de actualizacion de sigma
eta = 1; %Learning rate
tau2 = 1000000; %Constante de actualización de eta
```



En las pruebas realizadas, es muy frecuente encontrar a “UNO” o “CERO” muy separado del resto. También suelen disponerse todos los centroides a lo largo una diagonal (se ve en la imagen), lo que me hace pensar que hay algo no ajustado bien del todo en el programa.

Lo único que me hace pensar que hay algo funcionando bien en el mapa auto-organizativo es el hecho de que en la mayoría de los casos las parejas de centroides más cercanos tienen sentido de similitud visual. Por ejemplo las parejas {5,6},{1,7} y {4,9} aparecen juntas en la mayoría de las pruebas. Como comentario adicional, habría dicho que {0,8} también es una buena pareja candidata, pero el programa no lo cree.

Una posible explicación que le doy a que no haya sido capaz de diferenciarlos mejor, es que realmente no hay tanta variedad entre los dígitos manuscritos como la hay entre todas las posibles imágenes 28x28 en escala de grises y, en ese sentido, 100 neuronas no sea suficiente para diferenciar dichos 10 grupos.
También podría haber probado a usar una plantilla exagonal en lugar de una red cuadrada. Sería curioso ver si la diagonal permanece tal cual, cambia de forma o forma otro patrón totalmente diferente.

6. Conclusión

En definitiva, la red es capaz de mostrar cierta capacidad cuando aplicamos conocimientos previos sobre la similitud de los caracteres, pero por algún motivo que no he identificado, es incapaz de reconocer cada dígito como una clase diferente, a pesar de estar en un marco que teóricamente permitiría diferenciar 100 clases.
No me tomo esto como un fracaso, ya que soy consciente de que este conjunto de ejemplo tiene variaciones que dependen incluso de la personalidad de quien ha escrito cada dígito haciendo que sea muy difícil de controlar.

Podría o no creerme que de 100 clases de dígitos, 91 se corresponden con el 0 porque haya más formas diferentes de escribir dicho número

```
>> size(find(winner==0))
ans =  91     1
```

En cualquier caso, presenté a un mapa de Kohonen un problema con muchas dimensiones no binarias y consiguió aprender algunas características que los humanos somos capaces de diferenciar.

7. Anexo. Código

SOM.m

```
%% Constantes
% La cantidad de nodos que se actualizan tras cada muestra
% se reduce. La fuerza con la que se actualizan tambien
% disminuye. Ambas variables, sigma y eta, varían respecto
% a sus respectivas constantes.
iteraciones = 10;
sigma_ = 10; %Tamaño del vecindario en el tiempo
tau1 = 10000000000/log(sigma_); %Constante de actualizacion de sigma
% tau1 = 10000000000/log(sigma_); %Constante de actualizacion de sigma
eta = 1; %Learning rate
tau2 = 1000000; %Constante de actualización de eta
% tau2 = 1000000; %Constante de actualización de eta
%% Carga e Inicialización
X = loadMNISTImages('train-images.idx3-ubyte');
[ninputs,sujetos] = size(X);
nnodos = 100;
W = rand(nnodos,ninputs);
%% Variables auxiliares
unos = ones(nnodos,ninputs);
progress_steps = 40;
%% Entrenamiento
tic
fprintf('iter |progreso%32s|\n',' ')
for iter=1:iteraciones
    fprintf('  %3d|',iter)
    for ci=1:sujetos
        distX = dist(W,X(:,ci));
    %%  buscar Best Matching Unit
        [minVal,minIdx] = min(distX);
    %%  actualizar pesos
        % distancias a BMU
        distBMU = distancia(10,10,minIdx);% grafica
        % relevancia segun distancia
        h_ = exp(-(distBMU.^2)/(2*(sigma_^2)));
        % vecindad ponderada,
        v_p = eta.*(h_*unos(1,:)).*((unos(:,1)*(X(:,ci)')-W));
        W = W + v_p;
    %%  actualizar variables
        eta = eta*exp(-(ci/tau2));
        sigma_ = sigma_*exp(-(ci/tau1));
    %%  control
        if mod(ci,round(sujetos/progress_steps))==0
            fprintf('#')
        end
    end
    fprintf('|\n')
end
toc
```

muestra.m

```
la = loadMNISTLabels('train-labels.idx1-ubyte');
s_0 = find(la==0);
s_1 = find(la==1);
s_2 = find(la==2);
s_3 = find(la==3);
s_4 = find(la==4);
s_5 = find(la==5);
s_6 = find(la==6);
s_7 = find(la==7);
s_8 = find(la==8);
s_9 = find(la==9);

mapas = zeros(10,10,10);
mapas(:,:,1) = escribeCentroide(W,X(:,s_0),'CERO',10,10);
mapas(:,:,2) = escribeCentroide(W,X(:,s_1),'UNO',10,10);
mapas(:,:,3) = escribeCentroide(W,X(:,s_2),'DOS',10,10);
mapas(:,:,4) = escribeCentroide(W,X(:,s_3),'TRES',10,10);
mapas(:,:,5) = escribeCentroide(W,X(:,s_4),'CUATRO',10,10);
mapas(:,:,6) = escribeCentroide(W,X(:,s_5),'CINCO',10,10);
mapas(:,:,7) = escribeCentroide(W,X(:,s_6),'SEIS',10,10);
mapas(:,:,8) = escribeCentroide(W,X(:,s_7),'SIETE',10,10);
mapas(:,:,9) = escribeCentroide(W,X(:,s_8),'OCHO',10,10);
mapas(:,:,10) = escribeCentroide(W,X(:,s_9),'NUEVE',10,10);
axis([-1 11 -1 11]);

winner = zeros(10,10);
for i=1:10
for j=1:10
    [~,winner(i,j)] = max(mapas(i,j,:));
end
end
winner = winner - 1;
```

escribeCentroide.m

```
function mapa = escribeCentroide(W,conjunto,nombre,dim1,dim2)
    [x,y,mapa] = calculaCentroide(W,conjunto,dim1,dim2);
    text(x,y,nombre);
end

function [x,y,mapa] = calculaCentroide(W,distro,dim1,dim2)
    xs = zeros(1,size(distro,2));
    ys = xs;
    mapa = zeros(dim1,dim2);
    for i=1:size(xs,2)
        distX = dist(W,distro(:,i));
        [~,minIdx] = min(distX);
        [xtemp,ytemp] = getXY(dim1,dim2,minIdx);
        xs(i) = xtemp;
        ys(i) = ytemp;
        mapa(xtemp+1,ytemp+1) = mapa(xtemp+1,ytemp+1)+1;
    end
    x = mean(xs);
    y = mean(ys);
end
```

getXY.m

```
function [x,y] = getXY(val,~,dim2)
    y = floor((val-1)/dim2);
    x = mod((val-1),dim2);
end
```