Herramientas de diseño de RNAs
Practica 1

David Guillermo Morales Sáez
2010/11

Índice

Programación en alto nivel

El diseño de Redes Neuronales a partir de lenguajes de programación de medio o alto nivel (como C++) permite al creador de la red configurarla completamente, además de definir su funcionamiento a la perfección. Esto permite realizar modificaciones en la red de manera clara, pero requiere un alto conocimiento en programación, además de requerirse un compilador. Además, se ha de crear una interfaz para su uso.

Pros:

- Permite definir absolutamente todo
- No se requiere ningún tipo de licencia para utilizarlo
- Existen gran cantidad de entornos de desarrollo

Contras:

- Requiere un conocimiento exhaustivo del lenguaje
- Se ha de crear una interfaz obligatoriamente

JNNS

El Java Neural Network Simulator (JNNS) es una aplicación basada en Java, que permite simular fácilmente distintas redes sin necesidad de conocimientos previos. Su interfaz facilita mucho el trabajo a la hora de diseñar la red, además de permitir ver claramente los parámetros, la red en cada momento y gráficas de error. Un problema grave que tiene viene condicionado por estar basado en una máquina Java, y es su inestabilidad, es decir, no es difícil que el programa se bloquee, requiriéndose un reinicio del programa y de toda la red.

Pros:

- No requiere conocimientos previos de programación
- Muestra todos los parámetros y gráficas
- Facilidad de uso
- Es gratuito

Contras:

- Es inestable
- No permite salirse de los márgenes preestablecidos

MatLab

MatLab nos permite diseñar redes neuronales sin mucha dificultad, siempre y cuando conozcamos la sintaxis del programa. Tiene un paquete creado exclusivamente para las Redes Neuronales, por lo que lo único que debemos hacer es declarar la red que deseemos con los parámetros e iterar tantas veces como sea posible. Un claro problema de este entorno es su licencia, ya que es de pago y no todos pueden permitirse obtener una de ellas. Además, el paquete de Redes

Neuronales es totalmente cerrado, por lo que no podemos hacer nada que esté fuera de lo permitido.

Pros:

- Sólo se necesita definir los parámetros, declarar la red e iterar
- Tiene funciones que muestran gráficas rápidamente
- Permite realizar operaciones rutinarias sin interacción del usuario

Contras:

- Requiere ciertos conocimientos del entorno
- Tiene una licencia de uso de pago
- No permite salirse de los márgenes preestablecidos

Ejercicios del Seminario

1.- Utilizando el neurosimulador SNNS se pretende diseñar y entrenar varias redes neuronales feed-forward capaces de aprender a transformar valores numéricos entre diferentes tipos de codificaciones y la codificación utilizada por el código Braile. Para ello habrán de diseñarse los ficheros correspondientes de patrones de entrenamiento así como los que almacenan las diferentes redes en si.

Utilizaremos los números del 0 al 9, los cuales los codificaremos de cuatro diferentes maneras:

1) Utilizando una neurona de entrada o salida asociada a cada uno de estos números (un total de 10 entradas o salidas), en dicha codificación ha de estar activo un único elemento, que indica el número a representar.

2) En modo binario, esto es, mediante cuatro elementos, representando respectivamente las diferentes potencias de dos de la codificación binaria del número a representar. El elemento que este activo indica un uno en la codificación binaria y el inactivo un cero.

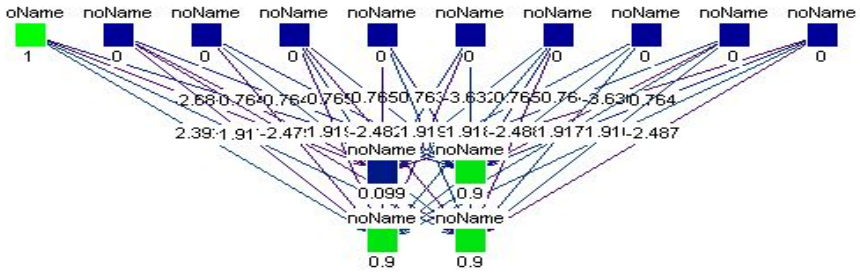
3) En modo decimal, con un solo elemento, de forma que la magnitud de éste indique el valor que se quiere representar, una neurona por tanto.

4) En el código Braile, cada numero viene representado en una matriz de 2x3 puntos (6 neuronas, dos de ellas inutiles) con los correspondientes valores de (on/off).

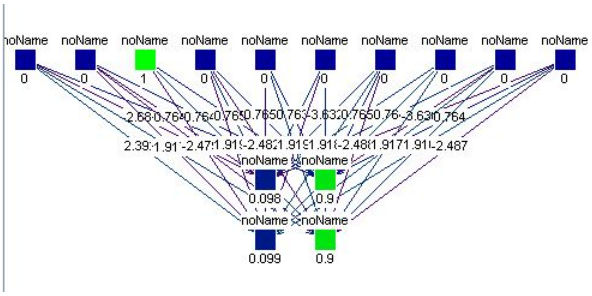
Se pretende por tanto generar y entrenar seis redes neuronales diferentes:

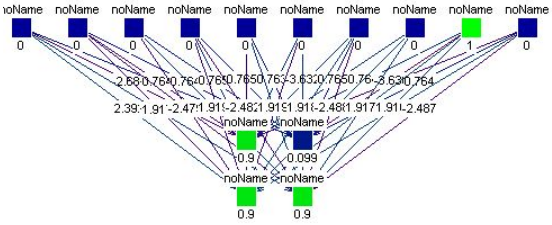
Conversor de código 1 al 4.

Vamos a crear una red que convierta la primera codificación de un número (por posición) al código Braile. Para ello usaremos la siguiente red:



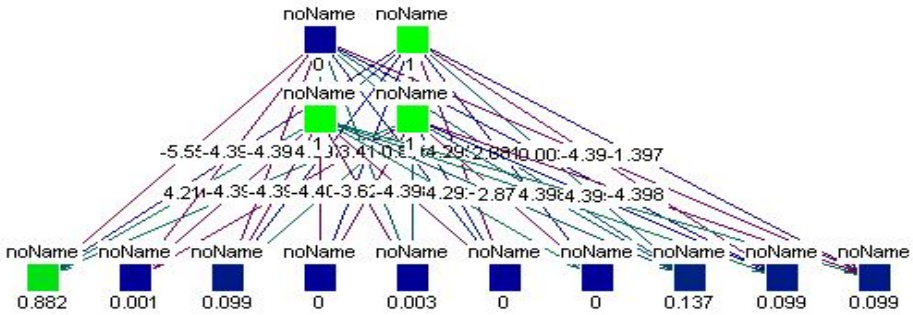
Y podemos comprobar que hemos conseguido que converja:



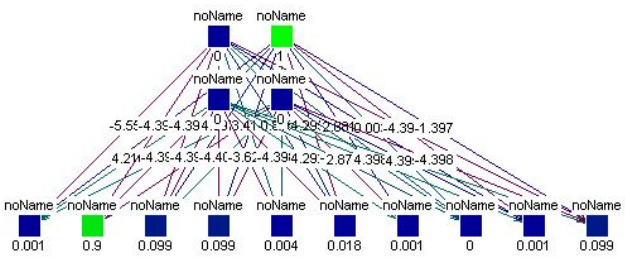


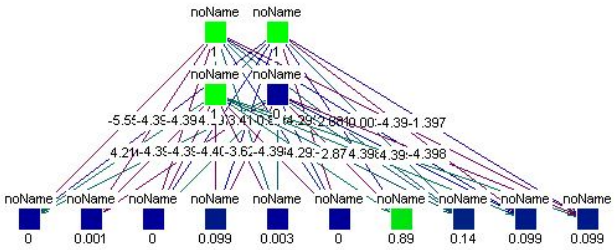
Conversor de código 4 al 1.

En este caso, haremos lo contrario del caso anterior, convertiremos el código Braile a una codificación posicional. Para ello utilizaremos la siguiente red:



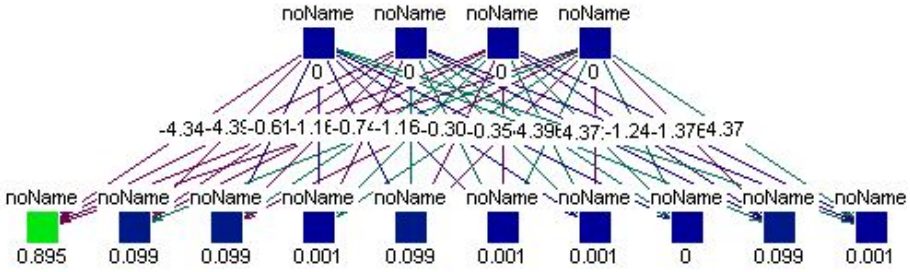
Volvemos a comprobar que la red converge:



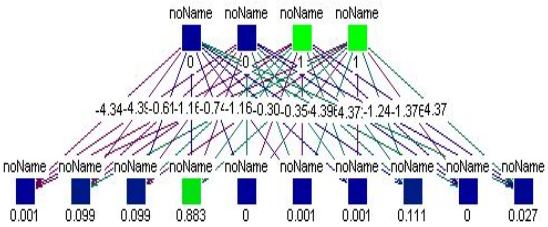


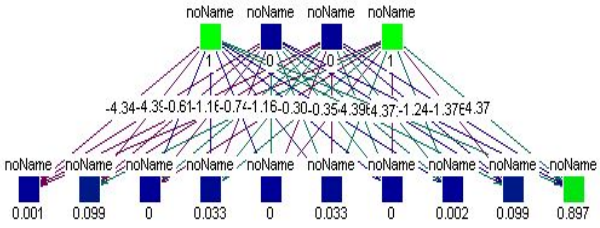
Conversor de código 2 al 4.

Se nos pide transformar un código binario a un código Braile, y para ello utilizaremos la siguiente red:



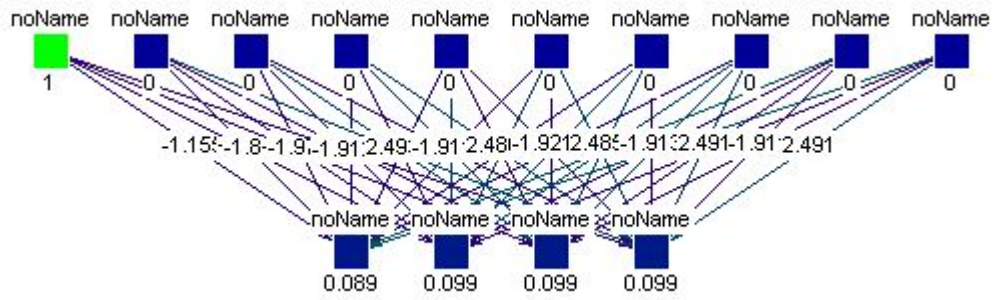
La red converge rápidamente, como podemos comprobar con algunos ejemplos:



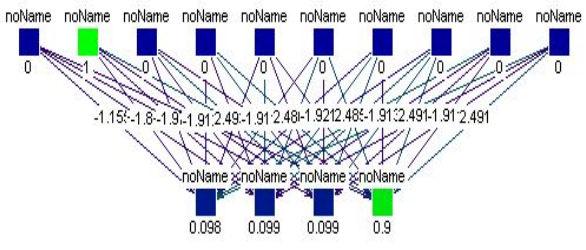


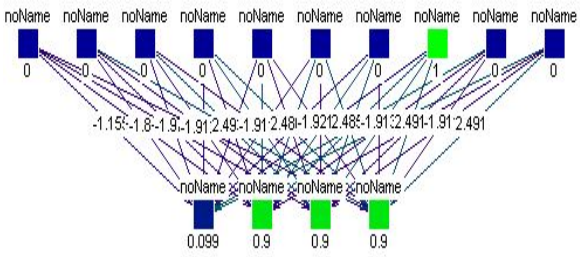
Conversor de código 4 al 2.

Como sucede en el par anterior, ahora crearemos una red que codifique el código Braile en binario:



Para comprobar la convergencia de la red:

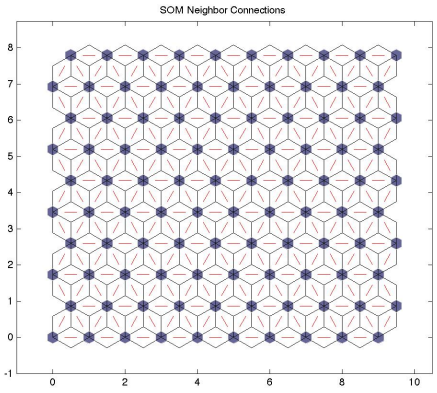


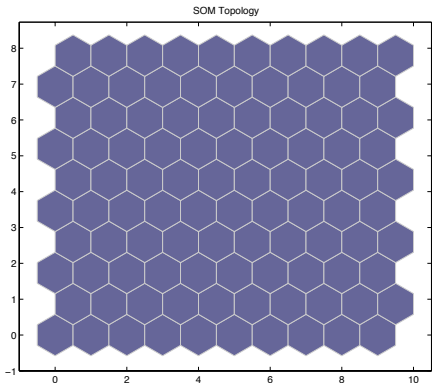


2.- El siguiente ejemplo de código MATLAB crea una red competitiva para clasificar ochos nubes de puntos en tres dimensiones ubicadas en los vértices de un cubo colocado en el origen de coordenadas. Se muestra en la figura 39 estas nubes de puntos

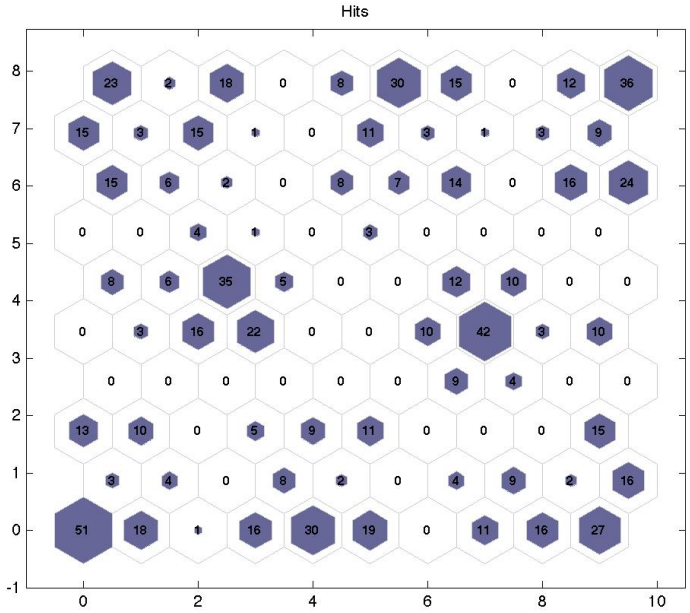
```
% Creamos los vectores de entrada.
p=[rands(3, 100)*.5 + [1* ones(1,100); 1 * ones(1,100); 1 * ones(1, 100)]...
   rands(3, 100)*.5 + [1* ones(1,100); 1 * ones(1,100); -1 * ones(1, 100)]...
   rands(3, 100)*.5 + [1* ones(1,100); -1 * ones(1,100); 1 * ones(1, 100)]...
   rands(3, 100)*.5 + [1* ones(1,100); -1 * ones(1,100); -1 * ones(1, 100)]...
   rands(3, 100)*.5 + [-1* ones(1,100); 1 * ones(1,100); 1 * ones(1, 100)]...
   rands(3, 100)*.5 + [-1* ones(1,100); 1 * ones(1,100); -1 * ones(1, 100)]...
   rands(3, 100)*.5 + [-1* ones(1,100); -1 * ones(1,100); 1 * ones(1, 100)]...
   rands(3, 100)*.5 + [-1* ones(1,100); -1 * ones(1,100); -1 * ones(1, 100)]];

f = figure('visible', 'off');
plot3(p(1, :), p(2, :), p(3, :), 'k+');
% Almacenamiento de la gr∑fica en disco
print(f, '-depsc2', strcat('figure_a_', num2str(0), '.eps'));
print(f, '-djpeg', strcat('figure_a_', num2str(0), '.jpg'));
close(f);
% Creamos la red de tipo SOM.
% net=newsom([-1.5 1.5; -1.5 1.5; -1.5 1.5], [8 8]);
net=newsom(p, [10 10]);
save('WS_a_0');
net.trainParam.showWindow = false;
net.trainParam.showCommandLine = true;
net.trainParam.show = 1;
f = plotsomnc(net);
% Almacenamiento de la gr∑fica en disco
print(f, '-depsc2', strcat('figure_top_', '.eps'));
print(f, '-djpeg', strcat('figure_nc_', '.jpg'));
close(f);

v_i = 0;
while v_i < 100
    v_i = v_i + 5;
    net.trainParam.epochs = 5; net=train(net,p);
    f = plotsomhits(net, p);
    % Almacenamiento de la gr∑fica en disco
    print(f, '-depsc2', strcat('figure_hits_', num2str(v_i), '.eps'));
    print(f, '-djpeg', strcat('figure_hits_', num2str(v_i), '.jpg'));
    close(f);
    f = plotsomnd(net);
    % Almacenamiento de la gr∑fica en disco
    print(f, '-depsc2', strcat('figure_nd_', num2str(v_i), '.eps'));
    print(f, '-djpeg', strcat('figure_nd_', num2str(v_i), '.jpg'));
    close(f);
    save(strcat('WS_a_', num2str(v_i)));
end
```

La topología de la red creada y las diferentes conexiones entre neuronas se pueden ver en las dos siguientes figuras, (figuras 40 y 41). Se utilizan las funciones **plotsomnc** y **plotsomtop** para obtener las siguientes representaciones.

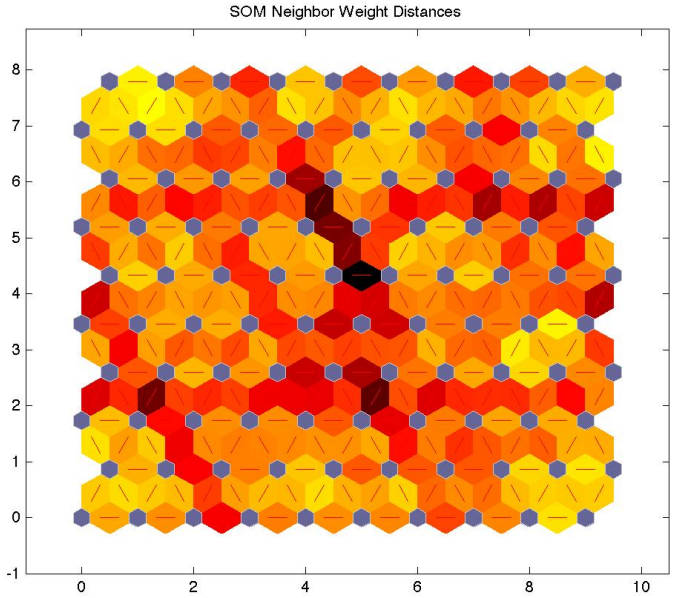




Los resultados de la red de 10x10 neuronas se muestran en las siguientes dos figuras. Se utilizan las funciones **plotsomhits** y **plotsomnd** para obtener dichas representaciones.



Este gráfica muestra el número de patrones de entrada clasificados por las neuronas de la red SOM. Se percibe claramente las ocho regiones, separadas por fronteras de neuronas que no clasifican patrones de entrada, en correspondencia a las 8 nubes de puntos ubicadas en los vértices de un cuadrado centrado en el origen de coordenadas.



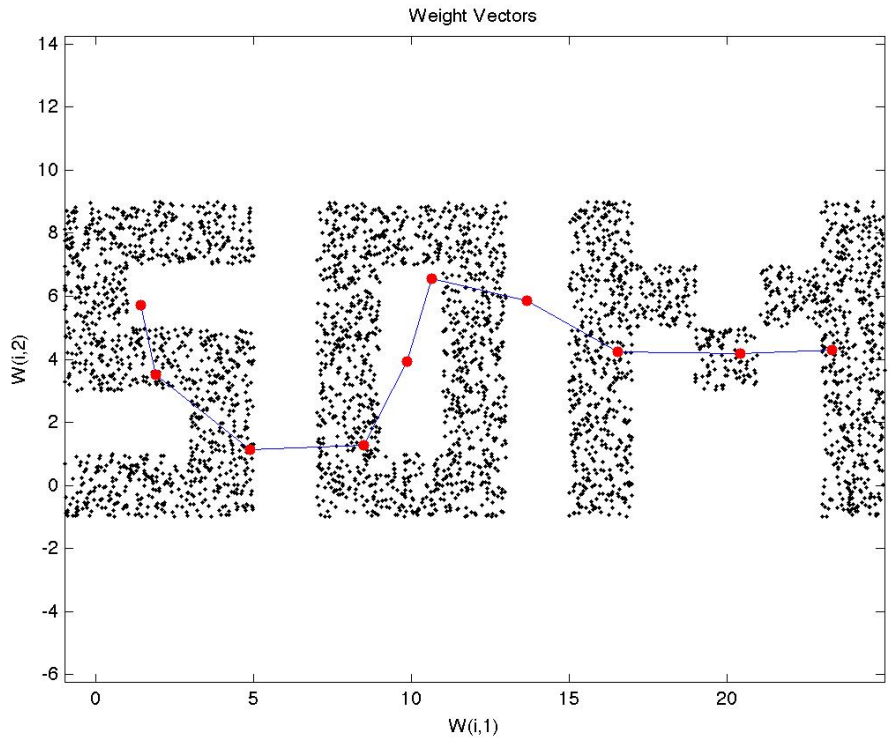
Esta gráfica muestra la distancia entre cada una de las neuronas vecinas de la red SOM. Colores oscuros representan grandes distancias y colores claros representan pequeñas distancias.

3.- El segundo ejemplo de código MATLAB crea una red competitiva para clasificar una nube de puntos en dos dimensiones, tal como se muestra en la figura 44.

```
% Creamos los vectores de entrada.
p=[ rands(2, 100) + [0 * ones(1,100); 0 * ones(1,100)]...
    rands(2, 100) + [2 * ones(1,100); 0 * ones(1,100)]...
    rands(2, 100) + [4 * ones(1,100); 0 * ones(1,100)]...
    rands(2, 100) + [4 * ones(1,100); 2 * ones(1,100)]...
    rands(2, 100) + [4 * ones(1,100); 4 * ones(1,100)]...
    rands(2, 100) + [2 * ones(1,100); 4 * ones(1,100)]...
    rands(2, 100) + [0 * ones(1,100); 4 * ones(1,100)]...
    rands(2, 100) + [0 * ones(1,100); 6 * ones(1,100)]...
    rands(2, 100) + [0 * ones(1,100); 8 * ones(1,100)]...
    rands(2, 100) + [2 * ones(1,100); 8 * ones(1,100)]...
    rands(2, 100) + [4 * ones(1,100); 8 * ones(1,100)]...
    rands(2, 100) + [8 * ones(1,100); 0 * ones(1,100)]...
    rands(2, 100) + [10 * ones(1,100); 0 * ones(1,100)]...
    rands(2, 100) + [12 * ones(1,100); 0 * ones(1,100)]...
    rands(2, 100) + [8 * ones(1,100); 2 * ones(1,100)]...
    rands(2, 100) + [12 * ones(1,100); 2 * ones(1,100)]...
    rands(2, 100) + [8 * ones(1,100); 4 * ones(1,100)]...
    rands(2, 100) + [12 * ones(1,100); 4 * ones(1,100)]...
    rands(2, 100) + [8 * ones(1,100); 6 * ones(1,100)]...
    rands(2, 100) + [12 * ones(1,100); 6 * ones(1,100)]...
    rands(2, 100) + [8 * ones(1,100); 8 * ones(1,100)]...
    rands(2, 100) + [10 * ones(1,100); 8 * ones(1,100)]...
    rands(2, 100) + [12 * ones(1,100); 8 * ones(1,100)]...
    rands(2, 100) + [16 * ones(1,100); 0 * ones(1,100)]...
    rands(2, 100) + [16 * ones(1,100); 2 * ones(1,100)]...
```

```
    rands(2, 100) + [16 * ones(1,100); 4 * ones(1,100)]...
    rands(2, 100) + [16 * ones(1,100); 6 * ones(1,100)]...
    rands(2, 100) + [16 * ones(1,100); 8 * ones(1,100)]...
    rands(2, 100) + [18 * ones(1,100); 6 * ones(1,100)]...
    rands(2, 100) + [20 * ones(1,100); 4 * ones(1,100)]...
    rands(2, 100) + [22 * ones(1,100); 6 * ones(1,100)]...
    rands(2, 100) + [24 * ones(1,100); 8 * ones(1,100)]...
    rands(2, 100) + [24 * ones(1,100); 6 * ones(1,100)]...
    rands(2, 100) + [24 * ones(1,100); 4 * ones(1,100)]...
    rands(2, 100) + [24 * ones(1,100); 2 * ones(1,100)]...
    rands(2, 100) + [24 * ones(1,100); 0 * ones(1,100)]];

% Creamos la red de tipo SOM.
net=newsom([-1 25; -1 9], [1 10]);

f = figure('visible', 'off');
plot(p(1, :), p(2, :), 'k.');
hold on;
plotsom(net.iw{1,1}, net.layers{1}.distances);
hold off;

% Almacenamiento de la gr∑fica en disco
print(f, '-djpeg', strcat('figure_', num2str(0), '.jpg'));

close(f);
save('WS_0');

net.trainParam.showWindow = false;
net.trainParam.showCommandLine = true;
net.trainParam.show = 1;

v_i = 0;
while v_i<100
    v_i = v_i + 5;

    net.trainParam.epochs = 5;
    net=train(net,p);

    f = figure('visible', 'off');
    plot(p(1, :), p(2, :), 'k.');
    hold on;
    plotsom(net.iw{1,1}, net.layers{1}.distances);
    hold off;

    % Almacenamiento de la gr∑fica en disco
    print(f, '-djpeg', strcat('figure_', num2str(v_i), '.jpg'));

    close(f);
    save(strcat('WS_', num2str(v_i)));
end
```

La siguiente figura (figura 100) muestra la ubicación de las neuronas después de entrenar la red durante 20 épocas.



En este caso se ha elegido una red SOM lineal de 10 neuronas.