Redes Neuronales

Trabajo Práctico 3

Alumno: Mauro Nacachian – 99619

Redes de Kohonen

Las redes de Kohonen realizan un aprendizaje del tipo no-supervisado, es decir, el sistema descubre por sí mismo la estructura de los datos y no hay una acción que le diga que hizo bien o mal. Son capaces de generar mapas auto-organizados, preservando distancias en el espacio de entrada de modo que orientaciones parecidas disparan neuronas cercanas en el espacio de neuronas.

La actualización de los pesos de las neuronas se hace de la siguiente manera:

$$\Delta w_{ij} = \eta \Lambda(i, i^*)(\xi - w_{ij})$$

Siendo $\Lambda(i, i^*)$ la función de vecindad entre la neurona i y la neurona ganadora i^*, que es la que tiene menor distancia con el patrón ξ:

$$\Lambda(i, i^*) = exp(-\| \frac{r_i - r_{i^*}}{2\sigma^2} \|^2)$$

Con esta función, la red es capaz de obtener la información de como se distribuyen espacialmente los patrones de entrada. Se dice que estas redes pueden preservar la topología del problema.

(1) Red Kohonen que aprende distribución uniforme

Para este ejercicio se decide utilizar $p = 50$ patrones de entrada (puntos uniformemente distribuidos dentro de una figura geométrica) y una red de 14×14 neuronas (espacio de neuronas de dimensión 2). Cada neurona se conecta con sus vecinos y tiene asociado pesos que son vectores de dimensión 2, al igual que el espacio de entrada.

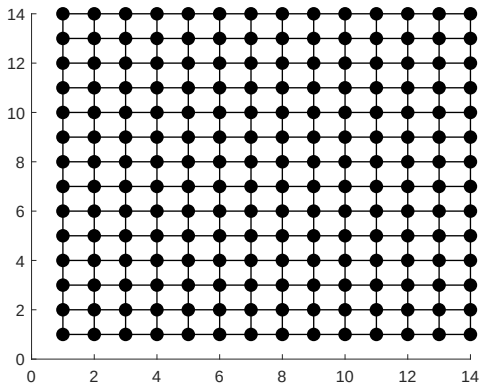

Fig. 1. Espacio de las neuronas.

Se utilizó el siguiente algoritmo para el aprendizaje de la Red de Kohonen:

1. Se elige uno de los patrones de entrada x al azar.
2. Se computa la distancia euclídea entre x y el vector w de cada neurona de la red
3. Se obtiene la neurona ganadora: La neurona con el peso con menor distancia al patrón x.
4. Se calcula la función vecindad para cada neurona utilizando la neurona ganadora y luego se utiliza para actualizar los pesos de todas las neuronas.
5. Se repiten los anteriores pasos hasta una cantidad limitada de iteraciones

A continuación se muestra la porción de código de entrenamiento, luego de haber inicializado los pesos de las neuronas, generado los patrones de entrada, definido σ (parámetro de la función de vecindad) y definido una cantidad limitada de iteraciones.

```
1
2  neurona_ganadora = zeros(p, 2); % coordenadas de la neurona ganadora
3  vecindad = zeros(p, n); % una matriz donde sus filas son listas con los
       valores de la vecindad de cada neurona;
4                          % cada fila es para un patron distinto
5  while(iter <= iteraciones)
6      display("Iteracion " + iter);
7
8      for mu = shuffle(1:p)
9
10         % 1. guardo el patron de entrada
11         x = xp(mu, :)';
12
13         % 2,3. calculo la distancia; encuentro las neuronas ganadoras;
14         dist = 0;
15         dist_min = 1000 * ones(p, 1);
16         for i = 1:n_x
```

```
            for j = 1:n_y

                dist = norm(x - w{i, j});

                if(dist <= dist_min(mu))
                    dist_min(mu) = dist;
                    neurona_ganadora(mu, :) = [i, j];
                end

            end
        end

        % 4,5. calculo las vecindades y actualizo los pesos
        idx_neu = 1;
        for i=1:n_x
            for j=1:n_y
                vecindad(mu, idx_neu) = func_vecindad([i, j],
                    neurona_ganadora(mu, :), sigma);
                w{i,j} = w{i,j} + eta * vecindad(mu, idx_neu) * (x - w{i,j
                    });
                idx_neu = idx_neu + 1;
            end
        end


    end
    sigma = sigma * alfa;
    iter = iter + 1;

end
```

La constante de aprendizaje η es de 0,1 para todas las redes del trabajo.

La varianza del ancho de la vecindad inicial es de $\sigma = 8$, y se va actualizando en cada iteración, de modo que comienza con un valor alto y a medida que evoluciona la red va disminuyendo de la siguiente manera: $\sigma = \alpha\sigma$, con α 0,95.

Respecto al valor de σ se probaron distintos valores de inicio y distintos métodos para actualizarlo, y la combinación óptima que dió mejores resultados fue la que se mencionó.

Distribución uniforme restringida al circulo unitario

Se generaron patrones en 2 dimensiones uniformenmente distribuidos dentro de un círculo unitario. De este modo, el espacio de entrada es de 2D al igual que el espacio de los pesos de las neuronas. El código para generarlos es el siguiente:

```
modulo = sqrt(unifrnd(0, radio, p, 1));
angulo = pi * unifrnd(0, 2, p, 1);
xp = [modulo .* cos(angulo), modulo .* sin(angulo)];
```

A su vez, los pesos se inicializaron de la misma manera, generando muestras uniformemente distribuidas dentro del circulo.

Para el entrenamiento de la red, se limitó la cantidad de iteraciones a 100 dado que se observó gráficamente que mas allá de esa cantidad de iteraciones los pesos se acercan bastante a los patrones enseñados.

A continuación los resultados, se graficaron los pesos en distintos pasos de iteración uniendo los pesos de neuronas vecinas en el espacio de neuronas:

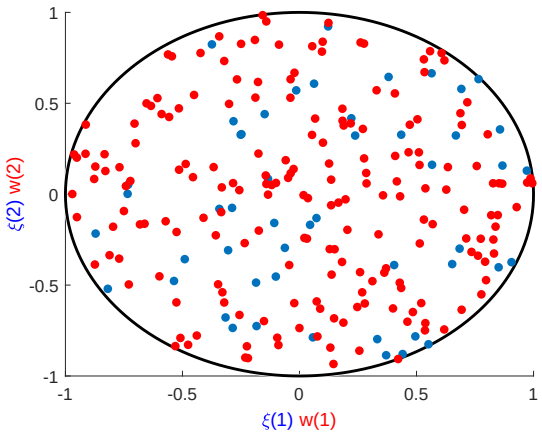

Fig. 2. Espacio de entrada. Patrones generados uniformemente dentro del circulo unitario.

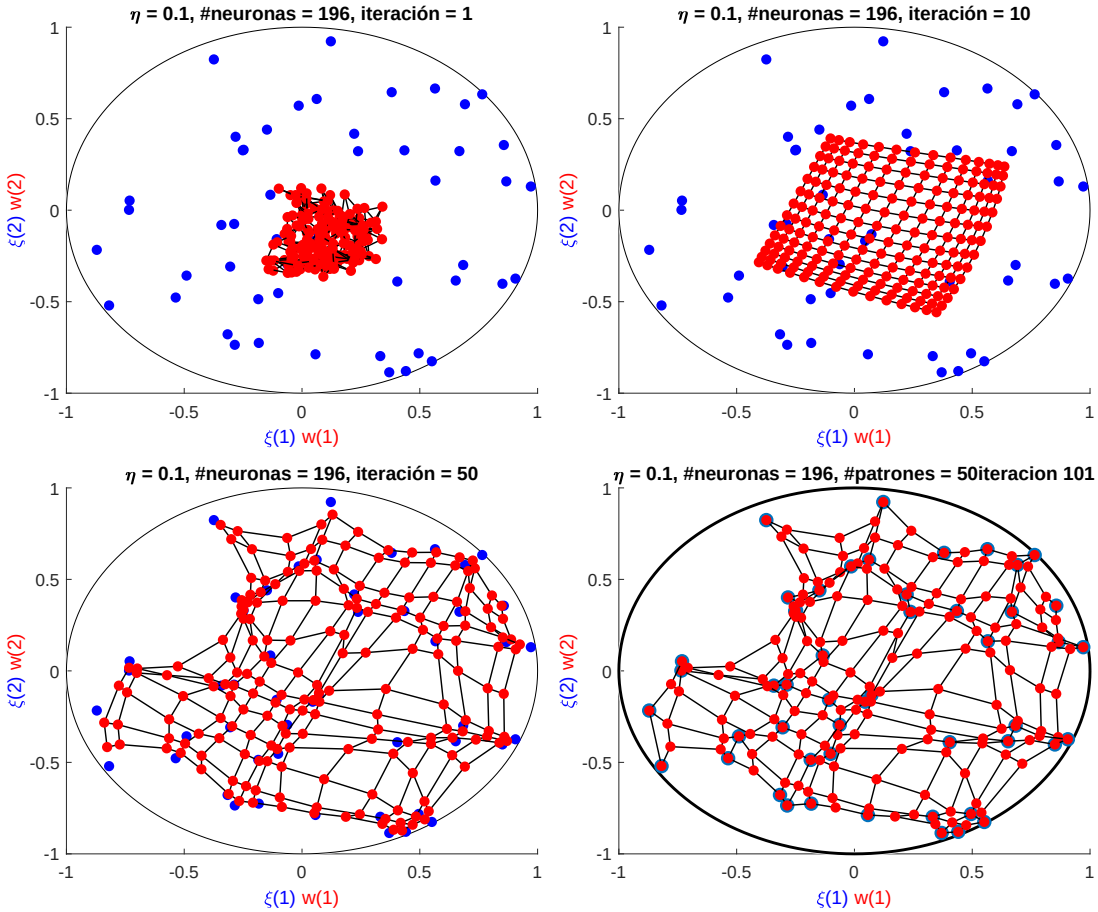

Fig. 3. Evolución de los pesos durante durante el aprendizaje.

Luego de la iteración 50 se puede ver que los pesos se distribuyen de acuerdo a los patrones de entrada y a las 100 iteraciones se terminan superponiendo con las entradas.

Distribución uniforme restringida a un rectángulo

A continuación se generan patrones de entrada restringidos a un cuadrado de lados 1. El entrenamiento de la red es el mismo que en el ejercicio anterior, pero utilizando otros patrones.

A continuación se muestra como van evolucionando los pesos a medida que avanza el entrenamiento:

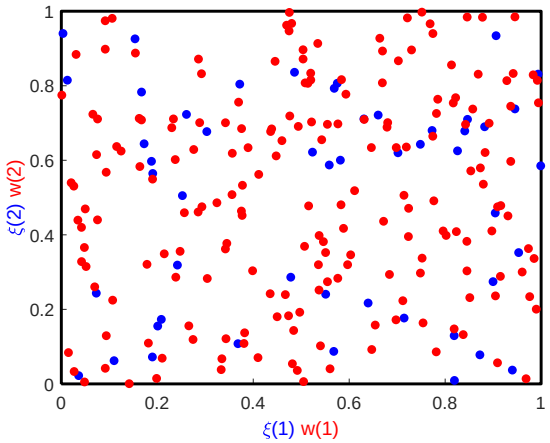

Fig. 4. Espacio de entrada. Patrones generados uniformemente dentro de un cuadrado de 1x1

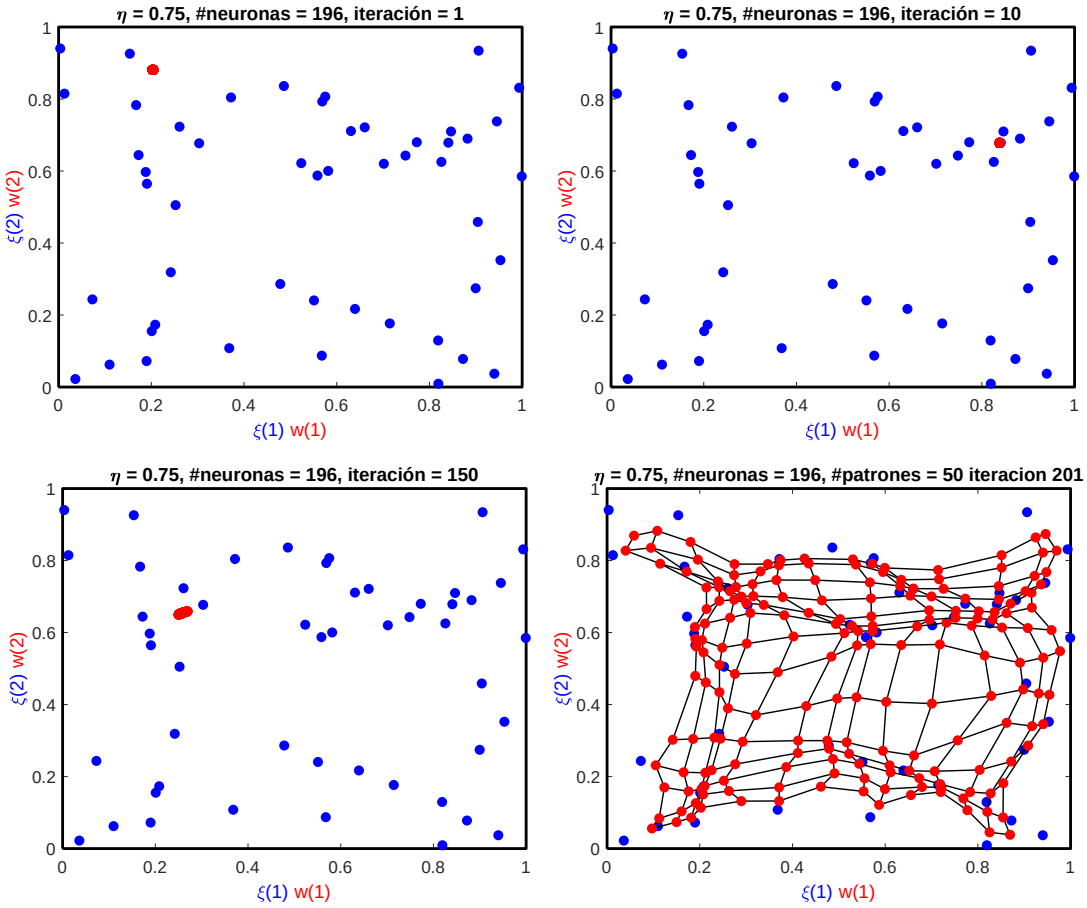

Fig. 5. Evolución de los pesos durante durante el aprendizaje.

(2) Resolución del problema del vendedor ambulante para 200 ciudades

En este ejercicio se pretende entrenar una red Kohonen que sea capaz de encontrar un camino óptimo que pase por todos los 200 puntos (ciudades) de entrada que se le enseñan.

Los patrones se generaron de la misma manera que en el problema anterior, dentro de un circulo unitario. La diferencia es que los pesos se inicializaron todos sobre el perímetro del círculo y con ángulo creciente desde 0 hasta 2π. Se decidió inicializarlos de esta manera luego de probar hacerlo de manera uniforme y al azar, y observar que los caminos resultantes que generaba la red se cruzaban, por lo tanto se procedió a usar dicho método de inicialización.

Para este problema se utilizaron 440 neuronas en un espacio de 1 dimensión, a diferencia del problema anterior que se utilizaban 2 dimensiones. Se usó el mismo método de actualización para la varianza de la función de vecindad.

Resultados

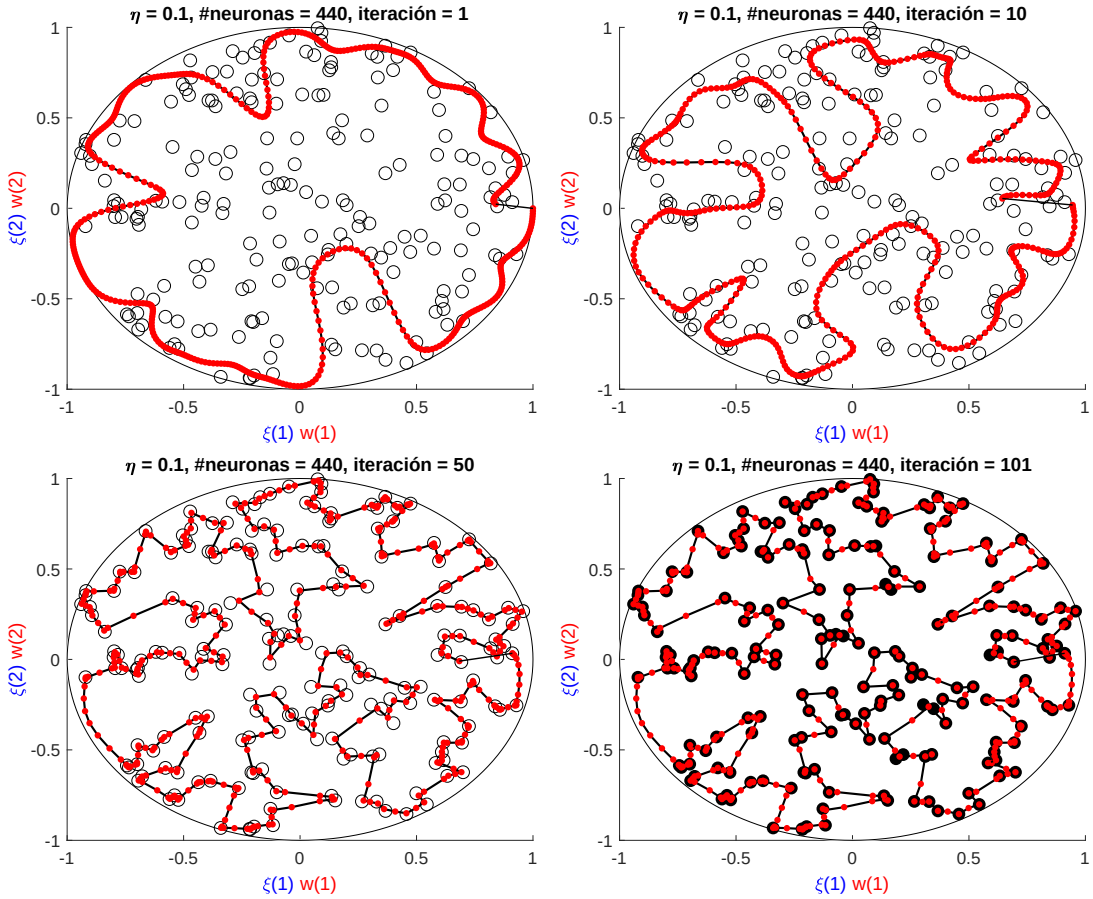

Fig. 6. Evolución de los pesos durante durante el aprendizaje. Se unen los pesos de neuronas consecutivas así como el de la última con la primera neurona.

La inicialización de los pesos de la siguiente manera:

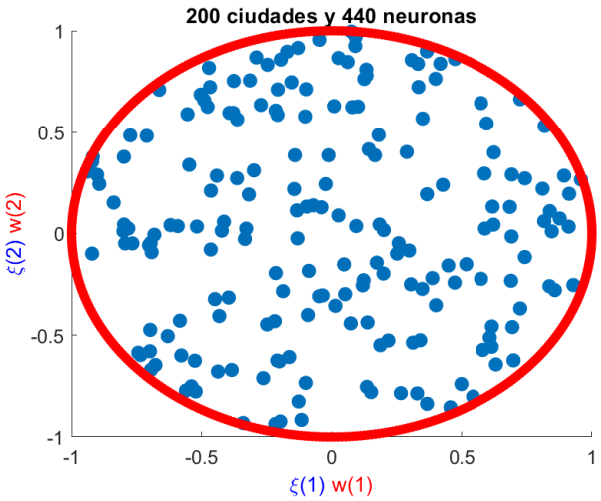

Fig. 7. Patrones generados aleatoriamente (ciudades a unir) e inicialización de los pesos sobre el perimetro del circulo.

(3) Reducción de dimensionalidad

En este ejercicio se utiliza la misma red que en el ejercicio 1, pero con la finalidad de reducir la dimensionalidad de un set de datos de 500 muestras de dimensión 100 a dimensión 2, de modo que graficamente se pueda visualizar los distintos clusters del set de datos.

A cada neurona se le asigna un valor de la función U, definida como:

$$U(i,j) = \sum_{v \in vecinos([i,j])} \|w(i,j) - w(v)\|$$

Esta función realiza la suma de todos los pesos vecinos de la neurona $[i,j]$, y si se visualiza en un mapa de colores se podrán observar las distancias de los pesos entre las neuronas, de modo que los puntos que forman un cluster son los que tienen distancias pequeñas y puntos de clusters diferentes tendrán distancias grandes.

Dado que se tienen 500 patrones de entrada, la cantidad de neuronas es de $23 \times 23 = 529$. La cantidad máxima de iteraciones es de 50.

A continuación se presentan los resultados considerando neuronas vecinas las diagonales y los laterales:

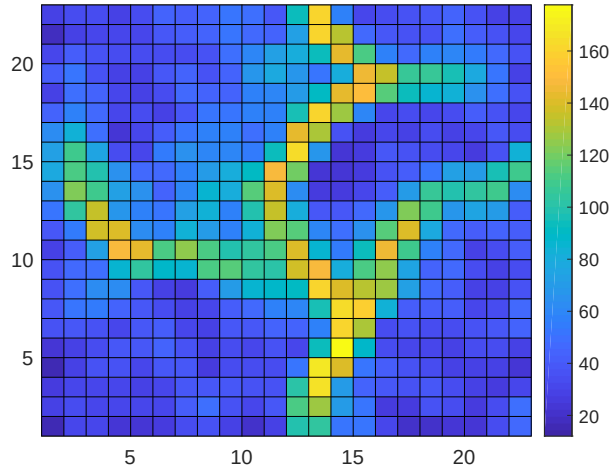

Fig. 8. Representación de la matriz $U(n)$ para cada neurona. Visualización de los clusters del set de datos.

Se corrió 2 veces y en ambas se inicializan los pesos de una manera distinta (aleatoriamente), y por ende los clusters obtenidos son distintos. Arriba se presenta uno de los resultados.

Se pueden observar 4 clusters diferentes tales que en cada uno los datos son similares entre ellos, por ende la red logró agrupar los datos y clasificarlos.

Una observación respecto a la performance del código, es que el entrenamiento de la red tuvo una duración de aproximadamente 1 hora, lo cual tiene sentido dada la cantidad de muestras de entrada que se le enseñan. Debido a esto, se corrió pocas veces y se guardó el resultado de los pesos para poder utilizarlos posteriormente y evitar volver a ejecutar el código.

Código

Ejercicio 1

```
clc; clear all; close all

%% Red Kohnonen de 2 entradas, distribucion dentro de un circulo
rng('shuffle');

% Defino los patrones de entrada, muestras uniformes restringidas a un
% circulo de radio R. x(1)^2 + x(2)^2 -R = 0
p = 50; % #patrones
radio = 1; % Radio del circulo que contiene a los patrones
N = 2; % dim. de entrada
% genero patrones
modulo = sqrt(unifrnd(0, radio, p, 1));
angulo = pi * unifrnd(0, 2, p, 1);
L = 1; B = 2;
a1 = 0;
b1 = L;
a = (b1-a1).*rand(p,1) + a1;
a2 = 0;
b2 = B;
b = (b2-a2).*rand(p,1) + a2;
%xp = [a, b];
xp = [modulo .* cos(angulo), modulo .* sin(angulo)];
patrones_plot = figure();
viscircles([0,0], radio, 'color', 'black');
hold on;
Rect = [ 0 0 ; L 0 ; L B ; 0 B ; 0 0] ;
%plot(Rect(:,1),Rect(:,2), 'color', 'black')
%scatter(xp(:, 1), xp(:, 2), 'fill');
scatter(xp(:, 1), xp(:, 2), 'fill');

% Defino las neuronas y sus pesos
dim_espacio_neuronas = 2;
n_x = 14; n_y = 14; % largo del eje x e y del espacio de coordenadas de
    neuronas.
n = n_x ^ dim_espacio_neuronas; % #neuronas
w = cell(sqrt(n), sqrt(n)); % vectores de pesos sinapticos de cada neurona.
% inicializo los pesos
for i = 1:n_x
    for j = 1:n_y
        % Vectores columna de pesos, asociados a cada neurona.
        angulo = pi * unifrnd(0, 2);
        modulo = sqrt(unifrnd(0, radio));
        w{i, j} = modulo .* [ cos(angulo) ; sin(angulo) ];
        % grafico pesos
        scatter(w{i,j}(1), w{i,j}(2), 'fill', 'r');

    end
end


set(gca,'fontsize', 12);
xlabel('\color{blue}\xi(1) \color{red}w(1)', 'interpreter', 'tex');
ylabel('\color{blue}\xi(2) \color{red}w(2)', 'interpreter', 'tex');
set(patrones_plot,'PaperSize', [20 10]); %set the paper size to what you
    want
print(patrones_plot,'Resultados/Ejercicio1/patrones','-dpdf') % then print
    it
```

```
55
56
57  % Pasos algoritmo
58  % 1. se elige un patron x al azar.
59  % 2. se calcula la distancia ese vector x y los vectores w de cada neurona.
60  % 3. obtengo la neurona ganadora, la cual tiene la menor distancia.
61  % 4. obtengo la funcion de vecindad
62
63  %
64  eta = 0.1;
65  patron_pa_graficar = randi([1, p]);
66  %sigma = [150:-1:1];
67  sigma = 8;
68  alfa = 0.95;
69
70  iteraciones = 100;
71
72  neurona_ganadora = zeros(p, 2); % coordenadas de la neurona ganadora
73  dist_min = 1000 * ones(p, 1);
74  vecindad = zeros(p, n); % una matriz donde sus filas son listas con los
        valores de la vecindad de cada neurona;
75                          % cada fila es para un patron distinto
76
77  iter = 1;
78
79
80  while(iter <= iteraciones)
81      display("Iteracion " + iter);
82
83      for mu = shuffle(1:p)
84
85          % 1. guardo el patron de entrada
86          x = xp(mu, :)';
87
88          % 2,3. calculo la distancia; encuentro las neuronas ganadoras;
89          dist = 0;
90          dist_min = 1000 * ones(p, 1);
91          for i = 1:n_x
92              for j = 1:n_y
93
94                  dist = norm(x - w{i, j});
95
96                  if(dist <= dist_min(mu))
97                      dist_min(mu) = dist;
98                      neurona_ganadora(mu, :) = [i, j];
99                  end
100
101             end
102         end
103
104         % 4,5. calculo las vecindades y actualizo los pesos
105         idx_neu = 1;
106         for i=1:n_x
107             for j=1:n_y
108                 vecindad(mu, idx_neu) = func_vecindad([i, j],
                        neurona_ganadora(mu, :), sigma);
109                 w{i,j} = w{i,j} + eta * vecindad(mu, idx_neu) * (x - w{i,j
                        });
110                 idx_neu = idx_neu + 1;
111             end
112         end
113
```

```
114
115     end
116     if(iter == 1 || iter == 10 || iter == 50)
117         plot_pesos = figure();
118         hold on;
119         viscircles([0,0], radio , 'color', 'black', 'linewidth', 0.5);
120         scatter(xp(:, 1), xp(:, 2), 50 ,'fill','blue' );
121
122         for i = 1:n_x
123             for j = 1:n_y
124                 % Vectores columna de pesos, asociados a cada neurona.
125                 % grafico pesos
126                 if j ~=  n_y
127                     plot( [w{i, j}(1), w{i, j+1}(1)], [w{i, j}(2), w{i, j
                            +1}(2)] ,'r' , 'linewidth', 1, 'color', 'black' )
128                 end
129                 if i ~= n_x
130                     plot( [w{i + 1, j}(1), w{i, j}(1)], [w{i + 1, j}(2), w{
                            i, j}(2)] ,'r' , 'linewidth', 1, 'color', 'black' )
131                 end
132                 scatter(w{i, j}(1), w{i, j}(2), 50 , 'fill', 'red');
133             end
134         end
135         % Uno el peso de la primera neurona con la ultima
136         plot_pesos.Renderer='Painters';
137
138         title("\eta = " + eta + ", #neuronas = " + n + ", iteracion = " +
                iter, 'interpreter', 'tex')
139         set(gca,'fontsize', 12);
140         xlabel('\color{blue}\xi(1) \color{red}w(1)', 'interpreter', 'tex');
141         ylabel('\color{blue}\xi(2) \color{red}w(2)', 'interpreter', 'tex');
142         set(plot_pesos,'PaperSize', [20 10]); %set the paper size to what
                you want
143         path = "Resultados/Ejercicio1/pesos" + iter;
144         print(plot_pesos,path,'-dpdf') % then print it
145
146
147     end
148     iter = iter + 1;
149     sigma = sigma * alfa;
150
151 end
152
153 plot_pesos = figure();
154 plot_pesos.Renderer='Painters';
155 title("\eta = " + eta + ", #neuronas = " + n + ", #patrones = " + p + "
        iteracion " + iter, 'interpreter', 'tex')
156 viscircles([0,0], radio, 'color', 'black');
157 hold on;
158 scatter(xp(:, 1), xp(:, 2), 100 , 'fill' );
159
160 for i = 1:n_x
161     for j = 1:n_y
162         % Vectores columna de pesos, asociados a cada neurona.
163         % grafico pesos
164         if j ~=  n_y
165             plot( [w{i, j}(1), w{i, j+1}(1)], [w{i, j}(2), w{i, j+1}(2)] ,'
                    r' , 'linewidth', 1, 'color', 'black' )
166         end
167         if i ~= n_x
168             plot( [w{i + 1, j}(1), w{i, j}(1)], [w{i + 1, j}(2), w{i, j}(2)
                    ] ,'r' , 'linewidth', 1, 'color', 'black' )
```

```
        end
        scatter(w{i, j}(1), w{i, j}(2), 50 , 'fill', 'red');
    end
end

set(gca,'fontsize', 12);
xlabel('\color{blue}\xi(1) \color{red}w(1)', 'interpreter', 'tex');
ylabel('\color{blue}\xi(2) \color{red}w(2)', 'interpreter', 'tex');
set(plot_pesos,'PaperSize', [20 10]); %set the paper size to what you want
print(plot_pesos,'Resultados/Ejercicio1/pesos','-dpdf') % then print it



%% Red Kohnonen de 2 entradas, distribucion dentro de un rectangulo
clc; clear all;
rng('shuffle');

% Defino los patrones de entrada, muestras uniformes restringidas a un
% circulo de radio R. x(1)^2 + x(2)^2 -R = 0
p = 50; % #patrones
N = 2; % dim. de entrada
% genero patrones
L = 1; B = 1;
a1 = 0;
b1 = L;
a = (b1-a1).*rand(p,1) + a1;
a2 = 0;
b2 = B;
b = (b2-a2).*rand(p,1) + a2;
xp = [a, b];
patrones_plot = figure();
hold on;
Rect = [ 0 0 ; L 0 ; L B ; 0 B ; 0 0] ;
plot(Rect(:,1),Rect(:,2), 'color', 'black','linewidth' , 2)
scatter(xp(:, 1), xp(:, 2), 'fill', 'b');

% Defino las neuronas y sus pesos
dim_espacio_neuronas = 2;
n_x = 14; n_y = 14; % largo del eje x e y del espacio de coordenadas de
    neuronas.
n = n_x ^ dim_espacio_neuronas; % #neuronas
w = cell(sqrt(n), sqrt(n)); % vectores de pesos sinapticos de cada neurona.
% inicializo los pesos
for i = 1:n_x
    for j = 1:n_y
        % Vectores columna de pesos, asociados a cada neurona.
        a = (b1-a1).*rand + a1;
        b = (b2-a2).*rand + a2;
        w{i, j} = [ a ; b ];
        scatter(w{i,j}(1), w{i,j}(2), 'fill', 'r');
    end
end


set(gca,'fontsize', 12);
xlabel('\color{blue}\xi(1) \color{red}w(1)', 'interpreter', 'tex');
ylabel('\color{blue}\xi(2) \color{red}w(2)', 'interpreter', 'tex');
set(patrones_plot,'PaperSize', [20 10]); %set the paper size to what you
    want
print(patrones_plot,'Resultados/Ejercicio1/patrones_rect','-dpdf') % then
    print it

```

```
228
229 % Pasos algoritmo
230 % 1. se elige un patron x al azar.
231 % 2. se calcula la distancia ese vector x y los vectores w de cada neurona.
232 % 3. obtengo la neurona ganadora, la cual tiene la menor distancia.
233 % 4. obtengo la funcion de vecindad
234
235 %
236 eta = 0.75;
237 patron_pa_graficar = randi([1, p]);
238 sigma = [200:-1:1];
239
240 iteraciones = size(sigma, 2);
241
242 neurona_ganadora = zeros(p, 2); % coordenadas de la neurona ganadora
243 dist_min = 1000 * ones(p, 1);
244 vecindad = zeros(p, n); % una matriz donde sus filas son listas con los
    valores de la vecindad de cada neurona;
245                         % cada fila es para un patron distinto
246
247 iter = 1;
248
249
250 while(iter <= iteraciones)
251     display("Iteracion " + iter);
252
253     for mu = shuffle(1:p)
254
255         % 1. guardo el patron de entrada
256         x = xp(mu, :)';
257
258         % 2,3. calculo la distancia; encuentro las neuronas ganadoras;
259         dist = 0;
260         dist_min = 1000 * ones(p, 1);
261         for i = 1:n_x
262             for j = 1:n_y
263
264                 dist = norm(x - w{i, j});
265
266                 if(dist <= dist_min(mu))
267                     dist_min(mu) = dist;
268                     neurona_ganadora(mu, :) = [i, j];
269                 end
270
271             end
272         end
273
274         % 4,5. calculo las vecindades y actualizo los pesos
275         idx_neu = 1;
276         for i=1:n_x
277             for j=1:n_y
278                 vecindad(mu, idx_neu) = func_vecindad([i, j],
                        neurona_ganadora(mu, :), sigma(iter));
279                 w{i,j} = w{i,j} + eta * vecindad(mu, idx_neu) * (x - w{i,j
                        });
280                 idx_neu = idx_neu + 1;
281             end
282         end
283
284
285     end
286
```

```
    if(iter == 1 || iter == 10 || iter == 50 || iter == 150)
        plot_pesos = figure();
        hold on;
        plot(Rect(:, 1), Rect(:, 2), 'color', 'black', 'linewidth', 2)
        scatter(xp(:, 1), xp(:, 2), 50 ,'fill','blue' );

        for i = 1:n_x
            for j = 1:n_y
                % Vectores columna de pesos, asociados a cada neurona.
                % grafico pesos
                if j ~=  n_y
                    plot( [w{i, j}(1), w{i, j+1}(1)], [w{i, j}(2), w{i, j
                        +1}(2)] ,'r' , 'linewidth', 1, 'color', 'black' )
                end
                if i ~= n_x
                    plot( [w{i + 1, j}(1), w{i, j}(1)], [w{i + 1, j}(2), w{
                        i, j}(2)] ,'r' , 'linewidth', 1, 'color', 'black' )
                end
                scatter(w{i, j}(1), w{i, j}(2), 50 , 'fill', 'red');
            end
        end
        % Uno el peso de la primera neurona con la ultima
        plot_pesos.Renderer='Painters';

        title("\eta = " + eta + ", #neuronas = " + n + ", iteracion = " +
            iter, 'interpreter', 'tex')
        set(gca,'fontsize', 12);
        xlabel('\color{blue}\xi(1) \color{red}w(1)', 'interpreter', 'tex');
        ylabel('\color{blue}\xi(2) \color{red}w(2)', 'interpreter', 'tex');
        set(plot_pesos,'PaperSize', [20 10]); %set the paper size to what
            you want
        path = "Resultados/Ejercicio1/pesos_rect" + iter;
        print(plot_pesos,path,'-dpdf') % then print it


    end

    iter = iter + 1;

end
%%
plot_pesos_rect = figure();

Rect = [ 0 0 ; L 0 ; L B ; 0 B ; 0 0] ;
plot(Rect(:, 1), Rect(:, 2), 'color', 'black', 'linewidth', 2)
hold on;
scatter(xp(:, 1), xp(:, 2), 50 , 'fill' , 'b');
title("\eta = " + eta + ", #neuronas = " + n + ", #patrones = " + p + "
    iteracion " + iter, 'interpreter', 'tex');

for i = 1:n_x
    for j = 1:n_y
        % Vectores columna de pesos, asociados a cada neurona.
        % grafico pesos
        if j ~=  n_y
            plot( [w{i, j}(1), w{i, j+1}(1)], [w{i, j}(2), w{i, j+1}(2)] ,'
                r' , 'linewidth', 1, 'color', 'black' )
        end
        if i ~= n_x
            plot( [w{i + 1, j}(1), w{i, j}(1)], [w{i + 1, j}(2), w{i, j}(2)
                ] ,'r' , 'linewidth', 1, 'color', 'black' )
        end
```

```
342         scatter(w{i, j}(1), w{i, j}(2), 50 , 'fill', 'red');
343     end
344 end
345
346 set(gca,'fontsize', 12);
347 xlabel('\color{blue}\xi(1) \color{red}w(1)', 'interpreter', 'tex');
348 ylabel('\color{blue}\xi(2) \color{red}w(2)', 'interpreter', 'tex');
349 set(plot_pesos_rect,'PaperSize', [20 10]); %set the paper size to what you
    want
350 print(plot_pesos_rect, 'Resultados/Ejercicio1/pesos_rect','-dpdf') % then
    print it
351
352
353 %%
354
355
356 plot_neuronas = figure();
357
358
359 hold on;
360
361 for i = 1:(n_x)
362         plot([1, n_x], [i, i] , 'linewidth', 1, 'color', 'black' )
363         plot([i, i], [1, n_y] , 'linewidth', 1, 'color', 'black' )
364 end
365 [X, Y] = meshgrid(1:n_x, 1:n_y);
366 scatter(X(:), Y(:), 100 , 'black', 'fill');
367
368 set(gca,'fontsize', 12);
369 set(plot_neuronas,'PaperSize', [20 10]); %set the paper size to what you
    want
370 print(plot_neuronas,'Resultados/Ejercicio1/neuronas','-dpdf') % then print
    it
```

Listing 1: ejercicio1

Ejercicio 2

```
1  clc; clear all; close all;
2
3  %% Problema del viajante con red neuronal Kohonen.
4
5  rng('shuffle');
6
7  % Defino los patrones de entrada, muestras uniformes restringidas a un
8  % circulo de radio R. x(1)^2 + x(2)^2 -R = 0
9  p = 200; % #patrones
10 radio = 1; % Radio del circulo que contiene a los patrones
11 N = 2; % dim. de entrada
12 % genero patrones
13 %angulo = pi * linspace(0, 2, p); angulo = reshape(angulo, [p, 1]);
14 angulo = pi * unifrnd(0, 2, p, 1);
15 modulo = sqrt(unifrnd(0, radio, p, 1));
16 xp = [modulo .* cos(angulo), modulo .* sin(angulo)];
17 patrones_plot = figure();
18 viscircles([0,0], radio, 'color', 'black');
19 hold on;
20 scatter(xp(:, 1), xp(:, 2), 80, 'fill');
21
```

```
% Defino las neuronas y sus pesos. Los pesos se inicializan sobre el perimetro del circulo y de manera creciente
% por su angulo.
dim_espacio_neuronas = 1;
n_x = round(2.2 * p); n_y = 1; % largo del eje x e y del espacio de coordenadas de neuronas.
n = n_x ^ dim_espacio_neuronas; % #neuronas
w = cell(n_x); % vectores de pesos sinapticos de cada neurona.
% inicializo los pesos
angulo = pi * linspace(0, 2, n);
for i = 1:n_x
    %for j = 1:n_y
    % Vectores columna de pesos, asociados a cada neurona.
    %angulo = pi * unifrnd(0, 2);
    modulo = sqrt(unifrnd(0, radio));
    w{i} = radio .* [ cos(angulo(i)) ; sin(angulo(i)) ];
    % grafico pesos
    scatter(w{i}(1), w{i}(2), 'fill', 'r');
    %end
end


set(gca,'fontsize', 12);
title(p + " ciudades y " + n + " neuronas");
xlabel('\color{blue}\xi(1) \color{red}w(1)', 'interpreter', 'tex');
ylabel('\color{blue}\xi(2) \color{red}w(2)', 'interpreter', 'tex');
set(patrones_plot,'PaperSize', [20 10]); %set the paper size to what you want
print(patrones_plot,'Resultados/Ejercicio2/patrones','-dpdf') % then print it

% Entreno la red

eta = 0.1;
sigma_init = 8;
sigma = sigma_init;
alfa = 0.95; % constante que decrementa la varianza

%iteraciones = size(sigma, 2);
iteraciones = 100;

neurona_ganadora = zeros(p, 2); % coordenadas de la neurona ganadora
vecindad = zeros(p, n); % una matriz donde sus filas son listas con los valores de la vecindad de cada neurona;
                        % cada fila es para un patron distinto

iter = 1;


while(iter <= iteraciones)
    display("Iteracion: " + iter);
    for mu = shuffle(1:p)

        % 1. guardo el patron de entrada
        x = xp(mu, :)';

        % 2,3. calculo la distancia; encuentro las neuronas ganadoras;
        dist = 0;
        dist_min = 1000 * ones(p, 1);
        for i = 1:n_x


```

```
            dist = norm(x - w{i});

            if(dist <= dist_min(mu))
                dist_min(mu) = dist;
                neurona_ganadora(mu, :) = i;
            end


        end

        % 4,5. calculo las vecindades y actualizo los pesos
        idx_neu = 1;
        for i=1:n_x

            vecindad(mu, idx_neu) = func_vecindad(i, neurona_ganadora(mu,
                :), sigma);
            w{i} = w{i} + eta * vecindad(mu, idx_neu) * (x - w{i});
            idx_neu = idx_neu + 1;

        end


    end

    % Grafico la evolucion de los pesos

    if(iter == 1 || iter == 10 || iter == 50)
        plot_ciudades = figure();
        hold on;
        viscircles([0,0], radio , 'color', 'black', 'linewidth', 0.5);
        scatter(xp(:, 1), xp(:, 2), 80 ,'black' );

        for i = 1:n_x

            % Vectores columna de pesos, asociados a cada neurona.
            % grafico pesos

            if i ~= n_x
                plot( [w{i + 1}(1), w{i}(1)], [w{i + 1}(2), w{i}(2)] ,'r' ,
                    'linewidth', 1.5, 'color', 'black' )
            end
            scatter(w{i}(1), w{i}(2), 20 , 'fill', 'red');

        end
        % Uno el peso de la primera neurona con la ultima
        plot([w{1}(1), w{n}(1)], [w{1}(2), w{n}(2)], 'r', 'linewidth', 1, '
            color', 'black');
        plot_ciudades.Renderer='Painters';

        title("\eta = " + eta + ", #neuronas = " + n + ", iteracion = " +
            iter, 'interpreter', 'tex')
        set(gca,'fontsize', 12);
        xlabel('\color{blue}\xi(1) \color{red}w(1)', 'interpreter', 'tex');
        ylabel('\color{blue}\xi(2) \color{red}w(2)', 'interpreter', 'tex');
        set(plot_ciudades,'PaperSize', [20 10]); %set the paper size to
            what you want
        path = "Resultados/Ejercicio2/recorrido" + iter;
        print(plot_ciudades,path,'-dpdf') % then print it


    end

```

```
136     iter = iter + 1;
137
138     % Actualizo el ancho de la vecindad
139     sigma = sigma * alfa;
140
141 end
142
143 %%
144 plot_ciudades = figure();
145
146 hold on;
147
148 viscircles([0,0], radio , 'color', 'black', 'linewidth', 0.5);
149 scatter(xp(:, 1), xp(:, 2), 80 , 'fill' ,'black' );
150
151 for i = 1:n_x
152
153     % Vectores columna de pesos, asociados a cada neurona.
154     % grafico pesos
155
156     if i ~= n_x
157         plot( [w{i + 1}(1), w{i}(1)], [w{i + 1}(2), w{i}(2)] ,'r' , '
            linewidth', 1.5, 'color', 'black' )
158     end
159     scatter(w{i}(1), w{i}(2), 20 , 'fill', 'red');
160
161 end
162 % Uno el peso de la primera neurona con la ultima
163 plot([w{1}(1), w{n}(1)], [w{1}(2), w{n}(2)], 'r', 'linewidth', 1, 'color',
    'black');
164 plot_ciudades.Renderer='Painters';
165
166 title("\eta = " + eta + ", #neuronas = " + n + ", iteracion = " + iter, '
    interpreter', 'tex')
167 set(gca,'fontsize', 12);
168 xlabel('\color{blue}\xi(1) \color{red}w(1)', 'interpreter', 'tex');
169 ylabel('\color{blue}\xi(2) \color{red}w(2)', 'interpreter', 'tex');
170 set(plot_ciudades,'PaperSize', [20 10]); %set the paper size to what you
    want
171 print(plot_ciudades,'Resultados/Ejercicio2/recorrido','-dpdf') % then print
     it
172
173
174
175 plot_neuronas = figure();
176 hold on;
177 plot([1, n], [0, 0], 'black')
178 [X, Y] = meshgrid(1:n_x, 1:n_y);
179 scatter(1:1:n ,zeros(n,1), 100 , 'black', 'fill');
180 xlim([1, n]);
181 xlabel("Neuronas");
182 set(gca,'ytick',[]); % 1D
183 set(gca,'fontsize', 12);
184 set(plot_neuronas,'PaperSize', [20 10]); %set the paper size to what you
    want
185 print(plot_neuronas,'Resultados/Ejercicio1/neuronas_rect','-dpdf') % then
    print it
```

Listing 2: ejercicio2

Ejercicio 3

```
clc; clear all; close all;

%% Reducir muestras de 100 dimensiones a 2D
rng('shuffle');


% Cargo los patrones de muestra
datos = cell2mat(struct2cell(load("datos_para_clustering.mat")));

% Parametros
dim_espacio_entrada = size(datos, 2);
dim_espacio_neuronas = 2;

p = size(datos, 1); % #patrones
xp = datos; % patrones

% Obtengo la muestra que tiene norma maxima.
norms = zeros(1, p);
for mu = 1:p
    norms(mu) = norm(datos(mu, :));
end
radio = max(norms);

n_x = 23; n_y = 23; % largo del eje x e y del espacio de coordenadas de
    neuronas.
n = n_x ^ dim_espacio_neuronas; % #neuronas

eta = 0.1;
sigma = 8; % varianza inicial
alfa = 0.95; % cte. de actualizacion de la varianza
iteraciones = 50; % iteraciones maximas
iter = 1;


% inicializo los pesos
w = cell(sqrt(n), sqrt(n)); % vectores de pesos sinapticos de cada neurona.

for i = 1:n_x
    for j = 1:n_y
        % Vectores columna de pesos, asociados a cada neurona.
        w{i, j} = rand(1, dim_espacio_entrada);
        w{i, j} = radio .* w{i, j} ./ norm(w{i, j});
    end
end

%% Entrenamiento de la red

% Entrenamiento de la red tarda como 2 horas! :O


neurona_ganadora = zeros(p, 2); % coordenadas de la neurona ganadora
dist_min = 1000 * ones(p, 1);
vecindad = zeros(p, n); % una matriz donde sus filas son listas con los
    valores de la vecindad de cada neurona;
                        % cada fila es para un patron distinto

while(iter <= iteraciones)
    display("Iteracion " + iter);
    for mu = shuffle(1:p)

```

```
59          % 1. guardo el patron de entrada
60          x = xp(mu, :)';
61          display("patron "+mu);
62          % 2,3. calculo la distancia; encuentro las neuronas ganadoras;
63          dist = 0;
64          dist_min = 1000 * ones(p, 1);
65          for i = 1:n_x
66              for j = 1:n_y
67
68                  dist = norm(x - w{i, j});
69
70                  if(dist <= dist_min(mu))
71                      dist_min(mu) = dist;
72                      neurona_ganadora(mu, :) = [i, j];
73                  end
74
75              end
76          end
77
78          % 4,5. calculo las vecindades y actualizo los pesos
79          idx_neu = 1;
80          for i=1:n_x
81              for j=1:n_y
82                  vecindad(mu, idx_neu) = func_vecindad([i, j],
                        neurona_ganadora(mu, :), sigma);
83                  w{i,j} = w{i,j} + eta * vecindad(mu, idx_neu) * (x' - w{i,j
                        });
84                  idx_neu = idx_neu + 1;
85              end
86          end
87
88
89      end
90
91      if(iter == 90 || iter == 50 || iter == 120 || iter == 140)
92
93      end
94
95      iter = iter + 1;
96
97      sigma = sigma * alfa;
98  end
99
100 %% Armado de la funcion U
101
102 w = struct2cell(load("w1.mat")); w= w{1}; % pesos de dimension 100!!!
103 w_mat = zeros(n_x, n_y); % matriz auxiliar que mapea las coordenadas con la
        matriz w de cell arrays.
104
105 for i = 1:n_x
106  % workaround: en la linea de actualizacion de los pesos estaba haciendo (x
107 % - w{i,j}) y como x es un vector columna y w un vector fila, esa resta
108 % termina dando una matriz donde en cada columna esta el resultado
        replicado. El fix seria hacer (x'-w{i,j}), pero para los w
109 % que me guarde me quedo con los vector columna.
110
111     for j = 1:n_y
112         w{i,j} = w{i,j}(:,1);
113     end
114 end
115 s = size(w);
116 B = zeros(s);
```

```
nn = numel(w);
matriz_vecinos = cell(s);

for ii = 1:nn
  B(ii) = 1;
  matriz_vecinos{ii} = w(bwdist(B,'ch') == 1);
  B(ii) = 0;
end

% Construyo la funcion U:
U = zeros(n_x, n_y); % matriz de la funcion U

for i = 1:n_x
    for j = 1:n_y

        % para la neurona [i,j]

        suma = 0; % inicializo la suma en 0 para luego guardarla en la
            matriz U.

        % recorro todos los vecinos, calculo la norma, y lo sumo en $suma.
        for vecinos = matriz_vecinos{i,j}
            for vecino_idx = 1:size(vecinos, 1)
                suma = suma + norm(w{i, j} - cell2mat(vecinos(vecino_idx)))
                    ;
            end
        end

        U(i, j) = suma;
    end
end


% Grafico la matriz U

plot_clusters = figure();
pcolor(U);
colorbar;
set(gca,'fontsize', 12);
set(plot_clusters,'PaperSize', [20 10]);
print(plot_clusters,'Resultados/Ejercicio3/clusters_1','-dpdf')
```

Listing 3: ejercicio3

Funciones auxiliares

```
function y = func_vecindad(neurona_i, neurona_ganadora, sigma)
    % recibe pares de coordenadas
    y = exp(  -norm(neurona_i - neurona_ganadora)^2 / (2 * sigma ^ 2) ) ;
end
```

Función que evalúa la función de vecindad entre una neurona y la ganadora.