



FACULTAD DE INGENIERIA

Universidad de Buenos Aires

86.54/66.63 - Redes Neuronales

GUÍA DE EJERCICIOS N°3

Rossi, Francisco 99540 frrossi@fi.uba.ar

16 de Febrero de 2021

Índice

1. Ejercicio 1: Implementación de una Red de Kohonen con formas geométricas dadas por distribuciones uniformes	2
1.1. Enunciado:	2
1.2. Implementación	2
1.2.a. Circulo Unitario	2
1.2.b. Cuadrado	5
1.2.c. Clusters	7
2. Ejercicio 2: Implementación de una Red de Kohonen para resolver el <i>Traveling salesman problem</i>	9
2.1. Enunciado:	9
2.2. Implementación	9
2.2.a. 10x10	10
2.2.b. 20x10	11
2.2.c. 200x200	12
2.2.d. 300x200	13
2.3. Implementación con neuronas como circulo	14
2.3.a. 20x10	14
2.3.b. 300x200	15

1. Ejercicio 1: Implementación de una Red de Kohonen con formas geométricas dadas por distribuciones uniformes

1.1. Enunciado:

Hacer una red de Kohonen de 2 entradas que aprenda una distribución uniforme dentro del círculo unitario. Mostrar el mapa de preservación de topología. Probar con distribuciones uniformes dentro de otras figuras geométricas.

1.2. Implementación

En este ejercicio se generaron diferentes patrones a partir de distribuciones uniformes, se realizaron tres figuras diferentes: círculo unitario, cuadrado y dos clusters. Una vez generados los N_p patrones se define la cantidad de neuronas a utilizar N_{neu} , vale la pena aclarar que estas deben ser tales de poder ordenarse en matrices de $\sqrt{N_{neu}} \times \sqrt{N_{neu}}$. Además se elige un vector de anchos de vecindad σ_v , por cada valor de este vector se recorrerán todos los patrones para entrenar a la red. Estos últimos se recorrerán todos en orden aleatorio en cada iteración. El algoritmo en cada iteración es:

- Elijo un patrón al azar
- Se elige el pesos sináptico W_i asociado a la neurona i más cercano al patrón elegido.
- Se computa la función vecindad para cada neurona, para cuál es necesario calcular previamente la distancia entre esa neurona y la ganadora
- Se actualizan los pesos sinápticos

Aclaraciones:

Para todas las distancias se utiliza la distancia euclídea, $dist(x, y) = ||\vec{x} - \vec{y}||$

$$\text{Función vecindad: } \Lambda(i, i^*) = e^{-\frac{\|r_i - r_{i^*}\|^2}{2\sigma_v^2}}$$

$$\text{Función delta: } \Delta w_{ij} = \eta \Lambda(i, i^*) (\xi_j - w_{ij})$$

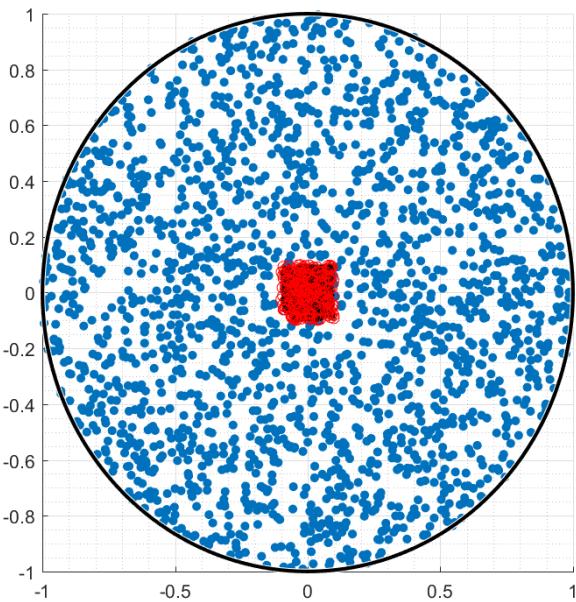
Luego una vez recorridos todos los patrones se procede a calcular C, de manera similar al algoritmo anterior, pero en vez de actualizar los pesos sinápticos se calcula C. Esto debe hacerse luego de actualizar todos los pesos, por esto son dos procesos separados.

1.2.a. Círculo Unitario

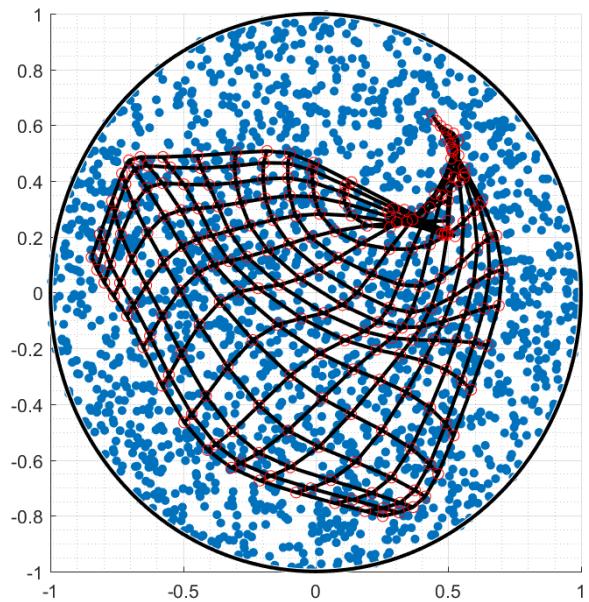
En este caso los patrones fueron generados de manera uniforme en el círculo unitario. En la **Fig. 1** se pueden ver cuatro figuras que representan la evolución de la red en cada iteración, la totalidad en detalle de los estados puede verse en el siguiente [repositorio](#).

La constante de aprendizaje es de $\eta = 0,4$ se fue modificando, cuanto más chica más lento se acomoda la red, sin embargo, si superamos cierto umbral algunos pesos se iban fuera del círculo unitario.

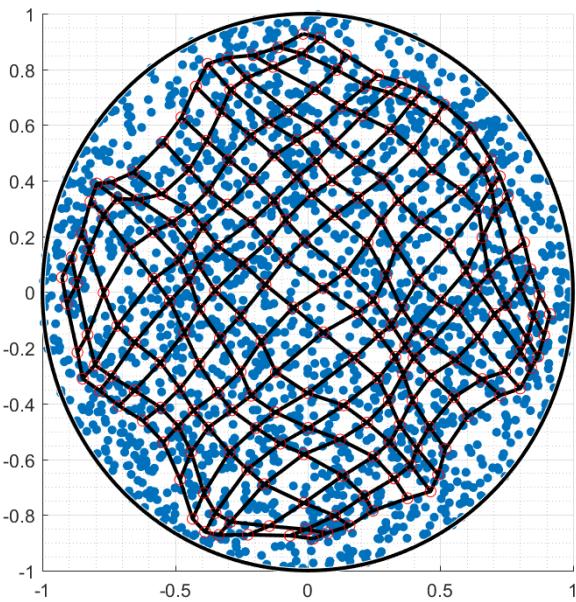
La cantidad de neuronas utilizadas fue de $N_{neu} = 15 \times 15$ y la cantidad de patrones $N_p = 2000$. El vector de ancho de vecindad recorre de 2 a 0,2 de a pasos de 0,1, si se empieza desde un valor mas alto se puede ver como la red es muy fluctuante en cuanto a su forma, cuanto menor es σ_v se puede ver como se empieza a acomodar de manera mas detallada a los patrones, se utilizaron los mismos parámetros para las tres figuras geométricas.



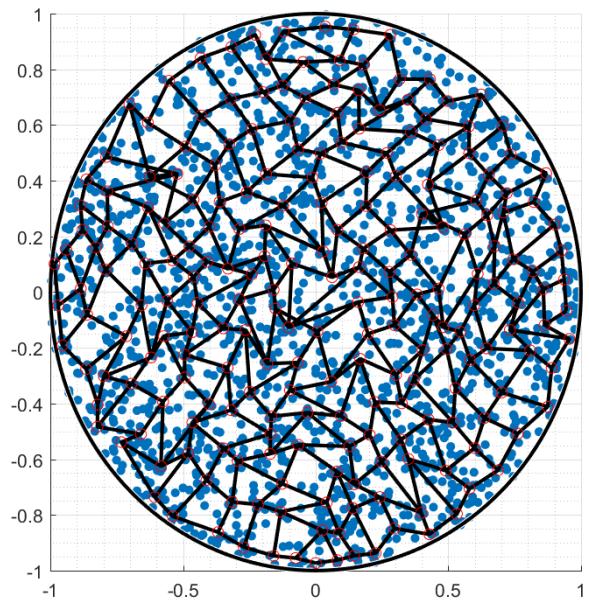
(a) Estado inicial



(b) Primer Iteración



(c) Iteración 11



(d) Estado final

Figura 1: Evolución de la red para el círculo unitario

Donde vemos que la red se acomoda a los patrones distribuidos de manera uniforme. Al no ser una red cuyo aprendizaje es supervisado, podemos medir el parámetro C que corresponde a una ponderación global de las distancias de los pesos sinápticos a los patrones de referencia, ponderados por la función vecindad.

$$C = \frac{1}{2} \sum_{i\mu} \Lambda(i, i^*) \| \xi^\mu - w_i \|^2$$

En la **Fig. 2** se observa el parámetro C en función de las iteraciones.

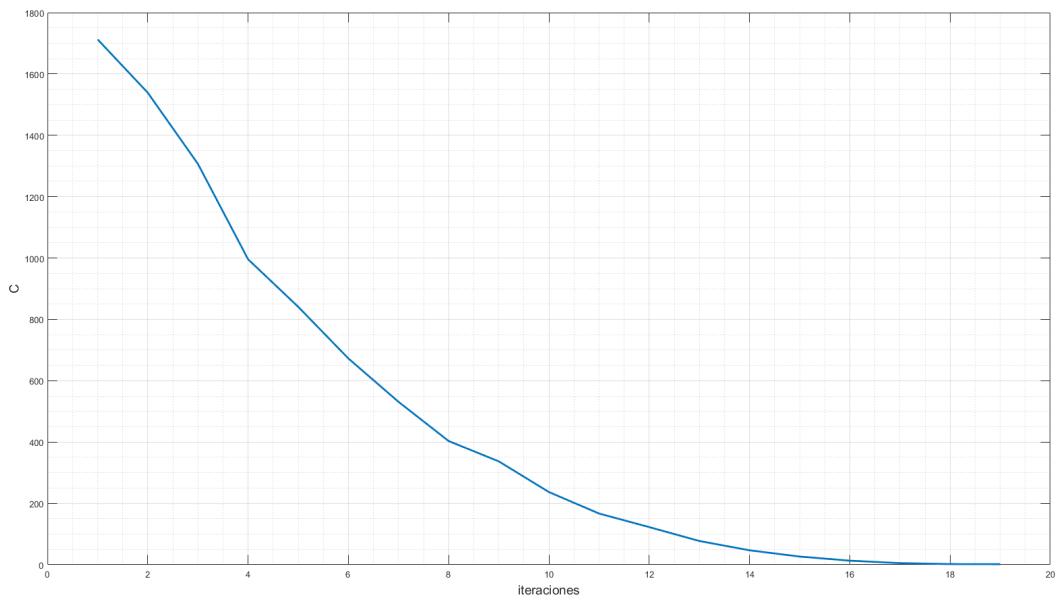
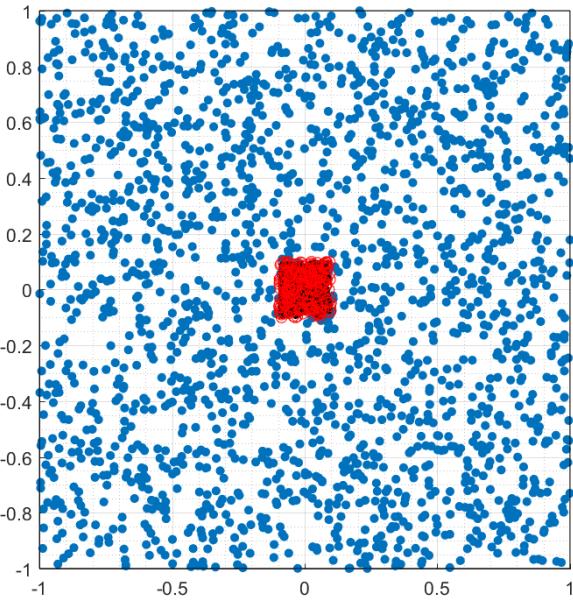


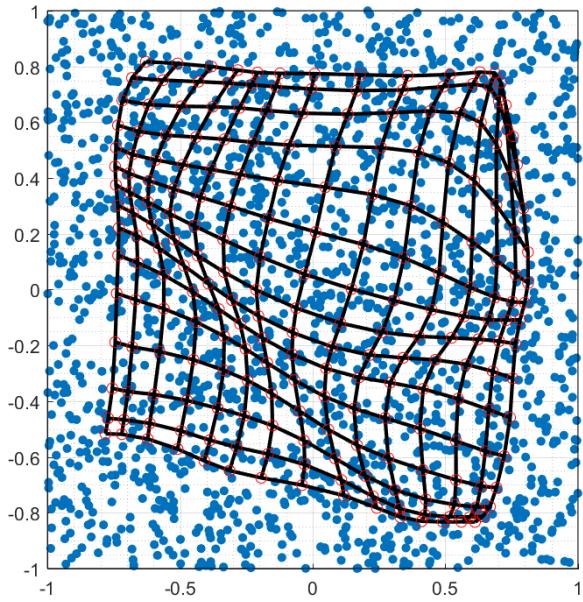
Figura 2: C en función de las iteraciones para el circulo unitario

1.2.b. Cuadrado

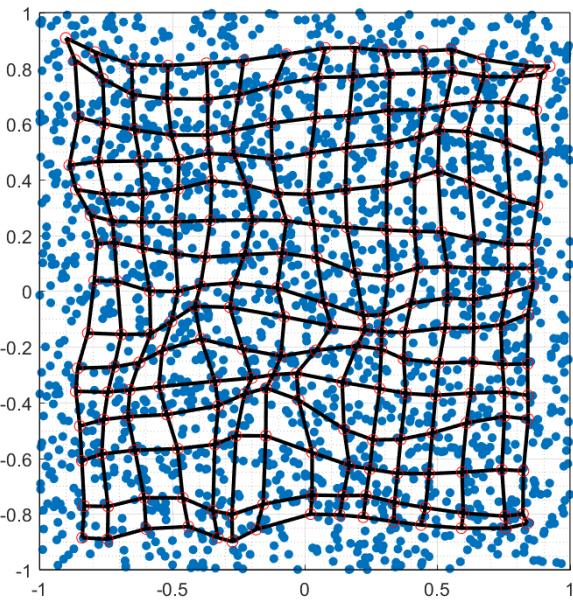
En este caso los patrones fueron generados de manera uniforme en el cuadrado $[-1, -1] \rightarrow [-1, 1] \rightarrow [1, 1] \rightarrow [1, -1]$. En la **Fig. 3** se pueden ver cuatro figuras que representan la evolución de la red en cada iteración, la totalidad en detalle de los estados puede verse en el siguiente [repositorio](#).



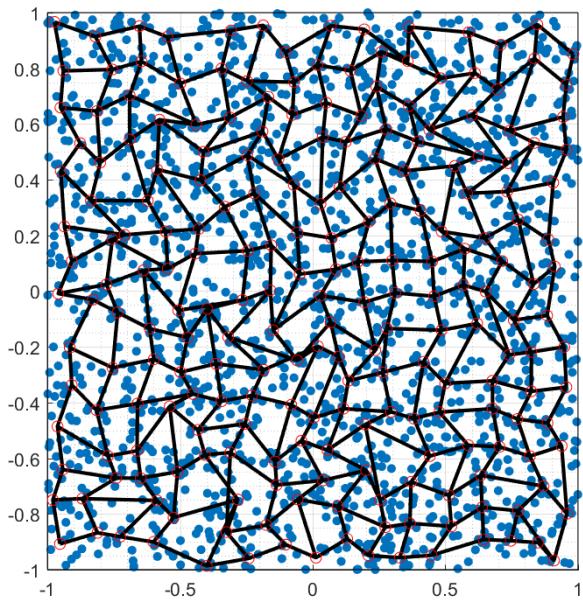
(a) Estado inicial



(b) Primer Iteración



(c) Iteración 11



(d) Estado final

Figura 3: Evolución de la red para el cuadrado

En la **Fig. 4** se observa el parámetro C en función de las iteraciones.

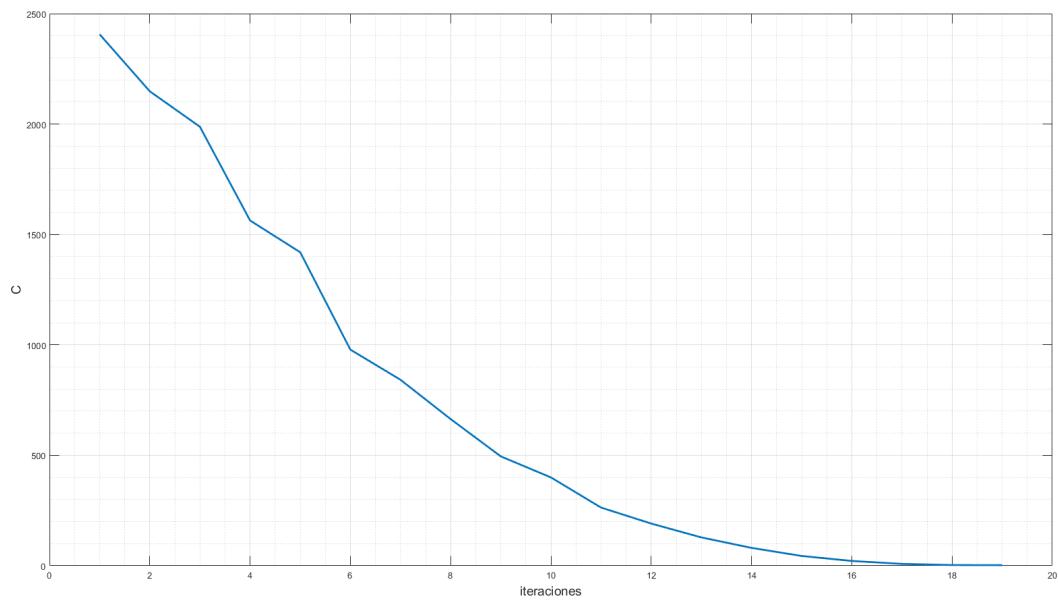
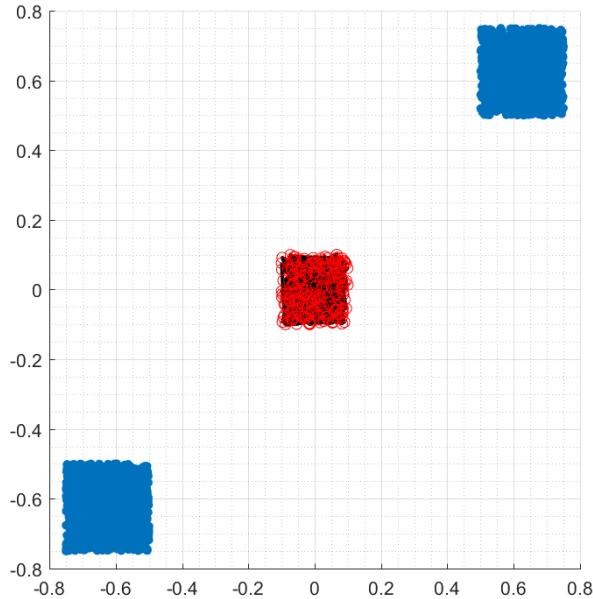


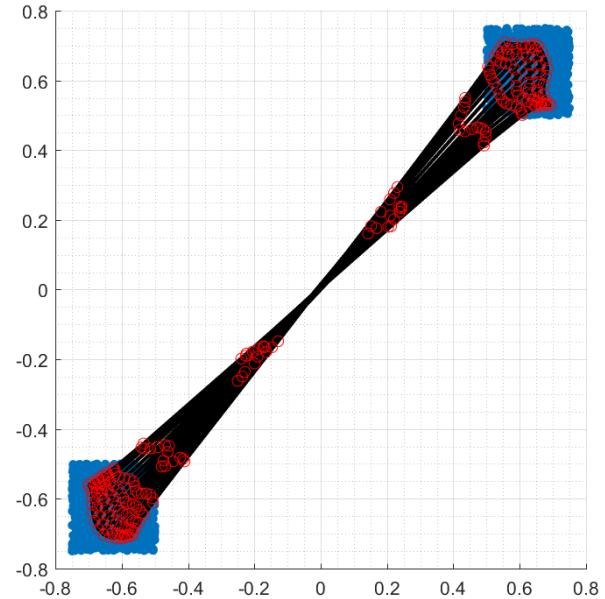
Figura 4: C en función de las iteraciones para el cuadrado

1.2.c. Clusters

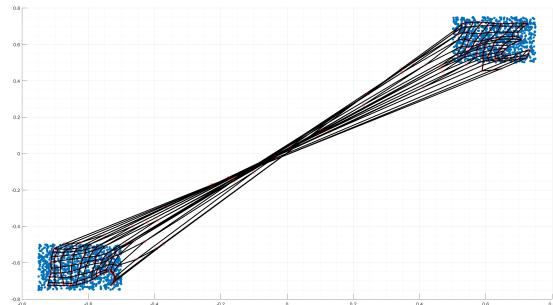
En este caso los patrones fueron generados de manera uniforme en dos clusters. En la **Fig. 5** se pueden ver cuatro figuras que representan la evolución de la red en cada iteración, la totalidad en detalle de los estados puede verse en el siguiente [repositorio](#).



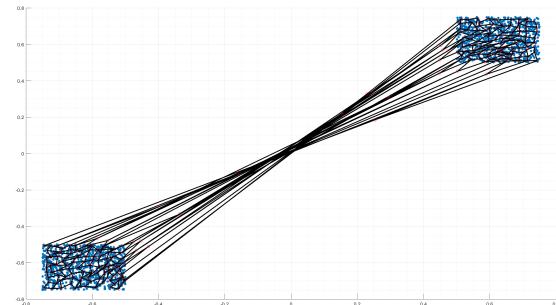
(a) Estado inicial



(b) Primer Iteración



(c) Iteración 11



(d) Estado final

Figura 5: Evolución de la red para el caso de dos clusters

En la **Fig. 6** se observa el parámetro C en función de las iteraciones.

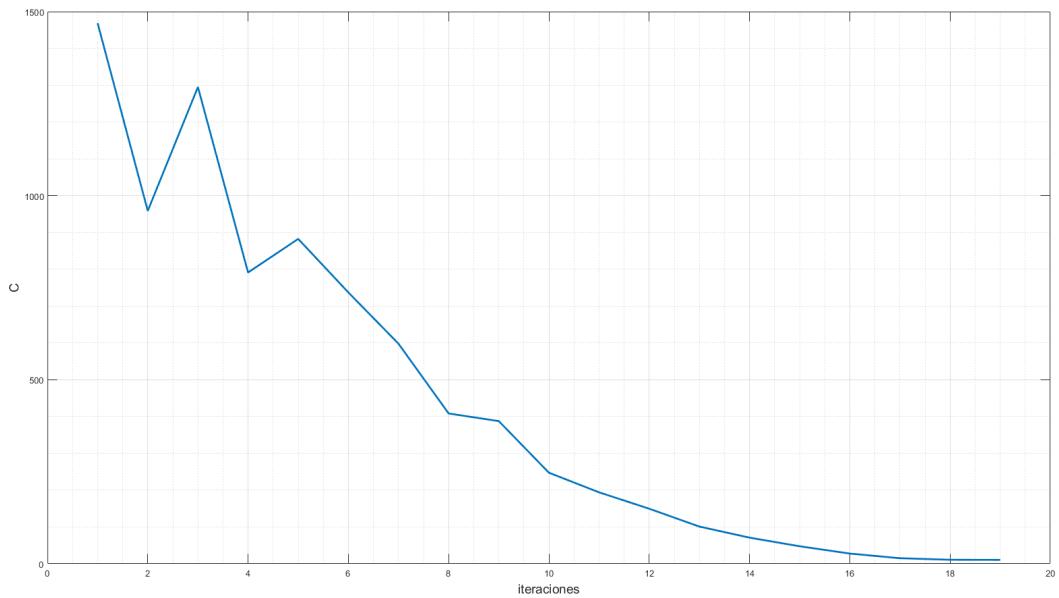


Figura 6: C en función de las iteraciones para el caso de los dos clusters

En este último caso vemos que C no es monótonamente decreciente, podemos imaginar que esto es debido a que algunos pesos cuando el ancho de la vecindad es suficientemente grande, pasan de un cluster a otro, generando estos saltos.

2. Ejercicio 2: Implementación de una Red de Kohonen para resolver el *Traveling salesman problem*

2.1. Enunciado:

Resolver (aproximadamente) el “Traveling salesman problem” para 200 ciudades con una red de Kohonen.

2.2. Implementación

En este ejercicio, se busca aproximar el camino óptimo que debe recorrer entre diferentes ciudades para recorrer la menor distancia posible.

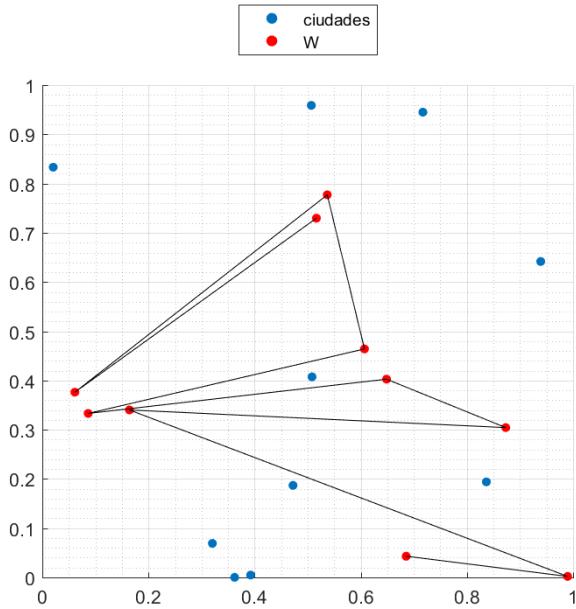
En este caso, las posiciones de las ciudades son distribuidas al azar, de manera que cada posición se distribuye como $P \sim [\mathcal{U}(0, 1), \mathcal{U}(0, 1)]$. Luego, queremos utilizar una red de Kohonen para resolver el problema, sabemos que los pesos sinápticos tienden a las posiciones de las ciudades y a la vez, que cada par de pesos sináptico corresponde a una neurona y que a pesos sinápticos cercanos, corresponden neuronas cercanas. En el ejercicio anterior utilizamos una disposición 2-D para ordenar las neuronas, pero en este caso, debido a que queremos movernos de una en una para así siempre ir a la ciudad más cercana, representada por el peso sináptico correspondiente a la siguiente neurona.

Una vez entendido como resolverlo con la red de Kohonen, el procedimiento es idéntico al realizado en el ejercicio anterior, solo que no debemos pasar de índice 1-D a índice 2-D cuando se encuentra la neurona ganadora.

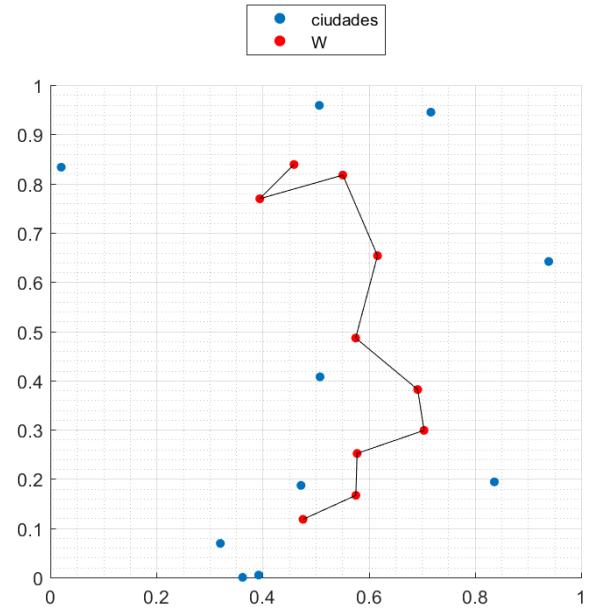
Ahora, cuantas neuronas elegimos para resolver este problema, dada la naturaleza de la solución que se propuso es evidente que como mínimo debemos tener la misma cantidad de neuronas que de ciudades, pues “mapeamos” un peso a una ciudad. Vamos a probar esto con pocas ciudades para empezar y veremos como avanzar para poder resolver el problema con 200 ciudades.

2.2.a. 10x10

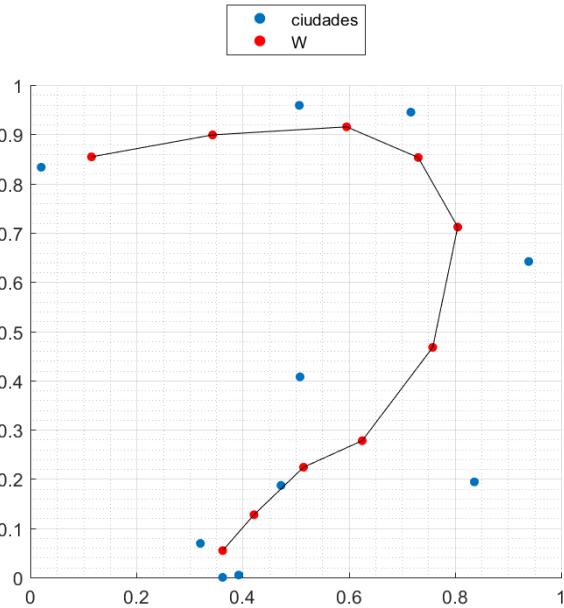
Se elige utilizar 10 neuronas para resolver el *traveling salesman problem* de 10 ciudades. Se obtiene la aproximación como se muestra en la **Fig. 7**



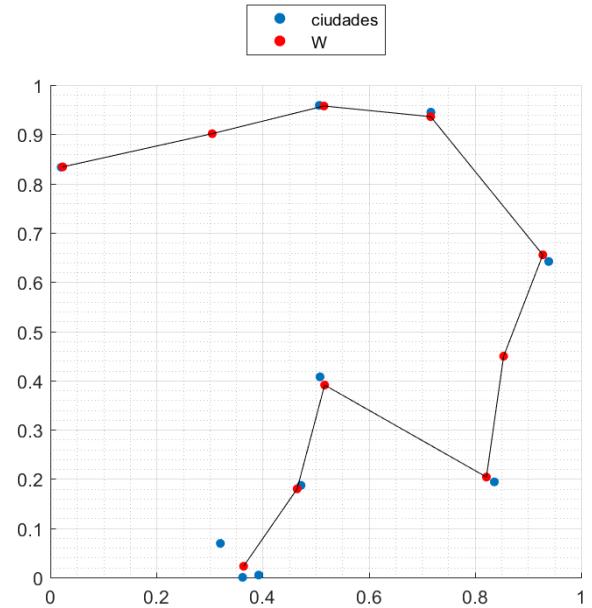
(a) Estado inicial



(b) Primer Iteración



(c) Iteración 11



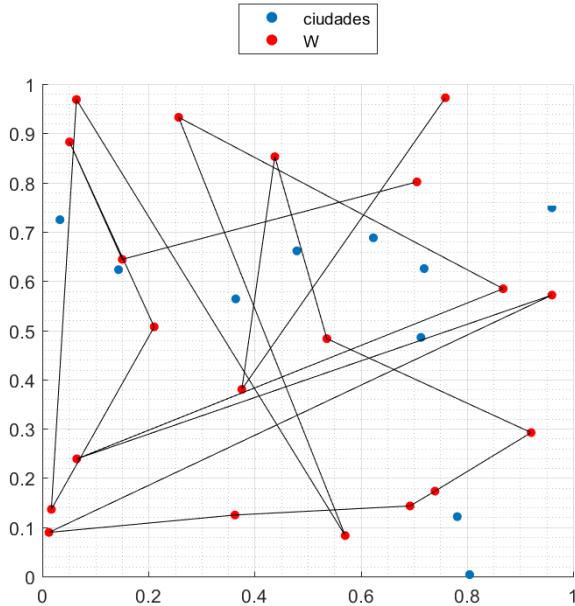
(d) Estado final

Figura 7: Evolución de la red para el caso 10x10

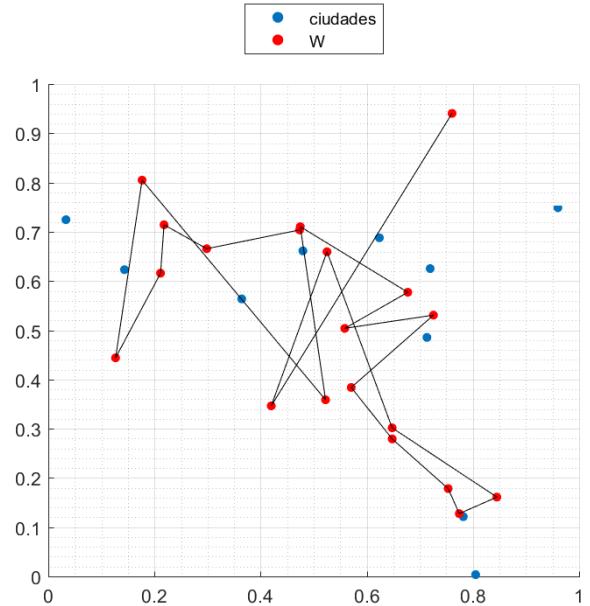
Podemos observar que si bien se aproxima, muchas ciudades quedan sin su equivalente del peso sináptico, esto podemos solucionarlo agregando más neuronas, probemos con 20 neuronas y 10 ciudades.

2.2.b. 20x10

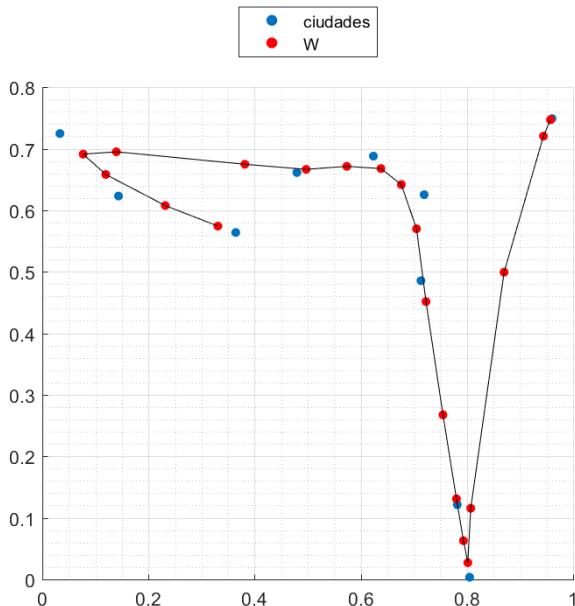
En este caso se obtiene la aproximación como se muestra en la **Fig. 8**.



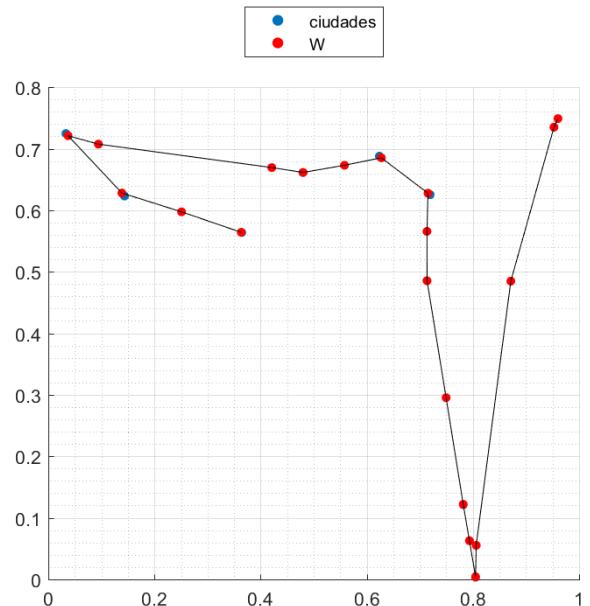
(a) Estado inicial



(b) Primer Iteración



(c) Iteración 11



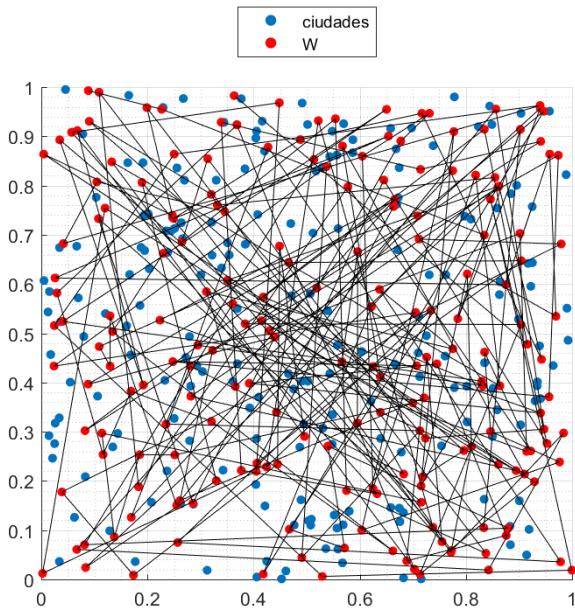
(d) Estado final

Figura 8: Evolución de la red para el caso 20x10

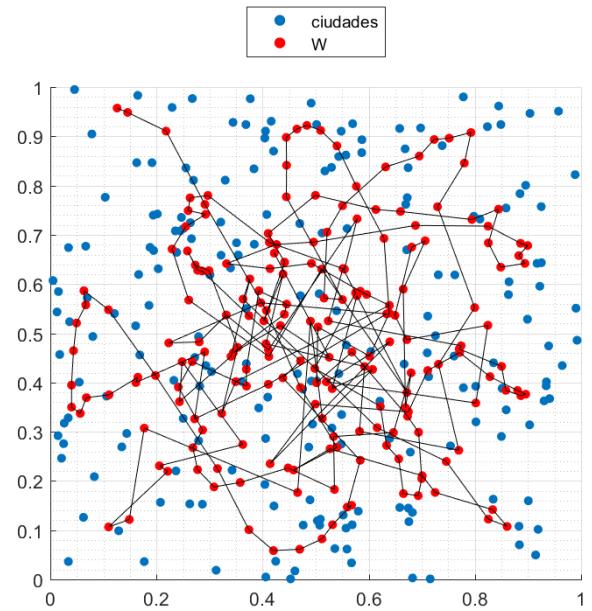
Donde vemos que el problema del caso anterior es solucionado, cada ciudad esta asociada a un peso, y otros pesos completan el camino.

2.2.c. 200x200

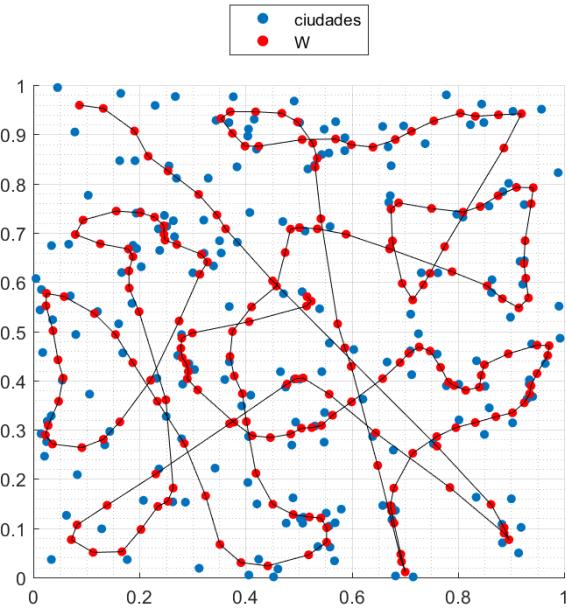
Ahora que conocemos la importancia de la cantidad de neuronas veamos que sucede con el caso de 200 ciudades, se obtiene la aproximación como se muestra en la **Fig. 9**



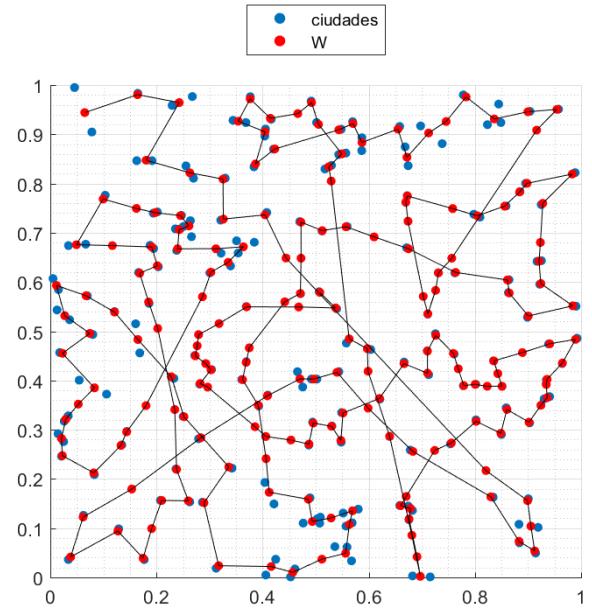
(a) Estado inicial



(b) Primer Iteración



(c) Iteración 11



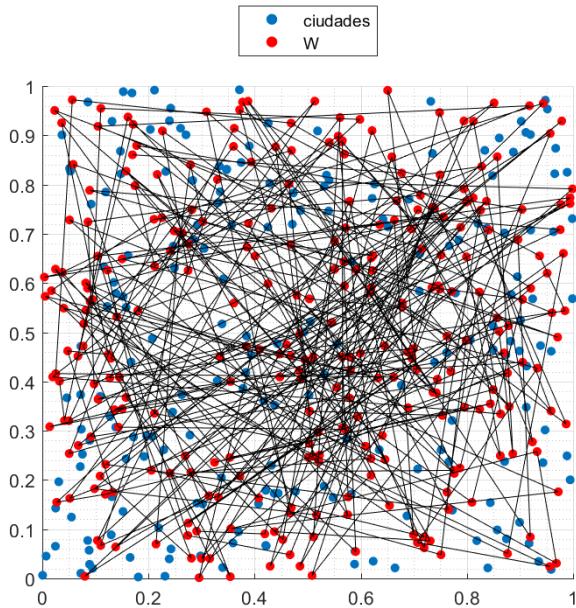
(d) Estado final

Figura 9: Evolución de la red para el caso 200x200

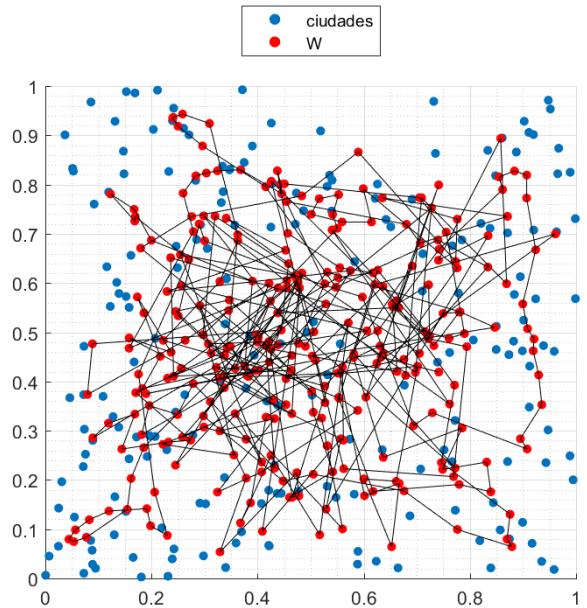
Donde es evidente el mismo problema que en el caso de 10x10.

2.2.d. 300x200

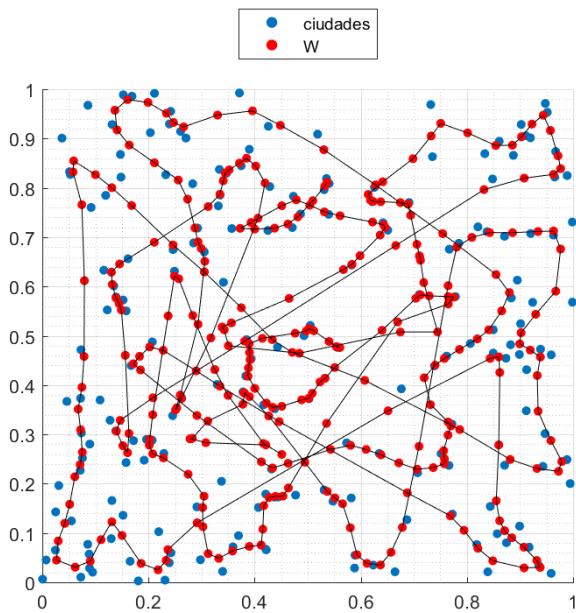
Para esto probamos con 300 neuronas obteniendo la aproximación como se muestra en la **Fig. 10**



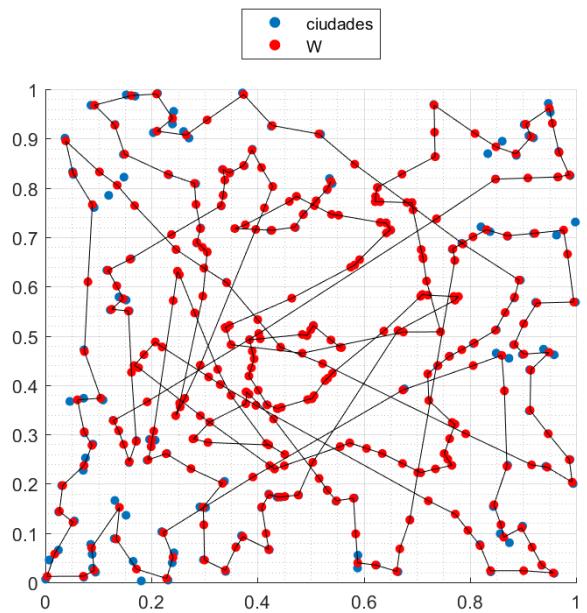
(a) Estado inicial



(b) Primer Iteración



(c) Iteración 11



(d) Estado final

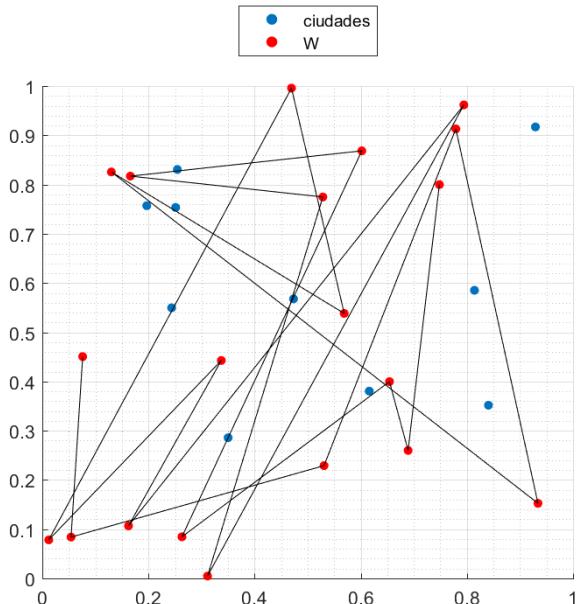
Figura 10: Evolución de la red para el caso 300x200

Donde vemos que ahora si es una buena aproximación de la solución del problema. Sin embargo, la primera y la ultima ciudad no están cercanas, esto es debido a que la topología de las neuronas hasta ahora es una linea recta, en cambio como el objetivo del problema es volver a la ciudad inicial, tenemos que forzar esa condición, al calcular la distancia entre la primera y la ultima neurona esta debe ser de valor unitario. Además agrandamos el valor inicial del ancho de la vecindad, esto ayuda a converger a una mejor solución con menor distancia entre ciudades.

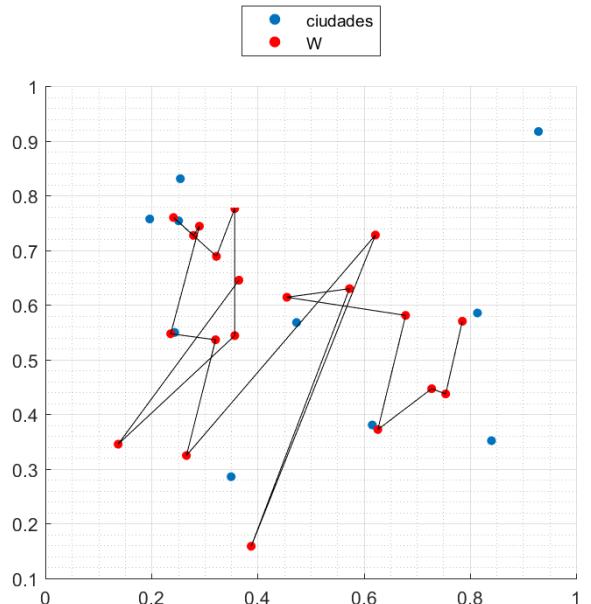
2.3. Implementación con neuronas como circulo

Pasando esto a una nueva versión del código que puede verse [aquí](#). Se obtienen los resultados mostrados en las **Figs. 11 y 12** donde se observa la mejora del resultado final en cuanto a la proximidad de la ultima y la primera ciudad y una distancia menor entre ciudades.

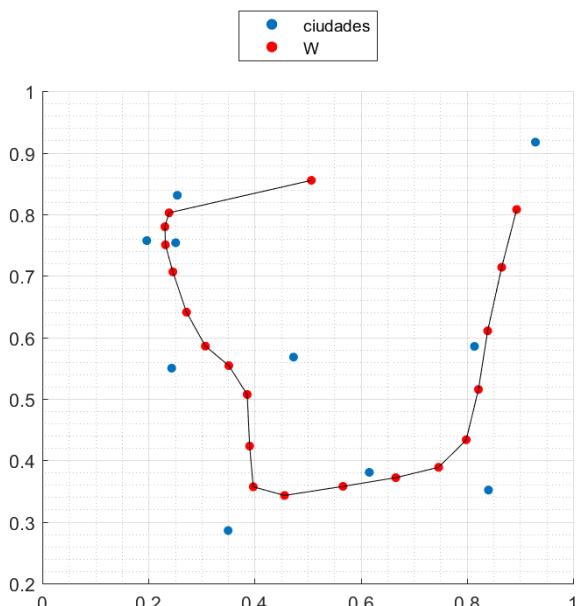
2.3.a. 20x10



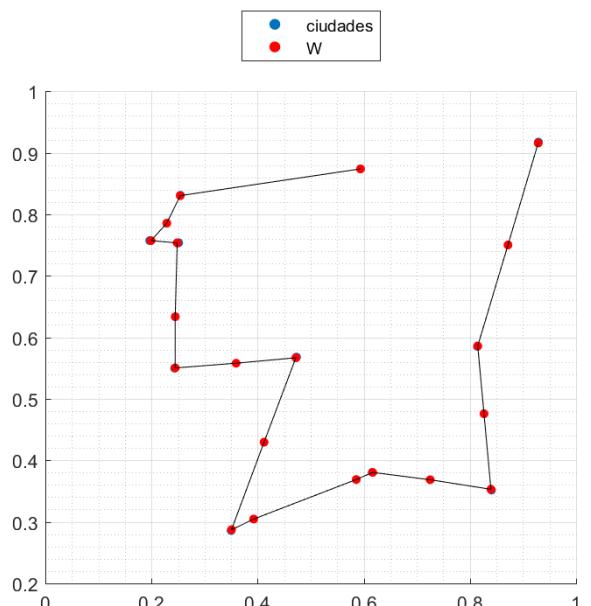
(a) Estado inicial



(b) Primer Iteración



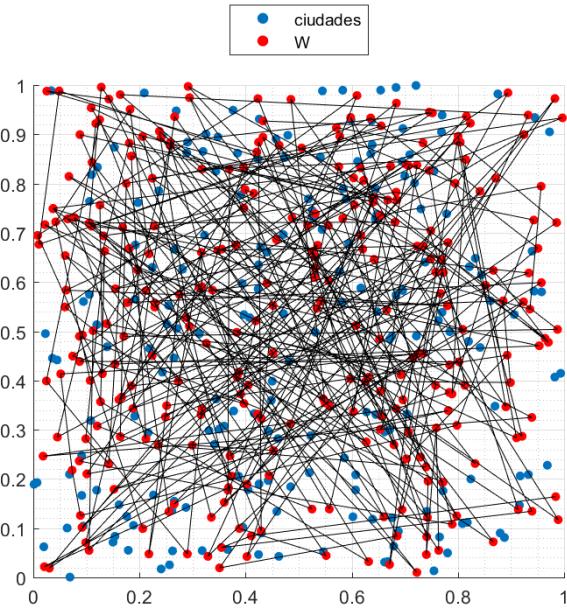
(c) Iteración 11



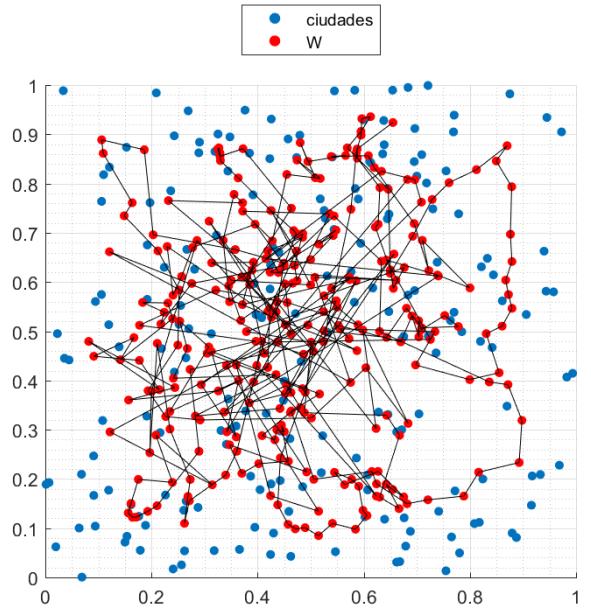
(d) Estado final

Figura 11: Evolución de la red para el caso 20x10

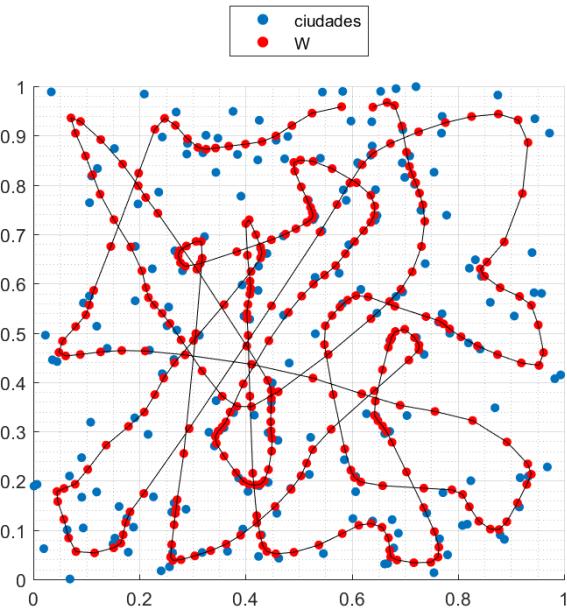
2.3.b. 300x200



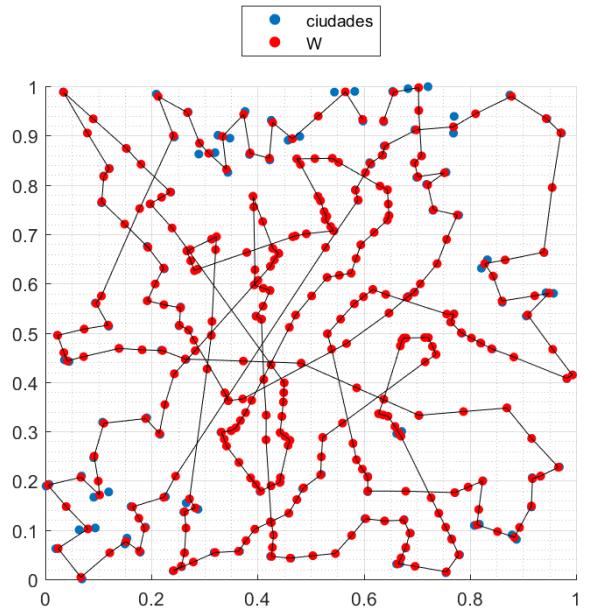
(a) Estado inicial



(b) Primer Iteración



(c) Iteración 11



(d) Estado final

Figura 12: Evolución de la red para el caso 300x200

Para otras pruebas de relación ciudades-neuronas pueden verse en este [link](#).