UNIVERSIDAD DE BUENOS AIRES
FACULTAD DE INGENIERÍA

MAESTRÍA EN SIMULACIÓN NUMÉRICA Y CONTROL

TRABAJO PRACTICO 3:

RED NEURONAL DE KOHONEN

Presentada como requisito parcial para la aprobación del curso
Sistemas Adaptativos y Redes Neuronales.

Estudiante:
Fredy Andrés Mercado Navarro
DNI: 94.872.342

Profesor titular:
Sergio Lew

Clases Prácticas:
Ana Laura Vadnjal

16 de enero de 2014
Buenos Aires, Argentina

Resumen

La red de Kohonen corresponde a una red de aprendizaje no supervisado, donde la intención es agrupar o categorizar los datos de entrada. Aquellas entradas similares son clasificadas por pertenecer a una misma categoría, lo que indica que deberían disparar la misma unidad de salida. Si por ejemplo tenemos varios vectores de entrada, donde podemos separarlos por pertenecer a regiones diferentes de un espacio, podemos encontrar pesos para cada grupo de vectores de entrada que corresponden a cada unidad o patrón que se desea agrupar. En este trabajo práctico se empleará esta red para demostrar el mapeo que se puede realizar sobre una distribución de puntos o muestras y para resolver el conocido "Travelling salesman problem", donde debemos hallar el camino más óptimo para visitar 500 ciudades ubicadas en puntos distintos.

Índice general

Índice de figuras

Capítulo 1

Teoría

1.1. Red de Kohonen

El algoritmo de Kohonen tiene en cuenta las interacciones laterales entre los vecinos para actualizar los pesos de salida y las diferencias entre los vectores de pesos y los vectores de entrada.

Tenemos N entradas contínuas ξ_1 a ξ_N que definen un punto ξ en un espacio real N-dimensional. Las unidades de salida O_i están contenidas en un arreglo de una o dos dimensiones, y están completamente conectadas a las entradas por los pesos w_{ij}. La regla de aprendizaje utiliza un ganador i^* como unidad de salida con el vector de pesos más cercano a la entrada actual ξ:

$$|w_{i*} - \xi| \leq |w_i - \xi| \quad \text{(para todo } i\text{)} \tag{1.1}$$

La regla de aprendizaje estuvo dada por Kohonen (ver sus trabajos de 1982 y 1989):

$$\Delta w_{ij} = \eta \Lambda(i, i^*)(\xi_j - w_{ij}) \tag{1.2}$$

para todo i y j. La función vecindad $\Lambda(i, i^*)$ es 1 para $i = i^*$ y cae mientras mayor es la distancia $|r_i - r_{i*}|$ entre las unidades i e i^* a la salida (vector).

Para construir un algoritmo práctico tenemos que especificar $\Lambda(i, i^*)(\xi_j - w_{ij})$ y η, los cuales pueden ser modificados durante el aprendizaje. Ambos pueden ser reducidos gradualmente. Tener $\eta = 0$ detendría el proceso de aprendizaje. Una elección típica para $\Lambda(i, i^*)$ es:

$$\Lambda(i, i^*) = \exp\left(\frac{-|r_i - r_{i*}|^2}{2\sigma^2}\right) \tag{1.3}$$

donde σ es un parámetro de ancho que decrece gradualmente. Nótese que la ecuación 1.3 es la regla de aprendizaje elegida por Kohonen.

Capítulo 2

Ejercicio teórico

2.1. Punto A

Una red de Kohonen de una sola entrada, con las mismas neuronas una al lado de la otra (en una sola fila) es entrenada con patrones x con cierta función distribución $P(x)$. Mostrar experimentalmente que, si la función densidad de probabilidad de las entradas es de la forma $p(x)\ \alpha\ x^{\alpha}$, entonces los pesos w de las neuronas tendrán la forma $w(x)\ \alpha\ x^{\beta}$. Cuánto vale β? (Ver *One-dimensional Equilibrium*, Hertz, Krogh and Palmer, pág. 242).

2.1.1. Desarrollo

En la red las neuronas están dispuestas unidimensionalmente, es decir, una al lado de la otra. Necesitamos calcular la constante de proporcionalidad c a partir de la función distribución de probabilidad, así:

$$P(x) = \int_0^x p(x)\ dx$$

Tenemos que la función densidad de probabilidad es proporcional a x^{α}, luego

$$p(x) = cx^{\alpha}$$

Luego

$$P(x) = \int_0^x cx^{\alpha}\ dx$$

Ahora integramos la función densidad de probabilidad entre 0 y 1 para hallar la constante de proporcionalidad, que debe ser igual a 1:

$$\int_0^1 cx^{\alpha}\ dx = \frac{cx^{\alpha+1}}{\alpha+1}\Big|_0^1 = \frac{c}{\alpha+1} = 1$$

Si despejamos obtenemos

$$c = \alpha + 1$$

Luego

$$p(x) = (\alpha+1)x^{\alpha}$$

Dado que tenemos la función densidad de probabilidad, podemos calcular la función distribución de probabilidad, así:

$$P(x) = y = \int_0^x p(x)\ dx = \int_0^x (\alpha+1)x^{\alpha}\ dx = x^{\alpha+1}$$

Reescribiendo, tenemos

$$y = x^{\alpha+1}$$

Requerimos los valores de la entrada x, que ahora proviene de entradas con una distribución uniforme y. Despejando obtenemos

$$x = y^{\left(\frac{1}{\alpha+1}\right)}$$

Las ecuaciones empleadas en el cálculo de la neurona ganadora, para el cálculo de la función vecindad y los deltas de pesos corresponden a las ecuaciones 1.1, 1.3 y 1.2.

Como el ejercicio pide demostrar experimentalmente el enunciado del problema, se realizó un programa de Matlab. El código tiene las siguientes variables:

n: cantidad de neuronas de la red unidimensional.
α: constante a elegir por el usuario.
η: constante de aprendizaje.
σ: parámetro de amplitud.
MAXITER: cantidad de veces que se entrenará la red por cada σ.

El funcionamiento del código es como sigue: primero, se llena un vector con pesos generados aleatoriamente en un intervalo [0,1]. Durante la corrida del programa se varía el parámetro sigma de un número grande (10) hasta uno pequeño (0.05) para permitir ajustes ΔW grandes al inicio y luego ajustes más pequeños conforme σ disminuye y la red aprende. Se tiene un ciclo encargado de entrenar a la red MAXITER veces. Dentro de este ciclo se genera aleatoriamente una entrada y se halla la unidad ganadora, que corresponde a aquella que presente la menor norma de la diferencia entre el valor de la entrada x y el peso de una neurona W. Luego empleo el algoritmo de Kohonen para actualizar los pesos de todas las unidades, hallando primero el resultado de la función vecindad (ver ecuación 1.3). Por último se grafican en orden los pesos obtenidos para cada unidad.

En las Figuras 2.1 y 2.2 se observan dos gráficas de los pesos ($\alpha = 0{,}5$ y $\alpha = 1{,}0$, respectivamente), teniendo escalados los números de las neuronas en el eje X para tenerlas todas en un intervalo [0,1] y poder comparar con la curva de $w(x) = x^{\beta}$, donde

$$\beta = \frac{1}{1 + (2/3)\alpha} \tag{2.1}$$

A ésta última curva se le asignó el color rojo, mientras que a la curva dada por los pesos que son resultado del entrenamiento de la red se le asignó el color azul. El número de neuronas n, la constante α y la constante de aprendizaje η no fueron modificadas durante las corridas del programa. Con esto demostramos que si la función densidad de probabilidad de las entradas es de la forma $p(x) \propto x^{\alpha}$ los pesos w de las neuronas tendrán la forma $w(x) \propto x^{\beta}$, con β dado por la Ecuación 2.1.

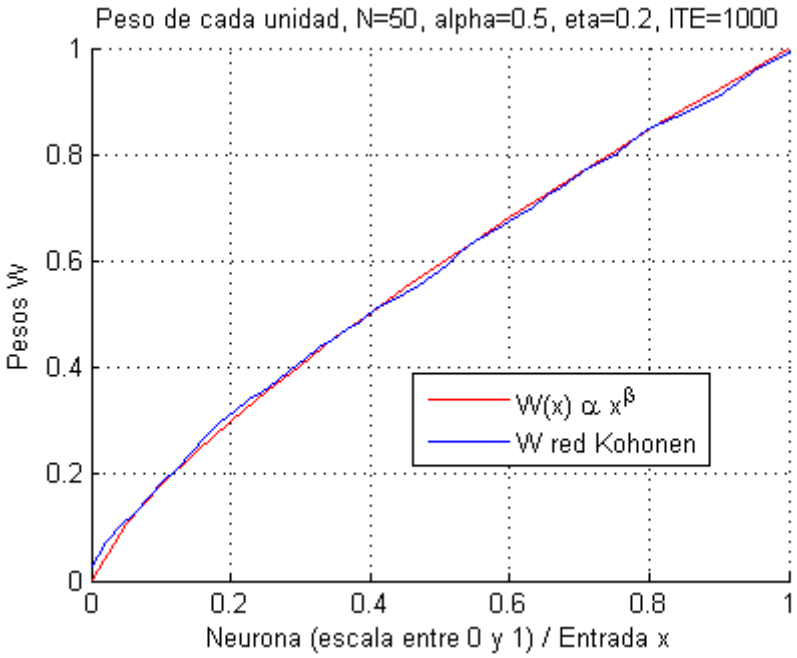

Figura 2.1: Pesos de cada unidad tras entrenamiento de la red con una sola entrada. $\alpha = 0{,}5$.

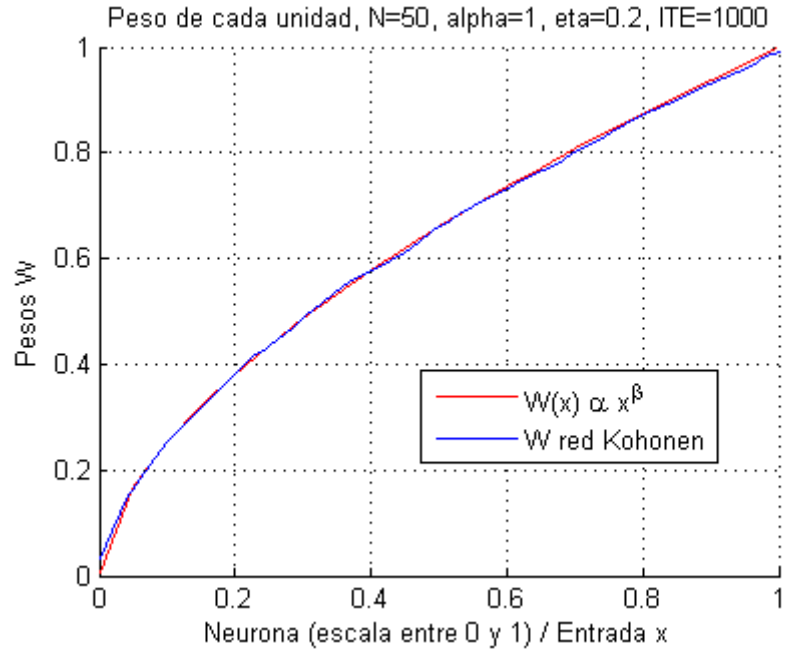

Figura 2.2: Pesos de cada unidad tras entrenamiento de la red con una sola entrada. $\alpha = 1{,}0$.

Capítulo 3

Ejercicios prácticos

3.1. Punto B

Hacer una red de Kohonen de 2 entradas que aprenda una distribución uniforme en el círculo unitario. Mostrar el mapa de preservación de topología.

3.1.1. Desarrollo

Se realizó un código en Matlab para mostrar el mapeo que realiza la red de Kohonen sobre los puntos aleatorios que están distribuidos dentro de un círculo de radio unitario. Las variables que determina el usuario del código son las siguientes:

- N: cantidad de neuronas por lado de la red.
- η: constante de aprendizaje. Se dejó constante.
- nmuestras: cantidad de muestras o puntos de distribución aleatoria que deseo que contenga el círculo de radio unitario.

El funcionamiento del código es el siguiente:

1. Se genera una red de pesos tridimensional, que estará conformada por dos matrices de pesos de $n \times n$. Una matriz para la coordenada X y otra para la coordenada Y. Con las parejas correspondientes podemos entonces obtener las coordenadas (X,Y) que se irán modificando a medida que se entrene la red para generar un mapeo de los puntos que conforman la muestra.

2. Se tiene un primer ciclo *for* que varía el parámetro σ de mayor a menor para lograr cambios grandes al comienzo y ajustes cada vez más pequeños conforme se entrena la red. Este valor inicia en 20 y finaliza en 0.05, disminuyendo en pasos de 0.05.

3. Luego, para un σ fijo se recorren todas las muestras y se determina la neurona ganadora, es decir aquella para la cual es mínima la distancia entre un punto (muestra) y una neurona.

4. Teniendo la neurona ganadora calculamos la función vecindad (ver ecuación 1.3) y actualizamos los pesos de todas las neuronas luego de calcular el ΔW correspondiente. Este procedimiento se repite hasta que queramos detener el proceso de aprendizaje, en este caso, hasta que $\sigma = 0{,}05$.

El resultado del entrenamiento de la red de Kohonen se puede apreciar graficando la topología de la red: en la Figura 3.1 se muestra el mapa de topología para el estado inicial de los pesos. Aquí la red aún no ha iniciado el proceso de aprendizaje, de modo que los pesos corresponden a los valores aleatorios iniciales que les fueron asignados.

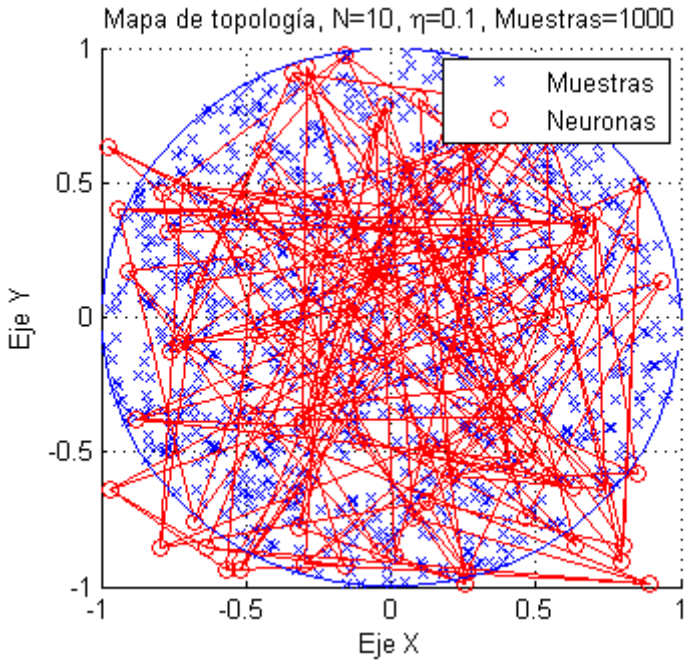

Figura 3.1: Mapa de topología inicial. Pesos aleatorios.

En la Figura 3.2 se aprecia el mapa de topología cuando la red ha iniciado su aprendizaje. Como $\sigma = 15$ es un valor grande, el cambio sufrido por los pesos es significativo. Se observa una contracción de la red como producto de estos ΔW grandes.

En la figura 3.3, con $\sigma = 3$ el mapa ha comenzado a expandirse para adaptarse mejor a la distribución de la muestra. Se puede observar la red de 10×10 ordenada y sin ninguna superposición.

En la figura 3.4, con $\sigma = 1$ el aprendizaje ha logrado que los pesos abarquen un area mayor, tratando de alcanzar las muestras más exteriores del círculo. Finalmente, en la Figura 3.5, con $\sigma = 0{,}05$ el entrenamiento de la red ha finalizado y el mapa de Kohonen ha logrado una distribución que cobija casi todo el cículo unitario y donde se observa que varias neuronas se sitúan en zonas donde se presentan concentraciones de puntos.

3.2. Punto 1

Resolver (aproximadamente) el "Travelling salesman problem" para 500 ciudades con una red de Kohonen.

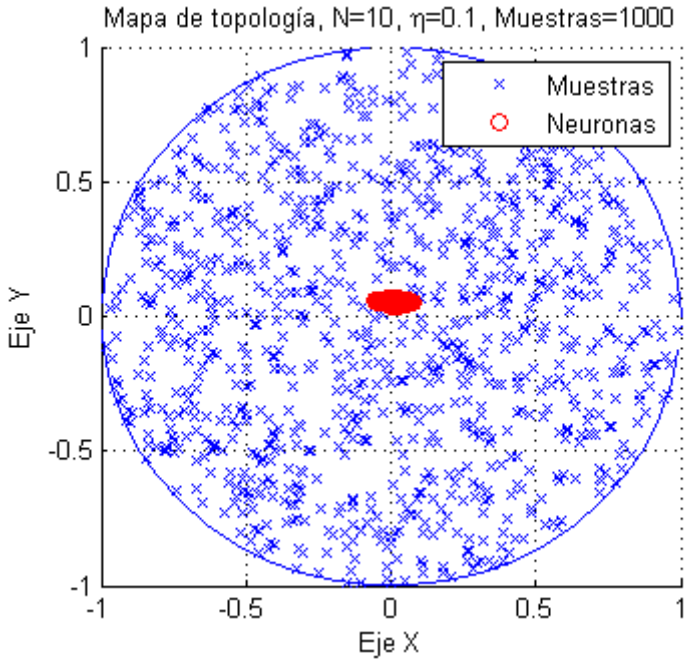

Figura 3.2: Mapa de topología tras haber entrenado la red desde $\sigma = 20$ hasta $\sigma = 15$.

3.2.1. Desarrollo

Se desarrolló un código donde las constantes que ingresa el usuario son:

- η: constante de aprendizaje.
- nmuestras: cantidad de puntos en el plano X-Y o muestras con las cuales se entrenará la red.
- σ: parámetro de amplitud presente en el cálculo de la función vecindad.
- MAXITER: cantidad máxima de iteraciones de entrenamiento por cada variación del paraámetro σ.
- N: cantidad de neuronas por lado de la red.

El funcionamiento del código es como sigue:

1. Se calcula N, el cual elegimos que fuera el doble de la cantidad de muestras, debido a que durante el proceso de aprendizaje pueden coincidir dos o más neuronas en el mismo punto de muestra.
2. Se genera un aro con las neuronas, cuyas coordenadas están dadas por los pesos. A medida que los pesos se actualizan la forma del aro cambia, adaptándose al camino óptimo del viajero. El código se diseñó para evitar discontinuidades entre los extremos, de modo que todos los puntos están unidos sobre un camino que permanece cerrado.
3. Se generan los puntos o muestras aleatoriamente, los cuales poseen coordenadas x e y.

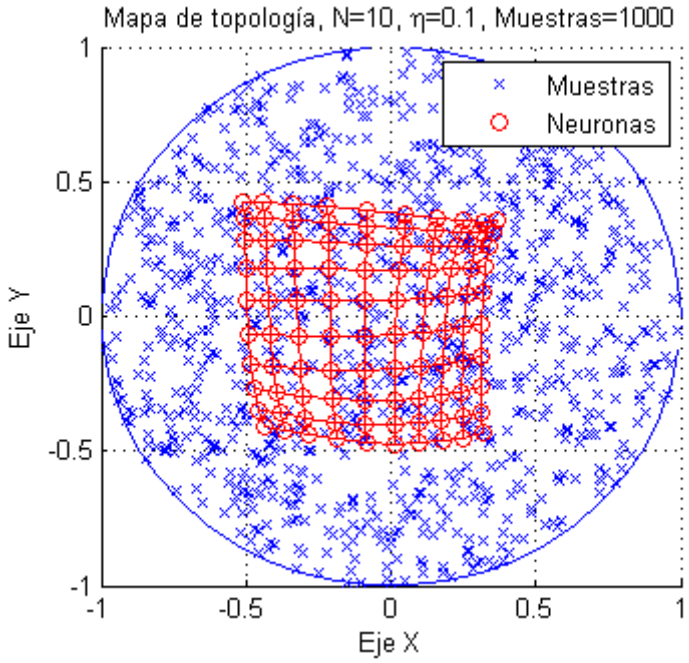

Figura 3.3: Mapa de topología tras haber entrnado la red desde $\sigma = 20$ hasta $\sigma = 3$.

4. Se calculan las distancias entre la posición de la neurona ganadora y las posiciones de las demás neuronas. Sólo se emplea la distancia menor. Esto permite que la red permanezca cerrada y que las neuronas de los extremos no terminen en posiciones alejadas. De no hacer esto las neuronas de los extremos terminarían en posiciones que no harían óptimo el camino del viajero, ya que éste debe iniciar y terminar el recorrido en la misma ciudad.

5. Se encuentra la neurona ganadora y con base en ella se actualizan los pesos de todas las neuronas.

6. Los últimos tres pasos se realizan para diferentes valores de σ y para MAXITER número de iteraciones.

Para la solución del problema para 500 ciudades se empleó $\eta = 0{,}2$, MAXITER= 5 y un número de neuronas $N = 1000$. El parámetro σ fue variado desde 10 en pasos de 0.25 hasta 0.25, donde termina el proceso de actualización de los pesos (coordenadas de las neuronas).

En la Figura 3.6 se aprecia la distribución inicial de las neuronas, donde las coordenadas son los pesos. Aquí el proceso de actualización aún no ha iniciado.

En la Figura 3.7 tomamos una imagen del camino del viajero cuando se han actualizado los pesos al ir iterando y variando σ desde 10 hasta 5. Se observa como el camino se ha modificado tratando de adaptarse a las 500 ciudades que debe recorrer.

Finalmente, en la Figura 3.8 el camino del viajero ha adoptado la forma más óptima que pudo hallar para los parámetros de simulación establecidos. El camino del viajero recorre prácticamente todas las ciudades. Muchas neuronas no están directamente sobre

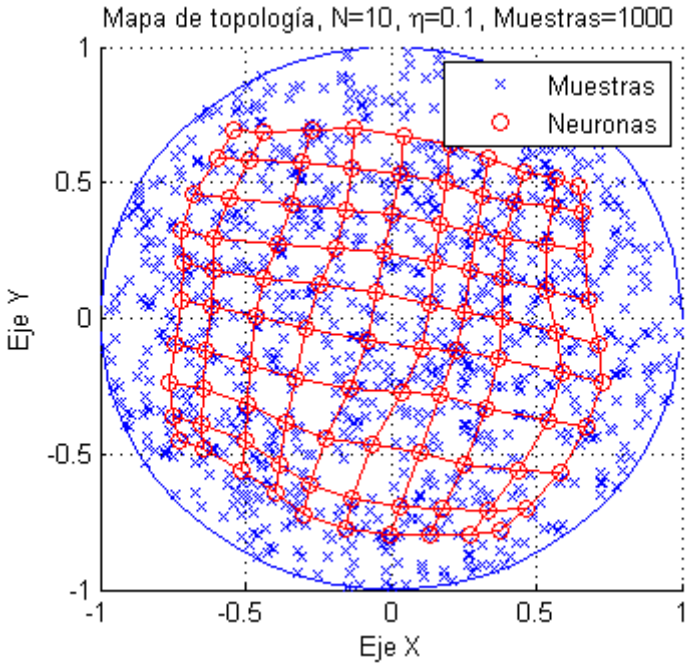

Figura 3.4: Mapa de topología tras haber entrnado la red desde $\sigma = 20$ hasta $\sigma = 1$.

una ciudad, pero si sobre los caminos más cortos entre dos de ellas. El camino hallado es un camino sin discontinuidades.

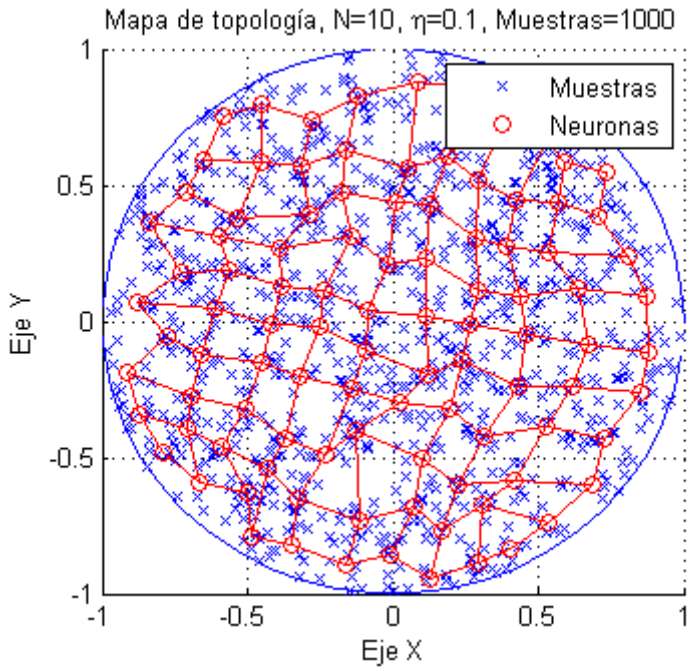

Figura 3.5: Mapa de topología. Resultado final tras haber entrenado la red desde $\sigma = 20$ hasta $\sigma = 0{,}05$.

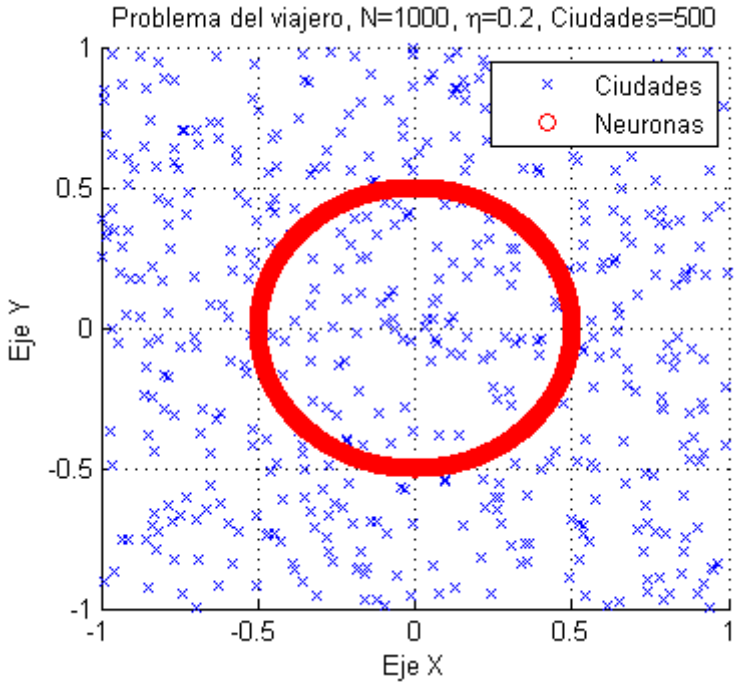

Figura 3.6: Distribución inicial de pesos (en rojo).

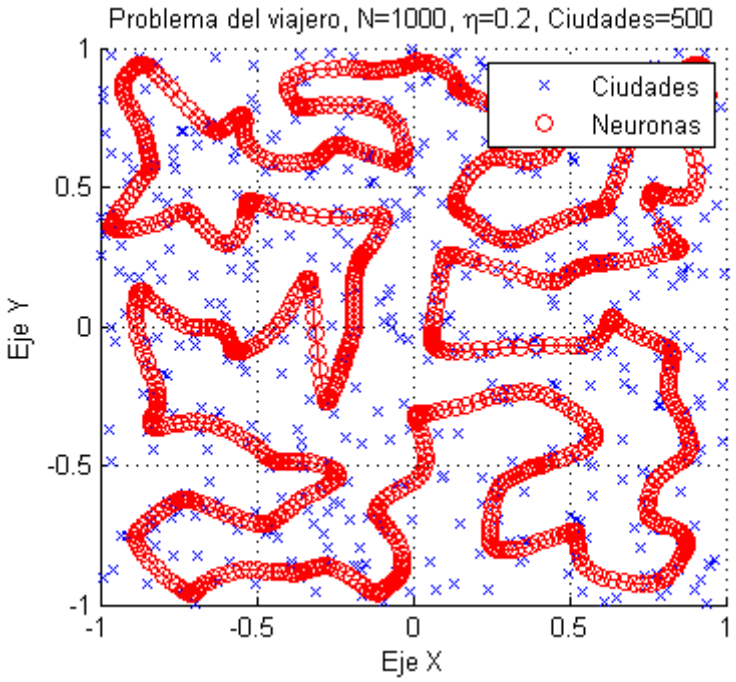

Figura 3.7: Camino del viajero tras haber entrenado la red desde $\sigma = 10$ hasta $\sigma = 5$.

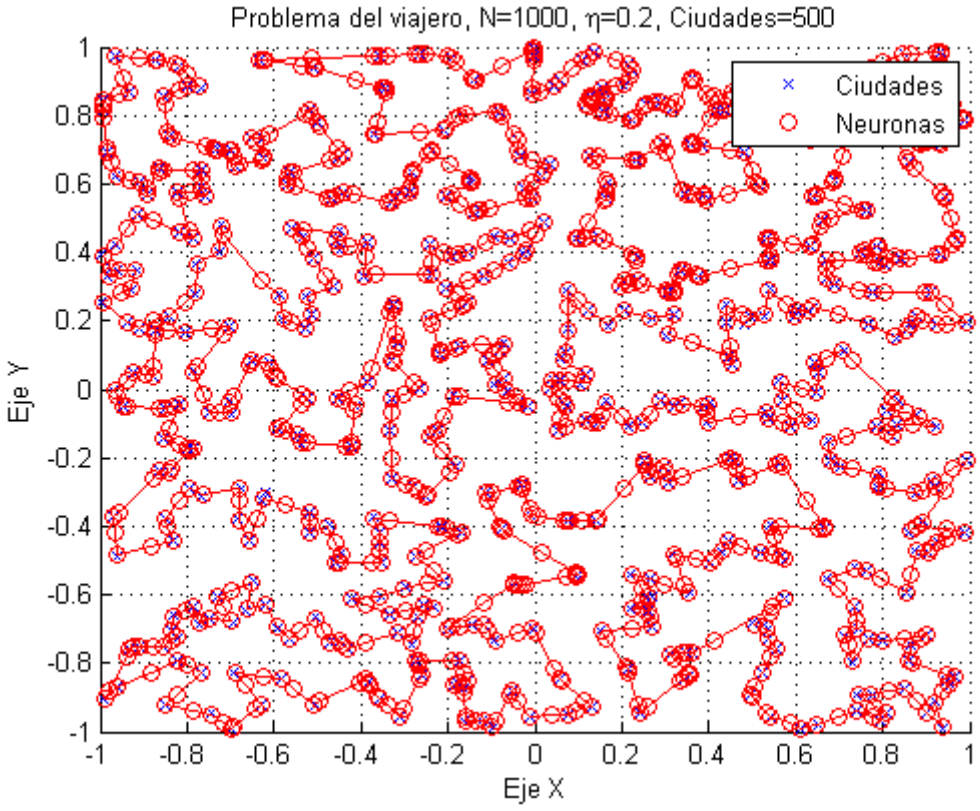

Figura 3.8: Camino del viajero. Resultado final tras haber entrenado la red desde $\sigma = 10$ hasta $\sigma = 0{,}25$ en pasos de 0,25.

Capítulo 4

Comentarios y Conclusiones

- Se mostró experimentalmente que si entrenamos una red de Kohonen unidimensional con una sola entrada, donde las entradas x tienen una función densidad de probabilidad proporcional a x^{α}, entonces los pesos w de las neuronas serán proporcionales a x^{β}, donde β está dado por la ecuación 2.1.

- Se desarrolló un mapeo de una distribución de puntos aleatorios dentro del círculo unitario. El progreso de la red muestra la adaptación a los puntos de la distribución. Al comienzo del entrenamiento el mapa de la topología de la red está totalmente desordenado. El comienzo de la actualización de los pesos en un número alto $\sigma = 20$ produce una contracción inmediata de la red. A medida que la red se actualiza la red se expande hasta cobijar casi todo el círculo. Las neuronas se ubican en aquellas zonas donde hay una mayor densidad de puntos de muestra.

- Se resolvió el problema del viajero utilizando una red de Kohonen, donde se empleó el método del anillo elástico para el inicio de la actualización de los pesos. El camino obtenido no posee discontinuidades.

- El problema del viajero se resolvió en forma aproximada sin variar la constante de aprendizaje, aunque el texto de Hertz, Krogh y Palmer sugiere que también se puede variar de mayor a menor, como se hizo con σ, para obtener mejores resultados.

- El mapeo de características utilizado en la actualización de pesos de la red de Kohonen está ligado a la actualización de pesos que se realiza una vez obtenida la neurona ganadora. Allí se emplea la función vecindad, que carga con mayor peso a las neuronas cercanas a la neurona ganadora, resta peso a las neuronas que se alejan y deja prácticamente iguales los pesos de las neuronas muy lejanas. Esto se puede apreciar gráficamente en la Figura 9.9 del texto de Hertz, Krogh y Palmer (también conocida con el nombre de "sombrero mexicano").