Capítulo 7

Redes Neuronales en optimización combinatoria

A las redes neuronales (conneccionismo, proceso paralelo distribuido, computación neuronal, redes adaptivas, computación colectiva) las podemos entender desde dos puntos de vista:

- Computacional: representar funciones usando redes de elementos con cálculo aritmético sencillo, y métodos para aprender esa representación a partir de ejemplos. La repesentación es útil para funciones complejas con salidas continuas y datos con ruido.
- Biológico: modelo matemático de la operación del cerebro. Los elementos sencillos de cómputo corresponden a neuronas, y la red a una colección de éstas.

La neurona es la unidad funcional fundamental del sistema nervioso. Cada neurona tiene un cuerpo (soma) que tiene un núcleo, tiene un grupo de fibras (dendritas), una de las cuales es más larga (axón). El axón se bifurca eventualmente en sinapses. Las señales se propagan en una reacción electroquímica complicada. Las substancias químicas transmisoras se liberan de las sinapses y entran a la dendrita, aumentando o disminuyendo el potencial eléctrico del cuerpo de la célula.

Tabla 7.1: Comparación gruesa de las capacidades computacionales de cerebros y computadoras (1994).

	Computadora	Cerebro Humano
Unidades Computacionales	1 CPU, 10^5 compuertas	10^{11} neuronas
Unidades de Almacenamiento	10^9 bits RAM, 10^{10} bits disco	10^{11} neuronas, 10^{14} sinapses
Ciclo (tiempo)	10^{-8} seg.	10^{-3} seg.
Anchobanda	10^9 bits/seg.	10^{14} bits/seg.
Actualizaciones/seg.	10^5	10^{14}

Cuando el potencial alcanza un umbral se transmite un pulso elétrico o acción potencial a través del axón. Las sinapses que aumentan el potencial se llaman exitatorias y los que disminuyen, inhibidoras.

La conección "sináptica" es *plástica* (cambia con la estimulación).

Se pueden formar nuevas conecciones y las neuronas migran de un lugar a otro. Esto se cree que forman la base de aprendizaje en el cerebro.

En general el mapeo de regiones con funciones puede ser múltiple y cambiar cuando un área es dañada (pero no se sabe bien como se hace).

Lo sorprendente es que una colección de células simples puedan dar pensamiento, acción y conciencia (*cerebros causan mentes* (Searle 92)).

A pesar de que una computadora es millones de veces más rápida por proceso individual, el cerebro finalmente es billones de veces más rápido (ver table 7.1).

Una de las atracciones, es construir un mecanismo que combine el paralelismo del cerebro con la velocidad de las máquinas.

Los cerebros son mucho más tolerantes (en 70-80 años, no se tiene que reemplazar una tarjeta de memoria, llamar al servicio o hacer *reboot*). La tercera atracción es su degradación gradual.

Las redes neuronales artificiales son modelos muy simplificados de las redes neuronales biológicas.

7.0.1 Historia

Existió mucho desarrollo en los primeros años de la computación: McCulloch y Pitts (43), Hebb (49), Minsky (51) (primera red), Ashby (52), Rosenblatt (57) (perceptrón), Selfridge (59) (pandemonium), Widrow y Hoff (60) (adalines), Nilsson (65 - 90), Minsky y Papert (69).

Después del libro de Minsky y Papert prácticamente no se hizo nada durante los siguientes 10 años.

El resurgimiento comenzó en la decada de los 80's: Hinton y Anderson (81), Hopfield (82), Hinton y Sejnowski (83 y 86) y los dos volumens de PDP (Parallel Distributed Processing) anthology (Rumelhart *et al.* 86).

Una red neuronal artificial está compuesta por nodos o unidades, conectados por ligas. Cada liga tiene un peso numérico asociado. Los pesos son el medio principal para almacenamiento a largo plazo en una red neuronal, y el aprendizaje normalmente se hace sobre la actualización de pesos.

Algunas unidades están conectadas al medio ambiente externo y pueden diseñarse como unidades de entrada o salida.

Los pesos se modifican para tratar de hacer que el comportamiento entrada/salida se comporte como el del ambiente.

Cada unidad tiene un conjunto de ligas de entrada (provenientes de otras unidades) y un conjunto de ligas de salida (hacia otras unidades), un nivel de activación, y una forma de calcular su nivel de activación en el siguiente paso en el tiempo, dada su entrada y sus pesos (cada unidad hace un cálculo local basado en las entradas de sus vecinos).

En la práctica, casi todas las implementaciones de RN son en software y utilizan un control síncrono en su actualización.

Para el diseño uno debe de decidir:

- Número de unidades.
- Cómo se deben de conectar.

- Qué algoritmo de aprendizaje utilizar.
- Cómo codificar los ejemplos de entradas y salidas.

Cada unidad recibe señales de sus ligas de entradas y calcula un nuevo nivel de activación que manda a través de sus ligas de salidas.

La computación se hace en función de los valores recibidos y de los pesos.

Se divide en dos:

1. Un componente lineal, llamado la función de entrada (in_i), que calcula la suma de los valores de entrada.
2. Un componente no lineal, llamado función de activación (g), que transforma la suma pesada en una valor final que sirve como su valor de activación (a_i).

Normalmente, todas las unidades usan la misma función de activación.

La suma pesada es simplemente las entradas de activación por sus pesos correspondientes:

$$in_i = \sum_j w_{j,i} a_j = \mathbf{w_i} \cdot \mathbf{a_i}$$

$\mathbf{w_i}$: vector de los pesos que llegan a la unidad i
$\mathbf{a_i}$: vector de los valores de activación de las entradas a la unidad i

El nuevo valor de activación se realiza aplicando una función de activación g:

$$a_i \leftarrow g(in_i) = g(\sum_j w_{j,i} a_j)$$

Se obtienen modelos diferentes cambiando g. Las opciones más comunes son (ver figura 7.1):

- Función escalón:

$$escalon_t(x) = \begin{cases} 1, & \text{si } x \geq t \\ 0, & \text{si } x < t \end{cases}$$

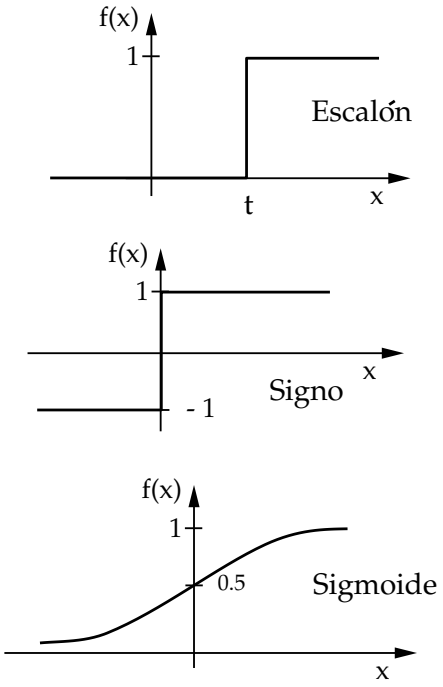

Figura 7.1: Funciones de activación comunes para Redes Neuronales.

- Signo:

$$signo(x) = \begin{cases} +1, & \text{si } x \geq 0 \\ -1, & \text{si } x < 0 \end{cases}$$

- Sigmoide:

$$sigmoide(x) = \frac{1}{1 + e^{-x}}$$

Una de las motivaciones iniciales en el diseño de unidades individuales fué la representación de funciones Booleanas básicas (McCulloch y Pitts, '43).

Esto es importante, porque entonces podemos usar estas unidades para construir una red que compute cualquier función Booleana.

Los modelos de redes neorunales principalmente usados para resolver problemas de optimización son:

- Las máquinas de Boltzmann.
- Las redes de Hopfield.
- Los mapas auto-organizativos (SOM).

7.1 Máquinas de Boltzmann

Las redes neuronales se pueden caracterizar por sus:

- Valores binarios o continuos.
- Transiciones determinísticas o probabilísticas.
- Conecciones unidireccionales o bidireccionales.
- Representaciones distribuidas o locales.
- Aprendizaje supervisado o no supervisado.
- Unidades ocultas o no.

La máquina de Boltzmann usa estados binarios, conecciones bidireccionales, transiciones probabilísticas y puede tener unidades ocultas.

Para ajustar los estados de las unidades individuales se usa un mecanismo de transición de estados que está regido por el algoritmo de recocido simulado.

El modelo matemático de la Máquina de Boltzmann tiene dos atributos:

1. Se puede considerar como un modelo de implementación paralela en forma masiva del algoritmo de recocido simulado.
2. Permite el diseño de un algoritmo de aprendizaje basado en conceptos relativamente simples.

Una máquina de Boltzmann se puede ver como una red de varias unidades de dos estados (prendida = 1, apagada = 0) conectadas de cierta forma. El conjunto de conecciones normalmente incluye "auto conecciones" (conecciones de una unidad a sí misma).

Si se tiene una conección entre dos unidades u y v, entonces la conección entre ellas $\{u, v\}$ está prendida si: $k(u) \cdot k(v) = 1$ (donde $k(i)$ denota el estado de la unidad i).

Con cada conección se asocia una fuerza (número real). La fuerza es una medidad cuantitativa de lo deseable de la conección.

La fuerza de la conección $\{u, u\}$ se llama el sesgo de la unidad u.

Una configuración k está dada por el estado global de una máquina de Boltzmann (i.e., por los estados de cada unidad).

La función de consenso asigna a cada configuración un número real dado como la suma de las fuerzas de las conecciones activadas

$$C(k) = \sum_{\{u,v\} \in C} w_{\{u,v\}} k(u) k(v)$$

.

donde: $w_{u,v}$ = la fuerza de conección y $k(i)$ = la configuración de la unidad i.

El consenso es grande si están activadas muchas de las conecciones exitatorias (y pequeño si son las inhibitorias).

El objetivo de la máquina de Boltzmann es encontrar una configuración que alcance un máximo global (i.e., una configuración de consenso máximo).

El ajuste está determinada por una función estocástica de los estados de los vecimos y sus fuerzas de conección.

Básicamente usa el criterio de aceptación del recocido simulado.

Existen dos modelos:

- Máquinas de Boltzmann sequenciales: las unidades cambian de estado una a la vez.
- Máquinas de Boltzmann paralelas: las unidades cambian de estado simultaneamente.

7.1.1 Máquinas de Boltzmann Secuenciales

Dada una máquina de Boltzmann en una configuración k, la configuración vecina k_u se define como la configuración que se obtiene del estado k al cambiar el estado de una unidad u (de 0 a 1 o viceversa).

$$k_u(v) = \begin{cases} k(v) & \text{si } v \neq u \\ 1 - k(v) & \text{si } v = u \end{cases}$$

La vecindad se define como todos los estados de configuraciones vecinas a la configuración k.

Si C_u denota todas las conecciones que inciden en la unidad u (eliminando $\{u, u\}$), y sea $C' = C - C_u - \{u, u\}$. La diferencia de consenso entre las configuraciones k y k_u se denota como:

$$\triangle C_k(u) = C(k_u) - C(k)$$

Dado que la contribución de las conecciones en C' al consenso es igual para k que para k_u, se obtiene:

$$\begin{aligned}\triangle C_k(u) &= \left[k_u(u)\sum_{\{u,v\}\in C_u} w_{\{u,v\}}k_u(v) + k_u^2(u)w_{\{u,u\}}\right] \\ &\quad - \left[k(u)\sum_{\{u,v\}\in C_u} w_{\{u,v\}}k(v) + k^2(u)w_{\{u,u\}}\right]\end{aligned}$$

que nos da al cambiar $k_u(v)$ por $1 - k(v)$ y desarrollar:

$$\triangle C_k(u) = (1 - 2k(u))\left[\sum_{\{u,v\}\in C_u} w_{\{u,v\}}k(v) + w_{\{u,u\}}\right]$$

Básicamente lo que dice es que el efecto del consenso, resultando de cambiar el estado de la unidad u, está completamente determinado por los estados de los vecinos y de sus fuerzas de conección.

Por lo mismo una unidad puede ser evaluada localmente, lo cual es muy importante por su potencial paralelización.

Una configuración de máximo local es aquella en donde no se puede aumentar cambiando transiciones de un solo estado.

Ejemplo: Las figuras 7.2 y 7.3 muestra un ejemplo de una máquina de Boltzmann.

Como en el caso de recocido simulado, se puede usar el concepto de cadenas de Markov para describir las transiciones de estado de una máquina de Boltzmann.

En una máquina secuencial consiste de dos pasos:

1. Dada una configuración k, seleccionar una unidad u en donde posiblemente se va a cambiar su estado y por lo tanto generar una configuración vecina k_u.

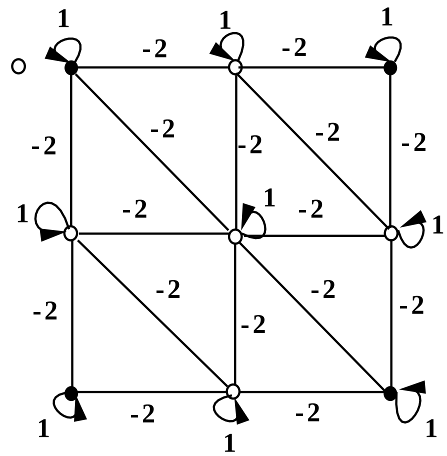

Figura 7.2: Ejemplo de una máquina de Boltzmann después de haber llegado a un mínimo.

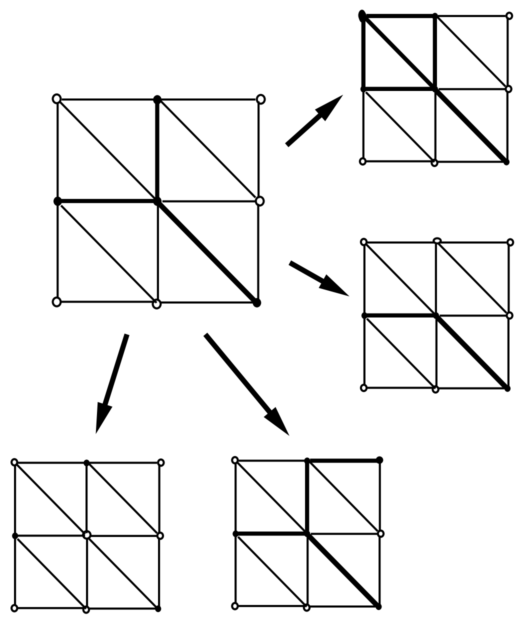

Figura 7.3: Diferentes estados vecinos para el ejemplo de la máquina de Boltzmann.

2. Evaluar si la configuración k_u se acepta o no.

También se introduce un parámetro de control c, que determina la probabilidad de aceptación de una transición.

La probabilidad de transición $P_{kl}(c)$ de la configuración k a la l se define como:

$$P_{kl}(c) = \mathbf{P}_c\{X(m) = l \mid X(m-1) = k\}$$

$$P_{kl}(c) = \begin{cases} G(u)A_k(u,c) & \text{si } l = k_u \\ 1 - \sum_{u \in U} P_{kk_u}(c) & \text{si } l = k \\ 0 & \text{de otra forma} \end{cases}$$

donde $G(u)$ denota la probabilidad de generar una transición de estado de la unidad u, $A_k(u,c)$ denota la probabilidad de aceptación de la transición, y c denota el parámetro de control.

La probabilidad de generación se escoge normalmente de forma uniforme sobre las posibles unidades e independiente de la configuración k o el parámetro de control c.

La probabilidad de aceptación se escoge como:

$$A_k(u,c) = \frac{1}{1 + exp(\frac{-\triangle C_k(u)}{c})}$$

Esto difiere un poco de la probabilidad de aceptación del algoritmo de recocido simulado, pero en realidad muy poco. Ambas probabilidades dan la misma distribución estacionaria y tienen las mismas propiedades de convergencia.

La figure 7.4 muestra esta función para varios valores de c.

Se puede probar que las máquinas de Boltzmann convergen asintóticamente al conjunto de configuraciones globales óptimas basandose en la existencia de una distribución estacionaria.

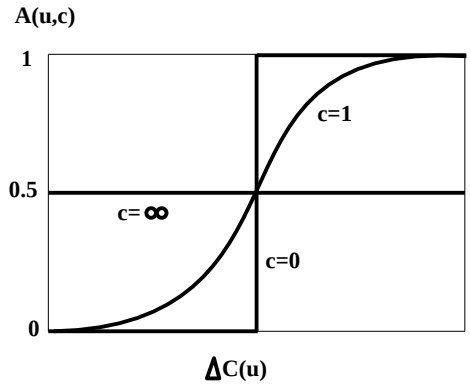

Figura 7.4: Probabilidad de aceptación para diferentes valores de c.

Como en recocido simulado, se tiene que hacer una aproximación en tiempo finito al especificar los parámetros que determinan el esquema de enfriamiento.

El esquema es básicamente el mismo que se sigue para el recocido simulado.

Pesos positivos (negativos) en las auto–conecciones indican una tendencia a estar las unidades prendidas (apagadas) y entre las conecciones con otras unidades indican que deben de estar las dos unidades conectadas prendidas.

7.1.2 Máquinas de Boltzmann Paralelas

Las máquinas de Boltzmann facilitan el paralelismo ya que la evaluación de transición puede hacerse localmente.

Algunas posibilidades son:

- Paralelismo síncrono: conjuntos de transiciones de estado se realizan en forma sucesiva, cada una haciendo transiciones individuales. Se actualiza la información y se continua con el proceso. Este esquema requiere de un mecanismo de reloj global para controlar la sincronización.

  Podemos tener paralelismo limitado (no cambiar al mismo tiempo unidades adyacentes), paralelismo ilimitado (como autómatas celulares).

En paralelismo ilimitado puede existir un cálculo erroneo en la evaluación del concenso (e.g., se prenden dos unidades vecinas para incrementar el consenso, tomando en cuenta que sus vecinos están apagado, que podría decrementar el consenso al prenderse los dos).

Sin embargo, estos errores decrecen al decrementar el parámetro c y en la práctica se acepta.

- Paralelismo asíncrono: se hace la transición pero no necesariamente con información actualizada. No se requiere un mecanismo de reloj global.

7.1.3 Estrategia para usar Máquinas de Boltzmann en problemas de optimización combinatoria

Para utilizar una máquina de Boltzmann para resolver un problema de optimización combinatoria, se define una función bijectiva $m : \mathcal{R} \rightarrow \mathcal{S}$ que mapea el conjunto de configuraciones $\mathcal{R}$ al conjunto de soluciones $\mathcal{S}$, siguiendo la siguiente estrategia general:

1. Formula el problema de optimización como un problema de programación de 0 y 1 (formula el problema tal que $X_i = \{0, 1\}$, para $i = 1, \ldots, n$).
2. Define una máquina de Boltzmann tal que el estado en cada unidad determine el valor de una variable.
3. Define el conjunto de conecciones y las fuerzas tal que la función de consenso sea factible y preserve el orden.

Una función de consenso es factible si todos los máximos locales de la función de consenso corresponden a soluciones factibles.

Factibilidad implica que siempre se encuentra una solución factible.

Una función de consenso se dice que preserva el orden si $\forall k, l \in \mathcal{R}$, con $m(k), m(l) \in \mathcal{S}'$, tenemos que:

$$f(m(k)) > f(m(l)) \Rightarrow C(k) > C(l)$$

.

Para un problema de minimización sería:

$$f(m(k)) < f(m(l)) \Rightarrow C(k) > C(l)$$

7.1.4 Ejemplo: el problema de Max Cut

El problema del Max Cut es el siguiente: Dado un grafo $G = (V, E)$ con pesos positivos en los arcos, el problema consiste en enconrar una partición de V en conjuntos disjuntos V_0 V_1, al que la suma de los pesos de los arcos que tienen un extremo en V_0 y el otro en V_1 sea máxima.

Los pesos son simétricos ($w_{i,j} = w_{j,i}$) y podemos definir una variable x_i binaria como sigue:

$$x_i = \begin{cases} 1 & \text{si } i \in V_1 \\ 0 & \text{si } i \in V_0 \end{cases}$$

Entonces el problema Max Cut lo podemos formular como sigue:

$$max f(x) = \sum_{i=i}^{n} \sum_{j=i+1}^{n} w_{i,j}\{(1 - x_i)x_j + x_i(1 - x_j)\}$$

Sea b_i la suma de los pesos de todos los nodos que inciden en el nodo i, i.e., $b_i = \sum_{j=1}^{n} w_{i,j}$. Si C_b son las conecciones de sesgo (auto-conecciones) y C_w son las conecciones de peso (entre nodos), una función de consenso que preserva el orden es:

$$\forall u_i, u_i \in C_b : w_{u_i,u_i} = b_i$$

$$\forall u_i, u_j \in C_w : w_{u_i,u_j} = -2w_{i,j}$$

En la figure 7.5 se ve un grafo de ejemplo y su máquina de Boltzmann correspondiente.

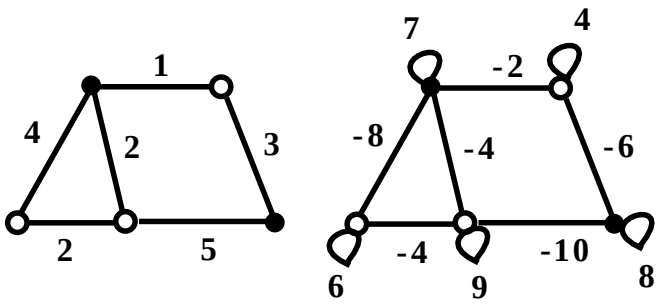

Figura 7.5: Ejemplo de una grafo para resolver el problema de Max Cut y su máquina de Boltzmann correspondiente.

7.1.5 Discusión

Las máquinas de Boltzmann tambien pueden usarse con nodos ocultos y pueden servir para clasificar objetos que se parezcan más a algún conocido.

Por otro lado son muy tolerantes a ruido.

Como en todo modelo neuronal, se tienen que contestar preguntas como: (i) cuantas unidades ocultas se necesitan, (ii) como se deben de conectar las unidades

7.2 Redes de Hopfield

Las redes de Hopfield son probablemente las mejor entendidas de redes recurrentes. Tienen conecciones bidireccionales con pesos simétricos (i.e., $W_{i,j} = W_{j,i}$). Todas las unidades son tanto unidades de entrada como de salida. La función de activación es la función signo, y los valores de activación pueden ser sólo ± 1.

Una red de Hopfield funciona como una memoria asociativa. Despues de entrenarse con un conjunto de ejemplos, un nuevo estímulo causa la red a "asentarse" en un patrón de activación correspondiente al ejemplo de entrenamiento que se parece más al nuevo estímulo.

Esto es, se alimenta un patrón de entrada y se observa su salida. La salida se vuelve a alimentar a la red y se ve la nueva salida. Este proceso continua hasta que no hay cambios en la salida.

Uno de los resultados teóricos interesantes es que una red de Hopfield puede almacenar en forma confiable hasta: $0.138N$ ejemplos de entrenamiento (donde N es el número de unidades de la red).

Cada neurona tiene un estado interno u_i y uno externo v_i. Los valores internos son continuos y los externos binarios. La actualización y relación entre estos valores es:

$$u_i(t+1) = \sum_{j=1}^{n} W_{i,j} v_j(t) + I_i$$

$$v_i(t+1) = f(u_i) = \begin{cases} 1 & \text{si } u_i > 0 \\ 0 & \text{si } u_i \leq 0 \end{cases}$$

donde I_i es una constante externa de entrada a la neurona i y $f()$ es la función de transferencia entre los estados internos y externos.

Las neuronas se actualizan es forma aleatoria.

La siguiente función de energía es la que se minimiza, esto es, el mínimo local de la función de energía corresponde con la energía del patrón almacenado.

$$E_d = -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} W_{i,j} v_i v_j - \sum_{i=1}^{n} I_i v_i$$

Las funciones de actualización hacen un gradiente decendiente en la función de energía.

También existe una versión continua de redes de Hopfield..

Para aplicarla a un problema de optimización combinatoria, se tienen que seleccionar pesos y entradas externas que representen en forma adecuada la función a minimizar.

En el caso de TSP, se puede hacer la siguiente formulación:

$$X_{i,j} = \begin{cases} 1 & \text{si la ciudad } i \text{ está en la posición } j \\ 0 & \text{de otra forma} \end{cases}$$

La función objetivo y las restricciones las podemos expresar como:

$$\text{minimizar} \sum_{i=1}^{N} \sum_{k=1,k\neq i}^{N} \sum_{j=1}^{N} d_{ik} X_{ij} (X_{j,j+1} + X_{k,j-i})$$

$$\text{sujeto a:} \sum_{i-1}^{N} X_{ij} = 1 \forall j$$

$$\sum_{i-1}^{N} X_{ij} = 1 \forall j$$

$$\sum_{j=1}^{N} X_{ij} = 1 \forall i$$

$$X_{ij} \in \{0,1\} \forall i,j$$

Para aplicarlo, las restricciones del problema se añaden a la función objetivo. Una posible formulación para N ciudades es:

$$\begin{aligned} E = \quad & \tfrac{A}{2} \textstyle\sum_{j=1}^{N} \sum_{i=1}^{N} \sum_{k=1,k\neq i}^{N} X_{ij} X_{kj} \\ & + \tfrac{B}{2} \textstyle\sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{l=1,l\neq j}^{N} X_{ij} X_{il} \\ & + \tfrac{C}{2} (\textstyle\sum_{i=1}^{N} \sum_{j=1}^{N} X_{ij} - N)^2 \\ & + \tfrac{D}{2} \textstyle\sum_{i=1}^{N} \sum_{k=1,k\neq i}^{N} \sum_{j=1}^{N} d_{ik} X_{ij} (X_{k,j+1} + X_{k,j-1}) \end{aligned}$$

Los primeros dos términos dicen que no puede existir más de un "1" por columna y por renglón respectivamente, el tercer término aseguro que existan N elementos "prendidos" en la matríz de solución y el último término minimiza la longitud del circuito.

En la formulación original de Hopfield y Tank (85) usaron los siguientes valores: $A = B = D = 500$ y $C = 200$.

Para ponerlo en términos de la función de energía de Hopfield, introducimos la función delta de Kronecker:

$$\delta_{ab} = \begin{cases} 1 & \text{si } a = b \\ 0 & \text{si } a \neq b \end{cases}$$

y expresamos la función de energía como:

$$E = \begin{array}{ll} -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{N}\sum_{l=1}^{N} & [-A\delta_{ik}(1-\delta_{jl}) - B\delta_{jl}(1-\delta_{ik}) - \\ & C - D\delta_{ik}(\delta_{l,j+1} + \delta_{l,j-1})]X_{ij}X_{kl} \\ -\sum_{i=1}^{N}\sum_{j=1}^{N}[CN]X_{ij} + \frac{CN^2}{2} & \end{array}$$

Lo que nos deja (ignorando la constante):

$$W_{ijkl} = -A\delta_{ik}(1-\delta_{jl}) - B\delta_{jl}(1-\delta_{ik}) - C - D\delta_{ik}(\delta_{l,j+1} + \delta_{l,j-1})$$

$$I_{ij} = CN$$

Una vez hecha la formulación, se usan valores aleatorios para los estados iniciales y se actualizan los valores internos y externos de acuerdo a las ecuaciones de actualización de $u_i(t+1)$ y $v_i(t+1)$ descritas anteriormente, seleccionando la neurona a actualizar de forma aleatoria.

Cláramente es difícil expresar problemas de optimización e incorporar las restricciones del problema, además que al seguir un gradiente decendiente es fácil de caer en mínimos locales.

7.3 Mapas autorganizados de Kohonen

Es una alternativa para formar *clusters* o grupos, a partir de datos, cuando se conoce el número de grupos a formar.

Se ajustan los pesos de un vector de entrada a nodos de salida organizados en una malla bidimensional (ver figura 7.6).

Se requiere definir una vecindad alrededor de cada nodo, la cual se va reduciendo con el tiempo (ver figura 7.7).

Las entradas se normalizan y se procesan una por una (ver tabla 7.2).

Los resultados dependen del orden de los ejemplos para poco datos.

La aplicación a problemas de optimización se hace normalmente escogiendo una topología en los nodos de salida particular que reflejen la posible solución del problema a resolver.

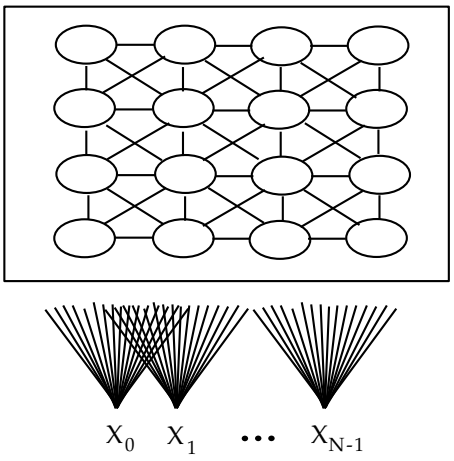

Figura 7.6: Red típica de Kohonen.

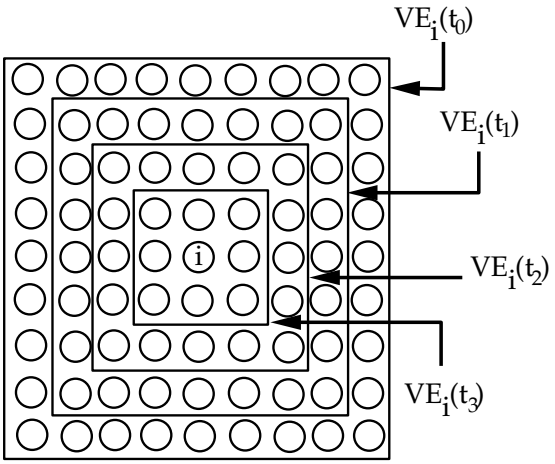

Figura 7.7: Vecindad en una red de Kohonen.

Tabla 7.2: Algoritmo de Kohonen

Inicializa aleatoriamente los pesos con valores bajos de las N entradas a las M salidas.
Dada una nueva entrada:

1. Calcula la distancia entre la entrada y cada nodo de salida j:

$$d_j = \sum_{i=0}^{N-1} (X_i(t) - W_{ij}(t))^2$$

   donde $X_i(t)$ es la i-ésima entrada al tiempo t y W_{ij} es el peso del nodo i al nodo de salida j.

2. Selecciona el nodo con la distancia menor (j^*).

3. Actualiza los pesos del nodo j^* y los de sus vecinos, de acuerdo al esquema de vecindad (ver figura 7.7):

$$W_{ij}(t+1) = W_{ij}(t) + \alpha(t)(X_i(t) - W_{ij}(t))$$

   donde $\alpha(t)$ $(0 < \alpha(t) < 1)$ decrese con el tiempo.

Posiblemente la real ventaja de usar estas redes en lugar de alguna de las metaheurísticas propuestas en la literatura, es su facilidad de ser implementadas en hardware.