CM

CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS DEL IPN

DEPARTAMENTO DE INGENIERIA ELECTRICA

SECCION DE COMPUTACION

MODELOS CONEXIONISTAS DE DEDUCION Y APRENDIZAJE AUTOMATICO

TESIS QUE PRESENTA LA **LIC. MARIA MARGARITA GOIRE CASTILLA**

PARA OBTENER EL GRADO DE

MAESTRA EN CIENCIAS

EN LA ESPECIALIDAD DE:

INGENIERIA ELECTRICA CON OPCION EN COMPUTACION

TESIS DIRIGIDA POR EL DR. GUILLERMO B. MORALES LUNA

MEXICO, D. F. NOVIEMBRE DE 1991

"hay cosas, bien se sabe que persisten,
que detienen la muerte como un muro,
nadie sabe por que beben la impura,
beben la amarga, beben y resisten"

Mirta Aguirre

A mis padres Clara y Suitberto
por amarme siempre y dejarme
volar libremente.

A mi tío Daniel, aunque ya no esté

A mis hermanos Suitberto y Andrés
por su inmenso amor.

AGRADECIMIENTOS

Deseo dejar constancia de mi agradecimiento a la Organización de las Naciones Unidas para la Educación, la Ciencia y la Cultura (U.N.E.S.C.O.) y al Centro de Investigación y de Estudios Avanzados del I.P.N., por la posibilidad que me brindaron para realizar los estudios de maestría y todo el apoyo recibido durante el desarrollo de los mismos.

Así mismo quiero agradecer a todo el personal de la Sección de Computación del Departamento de Ingienería Eléctrica del C.I.N.V.E.S.T.A.V., su colaboración y su solidaridad manifiesta.

En particular, mi más profunda gratitud al Dr. Guillermo Morales Luna por sus orientaciones, sus valiosos consejos, su paciencia y la confianza que depositó en mí.

Gratitud infinita a mis compañeros y amigos mexicanos por su estímulo, su apoyo, su cariño y su solidaridad constantes.

Muchas gracias a mi compañera Patricia Durán y a su esposo Héctor Teherán, por su ayuda desinteresada.

Especialmente quiero manifestar mi agradecimiento a dos amigas incondicionales, Isabel Negrete y Eréndira Correa, y a sus familias, por su apoyo y ayuda, por sus consejos y su constante estímulo, por todo el cariño demostrado y por creer en mí.

Un agradecimiento a las instituciones cubanas que hicieron posible mi estancia en México, al Ministerio de Enseñanza Superior de Cuba (M.E.S), al Instituto Superior Politécnico Julio Antonio Mella de Santiago de Cuba (I.S.P.J.A.M.) y a la Comisión Cubana de la U.N.E.S.C.O. Muy especialmente, al antiguo Departamento de Matemática y Computación del I.S.P.J.A.M. y al actual Departamento de Informática del mismo instituto.

Sencillamente gracias, a mis verdaderos compañeros y amigos cubanos.

INDICE

INTRODUCCION

Los modelos conexionistas de Máquinas de Boltzmann están siendo estudiados por sus capacidades de procesamiento en paralelo, por las propiedades interesantes de su algoritmo de aprendizaje y por constituir una herramienta de optimización.

Nuestro trabajo de tesis está enmarcado precisamente en este estudio, con el objetivo central de realizar un análisis del modelo y proponer una implementación del mismo y de su algoritmo de aprendizaje.

La implementación del algoritmo de optimización recocido simulado (*simulated annealing*), integrado a la máquina de Boltzmann para encontrar los mínimos globales de energía, y también aplicado a un problema de optimización combinatoria, es el otro objetivo de este trabajo.

La tesis se desarrolla en dos de las vertientes de estudio de la máquina de Boltzmann, su algoritmo de aprendizaje y su potencialidad como técnica de optimización combinatoria, cuyos puntos a destacar son los siguientes:

- Análisis de las habilidades de aprendizaje de la máquina de Boltzamnn.

- Análisis del algoritmo *simulated annealing* integrado a la máquina, y de sus parámetros de control, como algoritmo de minimización de energía.

- Análisis de la máquina de Boltzmann como máquina de minimización de energía, y en general como herramienta de optimización.

El trabajo está estructurado en cinco capítulos que abordan los aspectos planteados anteriormente, y cuyas temáticas expondremos a continuación.

El primer capítulo, "**Introducción al Modelo Conexionista**", introductorio sirve para ubicar a la tesis desde el punto de vista histórico y temático.

El segundo capítulo, "**Máquinas de Boltzmann**", presenta la formulación matemática del modelo con nuestra propuesta para realizar las transiciones de estados en la máquina, que define la estrategia a seguir para la localización de mínimos; aspecto de particular importancia para lograr soluciones cercanas al óptimo. El capítulo finaliza con la explicación de como se realizará el reconocimiento de patrones en nuestra implementación de la máquina de Boltzmann.

El tercer capítulo, "**Aprendizaje en una Máquina de Boltzmann**", trata el tema del aprendizaje del modelo y proponemos nuestra variante para que la red aprenda. También se exponen los detalles de nuestra implementación del algoritmo de aprendizaje y las pruebas realizadas para resolver el problema del XOR, es decir, el entrenamiento a que fue sometida la red para que aprenda el problema planteado.

Nuestra implementación del *annealing* para la máquina de Boltzamnn, utilizando como función de aceptación la función estandar del *simulated annealing* constituye un aspecto relevante del capítulo cuatro, "**Instrumentación del algoritmo de recocido simulado**". Se incluyen además, el análisis y la selección de los parámetros de la programación del recocido (*annealing schedule*) para el algoritmo de optimización.

El capítulo cinco "**Optimización**", está dedicado a la resolución del Problema del Agente Viajero, que es un conocido

problema de Optimización Combinatoria, aplicando el algoritmo *simulated annealing*. Aparecen las pruebas realizadas y una comparación de los resultados alcanzados con la solución exacta para un ejemplo de 42 ciudades y para un ejemplo de 12 ciudades.

Los alcances de este trabajo de tesis pueden ser resumidos en tres aspectos fundamentales que se enumeran a continuación :

1 *ANNEALING* PARA *LA MAQUINA DE BOLTZMANN*

Para lograr esta implementación, enfatizamos los siguientes puntos, por considearlos importantes:

1.1.- Estrategia de búsqueda para visitar las configuraciones vecinas.
1.2.- Localización de los mínimos locales por configuraciones vecinas.
1.3.- Generación de configuraciones aleatorias para pasar a otro mínimo.
1.4.- Realización de pruebas sobre las modificaciones planteadas.

2 ALGORITMO DE APRENDIZAJE DE LA *MAQUINA DE BOLTZMANN*

2.1.- Aplicación del *annealing* definido anteriormente.
2.2.- Reconocimiento patrones por completamiento.

3 OPTIMIZACION

3.1.- *Annealing* y el problema del Agente Viajero.
3.2.- *Annealing schedule* para el Agente Viajero.

Todas las implementaciones y programas fueron desarrolla dos en el lenguaje de programación C, con el compilador Turbo C, versión 2.0 de Borland y en C standard de la ANSI.

CAPITULO 1
INTRODUCCION AL MODELO CONEXIONISTA

1.1 CONEXIONISMO Y REDES NEURONALES

La integración de los procesos de percepción e inducción con el conocimiento, es una de las principales dificultades para los diseñadores de sistemas inteligentes. Los esquemas de procesamiento de información parecen ajenos a las operaciones que se realizan a bajo nivel para traducir entradas sensoriales a descripciones simbólicas, o a las características de flexibilidad y repetición de la adquisición de conocimiento. Los métodos del **Conexionismo** constituyen un enfoque para modelar el conocimiento que brinda una prometedora alternativa de solución a este tipo de problema.

El **Conexionismo**, puede considerarse como un **Procesamiento Masivo Paralelo** y proporciona modelos de **Redes Neuronales**, como un intento por crear un cerebro artificial. Sin embargo, dada su etapa actual de desarrollo y su habilidad para inferir e intuir a partir de información incompleta o confusa, sería más correcto describirlo como un intento de simular la forma en que el cerebro humano realiza diversas actividades.

En los últimos años ha resurgido el concepto de conexionismo relacionado con las redes o modelos neuronales. El creciente interés en el llamado enfoque conexionista, es decir, el estudio de formas con las cuales unidades simples puedan interconectarse para resolver problemas difíciles; está dado por el hecho de que la ciencia cognoscitiva busca un mayor contacto con la teoría del cerebro. Esta propuesta está desarrollada por analogías con las redes neuronales, y no por el intento de modelar estructuras neuronales reales en ningún

detalle. Originalmente se refería a la formación de conexiones asociativas directas entre los lugares donde se originan los estímulos (entradas) y donde se producían las respuestas (salidas).

Se debe puntualizar que la modelación neuronal se enfoca a modelar circuitos neuronales específicos de los cerebros reales y está basada en datos experimentales del cerebro. Por el contrario, el conexionismo es la modelación inspirada neurológicamente o la modelación en el estilo del cerebro, tomando los principios de computación empleados por éste.

Los modelos conexionistas son considerados por algunos investigadores, una reciente subclase de las redes neuronales que acentúan más el poder computacional que la fidelidad biológica de los mismos. Su meta es tratar de capturar los principios de cómputo del cerebro y el principal interés ha sido el estudio de los "algoritmos de aprendizaje" para generar una red "útil", a través de los cambios de las conexiones entre las neuronas, tal que los algoritmos de aprendizaje juegan dos papeles distintos, como modelos del aprendizaje humano y como medios para programar redes que satisfagan actuaciones específicas.

Los modelos conexionistas tienen la característica de que cada nodo o "neurona" de la red tiene un significado diferente. Uno de los objetivos que persiguen los modelos conexionistas es desarrollar la habilidad para resolver estructuras relacionales, implícitamente definidas, determinadas en una forma similar a como están constituídas las soluciones de los sistemas de ecuaciones algebraicas.

En las estructuras relacionales, el conocimiento almacenado constituye el universo de variables en el cual deben

buscarse las soluciones, mientras que las condiciones que se expresan a través de las relaciones, corresponden a las ecuaciones.

En los modelos conexionistas las relaciones binarias que existen entre los datos discretos permiten identificar las conexiones presentes entre las neuronas.

1.2 DESARROLLO HISTORICO

La teoría de las redes neuronales nace en los años cuarenta con el trabajo de Mc Culloch y Pitts [Ar87], [CP43], en donde se concibe al cerebro como una computadora integrada por elementos computacionales bien definidos, las neuronas. Su gran mérito consiste en que no se trata de un modelo fisiológico de una célula nerviosa real, sino de la representación lógica de los puntos más significativos del funcionamiento de aquella. Estos investigadores proponen un modelo formal de la neurona como una unidad de umbral lógico.

Definición de una neurona de Mc Culloch-Pitts

Una neurona es un elemento de m entradas $x_1,\ldots,x_m$ ($m \geq 1$) y una salida y. Está caracterizada por m+1 números, sus umbrales θ y sus pesos $w_1,\ldots,w_m$ donde w_i están asociados a x_i. Considerando que la neurona opera en una escala de tiempo $n = 1,2,3,4,\ldots$, el disparo (la activación) de su salida para el tiempo n+1 está determinado por los disparos de sus entradas para el tiempo n de acuerdo a la siguiente regla: la neurona dispara un impulso a través de su axón para el tiempo n+1 si la suma pesada de sus entradas para el tiempo n excede el umbral de la neurona, lo que podemos expresar simbólicamente como:

$$y(n+1) = 1 \quad \text{si y sólo si} \quad \sum_i x_i(n) > 0$$

En términos del modelo de McCulloch-Pitts de una neurona, se puede definir el primer modelo de red neuronal como una colección de neuronas de McCulloch-Pitts, cada una con la misma escala de tiempo, interconectadas de forma tal que las salidas de algunas neuronas forman las entradas de otras y con una entrada a la red bien definida.

En 1949, una importante contribución es presentada por Hebb [He49], [Ar87]. El proporciona un esquema de aprendizaje para las neuronas formales de McCulloch-Pitts, en el cual la conexión entre dos neuronas se refuerza si ambas están disparadas (activadas) simultáneamente.

Trás el desarrollo de los primeros modelos y de sus extensiones naturales, aparecen, casi simultáneamente, en los años cincuenta, dos trabajos enfocados a la síntesis de elementos clasificadores lineales. Estos son las adalinas de Widrow [WH60], [WS85], [Ar87] y los perceptrones de Rosenblatt [Ro59], [Ar87].

El esquema del perceptrón es un modelo significativamente diferente de las redes neuronales. La principal diferencia es no asumir que la función de una neurona es fija todo el tiempo, en su lugar, permite que los pesos de cada neurona cambien con el tiempo, con el propósito de que la red pueda autocambiarse con el tiempo, de una forma similar a como lo hace el aprendizaje por experiencia.

El grupo de Rosenblatt trabajó en el problema de como encontrar pesos apropiados de las conexiones para realizar tareas particulares, como resultado de sus investigaciones

proponen las redes llamadas perceptrones, en las cuales las unidades fueron organizadas en niveles con conexiones *feed-forward* entre niveles. Para clases simples de perceptrones sin ningún nivel intermedio, Rosenblatt probó la convergencia de un algoritmo de aprendizaje, es decir, una forma de cambiar los pesos iterativamente, tal que se realice un cálculo deseado.

En 1969, Minsky y Papert [MP69], [Ar87], [Am90] describieron con más detalle el principio de funcionamiento de los perceptrones, mostrando la limitación de los mismos por tener una capa simple de elementos de procesamiento, ya que el teorema de aprendizaje se aplica solamente a aquellos problemas para los cuales la estructura de la red es capaz de calcular.

Minsky y Papert mostraron que algunas computaciones muy simples no podrían realizarse con un perceptrón de un nivel, como es el caso de la resolución del problema del or exclusivo (XOR), que no puede ser resuelto por este tipo de red.

Hasta aquí hemos mencionado algunos de los hechos más destacados que determinaron los modelos neuronales básicos. Esta primera etapa alcanzó niveles teóricos que por falta de tecnología no se concretaron en realizaciones prácticas significativas, pero que constituyeron la base de los proyectos actuales.

El paralelismo y el procesamiento distribuído aparecen como una realidad con J. Feldman [Fe81], [FB82], [Ar87], el primer exponente del uso de representaciones *localist*. Feldman expuso su estilo *localist* en el cual los conceptos individuales son asociados con nodos individuales en la red y propuso junto a D. Ballard unas redes conexionistas para procesamiento visual.

En esta misma línea de trabajo, Rumelhart y McClelland [RHW86],[RHWi86], [Ar87], son dos investigadores claves en el desarrollo de las explicaciones psicológicas en términos de redes conexionistas. Ellos han mostrado como los fenómenos cognoscitivos pueden emerger de patrones de interacción entre unidades elementales de procesamiento, en el cual la computación está basada en la interacción paralela de muchas unidades pequeñas.

Barto [BSB81], [BSA83], [BA85], [Ar87], ha usado el aprendizaje de refuerzo para proporcionar esquemas de aprendizaje a redes de multiniveles, resolviendo una de las principales críticas hechas a las redes de aprendizaje por Minsky y Papert. Muestra como extender los algoritmos de aprendizaje de redes de nivel simple o de un solo nivel a redes de gran complejidad.

El algoritmo de Retropropagación (*Back-Propagation*), propuesto por Rumelhart, Hinton y Williams [RHW86], [SR86], [JH87], [Hi89], en 1986, proporcionó otro enfoque a los algoritmos de aprendizaje de redes multiniveles, muy popular actualmente, y que trabaja muy bien para ajustar los pesos de las unidades conectadas en capas sucesivas.

Hopfield [Ho82], [AJ85], [Hi85], [HT86], [Li87], [Ar87], [EPRV87], [ANF 90], presentó su diseño en 1982. Básicamente se trata de un red neuronal específica para el reconocimiento de imágenes, cuyo principal mérito reside en el algoritmo de identificación de las entradas. Hopfield vió la dinámica de una red simétrica como la minimización de una medida de energía global para intentar asumir el cálculo de una red, en términos de los sistemas de energía dinámica.

Carpenter y Grossber, [Gr75], [Ar87], [Li87], han desarrollado la denominada teoría de la resonancia adaptativa (*ART*), con la cual se diseña la conectividad de una red tipo Hopfield, pero modificada de forma que es capaz de generar clasificaciones entre las entradas y obedece a un mecanismo de aprendizaje sin supervisión.

Uno de los autores más significativos de estos años es T. Kohonen, [Ar87], [Li87], [AM90], quien ha formulado de una manera útil la definición analítica del funcionamiento de las neuronas artificiales y su conectividad en la red. Su planteamiento indica que la actividad de una neurona no es un valor binario sino, por el contrario, una magnitud analógica, que vendrá dada por un número real.

Sejnowski [Ar87],durante los últimos años de la década del 70 examinó las consecuencias del ruido y la variabilidad aleatoria en el sistema nervioso. En 1981, propuso un análisis estocástico formal de los modelos de redes neuronales dinámicas no lineales. La Máquina de Boltzmann, de Hinton y Sejnowski (1983) [AHS85], [Hi85], [HS86], [Hi89], [AM909], extiende las redes de Hopfiel al añadir una "temperatura", elemento con dependencia estocástica, que permite el uso del *simulated annealing*. El *simulated annealing* decrementa un término de ruido, el cual es reducido lentamente de acuerdo a cambios de estado de la red, esto sirve como técnica para escapar de los mínimos locales.

Kirkpatrick, Gelatt y Vecchi [KGV82], [Ki84] fueron los primeros en proponer y demostrar como aplicar los técnicas de simulación de la Física Estadística para resolver problemas de Optimización Combinatoria. Desde que introdujeron los conceptos de *annealing* en este campo, muchos esfuerzos han sido dedicados a investigar la teoría y la aplicación *simulated*

annealing a la resolución de muchos problemas. Este algoritmo está basado en técnicas de Monte Carlo, aplicando el algoritmo de Metropolis de la Física Estadística [MRT53]. Es un algoritmo estocástico apropiado para la solución aproximada de problemas complejos de optimización discreta.

Uno de los ejemplos clásicos de un problema complejo de Optimización Combinatoria, es el bien conocido problema del viajante o agente viajero. Aunque fácil de formular, la solución es difícil de encontrar debido a que pertenece a la clase de problemas NP-completos. Consiste en construir el viaje más corto entre N ciudades predefinidas, que el agente debe visitar sólo una vez. El uso del *simulated annealing* en la resolución de esta tarea se presenta como una opción interesante.

1.3 BASES DEL MODELO CONEXIONISTA

Los modelos conexionistas son una abstracción del procesamiento de información que realizan las neuronas biológicas. Por esto, la premisa fundamental del conexionismo es que las neuronas individuales no transmitan grandes cantidades de información simbólica, en su lugar, ellas calculan información parcial.

Los modelos de redes conexionistas están compuestos por muchos elementos computacionales no lineales, operando en paralelo y arreglados en patrones que son reminiscencias de redes neuronales biológicas. Los nodos o elementos computacionales están conectados a través de pesos, que son típicamente adaptados.

Los modelos de redes neuronales son especificados por:

- la topología de la red
- características de los nodos
- entrenamiento o reglas de aprendizaje

TOPOLOGIA DE LA RED

La topología de la red está determinada por la conexión entre los nodos.

Una red neuronal es un grafo, cuyos nodos (elementos de procesamiento) son llamados neuronas o unidades, están conectados por arcos usando conexiones pesadas.

El comportamiento de una red depende fuertemente de la forma en que las neuronas estén conectadas. En los principales modelos, las neuronas están agrupadas en niveles de tal manera que la salida de cada neurona en un nivel está completamente interconectada con las entradas de todas las neuronas en el siguiente nivel.

En una red *feedforward*, las neuronas en un nivel dado no tienen entradas de los niveles subsecuentes o de los niveles previos a los niveles inmediatamente anteriores. Así, las neuronas en una red *feedforward* no están conectadas cada una con las otras. Por ejemplo, la red de retropropagación tiene tres niveles: entrada, oculto y salida. Los modelos *feedback* son los que incluyen conexiones de las salidas de un nivel a las entradas del mismo o de un anterior nivel.

CARACTERISTICAS DE LOS NODOS

Los elementos computacionales o nodos, unidades , células o neuronas son típicamente análogos y no lineales. Están conectados por enlaces con pesos generalmente variables.

Cada elemento de procesamiento tiene solamente una salida. Estas salidas fluyen en todas direcciones para interconectarse con otros procesadores elementales. Cada nodo procesa la señal de entrada basado en los valores de las constantes almacenadas en él. Una neurona calcula su salida encontrando la suma pesada de sus entradas.

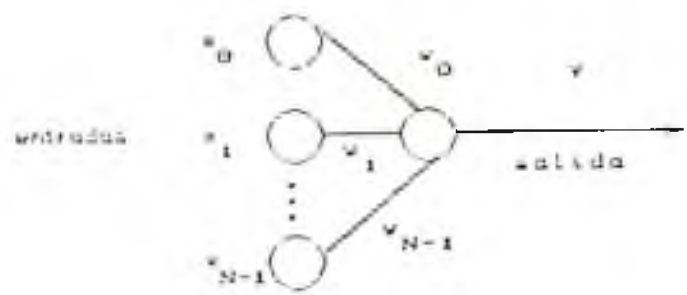


figura 1

Cada unidad tiene un estado o nivel de actividad que es determinado por la entrada recibida de las otras unidades de la red. La función de activación suma las entradas pesadas para producir un valor de activación.

En los principales modelos, las señales de entrada pueden ser excitatorias o inhibitorias; su valor es pasado a través de una función de salida o función de transferencia, que produce la salida actual para la neurona tratada, como se muestra en la figura 1.

La función de transferencia, o de activación de una neurona, se define como el valor de activación que está saliendo. Los primeros modelos usaron una función de transferencia lineal. Sin embargo, ciertos problemas no son reducibles a métodos puramente lineales. La función de transferencia de

umbral es la más simple de los modelos no lineales. Esta es una función todo o nada, si la entrada es más grande que una cantidad fijada (el umbral), la neurona pondrá su salida en 1; si el valor es menor que el umbral, pondrá su salida en 0.

A veces la función de transferencia es una función de tipo saturación: Mayor excitación por encima de algún nivel máximo de "disparo" (activación), no tiene efecto adicional. Una función de transferencia particularmente útil es la llamada función sigmoide o función logística, la cual tiene un límite de saturación alto y uno bajo, y un rango de proporcionalidad entre ellos. Esta función es cero cuando el valor de activación es un número negativo grande. La función sigmoide es uno cuando el valor de activación es un número grande positivo y hace una suave transición entre ellos.

Formas más usadas de la función no lineal de la entrada total:

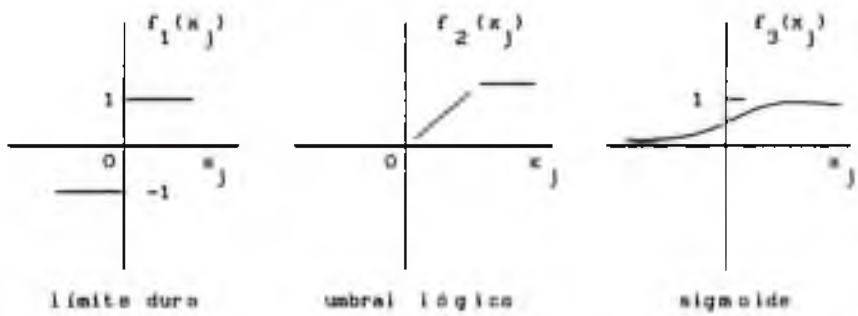


figura 2

Existen muchas posibles variaciones de este esquema de trabajo. Una suposición es que los efectos combinados del resto de la red, en la j-ésima unidad, son mediados (reconciliados) por una cantidad escalar x_j. Esta cantidad, llamada la

"entrada total" de la unidad j, es usualmente tomada como una función lineal de los niveles de actividad de las unidades que proporcionan la entrada a j:

$$x_j = -\theta_j + \sum_i y_i w_{ji}$$

donde

y_i :estado de la i-ésima unidad

w_{ji} :peso de la conexión de la j-ésima a la i-ésima unidad

θ_j :umbral de la j-ésima unidad

El estado de una unidad es una función no lineal de su entrada total, ver figura 2. Para unidades con estados discretos, esta función toma valor 1 si su entrada total es positiva y valor 0 (ó -1) en cualquier otro caso. Para unidades con estados continuos, una función típica de entrada-salida es la función logística:

$$y_j = \frac{1}{1 + e^{-x_j}}$$

ENTRENAMIENTO O REGLAS DE APRENDIZAJE

Las reglas especifican un conjunto inicial de pesos y como éstos deben ser adaptados durante el proceso de aprendizaje para mejorar la actuación de la red.

Una red neuronal aprende por adaptación mediante cambios en la entrada, provocando modificaciones en los pesos, tal que la red gana en experiencia. La regla de aprendizaje es el corazón de una red neural y determina como los pesos deben ser ajustados

El aprendizaje de una red neuronal se realiza en una de las siguientes formas: supervisado, no supervisado o sin supervisión y autosupervisado.

El **aprendizaje supervisado** ocurre cuando se proveen entradas de prueba y error, enseñando a la red respuestas correctas e incorrectas. Este aprendizaje es realizado en base a la comparación directa de la salida de la red con las respuestas correctas conocidas. También es llamado aprendizaje con maestro.

En el **aprendizaje sin supervisión**, no existe maestro. A veces la meta del aprendizaje no está definida en todos los términos de los ejemplos epecíficos correctos y la información disponible está solamente en la correlación del dato de entrada, es por esto que la red está esperando crear categorías de estas correlaciones y producir señales de salida correspondientes a una categoría de entrada. La red debe autodescubrir patrones, características, regularidades correlaciones o categorías en la entrada de datos y codificarlos para dar la salida.

El **aprendizaje autosupervisado** ocurre cuando la red se monitorea y corrige errores en la representación interna de la información por retroalimentación a través de la red.

En un modelo conexionista todo el conocimiento de término largo (*long-term knowledge*) está codificado en las conexiones o por sus pesos, así el aprendizaje consiste en cambiar los pesos o añadir o eliminar conexiones. El conocimiento de término corto (*short-term knowledge*) de la red está normalmente codificado por los estados de las unidades, pero algunos modelos también tienen pesos que pueden ser usados para codificar contextos temporalmente.

CAPITULO 2
MAQUINAS DE BOLTZMANN

2.1 FORMULACION MATEMATICA

La máquina de Boltzmann tiene su origen en la teoría de redes neuronales propuesta por Hopfield [Ho82], y fue introducida por Hinton y otros [AHS85], [HS86]. Una máquina de Boltzmann consiste de una red de elementos simples de procesamiento llamados **unidades**, **nodos** o "**neuronas**" indistintamente por varios autores, nosotros las llamaremos **unidades**. Estas unidades, en cada instante de computación de la máquina de Boltzmann, pueden tomar uno de dos estados discretos, prendido o apagado (*on* u *off*).

Entre las unidades existen **conexiones**. A cada conexión está asociado un valor que se conoce como **peso sináptico**, nos referiremos a él como **peso** simplemente. Un peso nos informa de cuan deseable es que las dos unidades conectadas estén en *on*. Un peso positivo indica que las unidades conectadas por él están en *on* simultáneamente, en tanto que uno negativo indica que ellas deben tener estados diferentes.

Para cada unidad distinguiremos un conjunto de unidades **vecinas**, conectadas a ella, mediante una conexión directa.

Una **configuración** de la máquina de Boltzmann está definida por los estados de todas sus unidades individuales.

En el conjunto de todas las configuraciones posibles se define una función, llamada indistintamente en la literatura, **función de energía**, **Hamiltoniano**, **función de costo**, de concenso e incluso **función de Liapunov**, y que será llamada por nosotros **función de energía**. Esta función de energía nos permite

distinguir configuraciones de energía mínima, las cuales indican que las restricciones implícitas en la máquina han sido logradas mayormente. Precisamente diremos que estas configuraciones son las aceptadas o reconocidas por la máquina. A partir de una configuración inicial, la máquina transita de una a otra configuración hasta lograr configuraciones aceptables.

Para pasar a una nueva configuración se define un mecanismo de transición. En cada transición, el estado de una unidad en la nueva configuración está determinado por una función estocástica de sus unidades vecinas en la configuración anterior.

A continuación, basados en una formulación de la teoría de gráficas, presentamos una definición formal de la estructura de una máquina de Boltzmann.

Una máquina de Boltzmann se define como una gráfica no dirigida $G = (U,C)$, donde $U = \{u_0,\ldots,u_{N-1}\}$ es el conjunto de unidades y $C \subseteq U \times U$, es el conjunto de conexiones entre las unidades. Una conexión $\{u_i,u_j\} \in C$, conecta a las unidades u_i y u_j. La estructura G se llamará también red de la máquina.

El conjunto de unidades U es la unión de dos conjuntos disjuntos M y H, donde M es el conjunto de unidades **visibles (o ambientales)** y H es el conjunto de unidades **ocultas (o intermedias)**. A su vez el conjunto de unidades visibles está dividido en un conjunto de unidades de **entrada** E y un conjunto de unidades de **salida** S. Las unidades visibles son la interfaz entre la red y el ambiente.

A cada unidad $u_i \in U$ se le asocia un valor $s_i \in I$, el cual es su **estado**. En nuestra formulación, el conjunto de valores I se considera como el conjunto $I = \{0,1\}$ que corresponde natu-

ralmente a los estados *on* y *off* de las unidades.

Una **configuración** de la máquina de Boltzmann es una función $S: U \longrightarrow I$. Escribiremos $\forall\, u_i \in U: s_i = S(u_i)$.

Al conjunto de todas las configuraciones posibles lo denotamos por $EC = I^N$, y es llamado **espacio de configuraciones**. Se tiene que $|EC| = 2^N$ donde N es el número de unidades.

Una **vecindad** $V \subset EC$ de una configuración $S \in EC$ está definida como el conjunto de configuraciones que pueden ser obtenidas a partir de la configuración S al cambiar el estado de una de las unidades, es decir, una configuración $SV \in EC$ es **vecina** de S si es obtenida a partir de S al cambiar el estado de una unidad u_i, dejando inalterados los estados de las unidades restantes. Formalizando:

$\forall\, S \in EC$, una configuración $SV \in EC$ es vecina de S si $\exists\, i = 1,\ldots,N$, tal que

$$SV_j = S_j \quad \forall j \neq i$$

$$SV_i = \begin{cases} 1 \text{ si } S_i = 0 \\ 0 \text{ si } S_i = 1 \end{cases} \qquad (2.1)$$

En este caso escribiremos que $SV = S^{(i)}.S^{(i)}$ coincide pues con S salvo en la unidad i.

A cada conexión $\{u_i, u_j\}$ se le asocia un valor que llamaremos **peso**, y que denotaremos w_{ij}. Se tiene así una función $W: C \longrightarrow \mathbb{R}$, donde $\forall\, i,j: \{u_i, u_j\} \longmapsto w(u_i, u_j) = w_{ij}$.

Si $w(u_i, u_j) > 0$, la conexión se dice ser **excitatoria** y si $w(u_i, u_j) < 0$ es llamada **inhibitoria**.

A cada configuración se le asocia un valor de energía, que es una función $E: EC \longrightarrow \mathbb{R}$, $S \longmapsto E(S)$ definida como

$$E(S) = -\sum_{i<j} w_{ij}\, s_i s_j \qquad (2.2)$$

donde

$w_{ij} \in \mathbb{R}$ es el peso de la conexión entre las unidades u_i y u_j

$S_i \in I$ es el estado de la unidad u_i

$S_j \in I$ es el estado de la unidad u_j

Introduzcamos las siguientes notaciones:

$S^{(i,0)}$: es la configuración que coincide con S en todas las unidades salvo posiblemente en la i-ésima donde tendrá el estado cero.

$S^{(i,1)}$: es la configuración que coincide con S en todas las unidades salvo posiblemente en la i-ésima donde tendrá el estado uno.

En cada unidad i, la diferencia de energía en esa unidad es:

$$\Delta E(i,S) = E(S^{(i,1)}) - E(S^{(i,0)}) = \sum_k w_{ik}\, s_i s_k \qquad (2.3)$$

abreviadamente escribiremos, $\Delta E_i = \Delta E(i,S)$.

Algunos autores introducen la noción de **umbrales** en unidades, los cuales son números reales y, vistos globalmente definen una función de *umbral* $\theta: U \longrightarrow \mathbb{R}$, $u_i \longmapsto \theta_i$.

Manejando umbrales, las expresiones de la función de energía y de la diferencia de energía dan las ecuaciones siguientes:

$$E(S) = -\sum_{i<j} w_{ij}\, s_i\, s_j + \sum_i \Theta_i s_i \qquad (2.4)$$

$$\Delta E_i = \sum_k w_{ik}\, s_i\, s_k - \Theta_i \qquad (2.5)$$

Si utilizamos una unidad extra que siempre tenga estado 1 y que esté conectada a todas las demás unidades de la red, entonces el enfoque de umbrales coincide con el anterior. En tal caso se sustituirían los umbrales por los pesos asignados a las conexiones establecidas con la unidad extra.

Hacemos pues una extensión como sigue:

1.- Definimos una nueva red $G(U_{Nueva}, C_{Nueva})$ donde

$U_{Nueva} = U \cup \{\text{unidad extra}\}$ y

$C_{Nueva} = C \cup \{\ \{u, \text{unidad extra}\} \mid u \in U\ \}$

Convendremos en que el estado de la unidad extra será siempre $s_0 = 1$.

2.- La nueva función de pesos será $W_{Nueva}: C_{Nueva} \to \mathbb{R}$, donde $w(u_i, u_j) = w_{ij}$ y $w(u, \text{unidad extra}) = -\Theta_i$.

3.- Podemos identificar a EC como un subconjunto de EC_{Nueva} mediante la función $C: EC \longrightarrow EC_{Nueva}$, $S \longmapsto S_{Nueva}$, donde $s_{Nueva}(u) = s(u)$ $\forall\, u \in U$ y $s_{Nueva}(\text{unidad extra}) = 1$.

Entonces podemos concluir que $E(S) = E(C(S))$, y trabajar con las ecuaciones de energía y de diferencia de energía sin explícitamente utilizar los umbrales de las unidades.

Un aspecto a destacar en la máquina de Boltzmann es que la función de energía da origen a una distribución de probabilidad en el espacio de configuraciones, que es la llamada **distribución de Boltzman:**

$$P(S) = \frac{e^{-E(S)/T}}{Z(E,T)} \qquad (2.6)$$

donde $T \in \mathbb{R}^+$ es un parámetro que actúa como la temperatura en sistemas físicos

$$Z(E,T) = \sum_{1 \in EC} e^{-E(1)/T}$$

es una constante de normalización para asegurar que la suma de las probablidades sea igual a uno

Cada distribución de probabilidad en *EC* puede ser interpretada como una probabilidad de **hallar a la máquina en una configuración particular.**

Consideremos la siguiente definición:

Definición: P es una de distribución de **probabalidad uniforme** en $A \subset EC$, si

$$P(a) = \begin{cases} \frac{1}{|A|} & \text{si } a \in A \\ 0 & \text{si } a \notin A \end{cases}$$

donde $|A|$ es la cardinalidad del conjunto A.

Para pesos fijos, esta distribución de probabilidad tiende a ser una distribución de probabilidad uniforme en el espacio de configuraciones cuando $T \longrightarrow +\infty$, en tanto que tiende a ser una distribución de probabilidad uniforme en el conjunto formado por las configuraciones de energía mínima $EC_{min} \subset EC$ cuando $T \longrightarrow 0$.

En efecto, enunciemos el siguiente teorema:

Teorema: Dada la distribución de Boltzmann

$$P(S) = \frac{e^{-E(S)/T}}{Z(E,T)}$$

$$\text{donde} \qquad Z(E,T) = \sum_{1 \in EC} e^{-E(1)/T}$$

se tiene que

1) la distribución de probabilidad tiende a ser uniforme en el espacio de configuraciones *EC* cuando $T \longrightarrow \infty$, es decir,

$$\lim_{T \longrightarrow \infty} P(S) = \frac{1}{|EC|} \quad \text{si } S \in EC$$

donde $|EC|$ es la cardinalidad del espacio de configuraciones *EC*.

2) la distribución de probabilidad tiende a ser uniforme en $EMin \subset EC$, conjunto de configuraciones de energía mínima, cuando $T \longrightarrow 0$, es decir,

$$\lim_{T \longrightarrow 0} P(S) = \begin{cases} \frac{1}{|EMin|} & \text{si } S \in EMin \\ 0 & \text{en otro caso} \end{cases}$$

donde $|EMin|$ es la cardinalidad del conjunto de configuraciones de energía mínima *EMin*.

Demostración

Para demostrar el primer punto recordemos que

$$\forall\, x \in \mathbb{R}^{*} \quad \lim_{T \longrightarrow \infty} e^{-x/T} = 1$$

Se cumplen entonces las igualdades siguientes

$$\lim_{T \longrightarrow \infty} P(S) = \lim_{T \longrightarrow \infty} \frac{e^{-E(s)/T}}{Z(E,T)} = \lim_{T \longrightarrow \infty} \frac{e^{-E(s)/T}}{\sum_{i \in EC} e^{-E(i)/T}}$$

$$= \lim_{T \longrightarrow \infty} \frac{1}{\sum_{i \in EC} e^{-E(i)/T}} = \lim_{T \longrightarrow \infty} \frac{1}{|EC|} = \frac{1}{|EC|}$$

Para demostrar el segundo punto, recordemos que

$$\forall\, x \in \mathbb{R}^{*} \quad \lim_{T \longrightarrow 0} e^{-x/T} = \begin{cases} 0 & \text{si } x > 0 \\ 1 & \text{si } x = 0 \end{cases}$$

Consideramos que E0 es un valor de enrgía mínima, y entonces planteamos que:

$$\lim_{T \longrightarrow 0} P(S) = \lim_{T \longrightarrow 0} \frac{e^{-(E(s)-E0)/T}}{\sum_{i \in EC} e^{-(E(i)-E0)/T}}$$

$$= \lim_{T \longrightarrow 0} \frac{1}{\sum_{i \in EC} e^{-(E(i)-E0)/T}} \quad \lim_{T \longrightarrow 0} \frac{1}{|EMin|} = \frac{1}{|EMin|} \blacksquare$$

El objetivo en una máquina de Boltzmann es alcanzar una configuración globalmente mínima, o lo que es lo mismo, una configuración cuya energía sea mínima pues como ya se dijo tales configuraciones son las que realizan mayormente las restricciones de la máquina.

Las configuraciones de energía mínima, que a su vez son las de mayor probabilidad, serán llamadas en lo sucesivo de **equilibrio.**

Para alcanzar las configuraciones de equilibrio se hace necesario transitar consecutivamente en *EC*. Se introduce pues un mecanismo de transición de configuraciones, el cual conlleva otro de transición entre estados de unidades que permite cambiar los estados de las unidades de cero a uno y viceversa.

El cambio de estados se hace mediante un proceso probabilístico. La probabilidad de que una unidad u_i acepte una transición de estado es seleccionada de acuerdo al valor

$$p_i = \frac{1}{1 + e^{-\Delta E_i / T}} \qquad (2.7)$$

ΔE_i es la diferencia de energía al cambiar el estado de la unidad u_i

$T \in \mathbb{R}^+$ es un parámetro de control que juega el mismo papel que la temperatura en los sistemas físicos

La actualización del estado de la unidad u_i se hace de acuerdo a la siguiente regla:

$$s_i = \begin{cases} 1 & \text{si } p_i > 0.5 \\ 0 & \text{en otro caso} \end{cases} \qquad (2.8)$$

Con este nuevo estado para la unidad i tendremos una configuración vecina $S^{(i,s_i)}$ de una configuración dada S. Diremos que $S^{(i,s_i)}$ es **aceptable** desde la configuración S.

Una configuración vecina aceptable por la transición de estado de la unidad u_i, puede ser la configuración a la que se transita a partir de S. La decisión de transitar a ella se hace considerando todavía el algoritmo de optimización conocido como **recocido simulado** o **revenido simulado** (*simulated annealing*) que entre otras cosas ha de evitar, en general, que la máquina de Boltzmann quede atrapada en un mínimo local cuando se hace la búsqueda de configuraciones de equilibrio. Estos aspectos serán vistos con más detalle en el capítulo cuatro de este trabajo.

Como un concepto importante para desarrollar los métodos de localización de configuraciones de equilibrio, presentaremos el de **probabilidades marginales** en configuraciones para máquinas de Boltzmann.

Se mencionó anteriormente que el conjunto de unidades de la máquina de Boltzmann está dividido en un conjunto de unidades visibles y un conjunto de unidades ocultas tales que $U = M \cup H$. Podemos considerar entonces, que $EC = I^N$ es el producto cartesiano de I^M y de I^H, y que cada elemento $S \in EC$ puede ser escrito como el par (α,β) con $\alpha \in I^M$ y $\beta \in I^H$. Al espacio de configuraciones asociado al conjunto M de unidades visibles, lo denotaremos por $EM \subset EC$, donde $|EM| = 2^M$.

Podemos hablar de una distribución de probabilidad en EM, que denotaremos por $MARG(P,M)$, como la distribución de probabilidad marginal de las unidades $s_i \in M$, inducida por la distribución de Boltzmann P, dada por

$$P^{-}(\alpha) = \sum_{\beta \in I^{H}} P(\alpha,\beta) \qquad \text{para } \alpha \in I^{M}$$

abreviadamente la denotaremos como

$$P^{-}_{\alpha} = \sum_{\beta \in I^{H}} P_{\alpha\beta} \qquad \text{para } \alpha \in I^{M}$$

Podemos decir entonces, que P^{-} es la probabilidad de encontrar a las unidades visibles en el estado α, independientemente del valor que tome β, es decir, de los estados de las unidades ocultas.

La distribución de Boltzmann y la distribución de probabilidad marginal $MARG(P,M)$ de las unidades $s_i \in M$, inducida por la distribución de Boltzmann en el espacio de configuración $EM = I^{M}$, son de particular interés, pues juegan un papel fundamental en el algoritmo de aprendizaje que será presentado en el capítulo tres de este trabajo de tesis.

2.2 SEMANTICA DE UNA MAQUINA DE BOLTZMANN

Cuando trabajamos con un problema concreto, la estructura del modelo de la máquina de Boltzmann puede ser interpretada. La red puede modelar un problema considerando que el conjunto de hipótesis es representado por los estados de las unidades, asumiendo que una hipótesis es aceptada si el estado de la unidad es *on* y rechazada si su estado es *off*.

Las conexiones entre las unidades representan relaciones de dependencia entre las hipótesis o restricciones entre estas hipótesis, por lo que los pesos asignados a estas conexiones serán una medida de la dependencia entre estas hipótesis. Un peso positivo indica que dos hipótesis tienden a apoyarse una a la otra. Contrariamente, un peso negativo sugiere que las hipótesis no deben ser aceptadas simultáneamente.

La energía de una configuración debe ser interpretada como una medida de qué tanto, una determinada combinación de hipótesis viola las restriciones implícitas del problema. Por lo tanto, minimizando la función de energía, el sistema evoluciona hacia interpretaciones de la entrada que satisfagan las restricciones del problema.

Las unidades individuales pueden ser tomadas para minimizar la energía. Si algunas de las unidades son externamente forzadas en estados particulares que representan una entrada particular, entonces, el sistema encontrará una configuración de energía mínima que sea compatible con esa entrada.

Cuando los pesos de una máquina de Boltzmann están fijos, ella puede ser vista como un dispositivo que calcula una función constante en el caso de que exista un mínimo de la función de energía, dejando variar libremente la red, o puede

ser vista como un dispositivo que calcula una función de una variable aleatoria, en el caso de que existan varios mínimos.

También puede considerarse como un dispositivo que calcula una función en el sentido de completar los estados de las unidades ocultas y de salida cuando fijamos los estados de las unidades de entrada, para buscar una configuración de equilibrio donde las unidades de salida tengan los valores correctos en dependencia de las entradas dadas. Ya que la localización de mínimos para la función de energía corresponde al cálculo de máximos para las distribuciones de probabilidad, esto último puede realizarse mediante el cálculo de máximos para las distribuciones de probabilidad marginales.

2.3 ESTRATEGIA PARA LA LOCALIZACION DE MINIMOS

La aplicación de algoritmos para la solución de problemas de optimización combinatoria (como es el caso de buscar una configuración de energía mínima en el espacio de configuraciones de una máquina de Boltzmann), genera estrategias para la localización de configuraciones cuya función de costo (para nuestro interés función de energía) tenga un valor óptimo.

Una de estas estrategias es el algoritmo de **mejoramiento iterativo**, que puede ser descrito como sigue: Comenzando con una configuración dada, se genera una sucesión de iteraciones, cada una de las cuales consiste de una posible transición de la configuración actual a una configuración seleccionada de sus vecinas. Si la configuración vecina tiene un valor menor de energía, la configuración actual es reemplazada por esta vecina. En caso contrario, otra configuración se selecciona y

se compara de igual manera. El algoritmo termina cuando se obtiene una configuración cuyo valor de energía no es menor que el de ninguna de sus vecinas.

Es obvio, que la principal desventaja de este procedimiento es que siempre termina en un mínimo local y generalmente no existe información de cuanto se desvía éste del global.

Otra posible alternativa sería realizar un *backtracking*, tratando de encontrar un mejor mínimo. Es evidente que la principal desventaja de este método es la explosión combinatoria que se provoca al realizar el *backtracking*.

Para evitar algunas de estas desventajas, una variante es la aceptación de transiciones que correspondan a incrementos de la función de energía en una forma limitada (en un algoritmo de mejoramiento iterativo solamente se aceptan transiciones correspondientes a decrementos en la función de energía). Un algoritmo que trabaja en esta forma es el *simulated annealing*.

En el algoritmo *simulated annealing* las nuevas configuraciones son aceptadas de acuerdo a un criterio de aceptación que permite también deterioros del sistema, es decir, permite aceptar configuraciones que provocan inicialmente con alta probabilidad, un incremento en la función de energía.

Producto del proceso de optimización, el criterio de aceptación es modificado en tal forma que la probabilidad para aceptar deterioros del sistema se vuelve más y más pequeña, y al final del proceso de optimización ésta se aproxima a cero. De esta forma se evita que el sistema quede atrapado en un mínimo local.

El decremento en la probabilidad para aceptar deterioros del sistema es gobernado por un parámetro de control del *annealing*, que se denota por T y se le llama temperatura por su similitud con el comportamiento de la temperatura en los sistemas físicos definidos con modelos similares.

La calidad de la solución final está determinada por el comportamiento de la convergencia del algoritmo, y la convergencia está determinada a su vez, por un conjunto de parámetros, lo cual es una desventaja, pues la solución final dependerá de cuan adecuadamente sean seleccionados estos parámetros. Estos aspectos del algoritmo se tratan con más detalle en el capítulo cuatro.

Debe destacarse, que la implementación del algoritmo *simulated annealing* lo convierte realmente en un algoritmo de aproximación, por lo que la obtención de la solución óptima, es decir, de la configuración cuya energía sea un mínimo global no está garantizada.

Teniendo en cuenta que la máquina de Boltzmann realiza la localización de sus configuraciones de equilibrio utilizando este algoritmo, proponemos una variante del mismo que brinda la posibilidad de acercarnos a un mejor mínimo y que resumimos en los siguientes pasos:

1. Partimos de una configuración inicial aleatoria o aleatoria "en un segmento", que es el caso cuando, por requerimientos del modelo de Boltzmann, las unidades de entrada y de salida están fijas.

2. Aplicamos el *annealing* para esta configuración y obtenemos un mínimo, que lo asumiremos como un mínimo local, lo salvaremos y lo retendremos en una variable a la que llamare-

mos OPTIMO, así como a su configuración asociada CONFIG_OPTIMA.

3. Generamos una nueva configuración inicial, le aplicamos el *annealing*, obtenemos un nuevo mínimo, si éste es mejor que OPTIMO lo sustituímos al igual que a CONFIG_OPTIMA con la configuración asociada.

4. Repetimos el paso 3 tantas veces como el número total de unidades que posea la red, suponiendo que al finalizar, en OPTIMO está el óptimo global (en realidad el mejor mínimo encontrado) y en CONFIG_OPTIMA la configuración que lo originó.

2.4 RECONOCIMIENTO DE PATRONES

Después de completar el algoritmo de aprendizaje, la máquina de Boltzmann es capaz de completar un ejemplo parcial (es decir, una situación donde solamente un subconjunto de las unidades ambientales o unidades visibles queda fijo) al minimizar la función de energía.

En esta forma la máquina de Boltzmann no sólo puede reproducir los ejemplos dados (simulando una capacidad de memoria), sino que también tiene capacidades asociativas e inductivas, aunque limitadas.

Para una entrada dada, la energía es minimizada usando los pesos obtenidos en la fase de aprendizaje. Las unidades ocultas y las de salida son consideradas libres, es decir, sólo a ellas les está permitido cambiar sus estados durante la optimización de la función de energía.

Después de la optimización, la prueba puede ser completada al comparar la salida resultante con las salida "esperada". La minimización de la energía es llevada a cabo usando el algoritmo *simulated annealing* nuevamente. Esto equivale a que los resultados obtenidos por el algoritmo de aprendizaje pueden ser probados fijando las unidades de entrada y dejando que el resto de las unidades, las ocultas y las de salida, modifiquen sus estados para completar el patrón deseado.

CAPITULO 3

APRENDIZAJE EN UNA MAQUINA DE BOLTZMANN

3.1 ESTRATEGIA DE APRENDIZAJE

Una de las metas de los investigadores que estudian modelos conexionistas es obtener algoritmos eficientes de aprendizaje para redes. Se han desarrollado muchos de estos algoritmos en redes conexionistas, entre los que podemos mencionar el perceptrón, retropropagación (*backpropagation*) y Boltzmann.

Este capítulo está dedicado precisamente al algoritmo de aprendizaje de las máquinas de Boltzmann. Quizás el aspecto más interesante de estas máquinas, es que su formulación conduce a un algoritmo de aprendizaje independiente del dominio del problema; los pesos de las conexiones entre las unidades se modifican en tal forma que toda la red desarrolla un modelo interno, el cual captura la estructura fundamental del ambiente.

El aprendizaje en una máquina de Boltzmann toma lugar por ejemplos que son inducidos en un subconjunto de las unidades, las visibles. El resto de las unidades, las ocultas, son usadas para construir una representación interna que captura las regularidades de los ejemplos inducidos en las unidades visibles.

Durante el aprendizaje, se intenta ajustar los pesos de las conexiones de forma tal que las configuraciones de las unidades visibles cumplan con una distribución de probabilidad deseada, determinada por el ambiente.

El número de unidades en la red y sus interconexiones definen un espacio de posibles modelos del ambiente, y cualquier conjunto particular de pesos define un modelo particular en este espacio.

El problema del aprendizaje es encontrar una combinación de pesos que origine un modelo bueno, lo que es lo mismo, se desea encontrar la combinación de pesos que sea la más probable de producir el conjunto de patrones ambientales observados.

En la práctica, el algoritmo de aprendizaje de la máquina de Boltzmann comienza poniendo todos los pesos de las conexiones en cero, realizando posteriormente una secuencia de ciclos de aprendizaje. Cada ciclo consiste de dos fases. En la primera fase, llamada "**situación de inducido**", un conjunto de ejemplos de aprendizaje es inducido en las unidades de entrada y de salida. Para cada ejemplo se debe alcanzar el equilibrio, en base al conjunto de pesos actuales, es decir, a las unidades de entrada y de salida no se les permite cambiar sus estados. En la segunda fase se deja a la red correr libremente hasta alcanzar el equilibrio ("**situación de corrida libre**"), es decir, a todas las unidades se les permite cambiar sus estados.

Entre cada ciclo de aprendizaje, los pesos de las conexiones son ajustados usando la información estadística obtenida en las dos fases del ciclo de aprendizaje anterior, mediante la siguiente expresión:

$$\Delta w_{ij} = \varepsilon \left(< p_{ij} > - < p'_{ij} > \right) \qquad (3.1)$$

donde

ε es una constante conocida como **rapidez de aprendizaje** que debe estar entre cero y uno.

$< p_{ij} >$ es la probabilidad promediada (sobre todas las entradas ambientales) y medida en equilibrio, de que la i-ésima y j-ésima unidades estén ambas en *on* en la situación de inducido.

$< p'_{ij} >$ es la probabilidad promediada (sobre el mismo número de entradas ambientales usadas en la fase 1) y también medida en equilibrio, de que la i-ésima y j-ésima unidades estén en *on* cuando la red está corriendo libremente.

Este proceso es ejecutado hasta que el cambio promedio de los pesos de las conexiones sea cero.

Por lo tanto, nuestro interés será encontrar un conjunto de pesos que, cuando la red esté corriendo libremente, los patrones de actividad que ocurran en las unidades visibles sean los mismos que estarían si el ambiente los estuviera induciendo.

A continuación explicaremos como llegar a este algoritmo de aprendizaje simple en apariencia.

Se recordará que toda asignación de valores de activación en las unidades determina una configuración. Viendo sólo a las unidades visibles (las unidades de entrada y de salida), toda configuración determina una configuración marginal. La estructura completa del conjunto de configuraciones marginales para las unidades visibles puede ser especificada dando la probabilidad P^+_α, de cada una de las 2^k configuraciones marginales sobre las k unidades visibles. P^+_α no depende de los pesos de la red porque el ambiente induce las unidades visibles.

El objetivo del algoritmo de aprendizaje es ajustar los pesos, de manera tal que la máquina de Boltzmann, en la situación de corrida libre, tienda a estar, con alta probabilidad, en las configuraciones ambientales que son usadas como ejemplos en la situación de inducido. Para lograr este fin, se utilizan las distribuciones de probabilidad P_α^+ y P_α^- definidas sobre el conjunto de configuraciones visibles.

P_α^+ denota la probabilidad de que los estados de las unidades visibles estén dados por la configuración ambiental α en la situación de inducido. Similarmente, P_α^- denota la probabilidad de que los estados en las unidades visibles estén dados por α, en la situación de corrida libre.

La distribución de probabilidad P_α^+ está determinada por las configuraciones visibles contenidas en el conjunto de aprendizaje y por la frecuencia con que ellas son inducidas en las unidades visibles. P_α^+ es grande si α está contenida en el conjunto de aprendizaje y es frecuentemente inducida en las unidades visibles.

La distribución de probabilidad P_α^- depende de los pesos y está relacionada con el mecanismo de transición estocástico aplicado para el algoritmo *simulated annealing*.

Para obtener la distribución de probabilidad P_α^- usamos el algoritmo *simulated annealing*, que es empleado solamente para generar un patrón estable que refleje las propiedades de la distribución de probabilidad.

El objetivo del algoritmo de aprendizaje puede ser formulado entonces, como la mofificación de los pesos que hace que P_α se aproxime a P_α^+.

Es posible derivar una medida de como los pesos en la red son usados para modelar la estructura del ambiente, y también es posible mostrar como los pesos deben ser cambiados para progresivamente mejorar esta medida.

Si asumimos que el ambiente fija un vector sobre las unidades visibles y lo conserva suficientemente para que la red alcance el equilibrio con ese vector, y también asumimos que no existe una estructura en el orden secuencial de los vectores ambientales inducidos, esto significa que la estructura completa del conjunto de vectores ambientales puede ser especificada dando la probabilidad P_α^+, de cada uno de los 2^k vectores sobre las k unidades visibles. Insistimos en el hecho de que P_α^+ no depende de los pesos de la red porque el ambiente fija las unidades visibles.

Un conjunto particular de pesos puede constituir un modelo perfecto de la estructura del ambiente si conduce, exactamente, a la misma distribución de probabilidad de los vectores visibles, cuando la red está corriendo libremente. Por el comportamiento estocástico de las unidades, la red recorrerá una variedad de configuraciones sin entrada ambiental fija, y por lo tanto generará una distribución de probabilidad P_α^- sobre todos los 2^k vectores visibles. Esta distribución puede ser comparada con la distribución P_α^+.

Una medida (proveniente de la teoría de la información) de la discrepancia entre el modelo interno de la red y el ambiente está dada por la función G, cuya expresión es:

$$G = \sum_\alpha P_\alpha^+ \ln \frac{P_\alpha^+}{P_\alpha^-} \qquad (3.2)$$

donde

P_{α}^{+} es la probabilidad del α-ésimo estado de las unidades visibles en la fase 1 cuando sus estados están determinados por el ambiente.

P_{α}^{-} es la probabilidad correspondiente en la fase 2 cuando la red está corriendo libremente sin entrada ambiental.

La función G es cero si y sólo sí las distribuciones son idénticas, en cualquier otro caso es positiva. Como el término P_{α}^{-} depende de los pesos, G puede ser alterada al cambiar éstos.

Por lo tanto, el objetivo del algoritmo de aprendizaje es: Minimizar G mediante el cambio de los pesos de las conexiones.

Si el problema del aprendizaje es visto como un problema de minimización de la distancia de la medida $P_{\alpha}^{-} = MARG(P,M)$, que describe el comportamiento de la red, y la medida P_{α}^{+}, que es la medida de probabilidad observada en el espacio de configuración I^{M}, entonces el gradiente de esta distancia, considerado como una función de los pesos, es la diferencia de dos términos, uno de los cuales es el valor esperado de $s_i s_j$ con respecto a la medida deseada, mientras que el otro, es el valor esperado respecto a la distribución actual de Boltzmann.

Para minimizar G se usa el método del gradiente descendente o método de descenso por el gradiente. Este algoritmo sugiere cambiar cada peso por una cantidad proporcional al gradiente de G, mejorando el conjunto de pesos por un desplazamiento de la pendiente en la superficie que se define en el espacio de pesos.

El gradiente de G es el vector $\nabla G = \left[\frac{\partial G}{\partial w_{ij}} \right]_{ij}$, que efectivamente tiene en cada una de sus coordenadas a la derivada parcial $\frac{\partial G}{\partial w_{ij}}$.

En cada pareja de índices ij, el incremento Δw_{ij}, puede ser considerado prácticamente igual a:

$$\Delta w_{ij} = - \eta \frac{\partial G}{\partial w_{ij}} \qquad (3.3)$$

En el subespacio de patrones, la regla del gradiente necesariamente decrementa a G si η es suficientemente pequeño.

Al aplicar el método del gradiente descendente a este tipo de redes, uno de los principales problemas es el de calcular el gradiente de una función que dé una medida del comportamiento global de la red, de forma que pueda ser implementada usando información localmente disponible para cada unidad. El procedimiento de aprendizaje de Boltzmann hace esto tomando las ventajas de las propiedades de equilibrio de las redes simétricas y estocásticas.

Para realizar el descenso por gradiente en G es necesario conocer su derivada con respecto a cada peso individualmente, además, ya que las probabilidades de las configuraciones globales están determinadas por sus energías y éstas su vez por los pesos, entonces, cuando la red está corriendo libremente la distribución de probabilidad sobre las unidades visibles, $P_\alpha = MARG(P,M)$, está dada por:

$$P_\alpha^- = \sum_\beta P_\alpha^- = \frac{\sum_\beta e^{-E_{\alpha\beta}/T}}{\sum_{\lambda\mu} e^{-E_{\lambda\mu}/T}} \qquad (3.4)$$

donde α es un vector de los estados de las unidades visibles

β es un vector de los estados de las unidades ocultas

$E_{\alpha\beta}$ es la energía del sistema en el estado $\alpha\beta$

$$E_{\alpha\beta} = -\sum_{i<j} w_{ij}\, s_i^{\alpha\beta}\, s_j^{\alpha\beta}$$

De aquí,

$$\frac{\partial e^{-E_{\alpha\beta}/T}}{\partial w_{ij}} = \frac{1}{T}\, s_i^{\alpha\beta}\, s_j^{\alpha\beta}\, e^{-E_{\alpha\beta}/T}$$

Diferenciando la ecuación (3.4) entonces tenemos:

$$\frac{\partial P_\alpha^-}{\partial w_{ij}} = \frac{\frac{1}{T}\sum_\beta e^{-E_{\alpha\beta}/T}\, s_i^{\alpha\beta}\, s_j^{\alpha\beta}}{\sum_{\alpha\beta} e^{-E_{\alpha\beta}/T}}$$

$$- \frac{\sum_\beta e^{-E_{\alpha\beta}/T}\, \frac{1}{T}\sum_{\lambda\mu} e^{-E_{\lambda\mu}/T}\, s_i^{\lambda\mu}\, s_j^{\lambda\mu}}{\left(\sum_{\lambda\mu} e^{-E_{\lambda\mu}/T}\right)^2}$$

$$= \frac{1}{T}\left[\sum_\beta P^-(\alpha\beta)\, s_i^{\alpha\beta}\, s_j^{\alpha\beta} - P^-(\alpha)\sum_{\lambda\mu} P^-(\lambda\mu)\, s_i^{\lambda\mu}\, s_j^{\lambda\mu}\right]$$

La derivada anterior se usa para calcular el gradiente de la medida G, ecuación (3.2), donde P_α^* es la distribución de probabilidad de las configuraciones inducidas en las unidades visibles y es independiente de las w_{ij}. Así

$$\frac{\partial G}{\partial w_{ij}} = -\sum_{\alpha} \frac{P^{+}(\alpha)}{P^{-}(\alpha)} \frac{\partial P^{-}(\alpha)}{\partial w_{ij}}$$

$$= -\frac{1}{T}\sum_{\alpha} \frac{P^{+}_{\alpha}}{P^{-}_{\alpha}} \left[\sum_{\beta} P^{-}_{\alpha\beta}\, s_i^{\alpha\beta} s_j^{\alpha\beta} - P^{-}_{\alpha} \sum_{\lambda\mu} P^{-}_{\lambda\mu}\, s_i^{\lambda\mu} s_j^{\lambda\mu} \right]$$

Conociendo que:

$$P^{+}_{\alpha\beta} = P^{+}(\beta\backslash\alpha)\; P^{+}_{\alpha}$$

$$P^{-} = P^{-}(\beta\backslash\alpha)\; P^{-}_{\alpha\beta}$$

y además que

$$P^{-}(\beta\backslash\alpha) = P^{+}(\beta\backslash\alpha)$$

porque la probabilidad de una configuración, dando las unidades visibles, debe ser la misma en equilibrio si las unidades visibles fueron inducidas en ese estado u obtenidas por corrida libre. De ahí que:

$$P^{-}_{\alpha\beta} \frac{P^{+}_{\alpha}}{P^{-}_{\alpha}} = P^{+}_{\alpha\beta}$$

Como

$$\sum_{\alpha} P^{+}_{\alpha} = 1$$

Entonces,

$$\frac{\partial G}{\partial w_{ij}} = -\frac{1}{T} \left[p_{ij} - p'_{ij} \right] \qquad (3.5)$$

donde

$$p_{ij} \overset{def}{=} \sum_{\beta} P_{\alpha}^{-} \, s_i^{\alpha\beta} s_j^{\alpha\beta}$$

$$p'_{ij} \overset{def}{=} \sum_{\lambda\mu} P_{\lambda\mu}^{-} \, s_i^{\lambda\mu} s_j^{\lambda\mu}$$

Para minimizar G, es suficiente recopilar las estadísticas p_{ij} y p'_{ij} cuando la red está en equilibrio y cambiar cada peso por una cantidad proporcional a la diferencia entre estas dos probabilidades:

$$\Delta w_{ij} = \varepsilon \, (\, p_{ij} - p'_{ij} \,) \qquad (3.6)$$

donde ε mide el tamaño de cada cambio de peso.

La ecuación anterior no es más que la ecuación (3.1) para realizar las actualizaciones de los pesos en el algoritmo de aprendizaje.

Esta regla solamente usa la información localmente disponible. El cambio en un peso sólo depende del comportamiento de las dos unidades conectadas por él, aunque el cambio optimice la función G, y el mejor valor para cada peso dependa de los valores de todos los otros pesos.

El algoritmo de aprendizaje posee unos parámetros que pueden originar variaciones en el mismo, estos son, el tamaño de ε (el cual determina el tamaño de cada paso tomado para realizar el descenso por el gradiente) y la duración del tiempo para estimar p_{ij} y p'_{ij}. Los valores de estos parámetros pueden tener un efecto significativo en el proceso de aprendizaje.

Una forma práctica para estimar p_{ij} y p'_{ij} tendrá necesariamente algún ruido en las estimaciones. El efecto del ruido en las estimaciones puede ser reducido usando un valor pequeño de ε o coleccionando las estadísticas por un mayor tiempo.

La función objetivo G es una métrica que especifica cuan bien dos distribuciones de probabilidad empatan. Los problemas se originan cuando el ambiente especifica que solamente un subconjunto de los patrones posibles sobre las unidades visibles ocurren.

Es evidente que los patrones no mencionados deben ocurrir con probabilidad cero, y la única forma en que la máquina de Boltzmann, corriendo a una temperatura distinta de cero, puede garantizar que ciertas configuraciones nunca ocurran es dando a estas configuraciones energías infinitamente altas, lo cual requiere pesos infinitamente grandes.

Una forma de evitar la necesidad de pesos infinitos es tomar ocasionalmente configuraciones marginales "ruidosas" de entrada. Esto puede ser hecho filtrando las configuraciones marginales de entrada "correctas", a través de un proceso que tenga una pequeña probabilidad de invertir cada uno de los bits.

Estas configuraciones marginales ruidosas son inducidas en las unidades visibles. Si el ruido es pequeño, las configuraciones marginales correctas dominan las estadísticas, pero cualquier configuración marginal tendrá la oportunidad de ocurrir y las energías infinitas no serán necesarias.

3.2 IMPLEMENTACION DEL APRENDIZAJE

Un ejemplo simple que puede ser resuelto por una máquinade Boltzmann es el clásico problema del XOR (tabla 3.1).

Tabla de la función XOR

0	0	0
0	1	1
1	0	1
1	1	0

Tabla 3.1

Esta función es uno de los ejemplos usados por Minsky y Paper [MP69] para mostrar las fuertes limitaciones del algoritmo de aprendizaje de los perceptrones para aprender. Esto es debido a que la regla de Widrow-Hoff o procedimiento de convergencia del perceptrón (Rosenblatt, 1962) es una regla de aprendizaje diseñada para capturar estructuras de segundo orden, y sin embargo el problema del XOR tiene una estructura de tercer orden, pues el conjunto consiste de configuraciones marginales (1 1 0), (1 0 1), (0 1 0) y (0 0 0), cada una con una probabilidad de 0.25. Claramente existe alguna estructura, porque cuatro de las ocho configuraciones marginales de tres componentes nunca ocurren.

En términos estadísticos, existen muchos tipos de estructuras en un conjunto de configuraciones marginales ambientales. La probabilidad de que cada unidad visible esté activada o no es la estructura de primer orden y puede ser capturada por los umbrales de las unidades visibles, las $v^2/2$ correlaciones por pareja entre las v unidades visibles, constituye la

estructura de segundo orden y ésta puede ser capturada por los pesos entre pares de unidades. Toda estructura de más alto orden no puede ser capturada por los pesos entre las unidades visibles directamente.

Esto puede resolverse introduciendo unidades ocultas, las que no forman parte de la definición del conjunto original, lo que permite expresar la estructura del conjunto como una estructura de segundo orden, pero de un conjunto de más unidades.

En el caso del XOR se puede añadir una cuarta componente y tomar el conjunto { (1101), (1010), (0110), (0000) }. Ahora es posible usar los umbrales y los pesos entre todas las unidades para expresar la estructura de tercer orden en las primeras tres componentes. En este caso se introdujo un detector de rasgo o detector de característica, el cual detecta el caso cuando las primeras dos unidades están en *on*.

La dificultad de introducir la unidad oculta es determinar cuando debe estar en *on* y cuando debe estar en *off*, decidiendo que rasgo debe detectar. (En este ejemplo existen seis formas diferentes de usar la unidad oculta para resolver la tarea).

Una forma de pensar en la estructura de más alto orden de un conjunto de configuraciones marginales ambientales, es que ella especifique implícitamente conjuntos de características fundamentales buenas, que pueden ser usados para modelar la estructura del ambiente.

El problema del aprendizaje difícil es determinar cuales de estas características están presentes, es decir, encontrar un conjunto de pesos, los cuales conviertan a las unidades ocultas en detectores de rasgos útiles que explícitamente

representen las propiedades del ambiente, las cuales están sólo presentes implícitamente como estadísticas de más alto orden en el conjunto de configuraciones marginales ambientales.

La regla de Widrow-Hoff puede realizar el llamado aprendizaje fácil, pero no puede hacer el tipo de aprendizaje difícil, que evoluciona decidiendo como usar las unidades ocultas, cuyo comportamiento no está directamente especificado por la tarea.

Otra variante para resolver el problema del XOR es introduciendo dos unidades extras. Desde el punto de vista de la red esto equivale a introducir de un nivel de unidades ocultas con dos unidades, donde las entradas no están conectadas directamente a la salida.

La red resolverá el problema siempre que logremos obtener un conjunto de pesos tal que, cuando ninguna entrada esté activada (0,0), ninguna unidad oculta esté activada, lo que origina que la unidad de salida quede en *off*. Cuando una unidad de entrada esté en *on* (0,1) o (1,0), la primera unidad oculta cambia, activando la unidad de salida. Si ambas unidades de entrada están activadas (1,1), ambas unidades ocultas se activan. De esta manera, las unidades ocultas actúan como detectores de rasgos o filtros para algunos tipos de entrada. Por combinación de estos rasgos la unidad de salida puede realizar una clasificación más poderosa que sin unidades ocultas.

El algoritmo de aprendizaje de la máquina de Boltzmann resuelve el problema del aprendizaje difícil, pues propone una fórmula general para determinar la variación de los pesos, independientemente de que estén asociados a conexiones que unan unidades de entrada con ellas mismas, de entrada con

ocultas u ocultas con de salida, es decir, no es significativo que tipo de unidades estén conectadas.

Vamos a usar el problema del XOR para probar nuestra implementación del algoritmo de aprendizaje, pues como ya hemos mencionado es uno de los problemas clásicos de aprendizaje en redes neuronales, presenta además severas limitaciones de velocidad en la simulación del algoritmo en una máquina secuencial, y también porque el aprendizaje necesita de la realización de muchos *annealings*. La prueba de implementación del algoritmo de aprendizaje comprende, inicialmente, una red con un nivel de unidades ocultas con dos unidades, un nivel de entrada con dos unidades y un nivel de salida con una unidad. También se utiliza una unidad extra, que estará permanentemente en el estado *on*, para sustituir los umbrales de las unidades por pesos de las conexiones de esta unidad extra con el resto de las unidades.

Otro detalle a destacar de la implementación es que calculamos la probabilidad de que dos unidades i y j estén simultáneamente en *on* en la fase 1 de aprendizaje, $p_{ij} = \langle s_i s_j \rangle$, contando el número de veces que ellas están ambas activadas promediado sobre todos los patrones (entradas-salidas inducidos) del conjunto de entrenamiento. De igual forma, en la fase 2 se realiza la determinación de p'_{ij}. Ambos conjuntos de estadísticas se coleccionan en equilibrio.

Las primeras simulaciones están basadas en un modelo de la máquina de Boltzmann en el cual:

- las unidades de entrada no están conectadas entre sí.
- todas las unidades ocultas están conectadas a todas las unidades de entrada y de salida.

Se realizaron otras simulaciones con un modelo en el que las unidades de entrada están mutuamente conectadas.

Los experimentos en la red fueron realizados usando el siguiente ciclo de aprendizaje:

1.- **Estimación de p_{ij}**: Cada vector ambiental en turno, fue inducido en las unidades visibles. Para cada vector ambiental, a la red le fue permitido alcanzar su equilibrio para una temperatura final (el *annealing schedule* será descrito posteriormente). Las estadísticas sobre cuando los pares de unidades conectadas estuvieron simultáneamente en *on* fueron tomadas en equilibrio.

2.- **Estimación de p'_{ij}**: La red estuvo completamente libre y se le permitió alcanzar su equilibrio tantas veces como se le permitió alcanzar el equilibrio para la estimación de p_{ij}.

3.- **Actualización de los pesos**: Todos los pesos en la red fueron modificados de acuerdo a la ecuación (3.1), realizándose pruebas para valores de ε de 0.1, 0.5, 0.9.

En una prueba inicial, la fase 1 fue repetida 20 veces, presentado los cuatro patrones a ser aprendidos cinco veces sucesivamente, alternando cada uno de ellos, lo que provocó cinco secuencias de cuatros configuraciones marginales distintos. Posteriormente, se presentó cada patrón 15 veces alternativamente, lo que provocó que esta fase se repitiera 60 veces. El mismo número de repeticiones se tomó en la fase 2 para recoger la información necesaria para el cálculo de las p.

El *simulated annealing* está controlado por un parámetro, llamado parámetro de control, que juega un rol similar al de la temperatura de los procesos físicos de *annealing*, y que hemos denotado por T. Para ajustar los estados de las unidades, un mecanismo de transición de estado probabilístico, el cual es función de T, es empleado, como se señaló anteriormente. Esta aleatoriedad es necesaria para evitar el mínimo lo-

Los experimentos en la red fueron realizados usando el siguiente ciclo de aprendizaje:

1.- **Estimación de p_{ij}**: Cada vector ambiental en turno, fue inducido en las unidades visibles. Para cada vector ambiental, a la red le fue permitido alcanzar su equilibrio para una temperatura final (el *annealing schedule* será descrito posteriormente). Las estadísticas sobre cuando los pares de unidades conectadas estuvieron simultáneamente en *on* fueron tomadas en equilibrio.

2.- **Estimación de p'_{ij}**: La red estuvo completamente libre y se le permitió alcanzar su equilibrio tantas veces como se le permitió alcanzar el equilibrio para la estimación de p_{ij}.

3.- **Actualización de los pesos:** Todos los pesos en la red fueron modificados de acuerdo a la ecuación (3.1), realizándose pruebas para valores de ε de 0.1, 0.5, 0.9.

En una prueba inicial, la fase 1 fue repetida 20 veces, presentado los cuatro patrones a ser aprendidos cinco veces sucesivamente, alternando cada uno de ellos, lo que provocó cinco secuencias de cuatros configuraciones marginales distintos. Posteriormente, se presentó cada patrón 15 veces alternativamente, lo que provocó que esta fase se repitiera 60 veces. El mismo número de repeticiones se tomó en la fase 2 para recoger la información necesaria para el cálculo de las p.

El *simulated annealing* está controlado por un parámetro, llamado parámetro de control, que juega un rol similar al de la temperatura de los procesos físicos de *annealing*, y que hemos denotado por T. Para ajustar los estados de las unidades, un mecanismo de transición de estado probabilístico, el cual es función de T, es empleado, como se señaló anteriormente. Esta aleatoriedad es necesaria para evitar el mínimo lo-

cal. Inicialmente, el valor del parámetro T provoca una alta aleatoriedad (es decir, las transiciones que causan una mejor solución son seleccionadas, virtualmente, con igual probabilidad como aquellas que causan soluciones peores).

Lentamente T va decreciendo, provocando que la aleatoriedad tienda a cero (es decir, solamente las transiciones que causan mejores soluciones serán seleccionadas. Cuando T se acerca a cero el proceso de optimización es completado. La calidad de la solución final depende altamente del camino usado para transitar de una alta aleatoriedad a una baja aleatoriedad. Este camino es llamado *annealing schedule*, este concepto será retomado en el próximo capítulo.

Para nuestra implementación, el *annealing schedule* utilizado fue el introducido por Kirkpatrick, donde el valor inicial de temperatura es determinado experimentalmente, tal que para este valor todas las configuraciones propuestas son virtualmente aceptadas. El decremento de la temperatura está dado por una regla de producto, tal que $T_{actual} = \alpha\, T_{vieja}$ donde α es una constante pequeña, pero cercana a uno.

Para analizar el comportamiento, se comenzó con una temperatura muy alta, 200. Entonces se se decidió tomar como temperatura inicial el valor de 20. Se utilizaron valores de la constante α entre 0.5 y 0.99, para finalmente tomar el valor de 0.9. La temperatura final se fijó originalmeante en 0.1, posteriormente se tomó como 1 y se realizaron pruebas con temperatura final de 10.

Sabemos que uno de los factores que puede influir en el algoritmo de aprendizaje es el valor de la constante de rapidez de aprendizaje ε. Esta constante, que debe estar entre cero y uno, es determinada por experimentación, teniendo en cuenta que la misma debe ser pequeña. Para las simulaciones de

prueba que se realizaron fueron probados los siguientes valores: 0.002, 0.005, 0.05, 0.01 y 0.5.

La posibilidad de calcular la derivada parcial de G por observación de p_{ij} y de p'_{ij} no determina completamente el algoritmo de aprendizaje. Todavía es necesario decidir que tanto cambiar cada peso, en que tiempo coleccionar las estadísticas de co-ocurrencia antes de cambiar el peso y que *schedule* de temperatura usar durante los *annealings*.

Una dificultad es que no existe, realmente, algo para prevenir al algoritmo de aprendizaje de generar pesos muy grandes, los cuales pueden crecer hasta límites tales de energía, que la red no pueda alcanzar el equilibrio en el tiempo asignado. Una forma de asegurar que la red quede cerca del equilibrio es conservar los peso pequeños.

Si para los cambios de peso causados por el aprendizaje, todos los pesos continuamente decaen hacia un valor de cero, con la velocidad del descenso siendo proporcional a la magnitud absoluta del peso, esto conserva a los pesos pequeños, y eventualmente, conduce a una situación relativamente estable, en la cual la razón de descenso de un peso es balanceada por la derivada parcial de G con respecto al peso. Esto tiene la propiedad de que la magnitud absoluta de un peso muestra cuan importante es para modelar la estructura del ambiente.

En nuestra implementación se realizaron pruebas para el control del crecimiento de los pesos, tal que todos los pesos tuvieron su magnitud absoluta decrementada por 0.005 veces la magnitud absoluta. Aunque primeramente se hicieron pruebas afectando directamente cada peso por un factor constante igual a 0.005, es decir, $W_{nuevo} = 0.005\ W_{viejo}$.

Otra forma para tratar de prevenir a los pesos de un crecimiento muy grande es la técnica del "ruido" fijado, que consiste en permitir a cada bit de los patrones ambientales cambiar su estado con una baja probabilidad, permitiendo así que patrones que no pertenecen al ambiente también puedan aparecer. En la implementación cada bit en *on* de un vector inducido fue puesto en *off* con una probabilidad de 0.15, y cada bit en *off* fue puesto en *on* con probilidada de 0.05.

El algoritmo es extraordinariamente lento, pues como se recordará, cada ciclo tiene dos fases y en cada fase por cada ejemplo presentado, hay que alcanzar el equilibrio realizando un *annealing* que alenta el proceso sustancialmente.

Las pruebas realizadas (en una micro VAX) tuvieron como máximo entre 7000 y 1500 ciclos de aprendizaje, y como mínimo 20 ciclos, pasando por valores de 800, 300, 100, 200, 50 y 20 ciclos de aprendizaje.

Una variación en el algoritmo de aprendizaje es considerar la regla de actualización de los pesos como sigue:

$$\Delta w_{ij} = \varepsilon \left(< q_{ij} > - < q'_{ij} > \right) \qquad (3.7)$$

donde ε sigue siendo la constante **rapidez de aprendizaje**

$< q_{ij} >$ es igual a $< p_{ij} >$ en la ecuación (3.1).

$< q'_{ij} >$ es igual a $< p'_{ij} >$ en la ecuación (3.1), pero considerando los estados de las unidades de entrada fijos.

Esto quiere decir, que en la fase 2 del algoritmo de aprendizaje las entradas también estarán fijas, y sólo está permitido hacer cambios en los estados de las unidades ocultas y de salida.

Se realizaron pruebas utilizando esta variante, con el problema del XOR, considerando las distintas posibilidades analizadas con la variante original del algoritmo.

La implementación realizada del algoritmo de aprendizaje de una máquina de Boltzmann fue realizada en el lenguaje de programación C, con el compilador de Turbo C 2.0 de Borland para la ejecución en microcomputadoras, y con el C standard para las pruebas realizadas en la Micro VAX-2 modelo Q4.

CAPITULO 4
IMPLEMENTACION DEL ALGORITMO DE 'RECOCIDO SIMULADO'

4.1 INTRODUCCION

Existe una profunda y útil conexión entre la Mecánica Estadística (el comportamiento de los sistemas con muchos grados de libertad en equilibrio térmico a una temperatura finita) y la Optimización Combinatoria (encontrar el mínimo de una función dependiendo de muchos parámetros). Una detallada analogía con el *annealing* en sólidos proporciona un ambiente de trabajo para la optimización de sistemas grandes y complejos. Esta conexión con la Mecánica Estadística expone nueva información y proporciona una perspectiva poco familiar en los problemas tradicionales de optimización.

Algunos resultados de gran relevancia en Optimización Combinatoria, vale decir, en la búsqueda de un óptimo de funciones de variables discretas, se obtuvieron recientemente, en los años setentas.

Sin embargo, todavía hoy muchos problemas de optimización de gran escala pueden ser resueltos sólo aproximadamente mediante el uso de computadoras. Tales problemas quedan resueltos aproximadamente mediante una cantidad de esfuerzo computacional, limitada por una función polinomial del tamaño del problema. Estos problemas no pueden ser resueltos óptimamente, porque la búsqueda de un óptimo requiere cantidades prohibitivas de tiempo computacional. Se está forzado a usar Algoritmos de Aproximación o Heurísticos, para los cuales no existe, en general, garantía alguna de que la solución encontrada por el algoritmo sea óptima.

Un problema de optimización combinatoria se formaliza como una pareja $(\mathcal{R},C)$, donde $\mathcal{R}$ es un conjunto finito de **configuraciones** (también llamado **espacio de configuraciones**), o posiblemente un conjunto infinito y numerable, y C una función de **costo**, $C : \mathcal{R} \longrightarrow \mathbb{R}$, la cual asigna un número real para cada configuración. Por conveniencia, se supone que C está definida de manera tal que el menor valor de C (el mejor con respecto al criterio de optimización) es la configuración correspondiente al óptimo, lo cual no significa pérdida alguna de generalidad. El problema es pues encontrar una configuración para la cual C tome su valor mínimo, es decir, una configuración (óptima) i_0 tal que

$$C_{opt} = C_{i_0} = \min_{i \in \mathcal{R}} C_i$$

donde C_{opt} es el óptimo (mínimo) de la función de costo.

Por ejemplo, para el problema del Agente Viajero, las configuraciones corresponden a recorridos del agente y el objetivo es encontrar el recorrido de menor longitud.

El problema de encontrar soluciones cercanas al óptimo para un problema de optimización combinatoria dado, puede ser resuelto heurísticamente al optimizar la función de costo. La función de costo es una función de varias variables y representa una medida cuantitativa de la "bondad", con respecto a cierto criterio, de cualquier configuración interna (estado) del sistema que tiene que ser optimizado.

Una configuración del sistema está representada por un vector de estado $r \in \mathcal{R}$, cuyas componentes únicamente determinan la configuración dada. El espacio de configuración $\mathcal{R}$ está dado por el conjunto finito de todas las configuraciones posibles del sistema.

Una solución óptima para el problema de optimización combinatoria se obtiene al localizar un mínimo global, es decir a un elemento $i^* \in \mathcal{R}$, con la propiedad de que $C_{i^*} \leq C_i$ para toda $i \in \mathcal{R}$. El conjunto de mínimos globales es denotado por $\mathcal{R}_*$.

Existen dos enfoques cuando tratamos de resolver un problema de optimización combinatoria. Uno, que dá algoritmos de optimización exactos, proporciona una solución global óptima en un tiempo de computación en general prohibitivo ya que consiste esencialmente de búsquedas exhaustivas; el otro dá algoritmos de aproximación, y proporciona soluciones aproximadas en un tiempo de cómputo aceptable.

Los Algoritmos de Aproximación pueden ser divididos en dos categorías: algoritmos confeccionados para un problema específico (*tailored algorithms*), y algoritmos generales aplicables a una gran variedad de problemas de optimización combinatoria. El algoritmo *simulated annealing* puede ser visto como uno de estos últimos: es una técnica de optimización general para problemas de optimización combinatoria. El algoritmo está basado en técnicas aleatorias; sin embargo, incorpora numerosos aspectos relacionados con los algoritmos de mejoramiento iterativo o algoritmos locales de búsqueda.

4.2 ALGORITMO SIMULATED ANNEALING

El algoritmo *simulated annealing* está basado en una combinación de ideas de dos campos diferentes, la Física Estadística y la Optimización Combinatoria. Por una parte puede ser visto como un algoritmo de simulación del proceso físico de *annealing* de sólidos para buscar estados de energía

mínima, y por otro lado, puede ser considerado como una generalización de los algoritmos locales de búsqueda o de mejoramiento iterativo, los cuales juegan un papel importante en la optimización combinatoria.

La estrategia implementada por el *simulated annealing* consiste en explorar el espacio de solución comenzando con una solución seleccionada arbitrariamente. Es decir, comienza con una configuración inicial dada, y continuamente trata de transformar la solución actual en una de sus vecinas por aplicación de un mecanismo de generación y un criterio de aceptación.

El criterio de aceptación toma en cuenta deterioros en la función de costo en una forma limitada. Esto es controlado por un parámetro de control que juega el mismo papel que la temperatura en los procesos físicos de *annealing*.

Al permitir deterioros de la función de costo se hace más general al algoritmo. El efecto resultante es que el algoritmo de *annealing* puede escapar de un mínimo local, cuando eventualmente cae en uno, con el objetivo de alcanzar, a la larga, un mínimo global.

Inicialmente, el criterio de aceptación es tomado de forma tal que las configuraciones que provocan deterioros al sistema son aceptadas con altas probabilidades. Producto del proceso de optimización, el criterio de aceptación es modificado en tal forma que la probabilidad para aceptar deterioros del sistema se vuelve más y más pequeña, y al final del proceso de optimización esta probabilidad se aproxima a cero. De esta manera se evita que el proceso quede atrapado en un mínimo local, y así se facilita arribar a un mínimo global empleando sólo reordenamientos simples del sistema.

El decremento en la probabilidad para aceptar deterioros del sistema es gobernado por el llamado parámetro de control del *cooling* o *annealing* . Es posible decrementar esta probabilidad durante el proceso de optimización después de cada reordenamiento.

Las soluciones obtenidas por el *simulated annealing* no dependen de la configuración inicial, y tienen un costo usualmente cercano al costo mínimo. Además, es posible dar un límite polinomial superior para el tiempo de cómputo para algunas implementaciones del algoritmo.

El *simulated annealing* puede ser visto como un algoritmo que no exhibe las desventajas del mejoramiento iterativo. Sin embargo, la ganancia de su aplicabilidad general es a veces deshecha por el esfuerzo computacional, puesto que el *simulated annealing* es más lento que los algoritmos de mejoramiento iterativo.

La calidad de la solución final está determinada por el comportamiento de la convergencia del algoritmo, que a su vez está gobernada por un conjunto de parámetros que constituyen el *annealing shedule o cooling schedule*.

Debido al hecho de que el *simulated annealing* está basado en los métodos de Monte Carlo, el algoritmo intrínsicamnete requiere grandes cantidades de tiempo de CPU.

Un problema fundamental en el estudio de este algoritmo es proporcionar un apropiado *cooling schedule* que asegure la convergencia rápida a soluciones cercanas al óptimo.

Para aplicar el algoritmo, necesitamos (como en los algoritmos de mejoramiento iterativo) un mecanismo para generar

una nueva solución (una vecina) a partir de la actual. El mecanismo de generación define una estructura de vecindades, donde para cada solución i la vecindad $\mathcal{R}_i$ está definida como el conjunto de soluciones que pueden ser alcanzadas desde i exactamente en un paso del mecanismo de generación.

El *simulated annealing* puede ser formulado como una secuencia de pruebas, donde el resultado de una prueba dada solamente depende del resultado de la prueba anterior.

El *simulated annealing* es un algoritmo aleatorio, que puede ser descrito matemáticamente usando cadenas de Markov. Aquí usamos una aproximación en la cual la probabilidad entre decrementos es conservada constante para un número de reordenamientos del sistema. La ventaja de esta aproximación es que el proceso de optimización puede ser descrito en términos de un conjunto finito de cadenas de Markov (es una familia de cadenas y cada una con un conjunto finito de estados).

En el algoritmo hace falta especificar tres funciones, la de generación , la de aceptación y la de actualización del parámetro de control y dos criterios de paro, un criterio para determinar la condición de equilibrio y otro que es propiamente para terminar el algoritmo.

Dada una configuración $i \in \mathcal{R}$, la función de generación selecciona la siguiente configuración $j \in \mathcal{R}$. Existen muchas formas en las cuales la configuración actual puede ser perturbada para generar la próxima. La acción tomada para generar una nueva configuración se llama **movimiento**, y la colección de todos los movimientos legales es llamado **conjunto de movimientos** $\mathcal{M}$. $\mathcal{M}$ determina los vecinos de cada configuración i. Más formalmente, $\mathcal{R}_i \subset \mathcal{R}$ denota el conjunto de todos los estados $j \in \mathcal{R}$, tal que existe un movimiento $m \in \mathcal{M}$ que va de i a j. Si

la configuración j es una perturbación de i, j es un vecino de i.

Con la topología inducida por $\mathcal{M}$, ahora podemos definir un conjunto característico de $\mathcal{R}$ como sigue:

$$\mathcal{R}_{min} \quad \left\{ i : C_i \leq C_j \text{ para toda } j \in \mathcal{R}_i \right\}$$

La función de aceptación determina cuando la nueva configuración generada es aceptada.

La función de actualización y los criterios de parada constituyen el llamado *cooling schedule*. Una cuidadosa selección de éste es importante para lograr un buen compromiso entre la velocidad del algoritmo y la calidad de la solución final.

Modelo Matemático

El modelo matemático representa al algoritmo como un proceso estocástico, cuyas probabilidades de transición dependen solamente de los estados involucrados en la transición, es decir, como una cadena de Markov.

Dada la naturaleza aleatoria de las funciones de generación y de aceptación, la configuración X_k visitada por el *simulated annealing* en el k-ésimo movimiento es una variable aleatoria. O lo que es lo mismo, X_k puede ser tomada como la salida de la k-ésima prueba.

La probabilidad de que la configuración visitada por el algoritmo en la (K+1)-ésima iteración sea j, dado que la k-ésima iteración fue i, consiste de dos contribuciones inde-

pendientes: la probabilidad de generar j de i para la k-ésima iteración, representada por $G_{ij}(T)$, y la probabilidad de que j sea aceptada actualmente como la nueva configuración, $A_{ij}(T)$, donde T es la temperatura correspondiente a la k-ésima iteración .

Para $i \neq j$, la expresión de la probabilidad de transición está dada por:

$$P_r\left\{ X_{k+1} = j \;/\; X_k = i \right\} = P_{ij}(T) \qquad (4.1)$$

$$P_{ij}(T) = \begin{cases} G_{ij}(T)\; A_{ij}(T) & \text{para toda } j \in \mathcal{R}_i \\ 0 & \text{de otra forma} \end{cases}$$

Si el espacio de configuración es contable (generalmente finito), el proceso estocástico representado por la secuencia de variables aleatorias $\{X_n\}$ producida por el *simulated annealing* es una cadena de Markov.

Bajo la condición de que la cadena de Markov inducida por la probabilidad de transición (4.1) sea irreducible y aperíodica, entonces la cadena de Markov converge asintóticamente (es decir, para cadenas de Markov infinitamente largas y $T \longrightarrow 0$) al conjunto de soluciones óptimas con probabilidad 1.

El valor de X_{n+1} depende solamente del valor de X_n, es decir, la probabilidad de cualquier comportamiento futuro del proceso, cuando la configuración presente es conocida exactamente, no es alterada por el conocimiento adicional correspondiente a su comportamiento pasado.

En su forma standard, el *simulated annealing* selecciona las siguientes probabilidades $G_{ij}(T)$ y $A_{ij}(T)$:

- la probabilidad de generación G_{ij} es seleccionada independiente de T, y uniformemente sobre las vecinas

- la probabilidad de aceptación es seleccionada como

$$A_{ij}(T) = \begin{cases} e^{-\Delta E_{ij}/T} & \text{si } \Delta E > 0 \\ 1 & \text{si } \Delta E \leq 0 \end{cases}$$

donde $\Delta E = E(j) - E(i)$

Algunas condiciones para la convergencia asintótica (por ejemplo, la longitud infinita de la cadena de Markov) no pueden ser garantizadas en la práctica. Un recurso común para implementar el algoritmo es tomar una secuencia de cadenas de Markov de longitud finita, generadas para valores decrecientes del parámetro de control, lo que origina una aproximación de tiempo finito. Esta aproximación de tiempo finito requiere de la especificación de cada uno de los siguientes parámetros:
- valor inicial del parámetro de control.
- función de decremento del parámetro de control o regla de actualización del parámetro de control (temperatura T).
- número de pruebas en cada cadena de Markov, es decir, criterio para determinar la condición de equilibrio.
- criterio de parada para terminar el algoritmo.

Este conjunto de parámetros es usualmente conocido como *annealing schedule* o *cooling schedule*, ya mencionado anteriormente. La clave de estas heurísticas es poder decidir cuando alcanzamos el equilibrio, es decir si la distribución de probabilidad de los estados de la cadena está cerca de la distribución estacionaria de probabilidad.

El criterio de equilibrio es satisfecho cuando un número de movimientos, proporcional al tamaño del problema, es intentado para cada temperatura. Los criterios más típicos son fijar el número de configuraciones generadas o fijar el número de configuraciones nuevas aceptadas.

El objetivo del criterio de parada para terminar el algoritmo es reconocer que el algoritmo ha alcanzado uno buena solución y que es tan pequeña que la probabilidad para escapar de ella es insignificante. Un criterio típico para terminar el algoritmo es cuando el costo promedio no cambia significativamente para valores de T consecutivos o cuando la función de costo ha permanecido sin cambios para un número determinado de ciclos de terminación.

Otra forma de determinar el criterio de parada, es comparar la diferencia entre el máximo y el mínimo costo entre los estados aceptados para ese valor de T, con el cambio máximo en el costo de cualquier movimiento aceptado para el mismo valor de T. Si ellos son iguales, aparentemente todos los estados accesados son de costos comparables y no existe necesidad de continuar con el *simulated annealing*.

Un *schedule* frecuentemente usado es el introducido por Kirkpatrick y otros [KGV82], el cual tiene los siguientes parámetros:

- El valor inicial del parámetro de control es seleccionado experimentalmente, tal que para cualquier valor, virtualmente todas las transiciones sean aceptadas.

- La función de decremento usa una simple regla geométrica: $T_{m+1} = \alpha\ T_m$, donde α es una constante pequeña cercana a uno.

- El número de pruebas en una cadena de Markov es seleccionado constante, es decir, el criterio para determinar la condición de equilibrio se satisface cuando un número de movimientos proporcional al tamaño del problema ha sido intentado.

- El algoritmo para si para un número de cadenas de Markov consecutivo ningún mejoramiento es obtenido, es decir, que el criterio de terminación del algoritmo es satisfecho cuando el costo no cambia en un ciclo de terminación.

En una aproximación de tiempo finito las condiciones asintóticas no son logradas y la convergencia no está garantizada, es decir, el algoritmo obtiene un óptimo local (posiblemente no global). Por esta razón, es considerado como un algortimo de aproximación.

Esta aproximación asume que el valor del parámetro de control T es conservado fijo hasta que la cadena de Markov alcanza su distribución estacionaria de probabilidad. La única limitación impuesta es que la función de actualización de la secuencia de temperaturas tiene que converger a cero. El requerimiento de que la estacionaridad sea alcanzada para cada valor de T es la limitación más severa de esta aproximación, que está dada por la teoría homogénea de las cadenas de Markov. En general, este requerimiento es equivalente a necesitar que un número infinito de iteraciones sean realizadas.

Encontrar un *schedule* efectivo para controlar al *simulated annealing* es esencial para reducir la cantidad de tiempo usado por el algortimo.

Cada *schedule* presentado recurre a heurísticas, por lo tanto, la eficiencia o rendimiento de los diferentes *schedules* puede ser comparada sólo experimentalmente.

4.3 SIMULATED ANNEALING EN UNA MAQUINA DE BOLTZMANN

Similar al algoritmo *simulated annealing*, el concepto de cadenas de Markov puede ser utilizado para describir las transiciones de estado de las unidades en una máquina de Boltzmann, cuando minimizamos la función de energía.

En una implementación secuencial de una máquina de Boltzmann, a las unidades se les permite cambiar sus estados solamente una vez. Este procedimiento es el que puede ser descrito como una secuencia de cadenas de Markov. Cada cadena consiste de una secuencia de pruebas donde el resultado de cada prueba depende probabilísticamente de la prueba previa (los resultados son configuraciones).

Una prueba consiste de dos pasos:

- dada una configuración k, generar una configuración vecina $k^{(1)}$ determinada por la unidad u_i que propone una transición de estado.
- determinar si la configuración $k^{(1)}$ es aceptada. Si es aceptada el resultado de la prueba es $k^{(1)}$ sino es k.

La probabilidad de aceptación de una transición de estado de la unidad u_i, dada la configuración k, es la conocida ecuación (2.7) referida en el capítulo dos.

Siguiendo al algoritmo *simulated annealing*, la minimización de la función de energía comienza con un valor inicial alto del parámetro de control T y una configuración inicial aleatoriamente determinada. Seguidamente una secuencia de cadenas de Markov se generan al tratar, continuamente, de cambiar los estados de las unidades y aplicar el criterio de aceptación. Entre las secuencias de cadenas de Markov, el valor de T es decrementado hasta que se aproxima a cero. Con-

forme T se aproxima a cero la aceptación de los estados comienza a ser menos frecuente, hasta que la máquina de Boltzmann se estabiliza en una configuración final.

La correspondiente probabilidad de transición $T_{kl}(T)$ está definida como sigue:

$$T_{kl}(T) = \begin{cases} G_{kk^{(l)}}(T)\ A_{kk^{(l)}}(T) & \text{si } l = k^{(l)} \\ 1 - \sum_{k^{(l)} \in \mathcal{R}_l} G_{kk^{(l)}}(T)\ A_{kk^{(l)}}(T) & \text{si } l = k \\ 0 & \text{en otro caso} \end{cases} \qquad (4.2)$$

donde $G_{kk^{(l)}}(T)$ es la probabilidad de generación de una transición de estado de una unidad u_l dada la configuración k.

$A_{kk^{(l)}}(T)$ es la probabilidad de aceptación, es decir, la probabilidad de aceptar la transición de la configuración k a la $k^{(l)}$.

T es el parámetro de control.

Una selección de la probabilidad de generación para el *simulated annealing* standard en una máquina de Boltzmann está dada por la siguiente expresión:

$$G_{kk^{(l)}} = \frac{1}{N} \qquad (4.3)$$

Es decir, $G_{kk}^{(l)}$ es seleccionada independientemente de T y de k, y uniformemente selecionada sobre las N unidades.

La probabilidad de aceptación es usualmente seleccionada como:

$$A_{kk}^{(l)} = \frac{1}{1 + e^{-\Delta E_{kk}^{(l)}}} \qquad (4.4)$$

La cadena de Markov inducida por la probabilidad de transición de las ecuaciones (4.3) y (4.4) es irreducible y aperíodica. Así podemos probar que en una forma similar al algoritmo *simulated annealing*, la Máquina de Boltzmann converge asintóticamente al conjunto de configuraciones óptimas.

Como en el algoritmo *simulated annealing*, una aproximación en tiempo finito es obtenida por especificación de un *annealing schedule*, es decir, la máquina de Boltzmann comienza con un valor inicial de T (suficientemente grande) y selecciona una configuración inicial, aleatoriamente. Posteriormente, una secuencia de cadenas de Markov de longitud finita es generada para los valores decrecientes de T.

Eventualmente, como T se acerca a cero, los estados de transición comienzan a ser menos frecuentes y finalmente la máquina se estabiliza en una configuración localmente mínima, que es tomada como configuración final. La aproximación de tiempo finito no garantiza una convergencia hacia configuraciones óptimas.

Al igual que en el *simulated annealing*, en las implementaciones prácticas de la máquina de Boltzmann, la convergencia a una configuración de mínima energía no está garantizada, por

lo tanto la máquina alcanzará configuraciones que corresponden a un mínimo local de la función de energía, el cual podría estar cerca (o aún ser igual) a la energía mínima.

4.4 PRESENTACION TECNICA DE LA IMPLEMENTACION

En nuestra implementación del *simulated annealing* para una máquina de Boltzmann usamos un modelo secuencial, por lo tanto, a las unidades se les permite cambiar sus estados solamente a una a la vez.

Para elaborar el trabajo se realizaron cambios en el algoritmo standard del *simulated annealing* para una máquina de Boltzmann. En la secuencia de configuraciones, donde una configuración depende probabilísticamente de la configuración anterior, los dos pasos para determinar la nueva configuración se implementaron de la siguiente forma:

Dada una configuración k, entonces para generar la configuración $k^{(i)}$, determinada por una unidad u_i que propone una transición de estado, se selecciona la unidad y se calcula el valor de p_i para decidir que estado deberá tener la unidad u_i, de acuerdo a la expresión de la ecuación (2.7) del capítulo dos:

$$s_i = \begin{cases} 1 & \text{si } p_i > 0.5 \\ 0 & \text{si } p_i \leq 0.5 \end{cases} \qquad (4.5)$$

independientemente del estado que tenía la unidad anteriormente.

Para aceptar esta configuración así generada utilizamos la condición o función de aceptación standard del *simulated annealing:*

$$A_{kk}(i)(T) = \begin{cases} 1 & \text{si } \Delta E_i \leq 0 \\ e^{-\Delta E_i / T} & \text{si } \Delta E_i > 0 \end{cases} \qquad (4.6)$$

Como es evidente, el caso $\Delta E_i > 0$ es tratado probabilísticamente. La probabilidad de que la configuración sea aceptada estará dada por $A_{kk}(i)(T)$. Se generarán números aleatorios uniformemente distribuídos en el intervalo (0,1), uno de estos números se selecciona y compara con $A_{kk}(i)(T)$. Si este número es menor que $A_{kk}(i)(T)$, la nueva configuración es retenida, si no, la configuración original es conservada y usada para comenzar el próximo paso del algoritmo.

Esta propuesta final se complementa, además, con el hecho de que, para aumentar la probabilidad de obtener mejores mínimos con esta aproximación, se realizaron tantos *annealings* como unidades tenga el modelo de Boltzmann, en la propuesta como ya se vio en el capítulo uno. Es obvio que el mínimo alcanzado tiene mayor probabilidad de ser el óptimo, pero el tiempo de cómputo es mucho mayor por el propio hecho de la repetición. Este aumento en la probablidad de obtener un mejor mínimo es a costa de un aumento del tiempo de CPU empleado y de la velocidad del algoritmo.

Las modificaciones al algoritmo se realizaron tomando en cuenta las funciones de generación y de aceptación. En la función de generación se añadió la determinación del estado de una unidad dependiente de una función probabilística, y la función de aceptación es ahora la clásica del *simulated annealing*.

El *annealing schedule* usado es una modificación del propuesto por Kirkpatrick, donde el valor inicial del parámetro de control es determinado experimentalmente. La función de actualización es la misma, y el criterio de terminación es igual también, pero el número de configuraciones generadas por cada valor del parámetro no es fijo y depende del número de cambios efectivos que se producen en los estados de las unidades, ya que los estados de las unidades se están determinando por la expresión (4.5).

Concretamente, para lograr la condición de equilibrio, se están generando, para cada temperatura, una secuencia de configuraciones hasta que no se produzca más cambio de estado en las unidades.

4.5 PRUEBAS REALIZADAS

Se consideró la máquina de Boltzmann como la herramienta de minimización de energía que es, para así estudiar al algoritmo *simulated annealing.* Se construyó una máquina asumiendo que todas las unidades están conectadas, generando aleatoriamente los pesos de las conexiones entre las unidades.

Inicialmente se hicieron pruebas para estudiar el comportamiento del algoritmo en la obtención de las configuraciones de energía mínima, evaluando como contribuía la probabilidad calculada para los cambios de estado en las unidades, es decir, evaluando si el camino seguido para llegar a la configuración de energía mínima era el de más alta probabilidad. Para investigarlo se generaron todas las posibles configuraciones del sistema, partiendo de una configuración inicial dada con sus correspondientes probabilidades asociadas. Se utilizó un árbol binario con este fin, cuyas hojas son precisamnete las

configuraciones posibles, los nodos intermedios son las configuraciones por las que se transita para alcanzar cada configuración final, y tienen asociadas las probabilidades generadas para el cambio de estado por cada unidad. En general, el camino seguido para obtener la configuración de energía global mínima no es el de mayor probabilidad.

Posteriormente probamos el algoritmo escogiendo las unidades en forma determinística para su actualización, utilizando el criterio de decidir colocar el estado de la unidad en uno si la probabilidad de poner un uno es mayor que la probabilidad de poner un cero, y de poner cero en caso contrario, es decir, activamos la unidad en el caso que la proabilidad fuese mayor que 0.5, y la desactivamos en cualquier otro caso.

Se experimentó con el parámetro de control para determinar su valor inicial, su actualización y su valor final. Se tomaron valores iniciales de temperatura entre 300 y 1, y se probaron decrementos constantes de 10, 1 y 0.5; y valores finales de 10, 1, 0.5 y de 0.1.

También se introdujo una modificación para realizar comparaciones, que consiste en hacer los cambios de estado de las unidades basándonos en la siguiente regla:

$$\text{Si } p_i < 0.5 \quad \text{y} \quad s_i = 1, \text{ entonces } s_i = 0$$

$$\text{Si } p_i \leq 0.5 \quad \text{y} \quad s_i = 0, \text{ entonces } s_i = 1$$

Para las pruebas, las matrices de pesos fueron generadas aleatoriamente considerando que los pesos fueran pequeños y que las configuraciones iniciales también fuesen generadas aleatoriamente.

Se implementó el *annealing* con las siguientes propuestas:
- Para lograr la condición de equilibrio, se genera la configuración tomando en cuenta el criterio dado por la ecuación (4.5), inmediatemente se analiza si la configuración es aceptada o no, y se checa si se produjo un cambio efectivo en el estado de la unidada seleccionada. Este proceso es desarrollado hasta que se ha analizado al menos una vez a cada unidad y hasta que no exista cambio de estado.

- Otra propuesta es realizar, por temperatura, la búsqueda de una configuración con el mecanismo antes descrito de la ecuación (4.5), hasta que no haya más cambio y se hayan recorrido todas las unidades, y después analizar si aceptamos o no esta configuración así obtenida.

CAPITULO 5
OPTIMIZACION

5.1 EL PROBLEMA DEL AGENTE VIAJERO

El problema del agente viajero es uno de los ejemplos clásicos de un problema de optimización complejo; es tal vez, en el área de Optimización Combinatoria, de los más conocidos.

Muchos problemas en el ámbito de la Computación y en el diseño de VLSI están relacionados con él, es por esta razón que la investigación para obtener soluciones buenas del mismo es un problema abierto que acapara la atención de los investigadores. Fácil de formular, es todavía difícil de resolver, pertenece a la clase de problemas NP-completo *[NP-complete (no deterministc polynomial time complete)]*.

En el problema del agente viajero se requiere construir el recorrido más corto entre N ciudades predefinidas, donde cada ciudad es visitada exactamente una vez.

Si N es el número de ciudades y $D = [D(i,j)]$ es una matriz de distancias, cuyas entradas $D(i,j)$ son la longitud del camino más corto de la ciudad i a la ciudad j, entonces el problema del agente viajero se define como la situación de encontrar un recorrido de longitud mínima , visitando cada una de las ciudades solamente una vez.

Podemos formalizar este problema como un par: (φ,d) donde φ es un conjunto finito de recorridos y d es una función de costo $d: \varphi \longrightarrow \mathbb{R}$, la que asigna un número real (la longitud del camino) para cada recorrido, y el problema es encontrar un recorrido con longitud mínima.

En nuestro caso, consideramos que todas las ciudades están conectadas entre sí, y además, definimos la función de costo como la suma de las distancias entre las ciudades, de la siguiente forma:

$$d = \sum_{i=1}^{N} D_{P(i)P(i+1)} \qquad (5.1)$$

donde P es una permutación y $P(i+1) \equiv P(1)$.

Solucionar el problema consiste en buscar una permutación P para la cual la longitud total del recorrido sea mínima.

Para encontrar la solución de este problema, como las de otros problemas de optimización combinatoria, existen muchos métodos, a los que nos referimos brevemente en el capítulo cuatro. En especial hablamos del algoritmo *simulated annealing*, el cual retomamos en este momento y lo aplicamos ahora, al problema del agente viajero. Nuestro interés en aplicar dicho algoritmo a este problema se basa en que el *simulated annealing*, un algoritmo estocástico basado en el método de Monte Carlo, ha sido reconocido como apropiado para solucionar, aproximadamente, problemas de optimización complejos, y por supuesto, a su relación con la máquina de Boltzmann. Con la aplicación del método al problema del agente viajero se logró analizar y compreder mejor como trabaja este algoritmo.

Para implementar el algoritmo *simulated annealing* en un problema de optimización combinatoria son necesarios los siguientes elementos:

- Una concisa descripción de una configuración del sistema.
- Un generador aleatorio de movimientos o reordenamientos de los elementos en una configuración.

- Una función objetivo que contenga los cambios que tienen que ser realizados.
- Un *annealing schedule* de las temperaturas y de la duración del tiempo para la cual el sistema debe desarrollarse.

Nuestra aplicación del *simulated annealing* al problema del agente viajero define el siguiente algoritmo:

Algoritmo

Dado N, el número de ciudades y D, una matriz N × N cuyos elementos D(i,j) dan las distancias entre la i-ésima ciudad y la j-ésima ciudad. Dadas además, $\{s_i\}_1^N$, $\{c_i\}_1^N$, $\{t_i\}_1^N$, permutaciones de los enteros 1,2,...,N (una permutación de enteros corresponde a una forma particular de tomar el recorrido del agente viajero). Consideraremos que s es la permutación inicial, c es la permutación actual y t es la permutación de prueba o temporal.

Entonces, el problema es encontrar una permutación c para la cual la longitud total d sea mínima.

Para ello seguimos los siguientes etapas, extrapolando la modificación propuesta en el capítulo dos para el *annealing* de la máquina de Boltzmann:

Etapa I.- Esta etapa sigue la secuencia descrita a continuación, que no es más que la aplicación del *annealing* a una permutación inicial escogida aleatoriamente:

Paso 0.- Seleccionamos arbitrariamente una permutación $\{s_i\}_1^N$. Seleccionamos un número real T (parámetro de control del *annealing*), es decir, fijamos el valor inicial del parámetro.

Paso 1.- Hacemos $c_k = s_k$ para $k = 1,2,\ldots,N$. Calculamos la longitud correspondiente para $\{c_i\}_1^N$.

Paso 2.- Generamos aleatoriamente una permutación de prueba $\{t_i\}_1^N$.

Paso 3.- Calculamos la correspondiente distancia total o longitud para la permutación de prueba: d'.

Paso 4.- Si d' < d, entonces saltamos al paso 5, si no generamos un número aleatorio num, uniformemente distriuído en 0 < num < 1. Analizamos si num < exp[-(d'-d)/T], si es menor vamos al paso 5, si no saltamos al paso 6.

Paso 5.- Hacer $c_k = t_k$ $k = 1,2,\ldots,N$ y hacer d = d'.

Paso 6.- Comprobar si la condición de equilibrio se ha alcanzado, si se alcanzó ir al paso 7, si no ir al paso 2.

Paso 7.- Comprobar si la condición de paro del algoritmo se alcanzó, si es así, ir a la **Etapa II**, si no decrementar el parámetro de control e ir al paso 2.

Etapa II.- El mínimo obtenido, que asumimos es un mínimo local, lo retenemos en la variable OPTIMO y la permutación asociada la conservamos en la variable PERM_OPTIMA.

Etapa III.- Generamos una nueva permutación inicial y realizamos la Etapa I, obtenemos un nuevo mínimo, si éste es mejor que OPTIMO lo sustituímos, al igual que PERM_OPTIMA con la permutación asociada.

Etapa IV.- Repetimos la **Etapa III**, un número de veces proporcional al tamaño del problema, suponiendo que al finali-

zar, en la variable OPTIMO está el óptimo global (realmente el mejor mínimo encontrado que puede coincidir con una mayor probabilidad con el óptimo absoluto), y en PERM_OPTIMA la permutación que lo originó.

Como ya es conocido, este algoritmo dependerá del *annealing schedule* que seleccionemos, y para cada problema se tiene que encontrar el valor apropiado del parámetro de control T. En el próximo epígrafe comentaremos el *annealing schedule* utilizado por nosotros en la aplicación del algoritmo.

5.2 ANNEALING ESPECIFICO PARA EL AGENTE VIAJERO

El *annealing schedule* puede ser desarrollado por prueba y error para un problema dado. En nuestra implementación del *annealing*, para el caso del agente viajero, el *annealing schedule* fue determinado empíricamente.

El valor inicial del parámetro de control se tomó teniendo en cuenta que todas las permutaciones tuvieran igual probabilidad de ser aceptadas. Se tomó como orientación la propuesta de Kirkpatrick [KGV82] para el ejemplo analizado en ese artículo, donde plantea que este valor inicial debe ser del orden de $N^{1/2}$.

La función de decremento del parámetro de control es la clásica propuesta por Kirpatrick, que ya referimos en el capítulo anterior, tomando la constante $\alpha \leq 0.99$.

Para realizar las primeras pruebas, se consideró la condición de parada del algoritmo únicamente dependiente del valor final de la temperatura, el cual fue prefijado.

En esta implementación se utiliza como condición o criterio de equilibrio, el hecho de realizar un número de movimientos proporcional al problema, que está determinado por el número de permutaciones que son aceptadas y el tamaño de la permutación.

Asumimos que el equilibrio es alcanzado cuando al menos una vez, cada elemento de la permutación, es decir cada posición o ciudad, fue tomada para generar una nueva permutación.

El criterio de parada es el de analizar si la función de costo, en este caso, la función de distancias, no sufre cambios para un número determinado de ciclos de parada, que hemos considerado también de manera empírica como proporcional al tamaño del problema.

Reiteramos que la calidad de la solución final obtenida por el algoritmo está determinada por la convergencia del mismo, la cual está gobernada por los parámetros del *annealing schedule*.

El comportamiento del algoritmo *simulated annealing*, como un algoritmo de aproximación, es analizada en una forma empírica, en general. Esto compromete el análisis de los tiempos de computación y de la calidad de las soluciones finales al correr el algoritmo en un conjunto grande de ejemplos o casos, para comparar los resultados con los encontrados por otra aproximación y otros algoritmos de optimización.

Sin embargo, para un problema fijo es también interesante analizar los parámetros del *annealing schedule*, pues el tiempo de computación y la calidad de la solución final son variables aleatorias, debido a la naturaleza probabilística del algoritmo.

5.3 IMPLEMENTACION Y PRUEBAS

Trataremos de resumir el algoritmo detallando los pasos presentados en la sección 5.1, enfatizando en la **Etapa I**:

Paso 0.- Seleccionamos la permutación inicial $\{s_i\}_1^N$, y seleccionamos el valor inicial de T.

Paso 1.- Hacemos $c_k = s_k$ para $k = 1,2,\ldots,N$. Calculamos la longitud correspondiente con la siguiente expresión:

$$d = D(c_N, c_1) + \sum_{k=1}^{N} D(c_k, c_{k+1}) \qquad (5.2)$$

Paso 2.- Hacemos $i = 1$

Paso 3.- Generamos aleatoriamente un entero j, $1 \leq j \leq N$, $j \neq i$.

Paso 4.- Construímos una permutación de prueba de la permutación actual, de la siguiente forma:

- Buscamos $i' = \min(i,j)$ y $j' = \max(i,j)$
- Hacemos $t_k = c_k$ para $k = 1,2,\ldots,(i-1)$
- Hacemos $t_{i'+k} = t_{j'-k}$ para $k = 0,1,2,\ldots,j'-i'$
- Hacemos $t_k = t_k$ para $k = j'+1, j'+2,\ldots,N$

Para construir esta permutación, las ciudades en la i-ésima y en la j-ésima posiciones son intercambiadas, y la ruta entre ellas es tomada en dirección inversa.

Paso 5.- Calcular la longitud para la permutación de prueba, como

$$d' = D(t_N, t_1) + \sum_{k=1}^{N} D(t_k, t_{k+1}) \qquad (5.3)$$

Si la matriz D es simétrica, entonces solamente dos términos de la ecuación (5.4) son diferentes de los términos de la ecuación (5.3). Este hecho puede ser usado para dar velocidad a los cálculos planteando: Δd y utilizando este resultado en el paso 6.

Paso 6.- Si $d' < d$, ir al paso 7, o lo que es lo mismo, aceptamos la permutación de prueba como la permutación actual, si no la configuración podrá ser aceptada, a pesar de ser peor, con probabilidad dada por $\exp[-(d'-d)/T]$, para esto generamos un número aleatorio num, uniformemente distribuído en el intervalo (0,1). Si num < $\exp[-(d'-d)/T]$, entonces vamos al paso 7, es decir, aceptamos la permutación de prueba, si no saltamos al paso 8, lo que equivale a no aceptarla.

Paso 7.- Hacemos $c_k = t_k$ $k = 1,2,\ldots,N$ y $d = d'$.

Paso 8.- Incrementamos i en uno. Si $i \leq N$ y la condición de equilibrio no se ha cumplido, ir a paso 3, de otra forma ir al paso 2.

Paso 9.- Checar si la condición de parada del *annealing* se alcanzó, si se alcanzó saltar a la **Etapa II**, si no decrementar el valor del parámetro de control e ir al paso 2.

Observese que estamos usando, implícitamente, la condición standard del *annealing* para aceptar configuraciones, en este caso permutaciones, que provoquen deterioros en la función de costo, en este ejemplo, aumentos en la función de distancia.

Los experimentos iniciales se realizaron considerando el número de ciudades pequeño. Posteriormente, se incrementó, tomándose N=12, N=24, N=42, N=48, N=100.

Para las pruebas donde el número de ciudades era pequeño, se realizó la búsqueda del mínimo global de forma exhaustiva, para comprobar el comportamiento del algoritmo.

Se tomaron valores de 0.99, 0.95, 0.90, 0.85 y 0.80 para la constante α de la función de decremento del parámetro de control, $T_{nueva} = \alpha\, T_{vieja}$.

Los valores iniciales del parámetro de control se obtuvieron por prueba y error, estando en el orden $N^{1/2}$ para todos los casos probados.

Los resultados de los experimentos realizados pueden calificarse de buenos. Aunque todas las pruebas realizadas aportaron algún elemento en la implementación, exponemos los resultados de los ejemplos realizados con 12 y 42 ciudades por considerar que el número total de ciudades es significativo y por disponer de las solución exacta para poder comparar los resultados obtenidos.

En el ejemplo de 12 ciudades, se generó la matriz de distancias no simétrica aleatoriamente. Para poder realizar las comparaciones se realizó la búsqueda exhaustiva del óptimo cuyo valor fue de 20 y la permutación asociada al mismo es la siguiente: 5 11 8 0 10 3 6 2 2 9 4 1 7. Escogimos una de las pruebas realizadas para ilustrar como trabajó el algoritmo con los valores de los parámetros siguientes:

- Temperatura inicial: 4
- Temperatura final: 0.05

- Constante α: 0.99
- Número de *annealings*: 5

Los resultados se muestran en la tabla 5.1

num annealing	distancia min
1	25
2	21
3	20
4	20
5	20

Tabla 5.1

Las pruebas realizadas con 42 ciudades brindaron soluciones, en todos los casos, muy cercanas al óptimo, que tiene un valor de 699 para la ruta mínima. Las permutaciones asociadas a esta solución exacta son de la forma $\lambda_1,\ldots,\lambda_N\ \lambda_1,\ldots,\lambda_{i-1}$ o $\lambda_1,\ldots,\lambda_i\ \lambda_N,\ldots,\lambda_{i-1}$ $i \in \{2,N\}$. La matriz de distancia empleada es simétrica, por lo tanto, estamos considerando que el camino de la ciudad i a la ciudad j es igual a la ruta de la ciudad j a la ciudad i.

En este ejemplo de 42 ciudades se experimentó con los siguientes valores para la temperatura inicial: 6, 6.5, 7, 10, 20. El mejor comportamiento se logró para el valor de 7, seguido del valor de 10. El valor final de temperatura fue variado desde 1 hasta 0.05, probándose con los valores de 0.01, 0.1, 0.5. Para la constante α se probaron los valores de 0.80, 0.85, 0.90, 0.95 y 0.99. El número de repeticiones del *annealing* fue variado, escogiendo valores de 20, 10 y 5.

A continuación presentamos, en la tabla 5.2, los resultados obtenidos de una de las pruebas realizadas, por considerarlos representativos del comportamiento del algoritmo, tomando los siguientes valores para los parámetros:

- Temperatura inicial: 7
- Temperatura final: 0.05
- Constante α: 0.99
- Número de *annealings*: 10

num annealing	distancia min
1	704
2	699
3	711
4	705
5	707
6	704
7	701
8	699
9	699
10	707

Tabla 5.2

Las permutaciones que provocaron el mínimo absoluto fueron las siguientes:

- Para el resultado obtenido en la segunda repetición es:

28 27 26 25 24 23 22 21 20 19 18 17
16 15 14 13 12 11 10 9 8 7 6 5
4 3 2 1 0 41 40 39 38 37 36 35 34 33 32 31 30 29

- Para la octava repetición el resultado es:

30 29 28 27 26 25 24 23 22 21 20 19 18 17
16 15 14 13 12 11 10 9 8 7 6 5 4 3
2 1 0 41 40 39 38 37 36 35 34 33 32 31

- Para la novena repetición se obtuvo:

14 13 12 11 10 9 8 7 6 5 4 3
2 1 0 41 40 39 38 37 36 35 34 33
32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 1

La implementación del algoritmo *simulated annealing* fue realizada en el lenguaje de programación C, utilizándose el compilador Turbo C de Borland para las pruebas en microcomputadoras XT con coprocesador matemático y en microcomputadoras 386. Además, se usó el C standard para las pruebas en una Micro VAX-2 modelo Q4. En particular, para el ejemplo de 12 ciudades se usó una estación de trabajo (*workstation*): Sistema DECstation 5000/200 PGX con procesador RISC a 25 MHZ con punto flotante y velocidad de proceso de 24 MIPS.

Para finalizar, quisiéramos comentar que recientemente se ha planteado que arquitecturas de computadoras basadas en los modelos de máquinas de Boltzmann, también pueden proporcionar el poder computacional requerido para llevar a cabo los cálculos complejos impuestos por los problemas de búsquedas cambinatorias y los problemas de optimización.

La máquina de Boltzmann puede ser usada para resolver muchos problemas de la optimización combinatoria, debido a que la estructura de muchos de ellos puede ser definida directamente en la estructura de una máquina de Boltzmann: seleccionando las conexiones correctas entre los elementos de computación (las unidades), transformando la función de costo del problema de optimización en una función de energía asociada a una máquina y tomando los pesos fijos, por lo que no existe entrenamiento para la red en una aplicación de este tipo.

CONCLUSIONES

En relación al comportamiento del algoritmo de aprendizaje de la máquina de Boltzmann, podemos plantear que durante cada ciclo de aprendizaje los ejemplos fueron fijados directamente a partir del conjunto de aprendizaje. En otras simulaciones fue añadido ruido a éstos para permitir que las unidades inviertan sus estados en la situación de inducido. Con estas dos variantes se realizaron experimentos para verificar como trabajaba el algoritmo.

En el primer experimento, cada entrada fue fijada cinco veces en una primera prueba, diez veces en una segunda prueba y quince veces en una tercera en las unidades de entrada de la red. El 50% de las salidas fueron correctas, esto indica que la máquina de Boltzmann obtuvo una comprensión de la relación entrada-salida del conjunto de aprendizaje.

En el segundo experimento, con ruido en las entradas, cada entrada del conjunto de aprendizaje fue fijada cinco, diez y quince veces en las unidades de entrada. El comportamiento de la máquina fue similar, y esto indica la capacidad de la máquina de Boltzmann para clasificar con ruido asociado. Por lo tanto, la robustez del algoritmo de aprendizaje, al menos para el ejemplo analizado, es evidente, ya que de las pruebas efectuadas podemos concluir que la selección del *annealing schedule*, el número de ejemplos por ciclo y el propio ajuste de pesos no son críticos, pues en todos los casos analizados el algoritmo trabajó de manera similar.

Como se mencionó en el capítulo tres, el algoritmo de aprendizaje original es lento, debido a que cada ciclo de aprendizaje consta de dos fases, y en cada fase, por cada ejemplo presentado hay que alcanzar el equilibrio para cada

valor de temperatura (parámetro de control del *annealing*), es decir, es lento por el tiempo necesitado para el equilibrio. Si a este hecho le sumamos que nuestra modificación conlleva repeticiones de este proceso tantas veces como unidades tenga la red, es obvio que nuestra propuesta es bastante más lenta que el algoritmo original, aunque aumenta la probabilidad de que el mínimo alcanzado esté más cerca del óptimo.

De cualquier forma, y a pesar de la lentitud del algoritmo de aprendizaje, que comparado con el popular *backpropagation* es sustancialmente más lento, obteniéndose resultados similares, debemos decir a favor de las Máquinas de Boltzmann que son más apropiadas para ponerse directamente en silicon, es decir, para implementarlas en *hardware*. Esto justifica la necesidad de las investigaciones del modelo y la necesidad de investigar como aumentar la velocidad del algoritmo.

Aunque consideramos buenos los resultados alcanzados por el algoritmo *simulated annaealing* con las modificaciones efectuadas, hay que utilizar las repeticiones de manera cuidadosa, debido a que el gasto de tiempo de CPU es alto y que la velocidad del algoritmo decrece.

Concretamente consideramos que el *annealing* modificado debe tener un ciclo de repetición proporcional al tamaño del problema, es decir, que casuísticamente se decida cuantas repeticiones del *annealing* realizar para el problema analizado.

Otros aspectos que deben señalarse en nuestra implementación del *simulated annealing*, son que los valores de *annealing schedule* son propios de los problemas resueltos, no son generales, y que las propuestas originan implementaciones de tiempo polinomial.

[AHS85] D. H. Ackley, G. E. Hinton and T. J. Sejnowski, "A Learning Algorithms for Boltzmann Machines", *Cognitive Science* 9, 147-169, 1985 o *Neurocomputing: Foundations of Research*, Edited by J. A. Anderson and E. Rosenfeld, Capítulo 38, 635-650, 1986.

[AJ85] Y. S. Abu-Mostafa and J.M. St. Jacques, "Information Capacity of the Hopfield Model", *IEEE Transactions on Information Theory 31(4)*, July 1985, 461-464.

[AK87] E. H. L. Aarts and J. H. M. Korst, Boltzmann Machines and Their Applications", Phillips Research Laboratories, *Lecture Notes in Computer Science 258* (Springer, Berlin 1987) 34-50.

[AK88-89] E. H. L. Aarts and J. H. M. Korst, "Computations in Massively parallel networks: based on the Boltzmann Machine: A review", *Parallel Computing 9* (1988/1989), 129-145.

[AKL88] E. H. L. Aarts, J. H. M. Korst and P. J. M. Van Laarhoven, "A Quantitative Analysis of the Simulated Annealing Algorithm: A Case Study for the Travelling Salesman Problem", *Journal of Statistical Physics,* Vol.50, Nos. 1/2, 1988.

[AK89] E. H. L. Aarts and J. H. M. Korst, "Boltzmann Machines for Travelling Salesman Problems", *European Journal of Operational Research 39* (1989) 79-95.

[AK91] E. H. L. Aarst and J. H. Korst, "Boltzmann Machines as a Model for Parallel Annealing", *Algorithmica 6* (1991) 437-465.

[AL85] E. H. L. Aarts and P. J. M. Van Laarhoven, "Statistical Cooling: A General Approach to Combinatorial Optimization Problems", *Philips J. Res. 40*, 193-226,1985.

[AM90] I. Aleksander and H. Morton, *An Introduction to Neural Computing*, Printed in Great Britain by T. J. Press (Padstow) Ltd., Padstow, Cornwall, Chapman and Hall,1990.

[ANF90] S. V. B. Aiyer, M. Niranjan and F. Fallside, "A Theoretical Investigation into the Performance of the Hopfield Model", *IEEE Transactions on Neural Networks*, Vol. 1, No. 2, June 1990.

[Ar87] M. A. Arbib, *Brains, Machines and Mathematics*, Edited by Springer-Verlag New York Inc, 1987.

[BA85] A. G. Barto and P. Anandan, 1985, "Pattern Recognizing Stochastic Learning Automata", *IEEE Trans. Systems, Man and Cybernetics 15*: 360-375.

[BSA83] A. G. Barto, R.S. Sutton and C. W. Anderson, 1983, "Neuronlike Adaptative Elements That Can Solve Difficult Lerning Control Problems", *IEEE Transc. Syst., Man and Cyben.* SMC-13: 834-846 o *Neurocomputing: Foundations of Research*, Edited by J. A. Anderson and E. Rosenfeld. Capítulo 32, 536-549, 1986

[BSB81] A. G. Barto, R.S. Sutton and P. Brouwer, 1981, "Associative Search Network: A reinforcement Learning Associative Memory", *Biol. Cybern. 40*: 201-211.

[Ce85] V. Cerny, "Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Algorithm", *Journal of Optimization Theory and Applications:* Vol. 45, No.1, 41-51,January 1985.

[CP43] W. S. McCulloch & Pitts, "A logical Calculus of The Ideas Immarent in Nervus Activity", *Bulletin of Mathematical Biophyics*, Vol. 5, 115-133, 1943.

[EPRV87] R. J. Mc Eliece, E. C. Posner, E. R. Rodemich and S. S. Venkatesh, "The Capacity of the Hopfield Associative Memory", *IEEE Transactions on Information Theory 33(4)*, July 1987, 461-82.

[Fe81] J. A. Feldman, 1981, "A Connectionist model of visual memory", Parallel Models of Associative Memory (G. E. Hinton and J. A. Anderson, Eds.), Lawrene Erlbaum Associates.

[Fe85] J. A. Feldman, "Connections", *BYTE*, April 1985, 277-284.

[FB82] J. A. Feldman and D. H. Ballard, "Connectionist Models and Their Properties", *Cognitive Science*, Vol.6, 205-254, 1982.

[Fu80] K. Fufushima, 1980, "Neocognitron: A Self-organization Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position", *Biological Cybernetics 36*: 193-202.

[FMI83] K. Fukushima, S. Miyake and T. Ito, "Neocognitron: A Neural Network Model for Mechanism of Visual Pattern Recognition", *IEEE Transaction on Systems Man and Cybernectics 13 (5)* 826-834 September/October 1983 o *Neurocomputing: Foundations of Research*, Edited by J. A. Anderson and E. Rosenfeld. Capítulo 31, 523-534, 1986 o *Computer Society Press Technology Series Neural Network, Artificial Neural Networks: Theorical concepts. V. Vemury*, 136-144.

[GG86] S. Geman and D. Geman, "Stochastic relaxation, Gibbs Distributions, and the Bayesian restoration of images", *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-6: 721-741 o *Neurocomputing: Foundations of Research*, Edited by J. A. Anderson and E. Rosenfeld. Capítulo 37, 611-634, 1986.

[Gi85] B. Gidas, "Nonstationary Markov Chains and Convergence of the Annealing Algorithm", *Journal of Statistical Physics*, Vol. 39, Nos. 1/2, 1985.

[Gr75] S. Grossberg, "*The Adaptative Brain I: Cognition, Learning, Reinforcement and Rhythm*" y "*The Adaptative Brain II: Vision,Vision, Speech*, Language and Motor Control", Elseiver/North-Holland, John Wiley & Sons, New York, 1975.

[He49] D. O. Hebb, *The Organization of Behavior*, John Wiley & Sons, New York, 1949.

[Hi85] G. E. Hinton, "Learning in Parallel Networks", *BYTE*, April 1985, 265-273.

[Hi89] G. E. Hinton, "Connectionist Learning Procedures", *Artificial Intelligence 40* (1989), 185-234.

[HKP91] J.Hertz, A.Krogh and R. G. Palmer, *Introduction to the Theory of Neural Computation*, Lecture Notes Vol.I Santa Fe Institute. Edited by Addisson-Wesley Publishing Company.

[HS86] G. E. Hinton and T. J. Sejnowski, "Learning and Relearning Boltzmann Machines", *Parallel Distributed Processing*, MIT, Vol. 1, Capítulo 7.

[Ho82] J. J. Hopfield, "Neural Network and Physical Systems with Emergent Collective Computational Abilities", *Proc. Natl. Acad. Sci. USA*, Vol. 79 2254-2558, Abril 1982 o *Neurocomputing: Foundations of Research*, Edited by J. A. Anderson and E. Rosenfeld. Capítulo 27, 457-464, 1986.

[HT86] J.J. Hopfield and D. W. Tank, "Computing With Neural Circuits: A Model",*Science*, Agosto 1986, Vol. 233, 625-633.

[JH87] W. P. Jones and J. Hoskins, "Back-Propagation", *BYTE*, October 1987, 155-162.

[Jo87] G. Josin, "Neural-Network Heuristics", *BYTE*, October 1987, 183-192.

[Ki84] S. Kirkpatrick, "Optimization by Simulated Annealing: Quantitative Studies", *Journal of Statistical Physics*, Vol. 34, Nos. 5/6, 1984.

[KT85] S. Kirkpatrick and G. Toulouse, "Configuration Space Analysis of Travelling Salesman Problems", *J. Physique* 46 (1985) 1277-1292.

[KGV82] S. Kirkpatrick, C. D. Gelatt,Jr. and M. P. Vecchi, "Optimization by Simulated Annealing", *Science 220*: 671-680 o *Neurocomputing Foundations of Research*. Edited by J. A. Anderson and E. Rosenfeld. Capítulo 33, 554-566, 1986.

[KA89] J. H. M. Korst and E. H. L. Aarts, "Combinatorial Optimization on Boltzmann Machine", *Journal of Parallel and Distributted Computing 6*, 331-357 (1989).

[LA87] P. J. M. van Laarhoven and E. H. L. Aarts, *Simulating Annealing: Theory and Applications*, Kluwer, Dordrecht, Netherlands, 1987.

[La90] J. Lawrence, "Untangling Neural Nets", *Dr. Dobb's Journal*, April 1990, 38-44.

[L89] M. P. Mc Laughlin, "Simulated Annealing", *Dr. Dobb's Journal*, September 1989, 26-37.

[Li87] R. P. Lippmann, "An Introduction to Computing with Neural Nets", *IEEE ASSP Magazine*, April 1987, 4-22.

[Li91] M. Livesey, "Clambing in Boltzmann Machines", *IEEE Neural Networks*, Vol. 2, No. 1, 143-148, January 1991.

[LK73] S. Lin and B. W. Kernighan, "An Effective Heuristic Algorithm for the Traveling-Salesman Problem", *Oper. Res. 21*:498-516 (1973).

[Ma87] P. K. Mazaika, "A Mathematical Model of the Boltzmann Machine", *IEEE First International Conference on Neural Networks. San Diego, California.* June 21-24, 1987, Vol.III.

[MP69] M. Minsky y S. Papert, *Perceptrons: An Introduction to Computacional Geometry*, MIT Press, Cambrige MA, 1969.

[MP86] M. Mézard and G. Parisi, "A Replica of The Travelling Salesman Problem", *J. Physique* 47 (1986), 1285-1296.

[MRT53] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, E. Teller, "Equation of State Calculations by Fast Computing Machines", *J.Chem. Phys. 21*, 1953, 1087-1092.

[OB89] K. K. Obermeir and J. J. Barron, "Time to Get Fired Up", *BYTE*, August 1989, 217-224.

[Pa88] D. Partridge, "Connectionist Networks Qua Graphs", *Comut. Math. Applic.*, Vol. 15, No. 4, 325-331, 1988.

[PS89] I. Pauberry and G. Schnitger, "Relating Boltzmann Machine to Conventional Models of Computation", *Neural Networks*, Vol. 2, 59-67, 1989.

[RG86] R. E. Randelman and Gary S. Grest, "N-City Traveling Salesman Problem: Optimization by Simulated Annealings", *Journal of Statistical Physics*, Vol. 45, Nos. 5/6, 1986.

RHW86] D. E. Rumelhart, G. E. Hinton y R. J. Williams, "Learning Internal Representations by Error Propagation", *Parallel Distributed Processing: Explorations in the Microstructuresof Cognition*, Vol.1 D. E. Rumelhart J. L. Mc Clelland (Eds.), Cambrige, MA: MIT Press 318-362 o *Neurocomputing Foundations of Research*. Edited by J. A.Anderson and E.Rosenfeld. Capitulo 41, 673-695, 1986.

[RHWi86] D. E. Rumelhart, G. E. Hinton y R. J. Williams, "Learning Representations by Back-propagation Errors", *Nature 323*:533-536 o *Neurocomputing, Foundations of Research*. Edited by J. A. Anderson and E. Rosenfeld. Capitulo 42, 696-699, 1986.

[Ri90] R. Richards, "An Efficient Algorithm for Annealing Schedules in Boltzmann Machines", *Theory Track Neural & Cognitive Scienses Track of the Proceedings of the International Joint Conference on Neural Network*, January 1990, Vol. 1, I-309-I-312.

[Ro59] R. Rosenblatt, *Principles of Neurodynamics*, New York, Spartan Book, 1959.

[RS91] F. Romeo and Alberto Sangovanni-Vincentrlli, "A Theorical Framework for Simulated Annealing", *Algorithimica 6* (1991): 302-345.

[Ru89] R. A. Rutenbar, "Simulated Annealing Algorithms: An Overview", *IEEE Circuits and Devices Magazine*, January 1989, 19-26.

[Sh88] L. Shastri, "A Connectionist Approach to Knowledge Representation and Limited Inference", *Cognitive Science 12*, 331-392 (1988).

[SR86] T. Sejnowsky and C. R. Rosberg, "NETtalk: A Parallel Network That Learn To Read Aloud", *Johns Hopkins, Uni. Technical Report JHU/EECS-86/01*, 1986 o *Neurocomputing Foundations of Research*, Edited by A. Anderson and E. Rosenfeld, Capítulo 40, 661-672, 1986.

[Su88] H. J. Sussmann, "Learning Algorithms for Boltzmann Machines", *Proceding of the 27th Conference on Decision and Control, Austin, Texas*, December 1988.

[TP89] D. S. Touretzky and D. A. Pomerleau, "What's Hidden in hidden Layers?", *BYTE*, August 1989, 227-233.

[WH60] B. Widrow and M. E. Hoff, "Adaptative Switching Circuits", *1960 IRE WESCON Conv. Record*, Part 4, 69-104, August 1960.

[WS85] B. Widrow and S. D. Stearns, *Adaptative Signal Processing*, Prentice-Hall, New Jersey ,1985.

CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS DEL I.P.N.
APARTADO POSTAL 14-740
MEXICO 07000, D.F.

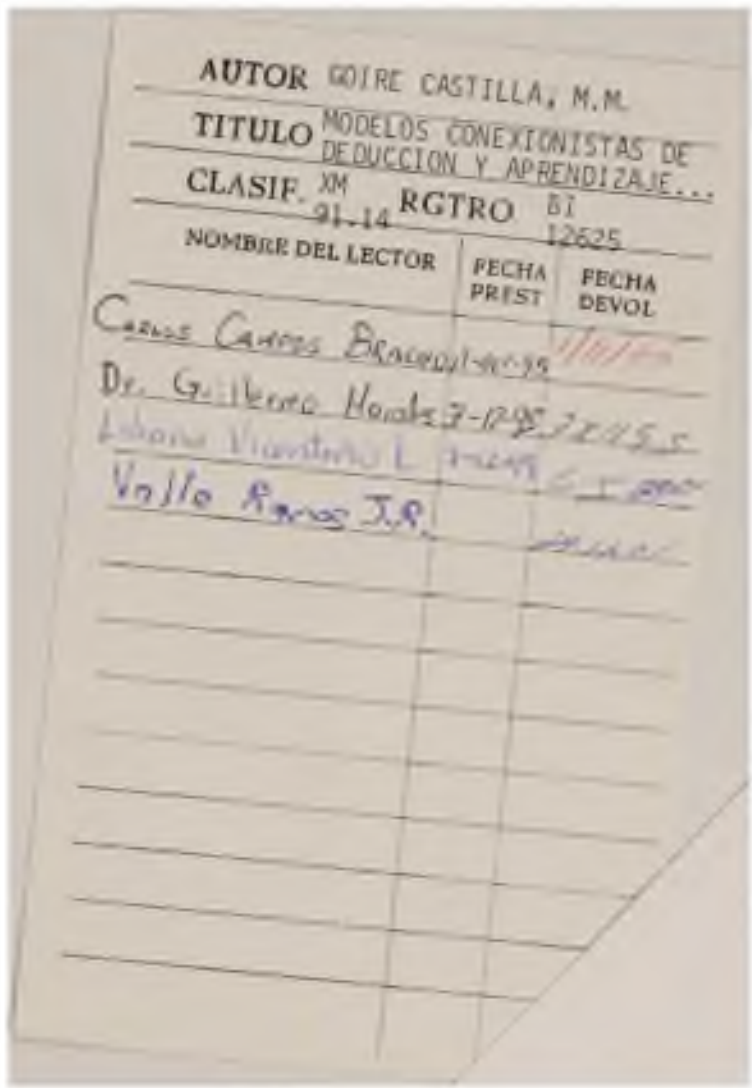
EL JURADO DESIGNADO POR LA SECCION DE COMPUTACION DEL DEPARTAMENTO DE INGENIERIA ELECTRICA, APROBO EL DIA 29 DEL MES DE Noviembre DEL AÑO DE 1991 EL TRABAJO DE TESIS "MODELOS CONEXIONISTAS DE DEDUCCION Y APRENDIZAJE AUTOMATICO".

DESARROLLADO POR EL ALUMNO: Ma. Margarita Goire Castilla

Dr. Guillermo Morales Luna
Jefe de la Sección de Computación.

Dr. Sergio V. Chapa Vergara
Coordinador Académico.

M en C. Jorge Buenabad Chávez
Profesor de la Sección.



AUTOR GOIRE CASTILLA, M.M.

TITULO MODELOS CONEXIONISTAS DE DEDUCCION Y APRENDIZAJE...

CLASIF. XM 91-14 RGTRO BI 12625

NOMBRE DEL LECTOR	FECHA PREST	FECHA DEVOL
Carlos Cantos Bracamonte	11/11/96	
Dr. Guillermo Morales [illegible]	[illegible]	
Liliana Vicentelo L. [illegible]	[illegible]	
Valle Ramos J.R.	[illegible]	