80
269

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CIENCIAS

APRENDIZAJE AUTOMATICO

T E S I S

QUE PARA OBTENER EL TITULO DE

A C T U A R I A

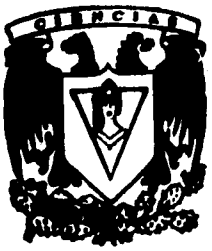
P R E S E N T A :

MARIA LUISA RAMIREZ ARRIAGA

DIRECTORA: DRA. MARIA DEL CARMEN LOPEZ LAISECA

FACULTAD DE CIENCIAS
SECCION ESCOLAR

1996



TESIS CON
FALLA DE ORIGEN

TESIS CON
FALLA DE ORIGEN

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

VNIVERSIDAD NACIONAL
AVFNºMA DE
MEXICO

M. en C. Virginia Abrín Batule
Jefe de la División de Estudios Profesionales de la
Facultad de Ciencias
P r e s e n t e

Comunicamos a usted que hemos revisado el trabajo de Tesis:

" APRENDIZAJE AUTOMATICO "

realizado por MARIA LUISA RAMIREZ ARRIAGA

con número de cuenta 6803102-6 , pasante de la carrera de ACTUARIA

Dicho trabajo cuenta con nuestro voto aprobatorio.

Atentamente

Director de Tesis Propietario	DRA. MARIA DEL CARMEN LOPEZ LAISECA
Propietario	M. EN C. ELISA VISO GUROVICH
Propietario	M. EN D. ALEJANDRO MINA VALDES
Suplente	MAT. ALVARO CASTILLO MARTINEZ
Suplente	M. EN C. VIRGINIA ABRIN BATULE

Consejo Departamental de Matemáticas
M. EN C. ALEJANDRO BRAVO MOJICA

FACULTAD DE CIENCIAS
CONSEJO DEPARTAMENTAL
DE
MATEMATICAS

A mis padres
Juan Ramírez Terán †
Clementina Arriaga Estava

A mis hermanos
Gloria, Juan Federico, María Teresa,
Jorge, Ana María y María de Jesús

Y a todas aquellas personas que han influido en mi vida mi más sincero agradecimiento.

Agradezco especialmente a la Dra. María del Carmen López Laiseca su valiosa dirección en la realización de este trabajo.

Doy las gracias al Dr. Gabriel Rivero González, jefe de la oficina de Capacitación del INSUNAVIT y a la Lic. Mercedes Reyes Arenas quienes hicieron posible la creación del seminario de tesis, mediante el cual pude titularme.

Agradezco a las siguientes personas
M. en C. Virginia Abrín Batule
M. en C. Elisa Viso Gurovich
M. en D. Alejandro Mina Valdés
Mat. Alvaro Castillo Martínez
sus valiosos comentarios y apoyo a este trabajo.

INDICE

INTRODUCCION 1

CAPITULO 1

ANTECEDENTES Y DEFINICIONES 2

1.1 ANTECEDENTES DE LA INTELIGENCIA ARTIFICIAL 3

1.2. DEFINICIONES 8

1.3 ANTECEDENTES DEL APRENDIZAJE AUTOMÁTICO 9

1.4 CONCEPTOS FUNDAMENTALES DEL APRENDIZAJE AUTOMÁTICO 11

CAPITULO 2

MODELOS SIMBÓLICOS 14

2.1 APRENDIZAJE DE MEMORIA 16

2.2 APRENDIZAJE POR AJUSTE DE PARÁMETROS 19

2.3 APRENDIZAJE CON MACRO OPERADORES 22

2.4 APRENDIZAJE POR CONCEPTO O APRENDIZAJE POR INDUCCIÓN 26

2.4.1 PROGRAMA DE APRENDIZAJE DE WINSTON 28

2.4.2 ESPACIOS DE VERSION 34

2.4.3 ÁRBOLES DE DECISION 45

2.5 APRENDIZAJE BASADO EN EXPLICACIONES 48

2.6 APRENDIZAJE POR DESCUBRIMIENTO 54

2.6.1 AM: DESCUBRIMIENTO DIRIGIDO POR LA TEORÍA 54

2.6.2 BACON: DESCUBRIMIENTO DIRIGIDO POR DATOS 60

2.6.3 DESCUBRIMIENTO POR AGRUPAMIENTO 61

CAPITULO 3

MODELOS CONEXIONISTAS 63

3.1 CONCEPTOS GENERALES SOBRE REDES NEURONALES Y

ESQUEMA GENERAL DEL APRENDIZAJE CONEXIONISTA 65

3.2 REDES DE HOPFIELD 70

3.3 PERCEPTRONES 77

3.4 REDES DE RETROPROPAGACIÓN 85

3.5 APRENDIZAJE POR REFUERZOS 94

3.6 APRENDIZAJE SIN SUPERVISIÓN Y APRENDIZAJE COMPETITIVO 95

3.7 MÁQUINAS DE BOLTZMAN 100

CONCLUSIONES 103

BIBLIOGRAFÍA 105

INTRODUCCIÓN

El campo del aprendizaje automático "estudia los metodos computacionales para adquirir nuevos conocimientos, nuevas habilidades, y nuevos caminos para organizar el conocimiento existente." [Encyclopedia of Artificial Intelligence]

Dentro de las áreas de la IA el aprendizaje automático es una de las más interesantes, sin embargo la mayor parte de la información se encuentra en articulos, por lo que si se desea tener un panorama general sobre el tema, la falta de conexión entre conceptos y métodos puede ser un inconveniente. Sólo en dos libros se encontró una presentación más estructurada sobre aprendizaje automático: el libro Inteligencia Artificial de Patrick Henry Winston que está traducido al español, pero presenta problemas de didáctica, ya que los métodos se pierden (no se apoyan) en los ejemplos, y el Artificial Intelligence de Elaine Rich que está escrito en inglés y en el cual se encuentra más ordenada la información.

El objetivo de esta tesis es presentar en forma lo más clara posible los conceptos fundamentales de los diferentes métodos de aprendizaje automático, describir algunos algoritmos y programas de aprendizaje, así como las ventajas y desventajas de los diferentes métodos.

En el capítulo 1 se presentan primero los antecedentes y algunas de las definiciones utilizadas en IA para dar un panorama general del entorno en el que se empezó a desarrollar el aprendizaje automático, y después se enuncian con un poco mas de detalle los antecedentes y definiciones correspondientes al aprendizaje automático, así como una comparación entre métodos y herramientas de la IA simbólica y la IA conexionista. En el capítulo 2 se presentan algunos de los principales métodos de aprendizaje automático dentro de lo que se conoce como modelos simbólicos. En el capítulo 3 se presentan algunos modelos de redes neuronales y métodos de aprendizaje automático conexionista. Finalmente se presentan algunas conclusiones a las que se llegó al desarrollar esta tesis.

CAPÍTULO 1

ANTECEDENTES Y DEFINICIONES

El tema de aprendizaje automático se encuentra dentro del área de Inteligencia Artificial (IA). En este capítulo se presentará una breve historia sobre el nacimiento de la IA, y del aprendizaje automático, algunas definiciones y conceptos importantes.

"Desde la antigüedad hubo un gran interés por crear una inteligencia que sirviera al hombre, este interés se avivó en el siglo XVIII con el éxito de las artes mecánicas que cautivaron la imaginación popular."[PITRAT].

Al iniciarse el siglo XX se asocian a los robots de la literatura de ciencia ficción estas inteligencias creadas por el hombre.

La IA aparece en la historia a partir de los años cincuenta con el surgimiento de las primeras computadoras, da sus primeros pasos en el campo de los juegos y de la demostración automática de teoremas.

Actualmente es considerada como una disciplina dentro de las Ciencias de la Computación. Con el desarrollo de sistemas que "razonan" como el hombre y que son capaces de "aprender", se trata de alcanzar en la actualidad el viejo sueño de la humanidad: construir máquinas que imiten en todos los aspectos a la inteligencia humana.

Se presentarán primero los antecedentes y algunas de las definiciones utilizadas en IA para dar un panorama general del entorno en el que se empezó a desarrollar el aprendizaje automático, y después tratar con un poco más de detalle los antecedentes y definiciones correspondientes al aprendizaje automático.

1.1 ANTECEDENTES DE LA INTELIGENCIA ARTIFICIAL

En esta sección se presentan hechos y conceptos que pueden considerarse como los antecedentes de la IA.

En el siglo XVII Descartes introdujo la idea de " hombre máquina ", ya que creia que algunas actividades del hombre podian ser simuladas por las máquinas pero sin ninguna inteligencia

En 1747 La Mettrie publicó L'Homme Machine. Dada su formación de médico y a los intentos de Vaucanson de construir un autómata parlante, llegó a la conclusión de que la distinción hombre máquina de Descartes no se sostenia, aunque no pudo explicar en que consistia el comportamiento inteligente del hombre.

A finales del siglo XVIII un autor anónimo publicó el primer trabajo considerado de IA, en el cual explicó un método automático para componer minuetos, el autor no contaba con una computadora pero pudo definir un algoritmo bastante sencillo que permitia producir los minuetos, lanzaba un dado y con el valor que obtenia elegía la nota siguiente entre aquellas que respetaban las reglas musicales del minueto. Al final del trabajo el autor dió un ejemplo de un minueto compuesto en esta forma. Actualmente su método es utilizado para obtener minuetos del mismo estilo de esa época en una computadora.

En la primera mitad del siglo XX se construyeron autómatas autónomos designados a menudo con nombres de animales debido a su comportamiento (tortugas, zorro, etc.) que eran capaces de detectar la luz e intentaban explorar espontáneamente su entorno.

En 1945 Konrad Zuse programó un conjunto de reglas para el juego de ajedrez.

En 1947 Alan Turing aprovechó su año sabático para considerar la idea de crear una criatura inteligente diferente al hombre; Turing vislumbró aplicaciones actuales de la IA como son la traducción automática, juegos y demostración automática de teoremas. Los logros por él alcanzados no fueron muy apreciados por sus colegas.

En 1949 Claude Shannon estableció los principios básicos de un método para el juego de ajedrez y un año después Alan Turing propuso un programa de ajedrez que había simulado a mano.

En 1953 Pierre de Latil, Albert Ducrocq, François le Lionnais, periodistas científicos publicaron diferentes trabajos sobre el pensamiento artificial y la era de los robots, con lo cual aportaron a la disciplina de la IA nuevas contribuciones debido al gran éxito que tuvieron estos trabajos.

La IA nació, entonces, debido a los resultados obtenidos conjuntamente en medicina, psicología, lógica, lingüística e informática.

En medicina por ejemplo se estudia el funcionamiento de las diversas partes del cuerpo humano, en IA los investigadores han estudiado el sistema nervioso tomando como hipótesis la fisiología de la célula nerviosa (neurona) y modelos de neuronas simuladas.

En IA se trata de programar los mecanismos que rigen el comportamiento humano.

Entre las disciplinas que han hecho posible el surgimiento de la IA está la lógica y en particular los trabajos realizados por Jaques Herbrand, quien descubrió un método que permite demostrar en forma automática ciertos teoremas matemáticos; aunque en ese tiempo no existían las computadoras, este método fue la base para los primeros programas realizados sobre el tema.

Durante la segunda guerra mundial debido al desarrollo con fines militares de los llamados dispositivos de regulación (radares, aeronáutica, entre otros) el estudio de dispositivos de control que permiten regular el funcionamiento de un aparato según las

variaciones de parámetros como temperatura, presión y orientación, etc. condujo a mecánicos, técnicos en electrónica y matemáticos a trabajar conjuntamente; de aquí nació la cibernética, cuyo objetivo era "explicar por medio de las matemáticas los fenómenos que ponen en juego los mecanismos de tratamiento de la información"[PITRAT].

El enfoque de la cibernética se ha traducido por tanto, en una visión muy matemática del comportamiento humano o animal; la mayoría de los precursores en la IA provinieron de esta disciplina (cibernética).

Por lo que respecta a la lingüística es necesaria para tratar automáticamente a las lenguas naturales.

En el caso de la informática es importante señalar que los primeros que tuvieron la idea de construir máquinas de calcular, empezando por Pascal y Babbage, no tenían en mente la idea de la IA, lo único que querían era realizar un gran número de operaciones aritméticas rápidamente.

Los informáticos consiguieron que sus máquinas, además de efectuar cálculos, pudieran manejar símbolos.

En 1956 apareció el primer programa de IA ejecutado por una computadora llamado "Logic Theorist" de Newell, Shaw y Simon. Este programa podía demostrar teoremas de la lógica de proposiciones.

En ese mismo año se reunieron en Dartmouth una docena de investigadores, entre ellos cibernéticos, que confiaban en hacer progresar la cibernética estudiando los modelos del cerebro y extendiendo las posibilidades de los mecanismos de regulación, fue en esa reunión en la que Newell y Simon presentaron su "Logic Theorist" mostrando a los participantes cómo utilizar las ventajas de manejar los símbolos de una computadora.

El nombre de Inteligencia Artificial ("Artificial Intelligence") fue propuesto y aceptado en esa reunión para designar esta nueva disciplina.

A partir de esa fecha surgió una comunidad de IA que tenía como objetivo experimentar los programas de computadoras capaces de realizar actividades hasta entonces reservadas a los seres humanos. Muy rápidamente aparecieron campos variados como juegos de ajedrez, de damas, demostración automática de teoremas, traducción automática.

Ya en 1956 existían programas que traducían del ruso al inglés y se basaban en enormes diccionarios y sólo tenían en cuenta algunos elementos de gramática. Los resultados no eran buenos porque los programas traducían mal, pero estos errores permitieron precisar los principios generales de la IA. Estos principios son :

- Hay que crear programas generales.
- Los conocimientos sobre un problema específico tienen que darse desde fuera del propio programa, de tal forma que el programa se convierte entonces en un simple "motor de inferencias" adaptable a cualquier dominio de conocimiento proporcionado como dato.

En cibernética se siguieron desarrollando modelos del cerebro y en 1958 Frank Rosenblatt presentó una máquina llamada "Perceptrón". El elemento básico de esta máquina era un modelo bastante primitivo de una neurona.

Varios de estos elementos estaban unidos formando una red a la manera de asociaciones de neuronas en el sistema nervioso. Algunas células recibían excitaciones externas, otras daban la respuesta del sistema a estímulos recibidos.

En 1965 gracias a los avances en las computadoras, se produjo un nuevo giro en la IA, surgiendo dos enfoques para realizar programas de solución de problemas:

- El primero llamado aproximación combinatoria que emplea la capacidad de la computadora para examinar el máximo de posibilidades hasta obtener la solución deseada.
- El segundo llamado aproximación humana, que se basa en el comportamiento humano. En la búsqueda de la solución se incluye

conocimiento heurístico, no únicamente conocimiento formal sobre el dominio del problema.

Al utilizar el primer enfoque la comunidad de IA tuvo que enfrentarse a las limitaciones de tiempo y espacio de las computadoras, por lo que se optó por poner el énfasis en el segundo enfoque, apoyándose en la idea de que el hombre es capaz de resolver muchos problemas recurriendo a sus conocimientos y a su experiencia; entonces la computadora debía trata de resolverlos utilizando *conocimientos formales y heurísticos.*

En 1969 tuvo lugar el primer congreso internacional de IA y a principios de 1970 apareció una revista internacional llamada "Artificial Intelligence".

Entre 1970 y 1975 se produjeron progresos importantes en dos sentidos.

- El primero, en el tratamiento de las lenguas naturales con el desarrollo de analizadores de textos.
- El segundo en la utilización de la computadora para manejar grandes cantidades de conocimientos.

A los sistemas para manejar grandes cantidades de conocimientos emulando el comportamiento de un humano experto en un dominio determinado se les denominó "sistemas expertos".

Una de las características esenciales de los sistemas expertos es la separación entre los conocimientos que les son necesarios (la base de conocimiento) y el programa que permite utilizar estos conocimientos (llamado motor de inferencia). En un programa habitual los conocimientos están mezclados con la metodología que permite utilizarlos, diciéndose que están en forma "procedural"; en un sistema experto, los conocimientos se presentan en forma "declarativa", sin modo de empleo, sin que se indique dónde, cuándo y como utilizarlos.

"Un sistema experto puede además explicar las razones de su respuesta, dando las etapas del razonamiento seguido. Se hace entonces fácil comprender por qué se ha

tomado una decisión determinada. El disponer de un sistema así no resuelve todos los problemas, pero facilita la tarea del manejo de la información."[PITRAT].

Aunque en 1965 E. Feigenbaum y otros investigadores de la universidad de Stanford, Estados Unidos realizaron un programa llamado Dendral que analizaba los resultados obtenidos por un espectrógrafo de masas y podría considerarse como el precursor de los sistemas expertos, es hasta 1974 que aparece el primer sistema experto, Mycin.

Mycin es un sistema experto capaz de diagnosticar una infección bacteriana y de prescribir los antibióticos capaces de luchar contra las bacterias que probablemente son las causantes de la infección.

Por el año de 1975, Alain Colmerauer definió el lenguaje Prolog que permite resolver un problema escribiendo un conjunto de conocimientos en forma declarativa, facilitando el trabajo del ser humano para describir el problema, ya que no tiene que preocuparse más de la forma en que esta descripción será utilizada.

En los años 80 la IA ingresa a la vida económica, con realizaciones prácticas evidentes en los campos precitados y paralelamente, un acrecentamiento notable del esfuerzo de investigación mediante proyectos muy ambiciosos, en la mayor parte de los países industrializados.

En la sección 1.3 se presentan los antecedentes relacionados directamente con el tema de interés de esta tesis, el aprendizaje automático

1.2 DEFINICIONES

Se presentan a continuación algunas de las definiciones más utilizadas en IA, debido a que no existe una definición aceptada por todos los investigadores.

"La IA es el estudio de cómo hacer que las computadoras hagan cosas, que hasta el momento la gente hace mejor."[RICH].

"La IA es una ciencia cuyo objetivo es lograr que una máquina haga todo lo que el hombre es capaz de hacer."[PITRAT].

"La IA es el estudio de las computaciones que permiten percibir, razonar y actuar."[WINSTON].

Algunas de las ramas de la disciplina de la IA son :

- Comprensión del lenguaje
- Sistemas expertos
- Aprendizaje automático
- Robótica
- Juegos

Algunas de ellas se mencionaron en la sección anterior, dejándose el aprendizaje automático para un tratamiento más detallado en secciones posteriores.

En cualquiera de las ramas de la IA los puntos fundamentales son la representación del conocimiento y su manejo.

1.3 ANTECEDENTES DEL APRENDIZAJE AUTOMÁTICO

Una de las ramas que más interés ha provocado dentro de la IA es el aprendizaje automático ya que " la habilidad para aprender es uno de los atributos más fundamentales del comportamiento inteligente ."[JOHN ANDERSON ET AL].

El campo del aprendizaje automático "estudia los métodos computacionales para adquirir nuevos conocimientos, nuevas habilidades, y nuevos caminos para organizar el conocimiento existente."[ENCYCLOPEDIA OF ARTIFICIAL INTELLIGENCE].

El estudio de este tema se iniciará con una breve historia de cómo surgió dicho campo.

El interés en programas capaces de "aprender" viene desde mediados de los 50's y los inicios de la IA. Las primeras técnicas de aprendizaje tendieron a enfocarse en las codificaciones numéricas y manejo de parámetros; ésto contrastaba con el énfasis creciente de la IA en manipulación simbólica y los métodos heurísticos; de hecho, las primeras investigaciones en aprendizaje automático se encontraban más cerca del campo automático de reconocimiento de patrones que de la IA misma; los investigadores sobre aprendizaje estaban especialmente interesados en conceptos de generalidad y trataban de construir sistemas que aprendieran con muy poco conocimiento inicial.

Esta situación continuó hasta mediados de los años 60's, cuando los investigadores de IA empezaron a desviar su atención hacia los sistemas puramente simbólicos y hacia el análisis del papel del conocimiento en el aprendizaje.

En este período se abandonó prácticamente la investigación en aprendizaje automático y sólo algunos trabajos en esta área se desarrollaron, tomando prestadas las representaciones simbólicas y los métodos heurísticos que se habían vuelto fundamentales en la IA. Fue en esta época cuando los primeros trabajos significativos de aprendizaje se llevaron a cabo y dejaron las bases para trabajos posteriores.

A finales de los años 70's surgió de nuevo el interés en el aprendizaje automático y empezó a desarrollarse rápidamente con el estudio de modelos conexionistas (redes neuronales).

Las investigaciones sobre el aprendizaje y la adquisición del conocimiento continuaron, pero esta vez trabajando el aprendizaje en el contexto de solución de problemas, así como en la formación de taxonomías, razonamiento analógico y descubrimiento automático. Métodos bien establecidos fueron usados para la construcción de elementos de aprendizaje para sistemas expertos, y nuevos métodos fueron formulados y probados constantemente.

Una fracción sustancial de los investigadores en aprendizaje siempre ha estado interesada en el aprendizaje humano, y ésto llevó a un nuevo periodo, cuando el número de publicaciones sobre aprendizaje automático creció dramaticamente.

1.4 CONCEPTOS FUNDAMENTALES DEL APRENDIZAJE AUTOMATICO

Las primeras definiciones de aprendizaje automático se tomaron de la psicología y la pedagogía, como por ejemplo.

Se considera el aprendizaje como " Un cambio relativamente permanente en el organismo que resulta de la práctica y produce cambios en el comportamiento." [MUSSEN ET AL].

"El aprendizaje puede ser entendido como la internalización de pautas de conducta que resulta de haber participado en un proceso intencionado de enseñanza." [QUESADA].

"El aprendizaje consiste en adquirir nuevas formas para hacer las cosas o para satisfacer los deseos."[STATON]

Más adelante, ya en base a experiencias relacionadas con IA surgen definiciones como:

"El aprendizaje, entendido como la incorporación de conocimientos obtenidos durante la ejecución de la tarea y a partir de sus resultados."[GALINDO ET AL].

"EL aprendizaje se basa en el hecho de acoplar nueva información al conocimiento ya adquirido."[WINSTON].

"El aprendizaje se basa en la extracción de una regularidad útil de los datos." [WINSTON].

"El aprendizaje denota cambios en el sistema que son adaptativos en el sentido de que ellos permitieron al sistema hacer la misma tarea o tareas diseñadas para la misma población más eficientemente y más efectivamente la siguiente vez."[SIMON].

Estas son sólo algunas definiciones encontradas sobre aprendizaje, sin que alguna de ellas sea aceptada por todos los investigadores del área.

Dentro del aprendizaje automático se han desarrollado dos formas de abordar el problema. El esquema de aprendizaje simbólico basado en métodos de representación y búsqueda de Inteligencia Artificial tradicional y dentro de este esquema se analizan los siguiente tipos de aprendizaje:

- Aprendizaje de memoria.
- Aprendizaje por ajuste de parámetros.
- Aprendizaje con macro operadores.
- Aprendizaje por concepto o aprendizaje por inducción.
- Espacios de versión.
- Árboles de decisión.
- Aprendizaje basado en explicaciones.
- Aprendizaje por descubrimiento.

Y el esquema de aprendizaje conexionista que se caracteriza por estudiar la forma como trabaja el cerebro humano simulado con la ayuda de redes neuronales artificiales. Dentro de este esquema se estudiarán diferentes tipos de redes neuronales simuladas como son:

- Redes Hopfield
- Perceptrones
- Redes de retropropagación
- Redes neuronales de alimentación positiva
- Máquinas de Boltzman

y los siguientes métodos de aprendizaje:

- Aprendizaje mediante entrenamiento de perceptrones.
- Aprendizaje mediante entrenamiento de redes neuronales.
- Aprendizaje por refuerzos.

- Aprendizaje sin supervisión y
- Aprendizaje competitivo.

A continuación se describirán con detalle, y en algunos casos con un ejemplo, los métodos arriba mencionados.

CAPÍTULO 2

MODELOS SIMBÓLICOS

Como se mencionó en la introducción, el objetivo principal de este trabajo es presentar en forma lo más clara posible los conceptos fundamentales de los diferentes métodos de aprendizaje automático, describir algunos algoritmos y programas de aprendizaje, así como las ventajas y desventajas de los diferentes métodos.

Una de las críticas más frecuentes a la IA es que las máquinas no puede decirse que sean inteligentes hasta que sean capaces de aprender a hacer cosas nuevas y adaptarse a nuevas situaciones, mas que simplemente hacer las cosas como se les dijo que las hagan. Existe una pequeña interrogante de si la habilidad a adaptarse a nuevos entornos y a resolver nuevos problemas es una característica importante de los entes inteligentes. ¿Se pueden simular estas habilidades en programas? Ada Augusta Lovelace, quien fuera una de las primeras filósofas de la computación, escribió lo siguiente: "La Máquina Analítica no tiene pretensiones de originar nada. Puede hacer cualquier cosa que nosotros *sepamos como ordenarle que haga*."[Lovelace, 1961].

Algunos críticos de tA interpretaron esta idea como que las computadoras no pueden aprender. Pero el aprendizaje cubre un amplio rango de fenómenos como son el refinamiento de habilidades y la adquisición del conocimiento. La mayoría de los programas de IA consideran el conocimiento como su fuente de poder. El conocimiento es generalmente adquirido a través de la experiencia, y tal adquisición incluye muchas actividades diferentes; simplemente, el almacenamiento de la información, o aprendizaje de memoria, es la principal de las actividades del aprendizaje. La mayoría de los programas de IA son capaces de mejorar su desempeño substancialmente con técnicas de aprendizaje de memoria, como por ejemplo el programa para jugar damas de Samuel[1963] que se analizará más adelante.

Otra forma de aprender es siguiendo consejos de los demás.

Las personas también aprenden através de sus propias experiencias en la solución de problemas: después de resolver un problema complejo, al recordar la estructura del problema y los métodos usados para resolverlo, si enfrentan de nuevo el mismo problema, pueden resolverlo más eficientemente, es más, llegan a generalizar sus experiencias para resolver problemas análogos más fácilmente. En contraste con el aprendizaje a partir de consejos, aprender de experiencias de solución de problemas no frecuentemente involucra reunir conocimiento que no estaba disponible previamente en el programa de aprendizaje, es decir que el programa recuerda sus experiencias y generaliza de ellas, pero no necesariamente las agrega a su base de conocimiento, como lo haría un programa que recibe consejos.

Los programas que aprenden de experiencias de solución de problemas pueden ser capaces de llegar a soluciones cualitativamente mejores en el futuro.

Otra forma de aprendizaje es la que involucra estímulos del exterior en forma de ejemplos, y que usualmente está supervisada por un profesor que indica si existe error en las respuestas del programa a problemas en los que utilizó el conocimiento adquirido. A este tipo de aprendizaje se le conoce como *aprendizaje a partir de ejemplos*. Una de las aplicaciones más importantes de este método es la definición de clases de objetos en problemas de clasificación. Sin embargo, algunas veces un programa puede descubrir cosas sin la ayuda de un profesor a lo que se le llama *aprendizaje sin supervisión*.

Los investigadores de IA han propuesto algoritmos para implementar los diferentes tipos de aprendizaje descritos anteriormente y que se estudiarán en este capítulo.

2.1 APRENDIZAJE DE MEMORIA

Cuando una computadora almacena información se está llevando a cabo una forma rudimentaria de aprendizaje, ya que este acto permite al programa realizar mejor sus funciones en el futuro. En ocasiones se guardan valores para no tener que recalcularlos, ya que esto es más costoso que accesarlos: a estos valores se les conoce como datos ocultos. El ocultamiento ha sido usado en programas de IA para producir algunas mejoras sorprendentes en desempeño y se les conoce como aprendizaje de memoria.

Uno de los primeros programas que utilizó este tipo de aprendizaje fue el programa de Samuel presentado en 1963. Este programa aprendía a jugar damas lo suficientemente bien como para vencer a su creador. Conjugaba dos tipos de aprendizaje: aprendizaje de memoria y aprendizaje por ajuste de parámetros que se verá en la siguiente sección.

Para determinar qué movimientos realizar, el programa analiza las posibles tiradas en su juego y las posibilidades de respuesta del contrincante. La forma de estructurar esta información es un *espacio de estados* representado en un árbol de juego. El primer nodo corresponde a la situación actual del tablero, cada nodo del primer nivel corresponde a la situación del tablero dependiendo de una tirada llevada a cabo por la computadora; en el siguiente nivel se tendrán la situación del tablero ante las diferentes respuestas del contrincante, y así sucesivamente.

El procedimiento de búsqueda que usa el programa de Samuel para explorar el árbol de juego y seleccionar la mejor tirada es el *minimax*, ya que debido a las restricciones de tiempo y espacio no pueden expanderse tantos niveles como pudieran requerirse y además el número de niveles necesarios depende de la situación del tablero. El método minimax consiste en calcular por medio de una función el valor de

2.1 APRENDIZAJE DE MEMORIA

Cuando una computadora almacena información se está llevando a cabo una forma rudimentaria de aprendizaje, ya que este acto permite al programa realizar mejor sus funciones en el futuro. En ocasiones se guardan valores para no tener que recalcularlos, ya que esto es más costoso que accesarlos; a estos valores se les conoce como datos ocultos. El ocultamiento ha sido usado en programas de IA para producir algunas mejoras sorprendentes en desempeño y se les conoce como aprendizaje de memoria.

Uno de los primeros programas que utilizó este tipo de aprendizaje fue el programa de Samuel presentado en 1963. Este programa aprendía a jugar damas lo suficientemente bien como para vencer a su creador. Conjugaba dos tipos de aprendizaje: aprendizaje de memoria y aprendizaje por ajuste de parámetros que se verá en la siguiente sección.

Para determinar qué movimientos realizar, el programa analiza las posibles tiradas en su juego y las posibilidades de respuesta del contrincante. La forma de estructurar esta información es un *espacio de estados* representado en un árbol de juego. El primer nodo corresponde a la situación actual del tablero, cada nodo del primer nivel corresponde a la situación del tablero dependiendo de una tirada llevada a cabo por la computadora; en el siguiente nivel se tendrán la situación del tablero ante las diferentes respuestas del contrincante, y así sucesivamente.

El procedimiento de búsqueda que usa el programa de Samuel para explorar el árbol de juego y seleccionar la mejor tirada es el *minimax*, ya que debido a las restricciones de tiempo y espacio no pueden expanderse tantos niveles como pudieran requerirse y además el número de niveles necesarios depende de la situación del tablero. El método minimax consiste en calcular por medio de una función el valor de

"deseabilidad" de cada uno de los nodos terminales del árbol y propagar los valores hacia los padres; esto se hace asignando al padre el valor máximo o mínimo de los valores de sus hijos, dependiendo si se trata de su turno o del contrincante.

El programa de Samuel al terminar la propagación de los valores hacia atrás calcula un valor para la posición representada por la raiz del árbol y la guarda para seleccionar el nodo del primer nivel que corresponde a la tirada que más conviene junto con su valor en una tabla de jugadas. Esta situación se muestra en las figuras 2.1a y 2.1b .

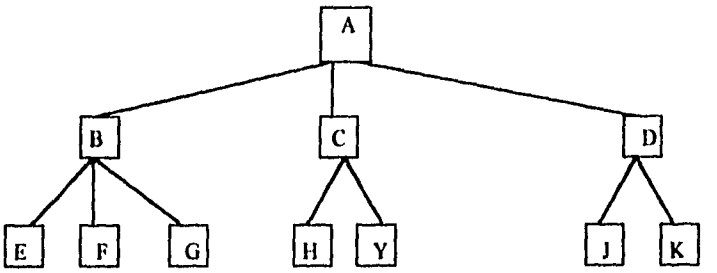


Tabla de jugadas A : 10

Figura 2.1(a) Árbol de juego

Si en un juego posterior se presenta A como un nodo terminal, figura 2.1b, en vez de usar la función de evaluación para calcular su valor, toma el valor guardado para A en la tabla, esto crea el efecto de haber analizado algunos niveles adicionales a partir de A.

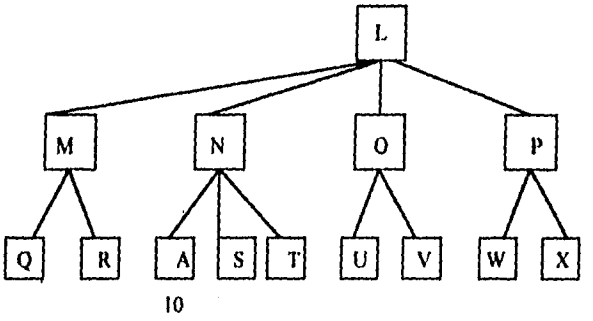


Figura 2.1(b) Nuevo Árbol de juego

El aprendizaje de memoria en el programa de Samuel consiste en guardar posiciones del tablero y sus valores asociados, para no tener que recalcular estos valores si vuelve a encontrarse en la misma situación.

El aprendizaje de memoria es muy fácil, y aunque no implica ninguna capacidad sofisticada de solución de problemas, requiere de algunas capacidades que se volverán muy importantes en sistemas de aprendizaje más complejos.

Estas capacidades son:

- Almacenamiento organizado de información, para ser más eficientes al usar valores almacenados en lugar de recalcularlos debe de existir una forma de accesar el valor guardado rápidamente. En el programa de Samuel, la información se almacena con respecto a un índice calculado a partir de algunas características importantes de la situación del tablero por ejemplo el número de piezas. Conforme la complejidad de la información almacenada aumenta, son necesarias técnicas más sofisticadas.

- Generalización - El número de objetos distintos que pueden potencialmente guardarse puede ser muy grande. Para mantener manejable el número de objetos guardados dentro de un nivel, es necesario algún tipo de generalización. En el

programa de Samuel, por ejemplo, el número de objetos distintos que pueden ser almacenados es igual al número de posiciones diferentes del tablero que pueden surgir en un juego. Sólo unas cuantas formas de generalización simples fueron usadas en el programa de Samuel para reducir ese número, por ejemplo, una de las reducciones en almacenamiento se obtiene al manejar simetrías, pues entonces de todas las posiciones sólo se almacena la mitad. De nuevo, conforme la complejidad del proceso de aprendizaje aumenta, también lo hace la necesidad de generalización.

El éxito del aprendizaje de memoria depende de una organización de conocimientos bien estructurada.

2.2 APRENDIZAJE POR AJUSTE DE PARÁMETROS

El aprendizaje por ajuste de parámetros no es una solución a todos los problemas de aprendizaje, pero es frecuentemente una técnica útil tanto en situaciones donde hay muy poco conocimiento adicional disponible, como en programas en los cuales puede combinarse con métodos que manejan más conocimiento sobre el dominio.

Numerosos programas de juegos cuentan con una función de evaluación de la forma $p_1c_1 + p_2c_2 + ... + p_nc_n$, en la cual una variedad de factores o características c_i, tales como el número de piezas de ventaja y su movilidad, se combinan en un solo valor que refleja la deseabilidad de una posición particular del tablero. En programas de clasificación de patrones habitualmente se toman en cuenta varias características y su importancia, definida en la función como un *peso* p_i asociado a cada característica, para determinar la categoría correcta en la cual un estímulo dado debe ser clasificado. En el diseño de este tipo de programas es difícil saber *apriori* qué peso debe ser

asignado a cada característica que se está usando. A continuación se presenta la idea fundamental de un algoritmo de aprendizaje por ajuste de parámetros.

Algoritmo de aprendizaje por ajuste de parámetros:

Dadas : Una serie de características y un conjunto de ejemplos o casos.
Calcular: El peso que debe ser asignado a cada característica.

1. Asignar un peso arbitrario a cada una de las características.
2. Dejar que el programa modifique los pesos de las características en base a sus experiencias en la solución de problemas presentados como ejemplos.

En general, en las fuciones de evaluación las variables corresponden a los valores de las características y los coeficientes a los pesos asignados a cada una de esas características.

La pregunta obligada en este tipo de aprendizaje es: ¿Cuándo debe ser incrementado el valor del peso de una característica y cuándo debe ser decrementado?. Una segunda pregunta es: ¿Qué tanto debe ser cambiado el valor? la respuesta a la primera pregunta es que los pesos de las características que llevan a una solución final adecuada deben ser incrementados, mientras que los coeficientes de los términos que llevan a una salida incorrecta deben ser decrementados. En algunos dominios es fácil hacerlo, por ejemplo, si un programa de clasificación de patrones usa su función de evaluación para clasificar una entrada y logra la respuesta correcta, pues entonces todos los coeficientes de los términos que predijeron esa respuesta deben incrementarse. En un programa de juegos el problema es más difícil porque el programa no llega a ninguna retroalimentación concreta de los movimentos individuales ya que hasta el final del juego sabe si ganó o no. Muchos movimientos contribuyeron al final obtenido, y aunque el programa gane, se pudieron haber hecho muchos movimientos

inadecuados en el camino. El problema de asociar apropiadamente la responsabilidad a cada uno de los pasos que condujeron a una sola salida se conoce como *problema de asignación de crédito.*

El programa de damas de Samuel [Samuel, 1963] utiliza este tipo de aprendizaje, además del aprendizaje de memoria tratado anteriormente, y proporciona un buen ejemplo de su uso. Como función de evaluación el programa usa un polinomio de la forma

$$c_1 t_1 + c_2 t_2 + \ldots + c_{16} t_{16}$$

Los términos t_i corresponden a los valores de las dieciséis características que contribuyen a la evaluación. Los términos c_i son los coeficientes (pesos) asignados a cada uno de esos valores. Como evolución del aprendizaje, los valores c_i cambian.

El programa de Samuel utiliza una técnica, aunque imperfecta, para resolver el problema de asignación de crédito: asume que los valores iniciales seleccionados para los coeficientes son lo suficientemente buenos como para que la función de evaluación total produzca valores que correspondan a medidas razonables del valor correcto, aún si no son tan adecuadas como se espera. Entonces esta función de evaluación puede ser usada para proporcionar retroalimentación para ella misma. Una secuencia de movimientos que lleva a posiciones con valores altos puede ser considerada buena y los términos en la función de evaluación que la sugieren pueden ser reforzados.

Debido a que al aplicar el ajuste de parámetros puede suceder que no exista mejora en el desempeño del programa, se proporciona una prueba adicional del progreso que se ha logrado, y se realizan cambios en los pesos de tal manera que progrese:

- Cuando el programa está en el modo de aprendizaje, juega contra otra copia de sí mismo. Sólo una de las copias cambia su función de evaluación durante el juego; la otra permanece fija. Al final del juego, si la copia con la función

modificada gana, entonces la función modificada es aceptada. Sino, la función anterior se retiene. Si esto último sucede varias veces, entonces un cambio drástico se hace a la función en un intento por conseguir que el proceso siga en una dirección más provechosa.

- Periódicamente un término de la función de evaluación es eliminado y sustituido por otro. Esto es posible porque, aunque el programa usa sólo 16 características cada vez, se dispone de cerca de 38. Este remplazo se distingue del resto del proceso de aprendizaje porque crea un cambio repentino en la función de evaluación, mas que un cambio gradual en sus pesos.

Este proceso de aprendizaje por modificaciones sucesivas de los pesos de los términos en una función de evaluación tiene bastantes limitaciones; la mayoría surgen por su falta de explotación de conocimiento adicional sobre la estructura del problema con el cual se está tratando y las relaciones lógicas entre los componentes del problema.

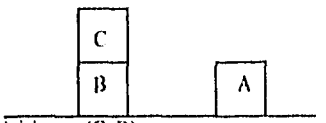
2.3 APRENDIZAJE CON MACRO OPERADORES

Un *macrop* es una secuencia de acciones que se aplicará para llevar a cabo una tarea determinada o resolver algún problema particular, es decir, un *macrop* es un plan con un objetivo determinado que en general corresponde a una submeta dentro de la meta principal a alcanzar.

Las condiciones para poder aplicar o hacer uso de un *macrop* son las condiciones iniciales del problema a resolver y sus postcondiciones corresponden a la meta por alcanzar. En su forma más simple, el ocultamiento de planes (secuencias de acciones) previamente determinados es similar al aprendizaje de memoria.

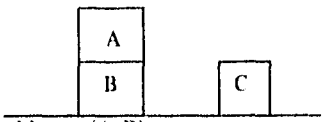
Los macrops se usaron por primera vez en un sistema de solución de problemas llamado STRIPS [Fikes and Nilsson, 1971, Fikes et al, 1972] El componente de aprendizaje en STRIPS después de cada proceso de solución de problemas, toma el plan que lleva a la solución y lo guarda como un *macrop*.

En el siguiente ejemplo de solución de problemas de STRIPS se supone que se da una situación inicial en un conjunto de bloques en el cual *en*(C, B) y *en*(A, Mesa) son ambos verdaderos como se muestra en la figura 2.2(a).



Inicio: *en*(C, B)

Figura 2.2(a)



Meta: *en*(A, B)

Figura 2.2(b)

STRIPS puede alcanzar la meta *en*(A, B) (figura 2.2b) encontrando un plan con los cuatro pasos {*levanta*(C, B), *baja*(C), *toma*(A), *encima*(A, B)}. STRIPS ahora construye un *macrop* con las precondiciones *en*(C, B) y *en*(A, Mesa) y como postcondiciones *en*(C, Mesa) y *en*(A, B). El cuerpo del macrop consiste en los cuatro pasos antes mencionados. En planes futuros STRIPS podrá utilizar este macro-operador como usaría cualquier otro operador.

Como rara vez STRIPS trata dos veces exactamente el mismo problema (los nuevos problemas serán diferentes de problemas anteriores) es conveniente que el programa de solución de problemas haga un uso más eficiente del conocimiento que ganó de experiencias anteriores. Strips generaliza macrops antes de guardarlos para resolver problemas análogos. La idea más simple para la generalización es remplazar todas las constantes del macro-operador por variables. En vez de guardar el macrop descrito en el párrafo anterior, STRIPS generaliza el plan que consiste de los pasos *levanta*(x_1, x_2), *baja*(x_1), *toma*(x_3), *encima*(x_3, x_2), donde x_1, x_2 y x_3 son variables

que en un momento dado podrán reemplazarse por nombres de bloques. Este plan se guarda con precondiciones *en*(x_1, x_2) y *en*(x_3, Mesa) y postcondiciones *en*(x_1,Mesa) y *en*(x_2, x_1). Tal macrop puede ahora aplicarse en una variedad de situaciones.

Sin embargo la generalización no es fácil, pues en ocasiones las constantes deben conservar sus valores específicos. Supongamos que nuestro dominio incluye un operador llamado *pon encima de* B(*x*), con condiciones previas de que tanto *x* como B no tengan nada encima, y con condiciones posteriores *en*(x, B). Si se considera el mismo problema que arriba:

STRIPS llega al plan {*levanta*(C, B), *baja*(C), *pon encima de*_B(A)}. Generalizando este plan y guardándolo como un macrop, la condición inicial se convierte en *en*(x_1, x_2) y el plan mismo se convierte en {*levanta*(x_1, x_2), *baja*(x_3), *pon encima de* B(x_1)}. Ahora, si se encuentra un problema un poco diferente: como el de la figura 2.3(a) que aparece con situación inicial *en*(E, C) y *en*(D, B) y meta final *en*(A, C), va a proceder como sigue:

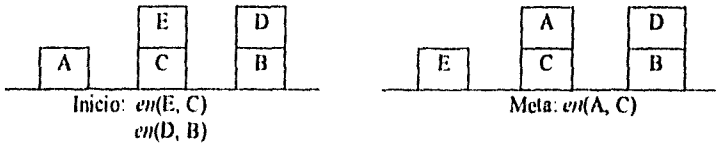


Figura 2.3(a)

El macrop generalizado que se acaba de guardar parece adecuado para resolver este problema si hacemos x_1 = A, x_2 = C, y x_3 = E, sus precondiciones son satisfechas y se construye el plan {levanta(E, C), *baja*(E), *pon encima de*_B(A)}. Este plan no será adecuado ya que la postcondición del macrop está sobregeneralizada dando como resultado que esta operación sea sólo útil para apilar bloques en B, que no es lo que se requiere en este ejemplo.

En este caso, esta dificultad será descubierta cuando trate de realizarse el último paso ya que a pesar de que se eliminó lo que está encima de C para colocar A, se fracasa al tratar de colocar A encima de B, que es donde el plan del macrop iba a tratar de colocarlo, puesto que al no estar B despejado *pon encima de*_B no puede ejecutarse. Si B hubiese estado despejado el macrop hubiera llevado a cabo la acción hasta el final, pero no hubiera alcanzado la meta establecida.

En realidad STRIPS usa un procedimiento de generalización más complejo: primero todas las constantes son remplazadas por variables y después para cada operador en el plan, reevalúa sus precondiciones. En el ejemplo de la figura 2.3(a) la precondición de los pasos 1 y 2 del plan de generalización son satisfechas, pero la única forma de asegurar que B esté despejado para el paso 3 es asumir que el bloque x_2 sea B, que debe ser realizado por el operador *levanta*. Probando el plan de generalización, STRIPS localiza restricciones de este tipo.

Un trabajo más reciente sobre macro-operadores es el de Korf [1985b] en donde se indica que el conjunto de problemas para los cuales los macro-operadores son adecuados es exactamente el que consiste de aquellos problemas con *submetas no seriales*. La no seriabilidad significa que al trabajar en una submeta no necesariamente interfiere con la solución previa de otra submeta. Los macro-operadores pueden ser útiles en problemas de planeación no lineal, ya que si un macro-operador produce un pequeño cambio en el Universo del problema, puede ocasionar cambios locales indeseables.

Por ejemplo considere el rompecabezas con 8 piezas; una vez que el programa ha colocado correctamente las primeras 4 piezas, es difícil, y en ocasiones imposible, colocar en su lugar la 5a. sin cambiar las 4 primeras. Como en general el cambiar las submetas resueltas previamente es detectado como una acción inadecuada por las funciones de evaluación heurísticas, pués el número de piezas bien colocadas se reduce

lo que se considera un retroceso en la solución, el uso de macro-operadores puede mejorar el desempeño del programa

En el ejemplo del rompecabezas con 8 piezas, se puede utilizar un macro-operador, que consista en una secuencia de operadores para colocar la 5a. pieza sin mover ninguna de las cuatro primeras externamente (aunque ellas se hayan movido dentro del macrop mismo).

Korf [1985b] da un algoritmo para recibir un conjunto completo de macro-operadores que contrasta con el de STRIPS en que construye sus macrops gradualmente de la experiencia. El algoritmo de Korf corre en un tiempo proporcional al tiempo que toma resolver un problema sencillo sin macro-operadores.

2.4 APRENDIZAJE POR CONCEPTO O APRENDIZAJE POR INDUCCIÓN.

"*Clasificación*" es el proceso de asignar a un "*elemento*" el nombre de la "*clase*" a la que pertenece.

Una de la aplicaciones fundamentales del aprendizaje es la de construir programas de clasificación que puedan generar sus propias definiciones de clases a partir de un conjunto de ejemplos.

Las técnicas usadas para esta tarea dependen de la forma en que los elementos a clasificar sean descritos.

La clasificación es un componente importante de muchas tareas de solución de problemas. En su forma más simple se presenta como una tarea de reconocimiento directo. Un ejemplo de esto es el reconocimiento de letras del alfabeto, de la clase de un animal etc. Frecuentemente la clasificación aparece como parte de otra operación. Por ejemplo, si se considera un sistema de solución de problemas que contiene la siguiente regla de producción:

Si: la meta actual es llegar del lugar A al lugar B, y hay una Pared separando los dos lugares.

Entonces: ver si hay Puerta en la pared y pasar por ella

Para usar esta regla exitosamente la rutina de comparación del sistema debe ser capaz de identificar un objeto Puerta; sin esto, la regla nunca podria aplicarse.

Para definir las clases de objetos pueden utilizarse dos tipos de enfoque.

• Enfoque estadistico: consiste en aislar un conjunto de caracteristicas relevantes en los objetos de cada clase y definir los objetos a partir de una suma ponderada de los valores de sus caracteristicas $c_1f_1 + c_2f_2 + c_3f_3 + \ldots$ Cada clase queda entonces determinada en un rango de valores de la función que caracteriza sus objetos.

Esta función es muy similar a los valores de funciones frecuentemente utilizadas en otras situaciones, tales como los juegos.

Cada f_i corresponde al valor de un parámetro relevante, y cada c_i representa el peso asociado a la f_i correspondiente. Los pesos negativos pueden ser usados para indicar caracteristicas cuya presencia usualmente constituye una evidencia negativa para una clase dada. Por ejemplo, si la tarea es la predicción del tiempo, algunos parámetros pueden ser la lluvia y la localización de frentes fríos. Diferentes funciones pueden definirse para combinar estos parámetros y predecir clima soleado, nublado, lluvioso o nevado.

• Enfoque estructural: se aisla el conjunto de caracteristicas que son relevantes a un dominio de una tarea para definir cada objeto como una estructura compuesta de estas caracteristicas; los objetos pertenecientes a una misma clase

tendrán esctructuras análogas. Por ejemplo, si la tarea es identificar animales se almacena el conocimiento del dominio como una estructura, con varias características representando atributos como el color, la longitud del cuello, las plumas, etc.

Hay ventajas y desventajas en cada uno de estos enfoques generales. El enfoque estadistico es frecuentemente más eficiente que el enfoque estructural, aunque este último es más flexible y más fácil de extender

En el enfoque estadistico a pesar de la forma en que las clases tienen que ser descritas, es difícil construir a mano una buena definición de clases, en particular en dominios que no están bien definidos o que cambian rápidamente, por lo que la idea de construir un programa de clasificación que pueda desarrollar sus propias definiciones de las clases es un reto interesante.

A continuación se describen 3 métodos de aprendizaje por inducción: el método de Winston, de espacio de versiones y los árboles de decisión.

2.4.1 PROGRAMA DE APRENDIZAJE DE WINSTON

Como se mencionó arriba, en Winston se propone un programa de aprendizaje por inducción basado en un enfoque estructural, cuya meta es definir clases de objetos en un dominio simple de bloques. Por ejemplo, apreader los conceptos de Casa, Tienda y Arco como se muestran en la figura 2.4

La figura muestra un ejemplo parecido para cada concepto. Un *objeto parecido* es un objeto que no es un caso particular del objeto en cuestión pero que es muy similar.

El programa usa procedimientos para analizar el dibujo y construye una representación utilizando una red semántica con la descripción estructural del (de los)

objeto(s). La descripción estructural del objeto es proporcionada como dato de entrada al programa de aprendizaje.

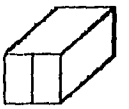
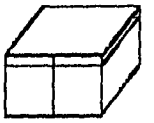
	CONCEPTO	SIMILARES
CASA		
TIENDA		
ARCO		

Figura 2.4: Algunos conceptos del mundo de Bloques.

Un ejemplo de tal descripción estructural para la Casa de la figura 2.4 se encuentra en la figura 2.5(a).

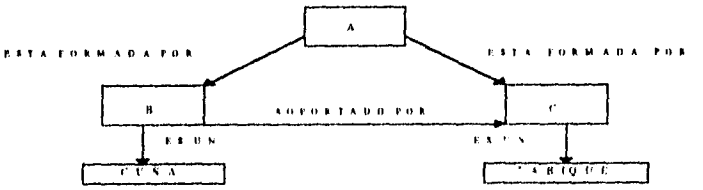


Figura 2.5(a): Descripción estructural de casa

El nodo A representa el objeto, que está compuesto por dos partes: el nodo B, una *Cuña*, y el nodo C un *Tabique*. Las figuras 2.5(b) y 2.5(c) muestran descripciones de las dos estructuras *Arco* de la figura 2.4. Estas descripciones son idénticas excepto por el tipo de objeto de encima, uno es un *Tabique* mientras que el otro es una *Cuña*. Los dos objetos que los soportan están relacionados no solamente por ligas *a-la-izquierda-de* y *a-la-derecha-de*, sino también por ligas *no-coincide-con*, lo que nos indica que los dos objetos *no-coinciden*. Dos objetos coinciden entre si, si tienen caras que se tocan y tienen un lado común. La relación *coincidir* es crítica en la definición de un Arco. Es la diferencia entre la estructura del primer arco y la estructura del objeto similar mostrado en la figura 2.4.

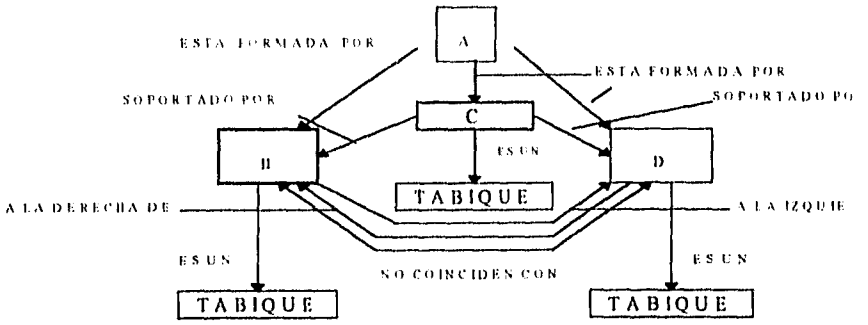


Figura 2.5(b): Descripción estructural de arco

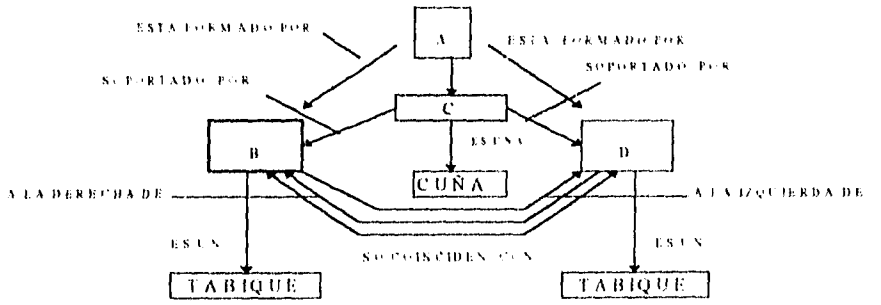


Figura 2.5(c): Descripción estructural de arco

El enfoque básico que el programa de Winston tonta para el problema de formación de concepto puede ser descrito como sigue:

1. Empieza con una descripción estructural de un caso conocido de concepto. Nombra a esa descripción *la definición del concepto*.
2. Examina descripciones de otros casos conocidos del concepto Generaliza la definición para incluirlos.
3. Examina descripciones de objetos parecidos del concepto. Restringe la definición para excluirlos.

Los pasos 2 y 3 se pueden intercambiar. Estos pasos se realizan en un proceso de comparación en el cual similitudes y diferencias entre estructuras pueden ser detectadas. Este proceso debe funcionar como cualquier otro proceso de comparación.. Dado que las diferencias y las similitudes deben ser detectadas, el procedimiento debe analizar tanto coincidencia exacta entre definición y ejemplo como también coincidencia aproximada entre ambos.

El resultado del procedimiento de comparación es una estructura que especifica los elementos comunes entre dos ejemplos de entrada. Esta estructura contiene un conjunto de notas de comparación que describen similitudes y diferencias específicas entre entradas.

Un ejemplo de la aplicación de este método, se puede observar a continuación através del proceso de aprendizaje de lo que es un Arco.

Si se supone que la descripción de Arco de la figura 2.5(b) se presenta primero, el programa la convierte en la definición del concepto de Arco Si la descripción de la figura 2.5(c) se presenta después, la rutina de comparación devolverá una estructura similar a las dos estructuras de entrada excepto que en ella se anotará que los objetos representados por los nodos etiquetados con C no son idénticos. Esta estructura se muestra en la figura 2.6.

La liga *nota-c* del nodo C describe la diferencia encontrada por la rutina de comparación. La liga indica que la diferencia ocurrió en la liga *es-un*, y que en la primera estructura la liga *es un* señala a un tabique, y en el segundo está señalando una cuña.

También hace notar que si se sigue la liga *es-un* de Tabique o de Cuña, las ligas se mezclarían. En este momento una nueva descripción del concepto de Arco puede ser dada. Esta descripción podría ser simplemente que el nodo C debe ser un Tabique o una Cuña. Dado que esta disyunción particular no tenía un significado previo, en la

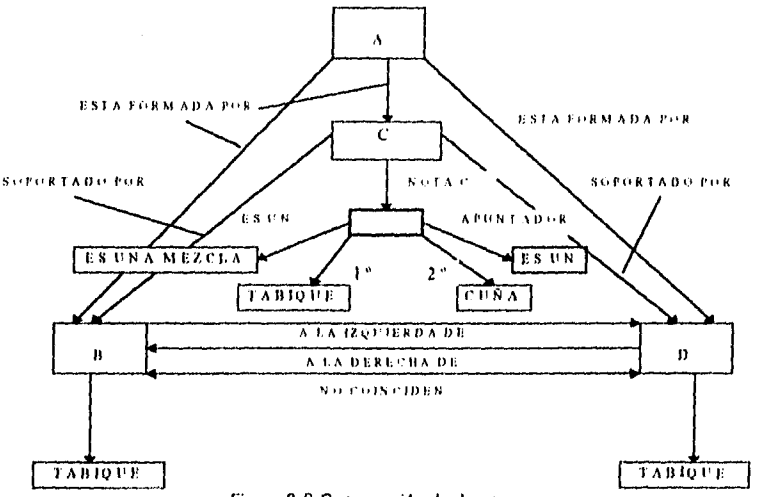


Figura 2.6:Comparación de dos arcos

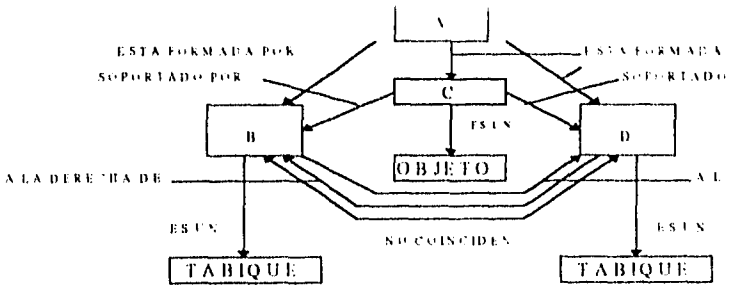


Figura 2.7: Descripción del arco después de dos ejemplos

estructura se crea un nodo Objeto, para la definición de Arco en el que se acepta un Arco o un Tabique quedando como se muestra en la de la figura 2.7

Si se presenta la otra figura similar al Arco de la figura 2.4 la rutina de comparación reparará en que la única diferencia entre la definición actual y la representación del nuevo objeto es la liga *no-coincide-con* entre los nodos B y D restringiendo la definición de tal manera que quede específicamente eliminado. En vez de la liga *no-coincide-con*, se utiliza la liga *no-debe-coincidir-con*. La nueva descripción de Arco queda como se muestra en la figura 2.8

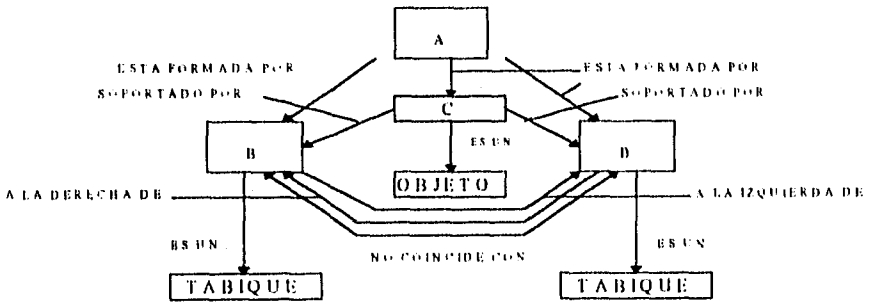


Figura 2.8 Descripción de arcos después de un ejemplo similar

Para describir estructuras de bloques se usaron redes semánticas y la jerarquía *es-un* se usó para describir relaciones entre objetos ya conocidos.

Este enfoque de aprendizaje de conceptos estructural tiene problemas. Uno de los principales es que un maestro debe guiar al programa de aprendizaje mediante una secuencia seleccionada de ejemplos.

2.4.2 ESPACIOS DE VERSIÓN

El espacio de versión es otro enfoque de aprendizaje por inducción de tipo estructural descrito por Mitchel en 1977. En este método la 'meta es la misma, producir una descripción que sea consistente con todos los ejemplos positivos pero no con los ejemplos negativos del conjunto de entrenamiento. '[RICH]. Mientras que en el sistema de Winston se hace esto desarrollando una sola descripción de concepto, en el método de espacio de versiones se trabaja manteniendo un conjunto de posibles descripciones y desarrollando ese conjunto conforme se presentan ejemplos nuevos y objetos parecidos. Se requiere entonces de un lenguaje para describir lo que el sistema 've" en cada ejemplo, por lo que se utiliza un lenguaje basado en *"marcos"* (frames) como el de la figura 2.9 en el que se presenta un marco para describir un automóvil particular.

Automóvil 023

Origen:	Japonés
Manufactura:	Honda
Color:	Azul
Modelo:	1970
Tipo:	Económico

Figura 2.9: Un ejemplo del concepto automóvil

Si cada elemento puede contener sólo los valores discretos mostrados en la figura 2.10, la selección de características y valores recibe el nombre de *inclinación* del sistema de aprendizaje. En el ejemplo de la figura 2.10 la inclinación está dada para aprender conceptos que tienen que ver con manufacturas de automóviles pero no con propietarios de automóviles.

Origen	$\in$	{Japón, Estados Unidos, Bretaña, Alemania, Italia}
Manufactura	$\in$	{ Honda, Toyota, Ford, Chrysler, Jaguar, BMW, Fiat}
Color	$\in$	{Azul, Verde, Rojo, Blanco }
Modelo	$\in$	{1950, 1960, 1970, 1980, 1990, 2000 }
Tipo	$\in$	{Económico, De lujo, Deportivo }

FIGURA 2.10: Representación en lenguaje para automóviles

En sistemas complejos la inclinación es menos visible. Una clara determinación de la inclinación de un sistema de aprendizaje es muy importante para su evaluación.

La descripción de conceptos, así como los ejemplos de entrenamiento, pueden ser establecidos en términos de huecos y valores. Por ejemplo, el concepto de "automóvil económico japonés" puede ser representado como en la figura 2.11.

Origen:	Japonés
Manufactura:	x_1
Color:	x_2
Modelo:	x_3
Tipo:	Económico

Figura 2.11: Concepto de "automóvil económico japonés"

Los nombres x_1, x_2 y x_3 son variables. La presencia de x_2, en el ejemplo, indica que el color del automóvil no es relevante si el automóvil es económico japonés.

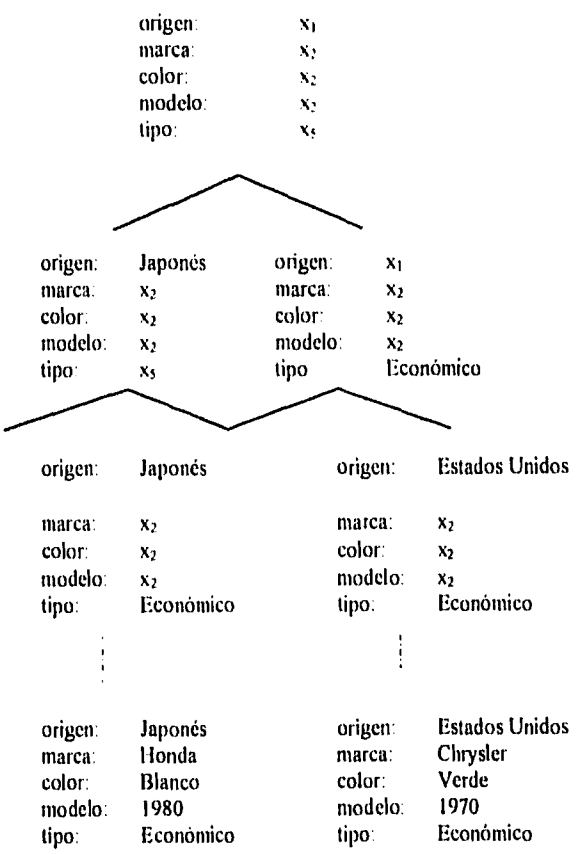


Figura 2.12: Ordenamiento Parcial de Conceptos Especificados para la Representación de Lenguaje

Algunas descripciones son más generales que otras, Por ejemplo, la descripción de la figura 2.11 es más general que la de la figura 2.9, de hecho el lenguaje de representación define un conjunto de descripciones con un orden parcial. En la figura 2.12 se muestra una porción de ese orden parcial.

El orden parcial es llamado *espacio de concepto*

El problema de aprendizaje para este ejemplo se puede plantear como:

> Dado un lenguaje de representación como el de la figura 2.10, y dados ejemplos de entrenamiento positivos y negativos como el de la figura 2.9, ¿cómo podemos producir una descripción de concepto como la de la figura 2.11 que sea consistente con todos los ejemplos de entrenamiento?.

El algoritmo de espacio de versiones parte de la descripción vacía, que consta sólo de variables, y de todos los posibles casos de entrenamiento, los cuales no contienen variables. Antes de recibir cualquier ejemplo de entrenamiento se sabe que los conceptos meta están de alguna manera en el espacio de concepto . Por ejemplo, si toda descripción posible es un caso del concepto, entonces la descripción vacía es la definición de concepto de *coincide-con-todo*. Por otra parte, si el concepto meta incluye sólo un único ejemplo, entonces una de las descripciones al final del espacio de concepto es la definición del concepto deseado.

El orden en que se procesan los ejemplos de entrenamiento es importante. La hipótesis vacía puede definirse por un subconjunto (conjunto formado exclusivamente por elementos del conjunto universal, en este caso el espacio de versión) del espacio de concepto llamado *espacio de versión*. El espacio de versión es la colección más grande de descripciones que es consistente con todos los ejemplos de entrenamiento que se presentarán más adelante.

¿Cómo se puede representar un espacio de versiones? El espacio de versiones es simplemente un conjunto de descripciones con un orden parcial, por lo que una idea inicial es mantener una lista de estas descripciones. Desafortunadamente, el número de descripciones del espacio de conceptos es exponencial con respecto al número de características y valores. Por lo que no es posible enumerarlos, sin embargo se concluye que el espacio de versiones tiene una representación breve, que consiste de dos subconjuntos del espacio de conceptos: el subconjunto, llamado G, contiene las descripciones más generales consistentes con los ejemplos de entrenamiento vistos anteriormente; otro subconjunto, llamado S, que contiene descripciones más específicas consistentes con los ejemplos de entrenamiento. El espacio de versiones es el conjunto de todas las descripciones que se encuentran entre los elementos de G y los elementos de S en el orden parcial del espacio de conceptos.

La representación del espacio de estados no sólo debe ser eficiente en el almacenamiento, sino también para su modificación. Intuitivamente cada vez que recibimos un ejemplo de entrenamiento positivo, queremos hacer al conjunto S más general. Los ejemplos de entrenamiento negativo sirven para hacer al conjunto G más específico. Si los conjuntos S y G convergen, nuestro rango de hipótesis se ajustará a una descripción de un concepto sencillo. El algoritmo para ajustar el espacio de versiones se llama *algoritmo de eliminación de candidatos.*

Algoritmo: Eliminación de candidatos

Dado: Un lenguaje de representación y un conjunto de ejemplos positivos y negativos expresados en ese lenguaje.

Determinar: Una descripción de concepto que sea consistente con todos los ejemplos positivos y ninguno de los ejemplos negativos.

1. Inicializa G con un elemento: la descripción vacía (todas las características son variables)

2. Inicializa S con un elemento: el primer ejemplo positivo.
3. Acepta un nuevo ejemplo de entrenamiento. Si es un ejemplo positivo, primero eliminar de G cualquier descripción que no cubra ese ejemplo. A continuación actualizar el conjunto S para que contenga el conjunto de descripciones más específico en el espacio de versiones que cubra el ejemplo y los elementos actuales del conjunto S.

   Es decir, generalice los elementos del conjunto S lo menos que se pueda de tal manera que cubra el nuevo ejemplo de entrenamiento.

   Si es un ejemplo negativo, primero quitar del conjunto S cualquier descripción que cubra el ejemplo. Después actualizar el conjunto G para que encierre el conjunto de descripciones más general en el espacio de versiones que *no* cubra el ejemplo. es decir, especializar los elementos de G lo menos posible de tal forma que los ejemplos negativos no sean cubiertos por cualquiera de los elementos de G.
4. Si ambos conjuntos S y G contienen un sólo elemento, analizar si son idénticos, en cuyo caso proporcionar el valor y detenerse. Si ambos son conjuntos de un sólo elemento pero son diferentes, entonces indicar que los casos de entrenamiento son inconsistentes y detenerse.

De otra manera, regrese al paso 3.

Continuamos con el algoritmo de eliminación de candidatos. Suponga que se quiere aprender el concepto de "automóvil económico japonés" del ejemplo de la figura 2.13 donde los conjuntos G y S empiezan con un sólo elemento. G contiene la descripción vacía, y S contiene el primer ejemplo de entrenamiento positivo.

El espacio de versiones ahora contiene todas las descripciones que son consistentes con este ejemplo:

$G = \{(x_1, x_2, x_3, x_4, x_5)\}$

S = {(Japón, Honda, Azul, 1980, Económico)}

origen: Japonés	origen: Japonés	origen: Japonés
marca: Honda	marca: Toyota	marca: Toyota
color: Blanco	color: Verde	color: Azul
modelo: 1980	modelo: 1970	modelo: 1990
tipo: Económico	tipo: Deportivo	tipo: Económico
(+)	(-)	(+)

origen: Estados Unidos	origen: Japonés
marca: Chrysler	marca: Honda
color: Rojo	color: Blanco
modelo: 1980	modelo: 1980
tipo: Económico	tipo: Deportivo
(-)	(+)

Figura 2.13: Ejemplos positivos y negativos de el concepto de "carro económico japonés"

En un segundo momento, el conjunto G debe ser especializado de tal manera que los ejemplos negativos no se encuentren más en el espacio de versiones es decir que no se encuentren sus descripciones . La especialización involucra el remplazar variables por constantes. (Nota: El conjunto G debe estar especializado sólo a descripciones que se encuentran dentro del espacio de versiones actual). Las especializaciones disponibles son:

G = {(x_1, Honda, x_2, x_3, x_4, x_5), (x_1, x_2, Azul, x_4, x_5), (x_1, x_2, x_3, 1980, x_5), (x_1, x_2, x_3, x_4, Económico)}. El conjunto S no se ve afectado por el ejemplo negativo.

A continuación se presenta un tercer ejemplo, ahora que sea positivo. Para este ejemplo lo primero que se hace es eliminar del conjunto G cualquier descripción que sea consistente con el ejemplo positivo. El conjunto G es entonces:

$$G = \{(x_1, x_2, \text{Azul}, x_4, x_5), (x_1, x_2, x_3, x_4, \text{Económico})\}$$

Generalizar al conjunto S para incluir el nuevo ejemplo. Esto significa remplazar constantes con variables para obtener el siguiente conjunto S:

$$S = \{(\text{Japón}, x_2, \text{Azul}, x_4, \text{Económico})\}$$

En este momento los conjuntos S y G especifican un espacio de versiones (un espacio de descripciones candidatas) que se puede traducir como: "El concepto meta puede ser tan específico como "Automóvil económico, azul japonés."

Ahora al tomar como ejemplo negativo, un automóvil cuyo origen es Estados Unidos. El conjunto S no es afectado, pero el conjunto G debe ser especializado para evitar que cubra el nuevo ejemplo. El conjunto G estará formado por:

$$G = \{(\text{Japón}, x_2, \text{Azul}, x_4, x_5), (\text{Japón}, x_2, x_3, x_4, \text{Económico})\}$$

Se concluye que el automóvil debe ser japonés, porque todas las descripciones en el espacio de versiones contienen a Japón como origen,. por último se presenta un ejemplo positivo. Primero se elimina del conjunto G cualquier descripción que no sea consistente con él, quedando G como:

$$G = \{(\text{Japón}, x_2, x_3, x_4, \text{Económico})\}$$

Generalizando el conjunto S para incluir el nuevo ejemplo:

$$S = \{(\text{Japon}, x_2, x_3, x_4, \text{Económico})\}$$

S y G tienen un sólo elemento por lo que el algoritmo converge a un concepto meta.

Hay que resaltar algunas cosas importantes acerca del algoritmo de eliminación de candidatos:

1. Es un algoritmo de mínimo-compromiso. El espacio de versiones es recortado lo menos posible en cada caso. Entonces, aún si todos los ejemplos de entrenamiento positivo son automóviles japoneses, el algoritmo rechazará la posibilidad de que el concepto meta pueda incluir automóviles de otros orígenes hasta que reciba un ejemplo negativo que lo obligue al rechazo, lo que significa que si los datos de entrenamiento son escasos o dispersos, los conjuntos S y G pueden no llegar a una convergencia esto es a una descripción sencilla; "el sistema puede aprender sólo conceptos especificados parcialmente"[RICH]
2. El algoritmo realiza búsqueda por amplitud, exhaustiva através del espacio de versiones. Esto lo podemos ver en el algoritmo para actualizar el conjunto G. Contrastando con el comportamiento por profundidad del programa de aprendizaje de Winston.
3. En el lenguaje de representación utilizado, el conjunto S siempre contiene un elemento, porque ninguna pareja de ejemplos positivos tiene una misma generalización exacta. Otros lenguajes de representación pueden no tener esta propiedad.

El enfoque de espacio de versiones se puede aplicar a un gran número de tareas de aprendizaje y lenguajes de representación y puede extenderse para manejar características con valores continuos y conocimiento jerárquico. Sin embargo, los espacios de versiones tienen algunas deficiencias: una de ellas consiste en que el espacio requerido para la búsqueda por amplitud es muy grande y la otra es que datos inconsistentes, también llamado ruido, puede provocar que el algoritmo de eliminación

de candidatos recorte el concepto meta del espacio de versiones anticipadamente. En el ejemplo del automóvil visto anteriormente, si el tercer caso de entrenamiento hubiera estado mal etiquetado

(-) en lugar de (+), el concepto meta de "automóvil económico japonés" nunca se alcanzaría. Además dando suficientes ejemplos negativos equivocados, el conjunto G puede ser tan especializado que el espacio de versiones se vuelve vacio, en tal caso el algoritmo termina en que *ningún* concepto coincide con los ejemplos de entrenamiento.

Mitchell en 1978 da una solución a este problema, manteniendo algunos conjuntos S y G. Un conjunto G que sea consistente con todos los casos de entrenamiento, otro consistente con todos menos con uno, otro con todos menos con dos, etc. (lo mismo se hace con el conjunto S). Cuando se tiene alguna inconsistencia, el algoritmo cambia a los conjuntos G y S que son consistentes con la mayoría, pero no con todos los ejemplos de entrenamiento. Conservar múltiples espacios de versiones puede ser costoso, ya que los conjuntos S y G son generalmente muy grandes. Si aceptamos inconsistencias en *frontera*, es decir que los casos cercanos a la frontera del concepto meta tiene más posibilidad de ser mal clasificados, entonces son posibles soluciones más eficientes.

Hirsh en 1990 presenta el siguiente algoritmo. Para cada caso, formamos un espacio de versiones consistente con ese caso más otros casos cercanos. Este espacio de versiones es intersectado con otro creado para todos los casos anteriores. Se sigue aceptando casos hasta que la versión de espacios se reduce a un pequeño conjunto de descripciones de conceptos candidatos. Debido a inconsistencias en los ejemplos de entrenamiento, sería raro que el espacio de versiones convergiera a un solo elemento. Entonces se compara cada una de las descripciones contra el conjunto de datos completo, y se elige el que clasifica los casos más adecuadamente.

Otro problema con el algoritmo de eliminación de candidatos es el aprendizaje de conceptos disyuntivos. Por ejemplo, si se supone que se quiere aprender el concepto de "automóvil europeo", el cual puede ser un automóvil alemán, británico o italiano. Dando ejemplos positivos de cada uno, el algoritmo de eliminación de candidatos generaliza a automóviles de cualquier origen. Dada esta generalización, un caso negativo (por ejemplo un automóvil japonés) sólo provoca una inconsistencia como la mencionada anteriormente.

Puede darse el caso de que el concepto meta sea "no Chrysler", pero se descarta esta posibilidad porque el lenguaje de representación no es suficientemente poderoso para expresar negación y disyunción. Se podría ampliar el lenguaje de representación para incluir disyunciones como " Automóvil azul de origen alemán o británico" y " deportivo italiano o automóvil de lujo alemán ".

Este enfoque presenta dos problemas:

1. El espacio de concepto es muy grande y la especialización se vuelve intratable.
2. La generalización puede fácilmente decaer al punto de que el conjunto S contenga simplemente una gran disyunción de todos los casos positivos.

De alguna forma se debe forzar la generalización mientras se permite la introducción de descripciones disyuntivas .

Mitchell en 1978 presenta un enfoque iterativo que involucra algunos pasos através de datos de entrenamiento. En cada paso, el algoritmo fabrica un concepto que cubre el máximo número de casos de entrenamiento positivo sin cubrir ninguno de los casos de entrenamiento negativo. Al final los casos de entrenamiento positivo cubiertos por el nuevo concepto se remueven del conjunto de entrenamiento, y entonces el nuevo concepto se vuelve una disyunción en la descripción de un concepto disyuntivo

eventual. Cuando todos los casos de entrenamiento positivo han sido eliminados, se dice que se ha llegado a un concepto disyuntivo que cubre todos ellos sin cubrir ningún caso negativo

Existen otras complicaciones, incluyendo la forma en la cual las características interactúan unas con otras, como en el siguiente ejemplo: Si el origen del automóvil es japonés, entonces la marca no puede ser Chrysler. El algoritmo de espacio de versiones descrito anteriormente no hace uso de esta información. También en este ejemplo resulta más natural sustituir el hueco de década con un campo de valor continuo como año . Se tienen que cambiar los procedimientos para actualizar los conjuntos S y G para tomar en cuenta este tipo de datos numéricos.

2.4.3 ARBOLES DE DECISIÓN

Un tercer ejemplo de aprendizaje por inducción con enfoque estructural es la aplicación de árboles de decisión, en donde un árbol de decisión es una distribución de pruebas que guían cada paso de un análisis, para determinar la clase de un objeto. Entonces

Un árbol de decisión es una representación formada por nodos en donde:

1. Cada nodo corresponde a una pregunta y está conectado con un conjunto de respuestas posibles.
2. Cada nodo que no sea hoja está conectado con una prueba que divide su conjunto de respuestas posibles en subconjuntos correspondientes a diferentes resultados de pruebas.
3. Cada rama lleva un subconjunto particular de resultados de prueba a otro nodo.

Quinlan en 1986 presentó un programa al que llamó ID3, en el cual usa una representación de árbol para los conceptos origen y tipo, mostrados en la figura 2.14. Para clasificar una entrada particular, se empieza en lo alto del árbol y contestando las preguntas se encuentra una hoja, donde la clasificación se almacena. Un signo positivo en un nodo terminal indica que es un elemento del concepto que se quiere representar.

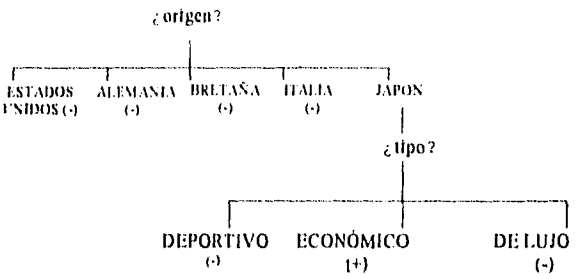


Figura 2.14: ÁRBOL DE DECISIÓN

La figura anterior corresponde al concepto de "coche económico japonés". ID3 construye árboles de decisión automáticamente, a partir de casos positivos y negativos del concepto.

En ID3 construir un nodo significa seleccionar algunos atributos a probar. En un punto dado en el árbol algunos atributos proporcionan más información que otros, por ejemplo, considerar el atributo *color* no aporta información si el color del coche no ayuda a clasificarlo correctamente. Idealmente, un atributo separará los casos de entrenamiento en subconjuntos cuyos elementos comparten un atributo común (de ejemplo positivo o negativo). En ese caso, la ramificación se termina, y los nodos hoja son clasificados.

ID3 usa un método iterativo para construir árboles de decisión. Empieza seleccionando un subconjunto aleatorio de ejemplos de entrenamiento. Este subconjunto es llamado *ventana*.

Algoritmo para construir un árbol de decisión que clasifique correctamente.

Dado : Un subconjunto de ejemplos de entrenamiento

Calcular: Un árbol de decisión que clasifique correctamente todos los ejemplos de entrenamiento de la ventana.

1. Selecciona un subconjunto aleatorio de ejemplos de entrenamiento llamado ventana.
2. Genera un árbol determinando los posibles atributos y ramificaciones.
3. Si todos los ejemplos de afuera de la ventana son clasificados correctamente, terminar.
4. De otra manera, añadir un número de ejemplos de entrenamiento a la ventana y repetir el proceso.

La estrategia iterativa es más eficiente que considerar el conjunto de entrenamiento completo de una vez.

Existen muchas variaciones de este algoritmo básico. Por ejemplo, cuando se agrega una prueba que divide el conjunto en más de dos subconjuntos, es posible que no existan ejemplos de entrenamiento para alguno de ellos. En ese caso, se puede dejar el nodo sin clasificar o tratar de adivinar un atributo en base a propiedades estadísticas del conjunto de casos que han sido probados hasta ese momento en el árbol. Las entradas ruidosas son otro problema. Una forma de trabajar entradas ruidosas es el evitar construir nuevas ramas si la información ganada es muy débil. Es decir, no se debe complicar el árbol para tomar en cuenta casos ruidosos aislados.

Otra fuente de incertidumbre se presenta cuando los valores de los atributos pueden ser desconocidos, por ejemplo, un registro médico incompleto de un paciente.

Una solución cuando los atributos no se conocen es proporcionar valores preestablecidos para que seleccione alguna de las ramas o construir una rama especial con la etiqueta "desconocido" en cada nodo durante el aprendizaje.

Cuando el espacio de versiones es muy grande, los algoritmos de aprendizaje por árbol de decisión son más rápidos que los de espacios de versiones. Además, la implementación de disyunciones es más directa. Por ejemplo, podemos fácilmente modificar la figura 2.14 para representar el concepto disyuntivo "coche americano o coche económico japonés" cambiando únicamente una de las hojas etiquetadas como negativa (-) a positiva (+). Un problema del enfoque de ID3 es que un árbol de decisión complejo y grande puede ser difícil entender por el ser humano, y entonces en el sistema de árbol de decisión es difícil de explicar las razones para su clasificación. Como se puede ver, se pueden inducir descripciones de conceptos de ejemplos positivos y negativos.

"Actualmente, la representación con árbol de decisión es más general: las hojas pueden denotar cualquiera de los números de clases, no sólo positivos y negativos." [RICH].

2.5 APRENDIZAJE BASADO EN EXPLICACIONES

Mucho del trabajo reciente de aprendizaje automático se ha alejado del enfoque del análisis de grandes cantidades de datos hacia un enfoque más analítico basado en el conocimiento formal y empírico del dominio en que se esté trabajando. Algunos estudios independientes llevaron a la caracterización de este enfoque como *aprendizaje basado en explicaciones.*

Así como para estudiar algunos conceptos complejos se requiere de un gran número de casos de entrenamiento, existen otros en los que es mejor proporcionar un solo ejemplo, ya que parece ser que las personas son capaces de aprender mejor de

ejemplos únicos como en el caso de un jugador de ajedrez que, como negro, ha alcanzado la posición llamada "tenedor" porque el caballero blanco ataca tanto al rey negro como a la reina negra. El negro debe mover el rey , aunque deje a la reina en peligro. De esta experiencia única, el negro es capaz de aprender más acerca de la trampa del tenedor: la idea es que si cualquier pieza *x* ataca al rey del oponente y a otra pieza *y*, entonces la pieza *y* se da por perdida. No es necesario ver varios ejemplos positivos y negativos de posiciones de tenedor para llegar a estas conclusiones. De una sola experiencia, se puede aprender a prevenir este tipo de engaño y en el futuro utilizarlo para lograr alguna ventaja.

¿Qué hace posible que ese único ejemplo lleve al aprendizaje? La respuesta es el conocimiento. El jugador de ajedrez tiene mucho conocimiento sobre un dominio específico incluyendo reglas de ajedrez y estrategias previamente adquiridas. Ese conocimiento puede ser usado para identificar los aspectos críticos de los ejemplos de entrenamiento: por ejemplo, en el caso del tenedor sabemos que un ataque doble simultáneo es importante mientras la posición precisa y el tipo de las piezas de ataque no.

Un sistema EBL (Explanation Based Learning) trata de aprender de un ejemplo único *x*, explicando por qué *x* es un ejemplo del concepto meta, la explicación se generaliza, y el comportamiento del sistema depende de la disponibilidad de este conocimiento.

El programa EBL de *generalización basada en explicaciones* fué descrito por Mitchell et al. en 1986 y consta de los siguientes dos pasos:

1. Explica
2. Generaliza

En el primer paso la teoría del dominio es usada para eliminar todos los aspectos no importantes del ejemplo de entrenamiento con respecto al concepto meta. Lo que queda es una *explicación* de por qué el ejemplo de entrenamiento es un caso del

concepto meta. Esta explicación está dada en términos que satisfacen el criterio de operacionalidad.

En el paso dos se trata de generalizar la explicación tanto como sea posible mientras continúe describiendo el concepto meta. En el ejemplo del ajedrez, el primer paso de EBL decide ignorar el peón blanco, el rey y la torre, y construir una explicación que involucre al caballero blanco, el rey negro y la reina negra, cada uno en sus posiciones específicas. La operacionalidad está asegurada: todos los programas que juegan ajedrez entienden los conceptos básicos de pieza y posición. Están listos para que después la explicación se generalice.

DeJong et al en 1986 describen esqueletos generales para programas EBL y dan algoritmos de aprendizaje generales.

La información que puede recibir un programa EBL es:

1. Un ejemplo de entrenamiento, del tipo de la figura 2.9.
2. Conceptos objetivos - Descripciones de alto nivel de lo que el programa se supone debe aprender.
3. Criterios de operacionalidad - Descripciones de qué conceptos son utilizables.
4. Una Teoría del dominio - Un conjunto de reglas que describen relaciones entre objetos y acciones en un dominio.

"EBL calcula una *generalización* del ejemplo de entrenamiento que es suficiente para describir el concepto meta, y también satisface los criterios de operacionabilidad"[RICH].

Un programa EBL busca operacionalizar el concepto meta. En el ejemplo del ajedrez, el concepto meta puede ser la "mala posición para el Negro", y el concepto operacionalizado sería una descripción generalizada de una situación parecida al ejemplo de entrenamiento dado en términos de piezas y sus posiciones relativas. En el

ejemplo del ajedrez la teoria de dominio que hay que proporcionar a un programa EBL son las reglas del juego sin las cuales es imposible llegar a una correcta generalización del ejemplo de entrenamiento. Usando el conocimiento del dominio encontramos que mover las piezas a un lugar diferente del tablero es malo para los Negros. También se puede determinar que otras piezas, además de los caballeros y reinas, pueden participar en un ataque de tenedor.

Los métodos actuales de EBL tienen dificultades en dominios tan complejos como el ajedrez, por lo que ahora veremos un caso en un dominio más simple. Considere el problema de aprendizaje del concepto "Taza" presentado por Mitchell et al., en 1986. A diferencia del programa de aprendizaje de arco visto anteriormente, se pretende llegar a una generalización con un sólo ejemplo de una taza.

Suponga que el ejemplo de entrenamiento es el siguiente:

propietario(objeto23,Ralph)∧tiene-parte(objeto23, concavidad12)∧

es(objeto23,ligero)∧color(objeto23,café)∧. . .

Algunas de las características de objeto23 son más importantes que otras para considerarlo como una taza. Algunos métodos que permiten separar características relevantes requieren de muchos ejemplos positivos y negativos, en cambio, EBL se basa en el conocimiento del dominio, por ejemplo:

- Conocimiento del dominio:

  es(x,ligero)∧tiene-parte(x,y)∧es un(y,manejable)----> levantable(x)

  tiene parte(x,y)∧es un(y,abajo)∧es(y,plano)-----> estable(x)

  tiene parte(x,y)∧es un(y,concavidad)∧es(y,apuntandohaciaarriba)

  ------> recipiente abiero(x)

Es necesario el concepto meta para operacionalizar:

- Concepto meta: taza

  x es una taza si y sólo si es levantable, estable y recipiente abierto

- Criterio de operacionalidad: La definición del concepto debe estar dado

en términos puramente estructurales (ej ligero, plano, etc).

Para construir una descripción estructural general de una taza a partir de un ejemplo de entrenamiento y de una descripción funcional. El primer paso consiste en explicar por qué objeto23 es una taza. Esto se hace construyendo una prueba como se muestra en la figura 2.15.

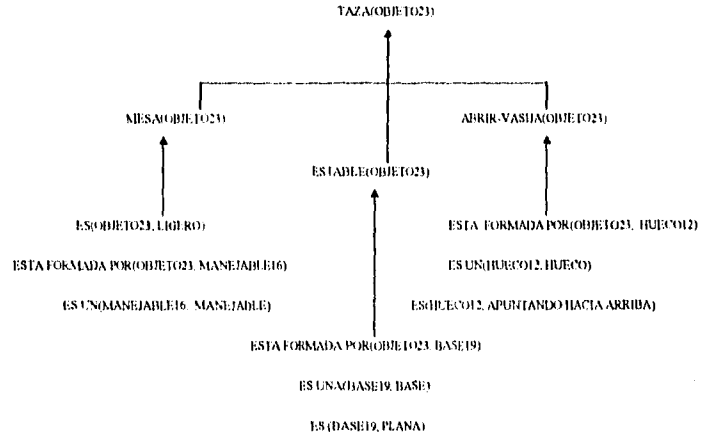


FIGURA 2.15: Aprendizaje basado en explicaciones

"Las técnicas estándar de demostración de teoremas pueden ser usadas para encontrar tal prueba. Note que la prueba aisla las características relevantes del ejemplo de entrenamiento; en ninguna parte en la prueba los predicados *propietario* y *color* aparecen. La prueba también sirve como una base para una generalización válida."[RICH].

Si se juntan todas las suposiciones y se reemplazan constantes con variables se obtiene la siguiente descripción de una taza:

tieneparte(x,y)∧esun(y,concavidad)∧es(y,apuntandohacinarriba)∧

tiene-parte(x,z)∧esun(z,abajo)∧es(z,plano)∧

tiene-parte(x,z)∧esun(z,abajo)∧es(z,plano)∧

tiene-parte(x,w)∧esun(w,manejable)∧es(x,ligero)

Esta definición cumple con el criterio de operacionalidad y puede ser usada por un robot para clasificar objetos.

La generalización se logra simplemente remplazando las constantes por las variables empleadas en este ejemplo, aunque en algunos casos es necesario retener ciertas constantes. Para detectar estos casos, debemos probar de nuevo la meta. Este proceso se vió en el aprendizaje de STRIPS y es llamado *regresión de la meta.*

Como ventos, EBL depende fuertemente de la teoría del dominio. Dada esta teoría, ¿por qué son necesarios los ejemplos? Podríamos operacionalizar el concepto meta *taza* sin hacer referencia a un ejemplo, si la teoría del dominio contiene todos los requisitos de información. La respuesta es que los ejemplos ayudan a enfocar el aprendizaje en operacionalizaciones relevantes. Si el ejemplo es una *taza*, EBL se enfrenta con la tarea de caracterizar el rango entero de los objetos que satisfacen el concepto meta. "La mayoría de estos objetos nunca se encontrarán en el mundo real, y entonces el resultado será sobregeneralizado."[RICH]

Proporcionar una teoría del dominio completa es una tarea difícil. Existe la certeza de que los humanos no aprendemos con relaciones muy primitivas, sino que los humanos creamos teorías de dominio incompletas e inclusive inconsistentes. Por ejemplo, regresando al ajedrez, tal teoría puede incluir conceptos como "estructura de peón débil". Hacer que EBL trabaje en "teorías de dominios mal estructurados es una área activa de investigación."[RICH].

EBL comparte varias características de los métodos descritos en las secciones anteriores. EBL empieza con un ejemplo positivo de algún concepto, como en aprendizaje por consejo, y la meta es operacionalizar alguna parte del conocimiento.

Las técnicas EBL son habitualmente usadas para mejorar el desempeño de los sistemas en la solución de problemas. La gran diferencia entre EBL y otros métodos de aprendizaje es que los programas de EBL están construidos para tomar ventaja del conocimiento del dominio. Lo que significa que aprender es justamente otro tipo de solución de problemas.

2.6 APRENDIZAJE POR DESCUBRIMIENTO

El aprendizaje es el proceso por el cual una entidad adquiere conocimiento. Frecuentemente ese conocimiento ya era poseído por algún número de otras entidades que podían servir como maestros. El descubrimiento es una forma limitada de aprendizaje en la cual una entidad adquiere conocimiento sin la ayuda de un maestro "Algunas veces, no hay nadie en el mundo que tenga el conocimiento que buscamos. En ese caso, el tipo de acción que debemos tomar es llamado descubrimiento científico'[RICH]. Con este enfoque se estudiarán tres métodos de descubrimiento automático:

1. Descubrimiento dirigido por la teoría.
2. Descubrimiento dirigido por datos.
3. Descubrimiento por agrupamiento.

2.6.1 AM: DESCUBRIMIENTO DIRIGIDO POR LA TEORÍA

'El descubrimiento es ciertamente aprendizaje. Pero también es, quizás más claramente que otros tipos de aprendizaje, un proceso de solución de problemas."[RICH]

El sistema AM desarrollado por Lenat en 1977 es un ejemplo de descubrimiento dirigido por la teoría. A partir de algunos conceptos básicos de la teoría de conjuntos "descubre" una buena cantidad de conceptos de la teoría de números.

AM utiliza una gran variedad de técnicas de IA de propósito general. Usa un sistema de marcos para representar los conceptos matemáticos

AM utiliza búsquedas heurísticas, guiado por un conjunto de 250 reglas heurísticas representando consejos acerca de operaciones que permiten llegar a "descubrimientos interesantes". Ejemplos de este tipo de heurísticas se muestran en la figura 2.16.

AM utiliza el método de genera y prueba para formar hipótesis en base a un pequeño número de ejemplos y ver si éstas se mantienen con un número mayor de ejemplos. Finalmente una agenda controla el proceso completo de descubrimiento. Cuando las heurísticas sugieren una tarea, se coloca ésta en la agenda central, junto con la razón y la prioridad con la que fue sugerida. AM opera en ciclos, cada vez seleccionando la tarea más prometedora de la agenda y llevándola a cabo.

En una corrida, AM descubrió el concepto de números primos. ¿Cómo lo hizo? habiendo encontrado los números naturales, AM examinó operaciones tales como la suma, multiplicación, la resta y la división. Creó el concepto de divisibilidad y notó que algunos números tienen muy pocos divisores. AM tiene una heurística de construcción que le dice que examine casos extremos. Se trató de listar todos los números con divisor cero (sin encontrar alguno), un divisor (encontrando uno: 1), y dos divisores. AM fue instruido para llamar al último concepto "primos". Antes de continuar con este concepto, AM listó los números con tres divisores, tales como el 49. AM trató de relacionar esta propiedad con otras propiedades del 49, como que es impar y un cuadrado perfecto. AM generó otros números impares y otros cuadrados perfectos para demostrar su hipótesis.

nombre: números primos

definiciones:

origen: Número-de-divisores-de(x) 2

cálculo-de-predicados: Primo(x) <==> (∀z) (z|x=>(z=1 ⊙z=x))

iterativo: (para x>1) : para y desde 2 hasta $\sqrt{x}$, i ≠x

ejemplos: 2, 3, 5, 7, 11, 13,17

extremos: 2, 3

fracasos-frontera: 0, 1

fracasos: 12

generalizaciones: número, número con número par de divisores

especializaciones: primos impares, primos pares, primos únicamente sumables

conjeturas: factorización única, conjetura de Goldbach, extremos del número-de-divisores-de

intuiciones: una metáfora del efecto que las primos son los bloques constructores de todos los números

analogías:

Números máximamente divisibles son convertidas en extremos de números-de-divisores-de

Factorizar un grupo no simple en grupos simples

interés: conjeturas uniendo primos a veces, a divisores de, a operaciones relacionadas

valor: 800

Figura 2.16

"Un efecto colateral de determinar la equivalencia de los cuadrados perfectos con números con tres divisores era para aumentar la razón de falta de interés del concepto

de división. lo cual llevó a AM a buscar formas en las cuales un número puede ser descompuesto en factores. AM entonces notó que había sólo una forma de descomponer un número en factores primos (conocido como el teorema de Factorización Única)"[RICH].

Cuando descomponiendo los números en componentes multiplicativos le pareció interesante de nuevo, AM decidió por analogía, continuar con los componentes aditivos. Llegó a algunas conjeturas poco interesantes, como que cada número puede ser expresado como una suma de 1's. También encontró fenómenos más interesantes, como que muchos números pueden ser expresados como la suma de dos primos. Listando los casos, AM determinó que todos los números pares mayores que 2 parece que tienen esta propiedad. Lo anterior se conoce como Conjetura de Goldbach, se cree que es verdadera pero todavía no se ha encontrado una demostración formal en matemáticas.

AM contiene una gran cantidad de heurísticas de propósito general tales como algunas usadas en este ejemplo. Frecuentemente diferentes heurísticas señalan a un mismo lugar. Digamos que mientras AM descubría los números primos usando una heurística que comprende analizar casos extremos, otra forma de derivar los números primos es usar las siguientes dos reglas:

- Si hay una gran analogía entre A y B pero existe una conjetura acerca de A que no se cumpla para todos los elementos de B, definir un nuevo concepto que incluya los elementos de B para los cuales no se sostienen.
- Si hay un conjunto cuyo complemento es más raro que él mismo, entonces crear un nuevo concepto que represente el complemento.

Existe una fuerte analogía entre la suma y la multiplicación de los números naturales, pero esa analogía se suspende cuando se observa que todos los números naturales mayores que 1 pueden ser expresados como la suma de dos números

naturales más pequeños (excluyendo la identidad), esto no se cumple para la multiplicación. Entonces la primera heuristica mencionada anteriormente sugiere la creación de un nuevo concepto que represente el conjunto de números compuestos. La segunda heuristica sugiere la creación de un concepto que represente el complemento de éste, llamado el conjunto de números primos.

Dos preguntas fundamentales surgen del trabajo de AM. ¿Por qué AM simplemente se mantiene descubriendo nuevos hechos interesantes acerca de los números en vez de posibles hechos desconocidos para los matemáticos?. Lenat en 1983 señala que el desempeño de AM esta limitado por la naturaleza estática de sus heurísticas. Dependiendo del progreso del programa, los conceptos en los cuales está trabajando evolucionan alejándose de los conceptos iniciales, mientras que las heurísticas que están disponibles para trabajar estos conceptos siguen siendo las mismas. Para evitar este problema, se sugirió que las heuristicas fueran tratadas como los conceptos que pueden ser creados y modificados por el mismo tipo de procesos (como generalización, especialización y analogia). Es decir, AM , para el estudio de las heurísticas mismas, aplica el método de descubrimiento en la misma forma en la que trabaja en el dominio de la teoria de números.

Una extensión de AM llamada EURISKO fué diseñada por Lenat en 1983 con esta idea.

La segunda pregunta es: "¿Por qué AM trabaja tan bien?". Una fuente de poder de AM es su enorme colección de heurísticas acerca de lo que se considera 'interesante". Aunque AM posee otra fuente de poder menos obvia, llamada, relación natural entre conceptos teóricos de número y su representación compacta en AM, "AM trabajó mutando sintácticamente viejas definiciones de conceptos guardadas esencialmente como pequeños programas LISP'[RICH] con la esperanza de encontrar nuevos conceptos interesantes, observándose que una mutación en un programa

pequeño en LISP da como resultado un programa en LISP bien-formado, significativo y muy parecido.

Mientras los investigadores estudiaban a AM explorando teoría de números, AM explora el espacio de pequeños programas en LISP, dando como resultado que AM alcanzará un gran éxito debido en gran parte a la relación entre teoría de números y programas LISP. Cuando AM y EURISKO se aplicaron en otros dominios, surgieron problemas, pues los conceptos en estos dominios eran más grandes y complejos que los de la teoría de números, y la sintaxis del lenguaje de representación no tan cercanamente reflejado en la semántica del dominio. Como resultado de ello, las mutaciones sintácticas de la definición de un concepto casi siempre daban como resultado un concepto mal formado o inservible, dificultando grandemente el proceso de descubrimiento.

'Quizás la moraleja de AM es que el aprendizaje es un negocio truculento."[RICH].

Ritchie y Hanna en 1984 sugieren la necesidad de ser cuidadosos en la interpretación de lo que están haciendo nuestros programas de IA. AM tiene una predisposición implícita hacia conceptos de aprendizaje en teoría de números, a partir de que esa predisposición fue explícitamente reconocida fue posible entender por qué el desempeño de AM era bueno en un dominio y pobre en otro.

AM probó cómo el descubrimiento puede ocurrir en un ambiente teórico. Científicos empíricos veían las cosas en una forma diferente: Tomaban datos del mundo y debían extraer algún sentido de ello. Hacían sus hipótesis, y para demostrarlas, ellos diseñaban y ejecutaban experimentos.

2.6.2 BACON: DESCUBRIMIENTO DIRIGIDO POR DATOS

El descubrimiento científico ha inspirado un gran número de modelos de computadora. Langley et al en 1981 presentó un modelo de descubrimiento científico dirigido por datos que se implementó en un programa llamado BACON, en honor a Sir Francis Bacon, uno de los primeros filósofos de la ciencia.

BACON se utilizó en el estudio de comportamiento de gases. Algunas de las variables con las que trabajó son p, la presión del gas, V, el volumen del gas, n, la cantidad de gas en mols, y T, la temperatura del gas. Y fué capaz de derivar la *ley del gas ideal*, que relaciona estas variables.

BACON mantiene primero las variables n y T constantes, llevando a cabo experimentos con diferentes presiones p_1, p_2 y p_3. BACON, al notar que el volumen V disminuye si la presión se incrementa, crea un término teórico pV, y lo considera constante. BACON sistemáticamente modifica las otras variables. Trata un experimento con diferentes valores de T, y encuentra que pV cambia. Los dos términos están linealmente relacionados y se intersectan en 0, entonces BACON crea un nuevo término pv/T. Finalmente, BACON cambia el valor de n y formula otra relación lineal entre n y pV/T. Para todos los valores de n, p, V y T, pV/nT=8.32. Esto corresponde a la ley del gas ideal. La figura 2.17 que a continuación se presenta muestra el razonamiento de BACON en forma de una tabla

n	T	p	V	pV	pV/T	pV/nT
1	300	100	24.96			
1	300	200	12.48			
1	300	300	8.32	2496		
1	310			2579.2		
1	320			2662.4	8.32	
2	320				16.64	
3	320				24.96	8.32

Figura 2.17: Descubrimiento de la Ley Ideal del Gas de BACON

BACON ha sido capaz de redescubrir una amplia variedad de leyes científicas, como por ejemplo la tercera ley de Kepler, la ley de Ohm, la conservación del momento, la ley de Joule. 'Las heurísticas que BACON usa para descubrir la ley del gas ideal incluyen reconocer constantes, encontrar relaciones lineales, y definir términos teóricos. Otras heurísticas permiten a BACON postular propiedades intrínsecas de los objetos y razonar por analogía.'[RICH] por ejemplo, si BACON encuentra una regularidad en un conjunto de parámetros, tratará de generar la misma regularidad en un conjunto similar de parámetros.

El procedimiento de BACON puede considerarse como una búsqueda en un espacio de estados, y las heurísticas le permiten llegar a soluciones mientras analiza solamente una porción del espacio de búsqueda. En el ejemplo del gas, BACON llega a la ley del gas ideal usando el mínimo número de experimentos.

Mucho trabajo debe hacerse en áreas de la ciencia que BACON no modela, como por ejemplo determinar que dato agrupar, seleccionando (o creando) instrumentos para medir los datos y usando analogías para previamente entender fenómenos.

2.6.3 DESCUBRIMIENTO POR AGRUPAMIENTO

Este tipo de aprendizaje por descubrimiento es muy parecido al de inducción. En el aprendizaje inductivo un programa aprende a clasificar objetos basándose en los atributos dados por un profesor, mientras que en agrupamiento, no se proporcionan etiquetas de clase, sino que el programa debe descubrir por sí mismo las clases naturales que existen para los objetos.

AUTOCLASS es un programa presentado por Cheeseman et al. en 1988 que acepta un conjunto de casos de entrenamiento y propone como hipótesis un conjunto de probabilidades de predecir en cuál(es) clase(s) el caso posiblemente caerá. En una

aplicación. AUTOCLASS encontró nuevas clases significativas de estrellas a partir de sus datos espectrales infrarrojos, lo cual se considera como un caso de verdadero descubrimiento por computadora, ya que los hechos que descubrió eran previamente desconocidos en la astronomía. AUTOCLASS usa razonamiento estadístico Bayesiano.

CAPÍTULO 3

MODELOS CONEXIONISTAS

A diferencia de la IA tradicional, en los modelos conexionistas se ha intentado construir máquinas inteligentes partiendo del modelo natural: el cerebro humano. En las dos ramas de la IA (la simbólica y la conexionista) una idea fundamental ha sido la de simular el funcionamiento del cerebro humano directamente en una computadora.

La idea de construir una computadora inteligente con neuronas artificiales ha sido trabajada desde hace algún tiempo y algunos de los primeros resultados en mecanismos parecidos al cerebro humano fueron logrados por McCulloch en 1954, Minsky y Selfridge en 1961, Block y Rosenblatt en 1962. Pero fue hasta los años 70's en que los estudios sobre estos mecanismos tuvieron un gran auge con la investigación de redes neuronales, aunque las redes estudiadas en ese entonces, mostraron ser muy débiles dadas las características de las computadoras de esa época.

Recientemente el interés por las redes neuronales ha resurgido debido a:

- La aparición de computadoras digitales más rápidas, en las cuales se pueden simular grandes redes.
- El interés en construir computadoras en forma masiva y, la más importante,
- El descubrimiento de arquitecturas de redes neuronales y poderosos algoritmos de aprendizaje.

"Las nuevas arquitecturas de redes neuronales han sido llamadas arquitecturas *conexionistas*"[RICH]. En general, estas arquitecturas se basan en la forma en que se supone trabaja el cerebro humano.

Estas arquitecturas están caracterizadas por tener:

- Un gran número de elementos simples de procesamiento parecidos a una neurona.

- Un gran número de conexiones entre elementos, que junto con los pesos en las conexiones tienen codificado el conocimiento de la red.
- Control distribuido y paralelo.
- Énfasis en el aprendizaje de representaciones internas de redes neuronales simuladas automáticamente

"Los investigadores conexionistas conjeturan que pensar acerca de computación en términos de *metáforas del cerebro* más que en *metáforas de las computadoras digitales* llevará a descubrimientos en la naturaleza del comportamiento inteligente" [RICH].

Actualmente existen programas expertos jugadores de ajedrez y programas expertos en diagnósticos médicos, "pero no existen programas con las habilidades de percepción básica de un niño. Las investigadores en redes neuronales afirman que existe una gran diferencia entre la tecnología de procesamiento de información de las computadoras estándar y la tecnología usada por el cerebro."[RICH].

El cerebro humano no trabaja como una computadora sino que trabaja en forma masivamente paralela. El funcionamiento paralelo de cientos de miles o millones de neuronas proporciona velocidad en los procesos de extracción de información.

'Dentro del cerebro humano, el conocimiento no se almacena en ninguna forma que se parezca a un programa de computadora convencional, sino que se almacena estructuralmente, en forma de patrones distribuidos de pesos sinápticos excitatorios e inhibitorios, cuyas magnitudes relativas determinan el flujo de las respuestas neuronales que constituyen la percepción y el pensamiento."[STEPHEN].

En este capítulo se presentan conceptos fundamentales de métodos conexionistas en los cuales puede observarse la influencia de las ideas antes mencionadas.

3.1 CONCEPTOS GENERALES SOBRE REDES NEURONALES Y ESQUEMA GENERAL DEL APRENDIZAJE CONEXIONISTA

Como el esquema de aprendizaje utiliza modelos que simulan redes neuronales, a continuación se presenta una breve descripción de una neurona real y la forma en que se representa por medio de una neurona simulada.

Las neuronas reales son células nerviosas con prolongaciones capaces de recibir y transmitir señales o impulsos eléctricos. "Las redes neuronales reales son un conjunto de neuronas interconectadas."[GARCIA]

Las neuronas reales están formadas por:

- La célula nerviosa y
- Sus prolongaciones, que pueden ser de dos tipos:

  - Las dendritas, prolongaciones generalmente cortas, ramificadas y de base ancha que reciben el impulso nervioso y lo conducen hacia el cuerpo celular.
  - Axón prolongación que se extiende desde el cuerpo de la neurona y que conduce los impulsos nerviosos hacia las dendritas de otras neuronas.

Las neuronas reales generalmente tienen un solo axon (figura 3.1)

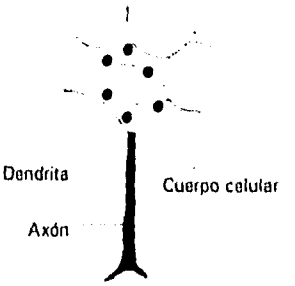


Figura 3.1: Esquema de una neurona real.

La transmisión del impulso nervioso entre las ramificaciones del axón y las dendritas de otra célula se hace por contacto de unas y otras. A esos contactos o uniones entre una neurona y la siguiente se le llama *sinapsis* (figura 3.2).

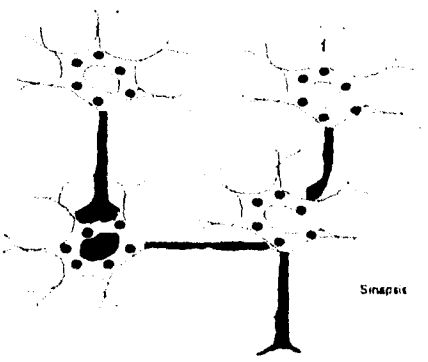


Figura 3.2: Esquema de una sinapsis.

La célula nerviosa recibe impulsos nerviosos por sus dendritas y a su vez transmite impulsos nerviosos por su axón.

La transmisión de un impulso nervioso se efectúa sólo si la suma de los impulsos nerviosos recibidos es superior a un umbral, cuando esto sucede, se dice que la neurona real se dispara. Debido a que una neurona real se dispara o no hace nada, se le considera como un dispositivo de todo o nada

"Existe una certeza cada vez mayor de que el aprendizaje se lleva a cabo en las cercanías de las sinapsis y que tiene que ver con el grado en que éstas traducen el impulso que viaja por el axón de una neurona en la excitación o inhibición de la siguiente neurona."[WINSTON].

Una vez vistos los elementos de una neurona real se muestra a continuación el modelo con el que se representa, esto es los elementos de una neurona simulada.

Una neurona simulada consta de multiplicadores, sumadores y umbrales. Se representa como un nodo al que llegan una serie de enlaces con un peso cada uno, y salen uno o varios enlaces (figura 3.3).

El peso que se asocia a cada enlace determina la intensidad de la acción que ejerce un nodo sobre otro, como en el caso de la sinapsis.

Los sumadores modelan la capacidad de las dendritas de combinar influencias; y la comparación con el umbral o nivel asignado a la neurona simulada determina si ésta se dispara o no, lo que modela la característica de todo o nada de una neurona simulada.

En una neurona simulada los valores de las conexiones (entradas) que llegan de otras neuronas simuladas se multiplican por determinados pesos y después se suman, la suma se compara con un nivel de umbral:

Si la suma está por encima del umbral, la salida es 1, de otro modo la salida es 0.

Las neuronas simuladas pueden organizarse en redes neuronales simuladas.

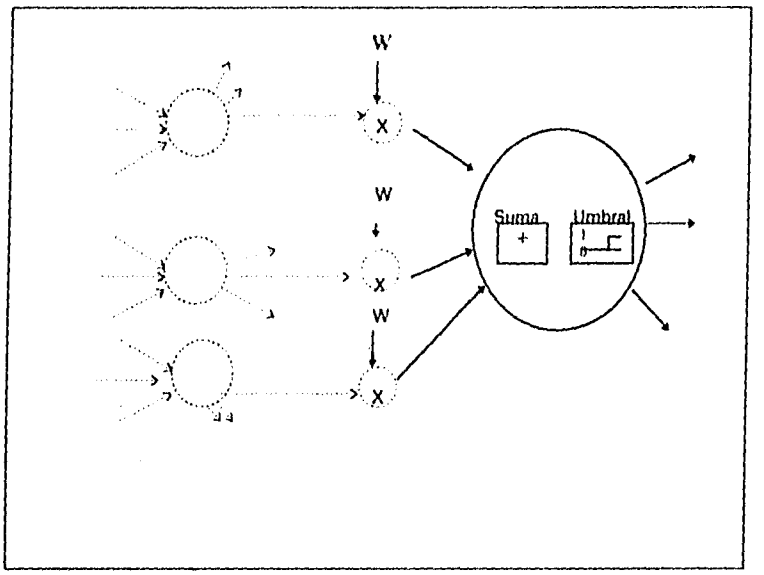


Figura 3.3: Esquema de una neurona simulada

Una red neuronal simulada en general es un diagrama compuesto por nodos que se representan como círculos, enlaces que corresponden a relaciones entre los nodos y que se representan como flechas que apuntan de un nodo inicial a un nodo final, y valores de las conexiones de enlaces que indican el tipo de relación (cuantitativa o cualitativa) entre los nodos.

En algunas redes la conexión entre nodos puede ser simétrica, es decir que el enlace va en ambas direcciones, esto es los nodos se afectan mutuamente en igual proporción, en algunas otras redes el enlace trabaja en una sóla dirección.

Las neuronas simuladas o unidades se pueden organizar también en redes de multicapas en donde cada capa es una red neuronal simulada y están conectadas por medio de enlaces entre capas; a la primera capa se le llama de entrada, a la última de salida y a todas las intermedias se les conoce como capas ocultas.

Las capas se encuentran en diferentes niveles que se denominan niveles de activación y representan la activación de una capa a otra.

El esquema general de aprendizaje conexionista parte de una red inicial compuesta por un conjunto de nodos con pesos en las conexiones entre ellas. A la red se le presenta un conjunto de ejemplos que provocan que los pesos en las conexiones cambien hasta reflejar el conocimiento que contienen los ejemplos.

El tipo de conexiones entre neuronas puede ser simétrico o no; los ejemplos presentados a la red pueden ser positivos y/o negativos; la forma de indicarle a la red si está bien o no su respuesta, con supervisión o no; cada uno de estos puntos mencionados se estudian en detalle más adelante.

Los diferentes métodos de aprendizaje en redes neuronales que se examinan en este capítulo son:

1. Aprendizaje mediante entrenamiento de perceptrones.
2. Aprendizaje mediante entrenamiento de redes neuronales.
3. Aprendizaje por refuerzos
4. Aprendizaje sin supervisión.
5. Aprendizaje competitivo

Y los diferentes tipos de redes que aparecen en este capítulo son:

1. Redes de Hopfield
2. Perceptrones.
3. Redes de retropropagación.
4. Redes neuronales de alimentación positiva.
5. Máquinas de Boltzman

Como las redes neuronales reales cumplen con ciertas características que se pueden simular mediante redes neuronales simuladas utilizando diferentes modelos de aprendizaje, a continuación se presenta el esquema general del aprendizaje conexionista para posteriormente analizar los conceptos básicos de diferentes modelos de redes neuronales simuladas, su método de aprendizaje, algoritmos y ejemplos de aplicación.

3.2 REDES DE HOPFIELD

Las redes de Hopfield han probado ser de interés para resolver problemas de optimización computacional.

La idea es ver a cada unidad de la red como una "hipótesis" y colocar pesos positivos en las conexiones entre unidades que representen hipótesis compatibles o que mutuamente se apoyen, y pesos negativos en las conexiones entre unidades que representen hipótesis incompatibles. "Como la red de Hopfield se dirige a un estado estable, trata de asignar verdadero y falso a varias hipótesis violando el menor número de restricciones posibles."[RICH].

En la figura 3.4 se muestra un ejemplo de una red simulada simple de Hopfield, en donde los elementos de procesamiento (unidades), están siempre en uno de dos estados: activos o inactivos. Las unidades coloreadas en gris se encuentran activas y las coloreadas en blanco están inactivas. Las unidades se encuentran conectadas unas a otras mediante conexiones simétricas con un peso determinado.

Una conexión con peso positivo indica que las unidades tienden a activarse una a la otra. Una conexión con peso negativo permite a una unidad activa desactivar a la unidad vecina.

Los orígenes de las redes de Hopfield se encuentran en un ensayo publicado en 1954 por el neuroanatomista Brian G.Cragg y el físico Nevill V.Temperley.

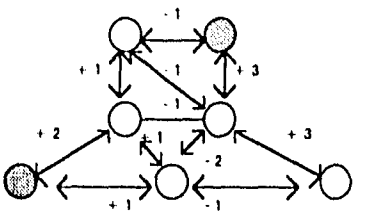


Figura 3.4: red simple de Hopfield

Algunas de las características de este tipo de red son:

- Representación distribuida - El conocimiento está almacenado por medio de patrones de activación através de un conjunto de unidades o elementos de procesamiento. Más aún, las representaciones distribuidas tienen la propiedad importante de que los objetos almacenados como patrones se pueden sobreimponer unos en otros.
- Control asíncrono distribuido - Cada elemento de procesamiento toma decisiones basándose únicamente en su propia situación local. Todas las acciones locales se suman para proporcionar una solución global.
- Memoria direccionable por contenido - Un número de patrones pueden ser guardados en una red. Para localizar un patrón sólo se necesita especificar una parte de él. Dado un conjunto de patrones y un caso particular la red automáticamente encuentra el patrón que mejor se ajusta al caso que se está proporcionando.

- Tolerancia de error - Si algunos elementos de procesamiento sólo coinciden parcialmente o fallan completamente la red continuará funcionando adecuadamente

La red simulada de Hopfield trabaja de la siguiente manera:

- Una unidad se selecciona aleatoriamente
- Si cualquiera de sus vecinas está activa, la unidad calcula la suma de los pesos en las conexiones de sus vecinas activas. Si la suma es positiva la unidad se convierte en activa; en caso contrario se convierte en inactiva.
- Se selecciona otra unidad en forma aleatoria, y el proceso se repite hasta que la red alcance un estado estable, es decir hasta que ninguna unidad pueda cambiar de estado. Este proceso es llamado *relajación paralela*

En la figura 3.4 se puede ver que la unidad en la esquina inferior izquierda tenderá a activar la unidad que está sobre ella. Esta unidad, a su vez, tratará de activar la unidad que está sobre ella. y ésta a su vez, tratará de activar la unidad a su derecha, pero la conexión es inhibidora y entonces hará fallar el intento, y así sucesivamente.

En la figura 3.5 se muestran los cuatro estados estables distintos que tiene la red simulada de la figura 3.4. Dado un estado inicial, la red simulada de la figura 3.4 necesariamente caerá en una de las cuatro configuraciones. El estado estable en el cual todas las unidades están inactivas puede ser alcanzado únicamente si se trata del estado inicial.

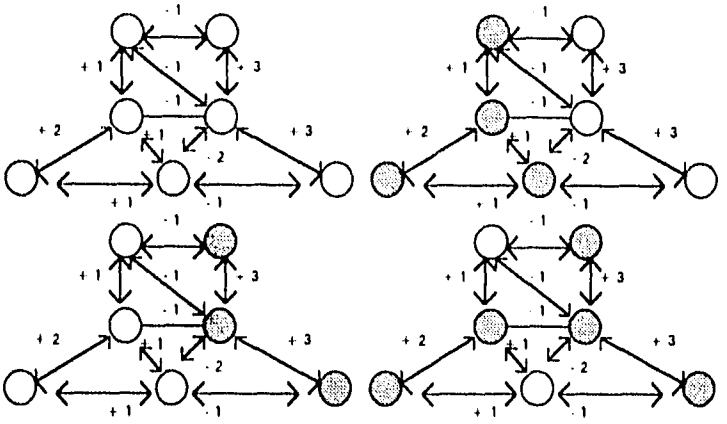


Figura 3.5: Los cuatro estados estables de una red de Hopfield en particular.

Hopfield demostró que dado cualquier conjunto de pesos y cualquier estado inicial, su algoritmo de relajación paralela eventualmente conduce a la red neuronal simulada a un estado estable. No puede haber divergencia u oscilación.

Para localizar un estado sólo se requiere proporcionar parte de él, ya que la red buscará el estado estable que mejor coincida con la información proporcionada. Por esto se dice que este tipo de búsqueda (relajación paralela) presenta un comportamiento de direccionamiento por contenido.

El relajamiento paralelo tiene también un comportamiento de corrección de errores ya que busca la solución que viole el mínimo número de restricciones, aún si el estado inicial contiene inconsistencias. Una restricción es violada, cuando por ejemplo dos unidades activas están conectadas con una conexión de peso negativo.

En este tipo de redes también puede lograrse un comportamiento de corrección de errores. Suponga que tenemos la siguiente descripción, "gris, grande, pez, come plankton". De inmediato imaginamos una ballena, aunque la ballena sea un mamífero no un pez.

Aunque el estado inicial tenga inconsistencias, la red simulada de Hopfield nos llevará a la solución que viola el menor número de restricciones. Con respecto a los procedimientos compara-localiza que generalmente se utilizan en IA tradicional, éstos son menos flexibles que los utilizados en esta red

Cuando en una red de Hopfield una unidad llega a fallar, es decir, se convierte en activa o inactiva cuando no debe, no se afecta el comportamiento general de la red ya que las unidades circundantes rápidamente la corregirán de nuevo.

Sería necesario un esfuerzo concertado desafortunado de muchas unidades equivocadas para impulsar a la red simulada a un estado estable equivocado.

En las redes simuladas formadas por miles de unidades altamente interconectadas, esa tolerancia a las fallas es aún mayor, ya que unidades y conexiones pueden desaparecer completamente sin afectar el comportamiento general de la red.

En las arquitecturas conexionistas los métodos de representación de conocimiento (conexiones), llevan por sí solo el aprendizaje automático.

Un problema que se presenta en la búsqueda de coincidencia con los estados estables es el de los mínimos locales.

Por ejemplo vease la figura 3.6. En esta figura podría pensarse que la curva representa un proceso que lleva del estado C al estado A; como puede verse podría presentarse el problema de que el patrón adecuado fuera B, por violar un menor número de restricciones, pero por el tipo de búsqueda al caer en A, que corresponde a un mínimo local, podría darse el caso de que nunca se llegase a B.

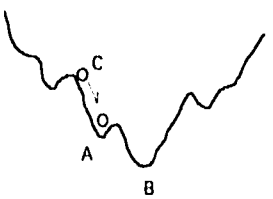


Figura 3.6: Una red de Hopfield como un modelo de memoria

En la figura 3.7 se muestra con más detalle el problema de mínimos locales y se presentan diferentes estados de una red simulada formando un espacio de búsqueda. Un estado aleatoriamente seleccionado se modificará al final en uno de los mínimos locales, llamado el estado estable más cercano. De ésta manera se llega a un comportamiento de direccionamiento por contenido.

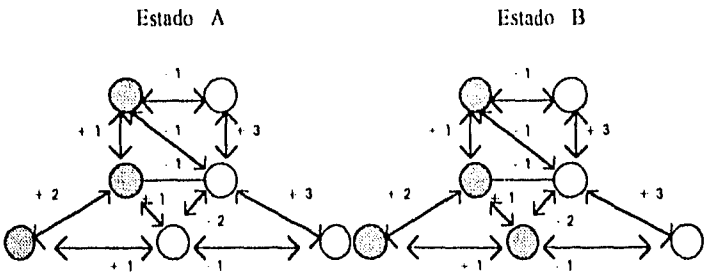


Figura 3.7: Una visión simple de una red de Hopfield en una computadora.

Desafortunadamente las redes neuronales simuladas de Hopfield no pueden encontrar soluciones globales porque llevan a estados estables. Si una red neuronal simulada alcanza un estado estable como el estado A en la figura 3.6, entonces ninguna unidad simple querrá cambiar su estado para moverse hacia arriba, por lo que la red nunca alcanzará el estado óptimo global B. Si algunas unidades deciden cambiar de estado simultáneamente, la red puede ser capaz de escalar la colina y llegar al estado B. Se necesita una forma de impulsar a las redes neuronales simuladas hacia los estados óptimos globales.

Las redes de Hopfield serán aplicadas para tratar de resolver problemas como el del "agente viajero". En este problema el agente viajero necesita visitar n ciudades una sola vez, en un plan de trabajo que minimice la longitud del viaje. Hopfield y Tank han usado estas redes para llegar a soluciones bastante favorables. A continuación se presenta la forma en que abordaron el problema.

En una red de Hopfield se programan *nxn* unidades neuronales donde *n* es el número de ciudades que deben ser visitadas. En la figura 3.8 aparece un ejemplo de recorrido. Cada renglón representa una ciudad y la columna indica el orden en que deben visitarse. La ciudad inicial se marca con una unidad activa en la columna 1, la siguiente ciudad en la columna 2, etc. El recorrido de la figura 3.8 pasa por las ciudades D, B, E, H, G, F, C, A y regresa a D.

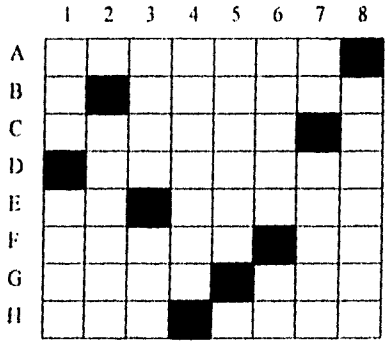


Figura 3.8: Representación del recorrido de un agente viajero en una red de Hopfield.

Como todas las redes de Hopfield, este arreglo de n por n contiene un número de conexiones ponderadas, las cuales son inicializadas para reflejar exactamente las restricciones de un caso particular del problema. Los pesos de las conexiones se codifican a mano, utilizándose el siguiente contexto:

1. Cada unidad es conectada con un peso negativo a cualquier otra en su columna, porque sólo puede ser visitada una ciudad a la vez.
2. Toda unidad inhibe cualquier otra en su renglón, porque cada ciudad sólo puede ser visitada una vez.
3. Las unidades en columnas adyacentes inhiben a las otras en proporción a sus distancias entre las ciudades representadas en sus renglones.

Por ejemplo, Si la ciudad D está lejos de la ciudad G, entonces la cuarta unidad en la columna 3 inhibe la séptima unidad en las columnas 2 y 4.

Cada unidad representa alguna hipótesis acerca de la posición de una ciudad particular en un pequeño recorrido. Se inicia dando valores de activación aleatorios a las unidades. Una vez que todos los pesos son inicializados, las unidades son actualizadas en forma asíncrona hasta que se alcanza un estado estable. En este ejemplo los estados estables de la red corresponden a pequeños recorridos, porque los conflictos entre restricciones son mínimos.

3.3 PERCEPTRONES.

La introducción de los perceptrones a finales de los años 50's creó una gran inquietud debido a que se trataba de dispositivos de modelado muy parecidos a una neurona con algoritmos de aprendizaje bien definidos. Mediante ejemplos de entrenamiento un perceptrón es capaz de aprender a reconocer modelos o patrones.

El perceptrón fue inventado por Rosemblatt en 1962 y fue uno de los primeros modelos de redes neuronales.

Un perceptrón modela una neurona tomando la suma ponderada de sus entradas y enviando como salida 1 si la suma es mayor que algún valor de umbral ajustable, en caso contrario envía 0. En la figura 3.9 se muestra un ejemplo de un perceptrón. La diferencia entre una red de Hopfield y un perceptrón es que las conexiones del perceptrón no son simétricas, sino que son en una sola dirección.

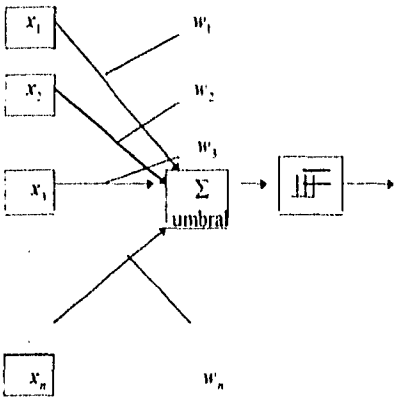


Figura 3.9 : Perceptrón

Las entradas $x_1, x_2, x_3, \ldots, x_n$ y los pesos $(w_1, w_2, w_3, \ldots, w_n)$ de las conexiones de la figura 3.9 son valores reales, positivos o negativos. Los pesos permiten ponderar el efecto de cada una de las entradas para activar o inhibir a la neurona simulada que los recibe.

El perceptrón está formado por un conjunto de pesos, un procesador de suma, y un procesador de ajuste de umbrales. El aprendizaje se da mediante un proceso de modificación de los valores de los pesos y los umbrales. El umbral se representa agregando una entrada adicional con peso w_0, como se muestra en la figura 3.10, este peso puede ser pensado como la inclinación del perceptrón a dispararse independientemente de sus entradas. El perceptrón de la figura 3.10 se dispara sólo si la suma ponderada es mayor que cero.

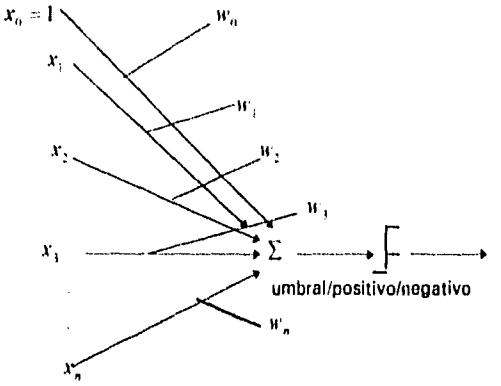


Figura 3.10: Perceptrón con umbral ajustable implementado como Peso Adicional.

Por ejemplo en el problema de clasificación de patrones mostrado en la figura 3.11, dados los valores para x_1 y x_2 se desea entrenar un perceptrón para que responda 1 si el dato pertenece a la clase de los puntos blancos y 0 si el dato pertenece a la clase de los puntos negros. Se debe inducir una regla del conjunto de casos de entrenamiento.

Este problema es linealmente separable, porque se puede dibujar una linea que separe una clase de la otra.

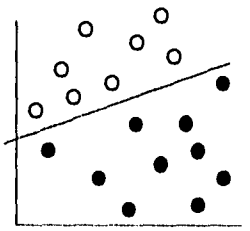


Figura 3.11 Problema de clasificación de patrones linealmente separables

ESTA TESIS NO DEBE
SALIR DE LA BIBLIOTECA

El comportamiento del perceptrón ante este problema se describe a continuación.

Si $\bar{x}$ es un vector de entrada $(x_1, x_2, \ldots, x_n)$ puede observarse que la función de suma ponderada $g(x)$ y la función de salida $o(x)$ pueden ser definidas como:

$$g(x) = \sum_{i=0}^{n} w_i x_i$$

Donde $w_0 x_0$ se refieren al umbral representados como otro peso con $x_0 = 1$ y $o(x)$ que representa la respuesta de la neurona.

$$o(x) = \begin{cases} 1 & si \quad g(x) > 0 \\ 0 & si \quad g(x) < 0 \end{cases}$$

Si consideramos el caso de sólo dos entradas (como en la figura 11) entonces :

Si $g(x)$ es exactamente 0, el perceptrón no puede decidir qué disparar. Un pequeño cambio en las entradas podría causar que el dispositivo tomase cualquier camino. Haciendo la ecuación $g(x) = 0$, se tiene la ecuación de una linea :

$$g(x) = w_0 + w_1 x_1 + w_2 x_2 = 0$$

Despejando x_2 se obtiene

$$x_2 = \frac{w_1}{w_2} x_1 - \frac{w_0}{w_2}$$

La posición de la línea que separa los dos conjuntos de puntos está completamente determinada por los pesos w_0, w_1 y w_2. Si un vector de entrada se encuentra de un lado de la línea, el perceptrón responderá 0. Una línea que correctamente separe los ejemplos de entrenamiento corresponde a un perceptrón que funciona perfectamente. Tal línea es llamada superficie de decisión.

Un algoritmo informal de aprendizaje es el siguiente:

- Si el perceptrón dispara cuando no debe disparar, hacer que los pesos w_i sean más pequeños en forma proporcional a x_i :
- Si el perceptrón no dispara cuando debe disparar, hacer cada peso w_i mayor por una cantidad parecida.

Por ejemplo se presenta a continuación el análisis al aplicar el algoritmo informal para entrenar un perceptrón con tres entradas para que dispare sólo cuando su primera entrada está encendida. Al proporcionar cada terna el perceptrón debe disparar en la presencia de una x_1 activa; si no lo hace se incrementará el peso w_1 (pueden también incrementarse los otros pesos); si el perceptrón dispara incorrectamente (es decir, dispara siendo $x_1 = 0$ hay que decrementar los otros pesos diferentes de w_1, no decrementar w_1 porque al ser $x_1 = 0$ multiplicado por w_0 no se afecta la función de evaluación. Por otro lado, w_0 hallará un valor basado en el número total de disparos incorrectos contra los tiros no realizados incorrectos, lo que ocasionará que w_1 sea lo suficientemente grande para sobrepasar w_0, mientras que w_2 y w_3 no serán suficientemente poderosos para disparar el perceptrón, aún en la presencia de ambos x_2 y x_3.

Regresando a las funciones $g(x)$ y $o(x)$, vemos que mientras el signo de $g(x)$ es crucial para determinar si el perceptrón se disparará, la magnitud o valor absoluto también es importante ya que nos dice que tan lejos un vector de entrada $\bar{x}$ se encuentra de la superficie de decisión. Lo que proporciona una forma de probar qué tan adecuado es el conjunto de pesos. Sea $\bar{w}$ el vector de pesos $(w_0, w_1, w_2, \ldots, w_n)$ y sea X el subconjunto de ejemplos de entrenamiento mal clasificados con ese conjunto de pesos, se define la *función de criterio del perceptrón*, $J(\bar{w})$, como la suma de las distancias de los vectores de entrada mal clasificados a la superficie de decisión:

$$J(\bar{w}) = \sum_{x \in X} \left| \sum_{i=0}^{n} w_i x_i \right| = \sum_{x \in X} |\bar{w}\bar{x}|$$

Para tener un conjunto de pesos mejor que el actual hay que disminuir $J(\bar{w})$ hasta que todas las entradas estén clasificadas correctamente, es decir que $J(\bar{w}) = 0$.

Para minimizar $J(\bar{w})$ se usa una forma de búsqueda local de ascenso de la colina, conocida como descenso del gradiente. El gradiente es un vector que nos indica en qué dirección movernos en el espacio de los pesos para reducir $J(\bar{w})$. Para encontrar un vector de pesos solución, simplemente se cambian los pesos en la dirección del gradiente y repetimos hasta que $J(\bar{w}) = 0$. La regla para actualizar los pesos en la iteración $t+1$ es:

$$\bar{w}_{t+1} = \bar{w}_t + \eta \nabla J$$

en donde el gradiente

$$\nabla J(\bar{w}) = \begin{cases} \bar{x} & si \ \ \bar{x}_provoco_que_con_los_pesos_\bar{w}_el_perceptron \\ & no_disparara_cuando_debia_hacerlo \\ -\bar{x} & si \ \ \bar{x}_provoco_que_con_los_pesos_\bar{w}_el_perceptron \\ & disparara_cuando_no_debia_hacerlo \end{cases}$$

η es un factor de escala que indica que tan lejos o rápido debe moverse en la dirección del gradiente. Una η pequeña conduce a un aprendizaje más lento, pero una η grande puede causar un movimiento através del espacio de pesos que pase tan rápido que no encuentre al vector solución. Si se toma η como una constante se llega a un *algoritmo de aprendizaje para perceptrón con incrementos fijos*:

Algoritmo: Aprendizaje para perceptrón con incrementos fijos

Dados: Un problema de clasificación con casos de n entradas $(x_1, x_2, \ldots, x_n)$ y dos clases de salida.

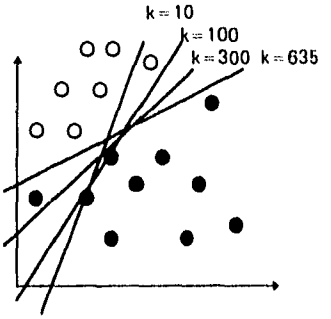
Calcular: un conjunto de pesos $(w_0, w_1, \ldots, w_n)$ que provoquen que el perceptrón se dispare si la entrada cae en la primera de las dos clases de salida.

1. Crear un perceptrón con $n+1$ entradas y $n+1$ pesos, donde la entrada extra x_0 se inicializa siempre en 1.
2. Inicializar los pesos $(w_0, w_1, \ldots, w_n)$ con valores reales aleatorios.
3. Iterar a través del conjunto de entrenamiento, determinar el subconjunto de casos de entrenamiento.
4. Si todos los ejemplos son clasificados correctamente, proporcionar como resultado del algoritmo esos pesos y salir. $\bar{S}$
5. En caso contrario, calcular el vector $\bar{S}$ formado con la suma de los casos de entrada mal clasificados donde todo vector de entrada de la forma $(x_1, x_2, \ldots, x_n)$ en $\bar{S}$, es el vector $\bar{x}$, si $\bar{x}$ es una entrada para la cual el perceptrón no disparó cuando debia hacerlo; es el vector $-\bar{x}$ si $\bar{x}$ es una entrada para la cual el perceptrón disparó cuando no debia disparar. Multiplicar la suma por un factor de escala η.
6. Variar los pesos $(w_0, w_1, \ldots, w_n)$ sumándoles los elementos del vector $\bar{S}$. Y regresar al paso 3.

"El algoritmo de aprendizaje para el perceptrón es un algoritmo de búsqueda. Empieza con un estado inicial aleatorio y encuentra un estado solución. El espacio de búsqueda es simplemente todas las posibles asignaciones de valores reales a los pesos del perceptrón, y la estrategia de búsqueda es el descenso del gradiente."[RICH]

Rosemblant en 1962 en su teorema de convergencia del perceptrón, garantiza que el perceptrón encontrará un estado solución, y entonces aprenderá a clasificar cualquier conjunto de entradas linealmente separables. Es decir que el teorema demuestra que en el espacio de pesos, no hay mínimos locales que no correspondan al mínimo global. Este teorema no se demostró para perceptrones de "multicapas". Por ejemplo en la figura 3.12 puede apreciarse el aprendizaje de un perceptrón para clasificar los casos proporcionados en la figura 3.11. En este ejemplo k es el número de pasadas a través de los datos de entrenamiento que se requieren para determinar cada línea de separación. El programa utilizó 635 veces el juego de datos para llegar a la recta que delimita correctamente los dos conjuntos.

En la tabla pueden verse los pesos a utilizar en las conexiones y el umbral.



k	w_0	w_1	w_2
10	0.41	-0.17	0.14
100	0.22	-0.14	0.11
300	-0.10	-0.08	0.07
635	-0.49	-0.10	0.14

Figura 3.12: Un perceptrón aprende a resolver un problema de clasificación

El algoritmo de aprendizaje del perceptrón puede ajustar correctamente pesos entre entradas y salidas, pero no puede ajustar pesos entre perceptrones.

Hubo mucha especulación acerca de cómo podrían construirse sistemas inteligentes a partir de bloques de percentrones. En su libro *Perceptrons*, Minsky y Papert pusieron fin a tal especulación analizando las capacidades computacionales de los perceptrones.

En la época en que *Perceptrons* fue publicado, nadie sabía cómo se podía hacer para que los perceptrones de multicapas aprendieran. Mynsky y Papert reflexionaban: "El perceptrón tiene muchas características que llaman la atención: su linealidad, su intrigante teorema de aprendizaje, no hay razón para suponer que cualquiera de estas virtudes se mantengan en la versión de multicapas. Sin embargo, consideramos como un problema de investigación importante aceptar (o rechazar) nuestro juicio intuitivo de que la extensión es estéril."[RICH].

La investigación sobre aprendizaje de perceptrones se detuvo en los 70's aunque resurgió en los 80's, cuando algunos procedimientos de aprendizaje para perceptrones de multicapas, también llamadas redes de multicapas, fueron propuestos. A continuación se estudiarán este tipo de redes.

3.4 REDES DE RETROPROPAGACIÓN.

La habilidad para entrenar redes de multicapas se considera un avance importante en la construcción de máquinas inteligentes que trabajan con componentes de tipo neurona.

La meta que se pretende alcanzar es que una masa, relativamente sin una forma determinada, de elementos del tipo neurona aprenda rápidamente a realizar tareas útiles, y que generalice las entradas que se le proporcionan para así poder construir estas redes en gran escala capaces de aprender eficientemente.

Las redes de multicapas son redes neuronales más complejas que los perceptrones, que como se mencionó antes, modelan a una sola neurona. Existen diferentes clases de redes de multicapas, entre ellas se encuentran las llamadas redes de alimentación positiva, en las cuales cada unidad en una capa está conectada hacia adelante a toda unidad de la siguiente capa. Las activaciones van de la capa de entrada hacia la capa oculta, y de ésta hacia la capa de salida.

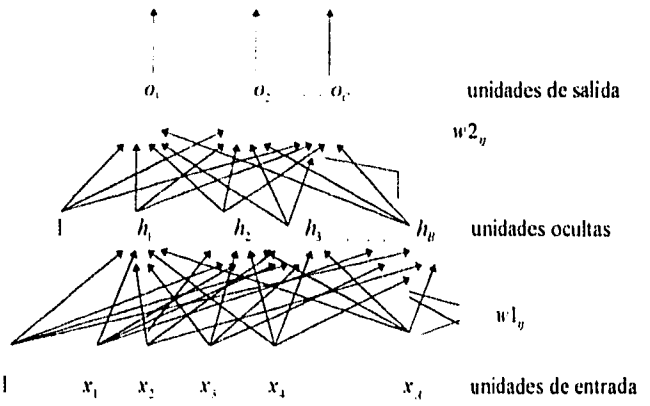


Figura 3.13: Red de multicapas

Un ejemplo de una red de multicapas con tres capas se muestra en la figura 3.13, en donde las variables x_i, h_i, y o_i representan niveles de activación de las unidades de entrada, de las unidades ocultas y unidades de salida respectivamente Los pesos en las conexiones entre las unidades de entrada y las capas ocultas están dadas por las variables $w1_{ij}$, mientras que los pesos en las conexiones entre las capas ocultas y las capas de salida están dadas por las variables $w2_{ij}$.

El aprendizaje en la red de alimentación positiva consiste en reconocer similaridades en los datos y se da al codificar los pesos de las conexiones entre las unidades.

El método de aprendizaje que se emplea es el método de retropropagación o propagación hacia atrás, que consta de dos etapas.

1. La etapa "hacia adelante", que consiste en estimular a la red al presentar una entrada muestra que permita activaciones que fluyan de la capa de entrada hasta llegar a una capa de salida.

2. En la etapa "hacia atrás", la salida de la red (resultado del paso hacia adelante) se compara con la salida meta y los estimadores de error se calculan para las unidades de salida. Los pesos de las conexiones que van a las unidades de salida se ajustan para disminuir estos errores. Los estimadores de error de las unidades de salida se usan para calcular estimadores de error para las unidades de las capas ocultas y se ajustan tanto pesos de conexiones como de las capas de salida hacia umbrales en las unidades de entrada.

El algoritmo en detalle se presentará más adelante.

Los pesos asociados a las unidades de salida se ajustan para disminuir errores.

Los errores provocan modificaciones en los umbrales de las unidades de las capas anteriores hasta llegar a cambiar los umbrales de las unidades de la capa de entrada, haciendo posible el aprendizaje a partir de los errores.

Las unidades en una retropropagación requieren de una función de activación un poco diferente a la del perceptrón, como se muestra en la figura 3.14. Una unidad de retropropagación suma sus entradas ponderadas y, a diferencia del perceptrón, produce como salida un valor real entre 0 y 1, basándose las redes de retropropagación en una función sigmoide (o de forma de S), que es continua y diferenciable.

Sea *suma* la suma ponderada de las entradas de una unidad. La ecuación para las unidades de salida está dada por :

$$salida = \frac{1}{1 + e^{-suma}}$$

Cuando la suma es 0, la salida es 0.5 (a diferencia del perceptrón que debe ser 0 o 1). Conforme *suma* se hace mayor, la salida se aproxima a 1. Conforme *suma* se hace más pequeña, la salida se aproxima a 0.

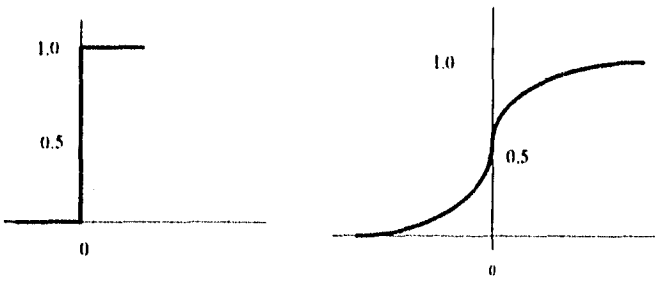


Figura 3.14 : La función de activación escalonada del perceptrón (izquierda), y la función de activación de la retropropagación en forma sigmoide (derecha)

A diferencia del algoritmo del perceptrón, el algoritmo de retropropagación frecuentemente actualiza sus pesos incrementándolos.

Después de haber visto todas las parejas de entradas-salida y ajustado los pesos de la red igual número de veces, se dice que *un período* ha sido completado. Entrenar una red de retropropagación por lo general requiere de muchos períodos.

A continuación se presenta el siguiente algoritmo que se basa en el método de retropropagación:

Algoritmo de Retropropagación.

Dados : Un conjunto de parejas de vectores entrada-salida

Calcular : Un conjunto de pesos para una red de tres capas que asocie entradas con sus correspondientes salidas.

1. Sea A el número de unidades en la capa de entrada, determinado por la longitud de los vectores de entrada de entrenamiento. Sea C el número de unidades en la capa de salida. Y sea B, el número de unidades de la capa oculta. Las capas de entrada y oculta, cada una tiene una unidad extra usada como umbral; por lo que, las unidades en estas capas algunas veces son indexadas con rangos (0, . . . ,A) y (0, . . . ,B).

Los niveles de activación de las unidades en la capa de entrada se representan utilizando variables x_i, en la capa oculta variables h_j, y en la capa de salida las variables o_k. Los pesos que conectan la capa de entrada a la capa oculta son indicados por $w1_{ij}$, donde el subíndice i indica la unidad de entrada y el j indica la unidad oculta. Los pesos que conectan la capa oculta se representan como $w2_{ij}$, con i indexando la unidad oculta y j indexando la unidad de salida.

2. Inicializar aleatoriamente cada uno de los pesos en la red, con un número entre -0.1 y 0.1.

$$w1_{ij} = random(-0.1,0.1) \quad \text{para toda } i = 0,\ldots,A,\ j = 1,\ldots,B$$

$$w2_{ij} = random(-0.1,0.1) \quad \text{para toda } i = 0,\ldots,B,\ j = 1,\ldots,C$$

3. Inicializar los niveles de activación los que permanecen fijos durante todo el proceso no debe cambiar nunca $x_0 = 1.0$ y $h_0 = 1.0$
4. Escoger un ejemplo de entrenamiento asumiendo que el vector de entrada es x_i, y la salida meta es y_k. Asignar niveles de activación a las unidades de entrada.
5. Propagar las activaciones de las unidades de la capa de entrada a las unidades en la capa oculta usando la función de activación sigmoide.

$$h_j = \frac{1}{1+\exp\left(-\sum_{i=0}^{A} w1_{ij} x_i\right)} \qquad \text{para toda } j = 1,\ldots,B$$

Notar que i varía de 0 a A, $w1_{0j}$ es el peso de umbral para la unidad oculta j (su inclinación a disparar es independiente de sus entradas), x_0 es siempre 1.0.

6. Propagar la activación de las unidades de la capa oculta a las unidades de la capa de salida, calculando

$$o_j = \frac{1}{1+\exp\left(-\sum_{i=0}^{B} w2_{ij} h_i\right)} \qquad \text{para toda } j = 1,\ldots,C$$

El peso del umbral $w2_{ij}$ para la unidad de salida j juega un papel importante en la suma ponderada.

7. Calcular los errores de las unidades de la capa de salida, denotados por $\delta 2_j$. Los errores están basados en la salida actual (o_j) de la red y en la salida propuesta en el ejemplo de entrenamiento (y_j).

$$\delta 2_j = o_j(1-o_j)(y_j - o_j) \qquad \text{para toda } j = 1,\ldots,C$$

8. Calcular los errores de las unidades en la capa oculta, denotados, por $\delta 1_j$

$$\delta 1_j = h_j(1-h_j)\sum_{i=1}^{C} \delta 2_i w2_{ij} \qquad \text{para toda } j = 1,\ldots,B$$

9. Ajustar los pesos entre la capa oculta y la capa de salida. Después de varios ejemplos de entrenamiento, un valor encontrado razonable de η es 0.35 donde η representa la razón de aprendizaje.

$$\Delta w2_{ij} = \eta \delta 2_j h_i \qquad \text{para } i = 0,\ldots,B,\ j = 1,\ldots,C$$

10. Ajustar los pesos entre la capa de entrada y la capa oculta

$$\Delta w1_{ij} = \eta \delta 1_j x_i \qquad \text{para } i = 0,\ldots,A,\ j = 1,\ldots,B$$

11. Regresar al paso 4 y repetir. Cuando todas las parejas de entrada y salida hayan sido presentadas a la red un periodo habrá sido completado. Repetir los pasos 4 a 10 para todos los periodos que se desee.

El algoritmo se puede generalizar para redes de más de 3 capas. Una red con una capa oculta puede calcular cualquier función que una red con muchas capas ocultas pueda calcular: con un número exponencial de unidades ocultas, una unidad oculta podría ser asignada a todo patrón de entrada posible. El aprendizaje es más rápido con capas múltiples ocultas, especialmente si la entrada es altamente no lineal, y entonces, es difícil separar con una serie de líneas rectas los patrones de entrada. Para cada capa oculta extra, se necesita un paso de propagación hacia adelante entre los pasos 6 y 7 del algoritmo antes propuesto, un paso de cálculo de error entre los pasos 8 y 9, y un paso de ajuste de pesos entre los pasos 10 y 11. El cálculo de errores para las unidades ocultas es presentado en la ecuación del paso 8, pero con i variando sobre las unidades de la siguiente capa, no necesariamente sobre las unidades de la capa de salida.

La rapidez de aprendizaje se puede incrementar modificando los pasos 9 y 10 con la actualización de pesos al incluir un término α. Las fórmulas para actualizar estos pesos son las siguientes:

$$\Delta w2_{ij}(t+1) = \eta \delta 2_j h_i + \alpha \Delta w2_{ij}(t)$$

$$\Delta w1_{ij}(t+1) = \eta \delta 1_j h_i + \alpha \Delta w1_{ij}(t)$$

donde h_i, x_i, $\delta 1_j$ y $\delta 2_j$ son valores de los errores calculados en el tiempo $t+1$. $\Delta w1_{ij}(t)$ es el cambio que experimenta el peso durante el paso anterior de adelante hacia atrás. Si α se inicializa con 0.9, la rapidez de aprendizaje se incrementará. En experimentos realizados, los mejores resultados se han obtenido al hacer el término α igual a cero para algunos de los primeros pasos de entrenamiento, y después incrementando a 0.9 para el resto del entrenamiento. De este modo el algoritmo primero encuentra una dirección generalmente buena y después se mueve en esa dirección con mayor rapidez.

Si se recuerda que la función de activación tiene una apariencia de sigmoide, en donde pueden darse pesos no definidos para salidas de la red, para alcanzar los valores de salida binarios 0.0 y 1.0, a las y_i's de los pasos 4 y 7 de arriba se les dan frecuentemente los valores 0.1 y 0.9 en lugar de 0.0 y 1.0. La sigmoide se requiere en la retropropagación porque la derivación de la regla para actualizar pesos necesita que la función de activación sea continua y diferenciable.

Existe una función de error que define una superficie en el espacio de pesos, y los pesos son modificados en la dirección del gradiente de la superficie ya que ésta indica la dirección en que se produce el máximo cambio. La superficie de error para redes de multicapas es más complicada que la superficie de error para perceptrones. Una diferencia importante en las superficies es la existencia de mínimos locales; si un conjunto de pesos se encuentra en un mínimo local, la red de retropropagación nunca alcanzará el conjunto de pesos óptimo.

Para superar el problema de los mínimos locales existen algunos métodos, por ejemplo, al utilizar el factor α, se pretende mantener los cambios de los pesos moviéndose en la dirección adecuada, lo que permite al algoritmo de retropropagación saltar sobre mínimos locales. El ajustar la función de activación de las unidades puede causar un efecto en la susceptibilidad de la red a los mínimos locales.

Las redes de retropropagación rara vez caen en mínimos locales. En redes más grandes, el espacio de pesos de muchas más dimensiones proporciona muchos grados de libertad al algoritmo, por lo que la falta de un teorema de convergencia no es un inconveniente en la práctica. El algoritmo de retropropagación fue propuesto por varios investigadores en el pasado, pero fue descartado por los grandes problemas que se creía podían surgir con los mínimos locales, ya que antes de que existieran las rápidas computadoras digitales, los investigadores sólo podían juzgar sus ideas probando teoremas acerca de ellas, y no tenían idea de que en la práctica, los mínimos locales surgían rara vez.

Las redes de retropropagación tienen un serio problema que es su lento aprendizaje, pues aún las tareas simples requieren períodos largos de entrenamiento, especialmente si los parámetros de aprendizaje no se seleccionan cuidadosamente.

La idea de incrementar la velocidad de aprendizaje construyendo hardware de retropropagación de propósito especial es atractiva. Aunque nuevas variaciones rápidas de retropropagación y otros algoritmos de aprendizaje aparecen habitualmente en la literatura, Fahlman en 1988 cita 'Para cuando un algoritmo es transformado en hardware e incluido en un sistema de computación, el algoritmo tiende a ser obsoleto."[RICH]

La posibilidad de construir redes neuronales simuladas que aprendan y que sean capaces de generalizar entradas que no se les hayan presentado anteriormente para la solución de problemas de clasificación de caracteres manuscritos o de reconocimiento de una cara es todavía lejana.

En estas redes, si todas las posibles entradas y todas las posibles salidas son mostradas a una red de retropropagación, la red encontrará (probablemente) un conjunto de pesos que asocie las entradas y las salidas, pero en gran cantidad de problemas de IA, es imposible dar todas las entradas posibles. Como en el caso del reconocimiento de una cara y el reconocimiento de un carácter, existe un número infinito de representaciones y expresiones para una cara, y un número infinito de tipos y tamaños para un carácter. Sin embargo, los seres humanos aprenden a clasificar estos objetos fácilmente con sólo algunos ejemplos.

Las redes de retropropagación que trabajan en dominios de clasificación, donde entradas similares se mapean en salidas similares, aplicarán interpolación cuando una entrada dada no haya sido vista antes, para poder resolver el problema. Por ejemplo, después de aprender a distinguir algunas A's de diferentes tamaños de algunas B's de diferente tamaño, una red usualmente será capaz de distinguir cualquier tamaño de A de cualquier tamaño de B. Además la generalización ayudará a sobreponerse a cualquier ruido indeseable en las entradas.

3.5 APRENDIZAJE POR REFUERZOS

Existe una manera de entrenar redes neuronales simuladas no con salidas muestra sino con premios y castigos como en el proceso de entrenamiento de animales.

Barto en 1985 describe una red neuronal simulada con aprendizaje por refuerzos que aprende de la siguiente manera

1. Se presenta a la red neuronal simulada una entrada del conjunto de entrenamiento.
2. La red neuronal simulada calcula lo que cree sería la salida.
3. El profesor proporciona a la red neuronal simulada un juicio de valor real.
4. La red neuronal simulada ajusta sus pesos, y el proceso se repite.

Un valor positivo en el paso 3 indica un buen desempeño, mientras que un valor negativo indica un mal desempeño. La red neuronal simulada busca un conjunto de pesos que le eviten juicios de valor negativos en el futuro.

La diferencia con otros métodos es que el aprendizaje requiere de un profesor que entrene a la red proporcionándole ejemplos y calificaciones.

3.6 APRENDIZAJE SIN SUPERVISIÓN Y APRENDIZAJE COMPETITIVO.

El aprendizaje sin supervisión no requiere de un profesor y es análogo al aprendizaje por descubrimiento en IA simbólica. Sólo necesita un conjunto de datos de entrada para que la red pueda jugar con ellos tratando de descubrir las regularidades y las relaciones entre ellos.

"El aprendizaje sin supervisión puede ser usado como un modelo de *descubrimiento de características* en un aprendizaje supervisado."[RICH]

	¿tiene cabello?	¿tiene escamas?	¿tiene plumas?	¿vuela?	¿vive en el agua?	¿Es ovíparo?
PERRO	1	0	0	0	0	0
GATO	1	0	0	0	0	0
MURCIÉLAGO	1	0	0	1	0	0
BALLENA	1	0	0	0	1	0
CANARIO	0	0	1	1	0	1
PETIROJO	0	0	1	1	0	1
AVESTRUZ	0	0	1	1	0	1
VÍBORA	0	1	0	0	0	1
LAGARTO	0	1	0	0	0	1
CAIMÁN	0	1	0	0	1	1

Figura 3.15: Datos en un ejemplo de aprendizaje sin supervisión

Por ejemplo, en la figura 3.15 se tiene un grupo de diez animales, cada uno con sus características propias; éstos deben caer en algún grupo de mamíferos, reptiles o aves.

Se desea construir una red que pueda aprender a qué grupo pertenece un animal en particular, y además generalice de tal manera que pueda identificar animales que no haya visto todavía. Se puede resolver este problema con una red de retropropagación de seis entradas y tres salidas. A la red se le entrena mediante un ejemplo, toma una entrada, se observa su salida y la red actualiza los pesos de acuerdo a los errores que cometa. Todo esto sin un profesor del cual pueda aprender; sin embargo los errores no pueden ser calculados, como en el método de retropropagación.

El problema al aplicar el método al ejemplo arriba mencionado, consiste en asegurar que sólo una de las tres salidas se active para cualquier entrada dada. Una solución es que las unidades de salida se disputen entre sí el control de un vector de entrada, como se muestra en la figura 3.16.

Las unidades de entrada están directamente conectadas a las unidades de salida, como en el perceptrón, pero las unidades de salida también están conectadas entre sí mediante conexiones negativas o inhibitorias. La unidad de salida con mayor valor de activación a lo largo de sus conexiones de entrada inicialmente, desanima mayormente a sus competidores, lo que da como resultado que los competidores se vuelvan más débiles perdiendo su poder de inhibición sobre la unidad de salida más fuerte, y su efecto inhibidor sobre las otras unidades de salida se vuelve débil, lo que ocasiona que las otras unidades de salida se encuentren completamente inactivas. A este tipo de inhibición mutua se le conoce como comportamiento de "el ganador toma todo".

A este esquema de aprendizaje sin supervisión se le conoce como aprendizaje competitivo; en este aprendizaje las unidades de salida luchan por el control sobre porciones del espacio de entradas.

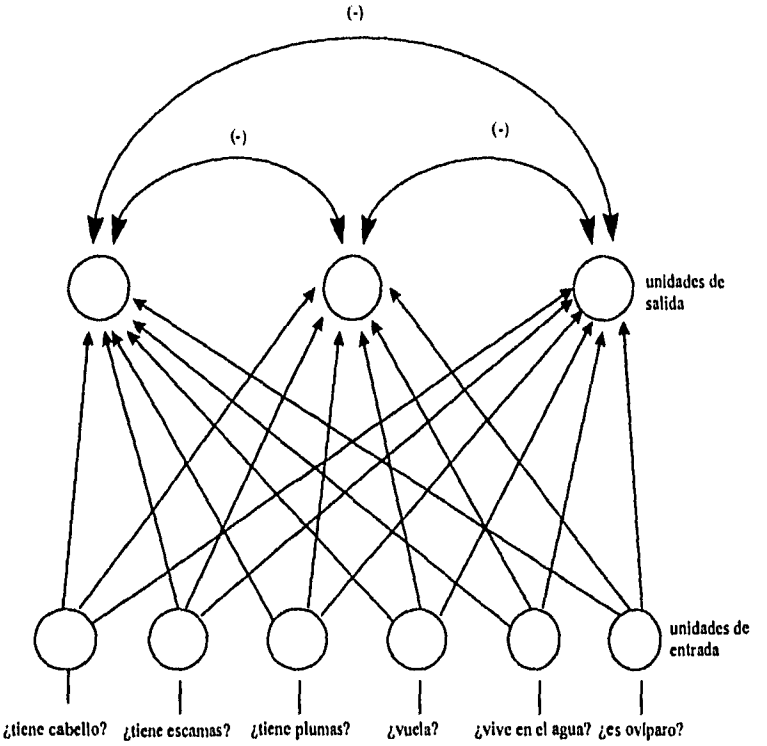


Figura 3.16: Red neuronal en un aprendizaje competitivo.

Un algoritmo de aprendizaje competitivo es el siguiente:

Dada: Una red de retropropagación consistente de n unidades de entrada de valores binarios directamente conectadas a cualquier número de unidades de salida.

Calcular: Un conjunto de pesos tales que las unidades de salida se vuelven activas de acuerdo a los niveles de activación de las unidades de entrada.

1. Presentar un vector de entrada, denotado por $(x_1, x_2, \ldots, x_n)$
2. Calcular la activación inicial para cada unidad de salida mediante la suma ponderada de sus entradas.
3. Dejar que las unidades de salida luchen hasta que sólo una esté activa.
4. Ajustar los pesos en las conexiones de entrada que lleven a una sola unidad de salida activa:

$$\Delta w_j = \eta \frac{x_j}{m} - \eta w_j \qquad \text{para toda } j = 1,\ldots,n$$

donde w_j es el peso n la conexión de la unidad de entrada j en la unidad de salida activa, x_j es el valor j-ésimo de entrada, n es el número de unidades de entrada que están activas en el vector de entrada que fue seleccionado en el paso 1, y η es la razón de aprendizaje (con un valor constante pequeño).

5. Repetir los pasos del 1 al 4 para todos los ejemplos de entrada. El algoritmo termina cuando los cambios de pesos sean muy pequeños.

Es importante hacer notar que:

- En el paso 2 no hay razón para pasar la suma ponderada a una función sigmoide como se hace en retropropagación, porque sólo se calculan los niveles de activación con el propósito de obtener la unidad de salida más altamente activa.

- En el paso 3 cualquier método para determinar la unidad de salida más altamente activa es suficiente.

La regla de modificación de pesos del paso 4 hace que la unidad de salida activa más próxima se dispare cuando ve la misma entrada de nuevo. Si se presenta una y otra vez la misma entrada, la unidad de salida ajustará sus pesos para la máxima activación ante esa entrada.

Debido a que los vectores de entrada llegan juntos, las unidades de salida nunca llegan a un conjunto ideal de pesos. La esperanza es que cada unidad de salida encuentre un grupo de vectores de entrada y se base en él, para alcanzar activaciones altas cuando se encuentre con estas entradas.

El algoritmo de aprendizaje competitivo es bueno aunque tiene algunos problemas, debido a que algunas veces una unidad de salida siempre puede ganar, a pesar de la existencia de más de un grupo de vectores de entrada. Si dos grupos de vectores de entrada están muy cercanos, una unidad de salida puede asignar pesos que den un alto nivel de activación cuando se encuentren con una entrada de cualquiera de los grupos de vectores.

En otras palabras, el algoritmo de aprendizaje competitivo puede fluctuar entre los dos grupos de vectores. Generalmente otra unidad de salida ganará eventualmente y se moverá a reclamar uno de los dos grupos. Sin embargo, si las otras unidades de salida son enteramente inexcitables por los vectores de entrada, ellas nunca ganarán la competencia. Una solución a este problema conocida con el nombre de "aprendizaje de escape", consiste en modificar los pesos pertenecientes a las unidades de salida relativamente inactivas así como también de la más activa. La regla de actualización de pesos para unidades de salida es la misma que el algoritmo antes mencionado con la excepción de que se mueven sus pesos con una razón de aprendizaje η mucho menor en orden de magnitud.

Otra solución alternativa es la de ajustar la sensibilidad de una unidad de salida a través del uso de desviaciones o umbral ajustable (las desviaciones se mencionaron en perceptrones y correspondia a la inclinación de una unidad a dispararse independientemente de sus entradas) A las unidades de salida que pocas veces ganan en los procesos de aprendizaje competitivos pueden asignarse umbrales mayores.

Con esto se obtiene el control sobre una porción muy grande del espacio de entradas. Así, a las unidades que consistentemente pierden, ocasionalmente se les da la oportunidad de ganar y ajustar sus pesos en la solución de un grupo en particular.

3.7 MÁQUINAS DE BOLTZMANN.

Las máquina de Boltzmann fueron propuestas por Geoffrey E. Hinton y Terrence J. Sejnowski en 1986 quienes combinaron los conceptos trabajados en las redes neuronales simuladas de Hopfield y el método de aprendizaje llamado reforzamiento simulado (el cual se utiliza para encontrar soluciones óptimas globales a problemas combinatorios).

Una máquina de Boltzmann es una red simulada de Hopfield con conexiones sólo simétricas en donde el proceso de aprendizaje se da de manera autoasociativa y sin supervisión, lo que representa un avance considerable en el aprendizaje automático sin supervisión, que se estudiará más adelante.

El procedimiento de aprendizaje para las máquinas de Boltzman, asigna pesos a las conexiones entre las unidades dando un conjunto de estados iniciales y estados finales de entrenamiento.

El aprendizaje de Boltzman consume más tiempo que el de retropropagación por el complejo proceso de reforzamiento, pero tiene algunas ventajas. Por ejemplo, es más fácil usar máquinas de Boltzman para resolver problemas de satisfacción de restricciones que otros métodos, ya que a diferencia de las redes neuronales de retropropagación, las máquinas de Boltzman no hacen una clara división entre "entrada" y "salida". Por ejemplo, en una máquina de Boltzman se pueden tener tres conjuntos importantes de unidades, cualesquiera dos de los cuales pueden tener sus valores "fijos" o "establecidos". Como en la capa de una red neuronal simulada de retropropagación, activaciones en el tercer conjunto de unidades serían determinadas por una relajación en paralelo.

Las máquinas de Boltzman actualizan sus estados binarios, (sólo pueden tomar valores 0 y 1) individuales mediante la siguiente regla: la probabilidad de que cualquier unidad dada se active está dada por:

$$p = \frac{1}{1 + e^{\frac{\Delta E}{T}}}$$

en donde ΔE representa la suma de los valores de las conexiones de entrada con unidades activas y T representa la "temperatura" de la red neuronal simulada.

La actualización estocástica de las unidades es muy parecida a la actualización en las redes neuronales de Hopfield, excepto por el factor de temperatura. A "altas" temperaturas, las unidades se comportan aleatoriamente, mientras que a "bajas" temperaturas, las unidades se comportan como las redes neuronales simuladas de Hopfield. El reforzamiento consiste en mover gradualmente una temperatura alta hacia una temperatura baja. La aleatoriedad añadida por la temperatura ayuda a la red neuronal simulada a escapar de los mínimos locales.

"Si el reforzamiento se lleva a cabo adecuadamente, las máquinas de Boltzman pueden esquivar los mínimos locales y aprender a calcular cualquier función computable de entradas y salidas de tamaño fijo" [RICH]

Una variación determinística del aprendizaje de Boltzman que prometía ser más eficiente fue presentada por Peterson y Anderson en 1987.

CONCLUSIONES

Entre los más importante resultados, en el estudio del aprendizaje automático se encuentra el hecho de que el aprendizaje en si mismo es un proceso de solución de problemas.

Los casos en los que se pueden aplicar algunos de los métodos de aprendizaje presentados en esta tesis son:

. Aprendizaje por consejo

- Estado inicial: consejo de alto nivel
- Estado final: una regla operacional
- Operadores: definiciones desarrolladas o descubiertas, análisis de caso, comparación, etc.

. Aprendizaje por ejemplos

- Estado inicial: colección de ejemplos positivos y negativos
- Estado final: descripción de consejos
- Algoritmos de búsqueda: eliminación de candidatos, inducción de árboles de decisión

. Aprendizaje en solución de problemas

- Estado inicial: trazas de la solución de problemas de ejemplo
- Estado final: nuevas heurísticas para resolver nuevos problemas eficientemente
- Heurísticas de búsqueda: generalización, aprendizaje en base a explicaciones, consideraciones de utilidad

. Descubrimiento

- Estado inicial: algún medio
- Estado final: desconocido
- Algoritmos de búsqueda: interés, analogía, etc.

El sueño de los sistemas de IA es una computadora que aprenda. Si la clave del comportamiento inteligente de las computadoras es tener una gran cantidad de conocimiento, una esperanza para hacer a un lado la tarea de proporcionar directamente ese conocimiento a las computadoras es dejar que ellas mismas lo adquieran independientemente, como lo hacen las personas, aprendiendo. No se tienen todavía programas que puedan ampliar su base de conocimientos indefinidamente, pero se han logrado algunos avances.

En los actuales programas de aprendizaje se observa que cuanto más conocimiento tiene el programa para empezar a trabajar más puede aprender. Esto es bueno en el sentido de que corrobora el poder del conocimiento.

Un programa de aprendizaje necesita adquirir nuevo conocimiento y nuevas habilidades de solución de problemas, pero el almacenamiento, organización y manejo del conocimiento en la solución de problemas son temas aún en estudio. Si no se entiende la naturaleza de los conceptos que se quieren aprender, el aprendizaje es difícil, por lo que no es sorprendente que los programas de aprendizaje que trabajan bastante bien sean los de áreas bien delimitadas como planeación, y no en áreas menos delimitadas como es comprensión de lenguaje natural.

BIBLIOGRAFIA

- Galindo et al, "Inteligencia artificial en México", U.T.M., 1992

- Gardner M.D. Ernest et al, "Anatomía", Salvat Editores S.A., 1971

- García Garrido, Luciano, "Modelos Conexionistas y Procesamiento Simbólico

- Graubard, Stephen R., "El nuevo debate sobre la inteligencia artificial", Editorial Gedisa, 1993

- Guyton, Arthur C. , "Tratado de Fisiología Médica", Nueva Editorial Interamericana S.A. de C.V., 1971

- Hammerly, Marcelo A., "Encliclopedia Médica Moderna Tomo 3", Ediciones Interamericanas, 1972

- Hanton J.P., Haton, M.Ch.,"La inteligencia artificial una aproximación", Paidos, 1989

- Houssay Bernardo, A., "Fisiología Humana", Librería "El Ateneo" Editorial Conexionista", Segunda Escuela Internacional de invierno en Temas Selectos De La Computación.

- Machine Learning, "An artificial intelligense aprroach", Tioga Publishing Company Palo Alto California, 1983

- Mussen et al, "Introduccción a la psicología", Compañía Editorial Continental, S.A., México, 1981

- Pitrat, Jaques,"El nacimiento de la inteligencia artificial", Mundo científico, No. 5

- Quesada Castillo Rocío, "Guía para evaluar el aprendizaje teórico y práctico", Limusa, 1991

- Rich, Elaine et al, "Artificial Intelligence Second Edition", International Edition 1991, McGraw Hill

- Shapiro, Stuart C., Editor in Chief, "Encyclopedia of artificial intelligence Vol. 1", Editorial Board , 1990

- Staton,Thomas F. ,"Como estudiar", Editorial Trillas, 1967

- Tucker Allen, B., "Lenguajes De Programación Segunda Edición", Mc Graw Hill, 1988.

- Winston, Patrick Henry,"Inteligencia Artificial tercera edición", Massachusetts Institute Of Technology, Addison Wesley Iberoamericana, 1992