INTELIGENCIA ARTIFICIAL

RESUMEN

INTRODUCCIÓN A LA INTELIGENCIA ARTIFICIAL

1.1 ¿Qué es la Inteligencia Artificial?

La **Inteligencia Artificial (IA)** tiene por objeto el estudio del comportamiento inteligente en las máquinas. El comportamiento inteligente supone percibir, razonar, aprender, comunicarse y actuar en entornos complejos.

- Una de las metas a largo plazo de la IA es el desarrollo de máquinas que puedan hacer estas cosas igual, o quizá mejor, que los humanos.
- Otra meta de la IA es llegar a comprender ese tipo de comportamiento, sea en las máquinas, en los humanos o en otros animales.

La cuestión básica de la IA es **¿pueden pensar las máquinas?** Alan Turing expresó esta misma cuestión, formulada en términos más adecuados para su comprobación empírica, en lo que se conoce como Test de Turing.

1.1.1 Pueden

¿Queremos decir que las máquinas pueden pensar ya ahora, o que algún día podrán pensar? ¿Queremos decir que las máquinas podrían ser capaces de pensar, o lo que perseguimos es una implementación real de una máquina pensante?

1.1.2 Máquinas

Aunque una máquina hecha de proteínas puede pensar, quizá una hecha de silicio no sería capaz de hacerlo. Un filósofo, John Searle, cree que la materia de la que estamos hechos es fundamental para la inteligencia. Para él, el pensamiento sólo puede ocurrir en máquinas muy especiales *–las máquinas vivientes hechas de proteínas*.

La hipótesis del sistema físico de símbolos de Newell y Simon está en oposición directa a las creencias de Searle. Esta hipótesis establece que *un sistema físico de símbolos dispone de los medios necesarios y suficientes para desarrollar una actividad general inteligente.* Un aspecto importante de esta hipótesis es que no importa de qué esté hecho el sistema físico de símbolos: decimos que la hipótesis de Newell y Simon es neutral respecto al sustrato.

Una entidad inteligente podría estar hecha de proteínas, de relés mecánicos, de transistores o de cualquier otra cosa, con tal que sea capaz de procesar símbolos.

Otros pensadores creen que no es importante si las máquinas están hechas de silicio o de proteínas; piensan que el comportamiento inteligente es resultado de lo que ellos llaman *procesamiento subsimbólico*, es decir, procesamiento de señales, no de símbolos.

Otras hipótesis sobre el tipo de máquinas que podrían pensar son:

- Procesamiento de información en paralelo.
- Uso de lógica difusa.
- Uso de neuronas artificiales realistas.

1.1.3 Pensar

En lugar de definir esta palabra, Turing propuso un test, mediante el cual pudiera decidirse si una máquina en particular es o no inteligente:

En el test participan un hombre (A), una mujer (B) y un interrogador (C). El interrogador permanece en una sala, separado de los otros dos, pero pudiendo comunicarse con ellos mediante un teletipo. El objetivo del test para el interrogador es determinar cuál de los otros dos es el hombre y cuál la mujer. Para ello, el

interrogador puede plantear preguntas a A y B. El objetivo de A en el juego es intentar que C haga una identificación errónea. El objetivo de B es ayudar al interrogador.

1.2 Aproximaciones a la IA

1.2.1 Basadas en procesamiento de símbolos

Se sustentan sobre la hipótesis del **sistema físico de símbolos de Newell y Simon**. Un miembro destacado de esta familia de aproximaciones es el que se basa en la aplicación de operaciones lógicas sobre bases de conocimiento declarativo. Este estilo de IA representa el "conocimiento" sobre un problema del dominio mediante sentencias declarativas, a menudo basadas en sentencias de la lógica de predicados, o equivalentes. Para deducir consecuencias a partir de este conocimiento, se aplican técnicas de inferencia lógica.

Encontramos tres niveles:

- Nivel del conocimiento: se especifica el conocimiento necesario para los objetivos de la máquina.
- Nivel simbólico: se representa este conocimiento mediante estructuras simbólicas (por ejemplo, listas escritas en LISP), y se especifican operaciones sobre dichas estructuras.
- Nivel de implementación: se implementan las operaciones de procesamiento de símbolos.

Muchas de estas aproximaciones utilizan el diseño **descendente**, comenzando por el nivel del conocimiento, continuando hacia abajo.

1.2.2 Aproximaciones subsimbólicas

Siguen usualmente el diseño **ascendente**, comenzando por el nivel más bajo, y continuando hacia arriba.

En los niveles más bajos, el concepto de señal es más apropiado que el concepto de símbolo. Según defensores de estas aproximaciones, habrá que seguir los pasos evolutivos de la inteligencia humana para conseguir máquinas inteligentes.

Las *redes neuronales* son un ejemplo de máquinas basadas en aproximaciones subsimbólicas. También se incluyen procesos que simulan ciertos aspectos de la *evolución biológica*: cruzamiento, mutación y reproducción de los organismos mejor adaptados. Otras aproximaciones ascendentes se basan en la *teoría del control* y en el *análisis de sistemas dinámicos*.

A medio camino entre las aproximaciones ascendentes y descendentes se encuentran aquellas basadas en **autómatas situados**. Se propuso un lenguaje de programación para especificar, en un alto nivel de abstracción, el comportamiento de un agente, así como un compilador para generar la circuitería necesaria.

1.3 Breve historia de la IA

El término "Inteligencia Artificial" fue acuñado por McCarthy, al usarlo como título de una conferencia en 1956.

El primer paso hacia la Inteligencia Artificial fue dado por Aristóteles cuando comenzó a codificar ciertos estilos de razonamiento deductivo que él llamó *silogismos*. El siguiente gran paso fue dado por George Boole, quien desarrolló los fundamentos de la lógica proposicional. Luego, Gottlieb Frege creó lo que hoy se conoce como cálculo de predicados.

Los lógicos del siglo XX formalizaron y clarificaron lo que puede ser hecho y lo que no puede ser hecho mediante sistemas lógicos y computacionales.

Más tarde, en los años ’60 y primeros de los `70, el esfuerzo se centró en explorar diversas representaciones de problemas, técnicas de búsqueda y heurísticas generales aplicables en programas de

ordenador. Algunos resultados de estos esfuerzos fueron el *Solucionador General de Problemas* y programas que resolvían problemas de integración simbólica y de álgebra, y puzles.

Luego, en los últimos años de los `70 y primeros de los ’80, se desarrollaron programas más realistas, que contenían conocimiento que representaba el comportamiento de expertos humanos. También fueron desarrollados métodos para representación de conocimiento para dominios específicos.

1.4 Mundo especial para agentes con IA

La investigación en IA ha conducido al desarrollo de numerosas técnicas e ideas referidas a la automatización de la inteligencia. Estas técnicas e ideas serán descriptas en el contexto de una seria de **agentes**. Hay muchos tipos de agentes y entornos que podemos considerar.

En lugar de considerar agentes del *mundo real*, lo que haremos será introducir una serie de agentes en un entorno ficticio al que llamaremos **mundo espacial cuadriculado**.

El mundo espacial cuadriculado es un espacio tridimensional demarcado por una matriz bidimensional de celdas a la que llamaremos *suelo*. Cada celda puede contener objetos, y pueden existir *paredes* que delimitan conjuntos de celdas. Los agentes están confinados al suelo y pueden moverse celda a celda. Los objetos pueden estar en el suelo o encima de otros objetos, apilados sobre el suelo.

Muchas de las técnicas de IA que describiremos se aplican a mundos discretos, y requerirían un cierto procesamiento subsimbólico para operar en mundos continuos.

1.5 Sistemas que resuelven problemas de la IA

Los problemas de la Inteligencia Artificial

Los primeros problemas en los que se hizo hincapié fueron los relacionados con **juegos** y **demostración de teoremas**.

Otra primera incursión dentro de la IA se centró en la clase de problemas que aparecen a diario –como cuando decidimos cómo llegar de un lugar a otro-, con frecuencia denominados de **sentido común**.

Conforme avanzaban las investigaciones aparecieron nuevas áreas de investigación. Estas áreas incluyeron la **percepción** (visión y habla), **comprensión del lenguaje natural** y **resolución de problemas en campos especializados**, como diagnósticos médicos y análisis químicos.

También están bajo el dominio de la Inteligencia Artificial problemas que necesitan de adquisición de experiencia, como el **diseño de ingeniería**, **descubrimientos científicos**, **diagnósticos médicos** y **planificación financiera**.

Un área importante donde la IA es próspera es la de los **sistemas expertos**, donde es necesario únicamente un conocimiento experto, sin ayuda del sentido común.

- Tareas formales
    - Juegos
    - Matemática
- Tareas de la vida diaria:
    - Percepción
        - Visión
        - Habla
    - Lenguaje natural
        - Comprensión
        - Generación
        - Traducción

    - Sentido común

- Tareas de los expertos
    - Ingeniería
        - Diseño
        - Detección de fallas
        - Planificación de manufactura
    - Análisis científico
    - Diagnosis médica
    - Análisis financiero

Las suposiciones subyacentes

Se define un **sistema de símbolos físicos** como sigue:

"Un sistema de símbolos físicos consiste en un conjunto de entidades, llamadas símbolos, que pueden funcionar como componentes de otro tipo de entidad, llamada expresión o estructura de símbolos. Además, el sistema contiene una colección de procesos que operan sobre las expresiones para producir otras expresiones: procesos de creación, modificación, reproducción y destrucción."

La **hipótesis del sistema de símbolos físicos** queda entonces enunciada así:

"Un sistema de símbolos físicos posee los medios necesarios y suficientes para realizar una acción inteligente genérica."

¿Qué es una técnica de IA?

Durante las primeras investigaciones en IA, el resultado obtenido más importante fue que la **inteligencia necesita conocimiento**. Sin embargo, el conocimiento posee propiedades poco deseables:

- Es voluminoso.
- Es difícil de caracterizarlo con exactitud.
- Cambia constantemente.
- Su diferencia con los datos radica en que se organiza de tal forma que se corresponde con la forma en que va a ser usado.

Concluimos que una **técnica de IA** es un método que utiliza conocimiento representado de tal forma que:

- El conocimiento representa generalizaciones, es decir, se agrupan las situaciones que comparten propiedades importantes.
- Debe ser comprendido por las personas que lo proporcionan.
- Puede modificarse fácilmente para corregir errores y reflejar los cambios en el mundo.
- Puede usarse en gran cantidad de situaciones aun cuando no sea totalmente preciso o completo.
- Puede usarse para ayudar a superar su propio volumen.

Por último, se ponen de manifiesto tres importantes técnicas de IA:

- **Búsqueda**: proporciona una forma de resolver los problemas en los que no se dispone de un método más directo.
- **Uso del conocimiento**: proporciona una forma de resolver problemas complejos explotando las estructuras de los objetos involucrados.
- **Abstracción**: proporciona una forma de separar aspectos y variaciones importantes de aquellos otros sin importancia.

1.5.1 Acciones que debe llevar a cabo el sistema

Para construir un sistema que resuelva un problema específico, es necesario realizar estas cuatro acciones:

- **Definir el problema con precisión**. Deben incluirse especificaciones tanto sobre la o las situaciones iniciales como sobre la o las situaciones finales que se aceptarían como soluciones del problema.
- **Analizar el problema**. Algunas características de importancia pueden tener efecto sobre la elección de determinada técnica para resolver el problema.
- **Aislar y representar el conocimiento**.
- **Elegir la mejor técnica** aplicable al problema, y aplicarla.

1.5.2 Definición del problema mediante una búsqueda en espacio de estados

La búsqueda permite **definir formalmente un problema**, a través de la necesidad de convertir alguna situación dada en una situación deseada, mediante un conjunto de operaciones permitidas. Es un proceso de gran importancia en la **resolución de problemas difíciles para los que no se dispone de técnicas más directas**.

Para poder producir una descripción formal de un problema, debe hacerse lo siguiente:

- **Definir un espacio de estados** que contenga todas las configuraciones posibles de los objetos más relevantes. Es posible definir este espacio sin tener que hacer una enumeración de todos los estados posibles.
- Identificar uno o más estados en los que comience el proceso de resolución. Éstos son los **estados iniciales**.
- Identificar uno o más estados en los que finalice el proceso de resolución, que pudieran ser soluciones aceptables del problema. Éstos son los **estados objetivo**.
- Especificar un conjunto de reglas que describan las acciones disponibles. Éstos son los **operadores**.

1.5.3 Sistemas de producción

Los sistemas de producción proporcionan estructuras para facilitar la descripción y el desarrollo de procesos de búsqueda.

Un sistema de producción consiste en:

- **Un conjunto de reglas** compuestas por una parte izquierda (patrón), y una parte derecha, que describe la operación que se lleva a cabo si se aplica la regla.
- **Una o más bases de datos o conocimiento**.
- **Una estrategia de control** que especifique el orden en que las reglas son comparadas con la base de datos, y la forma de resolver los conflictos que surjan cuando varias reglas puedan ser aplicadas a la vez.
- **Un aplicador de reglas**.

1.5.4 Análisis del problema

A fin de poder elegir el método más apropiado para un problema en particular, podemos analizarlo en base a varias dimensiones:

- ¿Puede descomponerse el problema?
- ¿Pueden deshacerse o ignorarse pasos hacia una solución?
    - **Ignorables**: pueden ignorarse pasos dados (por ejemplo: demostración de teoremas). Implementados mediante estructuras de control que nunca vuelven hacia atrás.
    - **Recuperables**: pueden deshacerse los pasos dados (por ejemplo: 8-puzzle). Implementados mediante estructuras de control que utilizan pilas “push-down”.

    - **No recuperables**: no pueden deshacerse los pasos dados (por ejemplo: ajedrez). Implementados mediante estructuras de control enfocados en la toma de decisiones, ya que son irrevocables.
- ¿Es predecible el universo del problema?
    - **Consecuencia cierta**: a través de una planificación puede conocerse exactamente qué ocurrirá.
    - **Consecuencia incierta**: no es posible un proceso de planificación con certeza.
- Una solución adecuada, ¿es absoluta o relativa?
    - **Algún camino**: son resueltos frecuentemente en tiempos razonables mediante alguna heurística.
    - **Mejor camino**: son más difíciles de computar.
- La solución, ¿es un estado o una ruta?
    - **Solución estado**: la solución es un estado.
    - **Solución camino**: la solución es una ruta hacia algún estado.
- ¿Cuál es el papel del conocimiento?
    - Conocimiento necesario **para acotar búsquedas**, por ejemplo, el ajedrez.
    - Conocimiento necesario **para reconocer una solución**, por ejemplo, comprensión de artículos periodísticos.
- ¿Necesita la computadora interactuar con una persona?
    - **Solitarios**: se da a la computadora una descripción del problema, y ésta proporciona una respuesta sin ningún tipo de comunicación y sin explicación de su razonamiento.
    - **Conversacionales**: existe una comunicación entre el hombre y la computadora, bien para proporcionar una ayuda adicional a la máquina, o para que la computadora brinde información al usuario.

1.5.5 Características de los sistemas de producción

Un sistema de producción **monótono** es aquel en el que la aplicación de una regla nunca prevé la posterior aplicación de otra regla que podría haberse aplicado cuando se seleccionó la primera.

Un sistema de producción **no monótono** es aquel en el que lo anterior no es cierto.

Un sistema de producción **parcialmente conmutativo** es aquel que tiene la propiedad de que si una determinada aplicación de una secuencia de reglas transforma el estado x en el estado y, entonces alguna permutación permitida de estas reglas también transforma el estado x en el estado y.

Un sistema de producción **conmutativo** es aquel que es monótono y parcialmente conmutativo a la vez.

Formalmente, no existe ninguna relación entre tipos de problemas y tipos de sistemas de producción, debido a que todos los problemas pueden resolverse utilizando todos los tipos de sistemas de producción.

Sin embargo, en la práctica existen relaciones naturales para representar tal o cual tipo de problema a través de determinado tipo de sistema de producción:

	Monótono	**No monótono**
Parcialmente conmutativo	Adecuados para resolver problemas ignorables. Son útiles para problemas que incluyan procesos de creación y no de modificación (por ejemplo, demostración de teoremas).	Adecuados para resolver problemas en los que se realizan cambios pero estos son reversibles, y en los que el orden de las operaciones no es crítico (por ejemplo, navegación de robots).
No parcialmente conmutativo	Adecuados para problemas en los que se producen cambios irreversibles, donde el orden de las operaciones es importante (por ejemplo, síntesis química para monótonos, y Bridge para no monótonos).	

BÚSQUEDA Y PLANIFICACIÓN

2.1 Técnicas de búsqueda a ciegas

Vimos que para resolver un problema, puede definirse un espacio de estados del problema (incluyendo los estados iniciales y finales), y un conjunto de operadores para trasladarse a través del espacio. El problema se reduce entonces a buscar una ruta a través del espacio que una un estado inicial con un estado final.

Estrategia de control

- El primer requisito que debe cumplir una buena estrategia de control es que cause algún cambio. Las estrategias de control que no causan cambios de estado nunca alcanzan la solución.
- El segundo requisito que debe cumplir una buena estrategia de control es que sea sistemática. Las estrategias de control que no son sistemáticas pueden llegar a utilizar secuencias de operadores no apropiadas varias veces hasta encontrar la solución.

2.1.1 Explosión combinatoria

Cada nodo añadido al espacio de búsqueda añadirá más de un camino. Debido al rápido crecimiento del número de posibilidades, sólo los problemas más simples se dirigen a búsquedas exhaustivas.

Teóricamente, una búsqueda exhaustiva funcionará siempre, aunque no es práctica por el alto consumo de tiempo y/o recursos.

2.1.2 Ramificación y acotación

Este método, también llamado branch and bound, comienza generando rutas completas, manteniéndose la ruta más corta encontrada hasta el momento. Deja de explorar una ruta tan pronto como su distancia total, hasta ese momento, sea mayor que la que se ha marcado como la más corta. Esta técnica garantiza encontrar la ruta más cota.

Desgraciadamente, aunque este algoritmo es más eficiente que el anterior, todavía se necesita un tiempo exponencial. La cantidad de tiempo necesario para encontrar la solución depende del orden en que se exploren las rutas.

2.1.3 Búsqueda primero en anchura

En este método se evalúa cada nodo del mismo nivel antes de proceder al siguiente nivel más profundo. La búsqueda primero en anchura garantiza que se encontrará una solución, si existe, porque eventualmente degenerará en una búsqueda exhaustiva.

2.1.4 Búsqueda primero en profundidad

Se explora cada camino posible hacia el objetivo, hasta su conclusión, antes de intentar otro camino.

En este tipo de recorrido, se va por la izquierda (derecha) hasta que o bien se alcanza un nodo terminal o bien se encuentra el objetivo. Si alcanza un nodo terminal, retrocede un nivel, va a la derecha (izquierda) y luego a la izquierda (derecha) hasta encontrar el objetivo o nodo terminal. Terminará cuando encuentre el objetivo o al haber examinado el último nodo del espacio de búsqueda.

Terminar la búsqueda por una ruta tiene sentido cuando se llega a un callejón sin salida, se produce un estado ya alcanzado o la ruta se alarga más de lo especificado según algún criterio. Si esto ocurre, se produce un backtracking, donde se revisita el estado más recientemente creado desde el que sea posible algún movimiento alternativo. Este tipo de backtracking se denomina cronológico debido a que el orden en el que se deshacen los pasos depende únicamente de la secuencia temporal en que se hicieron originalmente esos pasos.

Algunas ventajas y desventajas son:

	Búsqueda primero en anchura	**Búsqueda primero en profundidad**
Callejones sin salida	No queda atrapada explorando callejones sin salida.	Puede seguir una ruta infructuosa durante mucho tiempo, quizás para siempre, antes de acabar en un estado sin sucesores.
Obtención de la solución	Si existe, se garantiza que se logre encontrarla. Si hay varias, se encuentra la que requiere el mínimo número de pasos, ya que no se explora una ruta larga hasta haber explorado todas las rutas más cortas que ella.	Es posible encontrar una solución larga, cuando puede existir otra mucho más corta.
Uso de memoria	Debe almacenarse todo el árbol que haya sido generado hasta ese momento.	Necesita menos memoria ya que sólo se almacenan los nodos del camino que se sigue en ese instante.
Revisión del árbol	Deben revisarse todas las partes del árbol de nivel n antes de comenzar con los nodos de nivel n+1.	Con suerte, se puede encontrar una solución sin tener que revisar gran parte del espacio de estados.

2.2 Técnicas de búsqueda heurística

Con el fin de resolver problemas complejos con eficiencia, con frecuencia es necesario comprometer los requisitos de movilidad y sistematicidad, y construir una estructura de control que no garantice encontrar la mejor respuesta, pero que casi siempre encuentre una buena solución. De esta forma surge la heurística. **Una heurística es una técnica que aumente la eficiencia de un proceso de búsqueda, posiblemente sacrificando demandas de completitud**.

Existen heurísticas de propósito general que son adecuadas para una amplia variedad de dominios de problemas. Además, es posible construir heurísticas de propósito especial que exploten el conocimiento específico del dominio en estudio. Por ejemplo, la heurística del vecino más próximo es un caso de heurística de propósito general, consistente en elegir en cada paso la alternativa localmente superior.

Aplicadas a problemas específicos, la eficacia de las técnicas de búsqueda heurística depende en gran medida de la forma en que exploten el conocimiento del dominio particular, ya que, por sí solas, no son capaces de salvar la explosión combinatoria. Por esto se las denomina también métodos débiles.

Una clase de algoritmos heurísticos son los **métodos constructivos**, que consisten en ir agregando componentes individuales a la solución hasta obtener una solución factible

Un tipo de estos métodos constructivos son los **algoritmos greedy** (golosos o devoradores). Estos algoritmos van construyendo paso a paso la solución, buscando el máximo beneficio en cada paso.

2.2.1 Generación y prueba

Consiste en:

1. Generar una posible solución.
2. Verificar si la posible solución es realmente una solución, comparándola con el objetivo final o con objetivos aceptables.
3. Si se ha encontrado la solución, terminar. Sino, volver al paso 1.

Si las soluciones son generadas de forma sistemática y la solución existe, este procedimiento es capaz de encontrarla en algún momento, aunque puede demorar demasiado. Este algoritmo es un procedimiento de

búsqueda primero en profundidad, ya que las soluciones completas deben generarse antes de que se comprueben. Dicho de otra forma, es simplemente una búsqueda exhaustiva.

La generación de las soluciones puede hacerse de forma aleatoria, pero esto no garantiza que se pueda encontrar alguna vez la solución (esta forma de trabajo es conocida como **algoritmo del Museo Británico**). Por último, existe una variante llamada generación y prueba heurística.

2.2.2 Escalada o remonte de colinas

Es una variante del procedimiento de generación y prueba. En un procedimiento de generación y prueba puro, la función de prueba responde sólo sí o no. Pero si la función de prueba es ampliada mediante una función heurística que proporcione una estimación de lo cerca que se encuentra un estado del estado objetivo, el procedimiento de generación puede usar esta información para ayudar a decidir por cuál dirección debe moverse en el espacio de búsqueda.

Escalada simple

En cada paso, se genera un nuevo estado: si es un estado objetivo, se finaliza la búsqueda; si no es un estado objetivo, pero es mejor que el estado actual, se lo convierte en el nuevo estado actual; si no es mejor que el estado actual, se continúa.

La principal diferencia que existe entre este algoritmo y el de generación y prueba radica en el uso de una función de evaluación como una forma de introducir conocimiento específico de la tarea. **La utilización de este conocimiento específico es lo que hace a éste y otros métodos, métodos de búsqueda heurística**, dándoles la capacidad de resolver algunos problemas que, de otra forma, serían inabordables.

Escalada por la máxima pendiente

Una variación del método de escalada simple consiste en considerar todos los posibles movimientos a partir del estado actual y elegir el mejor de ellos como nuevo estado. En el método anterior, el primer estado que parezca mejor que el actual es seleccionado como el actual.

Tanto la escalada simple como la de máxima pendiente pueden no encontrar una solución. Cualquiera de los dos algoritmos puede acabar sin encontrar un estado objetivo, y en cambio encontrar un estado del que no sea posible generar nuevos estados mejores. Esto ocurre si el procedimiento se topa con:

- **Máximo local**: es un estado que es mejor que todos sus vecinos, pero que no es mejor que otros estados de otros lugares. En un máximo local, todos los movimientos producen estados peores.
- **Meseta**: es un área plana del espacio de búsqueda en la que un conjunto de estados vecinos posee el mismo valor. No es posible determinar la mejor dirección a la que moverse haciendo comparaciones locales.
- **Cresta**: es un tipo especial de máximo local. Es un área del espacio de búsqueda más alta que las áreas circundantes, y que además posee en ella misma una inclinación

Existen algunas formas de evitar estos problemas, aunque sin garantías:

- **Volver atrás hacia algún modo anterior e intentar seguir un camino diferente**. Es especialmente adecuado para máximos locales.
- **Realizar un gran salto en alguna dirección** para intentar buscar en una nueva parte del espacio de búsqueda. Es especialmente adecuado para mesetas.
- **Aplicar dos o más reglas antes de realizar la evaluación.** Esto se corresponde con movimientos en varias direcciones a la vez. Es especialmente adecuado para crestas.

2.2.3 Búsqueda el primero mejor

Combina las ventajas de la búsqueda primero en anchura y la búsqueda primero en profundidad.

En cada paso, se selecciona el nodo más prometedor que se haya generado hasta ese momento, por ejemplo, a través de una función heurística. A continuación se expande el nodo elegido, aplicando las reglas para generar a sus sucesores. Si alguno de ellos es una solución, el proceso termina. Si no es así, estos nuevos nodos se añaden a la lista de nodos que se han generado hasta el momento. De nuevo, se selecciona el más prometedor de ellos y el proceso continúa de igual forma. La búsqueda puede volver a una rama una vez que los nodos de otras ramas sean lo suficientemente malos.

Este método es similar al de escalada por máxima pendiente, excepto en que a) en la escalada, una vez que se selecciona un movimiento, todos los demás se abandonan, y b) en la escalada, el proceso se detiene si no se encuentra un estado sucesos mejor que el estado actual.

Para implementar este método sobre un grafo se necesitan dos listas de nodos:

- **Abiertos**: nodos que se han generado, pero que aún no se han generado sus sucesores.
- **Cerrados**: nodos que ya se han examinado.

Por último, definimos como f' a la función que representa una estimación del costo necesario para alcanzar un estado objetivo a través del camino que se ha seguido.

Transformación del espacio de configuraciones

Consiste en redefinir el problema en términos de otra representación más simple, resolver el problema en dicha representación y reescribir la solución en términos de la representación original.

2.2.4 Reducción de problemas

Consiste en convertir metas difíciles en una o más submetas más fáciles de lograr. Cada submeta, a su vez, puede dividirse más finamente en una o más submetas de nivel inferior.

La idea clave de este método es la exploración de un árbol de metas. Un **árbol de metas** es un árbol en el que los nodos representan metas y los hijos de cada nodo corresponden a submetas inmediatas.

Las metas que se satisfacen sólo cuando todas las submetas inmediatas que dan satisfechas se llaman **metas Y**. Las metas que se satisfacen cuando cualquiera de sus submetas quedan satisfechas se llaman **metas O**. Finalmente, algunas metas se satisfacen directamente, sin hacer referencia a ninguna submeta, y son llamadas **metas hoja**.

Los árboles de metas permiten responder preguntas sobre cómo y por qué se tomaron las acciones que se tomaron: la pregunta ¿cómo? es respondida a través de las submetas implicada, y la pregunta ¿por qué? es respondida a través de la supermeta implicada.

2.2.5 Verificación de restricciones

El objetivo de la verificación de restricciones consiste en **descubrir algún estado del problema que satisfaga un conjunto dado de restricciones**. Éste es un procedimiento de búsqueda en un espacio de restricciones. El estado inicial contiene las restricciones que se dan originalmente, y los estados objetivo son aquellos que satisfacen las restricciones "suficientemente", donde "suficientemente" debe definirse para cada problema particular.

El procedimiento consiste en propagar las restricciones tan lejos como sea posible. Entonces, si todavía no hay solución, se hace una suposición sobre algo y se añade como una nueva restricción. Entonces la propagación continúa con esta nueva restricción. Si en algún momento se encuentra una contradicción, puede usarse una vuelta atrás para reintentar con una suposición diferente.

La propagación puede terminar por: hallarse una contradicción, donde si dicha contradicción corresponde a las restricciones planteadas en la especificación del problema, no existe solución; o porque no puedan hacerse más cambios en base al conocimiento actual que se posea.

Por último, existen dos tipos de restricciones: posibles valores para un objeto y relaciones entre objetos.

2.2.6 Análisis de medios y fines

Está centrado en **la detección de diferencias entre el estado actual y el estado objetivo**.

Una vez que se encontró una diferencia, debe encontrarse un operador que pueda reducirla. Es posible que tal operador no pueda ser aplicado en el estado actual, por lo que se crea el subproblema que consiste en alcanzar un estado en que pueda aplicarse dicho operador. Esto recibe el nombre de realización de subobjetivos para el operador.

Sin embargo, es posible que el operador no produzca el estado objetivo que se desea. En ese caso, se tiene un segundo subproblema que consiste en llegar desde ese estado hasta un objetivo.

Las reglas, que transforman un estado en otro, son representadas mediante un **lado izquierdo**, que contiene las precondiciones de la regla, y un **lado derecho**, que describe los aspectos del problema que cambiarán una vez aplicada la regla. Existe además una estructura llamada **tabla de diferencias** que ordena las reglas según las diferencias que pueden reducir.

2.3 Búsqueda en problemas de juegos

2.3.1 Juegos de dos jugadores

Existen dos razones para que los juegos parezcan un buen dominio de exploración de la inteligencia en una máquina:

- Proporcionan una tarea estructurada en la que es muy fácil medir el éxito o el fracaso.
- No necesitan grandes cantidades de conocimiento, aunque esto no es cierto para juegos más complejos.

Para mejorar la efectividad de un programa resolutor de problemas es necesario:

- Mejorar el procedimiento de generación, de forma que sólo se generen buenos estados.
- Mejorar el procedimiento de prueba, de forma que sólo se exploren en primer lugar los buenos estados.

2.3.2 El procedimiento minimax

La idea consiste en comenzar en la posición actual y usar el generador de movimientos posibles para generar un conjunto de posiciones sucesivas. Entonces se puede aplicar la función de evaluación a esas posiciones y elegir simplemente la mejor. Después de hacer esto, puede llevarse hacia atrás ese valor hasta la posición de partida para representar nuestra evaluación de la misma. La posición de partida es exactamente tan buena para nosotros como la posición generada por el mejor movimiento que podamos hacer a continuación.

La alternancia de maximización y minimización en capas alternas cuando las evaluaciones se envían de regreso a la raíz, se corresponden con las estrategias opuestas que siguen los dos jugadores, y que da el nombre de **minimax**.

2.3.3 El procedimiento alfa-beta

Requiere el mantenimiento de dos valores umbral: uno que representa la cota inferior del valor que puede asignarse a un nodo maximizante, que llamaremos **alfa**, y otro que representa la cota superior del valor que puede asignarse a un nodo minimizante, que llamaremos **beta**.

En los niveles maximizantes podemos excluir un movimiento tan pronto como quede claro que su valor será menor que el umbral actual, mientras que en los niveles minimizantes la búsqueda terminará cuando se descubra un valor mayor que el umbral actual.

2.4 Búsqueda con sistemas evolutivos

2.4.2 Algoritmos genéticos

Son métodos basados en los procesos genéticos de organismos biológicos, codificando cada posible solución a un problema como un **cromosoma**, compuesto por una cadena de bits o caracteres.

Estos cromosomas son llevados a lo largo de varias **generaciones**, en forma similar a las poblaciones naturales, evolucionando de acuerdo a los principios de **selección natural** y **supervivencia** del más apto.

Estos algoritmos trabajan con una **población de individuos**, cada uno representando una posible solución al problema dado. A cada individuo se le asigna una **puntuación de adaptación**, dependiendo de qué tan buena fue la respuesta al problema: a los más adaptados se les da la oportunidad de reproducirse mediante cruzamientos con otros individuos de la población, produciendo descendientes con características de ambos padre; los miembros menos adaptados poseen pocas probabilidades de que sean seleccionados para la reproducción. Este proceso es llamado **selección**.

Si el algoritmo está bien diseñado, la población convergerá a una solución óptima o casi óptima.

Los dos procesos que más contribuyen a la evolución son el **crossover** (intercambio de porciones de cromosomas) y la selección. Por otro lado, existe la **mutación**, que introduce aleatoriedad a las poblaciones.

El proceso, en general, consiste en:

1. Generar aleatoriamente la población inicial.
2. Evaluar la adaptación de todos los individuos de la población.
3. Crear una nueva población efectuando operaciones de selección, crossover y mutación.
4. Eliminar la antigua población.
5. Iterar utilizando la nueva población, hasta que la población converja.

Cada iteración es conocida como **generación**.

Los algoritmos genéticos tienen cuatro diferencias principales respecto a los métodos tradicionales:

- Trabajan con una codificación del conjunto de parámetros, no con estos directamente.
- Buscan simultáneamente la solución en una población de individuos, no en uno solo.
- Utilizan la función objetivo directamente, y no funciones derivadas.
- Utilizan reglas de transición probabilísticas, y no determinísticas.

2.4.3 Poblaciones

Si un problema puede ser representado por un conjunto de parámetros, conocidos como genes, éstos pueden entonces ser unidos para formar cromosomas. A este proceso de lo llama **codificación**.

Se debe utilizar el alfabeto más pequeño posible para representar los genes.

2.4.4 Operadores genéticos

Son las diferentes funciones que se aplican a las poblaciones, las cuales permiten obtener nuevas poblaciones:

- Selección
- Crossover
- Mutación
- Migración
- Otros operadores

Selección

Determina cómo los individuos son elegidos para el apareamiento. Los más usados son el método de la ruleta o ranking.

Crossover

La forma básica de este operador toma dos individuos y corta sus cromosomas en una posición seleccionada al azar, para producir dos segmentos anteriores y dos posteriores. Los posteriores se intercambian para obtener dos cromosomas nuevos. Esto es conocido como **crossover de un punto**, aunque existen otros como crossover de múltiples puntos y crossover cíclico.

Mutación

Es la alteración en forma aleatoria de un individuo de la población. Esto es necesario para que no se produzca una convergencia prematura y para que todos los individuos de la población no tengan probabilidad cero de ser utilizados.

Migración

Es el operador que genera un intercambio de individuos entre subpoblaciones. Es aplicable en el caso de utilizar algoritmos genéticos paralelos para la resolución de un problema.

2.4.5 Función de evaluación

Dado un cromosoma, la función de evaluación consiste en asignar un valor numérico de adaptación, el cual se supone que es proporcional a la "utilidad" del individuo. Debe ser rápida, ya que será aplicada a cada individuo de cada población.

Convergencia

Si el algoritmo ha sido correctamente implementado, la población evolucionará a lo largo de las generaciones, de forma que el promedio general se incrementará hacia el óptimo global.

Convergencia prematura

Los genes de unos pocos individuos relativamente bien adaptados, pero no óptimos, pueden rápidamente dominar la población, causando que se converja a un máximo local. Una vez que sucede esto, la única vía de escape es la mutación.

Finalización lenta

Luego de muchas generaciones, la población habrá convergido pero no habrá localizado el máximo global.

Algoritmo genético canónico

Aquí, los cromosomas son **tiras binarias** de longitud L, por lo que cada gen es un dígito 0 o 1.

Se define además una función **fitness**, establecida como el cociente entre el valor de la función objetivo y el promedio de los valores de la función objetivo.

2.5 Planificación

2.5.2 Componentes de un sistema de planificación

En los sistemas de resolución de problemas basados en técnicas elementales, era necesario llevar a cabo las siguientes funciones:

- Elegir la mejor regla para aplicar a continuación, basándonos en la mejor información heurística disponible. La técnica más profunda para esto consiste en asilar el conjunto de diferencias entre el estado actual y el objetivo deseado para poder identificar aquellas reglas que pueden reducir estas diferencias.
- Aplicar la regla elegida para calcular el nuevo estado del problema que surge de su aplicación.
- Detectar cuándo se ha llegado a una solución. Un sistema de planificación tiene éxito al encontrar una solución cuando encuentra una secuencia de operadores que transforman el estado inicial del problema en un estado objetivo.
- Detectar callejones sin salida, es decir, caminos que nunca pueden conducir a una solución, de forma de que puedan abandonarse.

Además, suele ser importante una quinta función:

- Detectar cuándo se ha encontrado algo muy parecido a una solución correcta, y emplear técnicas especiales para hacer que sea totalmente correcta.

2.5.3 Planificación mediante una pila de objetivos

Una de las primeras técnicas que surgieron para componer objetivos fue el uso de una **pila de objetivos**. Esto fue lo que se usó en el sistema STRIPS. En este método, el resolutor de problemas usa una pila que contiene tanto objetivos como operadores que deben proponerse para satisfacer estos objetivos.

Este tipo de planificación aborda los problemas como objetivos conjuntos, resolviendo por orden los objetivos, uno cada vez. Genera un plan que contiene una secuencia de operadores que resuelven el primer objetivo, seguido por una secuencia completa para el segundo objetivo, y así.

2.5.4 Planificación no lineal mediante fijación de restricciones

Los problemas difíciles provocan interacciones entre los objetivos. Los operadores que se utilizan para resolver un subproblema pueden interferir en la solución de un subproblema anterior.

Así, se necesita un plan entrelazado en el que se trabaje simultáneamente con múltiples subproblemas. Este tipo de plan se denomina **plan no lineal**.

2.5.5 Planificación jerárquica

Para problemas complejos, los resolutores de problemas tienen que generar planes muy extensos. Para poder hacerlo eficientemente, es importante poder eliminar algunos de los detalles del problema hasta que se encuentre una solución que resuelta los principales obstáculos.

La asignación de valores de criticidad apropiados es un aspecto importante para el método de planificación jerárquica. Aquellas precondiciones que no tengan operadores que puedan satisfacerlas son las más críticas.

2.5.6 Sistemas reactivos

Hasta ahora se han descrito procesos de planificación en donde el plan que resuelve una tarea se construye antes de actuar. La idea de los **sistemas reactivos** consiste en evitar planificar totalmente y, en lugar de ello, utilizar la situación observable como pista a la que simplemente reaccionar. Un sistema reactivo debe tener acceso a alguna base de conocimiento que describa las acciones que deben realizarse bajo ciertas circunstancias.

La **principal ventaja** de los sistemas reactivos frente a los planificadores tradicionales es que funcionan de forma robusta en dominios difíciles de modelar con exactitud. Evitan un modelado completo y basan sus acciones directamente en sus percepciones del mundo.

REPRESENTACIÓN DEL CONOCIMIENTO Y RAZONAMIENTO

3.1 El problema de la representación del conocimiento

3.1.1 Correspondencia entre conocimiento y representación del conocimiento

En todas las representaciones del conocimiento estaremos manejando dos tipos de entidades:

- **Hechos**: verdades en un cierto mundo. Es aquello que queremos representar.
- **Representación de los hechos**: éstas son las entidades que realmente seremos capaces de manipular.

Una posible estructuración consiste en clasificar estas entidades en dos niveles:

- **Nivel del conocimiento**: se describen los hechos.
- **Nivel simbólico**: se describen los objetos del nivel del conocimiento en términos de símbolos manipulables por programas.

La relación establecida entre los hechos reales y la representación de los mismos se denomina **correspondencia de la representación**. La misma puede ser hacia **adelante**, donde se establece una correspondencia entre los hechos y sus representaciones, o hacia **atrás**, donde se establece una correspondencia desde las representaciones hacia los hechos. Estas funciones no suelen ser biunívocas.

3.1.2 Propiedades de un buen sistema de representación del conocimiento

Un buen sistema de representación del conocimiento en un dominio particular debe poseer las siguientes propiedades:

- **Suficiencia de la representación**: capacidad de representar todos los tipos de conocimiento necesarios en el dominio.
- **Suficiencia deductiva**: capacidad para manipular las estructuras de la representación con el fin de obtener nuevas estructuras que se corresponden con un nuevo conocimiento.
- **Eficiencia deductiva**: capacidad de incorporar información adicional en las estructuras de conocimiento con el fin que los mecanismos de inferencia puedan seguir las direcciones más prometedoras.
- **Eficiencia en la adquisición**: capacidad de adquirir nueva información con facilidad. El caso más simple es el de una persona insertando directamente el conocimiento en la base de datos.

3.1.3 Modelos de representación del conocimiento

Espectro sintáctico-semántico de la representación

Por un lado, hay **sistemas puramente sintácticos** en los que no se tiene en cuenta el significado del conocimiento que está siendo representado. Por otro lado, existen **sistemas puramente semánticos** en los que sí se tiene en cuenta el significado del conocimiento, y poseen reglas de inferencia complejas.

3.1.3.1 Conocimiento relacional simple

El modo más sencillo de representar los hechos declarativos es mediante un conjunto de relaciones del mismo tipo que las utilizadas en los sistemas de bases de datos. Se dice que esta representación es simple debido a la escasa capacidad deductiva que ofrece.

3.1.3.2 Conocimiento heredable

Una de las formas más útiles de inferencia es la **herencia de propiedades**, donde los elementos de una clase heredan los atributos y los valores de otras clases más generales en los que están incluidos.

Para dar soporte a la herencia de propiedades, los objetos deben ser organizados en **clases**. Las líneas que unen clases representan **atributos**; los nodos recuadrados representan los objetos y valores de los atributos de los objetos. Una estructura así es lo que se denomina **estructura de ranura y relleno**.

Los atributos **es-un**, que se utiliza para indicar que una clase está contenida en otra, e **instancia**, utilizado para indicar pertenencia a una clase, son la base de la herencia de propiedades.

3.1.3.3 Conocimiento deductivo

Utiliza la lógica tradicional para describir las inferencias. Un ejemplo es la lógica de predicados de primer orden.

3.1.3.4 Conocimiento procedimental

Especifica qué hacer cuando se da una determinada situación. La forma habitual de representar este conocimiento es especificarlo como un **código** (en algún lenguaje de programación como PROLOG). La máquina utiliza el conocimiento cuando ejecuta el código para llevar a cabo alguna tarea particular.

3.1.4 Problemas de la representación del conocimiento

- **¿Hay atributos tan genéricos que aparezcan en prácticamente todos los dominios de aplicación**? Hay dos atributos con especial significación: instancia y es-un.
- **¿Se pueden establecer relaciones relevantes entre los atributos de los objetos**? Hay cuatro propiedades que los atributos poseen independientemente del conocimiento específico que codifiquen:
    - **Inversos**: una representación se puede interpretar igualmente como una afirmación acerca de uno u otro objeto, que forman parte de la representación.
    - **Jerarquía es-un**: así como nos referimos a objetos y especializaciones, también podemos hablar de atributos y especializaciones de esos atributos.
    - **Técnicas para el razonamiento acerca de los valores**: a veces, los valores de los atributos se especifican explícitamente durante la creación de la base de conocimiento. Pero generalmente, el sistema debe razonar sobre valores que no ha recibido explícitamente.
    - **Atributos univaluados**: es aquel que puede tomar un único valor.
- **¿A qué nivel se debe representar el conocimiento?** Mientras más bajo sea el nivel de granularidad escogido, más sencillo será razonar en ciertos casos, a costa de un proceso de inferencia más complejo.
- **¿Cómo se deben representar los conjuntos de objetos?** Hay dos razones para disponer de algún medio de representación de conjuntos:
    - Existen propiedades que se verifican para un conjunto de objetos, pero no así en los elementos particulares de los conjuntos;
    - Cuando existe una propiedad que verifican todos los objetos, puede ser más eficiente asociar esa propiedad al conjunto.

  Hay dos formas de definir un conjunto y sus objetos:
    - Enumerar todos los elementos, en lo que se denomina **definición por extensión**.
    - Dar una determinada **regla**, de forma que cuando se evalúa un determinado objeto, dé como resultado verdadero o falso según el objeto pertenezca o no al conjunto. Esto se denomina **definición por comprensión**.
- **Dada una base de conocimiento extensa, ¿cómo acceder a los fragmentos relevantes?**

3.1.5 El problema del marco

Al problema de la representación de los hechos que cambian, así como de aquellos que no lo hacen, se lo conoce como **problema del marco**.

3.2 Lógica simbólica

3.2.1 La lógica y el lenguaje

3.2.1.1 Introducción

El estudio de la lógica es el estudio de los métodos y principios usados al distinguir entre los argumentos correctos (buenos) y los argumentos incorrectos (malos).

El lógico se pregunta siempre ¿se sigue la conclusión alcanzada de las premisas usadas o supuestas? Si las premisas son un fundamento adecuado para aceptar la conclusión, entonces el razonamiento es correcto. De otra manera, es incorrecto. El lógico se interesa en el razonamiento, sin prestar atención a su contenido.

3.2.1.2 Naturaleza del argumento

La **inferencia** es una actividad en la que se afirma una proposición sobre la base de otra u otras proposiciones aceptadas como punto de partida. Al lógico no le interesa el proceso de inferencia, sino las proposiciones iniciales y finales de ese proceso.

Las proposiciones son o **verdaderas o falsas**. Se distinguen de las **oraciones declarativas** en que éstas siempre son parte de un lenguaje, y la misma oración citada en diferentes contextos puede tener diferentes significados.

Un **argumento** puede definirse como un grupo cualquiera de proposiciones o enunciados, de los cuales se afirma que hay uno que se sigue de los demás, considerando éstos como fundamento de la verdad de aquél.

La **conclusión** de un argumento es la proposición afirmada basándose en las otras proposiciones del argumento, y estas otras proposiciones que se afirman como fundamento o razones para la aceptación de la conclusión son las **premisas** de ese argumento.

Toda proposición puede ser premisa o conclusión, dependiendo del contexto.

Se puede distinguir entre argumentos **deductivos** e **inductivos**. Sólo en un argumento deductivo se pretende que sus premisas provean un fundamento absolutamente concluyente para la verdad de sus conclusiones.

Un argumento deductivo es **válido** cuando sus premisas y conclusiones están relacionadas de modo tal que es absolutamente imposible que las premisas sean verdaderas, a menos que la conclusión lo sea también. En los argumentos inductivos sólo se pretende que sus premisas proporcionen algún fundamento para sus conclusiones; ni el término válido ni inválido son aplicables.

3.2.1.3 Verdad y validez

La **verdad** y **falsedad** caracterizan a las proposiciones o enunciados, pero los argumentos no se caracterizan propiamente por ser verdaderos o falsos. La **validez** e **invalidez** son quienes caracterizan a los argumentos.

La validez de un argumento no garantiza la verdad de su conclusión.

La falsedad de la conclusión no garantiza la invalidez de un argumento, pero sí garantiza que, o bien el argumento es inválido, o por lo menos una de sus premisas es falsa.

La verdad de la conclusión se da cuando el argumento es válido y todas sus premisas son verdaderas.

Al lógico sólo le interesa una de estas condiciones. La verdad o falsedad de las premisas es tarea de la investigación científica en general, pues las mismas pueden tratar de cualquier asunto. Pero determinar la validez o invalidez de los argumentos es el campo de la lógica deductiva. Al lógico le interesa la cuestión de la validez aun para argumentos cuyas premisas sean falsas.

3.2.1.4 Lógica simbólica

Los argumentos formulados en cualquier lenguaje natural son de difícil evaluación debido a la vaga y equívoca naturaleza de las palabras en que se expresan, ambigüedad de su construcción y expresiones idiomáticas.

Para evitar las dificultades periféricas ligadas al lenguaje ordinario, se han desarrollado vocabularios técnicos especializados.

3.2.2 Argumentos que contienen enunciados compuestos

3.2.2.1 Enunciados simples y compuestos

Un enunciado **simple** es aquel que no contiene otro enunciado como parte componente. Un enunciado **compuesto** contiene otro enunciado como componente.

Hablaremos de **valor de verdad** de un enunciado, siendo el valor de verdad de un enunciado verdadero, verdadero, y el valor de verdad de un enunciado falso, falso.

Cualquier enunciado compuesto cuyo valor de verdad esté determinado completamente por los valores de verdad de sus enunciados componentes es un **enunciado compuesto función de verdad**.

Representaremos los valores de verdad de un enunciado, que están determinados a partir de los valores de verdad de sus componentes, a través de una **tabla de verdad**.

Algunas formas de combinar enunciados en enunciados compuestos son:

- **Conjunción**: se forma insertando la palabra "y" entre dos enunciados. Dos enunciados así combinados se llaman conyuntos. Si p y q son dos enunciados cualesquiera, su conjunción se escribe p.q.

p	**q**	**p.q**
T	T	T
T	F	F
F	T	F
F	F	F

- **Negación**: introducimos el símbolo "~" para simbolizar la negación de un enunciado.

p	**~p**
T	F
F	T

- **Disyunción**: cuando dos enunciados se combinan disyuntivamente insertando la palabra "o" entre ellos, el enunciado compuesto que resulta es una disyunción, y los dos enunciados combinados se llaman disyuntos. Cuando la palabra "o" denota además conjunción, la llamamos "o" **débil o inclusiva** (puede reforzarse utilizando "y/o"). Cuando la palabra "o" denota sólo disyunción, la llamamos "o" **fuerte o exclusivo** (puede reforzarse utilizando "pero no ambos"). El **significado común parcial** es que al menos un disyunto sea verdadero, y es el significado completo de la disyunción inclusiva. A través del símbolo "v" denotaremos este significado parcial común.

p	q	pvq
T	T	T
T	F	T
F	T	T
F	F	F

La **puntuación** (paréntesis, corchetes y llaves) es necesaria en la lógica simbólica porque los enunciados compuestos son susceptibles de combinaciones para dar lugar a enunciados más complejos.

Como una **disyunción exclusiva** expresa que al menos uno de los disyuntos sea verdadero, pero no ambos, podemos enunciarla como (pvq). ~ (p.q).

Todo enunciado compuesto construido a partir de enunciados simples por aplicación repetida de conectivos de función de verdad, tendrá valores de verdad completamente determinados por los valores de verdad de esos enunciados simples.

3.2.2.2 Enunciados condicionales

Un enunciado **condicional** es aquel que se escribe de la forma "si… entonces…". El enunciado componente situado entre el "si" y el "entonces" es el **antecedente**, y el componente que sigue al "entonces" es el **consecuente**.

Un condicional no afirma que su antecedente sea verdadero o que su consecuente lo sea, sólo afirma que **si su antecedente es verdadero, entonces su consecuente también lo es**.

Podemos encontrar diferentes condicionales: **lógicos** (por ejemplo, "Si a todos los gatos les gusta el hígado y Dina es un gato, entonces a Dina le gusta el hígado); **por definición** (por ejemplo, "Si la figura es un triángulo, entonces tiene tres lados"); y **empíricos** (por ejemplo, "Si el oro se sumerge en agua regia, entonces el oro se disuelve"). El **significado parcial común** para los tres casos es que cualquier condicional de antecedente verdadero y consecuente falso debe ser falso.

Por tanto, **cualquier condicional "si p entonces q" será falso en el caso que la conjunción p.~q sea verdadera**. De esta forma, **para que el condicional "si p entonces q" sea verdadero, ~(p.~q) deberá ser verdadero**.

Introducimos el símbolo "⊃", llamado herradura, para representar el significado común parcial, definiendo "p ⊃ q" como una abreviación de ~ (p.~q).

p	q	~q	p.~q	~(p.~q)	p ⊃ q
T	T	F	F	T	T
T	F	T	T	F	F
F	T	F	F	T	T
F	F	T	F	T	T

3.2.2.3 Formas de argumentos y tablas de verdad

Dos argumentos cualesquiera que tienen la misma forma, **o ambos son válidos o ambos son inválidos**, independientemente de las diferencias de su contenido.

Al discutir las formas de los argumentos, es conveniente usar letras minúsculas de la parte media del alfabeto, "p", "q", "r",…, como **variables sentenciales**, que se definen como letras en lugar de las cuales se pueden sustituir enunciados.

Ahora definimos **forma argumental** como cualquier arreglo de símbolos que contienen variables sentenciales, de modo que al sustituir enunciados por las variables sentenciales, manteniendo siempre el mismo enunciado para la misma variable, el resultado es un argumento.

Cualquier argumento que sea resultado de la sustitución de enunciados en lugar de variables sentenciales de una forma argumental, se dice que tiene esa forma o que es una **instancia de sustitución de esa forma argumental**.

Definimos la **forma específica de un argumento** dado, como aquella forma argumental de la cual resulta el argumento reemplazando cada variable sentencial por un enunciado simple diferente.

Refutación por analogía lógica: si puede mostrarse que la forma específica de un argumento dado tiene una instancia de sustitución con premisas verdaderas y conclusión falsa, entonces el argumento dado es inválido. Los términos "válido" e "inválido" pueden extenderse para aplicarse a formas argumentales. Así, una **forma argumental inválida** es una que tiene, al menos, una instancia de sustitución con premisas verdaderas y conclusión falsa. Para determinar la validez o invalidez de una forma argumental, debemos examinar todas las instancias de sustitución posibles de ellas para ver si alguna tiene premisas verdaderas y conclusión falsa.

Las tres formas argumentales más simples son:

- Modus ponens: Si p entonces q
p
∴ q
- Modus tollens: Si p entonces q
~q
∴ ~p
- Silogismo hipotético: Si p entonces q
Si q entonces r
∴ Si p entonces r

Hay dos formas argumentales inválidas que tienen un parecido superficial con las formas Modus ponens y Modus tollens:

- Falacia de afirmación del consecuente: $p \supset q$
q
∴ p
- Falacia de negación del antecedente: $p \supset q$
~p
∴ ~q

3.2.2.4 Formas sentenciales

Definimos **forma sentencial** como cualquier sucesión de símbolos conteniendo variables sentenciales, de modo que al sustituir enunciados por variables sentenciales, el resultado es un enunciado. Así como distinguimos la forma específica de un argumento dado, también distinguimos la forma específica de un enunciado dado como

la forma sentencial de la que resulta el enunciado poniendo en el lugar de cada variables sentencial un enunciado simple diferente.

Una enunciado es una **verdad formal** cuando es una instancia de sustitución de una forma sentencial cuyas instancias de sustitución son todas verdaderas. Un enunciado es **formalmente falso** cuando es una instancia de sustitución de una forma sentencial cuyas instancias de sustitución son todas falsas.

Una forma sentencial que sólo tiene instancias de sustitución verdaderas es una **tautología**.

Una forma sentencial que sólo tiene instancias de sustitución falsas es una **contradicción**.

Aquellos enunciados y formas sentenciales que no son ni tautológicos ni contradictorios, se dice que son **contingencias**.

Dos enunciados son **materialmente equivalentes** cuando tienen el mismo valor de verdad, y simbolizamos esta equivalencia material insertando el símbolo "≡".

Un enunciado de la forma p ≡ q se llama bicondicional. Dos enunciados se dicen **lógicamente equivalentes** cuando el bicondicional que expresa su equivalencia es una tautología.

Un ejemplo de equivalencia lógica es el **Teorema de De Morgan**, enunciado de la siguiente forma: "la negación de la conjunción (disyunción) de dos enunciados es lógicamente equivalente a la disyunción (conjunción) de sus negaciones".

A todo argumento le corresponde un enunciado condicional cuyo antecedente es la conjunción de las premisas del argumento, y cuyo consecuente es la conclusión del argumento. Ese condicional es una tautología si y sólo si el argumento es válido.

3.2.3 El método de deducción

3.2.3.1 Prueba formal de validez

Que la conclusión se deduce de sus premisas usando argumentos válidos exclusivamente, prueba que el argumento original es válido.

Una **prueba formal de validez** para un argumento se define como una sucesión de enunciados, cada uno de los cuales es una premisa de ese argumento o se sigue de los precedentes por un argumento válido elemental, y tal que el último enunciado de la secuencia es la conclusión del argumento cuya validez se está demostrando.

Presentaremos nueve formas de argumento suficientemente obvias como para ser vistas como formas de argumento válidas elementales, que serán aceptadas como reglas de inferencia:

- Modus ponens: Si p entonces q
  p
  ∴ q

- Modus tollens: Si p entonces q
  ~q
  ∴ ~p

- Silogismo hipotético: Si p entonces q
  Si q entonces r
  ∴ Si p entonces r

- Silogismo disyuntivo: p v q
  ~p
  ∴ q

- Dilema constructivo: (p ⊃ q) . (r ⊃ s)
  p v r
  ∴ q v s

- Dilema destructivo: (p ⊃ q) . (r ⊃ s)
  ~q v ~s
  ∴ ~p v ~r

- Simplificación: p v q
  ∴p

- Conjunción: p
  q
  ∴ p . q

- Adición: p
  ∴ p v q

3.2.3.2 La regla de reemplazo

Si se reemplaza una parte cualquiera de un enunciado compuesto por una expresión que es lógicamente equivalente a la parte reemplazada, el valor de verdad del enunciado que resulta es el mismo que el del enunciado original. **A esto se le llama regla de reemplazo**.

Adoptamos las reglas de reemplazo como principios adicionales de inferencia. Nos permiten inferir de cualquier enunciado el resultado de reemplazar el resultado de reemplazar todo o parte de ese enunciado por otro enunciado lógicamente equivalente a la parte reemplazada.

Algunas sugerencias para construir demostraciones formales de validez son:

- Empezar deduciendo conclusiones de las premisas mediante las reglas de inferencia dadas.
- Eliminar enunciados que ocurren en las premisas, pero no en la conclusión.
- Introducir por adición un enunciado que ocurre en la conclusión, pero no en las premisas.

3.2.3.3 Demostración de la invalidez

No completitud de las reglas

Las reglas de inferencia presentadas son incompletas hasta el momento son incompletas, lo que significa que hay argumentos válidos cuya validez no es demostrable usando tan sólo esas reglas.

La regla de demostración condicional

A continuación presentamos una nueva regla para usarla en el método de deducción: la **regla de demostración condicional**, aplicable solamente a aquellos argumentos cuyas conclusiones son enunciados condicionales:

- A todo argumento le corresponde un enunciado condicional cuyo antecedente es la conjunción de las premisas del argumento, y cuyo consecuente es la conclusión del argumento.

- Como se ha indicado, un argumento es válido si y sólo si su correspondiente condicional es una tautología.

Por tanto, dado cualquier argumento cuya conclusión es un enunciado condicional, una demostración de su validez a través de la regla de demostración condicional se construye suponiendo que el antecedente de su conclusión es una premisa adicional, y luego deduciendo el consecuente de su conclusión por una sucesión de argumentos válidos elementales.

La regla de demostración indirecta

El método de **demostración indirecta**, o demostración por reducción al absurdo, consiste en suponer lo opuesto de lo que se quiere demostrar. Si este supuesto conduce a una contradicción o se reduce a un absurdo, entonces el supuesto debe ser falso, y su negación –lo que se quería demostrar– debe ser verdadero.

3.3 Lógica de predicados

3.3.1 Introducción y concepto de semidecidible

En la **deducción matemática** se puede concluir la verdad de un aserto sin otra cosa que demostrar que es consecuencia de lo ya conocido.

Decimos que la lógica de predicados no es **decidible**, pero sí **semidecidible**, ya que, aunque es posible deducir nuevas sentencias a partir de las antiguas, por desgracia no se dispone de un procedimiento de decisión. Existen procedimientos que permitirán encontrar la prueba de un teorema dado si es que en realidad se trata de un teorema, pero no está garantizado que estos procedimientos se detengan cuando la sentencia no sea un teorema.

3.3.2 Representación de hechos simples en lógica

En primer lugar, la lógica proposicional, como medio de representación del conocimiento, es atractiva debido a su simplicidad y a la disponibilidad de un procedimiento de decisión.

Sin embargo, hay cosas que no son representables utilizando lógica proposicional, por lo que es necesario utilizar la **lógica de predicados** o **de primer orden**. Mediante la lógica de predicados se pueden representar hechos del mundo real como sentencias escritas en forma de fórmulas bien formadas (fbf).

Hay tres cuestiones involucradas en el proceso de conversión de frases en lenguaje natural a sentencias lógicas, y el posterior uso de estas sentencias en la deducción de otras nuevas:

- Muchas frases en lenguaje natural son ambiguas.
- Normalmente hay más de una forma de representar el conocimiento.
- Aun en los casos más sencillos, no es probable que en un conjunto de frases dado esté contenida toda la información necesaria para razonar sobre el tema en cuestión. Normalmente es necesario especificar hechos que, por lo general, son demasiado obvios.

En el caso que haya que decidir cuál de dos sentencias demostrar, por ejemplo dos sentencias opuestas, puede hacerse lo siguiente:

- Razonar a partir de los axiomas y ver qué respuesta se obtiene, lo que se denomina razonar hacia adelante.
- Utilizar heurísticas que permitan decidir cuál es la respuesta más probable, y entonces intentar demostrarla. Después de un tiempo razonable en que no se encuentre una demostración, se intentaría con la otra respuesta.
- Demostrar ambas respuestas simultáneamente y parar cuando una de las dos demostraciones tuviese éxito.
- Demostrar la verdad de una respuesta y la falsedad de la contraria, y utilizar la información obtenida en cada proceso como ayuda en el otro.

3.3.3 La representación de las relaciones instancia y es-un

Para representar las relaciones instancia y es-un, pueden utilizarse predicados **monarios** o **binarios**, donde los primeros poseen un argumento, y los segundos, dos argumentos.

Sin embargo, no es necesario representar explícitamente las relaciones de pertenencia a una clase e inclusión de una clase en otra por medio de los predicados instancia y es-un. Además, normalmente es posible obtener varias representaciones de un hecho; la norma que siempre debe respetarse es el uso consistente de una representación determinada.

3.3.4 Representación de funciones calculables y predicados computables

En muchos casos, representar hechos uno por uno, explícitamente, puede resultar tedioso e incluso imposible. En lugar de ello, es posible calcularlos en el momento necesario, a través de una extensión de la representación, utilizando **predicados computables**.

También puede ser útil disponer de **funciones computables**, como por ejemplo mayor(2+3,1).

Además, se hace uso del concepto de **igualdad**, de modo que sea posible sustituir un objeto por otro que sea igual a él, cuando este cambio resulte de alguna utilidad en el proceso de prueba.

3.3.5 Método de resolución

El **procedimiento de resolución** obtiene demostraciones por refutación. Es decir, para demostrar la validez de una proposición se intenta demostrar que su negación lleva a una contradicción con las proposiciones conocidas.

3.3.6 Conversión a forma clausal

Hay casos en los que el uso de una fórmula bien formada complicada dificultaría la demostración. Si la fórmula fuese más simple, este proceso sería mucho más sencillo. La **forma normalizada conjuntiva** tiene las siguientes propiedades que lo hacen más sencillo:

- Hay menos anidamiento entre las expresiones.
- Los cuantificadores están separados de la fórmula, de modo que no es necesario tenerlos en consideración.

Sin embargo, para que la resolución funcione, es necesario convertir el conjunto de fórmulas bien formadas a un conjunto de cláusulas, donde una **cláusula** se define como una fórmula bien formada en forma normalizada conjuntiva que no contiene ninguna conectiva ^.

Algoritmo de conversión a forma clausal

1. Eliminar las implicaciones →.
2. Reducir el ámbito de las negaciones a un único término.
3. Ligar cada cuantificador a una sola variable.
4. Mover todos los cuantificadores a la izquierda de la fórmula sin cambiar su orden relativo.
5. Eliminar los cuantificadores existenciales ∃.
6. Eliminar el prefijo, ya que todas las variables están cuantificadas universalmente.
7. Crear una cláusula por cada conjunción.
8. Normalizar las variables.

Después de aplicar este algoritmo a un conjunto de fórmulas bien formadas, tendremos un conjunto de cláusulas, cada una de las cuales será una **disyunción de literales**. Estas cláusulas serán las que utilice el procedimiento de resolución para generar demostraciones.

3.3.7 Las bases de la resolución

El **procedimiento de resolución** es un proceso iterativo en el cual, en cada paso, se comparan (resuelven) dos cláusulas llamadas cláusulas padres, produciendo una nueva cláusula que se ha inferido de ellas.

La resolución opera tomando dos cláusulas tales que cada una contenga el mismo literal. **Dicho literal debe estar en forma positiva en una cláusula y en forma negativa en la otra**.

3.3.8 Resolución en lógica proposicional

En primer lugar, presentaremos el procedimiento para lógica proposicional. Posteriormente lo expandiremos para incluir la lógica de predicados.

Algoritmo de resolución de proposiciones

1. Convertir todas las proposiciones a forma clausal.
2. Negar la proposición que quiere demostrarse y convertir el resultado a forma clausal. Añadir la cláusula resultante al conjunto de cláusulas obtenidas en 1.
3. Hasta que se encuentre una contradicción o no se pueda seguir avanzando, hacer:
    a. Seleccionar dos cláusulas. Llamarlas cláusulas padres.
    b. Resolverlas juntas. La cláusula resultante se llamará resolvente.
    c. Si el resolvente es la cláusula vacía, es que se ha encontrado una contradicción. Si no lo es, añadirla al conjunto de cláusulas disponibles.

3.3.9 El algoritmo de unificación

En lógica proposicional es fácil determinar que dos literales no pueden ser ciertos al mismo tiempo. Basta con buscar L y ¬L.

En lógica de predicados el proceso de correspondencia es más complicado puesto que **se deben considerar los argumentos de los predicados**. Existe un procedimiento recursivo, denominado **algoritmo de unificación**, que realiza esta consideración.

La idea básica de la unificación es la siguiente:

- Para unificar dos literales, en primer lugar se comprueba **si los predicados coinciden**. Si es así, seguimos adelantes, sino, no hay forma de unificarlos, sean cuales sean sus argumentos.
- Si los predicados concuerdan, **se van comprobando los argumentos de dos en dos**. Si el primero concuerda, podemos continuar con el segundo, y así sucesivamente. Las reglas de emparejamiento son sencillas:
    - Aquellos predicados o constantes que sean diferentes no pueden emparejarse; aquellos que sean idénticos pueden hacerlo.
    - Una variable se puede emparejar con otra variable, con cualquier constante o con un predicado, con la restricción que el predicado no debe contener ninguna instancia de la variable con la que se está emparejando.

3.3.10 Resolución en lógica de predicados

Ahora disponemos de una forma sencilla para determinar si dos literales son **contradictorios**: lo son si uno de ellos puede unificarse con la negación del otro.

Algoritmo de resolución

1. Convertir todas las sentencias a forma clausal.
2. Negar la sentencia que quiere demostrarse y convertir el resultado a forma clausal. Añadir la cláusula resultante al conjunto de cláusulas obtenidas en 1.

3. Hasta que se encuentre una contradicción, no pueda seguir avanzando o se haya realizado una cantidad de esfuerzo predeterminado, hacer:
    a. Seleccionar dos cláusulas. Llamarlas cláusulas padres.
    b. Resolverlas. El resultado se llamará resolvente.
    c. Si el resolvente es la cláusula vacía, se ha encontrado una contradicción. Si no lo es, añadirla al conjunto de cláusulas sobre las que se está aplicando el procedimiento.

3.4 Representación del conocimiento mediante reglas

3.4.1 Comparación entre conocimiento procedimental y conocimiento declarativo

Una **representación declarativa** es aquella en la que el conocimiento está especificado, pero la manera en la que dicho conocimiento debe ser usado, no viene dado. Para utilizar una representación declarativa se la debe aumentar con un programa que especifique lo que debe hacerse con el conocimiento y de qué forma debe hacerse.

Una **representación procedimental** es aquella en la que la información de control necesaria para utilizar el conocimiento se encuentra embebida en el propio conocimiento. Para utilizar una representación procedimental se la debe aumentar con un intérprete que siga las instrucciones dadas por el conocimiento.

La verdadera diferencia entre ambas representaciones radica en dónde se encuentra la información de control.

3.4.2 Programación lógica

La **programación lógica** es un paradigma de programación en el cual las aserciones lógicas son consideradas como programas.

El sistema de programación lógica más popular es PROLOG. Un **programa PROLOG** está descrito como una serie de aserciones lógicas, cada una de las cuales es una cláusula de Horn. Una **cláusula de Horn** es una cláusula que tiene, como mucho, un literal positivo.

Dado que los programas PROLOG están compuestos sólo por cláusulas de Horn, se obtienen dos importantes consecuencias:

- Se puede escribir un **sencillo y potente intérprete** que resulte eficaz.
- La lógica de los sistemas de cláusulas de Horn es **decidible**, al contrario que la lógica de predicados de primer orden completa.

Los **hechos** únicamente contienen constantes, es decir, representan sentencias acerca de objetos específicos.

Las **reglas** contienen variables, es decir, representan sentencias acerca de las diferentes clases de objetos.

Existen algunas diferencias sintácticas entre la representación lógica estándar y la representación PROLOG:

- En la lógica, las variables están específicamente cuantificadas. En PROLOG, la cuantificación se realiza de un modo implícito.
- En la lógica existen símbolos explícitos para “y” (^) y “o” (v). En PROLOG existe un símbolo explícito para “y” (,), pero no para “o”. En su lugar, la disyunción se expresa mediante una lista de sentencias alternativas.
- En la lógica, las implicaciones de la forma “p implica q” se escriben como p → q. En PROLOG, la misma implicación se escribe de forma inversa, como q :- p.

La **principal diferencia** entre la representación de la lógica y la de PROLOG, es que el intérprete PROLOG fija una estrategia de control, y por tanto, las aserciones en un programa PROLOG definen un camino de búsqueda concreto para así encontrar una respuesta a cualquier pregunta. En contraste, las aserciones lógicas no dicen nada sobre cómo se debe elegir entre todas las respuestas, si es que existe más de una.

La **estrategia de control PROLOG** comienza con una sentencia problema, que es el objetivo a probar, y busca las aserciones que pueden probar el objetivo: considera hechos que prueban el objetivo y también cualquier regla cuya cabeza se empareje con el objetivo. Para decidir cuándo puede aplicar una regla o un hecho al problema, utiliza el procedimiento de unificación estándar. Razonará hacia atrás desde ese objetivo, hasta que encuentre el modo de terminar con las aserciones en el programa. Considera los caminos utilizando la estrategia de **búsqueda primero en profundidad**, así como la **vuelta atrás**.

3.4.3 Diferencia entre razonamiento hacia delante y hacia atrás

En el **razonamiento hacia delante** se comienza a partir de los estados iniciales, construyendo un árbol de secuencias de movimientos, con la configuración inicial en el nodo raíz. Se generará el siguiente nivel del árbol encontrando todas las reglas cuyos lados izquierdos se relacionen con el nodo raíz, y utilizando los lados derechos para crear nuevas configuraciones. Se continuará así hasta que se consiga una configuración que se empareje con el estado objetivo.

En el **razonamiento hacia atrás** se comienza a partir de los estados objetivo, construyendo un árbol de secuencias de movimientos, con la configuración objetivo. Se generará el siguiente nivel del árbol encontrando todas las reglas cuyos lados derechos estén ligados con el nodo raíz. Se utilizará el lado izquierdo de las reglas para generar los nodos en este segundo nivel del árbol. Se continuará hasta que se genere un nodo que se empareje con el estado inicial. Este método de razonamiento también se denomina razonamiento dirigido al objetivo.

3.4.3.1 Sistemas de reglas encadenadas hacia atrás

Los **sistemas de reglas encadenadas hacia atrás**, de los cuales PROLOG es un ejemplo, resultan muy eficaces para la resolución de problemas dirigidos al objetivo.

3.4.3.2 Sistemas de reglas encadenadas hacia adelante

Los **sistemas de reglas encadenadas hacia adelante** son útiles para dirigirse por la información que se va incorporando.

3.5 Razonamiento bajo incertidumbre

Hasta ahora se han descrito técnicas de razonamiento para un modelo del mundo **completo e inalterable**. Sin embargo, en muchos dominios no es posible crear tales modelos. Se han propuesto varios marcos lógicos y métodos computacionales para manipular estos problemas. Veremos dos enfoques:

- **Razonamiento no monótono:** se extienden los axiomas y/o reglas de inferencia para que sea posible razonar con información incompleta.
- **Razonamiento estadístico:** se extiende la representación para permitir algún tipo de medida numérica sobre la certeza.

Los sistemas convencionales de razonamiento, como la lógica de predicados de primer orden, están diseñados para trabajar con información que cumple tres importantes propiedades:

- **La información es completa respecto al dominio de interés**. Es decir, todos los hechos necesarios para resolver el problema, o están presentes en el sistema o pueden ser derivados de ellos mediante reglas convencionales de primer orden.
- **La información es consistente**.

- **La única forma en que puede cambiar la información es que se añadan nuevos hechos**. Si estos nuevos hechos son consistentes con todos los demás hechos que ya se han afirmado, entonces ninguno de los hechos pertenecientes al conjunto que eran ciertos pueden refutarse. Esta propiedad se denomina **monotonía**.

Los sistemas de razonamiento no monótono están diseñados para que puedan resolver problemas en los que quizá no aparezca alguna de estas propiedades.

3.5.1 Razonamiento no monótono

3.5.1.1 Razonamiento por defecto

En el **razonamiento por defecto** se pretende llegar a unas conclusiones basadas en los que es más probable que sea cierto.

A continuación se presentan diferentes enfoques para lograr el razonamiento por defecto.

Lógica no monótona

La **lógica no monótona** es un sistema que permite razonar por omisión, en donde el lenguaje de la lógica de predicados de primer orden se aumenta con un operador modal M, que se lee "es consistente".

Lógica por defecto

La **lógica por defecto** permite llevar a cabo un razonamiento basado en omisiones, en la que se introduce un nuevo tipo de regla de inferencia, del tipo "si A es probable y es consistente asumir B, entonces se concluye C".

Abducción

Supongamos que ∀x: Sarampión(x) → Manchas(x). Esto significa que, si se tiene sarampión, se tienen manchas. Pero podría ser bueno concluir también, a partir de la observación de manchas rojas, que se tiene sarampión. Esto no está permitido por las reglas de la lógica estándar. La derivación de conclusiones de esta forma es otra manera de razonamiento por defecto, y se denomina razonamiento por abducción.

Herencia

El razonamiento no monótono se utiliza con frecuencia en la **herencia de los valores de los atributos** desde la descripción prototipo de una clase hacia las entidades individuales que pertenecen a la clase.

3.5.1.2 Razonamiento minimalista

Se dirá que un modelo es **mínimo** si no existen otros modelos en los cuales sean ciertas menos cosas. La idea de usar modelos mínimos como base para el razonamiento no monótono es que **existen muchas menos sentencias ciertas que falsas**. Por tanto, se asumirá que las únicas sentencias ciertas son aquellas que necesariamente deben ser ciertas para que se mantenga la consistencia de la base de conocimiento.

La suposición de un mundo cerrado

La suposición de un mundo cerrado es una forma de razonamiento minimalista. Dicha suposición dice que **los únicos objetos que satisfacen un predicado P son aquellos que deben hacerlo**. Por ejemplo, podemos asumir que una base de datos sobre personal puede listar todos los empleados de una empresa. Si alguien pregunta si Gómez trabaja para la empresa, se puede responder "no" a no ser que aparezca explícitamente en la lista como empleado.

Circunscripción

En la circunscripción, se añaden nuevos axiomas a la base de conocimiento existente. El fin de estos axiomas es **forzar una interpretación mínima sobre una parte de la base de conocimiento**.

3.5.2 Razonamiento estadístico

En algunas resoluciones de problemas puede resultar adecuado **describir las creencias sobre las que no se tiene certeza, pero en las que existen algunas evidencias que las apoyan**. Podemos dividir estos problemas en dos grupos:

- Problemas en los que se da una **cierta aleatoriedad**, para los cuales se dispone de conocimiento sobre las probabilidades de los distintos resultados.
- Problemas en los que el mundo no es aleatorio, sino que se comporta normalmente **hasta que surge algún tipo de excepción**.

En muchos sistemas de resolución de problemas, un objetivo importante consiste en reunir evidencias sobre la evolución del sistema y modificar su comportamiento sobre la base de las mismas. Para modelas este comportamiento se necesita una teoría estadística de la evidencia. Las **estadísticas bayesianas** constituyen esta teoría.

Sin embargo, la estadística bayesiana es inaplicable por diversos motivos:

- El problema de la adquisición de conocimiento es inabarcable, ya que son necesarias demasiadas probabilidades.
- El espacio necesario para almacenar todas las probabilidades es demasiado grande.
- El tiempo empleado en calcular las probabilidades es demasiado grande.

A pesar de esto, las estadísticas bayesianas proporcionan una base para los sistemas que razonan bajo incertidumbre. En consecuencia, se han desarrollado distintos mecanismos que hacen uso de su potencialidad, pero que a la vez hacen que sea tratable:

- Factores de certeza.
- Redes bayesianas.
- Teoría de Dempster-Shafer.

3.5.2.1 Factores de certeza

Este enfoque surgió en el sistema MYCIN. Aquí, **a cada regla se le asocia un factor de certeza**, que representa una medida sobre la evidencia que existe de que la conclusión sea el consecuente de la regla en el caso de que se describa el antecedente de la misma.

Un factor de certeza se define en términos de dos componentes:

- **MB[h, e]**: una medida (measurement) entre 0 y 1 de la creencia (belief) de que la hipótesis h proporciona la evidencia e.
- **MD[h, e]**: una medida entre 0 y 1 sobre la incredulidad (disbelief) de que la hipótesis h proporciona la evidencia e.

A partir de estas dos medidas, se define el **factor de certeza** como:

$$\mathbf{FC[h, e] = MB[h, e] – MD[h, e]}$$

3.5.2.2 Redes bayesianas

Las **redes bayesianas** constituyen un enfoque alternativo al de los factores de certeza. La idea principal consiste en que, para describir el mundo real, **no es necesario utilizar una tabla de probabilidades enorme** en la que se listen las probabilidades de todas las combinaciones concebibles de sucesos. La mayoría de sucesos son condicionalmente independientes de la mayoría de los demás, por lo que no deben considerarse sus interacciones. Puede entonces usarse una representación más local en donde **se describen grupos de sucesos que interactúan**.

3.5.2.3 Teoría de Dempster-Shafer

Hasta ahora, en las técnicas vistas se consideraban proposiciones individuales y se les asignaba una estimación a cada una de ellas acerca del grado de creencia que se garantizaba.

La **Teoría de Dempster-Shafer** considera conjuntos de proposiciones, y le asigna a cada uno de ellos un intervalo

[Creencia, verosimilitud]

La **creencia** (Bel, de belief) mide la fuerza de la evidencia a favor de un conjunto de proposiciones. Su rango va de cero (que indica evidencia nula) a uno (que indica certeza).

La **verosimilitud** (Pl, de plausibility) se define como Pl(s) = 1 – Bel(¬s), y mide el alcance con que la evidencia a favor de ¬s deja espacio para la creencia en s. Su rango también va de cero a uno.

Sistemas semánticos para representación del conocimiento

Una buena descripción del problema, acompañada por una buena representación, es una puerta abierta para la resolución del mismo; una mala descripción, que utiliza una mala representación, es un obstáculo que impide la resolución del problema.

3.6 Estructuras de ranura y relleno débiles

Estas estructuras fueron introducidas para soportar adecuadamente la herencia.

La herencia monótona se puede desarrollar más eficazmente con estas estructuras que con la lógica pura, y la herencia no monótona puede soportarse muy fácilmente. La razón por la que la herencia se ejecuta de un modo sencillo es que, en los sistemas de ranura y relleno, el conocimiento está estructurado como un conjunto de entidades y todos sus atributos.

En las estructuras de ranura y relleno **débiles** hablamos de "conocimiento pobre". En las estructuras de ranura y relleno **fuertes** se establecen mayores compromisos en relación con el contenido de las representaciones.

Se describirán dos enfoques para este tipo de estructuras: las redes semánticas y los marcos (frames).

3.6.1 Redes semánticas

La idea principal de las redes semánticas es que la información contenida en ellas se representa como un conjunto de nodos conectados unos con otros mediante un conjunto de arcos etiquetados que representan las relaciones entre los nodos.

3.6.2 Marcos (frames)

Un **marco** (frame) es una colección de atributos con valores asociados, que describe alguna entidad del mundo. Cada marco puede representar una clase (es decir, un conjunto) o una instancia (es decir, un elemento del conjunto).

3.7 Estructuras de ranura y relleno fuertes

Las redes semánticas y los marcos implementan **estructuras muy generales** para representar el conocimiento

La dependencia conceptual, los guiones y los CYC implementan poderosas teorías sobre la forma en que los programas de IA pueden representar y utilizar el conocimiento sobre **situaciones comunes**.

3.7.1 Dependencia conceptual

La **dependencia conceptual** (DC) es una teoría sobre la representación del tipo de conocimiento sobre los eventos que normalmente aparecen en las frases del lenguaje natural.

Su objetivo consiste en representar el conocimiento de tal forma que:

- Facilite extraer inferencias de las frases.
- Sea independiente del lenguaje en el que están las frases originalmente.

Así, la representación en DC de una frase no se construye con primitivas que se corresponden con palabras que aparecen en la frase, sino con **primitivas conceptuales** que pueden combinarse para formar el significado de las palabras, independientemente del lenguaje concreto.

3.7.2 Guiones

DC es un mecanismo para representa y razonar sobre eventos. Sin embargo, los eventos rara vez ocurren por separado.

Un **guion** (script) es una estructura que describe una secuencia estereotipada de eventos en un contexto concreto. Los guiones son útiles ya que en el mundo real aparecen patrones en la ocurrencia de los eventos. Dichos patrones surgen debido a las relaciones de causalidad entre los eventos.

3.7.3 CYC

CYC es un proyecto de una gran base de conocimiento cuyo propósito es el de capturar el conocimiento humano de sentido común.

INGENIERÍA DEL CONOCIMIENTO

4.1 PROLOG (Programación lógica)

PROLOG (PROgramming LOGic) es un lenguaje de programación que se utiliza para resolver problemas en los que intervienen objetos y relaciones entre ellos. Actualmente es el principal **entorno de programación para IA**.

4.1.2 ¿Para qué sirve PROLOG?

PROLOG está diseñado para manejar **problemas lógicos**, es decir, problemas en los que se necesita tomar decisiones en forma ordenada, e intenta hacer que la computadora "razone" la forma de encontrar una solución.

4.1.3 Lenguaje procedural vs. lenguaje declarativo

Los lenguajes **procedurales** permiten al programador decirle a la computadora lo que tiene que hacer, paso a paso, procedimiento por procedimiento, hasta alcanzar una conclusión.

Los lenguajes **declarativos**, como PROLOG, necesitan que se declaren reglas y hechos sobre símbolos específicos, y luego se le pregunte sobre si un objetivo concreto se deduce lógicamente a partir de los mismos. El propio lenguaje realiza el trabajo de decidir cómo alcanzar dicho objetivo.

4.1.4 Inteligencia Artificial

Un **programa inteligente** exhibe un comportamiento similar al de un ser humano cuando se enfrenta a un problema similar. El programa no necesita pensar como un humano, pero debe actuar como tal.

Los dos problemas más significativos de IA son los sistemas expertos y el procesamiento de lenguaje natural. Un **sistema experto** es un programa de computadora que contiene conocimientos acerca de un determinado campo, y cuando es interrogado responde como un experto humano. El **procesamiento del lenguaje natural** es la técnica que fuerza a las computadoras a entender el lenguaje humano.

4.2 Relación con la lógica

PROLOG trabaja con lógica proposicional, también conocida como lógica de predicados o cálculo proposicional. Hace que la computadora maneje la parte de inferir, a través de un motor de inferencia incorporado que automáticamente busca los hechos y construye conclusiones lógicas.

La programación en PROLOG consiste en:

- Declarar algunos hechos sobre los objetos y sus relaciones.
- Definir algunas reglas sobre los objetos y sus relaciones.
- Hacer preguntas sobre los objetos y sus relaciones.

4.2.1 Hechos

Un **hecho** es una forma de combinar un objeto y una relación, mediante la sintaxis

relacion(objeto).

La relación se conoce como **predicado** y el objeto como **argumento**. A continuación se presentan algunas consideraciones:

- Los nombres de todos los objetos y relaciones deben comenzar con una letra minúscula.
- Primero se escribe la relación y luego los objetos, separándolos mediante comas y encerrándolos entre paréntesis.

- Al final de cada hecho debe ir un punto.
- El carácter "_" en el nombre del predicado indica que todo es una única palabra.
- Dos hechos coinciden si sus predicados se escriben de igual forma y si cada uno de sus correspondientes argumentos son iguales entre sí.

4.2.2 Variables

En PROLOG, además de poder nombrar determinados objetos, también se pueden utilizar nombres como X que representen objetos a los que el mismo PROLOG les dará valor. Este tipo de nombres se llama **variable**.

Cuando PROLOG utiliza una determinada variable, ésta puede estar **instanciada o no**. El primer caso se da cuando existe un objeto determinado representado por la variable; en el segundo caso, todavía no se conoce lo que la variable representa.

Las variables deben comenzar con una letra mayúscula.

Cuando se intenta establecer una relación que contenga una variable, PROLOG efectuará una búsqueda recorriendo todos los hechos que él tiene almacenados para encontrar un objeto que pueda ser representado por la variable.

A veces es necesario utilizar una variable aunque su nombre no se utilice nunca. Por ejemplo, supongamos que queremos averiguar si a alguien le gusta Eduardo, pero no estamos interesado en saber quién. Aquí es donde utilizamos **variables anónimas**, a través de la siguiente sintaxis:

le_gusta_a(_, Eduardo)

4.2.3 Reglas

En PROLOG se utilizan las **reglas** cuando se quiere expresar que un hecho depende de un grupo de otros hechos.

Una regla consiste en una **cabeza** y un **cuerpo**. Estas partes se encuentran separadas por el símbolo ":-", que se lee "si". La regla describe qué hecho es el que la regla intenta definir, mientras que el cuerpo expresa la conjunción de objetivos que deben satisfacerse, uno tras otro, para que la cabeza sea cierta.

Las reglas son mucho más compactas que una lista de hechos, y **hacen que PROLOG pase de ser solo un diccionario o una base de datos en el que se puede buscar, a ser una máquina lógica, pensante**.

Por tanto, decimos que una regla es una **afirmación general sobre objetos y sus relaciones**.

4.2.4 Cláusulas

Utilizaremos la palabra **cláusula** siempre que nos refiramos a un hecho o a una regla.

4.2.5 Preguntas

Una vez que disponemos de hechos podremos hacer preguntas acerca de ellos. En PROLOG, una pregunta se representa igual que un hecho, salvo que se antepone un símbolo "?-".

Cuando se hace una pregunta, PROLOG efectúa una búsqueda por toda la base de datos, localizando hechos que coincidan con el hecho en cuestión. Si se encuentra uno que coincida, se responderá "sí", en caso contrario, "no".

4.2.6 Conjunciones y backtracking

En caso de querer contestar preguntas sobre relaciones un poco más complejas, como "¿se gustan José y María?", primero debería preguntarse si José gusta de María y luego, si la respuesta fue "sí", si María gusta de

José. Esto representa un problema con dos objetivos separados. El "y" expresa el interés de la **conjunción** de los dos objetivos: lo que se intenta hacer es satisfacer ambos. Esto, en PROLOG, se expresaría:

?- le_gusta_a(jose,maria),le_gusta_a(maria,jose).

La coma se lee "y", y separa los objetivos de la pregunta.

Cuando PROLOG tiene que satisfacer, digamos, dos objetivos, buscará el primero en la base de datos: si lo encuentra, marcará el lugar e intentará satisfacer el segundo, y si lo logra, marcará también dicha posición en la base de conocimientos, lo que determina que se encontró la solución.

Si, por el contrario, el segundo objetivo no se logra, PROLOG intentará volver a satisfacer el objetivo anterior. Lo que hace es volver hacia atrás e intentar satisfacer nuevamente su anterior objetivo comenzando desde su correspondiente marca de posición, y no desde el principio de la base de datos. Luego, intentará satisfacer el segundo objetivo. Al comportamiento de intentar repetidamente satisfacer o volver a satisfacer los objetivos de una conjunción se lo llama **reevaluación o backtracking**.

4.3 Estructura de un programa PROLOG

4.3.1 Composición de un programa. Cláusulas. Predicados. Dominios

La mayoría de los programas PROLOG están organizados en cuatro etapas:

- Cláusulas.
- Predicados.
- Dominios.
- Objetivos.

Cláusulas

Aquí se listan los hechos y las reglas.

Predicados

En este apartado se declaran todos los predicados no predefinidos que se utilizarán en la sección "Cláusulas".

Dominios

En esta sección se declaran los argumentos que utilizan los predicados. Si son dominios predefinidos no es necesario declararlos.

Objetivos

Ésta es la sección que le dice al PROLOG lo que ha de encontrar.

Un programa PROLOG puede tener dos tipos de objetivos:

- **Internos**: el programa se ejecutará y automáticamente procederá a aprobar o desaprobar el objetivo, informando el resultado.
- **Externos**: el programa se ejecutará y luego esperará que se le introduzca un objetivo. Una vez verificado el mismo, el programa pedirá uno nuevo, y así sucesivamente.

4.4 Ingeniería del conocimiento

La **ingeniería del conocimiento** es la disciplina que trata la forma en que se organizan, construyen y verifican las bases de conocimiento.

El **ingeniero de conocimiento** es el encargado de entrevistar a los expertos reales para aprender sobre lo que ellos saben, y poder así organizar la información obtenida para construir bases de conocimiento.

4.4.1 Organización de una base de conocimientos

Podemos identificar dos enfoques para la organización de la información dentro de una base de conocimientos:

- **Situar los objetos más probables lo más cerca del comienzo**. El problema aquí es decidir qué objetos son los más y menos probables.
- **Situar los atributos que causan la poda de la mayor parte del "árbol" lo más cerca del comienzo**. De nuevo, puede ser difícil determinar qué atributos causan la mayor poda.

4.4.2 Encontrando el experto

Nos encontramos con algunos problemas a la hora de encontrar un experto humano de quien pueda extraerse fácilmente el conocimiento:

- A menudo **es difícil determinar quién es o no un verdadero experto**.
- **Dos expertos diferentes en la misma materia pueden tener opiniones contradictorias**. Cuando sucede esto, generalmente puede elegirse entre tres estrategias:
    - Seleccionar un experto al azar e ignorar al otro. Aunque esta opción es fácil, la base de conocimientos podría llegar a contener alguna información errónea.
    - Promediar la información, esto es, usar sólo la información que es común entre los expertos. También puede hacerse sin mucho problema, pero la base de conocimientos podría no reflejar el conocimiento pleno de ningún experto.
    - Incluir el conocimiento de ambos expertos y dejar que el usuario decida. Aunque esto puede ser aceptable, en algunas aplicaciones se querrá que el sistema experto sea la autoridad, sin forzar al usuario a decidir.
- **La mayoría de los expertos humanos no sabe realmente lo que saben**. Por tanto, puede ser difícil extraer toda la información necesaria.
- Algunos expertos estarán simplemente **menos inclinados a perder tiempo** diciendo todo lo que saben sobre la materia.

4.4.3 Verificaciones de la base de conocimientos

El **testeo de la base de conocimientos** consiste en verificar que se ha transcripto e introducido correctamente el conocimiento en la computadora.

Realizar una búsqueda exhaustiva sobre algo que no sea un conjunto de datos extremadamente pequeño es impracticable debido a la explosión combinatoria. Esto significa que, para la mayoría de los sistemas expertos del mundo real, **no hay forma de verificar completamente que la base de conocimientos sea exacta**.

En consecuencia, la mejor solución es hacer suficiente testeo de forma que se pueda confiar bastante en la base de conocimientos. Usando **técnicas de muestreo estadísticas**, puede idearse una serie de test que producirán cualquier nivel de confianza que se desee.

En otro enfoque, se tiene la **autocomprobación** del sistema por consistencia y ver que toda la información de la base de conocimientos concuerda consigo misma.

4.5 Definición de sistema experto

4.5.1 Definición funcional

Para que un sistema informático pueda ser llamado "experto", ha de satisfacer un criterio similar al Test de Turing. Es decir que, **visto como una caja negra sea indistinguible, por su comportamiento, de un experto humano**.

4.5.2 Definición estructural

En un **sistema elemental**, el programa contiene no solo el conocimiento y los procedimientos, sino también la información sobre la forma de comunicación con el usuario y la forma de acceder a los datos.

La solución a este problema es bien conocida. Por una parte, el programa se independiza de los formatos de comunicación con el usuario gracias a una **capa de entrada-salida**. Por otra parte, se independiza de las estructuras de datos mediante el uso de un **sistema de gestión de bases de datos**.

En un **sistema experto** se independiza el conocimiento de los procedimientos que hacen uso de él, por tanto, obtenemos dos módulos bien diferenciados: la base de conocimiento y el motor de inferencia.

Así, podemos distinguir tres **componentes estructurales básicos**:

- **Base de hechos**: contiene el conocimiento declarativo sobre el problema particular que, en un momento dado, se intenta resolver.
- **Base de conocimiento**: contiene el conocimiento específico y procedimental acerca de la clase de problemas en los que el sistema es experto.
- **Motor de inferencia**: controla al resto del sistema en sus funciones deductivas.

Por otro lado, tres son los **tipos de personas que se relacionan con un sistema experto**:

- **Usuario final**: dialoga con el sistema para resolver problemas o aprender.
- **Experto humano**: comunica su conocimiento para construir el sistema.
- **Ingeniero de conocimiento**: diseña las estructuras de datos más adecuadas para la representación del conocimiento, y traduce los conocimientos del experto humano a tales estructuras.

Por último, hay dos grandes diferencias entre la línea de trabajo actual sobre sistemas expertos respecto de los trabajos clásicos:

- Concentrarse en áreas muy específicas del saber y no en problemas generales.
- Dar importancia a que los sistemas puedan explicar sus razonamientos y dar justificaciones al usuario.

4.6 Armazones de sistemas expertos

En un principio, los sistemas expertos que se construían eran creados desde cero. Luego se vio que tenían varios puntos en común. En particular, desde que los sistemas se creaban como un conjunto de representaciones declarativas, acompañadas por un intérprete para dichas declaraciones, fue posible separar el conocimiento específico del intérprete. Los intérpretes resultantes se llaman **armazones o shells**.

Los primeros armazones de sistemas expertos proporcionaban mecanismos para **representación del conocimiento, el razonamiento y la explicación**. Más tarde, se les añadieron herramientas para la **adquisición del conocimiento**. Por último, se facilitó la **integración con otros tipos de programas**.

4.7 Aplicaciones de los sistemas expertos

4.7.1 Ventajas de la aplicación de sistemas expertos

- A diferencia de un experto humano, el sistema experto está accesible para su uso las veinticuatro horas del día, todos los días del año.
- Pueden crearse muchos sistemas expertos, mientras que el número de expertos humanos puede ser limitado.
- Los sistemas expertos nunca mueren, llevándose consigo el conocimiento.
- Los sistemas expertos siempre rinden al máximo. No se agotan.
- No tienen personalidad, a diferencia de un experto humano que puede tener reservas para expresar su conocimiento.
- Para adquirir un nuevo sistema experto, simplemente se copia el programa de una máquina a otra, mientras que un humano necesita mucho tiempo para convertirse en experto.

REDES NEURONALES

5.1 Modelos conexionistas

Los **modelos de computación conexionista**, o **redes neuronales artificiales**, suponen en primer lugar la deducción de las características esenciales de las neuronas y sus conexiones, y en segundo lugar la implementación del modelo en una computadora de forma que se pueda simular.

5.1.1 Origen del paradigma de computación conexionista

La inteligencia artificial se separó, casi desde su inicio, en dos ramas bien diferenciadas:

- Por un lado se trató de modelar la actividad racional mediante **sistemas formales de reglas y manipulación simbólica**: se postula una serie de reglas y el sistema resuelve los problemas realizando deducciones sobre las reglas existentes.
- Por otro lado se desarrollaron modelos computacionales inspirados en las redes neuronales biológicas, denominados **inductivos o subsimbólicos**.

Una **neurona formal** es un dispositivo binario con varias entradas y salidas.

Las **hipótesis de Hebb** son:

- Una percepción o un concepto se representa en el cerebro por un conjunto de neuronas activas simultáneamente.
- La memoria se localiza en las conexiones entre las neuronas.

Luego, la **regla de aprendizaje de Hebb** indica que las conexiones entre dos neuronas se refuerzan si ambas son activadas.

5.3 Redes neuronales artificiales

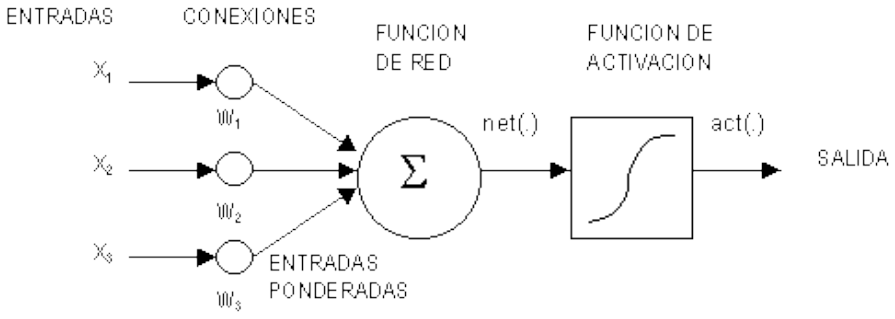
5.3.1 Definición de red neuronal

Una **red neuronal** se define como un sistema compuesto de muchos elementos simples de procesamiento, los cuales operan en paralelo y cuya función es determinada por la estructura de la red y el peso de las conexiones, realizándose el procesamiento en cada uno de los elementos de procesamiento.

5.3.2 Estructura de las redes neuronales

Las neuronas son modeladas mediante **unidades de proceso**. Cada unidad de proceso se compone de:

- Una red de conexiones de entrada.
- Una función de red, o propagación, encargada de computar la entrada total combinada.
- Un núcleo central de proceso, encargado de aplicar la función de activación.
- Una salida, por donde se transmite el valor de activación a otras unidades.



Conexiones ponderadas: cumplen el papel de las conexiones sinápticas. La existencia de conexiones determina si es posible que una unidad influya sobre otra.

Peso sináptico (w_i): el peso de la conexión equivale a la fuerza o efectividad de la sinapsis. El valor de los pesos y el signo del mismo definen el tipo (excitatorio o inhibitorio) y la intensidad de la influencia.

Función de red o propagación: calcula el valor de entrada total a la unidad, generalmente como una simple suma ponderada de todas las entradas recibidas, es decir, de las entradas multiplicadas por el peso o valor de las conexiones. Equivale a la combinación de las señales excitatorias e inhibidoras de las neuronas biológicas. Entre las funciones más importantes podemos citar:

- Función lineal de base: es la más utilizada.

$$net_j(X, W_j) = \sum_{i=1}^{n} x_i w_{ij}$$

- Función radial de base.

$$net_j(X, W_j) = \sqrt{\sum_{i=1}^{n} (x_i - w_{ij})^2}$$

En ambos casos, hay una unidad j con n unidades conectadas a ella. X es el vector de entradas y W_j es el vector de pesos de las conexiones correspondientes.

Función de activación: se encarga de calcular el nivel de activación de la neurona en función de la entrada total. Es quizá la característica principal de las neuronas, la que mejor define el comportamiento de las mismas. Podemos distinguir entre:

- Funciones lineales: la salida es proporcional a la entrada.
- Funciones de umbral: la salida es un valor discreto, típicamente binario, que depende de si la estimulación total supera o no un determinado valor de umbral.
- Funciones no lineales: la salida no es proporcional a la entrada.

Salida: calcula la salida de la neurona en función de la activación de la misma, aunque normalmente se aplica la función identidad, tomándose como salida el valor de activación. El valor de salida cumpliría la función de la tasa de disparo en las neuronas biológicas.

5.3.3 Comparación entre RNB y RNA

Redes neuronales biológicas	Redes neuronales artificiales
Neuronas	Unidades de proceso
Conexiones sinápticas	Conexiones ponderadas
Efectividad de la sinapsis	Peso de las conexiones
Efecto excitatorio o inhibitorio de una conexión	Signo del peso de una conexión
Efecto combinado de la sinapsis	Función de red o propagación
Activación → tasa de disparo	Función de activación → salida

5.3.4 Formas de interconexión de las RNA

Para diseñar una red debemos establecer cómo estarán conectadas unas unidades con otras y determinar adecuadamente el peso de las conexiones. Lo más usual es **disponer las unidades en forma de capas**, pudiéndose hablar de redes de una, dos o más capas, llamadas redes multicapa.

- **Capa de entrada**: es la primera capa y actúa como buffer de entrada, almacenando la información en bruto suministrada a la red, o realizando un sencillo pre-proceso de la misma.
- **Capas ocultas**: son las capas intermedias, principales encargadas de extraer, procesar y memorizar la información.
- **Capa de salida**: actúa como interfaz o buffer de salida, almacenando la respuesta de la red para que pueda ser leída.

Además de por el número de capas, las redes pueden clasificarse en función de cómo se interconectan unas capas con otras:

- **Redes en cascada**: la información fluye unidireccionalmente de una capa a otra (desde la capa de entrada a las capas ocultas, y de éstas a la capa de salida) y, además, no se permiten conexiones intracapa.
- **Redes recurrentes**: la información puede volver a lugares por los que ya pasó, y se admiten las conexiones intracapa, incluso entre una red consigo misma.

Las conexiones entre una capa y otra pueden ser:

- **Totales**: cada unidad se conecta con todas las unidades de la capa siguiente.
- **Parciales**: una unidad se conecta con sólo algunas de las unidades de la capa siguiente, generalmente siguiendo algún patrón aleatorio o pseudoaleatorio.

Desde una aproximación temporal, podemos distinguir redes con **conexiones sin retardo** y con **conexiones con retardo**.

5.3.5 Características de las RNA

- **Aprendizaje inductivo**: no se le indican las reglas para dar una solución, sino que extrae sus propias reglas a partir de los ejemplos de aprendizaje. Esas reglas quedan almacenadas en las conexiones.
- **Generalización**: una vez entrenada, se le puede presentar a la red datos distintos a los usados durante el aprendizaje.
- **Tolerancia al ruido**: son capaces de extraer las características esenciales de las entradas aprendidas, de forma que pueden procesar correctamente datos incompletos.
- **Procesamiento paralelo**: la estructura y modo de operación de las redes neuronales las hace especialmente adecuadas para el procesamiento paralelo real mediante multiprocesadores.
- **Memoria distribuida**: el conocimiento acumulado por la red se halla distribuido entre numerosas conexiones, lo que resulta en tolerancia a fallos.

5.4 Ventajas y desventajas de las RNA

5.4.1 Ventajas que ofrecen las RNA

- **Aprendizaje adaptativo**: aprenden a llevar a cabo ciertas tareas mediante un entrenamiento con ejemplos ilustrativos.
- **Auto organización**: mientras que el aprendizaje es la modificación de cada elemento procesal, la auto organización consiste en la modificación de la red neuronal completa para llevar a cabo un objetivo específico.

- **Tolerancia a fallos**: la destrucción parcial de una red conduce a una degradación de su estructura, sin embargo, algunas capacidades de la red se pueden retener.
- **Operación en tiempo real**: los cómputos neuronales pueden ser realizados en paralelo, para lo cual se diseña hardware especial.
- **Fácil inserción dentro de la tecnología existente**: pueden obtenerse chips especializados para redes neuronales que mejoran su capacidad.

5.4.2 Desventajas que ofrecen las RNA

- **Definición de muchos parámetros**: requieren la definición de muchos parámetros antes de poder aplicar esta metodología.
- **Caja negra**: no ofrecen una fácil interpretación de cómo toman sus decisiones.

5.5 Mecanismos de aprendizaje

El **aprendizaje** es el proceso por el cual una red neuronal modifica sus pesos en respuesta a una información de entrada, por tanto, se puede afirmar que este proceso ha terminado, es decir, **la red ha aprendido**, cuando los valores e los pesos permanecen estables.

Existen dos formas de clasificar los métodos de aprendizaje. Por un lado:

- **Aprendizaje supervisado**.
- **Aprendizaje no supervisado**.

y, por otro lado:

- **Aprendizaje online**: la red puede aprender durante su funcionamiento habitual.
- **Aprendizaje offline**: el aprendizaje supone la desconexión de la red hasta que el proceso termine.

5.5.1 Aprendizaje supervisado

El **aprendizaje supervisado** se caracteriza porque el proceso de aprendizaje se realiza mediante un entrenamiento controlado por un agente externo que determina la respuesta que debería generar la red a partir de una entrada determinada.

Encontramos tres formas de llevarlo a cabo:

- **Aprendizaje por corrección de error**: consiste en ajustar las conexiones de la red en función de la diferencia entre los valores deseados y los obtenidos a la salida de la red, es decir, en función del error cometido en la salida.
- **Aprendizaje por refuerzo**: la función del supervisor se limita a indicar mediante una señal de refuerzo si la salida obtenida en la red se ajusta a la deseada (éxito = +1, fracaso = -1), y en función de ello se ajustan los pesos. Aquí, la función del supervisor se asemeja a la de un crítico.
- **Aprendizaje estocástico**: consiste en realizar cambios aleatorios de los valores de los pesos y determinar la energía de la red (el estado de mínima energía se corresponde a una situación en la que los pesos de las conexiones consiguen que el funcionamiento sea el que más se ajusta al objetivo buscado). Si la energía es menor después del cambio, se acepta el mismo. Por el contrario, se aceptaría el cambio en función de una distribución de probabilidades predeterminada.

5.5.2 Aprendizaje no supervisado

Las redes con aprendizaje no supervisado no requieren influencia externa para ajustar los pesos de las conexiones entre sus neuronas. La red no recibe ninguna información por parte del entorno que le indique si la salida generada en respuesta a una entrada es correcta o no.

5.6 El perceptrón

Un **perceptrón** es una red neuronal en la que:

- Sólo hay una neurona.
- Las entradas son binarias: 0 y 1.
- Las cajas lógicas pueden interponerse entre las entradas y los pesos del perceptrón. Cada caja lógica puede verse como una tabla que produce un valor de salida de 0 o 1 para cada combinación de 0 y 1 que pueda aparecer en sus entradas.
- La salida del perceptrón es 0 o 1, dependiendo de si la suma ponderada de las salidas de las cajas lógicas es mayor o menor que el umbral.

En un **perceptrón limitado por el orden de orden n**, cada caja lógica atiende n o menos entradas. En un **perceptrón directo**, cada caja lógica tiene sólo una entrada, y la salida es siempre la misma que la entrada.

5.6.1 Aprendizaje del perceptrón

Existe un procedimiento que determina un buen conjunto de pesos para un perceptrón.

Se empieza con todos los pesos en cero. Después se intenta el perceptrón con todas las muestras, una a la vez. Siempre que el perceptrón cometa un error, se cambian los pesos de modo que el error se haga menos probable; en cualquier otro caso, no se hace nada.

- Se aumentan los pesos de las cajas lógicas que producen 1, si el perceptrón indica un 0 cuando debía ser 1.
- Se disminuyen los pesos de las cajas lógicas que producen 1, si el perceptrón indica un 1 cuando debía ser 0.
- En ningún caso se alteran los pesos asignados a las cajas que producen 0 ya que, al estar tales pesos multiplicados por 0, no sería posible cambiar el resultado actual.

5.6.2 La separación lineal y el problema del XOR

El **teorema de convergencia** del perceptrón garantiza que el perceptrón encontrará un estado solución, es decir, aprenderá a clasificar cualquier conjunto de entradas linealmente separables.

Sin embargo, el perceptrón es incapaz de aprender a resolver algunos problemas sencillos, como por ejemplo el de la función XOR. No hay forma de separar las salidas 0 de las salidas 1.

El problema aquí no está en el algoritmo de aprendizaje del perceptrón, sino en el modo en que el perceptrón representa el conocimiento. Dos posibles soluciones son:

- Poder dibujar una **superficie de decisión elíptica**, siendo capaz así de separar las salidas 0 y 1. Sin embargo, los perceptrones son incapaces de modelar dichas superficies.
- Emplear dos escenarios diferentes para el dibujo de las líneas. Utilizando esta idea se podría construir un **perceptrón multicapa**.

5.7 Redes de Hopfield

Una **red de Hopfield** tiene las siguientes características:

- **Representación distribuida**: una memoria se almacena como un patrón de activación a través de un conjunto de elementos de proceso. Las diferentes memorias se representan por diferentes patrones sobre el mismo conjunto de elementos de proceso.
- **Control asíncrono y distribuido**: cada elemento de proceso toma decisiones basadas únicamente en su propia situación actual. Todas estas situaciones locales se unen para alcanzar una solución global.

- **Memoria direccionable por contenido**: para recuperar un patrón únicamente se necesita una parte específica de él.
- **Tolerancia a fallos**: aunque algunos de los elementos procesadores de la red fallasen, ésta todavía funcionará adecuadamente.

En una red de Hopfield, los elementos de proceso, llamados también **unidades**, siempre se encuentran en estado activo o inactivo. Las unidades están conectadas unas con otras por conexiones simétricas y con pesos. Una conexión con **peso positivo** indica que las dos unidades tienden a activarse la una a la otra. Una conexión con **peso negativo** indica que una unidad activa desactivará a su unidad vecina.

El funcionamiento de la red comienza eligiendo una unidad aleatoriamente. Si alguna de sus vecinas está activada, la unidad calcula la suma de los pesos en las conexiones entre esas unidades. Si la suma es positiva, la unidad se activa, de forma contraria, se desactiva. Entonces se elige otra unidad aleatoriamente y se repite el proceso hasta que la red alcanza un estado estable, es decir, hasta que no quede ninguna unidad que pueda cambiar de estado. Este proceso se denomina **relajación paralela**.

Dado un estado inicial, una red necesariamente se asentará en algún estado estable, por lo que se la puede ver como un almacenador de patrones. La mayor contribución de las redes de Hopfield es la de mostrar que **el algoritmo de relajación paralela llevará en algún momento hacia un estado estable**. No puede existir divergencia u oscilación.

Un estado que se haya elegido aleatoriamente se transformará a sí mismo en uno del tipo de mínimo local que es el estado estable más cercano. **Ésta es la forma de obtener un comportamiento direccionable por contenido**.

Ahora supóngase que una unidad falla de repente haciéndose activa o inactiva cuando no debe. Esto no representaría mayor problema, ya que las unidades que le rodean rápidamente volverían a ponerle en el estado correcto. **Ésta es la forma de obtener tolerancia a fallos**.

5.8 Máquinas de Boltzman

El principal problema de las redes de Hopfield es que convergen hacia mínimos locales. No pueden encontrar soluciones globales.

El **enfriamiento simulado** es una técnica para encontrar soluciones globalmente óptimas en problemas combinatorios. Decimos que a altas temperaturas, las unidades presentan un comportamiento aleatorio, mientras que a bajas temperaturas, las unidades se comportan como redes de Hopfield. El enfriamiento es el proceso de ir gradualmente de altas a bajas temperaturas. La aleatoriedad que añade la temperatura ayuda a la red a escapar de los mínimos locales.

Una **máquina de Boltzman** es una variación de las redes de Hopfield, en la que se las combina con el enfriamiento simulado. Las unidades de una máquina de Boltzman actualizan sus estados binarios individuales mediante una regla estocástica, en lugar de una regla determinística.

5.9 Redes recurrentes

Las **redes recurrentes** constituyen un intento de remediar la deficiencia de las redes neuronales de tratar con tareas temporales de IA, como planificación y lenguaje natural. Estas redes nunca convergen hacia un estado estable sino que, en lugar de esto, cambian con cada paso del tiempo. En cada paso se comparan las activaciones de las unidades de salida con las activaciones deseadas, y se propagan los errores hacia atrás por la red.