

Inteligencia Artificial, Robótica, Neurocomputación, Programación Neuronal y otras hierbas

Por David Díaz (editor)

¿Qué es la IA?

Existen numerosas definiciones de Inteligencia Artificial, dependiendo del autor o el campo de especialización. Para hacernos una idea, aquí hay cuatro modelos bastante representativos de esta ciencia.

- IA es la atribuida a las máquinas capaces de hacer operaciones propias de seres inteligentes (DRAE).
- La IA es el estudio de las computaciones que permiten percibir, razonar y actuar (Winston).
- La IA es el estudio de técnicas de resolución de problemas de complejidad exponencial mediante el uso de conocimiento sobre el campo de aplicación del problema (Rich)

La Inteligencia Artificial tiene dos aspectos, uno como ciencia cognitiva y otro como tecnología informática, y sus características esenciales son:

- Información simbólica preferente a la numérica.
- Métodos heurísticos preferente a los algorítmicos.
- Uso de conocimiento específico-declarativo.
- Informaciones incompletas o con incertidumbre.
- Multidisciplinaridad.

Inteligencia Artificial, programación neuronal y otras hierbas por David Díaz (editor)

Nacimiento de la IA

Si bien a principios de la década de los años 50 aparecieron ya los primeros programas de cálculo formal (que permitían a los ordenadores, utilizados hasta entonces únicamente como máquinas de calcular, manipular símbolos), lo que más tarde recibiría el nombre de inteligencia artificial nació en realidad, en el campo de la informática, con la aparición del primer programa capaz de demostrar teoremas de la lógica de las proposiciones (el **Logic Theorist** creado por **Newell, Shaw y Simon**). Dicho programa fue presentado durante la conferencia de investigadores que se celebró en el colegio de Darmouth (1956). En aquella ocasión se acuñó, también, el término de inteligencia artificial. Este avance era consecuencia de la carencia de algoritmos que fuesen capaces de describir una cierta serie de actividades cognitivas como el reconocimiento visual de un objeto, la comprensión de los lenguajes naturales (hablados o escritos), el diagnóstico de enfermedades en el ser humano o de averías en las máquinas, etc. La inteligencia artificial nació, pues, como resultado de la confluencia de dos corrientes diversas: por un lado, la científica, que tenía como objetivo intentar comprender los mecanismos de la inteligencia humana empleando para ello, como modelo de simulación, los ordenadores y,

por otro lado, la técnica, que pretendía equipar a los ordenadores de capacidades de pensamiento lo más similares posible a las humanas pero sin la pretensión de imitar con toda exactitud los pasos que sigue el ser humano para llevar a cabo dichas actividades intelectuales. Este proceso se vio además reforzado por la aparición de lenguajes de programación bien adaptados a la inteligencia artificial, el **LISP** (creado por **McCarthy** a partir de 1960).

El periodo que abarca 1956 a 1968 se caracterizó, en este campo, por los intentos dirigidos a la búsqueda y modelización de determinados **principios generales de la inteligencia** (aplicaciones como la traducción automática, la percepción visual, etc.). A finales de la década de los años 60, los trabajos se encaminaron hacia el desarrollo de sistemas inteligentes de aplicación en la **robótica** (visión artificial, manipulación de objetos, navegación automática, etc.) en los que era necesario incorporar una gran cantidad de conocimientos específicos referidos a los problemas que se pretendía resolver con dichas técnicas. Este proceso marcó el inicio del estudio de los llamados **sistemas expertos**.

¿Para qué sirve?

Clasificación de campos de la IA según la ACM:

- Programación automática: Verificación y síntesis.
- Razonamiento automático.
- Representación del conocimiento.
- Metodología de la programación en IA.
- Aprendizaje.
- Procesamiento del lenguaje natural.
- Resolución de problemas, métodos de control y búsqueda.
- Robótica.
- Interpretación de imágenes y visión artificial.
- Inteligencia artificial distribuida.

Aplicaciones de la IA:

- **Tareas de la vida diaria:**
 - Percepción: visión y habla.
 - Lenguaje natural: comprensión, generación y traducción.
 - Sentido común.
 - Control de robot.

- **Tareas formales:**
 - Juegos: damas, ajedrez, go, ...
 - Matemáticas: cálculo simbólico, demostración de teoremas.
 - Computación: verificación de programas, aprendizaje automático.
- **Tareas de expertos:**
 - Ingeniería: diseño, detección de fallos, planificación de manufacturación.
 - Análisis científico.
 - Diagnóstico y tratamiento médico.
 - Análisis financiero.

Robótica

El nombre de robot procede del término checo **robota** (trabajador, siervo) con el que el escritor Karel Capek designó, primero en su novela y tres años más tarde en su obra teatral **RUR** (Los robots universales de Rossum, 1920) a los androides, producidos en grandes cantidades y vendidos como mano de obra de bajo costo, que el sabio Rossum crea para liberar a la humanidad del trabajo. En la actualidad, el término se aplica a todos los ingenios mecánicos, accionados y controlados electrónicamente, capaces de llevar a cabo secuencias simples que permiten realizar operaciones tales como carga y descarga, accionamiento de máquinas herramienta, operaciones de ensamblaje y soldadura, etc. Hoy en día el desarrollo en este campo se dirige hacia la consecución de máquinas que sepan interactuar con el medio en el cual desarrollan su actividad (reconocimientos de formas, toma de decisiones, etc.).

La disciplina que se encarga del estudio y desarrollo de los robots es la **robótica**, una síntesis de la automática y la informática. La robótica se centró, en primer lugar, en el estudio y desarrollo de los robots de la llamada **primera generación**; es decir, incapaces de detectar los estímulos procedentes del

entorno y limitados a las funciones con una secuencia predeterminada y fija. Estos robots han dado paso a los que constituyen la **segunda generación**, capaces de desarrollar algún tipo de actividad sensorial. Los prototipos multisensoriales que interactúan en un grado muy elevado con el entorno se agrupan en la **tercera generación**. Para ello, la robótica se sirve de disciplinas como la mecánica, la microelectrónica y la informática, además de incorporar a los ingenios técnicas como el reconocimiento y análisis digital de las imágenes, el desarrollo de sistemas sensoriales, etc.

El creciente desarrollo de los robots y su contante perfeccionamiento ha hecho que cada día se apliquen en mayor medida a los procesos industriales en sustitución de la mano de obra humana. Dicho proceso, que se inició hacia 1970, recibe el nombre de **robotización** y ha dado lugar a la construcción de plantas de montaje parcial o completamente robotizadas. Este proceso conlleva, según sus detractores, la destrucción masiva de puestos de trabajo, mientras que para sus defensores supone la satisfacción de necesidades socioeconómicas de la población y lleva aparejado un aumento muy considerable de la productividad.

La IA en la robótica:

A finales de los años 70, se produjo un nuevo giro en el campo de la investigación relacionada con la inteligencia artificial: la aparición de **robots**. Los robots experimentales creados para estos efectos eran automatismos capaces de recibir información procedente del mundo exterior (p. ej., sensores, cámaras de televisión, etc.), así como órdenes de un manipulador humano (expresadas en lenguaje natural). De este modo, el robo determinaba un plan y, de acuerdo con él, ejecutaba las órdenes recibidas mediante el empleo de un modelo del universo en el que se encontraba. Era incluso capaz de prever las consecuencias de sus acciones y evitar, así, aquéllas que más tarde pudieran resultarle inútiles o, en algún momento, perjudiciales. Estos primeros robots experimentales eran bastante más inteligentes que los robots industriales, y lo eran porque disponían de un grado mucho mayor de percepción del entorno que los robots empleados en las cadenas de producción.

El principal problema con el que se enfrenta la inteligencia artificial aplicada a los robots es el de la visión. Mientras que la información recibida a través de sensores se puede interpretar con relativa facilidad y entra a formar parte de la descripción del modelo de universo que emplea el robot para tomar decisiones, la percepción de las imágenes captadas y su interpretación correcta es una

labor muy compleja. En cuanto a la interpretación de las imágenes captadas mediante cualquier sistema, se ha logrado ya el reconocimiento de formas preprogramadas o conocidas, lo que permite que ciertos robots lleven a cabo operaciones de reubicación de piezas o colocación en su posición correcta a partir de una posición arbitraria.

Sin embargo, no se ha logrado aún que el sistema perciba la imagen tomada mediante una cámara de ambiente y adapte su actuación al nuevo cúmulo de circunstancias que esto implica. Así, por ejemplo, la imagen ofrecida por una cámara de vídeo de las que se emplea en vigilancia y sistemas de seguridad no puede ser interpretada directamente por el ordenador.

Sistemas Expertos

Los sistemas expertos se basan en la simulación del razonamiento humano. El razonamiento humano tiene para ellos, un doble interés: por una parte, el del análisis del razonamiento que seguiría un experto humano en la materia a fin de poder codificarlo mediante el empleo de un determinado lenguaje informático; por otra, la síntesis artificial, de tipo mecánico, de los razonamientos de manera que éstos sean semejantes a los empleados por el experto humano en la resolución de la cuestión planteada. Estos dos campos de interés han conducido a los investigadores que trabajan en el campo de la inteligencia artificial (de la cual los sistemas expertos son un campo preferente) a intentar establecer una metodología que permita verificar el intercambio con los expertos humanos y aislar los diversos tipos de razonamiento existentes (inductivo, deductivo, formal, etc.), así como construir los elementos necesarios para modelizarlos.

Los sistemas expertos son, por lo tanto, intermediarios entre el experto humano, que transmite sus conocimientos al sistema, y el usuario de dicho sistema, que lo emplea para resolver los problemas que se le plantean con la

competencia de un especialista en la materia y que, además, puede adquirir una destreza semejante a la del experto gracias a la observación del modo de actuar de la máquina. Los sistemas expertos son, pues, simultáneamente, un sistema de ejecución y un sistema de transmisión del conocimiento. Asimismo, los sistemas expertos se definen mediante su arquitectura; obtienen, por lo tanto, una realidad palpable. Mientras que en las operaciones de programación clásicas se diferencia únicamente entre el propio programa y los datos, en el caso de los sistemas expertos se diferencian tres componentes principales. Son los siguientes:

- Base de conocimientos
- Base de hechos
- Motor de inferencia

La base de conocimientos aloja la totalidad de las informaciones específicas relativas al campo del saber deseado. Está escrita en un lenguaje específico de representación de los conocimientos que contiene y en el cual el experto puede definir su propio vocabulario técnico. A la inversa de lo que sucede en los programas clásicos, en la base de conocimientos las informaciones entran tal como llegan, ya que el orden no influye en los resultados obtenidos. Sucede así

porque cada elemento de conocimiento es comprensible por sí mismo tomado de forma aislada y, por lo tanto, no es necesario referirse al contexto en el cual está inserto. La información se representa, por regla general, mediante reglas de producción o redes semánticas. Las reglas de producción constituyen el método más utilizado para construir bases de conocimientos en los sistemas expertos. Llamadas también implicaciones lógicas, su estructura es la siguiente: para unas ciertas causas, unos efectos; o, para determinadas condiciones, ciertas consecuencias. Junto a cada regla, se almacena también su porcentaje en forma de probabilidad. Éste indica, mediante un tanto por ciento, el grado de certeza de las consecuencias que se obtienen como resultado de la aplicación de la regla de producción. En cuanto a las redes semánticas, se trata de un método de construcción de bases de conocimientos en el cual los conocimientos se muestran mediante un grafo en el que los vértices representan los conceptos u objetos y las aristas indican las relaciones entre ellos.

Además el sistema dispone de la llamada base de hechos, que alberga los datos propios correspondientes a los problemas que se desea tratar con la ayuda del sistema. Asimismo, a pesar de ser la memoria de trabajo, la base de hechos puede desempeñar el papel de memoria auxiliar. La memoria de

trabajo memoriza todos los resultados intermedios, permitiendo conservar el rastro de los razonamientos llevados a cabo. Puede, por eso, emplearse para explicar el origen de las informaciones deducidas por el sistema en el transcurso de una sesión de trabajo o para llevar a cabo la descripción del comportamiento del propio sistema experto. Al principio del período de trabajo, la base de hechos dispone únicamente de los datos que le ha introducido el usuario del sistema, pero, a medida que va actuando el motor de inferencias, contiene las cadenas de inducciones y deducciones que el sistema forma al aplicar las reglas para obtener las conclusiones buscadas.

El último elemento, el motor de inferencias, es un programa que, mediante el empleo de los conocimientos puede resolver el problema que está especificado. Lo resuelve gracias a los datos que contiene la base de hechos del sistema experto. Por regla general, el tipo de reglas que forman la base de conocimientos es tal que, si A es válido, puede deducirse B como conclusión. En este caso, la tarea que lleva a cabo el motor de inferencias es la de seleccionar, validar y activar algunas reglas que permiten obtener finalmente la solución correspondiente al problema planteado.

El sistema experto establecido se compone, por lo tanto, de dos tipos bien

diferenciados de elementos, los propios del campo de los expertos relacionados con el problema concreto (es decir, la base de conocimientos y la base de hechos) y el que se puede aplicar de forma general a una gran variedad de problemas de diversos campos (como el caso del motor de inferencias). Sin embargo, el motor de inferencias no es un mecanismo universal de deducción, ya que hay dos tipos diverso: los que emplean el razonamiento aproximativo (para el cual el resultado puede ser erróneo) y aquellos que emplean un tipo de razonamiento capaz de obtener un resultado (si llegan a él), con toda seguridad, verdadero.

Fases del proceso

Sin embargo, a pesar de no existir una metodología generalmente aceptada en cuanto a la concepción de los sistemas expertos, se admite por regla general un esquema que consta de tres fases. En la primera fase, la discusión con el experto o los expertos humanos en la cual se intenta, por un lado, delimitar el problema a resolver y, por el otro, los modos de razonamiento que se emplearán para su solución. La segunda fase comprende el desglose del formalismo de expresión del conocimiento y la determinación del motor de

inferencias adecuado a dicho formalismo. Por último, la tercera etapa, corresponde a la creación de la base de conocimientos (en colaboración con los expertos humanos), así como a la comprobación y ajuste del funcionamiento del sistema experto mediante el empleo de ejemplos.

Niveles de conocimiento

A pesar de no disponerse de un modelo general comúnmente aceptado, existe unanimidad en cuanto a la aprobación de tres niveles distintos de conocimientos, a los que corresponde tres fases diferentes de estudio y sobre los que se basa, en general, la concepción de un sistema experto. Esos niveles son el de estructuración, el conceptual y el cognoscitivo. El primero es el que define el mecanismo que genera la certeza. Este mecanismo varía según el campo al que se aplique el sistema experto, ya que las evidencias asociadas a cada campo no son idénticas. La determinación del nivel de estructuración permite definir un formalismo de representación del conocimiento así como un mecanismo adecuado de deducción. El nivel conceptual es el que determina el conjunto de los conceptos que emplea el experto humano en la materia; cada uno de ellos corresponde a un nudo del razonamiento del experto. Se le asocia un descriptor que se experimenta con el formalismo correspondiente al nivel de

estructuración. Finalmente, el nivel cognoscitivo corresponde al conjunto de los conocimientos que el experto humano pone en práctica para la resolución del problema planteado. Este conjunto de conocimientos debe poder traducirse al lenguaje definido mediante el formalismo de representación del conocimiento adoptado. En cuanto al desarrollo actual de la investigación en el campo de los sistemas expertos, la primera fase corresponde al desarrollo de sistemas y programas que traten directamente el lenguaje natural, si bien persisten todavía dos escollos importantes. Por un lado, el problema de cómo emplear de un modo eficaz una gran cantidad de información sin necesidad de echar mano de la combinatoria; es decir, cómo conseguir un sistema dotado de conocimientos (metaconocimientos) que le permitan utilizar los conocimientos del sistema y que, a su vez, le permitan deducir automáticamente nuevos conocimientos, ya que no cabe pensar en la reunión de todos los conocimientos necesarios en casos de campos tan sumamente vastos como el del diagnóstico en la medicina.

Neurocomputación

La neurocomputación trata de alcanzar algunas de las ventajas que proporcionan las redes neuronales biológicas, imitándolas tanto desde el punto de vista morfológico como desde el punto de vista funcional, para lo que se basa en la realización de Redes Neuronales Artificiales o ANN (del inglés Artificial Neural Networks). El motivo por el que se sigue esta vía de investigación es la observación de que el cerebro humano (y no sólo el humano) presenta ciertas notables y deseables características que no se encuentran ni en los ordenadores, que siguen la clásica arquitectura propuesta por Von Neumann, ni en los ordenadores paralelos modernos. Entre estas características cabe destacar el paralelismo masivo, lo que les dota de una gran robustez y tolerancia a fallos, la capacidad de aprender, la capacidad de generalizar, la capacidad de adaptación, la capacidad inherente de procesar información contextual, y aunque parezca lo más anecdótico, un bajo consumo. Es común el asombro que el ciudadano medio siente ante la enorme capacidad de proceso numérico y simbólico asociado que presentan los ordenadores, capacidad que supera con mucho, y desde hace tiempo, a la de cualquier ser humano. Sin embargo, cualquier persona que haya tenido cierto contacto con un ordenador sabrá que es una "máquina tonta", incapaz de realizar ninguna de las funciones citadas en el párrafo anterior. Y, de hecho, en otro tipo de problemas, es el hombre (y cualquier animal) el que deja en ridículo al

superordenador más potente. Uno de los ejemplos más claros consiste en la resolución de complejos problemas de percepción, como el reconocer a una persona entre una multitud, problema que un ser humano puede resolver en un simple golpe de vista.

Por todos estos motivos, las redes neuronales artificiales tratan de imitar los principios de organización, que se cree rigen el cerebro humano. La unidad funcional de estas redes es, por tanto, la neurona artificial. Una neurona artificial no es otra cosa que un procesador muy simple capaz de realizar instrucciones muy primitivas, pero a gran velocidad, y que guarda la información aprendida en las conexiones con otras neuronas.. Pero antes de profundizar en el estudio de la neurona artificial, es conveniente echar una mirada a la neurona biológica y lo que se puede aprender de ella.

La neurona biológica

Nuestro sistema nervioso y, en especial, nuestro cerebro, está compuesto por un conjunto de células nerviosas, también llamadas neuronas.

Una neurona es una célula altamente especializada en el proceso de la

información, cuya estructura básica puede observarse en la Figura 1. En ella se puede observar como la morfología de una neurona comprende tres elementos principales: el *soma*, o cuerpo principal; las *dendritas*, o terminaciones de la neurona que actúan como contactos funcionales de entrada con otras neuronas; y el *axón* o eje, una rama más larga que será la encargada de conducir el impulso nervioso y que finaliza también en diversas ramificaciones. La comunicación entre neuronas se realiza a través de las llamadas *sinapsis*, que son los puntos de conexión entre las fibras terminales del axón de una neurona y una dendrita de otra. Las sinapsis también reciben el nombre de *saltos sinápticos*, ya que en la mayoría de los casos, fibras terminales y dendritas no están en contacto, sino separadas por una pequeña distancia. Por ello, al contrario de lo que mucha gente piensa, el impulso nervioso no es de naturaleza eléctrica, sino electroquímica.

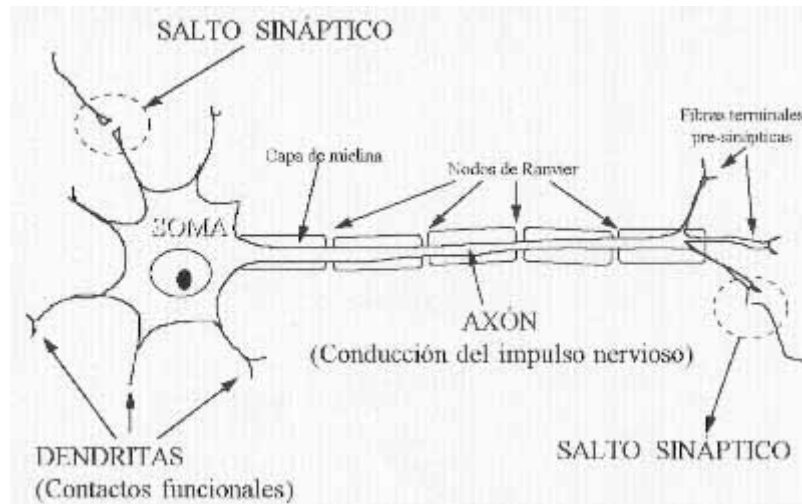


Figura 1. Anatomía de una neurona biológica.

El impulso nervioso producido por una neurona se propaga por el axón y al llegar al extremo, las fibras terminales presinápticas liberan unos compuestos químicos llamados *neurotransmisores*. Los neurotransmisores se liberan en la membrana celular pre-sináptica y alteran el estado eléctrico de la membrana

post-sináptica. En función del neurotransmisor liberado, el mecanismo puede resultar *excitador* o *inhibidor* para la neurona "receptora".

En el soma de una neurona se integran todos los estímulos recibidos a través de todas las dendritas (Figura 2 (A)). Si como resultado se supera un potencial de activación, la neurona se "dispara" generando un impulso que se transmitirá a través del axón. Como puede observarse en la Figura 2(A), este impulso no es, ni mucho menos, constante, y la capacidad de reaccionar de la neurona varía con el tiempo hasta volver al estado de reposo. La comunicación tiene lugar a través de trenes de pulsos, por lo que los mensajes se encuentran modulados en frecuencia.

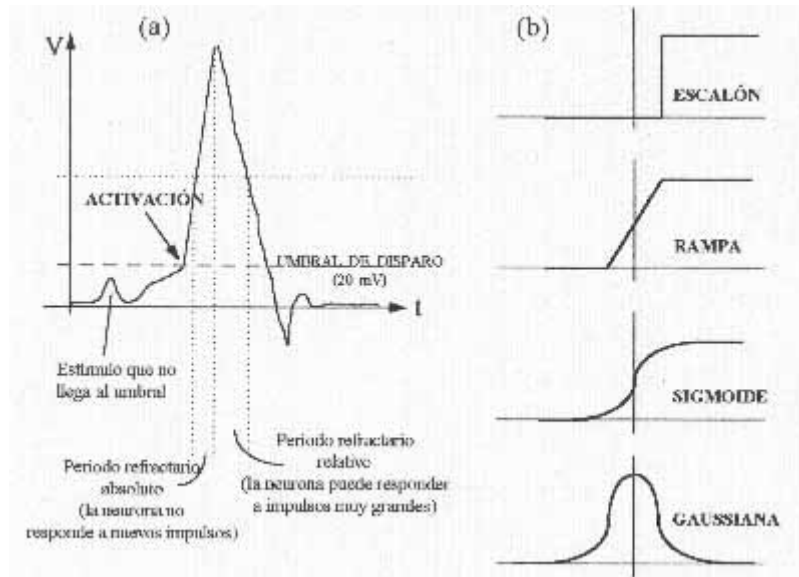


Figura 2. A) Respuesta de una neurona biológica. B) Distintas funciones de respuesta de una neurona artificial.

Tanto el salto electroquímico como la aparición de períodos refractarios, limitan mucho la velocidad de la neurona biológica y el rango de frecuencia de los

mensajes, que oscila entre unos pocos y algunos cientos de hertzios. Por ello, el tiempo de respuesta se ve limitado al orden de milisegundos, mucho más lenta que un circuito electrónico.

Los orígenes: La neurona de McCulloch y Pitts

En 1943 McCulloch y Pitts publicaron un trabajo en el que se hablaba, por primera vez, de las neuronas artificiales y de cómo éstas podrían realizar cálculos lógicos en redes neuronales. El modelo de neurona artificial propuesto consistía en una *unidad binaria de umbral* similar a la que se observa en la Figura 3 y en la que se pueden distinguir los siguientes elementos:

- Conjunto de conexiones de entrada.
- Valor de salida (binario).
- Función de proceso.
- Umbral de activación.

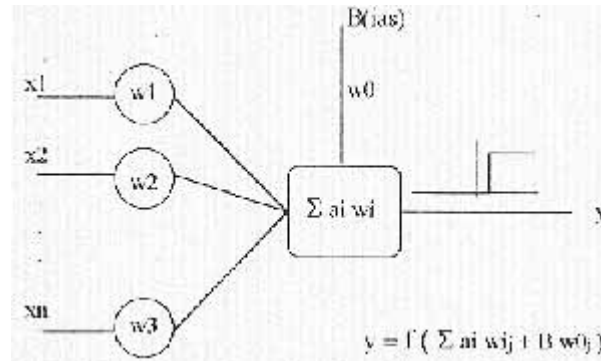


Figura 3. Neurona de McCulloch-Pitts

En definitiva, se trata de una unidad de proceso con n entradas, $x_1, \dots, x_n$ y una única salida (y), con dos únicos valores de salida posibles y constantes. La unidad realiza una *función de proceso* sobre los valores de las entradas, y si el valor obtenido supera un *umbral de activación*, se activa produciendo como respuesta el *valor de activación*. Por el contrario, si el resultado de aplicar la función de proceso a los valores de entrada no superara este umbral, la neurona permanecería inactiva y su salida sería nula.

La función de proceso de las entradas es, típicamente, una suma ponderada de los valores de las mismas, lo que significa que no todos los valores de entrada tienen igual aportación para la suma, sino que cada uno de ellos se modifica multiplicándolo por un valor de ponderación o peso, w_i , antes de proceder a sumar. Expresado de manera formal, el comportamiento de la neurona se correspondería con una función de la forma:

$$y = \theta \left[\sum_{i=1}^n w_i x_i - u \right] \quad [1]$$

donde representa a la función escalón; w_i son los pesos sinápticos y u el umbral de activación. Obsérvese que en función del signo de los pesos sinápticos, estos pueden actuar tanto como activadores (signo positivo) como inhibidores (signo negativo).

Como se puede observar, este modelo de neurona artificial presenta

numerosas analogías con las neuronas biológicas: los cables o conexiones son análogos a dendritas y axones, los pesos de ponderación de las conexiones equivalen a las sinapsis y el umbral de activación representa la actividad del soma.

En la practica, y por simplificar la notación y la programación, se suele emplear el "truco" de incluir el umbral de activación dentro del sumatorio. Para ello, se le modela como una entrada "fantasma" x_0 de valor siempre idéntico a la unidad, a la que se asigna un peso de $w_0=u$. De esta forma, la expresión queda como:

$$y = \theta \left[\sum_{i=0}^n w_i x_i \right] \quad [2]$$

similar a la anterior, con la salvedad de que, en esta ocasión , ha desaparecido el umbral, pero se consideran las entradas a partir del subíndice 1 y no del subíndice 0. Ahora bien, si se observa la expresión, no cuesta reconocer en

ella la del producto escalar de dos vectores. Por tanto, basta considerar el conjunto de n entradas de una neurona como un vector X de dimensión $n+1$ (recuérdese que se arrastra la entrada $x_0=1$), y el conjunto de pesos de la neurona como otro vector W de la misma dimensión, para que la expresión de la función de proceso de la neurona de McCulloch y Pitts quede reducida a

$$y = \theta[W * X] \quad [3]$$

donde el operador "asterisco" (*) representa el producto escalar de dos vectores.

Desde la publicación del trabajo de McCulloch y Pitts ha habido numerosas generalizaciones a su modelo de neurona, pero la más empleada consiste en la sustitución de la función de salida que, en lugar de realizarse a través de una función escalón, se suele sustituir por una función rampa, gaussiana o sigmoide (Figura 2(B), capítulo anterior).

La más empleada de todas es esta última, que corresponde a la expresión: $g(x)=1/[1+\exp(-bx)]$, donde b es el parámetro que define la pendiente de la curva. El Listado 1 define en C++ todo el código necesario para una neurona artificial como la expuesta. Obsérvese que las expresiones son tan simples porque tanto los pesos como las entradas se expresan en forma de vector. En el ejemplo, con fines didácticos, no se ha empleado el "truco" de la entrada número 0 y se indica explícitamente el valor de umbral. Como ilustración de la simplicidad del concepto, el Listado 1 es adecuado. Sin embargo, la neurona artificial del mismo es "tonta", ya que no es capaz de aprender. Su comportamiento dependerá de la forma en que se inicialice el vector de pesos. Por tanto, para que sea de utilidad, será necesario dotarla de la capacidad de aprender y generalizar

Listado 1. Neurona Artificial "tonta"

```
class neurona_t {
private:
    vector_t __pesos;
    float __umbral_activacion;
public:
    neurona_t(void):
        __pesos(NUM_ENTRADAS) {
        __umbral_activacion=0;
    }
    int salida(vector_t entrada {
        return(__pesos * entrada >
        __umbral_activacion) ? 1 : 0;
    }
}; //class neurona
```

Aprendiendo y generalizando

¿Cómo se enseña a un niño a reconocer lo que es un osito? Se le enseñan ejemplos de osito y se le dice "esto es un osito". Cuando el niño ve un "patito" y dice "osito", se le corrige y se le dice "esto es un osito". Durante esta etapa del aprendizaje del niño, se puede esperar que si se vuelven a presentar algunos de los ejemplos que ya han aparecido y se le pregunta "¿es esto un osito?", el niño responderá correctamente. Pero no sólo eso. A partir de cierto momento, si se le enseña al niño nuevos objetos, que no haya visto jamás antes, se podrá esperar que sea también capaz de identificar cuáles de ellos son ositos y cuáles no. Por supuesto, para que el aprendizaje sea efectivo, será necesario que el niño haya tenido un número suficiente de ejemplos y que estos hayan sido suficientemente representativos. No basta con enseñarle sólo un osito, ni un conjunto de ositos "picassianos". Por otra parte siempre habrá casos en los que no estará muy claro si determinado peluche es un osito o cualquier otro animal y es que, como diría Platón, intuimos la idea (el eidos) de "el osito" a partir de realizaciones concretas de osito, que no son sino la sombra de éste en la caverna.

La capacidad de generalización de los ejemplos concretos conocidos a la clase general, es uno de los ingredientes de la inteligencia. Pero, ¿cómo puede aprender una neurona artificial? El primero en responder a esta cuestión fue Donald O. Hebb, que de las observaciones de experimentos neurobiológicos dedujo que si las neuronas de ambos lados de una sinapsis se activan simultáneamente repetidas veces, el valor de la sinapsis se incrementa. Este principio es lo que se conoce como *postulado de aprendizaje de Hebb*. Así pues, el aprendizaje de una neurona y de una red neuronal se traduce en la adaptación de los pesos de sus conexiones sinápticas, para disminuir el error entre la salida producida ante una entrada y la salida correcta.

En el Listado 2, se puede observar una neurona ya provista de capacidad de aprendizaje gracias al método *entrenamiento_supervisado*, que acepta como argumentos un vector de entrada y una salida correcta, y devuelve 0 en caso de acierto y 1 en caso de fallo.

Listado 2. Neurona Artificial capaz de aprender

```
class neurona_t {
private:
    vector_t __pesos;
    float __umbral_activacion;
    float __tasa_aprendizaje;
public:
    neurona_t(void): __pesos(NUM_ENTRADAS)
    {
        __umbral_activacion=0;
        __tasa_aprendizaje=0.1;
    }
    int salida(vector_t entrada) {
        return (__pesos * entrada >
            __umbral_activacion) ? 1: 0;
    }
    int entrenamiento_supervisado(vector_t
    entrada, int salidaOK) {
        int resultado=salida(entrada);
        if (resultado != salidaOK) {
            if (resultado > salidaOK) {
                __pesos=__pesos -
                    (__tasa_aprendizaje * entrada);
                __umbral_activacion +=
                    __tasa_aprendizaje;
            }else{
                __pesos=__pesos +
                    (__tasa_aprendizaje * entrada);
                __umbral_activacion -=
                    __tasa_aprendizaje;
            }
            return 1;
        }else{
            return 0;
        }
    }
    float getUmbralActivacion (void) const {
        return __umbral_activacion; }
    float getTasaAprendizaje (void) const {
        return __tasa_aprendizaje; }
    vector_t getPesos (void) const {
        return __pesos; }
}; //class neurona_t
```

El perceptrón de Rossenblatt

El primero al que se le ocurrió una aplicación práctica para la neurona de McCulloch y Pitts, y aplicando las ideas de aprendizaje de Hebb, fue Frank Rossenblatt que a finales de los años 50 le iba dando vueltas a la idea de construir una máquina capaz de reconocer objetos de forma visual. De esta forma, en 1958 nació el *Perceptrón*, la primera red neuronal de la historia, compuesta por la astronómica cantidad de ¡1 neurona! El diseño del perceptrón consistía en una capa de elementos sensores, cuyas salidas se conectaban a las entradas de una neurona de McCulloch-Pitts (como se puede observar en la Figura 4) a través de detectores de umbral. El número de detectores de umbral era inferior al de elementos sensores, por lo que un detector podía conectarse a más de un sensor. La distribución de estas conexiones era aleatoria.

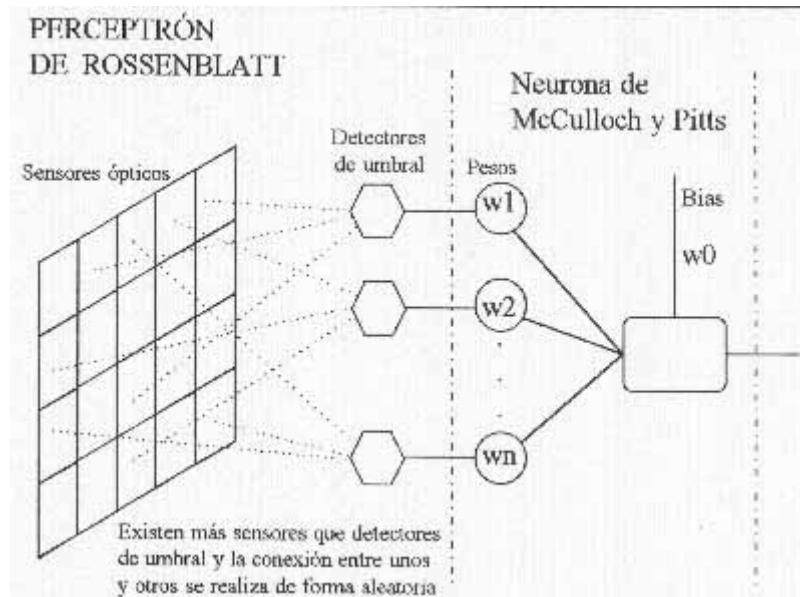


Figura 4. El perceptrón de Rossenblatt

El perceptrón generó gran interés no sólo por su capacidad de generalizar a partir de sus vectores de entrenamiento, sino también por el hecho de

comportarse correctamente, aún cuando sus entradas se conectaban de forma aleatoria. En primer lugar se procede a entrenar a la red neuronal de una única neurona. Si el entrenamiento finaliza con éxito (como se verá más adelante, Minsky y Papert demostraron que esto no es siempre posible), se aprovechará la capacidad de generalización del perceptrón para "pedirle" que clasifique nuevas entradas en función de lo aprendido.

Si se presenta a la red un vector de entrada que no se encuentra en el conjunto de entrenamiento, el perceptrón mostrará una capacidad de generalización, clasificándolo correctamente (si ha aprendido correctamente, claro está).

Aprendizaje del perceptrón.

El proceso de aprendizaje del perceptrón consiste en la presentación al mismo de un conjunto de vectores de aprendizaje. El perceptrón no aprende cuando su salida es correcta. Sin embargo, si es incorrecta se reajustan los pesos sinápticos de acuerdo a una regla de aprendizaje.

Se considera que el aprendizaje se ha completado cuando el perceptrón supera una pasada completa de todos los vectores de entrenamiento, sin cometer ningún error.

Rossenblatt desarrolló una regla de aprendizaje para el perceptrón, que consiste en actualizar los pesos de la siguiente forma:

$$w_i = w_i + n \cdot (d - y) \cdot x_i$$

donde w_i , es el peso asignado a la entrada i -ésima, x_i ; n es un valor denominado tasa de aprendizaje que se suele ajustar entre 0 y 1; d es el valor teóricamente correcto de la salida; e y y es el valor de salida que ha producido el perceptrón. La actualización del valor umbral, queda implícita en la ecuación anterior si se consideran las entradas y pesos de índice 0. De forma explícita:

$$u = u + n \cdot (d - y)$$

Siendo u , nuevamente, el umbral de activación.

De la tasa de aprendizaje dependerá la velocidad y la corrección con que el perceptrón aprenderá. A tasa mayores, mayor velocidad en principio, aunque

también mayor riesgo de producirse oscilaciones en el ajuste. La neurona del Listado 2 (capítulo III), aprende según la regla de aprendizaje de Rossenblatt. Rossenblatt también demostró, con su regla de aprendizaje, que si los patrones de entrenamiento se tomaban de dos clases *linealmente separables* (concepto que se explica en el próximo capítulo), el procedimiento de aprendizaje del perceptrón convergía en un número finito de pasos. A esto se le llama *teorema de convergencia del perceptrón*. En lenguaje simple y llano significa que "si el perceptrón puede aprender, con esta regla aprende".

Sin embargo, los patrones de entrenamiento no son siempre linealmente separables y, por tanto, existen categorías que un perceptrón no es capaz de aprender a clasificar. Esto es lo que Minsky y Papert demostraron, como se explica en el siguiente apartado.

La importancia de tener un buen profesor

Obsérvese que el conocimiento adquirido por una red neuronal es subsimbólico, ya que se encuentra en los pesos. Esto es un problema debido a que, si bien con redes simples es posible saber qué ha aprendido la red, con redes complejas apenas se puede obtener más que una compleja expresión matemática de su función de transferencia. Esto ha conducido a

situaciones problemáticas como la de hace algunos años en la que el ejército americano empleó una red neuronal para analizar las fotografías tomadas desde satélites espía. El objetivo era reconocer la presencia de tanques en las fotografías, pero cometieron la torpeza de emplear un conjunto inapropiado de fotografías de entrenamiento. Así la red aprendió a distinguir sin ningún error entre todas las fotografías utilizadas para el entrenamiento, aquellas que presentaban tanques de las que no los presentaban. Pero durante la fase posterior de operación comenzó a cometer muchos errores. Al final se descubrió que la red había aprendido a reconocer nubes, no tanques, ¡y todas las fotografías de entrenamiento que presentaban tanques, también presentaban nubes! Por tanto, la elección de los vectores de entrenamiento es un aspecto fundamental en el entrenamiento de una red neuronal.

Minsky y Papert: comienza la etapa oscura

Cuando la investigación sobre redes neuronales se ofrecía tan prometedora, Minsky y Papert publicaron un trabajo que sumiría esta línea de investigación poco menos que en el conjunto de las proscritas durante un período de veinte años, ya que demostraron muchas de las limitaciones del perceptrón. Entre estas limitaciones se encontraba la de no ser capaz de aprender una operación lógica tan básica como el *Or-exclusivo* o *XOR*, que da la casualidad de que es el operador de comparación. Así pues, un perceptrón no era capaz de indicar si sus entradas eran iguales o distintas.

Como se ha indicado, un perceptrón es una red neuronal de una única neurona. Por tanto, su función de salida se corresponderá con la de la fórmula 1 del capítulo II. Ahora bien, dado que la salida del perceptrón es binaria, o devuelve el valor de activación o devuelve el valor nulo. Por tanto, está claro que divide el espacio muestral de vectores de entrada en dos regiones. si el perceptrón tiene dos entradas, el vector de entrada tendrá dimensión dos, por tanto, todas las entradas se pueden representar como puntos en un plano, considerando la primera entrada como coordenada x y la segunda como coordenada y. La función de salida del perceptrón será:

$$\begin{aligned} &1 \text{ si } w_1*y_1 + w_2*y_2 - u > 0 \\ &0 \text{ si } w_1*y_1 + w_2*y_2 - u \leq 0 \end{aligned}$$

y la ecuación de la línea límite entre las dos regiones, con salida 0 y con salida 1, será, precisamente,

$$w_1*y_1 + w_2*y_2 - u = 0$$

la ecuación de una recta en el plano. Si el perceptrón tiene tres entradas, por el mismo razonamiento los vectores de entrada pueden considerarse coordenadas en el espacio y la división entre las dos regiones

correspondientes a cada posible salida, será un plano. En general, para cualquier número de entradas, el límite entre estas dos regiones viene determinado por la zona de transición de la función escalón, es decir:

$$\sum_{j=1}^n w_j x_j - u = 0$$

que se corresponde con un hiperplano de dimensión $n-1$ en un espacio vectorial de dimensión n . Esto es lo que se conoce como *condición de separabilidad lineal* del espacio de entrada Figura 5(A).

Esta condición nos dice que, para que un perceptrón pueda distinguir entre dos categorías, éstas deben ser *separables linealmente* o de lo contrario el

perceptrón será incapaz de aprender. Ahora bien, supóngase que se trata de entrenar un perceptrón para que clasifique entradas según dos categorías que se corresponden con una operación lógica tan simple como un XOR. Es decir, las entradas pertenecen a una categoría si ambas son iguales y a otra si ambas son distintas. Bien, fíjese el lector en el problema del XOR, representado en la Figura 5(B). Trate de separar las 2 categorías definidas por la tabla de verdad del XOR, mediante una única línea recta: imposible. El problema surge cuando se desea emplear un perceptrón para reconocer patrones y no se sabe, a priori, si estos son linealmente separables.

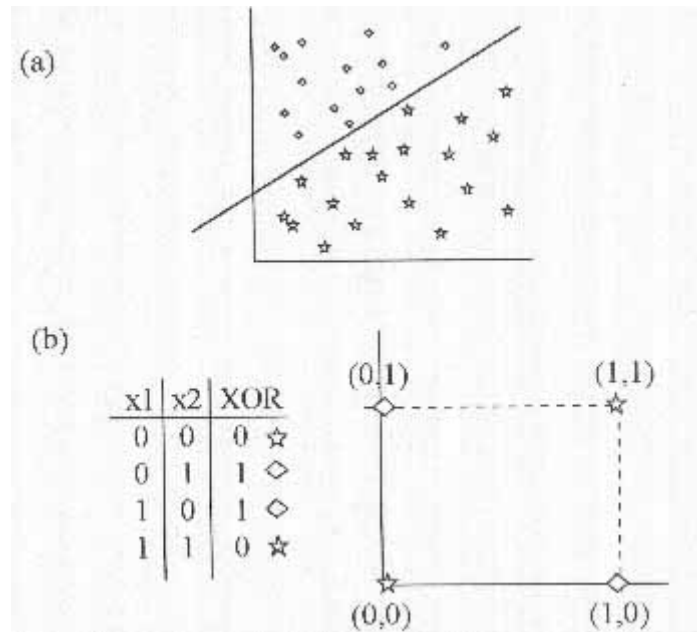


Figura 5. A) Condición de separabilidad lineal. B) El problema del XOR.

El problema del perceptrón no era difícil de encontrar. Es fácil demostrar que si se generaliza la idea del perceptrón a un perceptrón multicapa (con una segunda capa de neuronas, procesando la salida de la primera capa), es posible realizar cualquier operación lógica. El problema es que, al trabajar con redes multicapa, no se sabe sobre los pesos de qué neurona ni de qué capa recae la responsabilidad de los errores de la red. Así pues, no existía posibilidad de entrenar este tipo de redes neuronales. En este punto se quedó la investigación en temas de redes neuronales a finales de los años 60. Durante un período de 20 años, las instituciones apenas se atrevían a financiar ningún trabajo que girara alrededor de las mismas, ya que parecía un camino sin salida.

Sin embargo, a principios de los 80, esta disciplina resurgió de sus cenizas. Apareció el algoritmo de retropropagación, capaz de entrenar perceptrones multicapa, y con él una pléyade de nuevas arquitecturas y algoritmos de entrenamiento basadas en diversas ideas, desde la original de Rosenblatt, hasta la mecánica estadística. Y en este nuevo período de esplendor de la neurocomputación, se ha encontrado en otra disciplina de *soft-computing*, una

de las mejores formas de entrenar una red neuronal de cualquier grado de complejidad: el uso de algoritmos genéticos.

Fernando J. Echevarrieta (Programación Actual Nº 30)

Algoritmos genéticos

Los algoritmos genéticos (AG) proporcionan un método de aprendizaje basado en la analogía con la evolución de las especies. Los AG generan un conjunto de hipótesis mediante la mutación y recombinación de parte del conjunto de hipótesis conocido. En cada paso el conjunto de hipótesis conocido como “población actual” se renueva remplazando una proporción de esta población por los sucesores de las hipótesis más “adecuadas” (mediante el uso de una función de evaluación).

La popularidad de los AG se debe en parte a que la evolución es un método robusto y bien probado dentro de los sistemas biológicos naturales. Además son fácilmente paralelizables, lo que supone una ventaja gracias al abaratamiento actual de los costes en hardware. Por otra parte, los AG pueden realizar búsquedas en espacios de hipótesis que contienen complejas interacciones entre las distintas partes, donde el impacto de cada parte sobre la función de evaluación es difícil de especificar.

Aunque no se garantice encontrar la solución óptima, los AG generalmente encuentran soluciones con un alto grado de acierto.

El objetivo de los AG es buscar dentro de un espacio de hipótesis candidatas la mejor de ellas. En los AG la “mejor hipótesis” es aquella que optimiza a una métrica predefinida para el problema dado, es decir, la que más se aproxima a dicho valor numérico una vez evaluada por la función de evaluación.

El comportamiento básico de un algoritmo genético es el siguiente: de forma iterativa va actualizando la población de hipótesis. En cada iteración, todos los miembros de la población son procesados por la función de evaluación, tras lo cual una nueva población es generada. La nueva generación estará compuesta por:

- Las mejores hipótesis de la población actual (seleccionadas probabilísticamente)
- Y el resto de hipótesis necesarias para mantener el número, que se consiguen mediante el cruce de individuos. A partir de dos hipótesis padre (seleccionadas probabilísticamente a partir de la población actual) se generan dos hipótesis hijas recombinando sus partes siguiendo algún criterio establecido.

Una vez llegados a este punto (con una nueva población con el mismo número de individuos), a un determinado porcentaje de la población se le aplica un operador de mutación.

Representación de Hipótesis

Las hipótesis en los AG se suelen representar mediante cadenas de bits, de forma que puedan ser fácilmente manipulables por los operadores genéticos de mutación y cruce.

Primero veamos como usar una cadena de bits para representar los posibles valores de un atributo. Si un atributo puede tomar tres valores posibles (A,B y C), una manera de representarlo es mediante tres bits de forma que:

Atributo (100) = puede tomar el valor A

Atributo (010) = puede tomar el valor B

Atributo (001) = puede tomar el valor C

Atributo (110) = puede tomar el valor A ó B (A or B)

Atributo (111) = puede tomar el valor A, B ó C (A or B or C). No importa el valor del atributo.

De esta forma podemos representar fácilmente conjunciones de varios atributos para expresar restricciones (precondiciones) mediante la concatenación de dichas cadenas de bits. Ejemplo:

“Tiempo” puede ser Despejado, Nublado o Lluvioso.

“Viento” puede ser Fuerte o Moderado.

(Tiempo = Nublado ó Lluvioso) y (Viento = Fuerte) se representaría con la siguiente cadena: **011 10**.

Las postcondiciones de las reglas se pueden representar de la misma forma. Por ello una regla se puede describir como la concatenación de la precondición y la postcondición. Ejemplo:

“JugaralTenis” puede ser Cierto o Falso.

Si Viento = Fuerte entonces JugaralTenis = Cierto se representaría mediante la cadena **111 10 10**. Donde los tres primeros bits a uno indican que el atributo “Tiempo” no afecta a nuestra regla.

Cabe destacar que una regla del tipo 111 10 11 no tiene demasiado sentido, puesto que no impone restricciones en la postcondición. Para solucionar esto, una alternativa es codificar la postcondición con un único bit (1 = Cierto y 0 = Falso). Otra opción es condicionar los operadores genéticos para que no produzcan este tipo de cadenas o conseguir que estas hipótesis tengan una adecuación muy baja (según la función de evaluación) para que no logren pasar a la próxima generación de hipótesis.

Operadores Genéticos

Los dos operadores más comunes son la “mutación” y el “cruce”. El operador de cruce genera dos nuevos hijos a partir de dos cadenas padre recombinando sus bits. Para elegir con que bits contribuye cada padre hacemos uso de una “máscara de cruce”. Veamos un par de ejemplos: Cruce Simple:

Cadenas padre	Mascara de Cruce	Hijos
<u>11101</u> 001000		11101010101

	11111000000	
00001 <u>010101</u>		00001001000
Cruce Doble:		
Cadenas padre	Mascara de Cruce	Hijos
111 <u>10100</u> 1000		11001011000
	00111110000	
<u>00001010101</u>		00101000101

El operador mutación produce un nuevo hijo de un solo padre cambiando el valor de uno de sus bits elegido al azar. Generalmente se aplica después de hacer uso del operador cruce.

Otros operadores genéticos son “AddAlternative” y “DropCondition”. El primero de ellos cambia un bit de un atributo de 0 a 1, es decir, permite que el atributo

pueda tomar el valor representado por dicho bit. El segundo pone todos los bits de un atributo a 1, con lo que elimina la restricción impuesta por el atributo.

Una posibilidad interesante que surge con la aplicación de estos operadores, es la de incluir nuevos bits en las cadenas que representan las hipótesis y que indiquen que operadores pueden ser aplicados a dicha hipótesis (añadiendo un bit por operador). Como estos bits van a sufrir modificaciones a causa de los operadores aplicados a la cadena, estaremos alterando dinámicamente los métodos de búsqueda del algoritmo genético.

Función de evaluación y selección

La función de evaluación define el criterio para ordenar las hipótesis que potencialmente pueden pasar a formar parte de la siguiente generación.

La selección de las hipótesis que formarán parte de la siguiente generación o que serán usadas para aplicarles los operadores genéticos, puede realizarse de varias formas. Las más usuales son:

- Selección proporcional al ajuste dado por la función de evaluación. En este método la probabilidad de que una hipótesis sea seleccionada viene determinada por:

$$\Pr(h_i) = \frac{\text{Fitness}(h_i)}{\sum_{j=1}^p \text{Fitness}(h_j)}$$

Figura 1 - Probabilidad de que la hipótesis h_i sea elegida.

- Selección mediante torneo. Se eligen dos hipótesis al azar. La más “adecuada” (según la función de evaluación) tiene una probabilidad p (prefijada de antemano) de ser elegida, mientras que la otra tiene una probabilidad $(1 - p)$.
- Selección por rango. Las hipótesis de la población actual se ordenan de acuerdo a su adecuación. La probabilidad de que una hipótesis sea seleccionada será proporcional a

su posición en dicha lista ordenada, en lugar de usar el valor devuelto por la función de evaluación.

Búsqueda en el espacio de Hipótesis

Una de las dificultades que nos encontramos en algunos algoritmos genéticos es el problema del “crowding” (muchedumbre). Se trata de un fenómeno por el cual las mejores hipótesis se reproducen rápidamente de manera que las nuevas generaciones una gran proporción se debe a éstas hipótesis y a otras muy similares (descendientes), reduciendo así la diversidad de la población, y por lo tanto, las posibilidades de la evolución. Para reducir los efectos del “crowding” se usan varias estrategias. Una solución consiste en cambiar el método de selección habitual (selección proporcional al ajuste) por alguno de los otros vistos con anterioridad (selección mediante torneo o selección por rango). Otra opción consiste en usar otra función de evaluación, “ajuste compartido”, de manera que el valor devuelto por esta función se devalúa ante la presencia de otras hipótesis similares en la población. Y una tercera alternativa es restringir el tipo de hipótesis a los que se les permite la recombinación.

Programación Genética

La programación genética (PG) es una forma de computación evolutiva en la que los individuos de la población son programas, en lugar de cadenas de bits.

Representación de Programas

Los programas usados en la programación genética suelen representarse mediante sus árboles sintácticos. En esta notación, cada llamada a una función se representa por un nodo en el árbol, y los argumentos de la función corresponden con los nodos hijos de éste. Para aplicar programación genética a un dominio particular, es necesario que el usuario defina de antemano las primitivas que se van a emplear así como el tipo de los nodos hoja. Por lo tanto, la programación genética realiza una búsqueda evolutiva en un espacio de programas descritos mediante sus árboles sintácticos.

De igual forma que los algoritmos genéticos, la programación genética itera sobre una población de individuos produciendo una nueva generación mediante el empleo de la selección, el cruce y la mutación. La adecuación de

un programa viene determinada por los resultados de su ejecución sobre unos datos de entrenamiento (función de evaluación).

Así, por ejemplo, el operador de cruce se implementa mediante el intercambio de subárboles entre programas padres.

En la mayoría de los casos, el rendimiento de la programación genética depende básicamente de la representación elegida y de la elección de la función de evaluación.

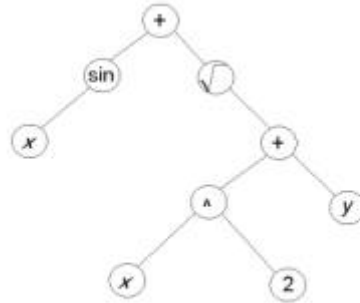


Figura 2 - Representación de un programa mediante su árbol.

Modelos de evolución y aprendizaje

En muchos sistemas naturales, los individuos aprenden a adaptarse durante sus vidas. Al mismo tiempo distintos procesos biológicos y sociales permiten a la especie adaptarse a lo largo de las distintas generaciones, por lo que surge una interesante pregunta: ¿Cuál es la relación entre el aprendizaje individual de un organismo durante su vida y el aprendizaje colectivo de la especie?

A esta cuestión intentan responder la teoría de la evolución Lamarckiana y la teoría del efecto Baldwin.

Evolución Lamarckiana

Lamarck (científico de finales del siglo XIX) propuso que la experiencia de un individuo afecta directamente a su descendencia, por lo que el conocimiento sería hereditario y las siguientes generaciones no necesitarían adquirirlo. Se trata de una conjetura que mejora la eficiencia de los algoritmos genéticos y la programación genética, en las cuales se ignoraba la experiencia adquirida por el individuo durante su vida.

Aunque biológicamente esta teoría no es correcta como modelo de evolución, sin embargo si es perfectamente aplicable al caso de la computación genética, donde recientes estudios han demostrado su capacidad para mejorar la efectividad.

El efecto Baldwin

Se trata de otro mecanismo que sugiere como el aprendizaje de un individuo puede alterar el curso de la evolución de la especie. Se basa en las siguientes afirmaciones:

- Si una especie evoluciona en un entorno cambiante, entonces los individuos capaces de aprender durante su vida se verán favorecidos. De hecho, la habilidad para aprender permite a los individuos maximizar su capacidad de adaptación en el entorno.
- Los individuos con capacidad de aprendizaje dependerán en menor medida de su código genético. Como consecuencia de lo anterior las nuevas generaciones poseerán una diversidad genética mayor, lo que permitirá una evolución más rápida. En resumen, la capacidad de aprendizaje de algunos individuos provoca indirectamente una aceleración en la evolución de toda la población.

Paralelismo en Algoritmos Genéticos

Los algoritmos genéticos son fácilmente paralelizables. A continuación se muestran las posibilidades de paralelismo con AG's:

- Grano Grueso

Se subdivide la población en una serie de grupos distintos (siguiendo algún criterio), llamados “demes”. Cada uno de estos grupos se asigna a un nodo de computación distinto, y a continuación se aplica un AG en cada nodo. Los operadores de cruce se aplicaran generalmente entre individuos del mismo grupo, y en menor porcentaje, entre individuos de distintos grupos. En este entorno surge un nuevo concepto, denominado migración, que se produce cuando un individuo de un grupo se copia o se traslada a otro/s. Un beneficio del modelo de grano grueso es que se reduce el efecto “crowding” que aparecía en los AG no paralelizados.

- Grano Fino

Se asigna un procesador a cada individuo de la población. La recombinación se efectúa entre individuos vecinos en la red de

computación. Algunos ejemplos de redes de procesadores (especifican las reglas de vecindad) son la malla, el toroide, etc.

Ejercicio

Diseñar un algoritmo genético que aprenda a clasificar conjunciones de reglas ara el problema “Jugar al Tenis” descrito en el capítulo 3 [nota]. Describe detalladamente la codificación de la cadena de bits para las hipótesis y el conjunto de operadores de cruce.

La expresión que soluciona el problema es la siguiente:

(Tiempo=Soleado y Humedad=Normal) o

(Tiempo=Nublado) o

(Tiempo=Lluvioso y Viento=Debil)

Los posibles valores de los atributos son:

Tiempo=(Soleado,Nublado,Lluvioso)

Humedad=(Alta,Normal)

Viento=(Fuerte,Debil)

JugarTenis=(Si,No)

Por lo tanto, para la cadena de bits de las hipótesis usaremos tres bits para “Tiempo”, dos para “Humedad”, dos para “Viento” y uno para “JugarTenis”. Así, por ejemplo:

Si (Tiempo=Soleado y Humedad=Normal y Viento=Fuerte) -> JugarTenis=Si
se expresaría como: 100 01 10 1

Como la solución buscada es la disyunción de tres de estas reglas, las cadenas de bits de nuestras hipótesis estarán formadas por la concatenación de tres cadenas. Ejemplo:

100 01 10 1 + 010 10 10 1 + 100 10 01 1

En cuanto al operador de cruce, elegiremos uno de tipo uniforme con la siguiente máscara:

Máscara de cruce = 11000011 11000011 11000011

Para elegir las hipótesis que pasarán a la siguiente generación, o que serán elegidas para sufrir las operaciones de cruce, usaremos una selección probabilística tal que: ver Figura 1

A la hora de implementar el algoritmo, haremos uso de los siguientes parámetros:

r = % población que sufrirá la operación de cruce

p = número de hipótesis de la población

m = % de individuos (hipótesis) que mutan

Durante las iteraciones del algoritmo:

- a) $(1-r)p$ pasarán a la siguiente generación (P_s) intactos
- b) $(r*p)/2$ parejas de hipótesis realizarán la operación de cruce
- c) Y el m por ciento de la “nueva” población sufrirán mutaciones. Este m por ciento se elige con una probabilidad uniforme.

Función Fitness (función de evaluación)

Dispondremos de una base de datos de entrenamiento, compuesta por cadenas de 7 bits tal que nos indique que condiciones habrán de darse para jugar o no al tenis. Por ejemplo:

100 01 10 1 = (Tiempo=Soleado y Humedad=Normal) -> Si

En nuestro ejemplo, el número de combinaciones posible es $3 \times 2 \times 2 = 12$ (lo cual no es mucho y nos permite tener una muestra de entrenamiento completa). El objetivo de la función es el de comprobar el porcentaje de reglas de entrenamiento que nuestra hipótesis es capaz de clasificar correctamente para, de esta forma, dotar a cada individuo de un valor numérico indicativo de su precisión (ajuste al modelo buscado).

Nuestra hipótesis (individuo) se puede representar en tres partes (A,B y C) donde Z es el bit que indica si se juega o no al tenis en cada caso.

A	B	C
X X X X X X X X X	X X X X X X X X X	X X X X X X X X X
Z	Z	Z

Nuestra función de evaluación tomará una hipótesis y la procesará con todo el conjunto de entrenamiento. Una cadena de entrenamiento se representa de la siguiente forma (donde R es el bit que indica si se juega o no al tenis):

Entrenamiento

X X X X X X X X

Data (7 bits) R

Esta cadena de entrenamiento se compara con cada una de las partes de la hipótesis para ver si alguna de las tres la clasifica correctamente. Para saber si una parte clasifica bien una regla de entrenamiento procederemos de la siguiente forma:

Si $R=1$ entonces

Si ((Entrenamiento AND A) = Entrenamiento)
devolver SI

Sino devolver NO

Si $R=0$ entonces

Si ((Data AND A) = Data)
Si $R=Z$ devolver SI

Sino devolver NO

Sino devolver NO

Si se ha devuelto SI, procedemos con el siguiente dato de entrenamiento. Si se ha devuelto NO, evaluamos la siguiente parte del individuo (B y después C). Para cada individuo, la función de evaluación realiza esta tarea con todos los datos de entrenamiento, devolviendo un número entre 0 y 1 que indique el porcentaje de clasificaciones correctas de la hipótesis actual (0% a 100%). Así pues, cada individuo estará asociado a un valor numérico que usará posteriormente la función de selección.

Función AG (Fitness, Umbral, p, r, m)

```
{
    Inicializar población P. Para
    cada hipótesis en P, calcular Fitness.
    Mientras que max(Fitness) < Umbral hacer
        Seleccionar (1-r)p individuos de P que
        pasan a Ps.
        Elegir (r*p)/2 parejas de P y aplicarles
        operadores de cruce. Añadir los hijos a
        Ps.
        Elegir m porciento sobre Ps y aplicar
```

```
        mutación.  
        P                <-                Ps  
        Para cada hipótesis en P, calcular  
        Fitness.  
    Fin                                mientras  
    Devolver la hipótesis con mayor fitness.  
}
```

En nuestro caso el Umbral será 1 (100% de clasificaciones correctas) ya que disponemos en la base de entrenamiento de todas las combinaciones posibles (algo que no ocurrirá en problemas reales y no de juguete, como es éste)

Nota: Se trata de un ejemplo sencillo en el que una determinada persona juega al tenis siempre y cuando se cumplan unas determinadas condiciones atmosféricas.

Como hemos podido observar, la principal ventaja de los algoritmos genéticos radica en su sencillez. Se requiere poca información sobre el espacio de búsqueda ya que se trabaja sobre un conjunto de soluciones o parámetros codificados (hipótesis o individuos). Se busca una solución por aproximación de la población, en lugar de una aproximación punto a punto. Con un control

adecuado podemos mejorar la aptitud promedio de la población, obteniendo nuevos y mejores individuos y, por lo tanto, mejores soluciones.

Se consigue un equilibrio entre la eficacia y la eficiencia. Este equilibrio es configurable mediante los parámetros y operaciones usados en el algoritmo. Así, por ejemplo, bajando el valor del umbral conseguiremos una rápida solución a cambio de perder en “calidad”. Si aumentamos dicho valor, tendremos una mejor solución a cambio de un mayor tiempo consumido en la búsqueda. Es decir, obtenemos una buena relación entre la calidad de la solución y el costo.

Quizás el punto más delicado de todo se encuentra en la definición de la función de evaluación. Al igual que en el caso de la heurística, de su eficacia depende el obtener buenos resultados. El resto del proceso es siempre el mismo para todos los casos.

La programación mediante algoritmos genéticos suponen un nuevo enfoque que permite abarcar todas aquellas áreas de aplicación donde no sepamos como resolver un problema, pero si seamos conscientes de que soluciones son buenas y cuales son malas. Desde aplicaciones evidentes, como la biología o

la medicina, hasta otros campos como la industria (clasificación de piezas en cadenas de montaje). Los algoritmos genéticos poseen un importante papel en aplicaciones de búsqueda y optimización, pero desde nuestro punto de vista, es en el aprendizaje automático donde encuentra un estupendo marco de trabajo. La capacidad que poseen para favorecer a los individuos que explican bien los ejemplos, a costa de los que no lo hacen, consigue una nueva generación con mejores reglas y, por lo tanto el sistema será capaz de ir aprendiendo a conseguir mejores resultados.

© Gastón Crevillén y David Díaz, 2001

Bibliografía: "Machine Learning" (Tom M. Mitchell)