1 Introducción

1.2 Modelos de Computación: de la Computabilidad Clásica a las Redes Neuronales

La mayoría de los ordenadores actuales tienen tres componentes: la unidad central de proceso (CPU), la memoria y dispositivos de entrada y salida. La unidad central de proceso se ocupa de realizar las computaciones aritméticas y las decisiones lógicas a partir de los datos de entrada. La memoria es el dispositivo encargado de guardar información, debe tener una capacidad suficiente para recoger toda la información generada por la CPU, y los dispositivos de entrada y salida son los medios para recoger la información generada y comunicarla al mundo exterior, como puede se una pantalla o una impresora.

1.2.1 La computabilidad clásica

La teoría clásica de la Computabilidad trata de determinar qué problemas algorítmicos se pueden resolver por ordenador en condiciones ideales de disposición ilimitada de memoria y tiempo. Las máquinas de Turing han sido aceptadas como modelos estándar para el estudio de la computabilidad durante más de medio siglo. Comenzaremos analizando el modelo más simple de máquinas de estados finitos.

Las máquinas de estados finitos fueron introducidas formalmente por Rabin y Scott (1959). Una **máquina de estados finitos** consta de un dispositivo memoria primitiva representado por una colección finita de estados y de una función de transición que actualiza el estado actual como una función del estado previo y de la entrada en curso. Se supone que toda la información viene codificada por cadenas de símbolos (caracteres) de un alfabeto fijo, que constituyen la entrada a la máquina. Una formalización sencilla de una máquina de estados finitos es el autómata finito.

Un **autómata finito determinístico** es una quíntupla $M = (K, \Sigma, \delta, s, F)$, en la que

K es un conjunto finito de **estados**
Σ es un alfabeto
$s \in K$ es el **estado inicial**
$F \subseteq K$ es el conjunto de estados **finales**
δ es una función de $K \times \Sigma$ en K, llamada **función de transición**.

Un autómata finito determinístico funciona de la siguiente manera: la información que recibe la máquina viene codificada en símbolos de un alfabeto que se escriben en una **cinta de entrada** divida en cuadrados (en número ilimitado) y se escribe un símbolos en cada cuadrado (casilla). La parte principal de la máquina es una “caja negra”, llamada **unidad de control finito**, que presenta, en cada momento específico, un estado concreto del conjunto de estados posibles K. La unidad de control finito puede conocer el símbolo escrito en la casilla correspondiente de la cinta de entrada mediante una cabeza de lectura movible. Inicialmente, la cabeza grabadora se coloca al comienzo de la cinta y la unidad de control finito se encuentra en el estado inicial. A continuación el autómata lee el símbolo de la primera casilla y la unidad de control finito pasa a un nuevo estado que viene

especificado por su función de transición; el nuevo estado depende del estado previo y del símbolo leído. Entonces la cabeza de lectura se mueve una casilla a la derecha y lee el símbolo escrito en dicha casilla. Este proceso se va repitiendo a intervalos regulares, se lee un símbolo, la unidad de control finito actualiza su estado y la cabeza de lectura pasa a la casilla siguiente, así sucesivamente hasta que la cabeza de lectura llega al final de la cadena de símbolo escritos en la cinta de entrada. El autómata **acepta** (aprueba),o no (desaprueba), la cadena de símbolos de entrada (palabra) según que el estado que presenta la unidad de control finito sea, o no, un estado del conjunto final de estados *F*. El conjunto de palabras (cadenas de símbolos) que acepta un autómata finito determinístico constituye el **lenguaje aceptado** por la máquina.

Ahora surge la siguiente cuestión: ¿qué propiedades tienen los lenguajes aceptados por un autómata finito determinista? Se puede demostrar que son cerrados bajos las siguientes operaciones unión, intersección, complementación, concatenación y la operación estrella de Kleene. Por otra parte, la clase de los **lenguajes regulares** (o conjuntos regulares) sobre un alfabeto Σ es el conjunto minimal de lenguajes que contienen a ϕ, a todos los conjuntos formados por un solo símbolo, *{a}*, para $a\in\Sigma$, y son cerrados bajo las operaciones de unión, concatenación y estrella de Kleene. Así, un resultado importante (ver Lewis y Papadimitriou) es que un lenguaje es regular siempre y cuando sea aceptado por un autómata finito.

Una posible generalización de los autómatas finitos determinísticos es sustituir la función de transición δ por una **relación de transición** Δ que es un subconjunto finito de $K\times\Sigma^*\times K$, donde Σ^* es el conjunto de todas las cadenas finitas que se pueden formar con símbolos del alfabeto Σ, incluyendo la cadena vacía. Es decir, ahora se permiten varios estados siguientes para un símbolo de entrada dado y un estado previo de la unidad de control. A este tipo de máquinas se le llama **autómatas finitos no deterministas**. Sin embargo, estas maquinas son equivalentes a los autómatas finitos determinísticos, es decir, aceptan los mismos lenguajes. Además, para cada autómata finito no deterministico se puede construir un autómata finito determinístico equivalente.

Sin embargo, los autómatas finitos determinísticos no se pueden considerar como modelos verdaderamente generales para el diseño de computadores puesto que no son capaces de reconocer lenguajes tan simples como $L = \{ a^n b^n c^n : \ n\in Z_+ \}$. Por ello, es necesario introducir nuevos mecanismos que puedan reconocer estos lenguajes y lenguajes mucho más complicados. Uno de estos dispositivos más potentes es la máquina de Turing que fue ideada por Alan Turing (1912-1954). La máquina de Turing consta de una unidad de control finito, de una cinta y de una cabeza grabadora que puede usarse para leer o para escribir sobre la cinta. Una máquina de Turing no determinística es una cuádrupla $M = (K, \Sigma, \delta, s)$, en la que

- K es un conjunto finito de **estados**, que no contiene al estado de parada *h*.
- Σ es un alfabeto que contiene el símbolo blanco #, pero no contiene los símbolos *L* y *R*.
- $s\in K$ es el **estado inicial**
- δ es una función de $K\times\Sigma$ en $(K\cup\{h\})\times(\Sigma\cup\{L,R\})$, llamada **función de transición**.

La máquina de Turing funciona de la siguiente manera:

1. La información de entrada viene constituida por un conjunto de símbolos de entrada colocados al comienzo de la cinta, uno por casilla, y las demás casillas tienen inicialmente el símbolo blanco. La máquina puede ir alterando sus entradas escribiendo sobre ellas nuevos símbolos y escribiendo símbolos sobre las demás casillas que contienen símbolos blancos. La máquina podrá también escribir la respuesta sobre la cinta y leerla al final de la computación y así no necesita un conjunto de estados finales.
2. La unidad de control opera a intervalos regulares, es decir, en pasos discretos; en cada paso realiza dos funciones, dependiendo del símbolo leído en la cinta y de su estado actual, pasa a un nuevo estado especificado por su función de transición, y escribe un símbolo en el cuadrado de la cinta donde está situada la cabeza grabadora o mueve la cabeza grabadora a la casilla de la izquierda (*L*) o a la casilla de la derecha (*R*).
3. La máquina se parará solamente cuando alcanza el estado de parada o intenta moverse a la izquierda del comienzo de la cinta.

Para estudiar si una función es o no computable, utilizaremos la máquina de Turing. Consideremos dos alfabetos Σ_o y Σ_1 y una f definida de Σ_0^* a Σ_1^*. Un máquina de Turing, $M = (K, \Sigma, \delta, s)$, se dice que **computa la función** f si Σ_o, $\Sigma_1 \subseteq \Sigma$ y para cualquier entrada $w \in \Sigma_0^*$ la máquina se para y en las celdas de salida escribe la cadena de símbolos *u*, siendo *f(w)=u.* Si existe una máquina de Turing que computa la función f se dice que f es computable según Turing.

Si Σ_o es un alfabeto que no contiene el símbolo blanco, ni los símbolos fijos *Y* y *N*, entonces diremos que un lenguaje $L \subseteq \Sigma_0^*$ es **decidible** según Turing si la función indicador

$$f_L : \Sigma_0^* \to \{Y, N\}$$

definida por la expresión:

$$f_L(w) = \begin{cases} Y & \text{si } w \in L \\ N & \text{si } w \notin L \end{cases}$$

es computable según Turing. También se dice que *M* decide *L* o que *M* es un procedimiento de decisión para *L*.

También una máquina de Turing se puede utilizar como un aceptador de lenguajes. Si Σ_o es un alfabeto que no contiene el símbolo blanco, diremos que una maquina de Turing *M* acepta una palabra $w \in \Sigma_0^*$ si *M* se para con dicha palabra. Asimismo, diremos que *M* acepta el lenguaje $L \subseteq \Sigma_0^*$ si y sólo si $L = \{ w \in \Sigma_0^* : M \text{ acepta } w \}$.

Ahora surge la pregunta ¿cuál es la clase de las funciones computables según Turing? la respuesta está clara, es la clase de las funciones μ-**recursivas**.

Una posible generalización de la máquina de Turing es la máquina de **Turing no determinística**, en la que se sustituye la función de transición por una relación de transición que asigna varios estados posibles a la unidad de control para un mismo símbolo de entrada leído y un mismo estado previo. Sin embargo, se demuestra que cualquier lenguaje aceptado por una máquina de Turing no determinística es aceptado también por una cierta máquina de Turing deterministica.

Por otra parte, si las palabras (cadenas de símbolos) se utilizan para representar lenguajes, no podemos representar a todos los lenguajes, puesto que sólo disponemos de un conjunto numerables de palabras para cada lenguaje y sin embargo hay lenguajes no numerables. Las máquinas de estados finitos son objetos que se puede utilizar para especificar lenguajes. Sin embargo, con ellas sólo conseguimos especificar, es decir, decidir o aceptar, un conjunto muy reducido de los lenguajes posibles.

Hay problemas acerca de las máquinas de Turing que no admiten solución algorítmica del tipo Turing. Si *D* es un diccionario infinito que da respuesta a cada cuestión del tipo: ¿acepta (es decir, consigue parar) la máquina de Turing *M* la entrada *w*? No hay máquinas de Turing que decidan dicho lenguaje D. A este problema se le conoce con el nombre de **problema de la parada** para máquinas de Turing y consiste en determinar, para una máquina de Turing arbitraria *M* y una entrada dada *w*, si *M* parará alguna vez partiendo de dicha entrada.

También se han propuesto otras máquinas diferentes a la de Turing, como las máquinas RAM (memorias de registros direccionables), los algoritmos de Markov, los sistemas de Post, las gramáticas formales de Chomsky, etc. Todas ellas son equivalentes computacionalmente a la máquina de Turing. Ello nos lleva a la tesis de Church-Turing: *Cualquier algoritmo se puede implementar en una máquina de Turing*.

Sin embargo, a continuación vamos a ver que el problema de la parada es resoluble neuronalmente.

1.2.2 Computación sobre grafos: Redes de autómatas, autómatas celulares y RNA.

Una **red de autómatas** consiste en un conjunto de elementos de proceso, que van a ser máquinas de estados finitos, localizados en los vértices de un digrafo (grafo dirigido), uno en cada vértice. Cada elemento de proceso recibe su entrada de las unidades vecinas y comunica su salida a sus unidades vecinas. Formalmente, una red de autómatas es un par $A=\langle C,\{M_i\}\rangle$ constituido por un **espacio celular** *C=(G,{Qi})*, y una familia de máquinas de estados finitos M_i (sólo un número finito de ellas pueden ser distintas) con alfabeto de entrada $\Sigma_i = Q_{i_1} \times \ldots \times Q_{i_{d_i}}$ y función de transición local

$$\delta_i : Q_i \times \Sigma_i \rightarrow Q_i$$

que aplica $(x_i, x_{i1},..., x_{id_i})$ en $\delta_i(x_i, x_{i1},..., x_{id_i})$, donde $i_1,..., i_{d_i}$ son los nodos (vértices) conectados con el nodo *i*. El espacio celular *C* está formado por un dígrafo *G* numerable, pero localmente finito, es decir, sólo un número finito de arcos entran o salen de cada vértice, cuyos vértices se llaman *células* o nodos, y por una familia de subconjuntos finitos Q_i de un conjunto *Q* que nos dan los estados o niveles de activación posibles de cada célula *i* (todos contiene el estado inactivo *0*).

Una red de autómatas opera localmente de la siguiente manera: Una copia de una máquina de estados finitos M_i, llamada elemento de proceso, ocupa cada vértice *i* del grafo *G*, que es por ello llamado célula o nodo. Cada copia M_i recibe como entrada los estados que presentan las células vecinas y su propio estado y entonces actualiza su estado según la dinámica local establecida por su función de transición δ_i. La red repite esta actualización de los estados de sus células repetidas veces.

Una **red neuronal** (discreta) es una red de autómatas donde cada una de sus células actualizan sus estados aplicando una función de activación a una suma ponderada de las entradas (estados) que recibe de las células vecinas. En una red neuronal,

- cada arco *(i,j)* del dígrafo que conecta el vértice *i* con el vértice *j* tiene asociado un peso w_{ij}, llamado peso sináptico;
- la entrada neta de cada unidad celular viene dada por una suma ponderada de los estados que presentan las unidades celulares vecinas y así el nuevo estado de la unidad celular *i*;
- La actualización de los estados se realiza según una función de activación f_i ,

$$f_i : \mathrm{A} \to \mathrm{A}$$

que verifica que $f_i(0)=0$ (para evitar generación espontánea de la activación), siendo A el conjunto de estados (niveles de activación) que puede tomar las unidades celulares. Así, el nuevo estado de la unidad celular *i*, en el tiempo *t*+1, viene dado por la expresión:

$$x_i(t+1) = f_i\left(\sum_j w_{ij} x_j(t)\right) \qquad (1)$$

Formalmente, una **red neuronal** (discreta) es un tripleta $\mathtt{N}=\langle \mathrm{A},\mathrm{G},\{f_i\}\rangle$ constituida por un conjunto A de estados posibles finito (valores de activación), que admite una estructura aditivo-multiplicativa (permite la suma de estados ponderados), un dígrafo numerable G, pero localmente finito, y una familia de funciones de activación, $\{f_i\}$, una función de activación para cada unidad celular. La dinámica local de computación de la red viene dada por la expresión (1).

Un **autómata celular** es una red de autómatas definida sobre un grafo regular con una máquina de estados finita idéntica para cada unidad celular. Generalmente, los autómatas celulares están definidos sobre retículos (enrejados) Euclídeos, de una o dos dimensiones

(ver la figura 1.3.2), que forman un grafo homogéneo y sobre cuyos puntos enteros se colocan las unidades celulares. Las conexiones entre vértices son los ejes de la rejilla. Formalmente, un autómata celular es una tripleta $M=\langle C,N,M\rangle$ constituida por un espacio celular $C=(\Gamma,Q)$ construido sobre un grafo de Cayley, Γ, un conjunto finito N de vértices de Γ, una copia de una máquina de estados finitos M en cada vértice i con alfabeto de entrada $Q^d = Q\times...\times Q$ (d veces) y una función de transición local

$$\delta:\ Q\times Q^d \rightarrow Q$$

$$(x_i, x_{i+i_1},...,x_{i+i_d}) \rightarrow \delta((x_i, x_{i+i_1},...,x_{i+i_d})$$

donde $N = \{i_1,...,i_d\}$. La dinámica global de un autómata celular viene definida por la siguiente expresión:

$$T(x_i) = \delta(x_i, x_{i_1},...,x_{i_d}).$$

Garzon y Franklin (1989) demostraron que cada autómata celular es una red neuronal. Además, demostraron que el problema de la parada de una máquina de Turing es resoluble mediante una red neuronal. Este resultado puede parecer un poco sorprendente y puede creerse que constituye un contraejemplo para la tesis de Churh-Turing, pero no es así, ya que dicha tesis se refiere a la resolución algorítmica de problemas por medios finitos, mientras que la red neuronal que se emplea en la demostración requiere la utilización de un objeto infinito (el dígrafo G). Sin embargo, aunque utiliza un conjunto infinito de unidades celulares estás operan sobre estados finitos.

Asimismo, hay en la literatura varias implementaciones diferentes de una máquina de Turing mediante redes neuronales. Por otra parte, si nos limitamos a redes neuronales acotadas, es decir, con un número finito de unidades celulares y de valores de activación, éstas se pueden modelar mediante un autómata finito M, pensando en las configuraciones como el conjuntos de estados del autómata finito. La función de transición de M vendrá determinada por el procedimiento de actualización de la red neuronal. Así, las redes neuronales finitas son equivalentes computacionalmente a los autómatas finitos.

Finalmente, nos preguntamos si hay problemas que no sean resolubles por redes neuronales infinitas. Un problema no resoluble por redes neuronales infinitas es el problema de la estabilidad (Garzon y Flanklin, 1989). Una cuestión todavía en estudio es ver qué se puede computar con redes neuronales discretas ¿definen un nuevo dominio de computabilidad diferente al de los modelos clásicos?

1.3. La computación celular

La arquitectura de von Neumann, que se basa en el principio de un procesador complejo que realiza una tarea en cada momento, ha dominado la tecnología de la computación en los últimos 50 años. Sin embargo, una nueva forma de computación está emergiendo basada en principios completamente diferentes, que podemos llamar Computación Celular. Este nuevo paradigma computacional suministra nuevas formas de hacer la computación más eficiente (en términos de velocidad, coste, disipación, almacenamiento y calidad de la solución) para tratar grandes problemas en dominios de aplicación específicos.

La computación celular se basa en tres principios:

- Simplicidad
- Paralelismo inmenso
- Localidad

La Computación Celular utiliza unidades de proceso simples que por sí solas pueden hacer muy pocas cosas, frente a un procesador de propósito general que realiza tareas muy complicadas. Así, la unidad de proceso de una red neuronal es sencilla y no lo es un procesador Pentium IV.

Por otra parte, frente al paralelismo masivo que usa cientos de procesadores, la computación celular utiliza millones de unidades de proceso, que generalmente se expresan en forma exponencial, 10^n.

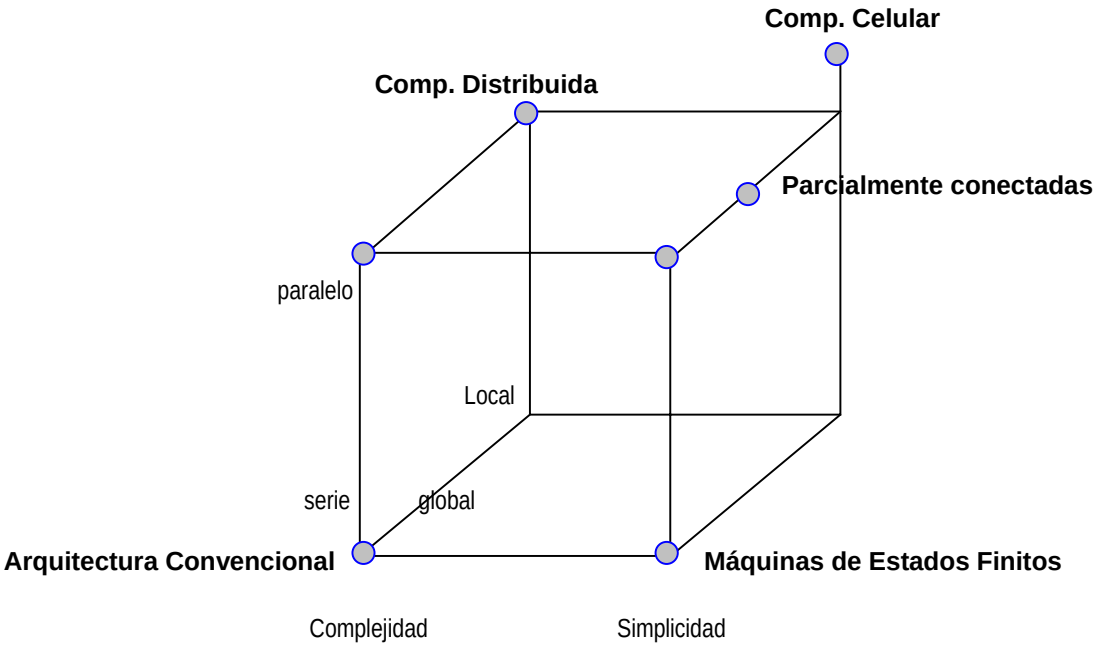

Figura 1.3.1. Cubo de representación de los tipos de computación.

La Computación Celular viene también caracterizada por la conectividad local de las unidades de proceso de manera que cada unidad de proceso sólo se comunica con un reducido número de unidades de proceso más o menos cercanas a ella. Además, las líneas de conexión son portadoras de una pequeña cantidad de información. El principio de localidad conlleva que ninguna unidad de proceso tenga una visión global del sistema; no hay un controlador central.

Por lo tanto, podemos decir esquemáticamente que (ver la figura 1.3.1):

Computación Celular = Simplicidad + Paralelismo inmenso + Localidad

Algunos modelos de computación celular son:

a) Los **autómatas celulares**, que fueron concebidos a finales de los años 40 por Stanislaw Ulam y Jonh von Neumann. Constan de un conjunto de unidades de proceso, llamadas células, colocadas sobre los nodos de una rejilla, cada una de las cuales puede presentar un estado de un conjunto finito de estados posibles. El estado que presentan dichas unidades se actualiza en pasos discretos según una cierta regla de interacción (función de transición) que depende del estado que presentan en ese momento las unidades de proceso vecinas. Desde el principio, los autómatas celulares se han utilizado como modelos formales en la física, la biología y la informática.

b) Las **redes neuronales celulares**, constituidas por un conjunto de unidades de proceso, llamadas neuronas, cuyos valores posibles pueden ser discretos o continuos y el valor que presenta cada unidad neuronal viene dado por una función que depende de una combinación lineal de los estados de las unidades vecinas. Se puede aplicar, por ejemplo, en análisis de imágenes para extraer el contorno de los objetos para su posterior reconocimiento e identificación.

c) La **computación ADN,** que utiliza unidades moleculares (ADN) y esta inspirada en la biología molecular. Aunque las unidades moleculares no son demasiado simples pues pueden exhibir un comportamiento biológico complejo, sin embargo, se pueden tratar como elementos simples, desde un punto de vista computacional, al poder realizar sólo unas cuantas operaciones básicas en el tubo de ensayo. Para encontrar el camino Hamiltoniano entre dos vértices dados (un camino que pasa sólo una vez por cada vértice), Adleman (1994) utilizó *oligonucleótidos*, es decir, cadenas cortas de hasta 20 nucleótidos, para codificar los vértices y la aristas de un grafo dado. A continuación, colocó múltiples copias de los oligonucleótidos en un tubo de ensayo; los oligonucleótidos se enlazaban unos con otros formando moléculas que representan caminos del grafo. Aplicó entonces procedimientos de la biología molecular para tamizar la plétora de soluciones moleculares ADN candidatas. Es decir, encontrar si existe o no un camino Hamiltoniano. El tamaño enormemente pequeño de estas moléculas permite un paralelismo inmenso sobre una escala completamente nueva. Además, serán más rápidas, más eficientes

energéticamente y capaces de almacenar mucha más información que los ordenadores actuales.

d) Las **estructuras autorreproductoras**, que fueron estudiadas por von Neumann a finales de los cuarenta. Chou y Reggia (1998) han usado bucles reproductores para resolver el problema *NP-completo* de la *satisfacibilidad* que consiste en ver si existe valores de verdad que hacen verdadero un predicado (literal). Cada bucle representa a una posible solución de factibilidad para el problema; cuando se reproduce un bucle se obtiene un bucle hijo que representa a una solución candidata diferente, que a su vez se vuelve a reproducir. Bajo un forma de selección artificial, las reproducciones que representa soluciones prometedoras proliferan y las que no se eliminan.

1.3.1. Características de un sistema básico de Computación Celular

Los modelos de computación celular son muy flexibles y se puede adaptar para la realización de tareas específicas, según las características de las unidades de proceso, que llamaremos unidades celulares o simplemente **células**.

1.3.1.1 Tipos de células

Las células toman una valor de un conjunto de valores posibles, que puede ser discreto (en cuyo caso el valor se llama *estado*) o continuo (analógico).

1.3.1.2 Comportamiento

El **comportamiento dinámico** de la célula viene determinado por una función de los valores de sus células vecinas. Dicha función puede venir dada por alguna de las siguientes formas:

- *Una enumeración exhaustiva*, utilizada para células discretas, donde el estado que debe presentar cada célula, según la configuración de estados de la células vecinas, viene recogido en una lista.
- *Una función parametrizada* (lineal o no lineal) que describe el estado siguiente de cada célula como una correspondencia con los estados de las células vecinas.
- *Un programa* que computa el estado siguiente de cada célula según los estados de las células vecinas.
- *Una regla de conducta* que especifica el comportamiento en diferentes situaciones. Dicha regla puede estar inspirada en la biología molecular o en la física cuántica.

1.3.1.3 Movilidad

Además de cambiar sus valores, las células se puede cambiar también su situación dentro de un ambiente dado y, por lo tanto, pueden ser, o no, **móviles**.

1.3.1.4 Conectividad

La comunicación entre células se realiza según un esquema de conectividad específico. Así, las células pueden estar colocadas sobre los vértices de una *rejilla regular* con una geometría dada, como puede ser rectangular, triangular, hexagonal, etc. En la rejilla rectangular cada célula sólo tiene 4 vecinas (figura 1.3.2). La *regularidad* quiere decir que todas las células tienen que tener el mismo esquema de conexión. Así, el esquema de conectividad puede ser regular o irregular.

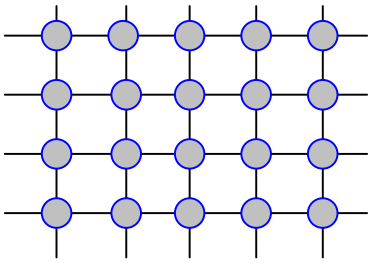

Figura 1.3.2. Células colocadas sobre una rejilla rectangular

Cuando la rejilla es finita tenemos que especificar las condiciones de las células que están en la frontera, por ejemplo, como neuronas que no cambian de estado con el tiempo. También podemos evitar tener células en la frontera utilizando superficies como el toro.
Asimismo, las células pueden constituir los vértices de un *grafo* dirigido. Aunque se puede considerar esquemas de conexión donde todas las células estén conectadas entre sí, pero este esquema cae fuera del ámbito de la computación celular.

1.3.1.5 Líneas de conexión

La conexión entre las unidades celulares es simple, es decir, la información transmitida por ellas es pequeña, generalmente sólo se transmiten los valores de estado de las células vecinas.

1.3.1.6 Topología celular

El propio esquema de conectividad induce una topología en las unidades celulares. Sin embargo, en algunas formas de computación celular las unidades celulares no están colocadas sobre rejillas o grafos sino que están en un ambiente que provoca contactos a causa del movimiento aleatorio o dirigido de las mismas, como ocurre en la computación ADN, donde no hay una topología rígida. Esto mismo ocurre también en la computación basada en hormigas.

1.3.1.7 Dinámicas temporales

El sistema se va actualizando (cambiando, o no, de estados) con el tiempo, y cada actualización se hace en cada instante o periodo de tiempo.

En una *dinámica temporal sincronizada* (paralela) se actualizan todas las células al mismo tiempo (simultáneamente) o un bloque específico de las mismas, mientras que en la *computación asíncrona* (secuencial) se actualiza sólo una célula cada vez, siguiendo una cierta secuencia, que puede ser aleatoria.

La actualización se puede hacer a intervalos regulares de tiempo, es decir, en términos de sucesos temporales discretos, y la llamaremos actualización *discreta*. Si no se hace ninguna división discreta del tiempo, la actualización es instantánea, es decir, *continua*.

1.3.1.8 Uniformidad

Se dice que el sistema es uniforme si todas las células son del mismo tipo, es decir, son idénticas y así ejecutan la misma función. Hay sistemas celulares no uniformes, donde células diferentes ejecutan funciones de transición diferentes. La no uniformidad puede ser una ventaja computacional.

1.3.1.9 Determinismo frente al no determinismo

El sistema es determinístico si para una entrada dada el sistema siempre toma la misma configuración de estados, y por tanto, termina con la misma salida. En un sistema no determinístico para una misma entrada podemos tener varias configuraciones de estados posibles que conducirán, posiblemente, a salidas diferentes. Según como sea la función de transición de la célula, podemos tener un sistema determinístico o no determinístico.

1.3.2 Comportamiento

Para ver el comportamiento de un sistema de computación celular como un todo vamos a analizarlo en diferentes facetas.

1.3.3.1 Programación

La computación celular requiere nueve técnicas de programación. Distinguiremos dos tipos:

a) *Programación directa*, donde el programador especifica completamente el sistema en su conjunto de salidas. Así, cuando el sistema recibe una entrada que es un ejemplo o instancia del problema en consideración, el sistema computa una salida correcta.
b) *Métodos adaptativos*, donde el programador no puede especificar completamente todo el conjunto de salidas, sino sólo parcialmente, y entonces recurre a procesos de aprendizaje, evolución o autoorganización para conseguir la funcionalidad deseada. Así, métodos de aprendizaje y algoritmos evolutivos se ha aplicado a las redes neuronales celulares para encontrar los parámetros de la función de transición con el fin de resolver algún problema dado. Los algoritmos evolutivos se han aplicado a la computación ADN para encontrar buenos códigos de secuencias de nucleótidos.

1.3.3.2 Entradas y salidas

Tenemos que especificar las entradas y salidas del sistemas según el problema que vayamos a resolver y utilizar una representación adecuada en el modelo celular. Además, tenemos que ver cómo se presentan las entradas a dicho modelo y como se leen las salidas.

1.3.3.3 Implementación

Aunque la mayoría de los experimentos en computación celular se llevan a cabo en ordenadores convencionales, el objetivo fundamental es la construcción de máquinas basadas en unidades celulares para conseguir realmente la potencia de la computación celular. En estas el coste se deberá fundamentalmente a las conexiones y no a las unidades de proceso. Hasta la fecha se ha desarrollado varias implementaciones, por ejemplo:

- Implementación de autómatas celulares como hardware digital de propósito general.
- Implementación de autómatas celulares usando procesadores configurables.
- Un chip analógico de redes neuronales celulares.
- Computación molecular en tubos de prueba
- Autómata celular de puntos cuánticos donde los estados no se codifican como voltajes, como con las arquitecturas digitales convencionales, sino por la posiciones de electrones individuales.

En computación celular no se diseña un modelo teórico y después se implementa sino que ambas cuestiones se deben considerar simultáneamente.

1.3.3.4 Escalabilidad

La computación celular permite una mayor escalabilidad que la computación clásica debido a su conectividad local, a la ausencia de un procesador central que tenga que comunicarse con cada célula y a la propia simplicidad de las unidades celulares. La adición de nuevas unidades celulares no es ningún problema.

1.3.3.5 Robustez

La robustez de un sistema es su capacidad para funcionar adecuadamente frente a fallos en alguna de sus partes. Cuando una célula funciona incorrectamente entonces los enlaces de comunicación fallan y nosotros deseamos que el sistema continúe funcionando correctamente o que la degradación sea aceptable. La conectividad local favorece la contención de los fallos más fácilmente al reducir los fallos a una región y previene así la expansión al sistema, de manera que una enorme cantidad de células permanecerán operativas y funcionando correctamente.

1.3.3.6 Jerarquización

La descomposición jerárquica está presente en las ciencias de la computación, a nivel de lenguajes de programación, de códigos, de lenguajes máquina y a nivel de transistores. Las jerarquías aparecen en la naturaleza, de la molécula se pasa a la célula y de la célula a un organismo, así como las jerarquías de procesamiento en el sistema visual, que comienza con el registro de la imagen en la retina (bajo nivel) y termina con el reconocimiento de objetos (alto nivel). Estas formas jerárquicas también aparecen en la computación celular, bien fijadas en el conjunto de salidas o emergen mediante la programación adaptativa.

1.3.3.7 Problemas locales frente a problemas globales

Un problema local contempla la computación de una propiedad en términos puramente locales, como ocurre con el funcionamiento de una unidad celular. Un problema global contempla la computación de una propiedad general del sistema. Así, un reto para el diseñador de modelos celulares es encontrar reglas de interacción local para resolver problemas globales.

1.3.3 ¿Para qué áreas de aplicación es adecuada la computación celular?

- La computación celular encuentra soluciones rápidas y eficientes para problemas *NP-completos* que se presentan en diferentes dominios, como en diseño de redes, teoría de grafos, lógica, optimización combinatoria, secuenciación, localización, búsqueda, determinación de rutas óptimas, etc.
- Los modelos celulares encuentran una aplicación natural al *procesamiento de imágenes digitales*, como consecuencia de su arquitectura, pues al ser las imágenes matrices bidimensionales podemos utilizar rejillas rectangulares y representar cada píxel de la imagen mediante una unidad celular, y diseñando la máquina para que haga tareas propias del procesado de imágenes.
- También se ha utilizado modelos celulares para *generar números aleatorios* de alta calidad.
- La capacidad de la computación celular para realizar operaciones aritméticas permite que se puedan construir calculadoras rápidas a una escala increíblemente pequeña (nanométrica).