UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES ACATLÁN

MODELO DE MEMORIA ASOCIATIVA CON UN ENFOQUE DE SISTEMAS DINÁMICOS EN UN PARADIGMA DE RECONOCIMIENTO DE PATRONES

TESIS

QUE PARA OBTENER EL TÍTULO DE

LICENCIADA EN MATEMÁTICAS APLICADAS Y COMPUTACIÓN

PRESENTA

AIDE GUADALUPE VILLAFRANCO RAMÍREZ

Asesor: **MAESTRA JEANETT LÓPEZ GARCÍA**

Santa Cruz Acatlán, Naucalpan, Estado de México Agosto de 2016

A mis padres,
quienes me obsequiaron el soplo de viento
que acompaño a esta travesía
hasta su arribo en el puerto de la culminación.

Agradezco a la vida,
que me permitió concluir satisfactoriamente esta obra,
y a mi familia toda,
por haber permanecido a mi lado en el instante justo.

Un agradecimiento muy especial a mi tutora
la Mtra. Jeanett López García,
cuyo apoyo fue el timón de esta embarcación.

Contenido

Introducción 1

Capítulo 1 Introducción a las Redes Neuronales Artificiales 5

1.1 Características de redes neuronales artificiales 5

1.2 El cerebro 7

1.3 La neurona biológica 9

1.4 La neurona artificial 10

1.4.1 Tipos de funciones de activación 13

1.5 Evolución histórica de las RNAs 18

Capítulo 2 Elementos que integran el diseño de RNAs 23

2.1 Arquitecturas de redes neuronales artificiales 23

2.2 Representación de la información 26

2.3 Proceso de aprendizaje 27

2.3.1 Reglas de aprendizaje 28

2.3.2 Paradigmas de aprendizaje 31

Capítulo 3 Memoria Asociativa con RNAs 33

3.1 Definición de memoria asociativa 33

3.1.1 Tipos de memoria asociativa 34

3.1.2 Estructura de una memoria asociativa 34

3.1.3 Aprendizaje asociativo 36

3.1.3.1 Aprendizaje Hebbiano 37

3.2 RNAs como sistemas dinámicos 42

3.2.1 Problemas de selección de conjuntos 45

3.3 Memoria asociativa con sistemas dinámicos 57

3.3.1 Hipergrafos y redes neuronales artificiales 57

3.4 El modelo de memoria 63

3.4.1 Dinámicas del modelo de memoria 67

3.4.2 Aplicaciones del modelo de memoria 69

3.4.3 Simulador del modelo de memoria 79

3.4.4 Análisis de resultados 83

Conclusiones 87

Bibliografía 89

Apéndices 93

Apéndice A 93

Apéndice B 99

Introducción

Pensar en cómo un órgano como el cerebro es capaz de responder de manera casi inmediata a un estímulo dado, aprender, generar conocimiento, tomar decisiones, crear y desenvolverse en un ambiente. Son el tipo de cuestionamientos que han motivado a muchas generaciones a interesarse en el tema, primeramente a los encargados del estudio de la vida como son biólogos, médicos, neurobiólogos, psicólogos, químicos, entre otros, siguiendo aquellos interesados en modelar dichos comportamientos como los matemáticos, físicos e ingenieros. El trabajo multidisciplinario de todas estas ramas del conocimiento ha aportado grandes avances que poco a poco dan respuesta a muchas de las interrogantes que han surgido a lo largo del tiempo. Son precisamente estas aportaciones el génesis de lo que se conoce como *Redes Neuronales Artificiales*, una disciplina que, inspirada en el cerebro, desarrolla modelos que buscan imitar las capacidades de las redes neuronales biológicas.

En este orden de ideas el presente trabajo tiene como objetivo realizar un estudio de la teoría de redes neuronales artificiales y aplicar sus principios en la modelación de una memoria asociativa para el reconocimiento de patrones, que pueda trasladarse a un sistema computarizado a fin de facilitar la generalización del modelo para diferentes conjuntos patrón. El modelo radica en representar a las memorias como vectores en un espacio n-dimensional con componentes ± 1 y el reconocimiento consiste en que el sistema converge desde cualquier vector de entrada incompleto o con ruido, a alguno de los patrones almacenados. Este comportamiento se modela a través de un sistema de ecuaciones diferenciales que guían al vector inicial hasta un patrón conocido y permiten conocer sus condiciones en el instante t.

Para alcanzar el objetivo anterior la investigación ha sido dividida en tres capítulos. El capítulo uno comprende el marco teórico-histórico de la investigación, inicia con una introducción a las redes neuronales como resultado de la búsqueda por modelar el cerebro y la tendencia a construir tecnología basada en las capacidades humanas capaz de adaptarse al medio ambiente sobre el que trabaja.

Posteriormente se expone un estudio breve del cerebro, órgano responsable de las actividades motoras, cognitivas y de carácter emocional inherentes al ser humano. Es Ramón y Cajal quien en 1911 introduce a la neurona como unidad fundamental en la estructura cerebral y gracias a este principio, investigaciones subsecuentes han determinado que las conexiones neuronales no son producto del azar sino que siguen un comportamiento que bien puede ser modelado [1]. Aún cuando al día de hoy no se cuenta con un diagrama de conexiones neuronales completamente fiable. Asimismo se aborda el tema de la neurona biológica, los elementos que la constituyen, su clasificación y la función orgánica para la que ha sido creada. Conocer cómo una neurona real ejecuta sus actividades es la base del modelado de redes neuronales. En este mismo apartado se expone el proceso que se sigue dentro del cerebro para generar algún tipo de actividad que llegue hasta las últimas ramificaciones de la estructura nerviosa.

A partir del estudio de la neurona biológica se toma la neurona artificial no lineal propuesta por Haykin [2] para diferentes funciones de activación, conjuntos de sinapsis, valores umbral y un operador para la agregación de señales de entrada. Es este modelo de neurona el que se utiliza a lo largo de la investigación.

Una vez conformado un panorama general del concepto de redes neuronales artificiales se da paso al marco histórico que las acompaña con un estudio breve sobre la evolución que han sufrido a lo largo del tiempo.

Una vez completado el marco teórico-histórico que gira alrededor de las redes neuronales artificiales es preciso continuar con el esquema general que conforma el diseño de las mismas. Para ello el capítulo dos abarca los elementos que integran este tipo de modelos. Inicia con los tipos de arquitectura que toman las neuronas, para continuar con las reglas que hay que seguir para representar los datos con los que trabaja el modelo. La correcta representación de la información es de vital importancia porque es la responsable del proceso de aprendizaje en la red. Este proceso de aprendizaje inicia con una estimulación proveniente del ambiente provocando cambios en los parámetros libres del sistema que, a la larga, pueden ocasionar alteraciones en la estructura interna modificando la respuesta del sistema. Para regular el proceso mediante el cual se modifican estos parámetros, algunos especialistas han desarrollado una serie de algoritmos conocidos como reglas de aprendizaje. Estas pueden trabajar sobre un entorno supervisado por un maestro quien representa el conocimiento a través de un conjunto de ejemplos entrada/salida, donde los parámetros se ajustan de manera iterativa hasta que la red imita al maestro, o bien, sobre un entorno no supervisado con el uso de programación neurodinámica o a través de un índice que mide la calidad de las tareas que ejecuta la red.

Posteriormente se da paso a un modelo de memoria asociativa basado en neuronas artificiales. El capítulo tres comienza con una definición general de memoria asociativa, donde un estímulo se enlaza a una respuesta de tal suerte que la memoria genera un tipo de asociación entre el estimulo y la respuesta. El

tipo de aprendizaje más cercano a este comportamiento es el aprendizaje asociativo, donde el cambio sináptico está en función de la actividad pre-sináptica y pos-sináptica. El objetivo final del modelo es relacionar vectores de entrada con vectores de salida aprendidos previamente, aún cuando las entradas contengan alguna clase de ruido.

Dentro del mismo capítulo se abordan de forma directa, los modelos neuronales como sistemas de ecuaciones n-dimensionales, donde cada neurona se define por un valor de estado (encendido y apagado), una función de activación y un valor umbral, que acorde a la dinámica del sistema modifican el estado de la unidad. El objetivo de la red es generar trayectorias que se aproximen asintóticamente a un punto fijo, donde se ha de analizar el comportamiento del sistema dentro y fuera de una vecindad para determinar la estabilidad de las trayectorias.

Posteriormente la investigación conduce al planteamiento de problemas de selección de conjuntos con un enfoque práctico de redes neuronales, cuyas trayectorias constantes representan soluciones al problema. Asimismo se aborda la cobertura de combinatorias como una variante que sigue los mismos principios. En ambos casos, se inicia con la definición del problema para después proponer el modelo neuronal que se ajusta. Posteriormente se expone el desglose de la dinámica del modelo para obtener las trayectorias constantes que han de ser sometidas a una evaluación de estabilidad y así determinar cuáles corresponden a una solución real del problema.

La investigación sigue con un modelo de memoria que plantea como patrones a vectores en el espacio n-dimensional con componentes ± 1, y entiende el reconocimiento como la convergencia desde cualquier vector de entrada a un patrón almacenado. De igual forma se presenta un modelo con ciclos límite donde cada trayectoria atractor recorre algunos de los 2^n vértices del n-cubo con coordenadas ± 1 en el espacio n-dimensional. Con trayectorias cíclicas disjuntas por los diferentes ortantes del hiperespacio.

Una vez definida la dinámica básica de una memoria y su función de activación, se propone un modelo general para la composición de un sistema de ecuaciones no lineales. Dicho modelo se somete a un análisis para asegurar la existencia de trayectorias constantes estables como soluciones del sistema. Con el modelo de memoria definido se procede a analizar su comportamiento, de tal suerte que se tiene un primer ejemplo funcional del modelo de memoria. Así mismo, se realiza un estudio del comportamiento del modelo de memoria dentro de la zona de transición de acuerdo a los atractores estables en regiones adyacentes del hiperespacio.

Finalmente se presenta una serie de problemas que dan noción de algunas de las aplicaciones del modelo de memoria. Por cada problema se define la función de activación y el sistema de ecuaciones obtenido a partir del modelo general. Así mismo se utiliza el método de Euler y una hoja de cálculo para conseguir las trayectorias constantes del sistema partiendo de estados de inicio aleatorios. En

cada caso ha de realizarse un estudio de los resultados arrojados por el método numérico. La investigación cierra con la presentación de un simulador de memoria sobre el que se prueba el reconocimiento de los primeros diez dígitos numéricos, siendo este último apartado el que busca demostrar la viabilidad de tomar como fuente de inspiración el cerebro, descifrar su comportamiento y abstraerlo a modelos matemáticos que permitan construir soluciones basadas en los principios por los que se rige.

Capítulo 1
Introducción a las Redes Neuronales Artificiales

1.1 Características de redes neuronales artificiales

Las redes neuronales artificiales son un conjunto de unidades (neuronas) interconectadas inspiradas en las redes neuronales biológicas. Las capacidades de una red neuronal dependen del número de neuronas con el que cuenta así como la arquitectura de diseño.

Los modelos de redes neuronales artificiales surgen a partir de los estudios realizados por diversos investigadores persiguiendo modelar sistemas nerviosos biológicos para dar una explicación de cómo el cerebro a partir de la interacción entre sus neuronas da lugar al aprendizaje y la memoria.

Por otro lado, las redes neuronales artificiales exhiben varias de las funciones del cerebro siendo capaces de aprender, memorizar un conjunto de patrones, clasificarlos, generalizar a qué clase pertenece un nuevo objeto a partir de la experiencia acumulada durante el reconocimiento de otros previos, establecer asociaciones entre objetos siendo por tanto capaces de reconocer símbolos tales como letras, números o cualquier objeto o patrón [3].

En el caso particular del reconocimiento de patrones, donde a partir de una versión incompleta o corrompida de un patrón se obtiene la versión correcta. Las características ausentes existen en asociación con aquellas presentes en el patrón de prueba a través de las fuerzas de conexión. Por lo tanto, la recuperación no se hace a partir de una clave o índice como ocurre en una memoria de computadora.

A grandes rasgos, una red neuronal es un algoritmo inspirado en el comportamiento observado en las neuronas durante la ejecución de una tarea o función de interés particular. Las redes neuronales artificiales usualmente son implementadas en dispositivos electrónicos o simuladas en software de computadora. Véase a continuación la Fig. 1.1 que ilustra la estructura organizacional de un sistema basado en redes neuronales artificiales.

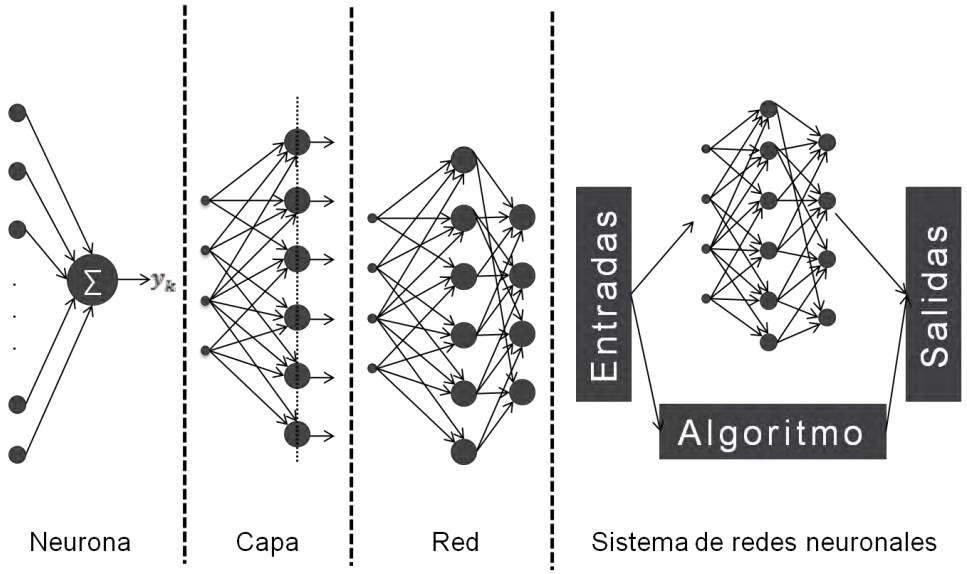


Fig. 1.1 Estructura organizacional de un sistema de redes neuronales [4]

A continuación se muestra una definición formal de red neuronal:

> "Una red neuronal es un procesador distribuido en paralelo compuesto de unidades de procesamiento que tiene una propensión natural para el almacenamiento de conocimiento experimental y que es puesto a disposición para su uso. Se parece al cerebro en dos aspectos:
>
> - El conocimiento de la red es adquirido de su entorno a través de un proceso de aprendizaje.
> - Fuertes conexiones interneuronales, conocidas como pesos sinápticos, que se utilizan para almacenar el conocimiento adquirido"[2].

El procedimiento usado para la ejecución del proceso de aprendizaje es llamado algoritmo de aprendizaje, el cual determina la función que modifica sistemáticamente los pesos sinápticos de la red para conseguir el objetivo deseado. Es importante señalar que una red neuronal artificial tiene la capacidad de modificar sus pesos sinápticos en analogía con las neuronas del cerebro humano donde éstas pueden fortalecer o debilitar sus conexiones sinápticas.

A continuación se exponen de Haykin [2] e Hilera González [5] algunas de las características que ofrecen las redes neuronales artificiales:

I. *Auto-organización*. La organización interna del sistema aumenta de complejidad sin la guía de un agente externo. Esta característica permite la modificación de la red, lo que se traduce en un aprendizaje específico y permite que responda a situaciones que no han sido experimentadas con anterioridad.

II. *Adaptabilidad*. La modificación de los pesos sinápticos en las redes neuronales les proporciona la capacidad de adaptación a los cambios sufridos alrededor de su medio. Aún cuando una red neuronal se entrene para operar en un ambiente específico, ésta puede fácilmente volverse a entrenar para enfrentarse a cambios mínimos en las condiciones de operación del sistema. Más aún, las redes neuronales pueden ser diseñadas para cambiar sus pesos sinápticos en tiempo real, con el objetivo de hacer frente a situaciones que operan en un ambiente no estacionario.

III. *Información contextual*. Debido a la interconexión que existe en la arquitectura de la red neuronal, cada una de las neuronas que la conforman puede ser afectada por la actividad global de otras neuronas en la red. Consecuentemente, la información contextual puede ser distribuida a lo largo de la red.

IV. *Tolerancia a fallos*. Una red neuronal tienen el potencial de ser tolerante a fallos, en el sentido de que puede continuar funcionando aún cuando las condiciones de operación sean adversas, por ejemplo, si una neurona o sus conexiones son destruidas, la red tiene la capacidad de continuar operando el sistema. Por otro lado, las redes neuronales son capaces de reconocer información con ruido, distorsionada o incompleta.

1.2 El cerebro

El cerebro es aún hoy en día una fuente de conocimiento inagotable que causa fascinación en el mundo científico. Desde las primeras apariciones de computadoras el hombre ha buscado imitar las capacidades del cerebro y hoy continúa siendo un estímulo para la construcción de máquinas inteligentes.

El cerebro es el órgano humano responsable de una inmensa gama de actividades que van desde las motoras básicas (como caminar, dormir o respirar) hasta las complejas o de carácter cognitivo (como pensar, hablar o aprender). Así mismo es el centro del sistema nervioso con un peso de 1,400 gramos aproximadamente en un humano adulto [2] y consta una red de neuronas que constantemente recibe información, la procesa y emite una respuesta. Puede ser visto como un sistema de tres estados tal y como se muestra en la Fig. 1.2.

Fig. 1.2 Diagrama de bloques que representa el sistema nervioso [2]

Los receptores convierten estímulos provenientes del cuerpo o del medio externo, en impulsos que transmiten la información a la red neuronal. Los emisores convierten los impulsos eléctricos generados por la red en las respuestas de salida del sistema.

El pionero en el estudio del cerebro fue el médico Ramón y Cajal [3], quien en 1911 definió a la neurona como la unidad fundamental para el procesamiento de información en el cerebro. Gracias a su arduo trabajo y al método de tinción utilizado en el microscopio, demostró que el cerebro no era una masa continúa de células cerebrales, sino una red de neuronas en la que cada neurona conservaba su individualidad. Cabe destacar que los eventos en una neurona suceden en un rango de milisegundos (10^{-3}), mientras que los eventos en un chip suceden en nanosegundos (10^{-9}) [2], sin embargo, el cerebro reduce su índice de operación gracias a su asombroso número de neuronas y conexiones neuronales.

El cerebro funciona a través de señales o impulsos eléctricos que responden a necesidades de carácter biológico o estímulos provenientes del ambiente, así como a las situaciones a las que un individuo es expuesto. A manera de ilustración sobre los procesos que toman lugar en el cerebro para la generación de una señal de respuesta, véase la Fig. 1.3 que muestra la propagación de los impulsos nerviosos en la red de neuronas cerebrales.

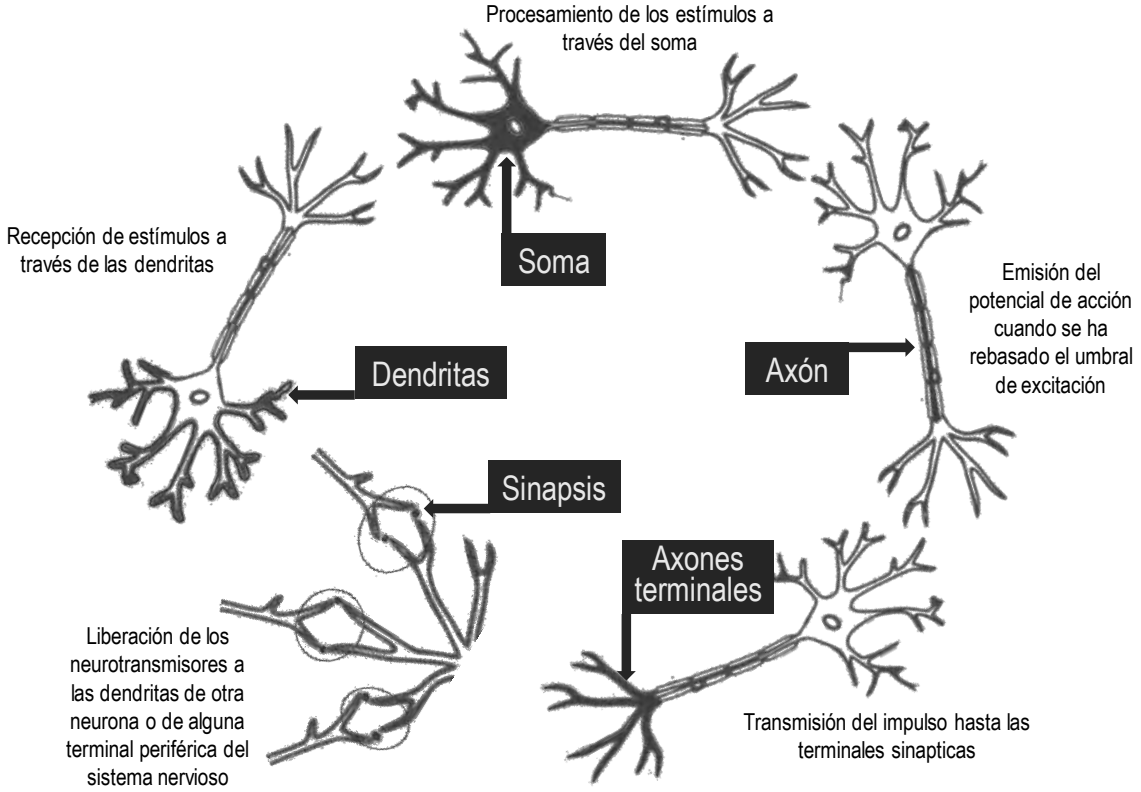


Fig. 1.3 Propagación de los impulsos nerviosos

Se estima que existen alrededor de 10 billones de neuronas en la corteza cerebral humana y aproximadamente 60 trillones de sinapsis o interconexiones, convirtiéndolo en una enorme y eficiente estructura.

1.3 La neurona biológica

Las redes neuronales biológicas son estructuras compuestas de millones de neuronas en las que cada neurona funciona dependiendo de las interconexiones que tenga con las demás. Una neurona biológica es una célula animal cuya función puede reducirse a un comportamiento que permite o no la transferencia de impulsos eléctricos dependiendo de si la cantidad almacenada de energía rebasa o no el umbral de activación.

La definición dada por Ramón y Cajal a la neurona biológica es:

> *"*Una neurona es una célula individual que se relaciona, y por tanto se conecta con otras neuronas, a través de una conexión muy particular llamada sinapsis. El conjunto de sus actividades están orientadas al procesamiento de señales de entrada así como al almacenamiento y modificación de las sinapsis" [3].

Y de acuerdo con Córdova Martínez [6], desde el punto de vista morfológico una neurona está compuesta por:

I. *Cuerpo celular o soma*. Centro metabólico de la neurona en donde se procesa la información.
II. *Dendritas*. Prolongación del cuerpo celular de neurona que tiene como función ser el centro receptor de la información.
III. *Axón*. Prolongación del cuerpo celular de la neurona que constituye una fibra nerviosa encargada de transportar las señales generadas por la neurona.
IV. *Terminales axónicos o sinápticos*. Elementos de transmisión que permiten que una neurona contacte y transmita información a la zona receptora de otra neurona. La zona de contacto se denomina *sinapsis*, así que la *pos-sinapsis* generalmente se ubica en la dendrita, aún cuando llegan a presentarse contactos en el cuerpo celular y el axón.

A manera de ilustración véase la Fig. 1.4 que muestra la morfología de una neurona biológica.

Se puede decir que las neuronas están funcionalmente polarizadas [7], es decir, reciben señales a través de las dendritas, las procesan y acumulan en el soma (integración), y envían la respuesta a otras neuronas a través del axón. En caso de existir suficientes impulsos eléctricos a un número adecuado de conexiones pre-sinápticas, se libera una cantidad suficiente de neurotransmisor que estimula a la neurona pos-sináptica hasta llegar a un umbral de excitación, ocurriendo así un cambio en la corriente eléctrica denominado *potencial de acción*.

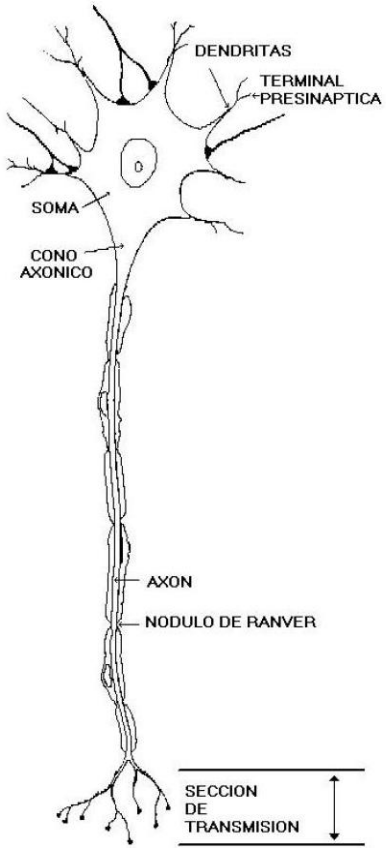


Fig. 1.4 Diagrama de una neurona biológica[8]

El potencial de acción, es un evento electrofisiológico breve que ocurre porque la membrana de la neurona tiene propiedades activas (es un medio excitable), usualmente se origina en el segmento inicial del axón y se propaga a lo largo de este. Es un mecanismo básico para la comunicación entre las neuronas y puede pensarse en él como una señal que es enviada por una neurona a otras. Cabe destacar que una neurona puede recibir muchas señales de otras neuronas (convergencia) y regresar señales a muchas otras (divergencia) [7].

1.4 La neurona artificial

La neurona representa el elemento principal para la operación de las redes neuronales artificiales, y se puede definir como una unidad de procesamiento de datos. Es de suma importancia señalar que las neuronas empleadas en el diseño de redes neuronales artificiales son sumamente primitivas en comparación con las neuronas biológicas, por lo que las redes generadas a partir de las mismas no tienen sentido de comparación con los circuitos locales e interregionales del cerebro.

La Fig. 1.5 muestra el modelo *no lineal* de una neurona que es la base para el diseño de las redes neuronales artificiales. Cabe aclarar que la no linealidad se debe a que la salida de la neurona no es linealmente dependiente de sus entradas

(es decir, proporcional a la suma de sus señales de entrada), ya que este modelo utiliza funciones de activación no lineales (función escalón, rampa o sigmoide) que producen respuestas acotadas.

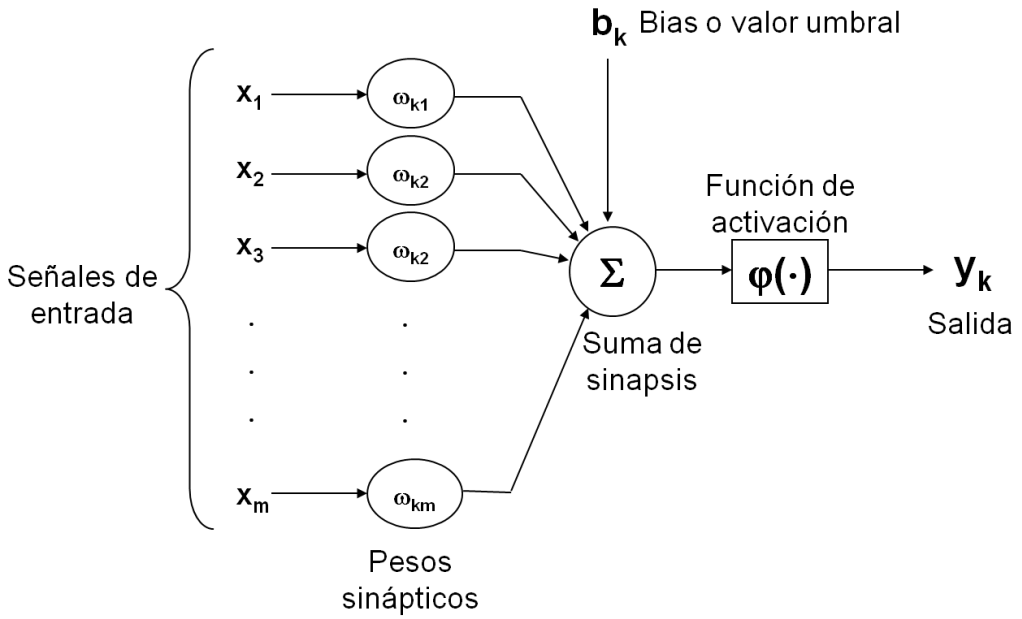


Fig. 1.5 Modelo no lineal de una neurona [2]

Los elementos básicos que conforman un modelo de neurona artificial son:

I. *Conjunto de sinapsis o vínculos conectores*. Caracterizados por una fuerza o peso propio para cada una de las neuronas representadas en el modelo.

x_j: Señal de entrada de la sinapsis j conectada a la neurona k.

ω_{kj}: Peso de la sinapsis j conectada a la neurona k, donde $\omega_{kj} \in \mathcal{R}$.

En términos matemáticos una sinapsis es representada por la multiplicación de la entrada x_j por el peso ω_{kj}.

$$Sinapsis = x_i \omega_{ki} \qquad 1.1$$

La sinapsis es el vínculo que une una neurona con otra formando un canal de comunicación entre ambas. En la descripción tradicional de la organización neural, se asume que la sinapsis es una simple conexión que puede generar excitación o inhibición en la neurona receptora, pero no ambas.

II. *Operador de agregación de entradas*.

$\sum$ *:* Combinación lineal de las sinapsis.

III. *Función de activación*.

$\varphi(\cdot)$ *:* Limita la amplitud de la salida de una neurona a valores finitos; en general, el rango de amplitud normalizado se encuentra dentro de los intervalos $[0,1]$ o $[-1,1]$.

IV. *Bias o valor umbral*.

b_k*:* Disminuye o incrementa el peso neto del valor de entrada en la función de activación dependiendo del signo.

Partiendo de la neurona *no lineal* mostrada en la Fig. 1.5, el modelo matemático correspondiente es:

$$Neurona = \begin{cases} u_k = \sum_{j=1}^{m} x_j \omega_{kj} \\ y_k = \varphi(u_k + b_k) \end{cases} \quad 1.2$$

Siguiendo el modelo anterior y tomando a

$$v_k = u_k + b_k \quad 1.3$$

como el *potencial de activación de la neurona*, entonces se puede observar que si

$$u_k = 0 \Rightarrow v_k = b_k \quad 1.4$$

Sustituyendo *1.2* en *1.3* se obtiene *1.5*.

$$v_k = \sum_{j=1}^{m} x_j \omega_{kj} + b_k \quad 1.5$$

Haciendo $x_0 = +1$ y $\omega_{k0} = b_k$, se tiene

$$v_k = \sum_{j=0}^{m} x_j \omega_{kj} \quad 1.6$$

$$y_k = \varphi(v_k) \quad 1.7$$

Aunado a lo anterior, se obtiene el modelo equivalente de neurona que se muestra en la Fig. 1.6. Donde el umbral de activación se considera una entrada fija del sistema.

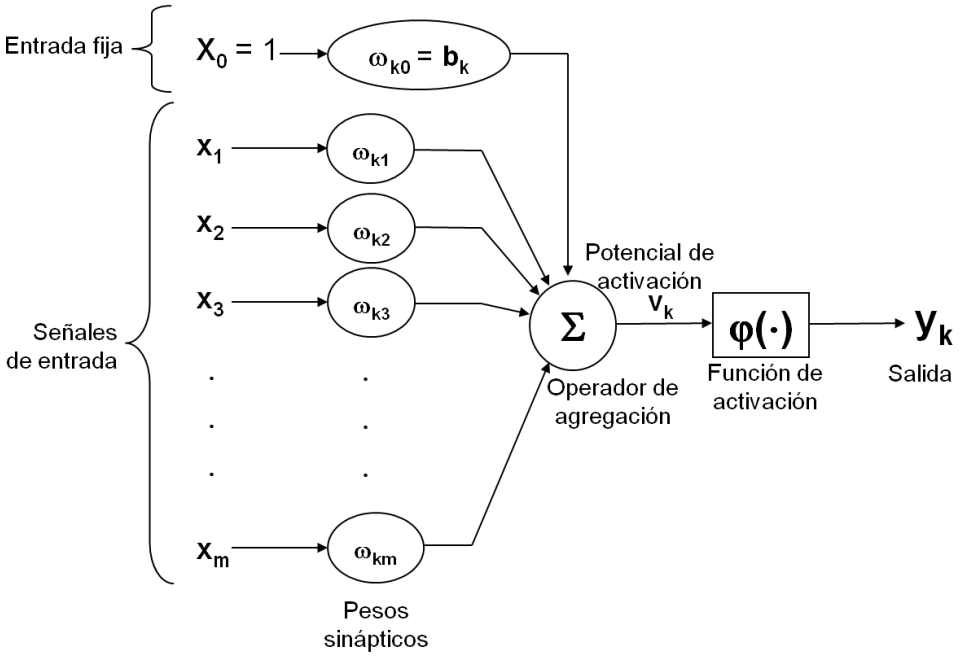


Fig. 1.6 Modelo equivalente de neurona [2]

1.4.1 Tipos de funciones de activación

En analogía con la neurona biológica la función de activación $\varphi(v)$ representa la integración de los impulsos en el soma y su salida y_k la señal en el axón. La función de activación puede ser lineal o no lineal y es seleccionada para satisfacer alguna especificación del problema que la neurona está intentando resolver [9]. Dentro de las funciones de activación mayormente utilizadas se encuentran:

I. *Función escalón*

$$\varphi(v) = \begin{cases} 1, & si\ v \geq 0 \\ 0, & si\ v < 0 \end{cases} \quad 1.8$$

Por lo tanto, la salida de la neurona *k* es:

$$y_k = \begin{cases} 1, & si\ v_k \geq 0 \\ 0, & si\ v_k < 0 \end{cases} \quad 1.9$$

Donde v_k representa el campo local de excitación de la neurona.

$$v_k = \sum_{j=1}^{m} x_j \omega_{kj} + b_k \quad 1.10$$

La gráfica de la función se muestra en la Fig. 1.7. Este modelo también es conocido como el modelo de McCulloch y Pitts con la única diferencia de que la señal de salida de la neurona no corresponde a unos y ceros sino a valores positivos y negativos respectivamente.

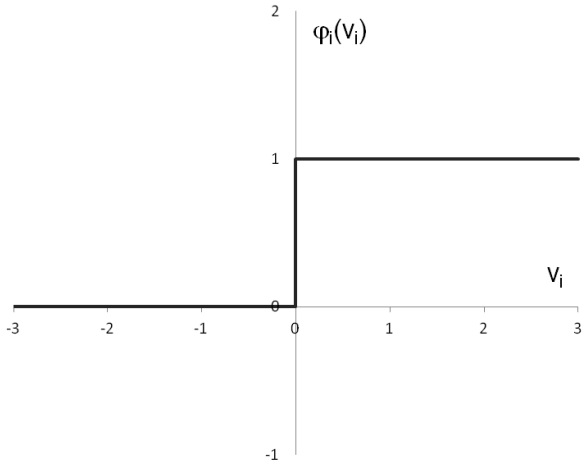


Fig. 1.7 Función escalón

II. *Función lineal con umbral*

$$\varphi(v) = \begin{cases} 1 & v \geq \frac{1}{2} \\ v & -\frac{1}{2} < v < \frac{1}{2} \\ 0 & v \leq -\frac{1}{2} \end{cases} \qquad 1.11$$

Donde el factor de amplificación dentro de la región lineal de operación es limitado al cuadrado unitario. Véase la Fig. 1.8.

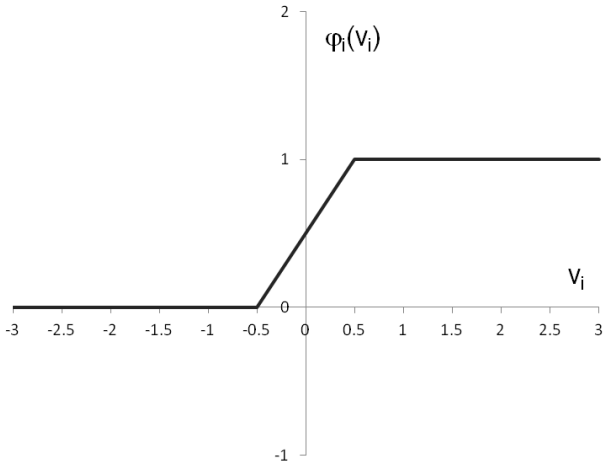


Fig. 1.8 Función lineal con umbral

III. *Función sigmoide*
Es la función de activación más usada en la construcción de redes neuronales. Proporciona un excelente balance entre la linealidad y la no linealidad al ser continuamente diferenciable. Un ejemplo de función sigmoide es la función logística.

$$\varphi(v) = \frac{1}{1 + e^{-av}} \tag{1.12}$$

Donde a es el parámetro de inclinación. Véase la Fig. 1.9

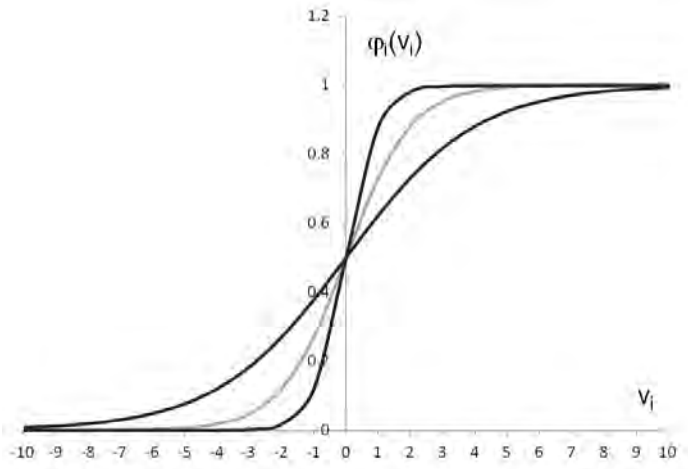


Fig. 1.9 Función sigmoide

Cabe señalar que si $a \to \infty$ entonces $\varphi(v)$ tiende a la función escalón. Y mientras la función escalón asume valores de 0 o 1, la función sigmoide toma valores continuos entre 0 y 1.

Es importante hacer hincapié en que las funciones de activación antes mencionadas devuelven valores binarios (0 o 1). Sin embargo, en ocasiones es conveniente que la salida se encuentre dentro del intervalo [-1,1], ya que vectores con componentes ± 1 tienen mayor probabilidad de ser ortogonales. Además, este tipo de cambio no afecta las propiedades esenciales de la red pero sí la simetría de las regiones solución [10]. Es por ello que a continuación se definen las funciones equivalentes.

I. *Función escalón*

$$\varphi(v) = \begin{cases} 1 & v > 0 \\ 0 & v = 0 \\ -1 & v < 0 \end{cases} \tag{1.13}$$

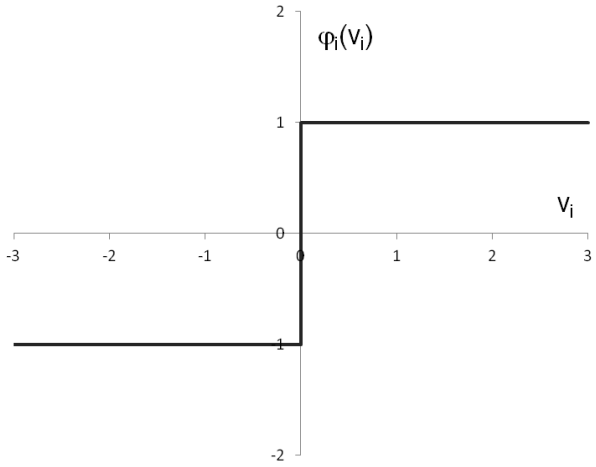


Fig. 1.10 Función escalón con rango de -1 a $+1$

II. *Función lineal con umbral*

$$\varphi(v) = \begin{cases} 1 & v \geq \frac{1}{2} \\ v & -\frac{1}{2} < v < \frac{1}{2} \\ -1 & v \leq -\frac{1}{2} \end{cases} \qquad 1.14$$

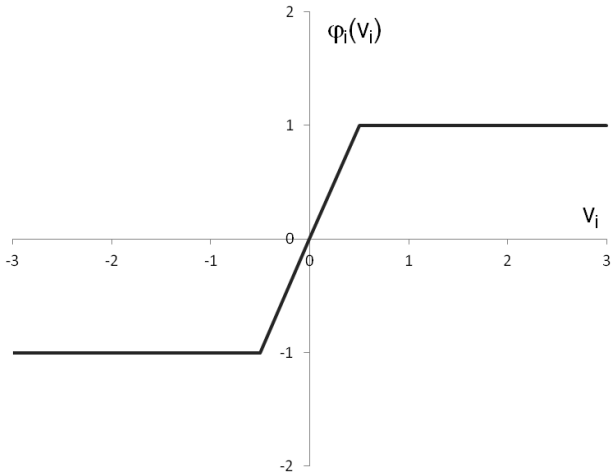


Fig. 1.11 Función lineal con umbral de rango -1 a $+1$

III. *Función sigmoide*, que es sustituida por la *función tangente hiperbólica* definida como:

$$\varphi(v) = \tanh(av) \qquad 1.15$$

Donde al igual que en la función sigmoide, a corresponde al parámetro de inclinación.

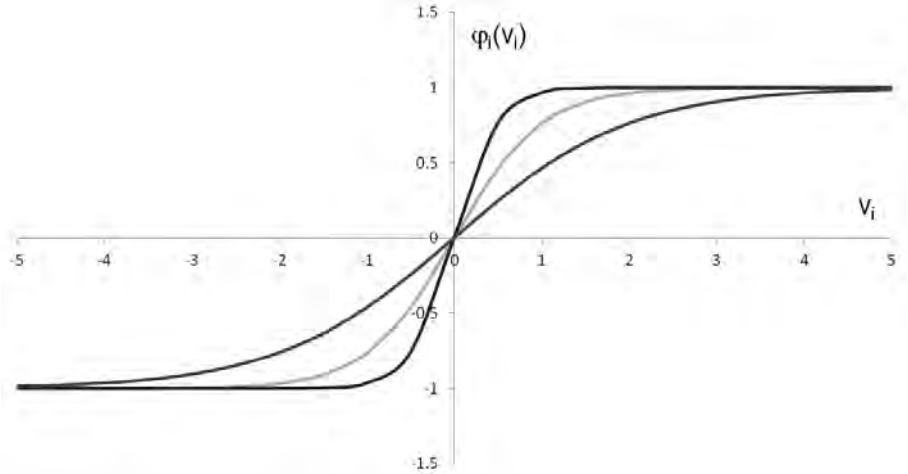


Fig. 1.12 Función tangente hiperbólica

Es importante desatacar que la función de activación es seleccionada durante la fase de diseño de tal forma que los parámetros, al ser ajustados con alguna regla de aprendizaje, permitan que la relación entrada-salida de la neurona cumpla algún objetivo específico. Por ejemplo, la función escalón puede ser usada para crear neuronas que clasifiquen entradas en dos categorías distintas[9]. Por otro lado, la función lineal con umbral aporta un grado de proporcionalidad entre las señales de entrada y la salida de la neurona dentro del intervalo acotado por $\pm\varepsilon$. Mientras la función sigmoide tiene la ventaja de ser continua. Aunque todas ellas pueden utilizarse para resolver un mismo problema con algoritmos de aprendizaje y arquitecturas diferentes, o combinarse en redes de capas múltiples, la selección depende de las especificaciones del problema que la neurona está intentando resolver.

Un comportamiento fundamental en las redes neuronales artificiales es el que se conoce como *feedback* o *retroalimentación*, que ocurre cuando en el sistema dinámico la salida de un elemento está en relación con la entrada de otro, ambos presentes en el mismo sistema.

Un ejemplo gráfico de un feedback básico es el que muestra la Fig. 1.13. Donde:

$x_j(n)$: Señal de entrada
$x'_j(n)$: Señal interna
$y_k(n)$: Señal de salida
n: Variable discreta de tiempo
A: Operador forward (hacia adelante)
B: Operador feedback (hacia atrás)

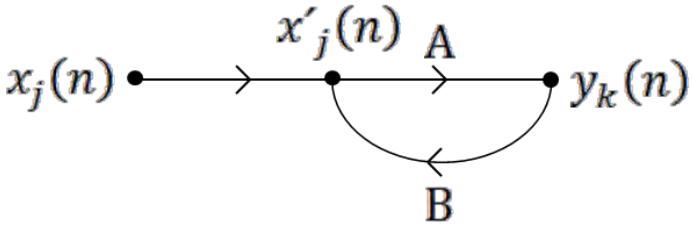


Fig. 1.13 Feedback [2]

Nótese que las entradas y salidas tienen la relación siguiente:

$$y_k(n) = \mathrm{A}[x'_j(n)] \quad 1.16$$

$$x'_j(n) = x_j(n) + \mathrm{B}[y_k(n)] \quad 1.17$$

donde A y B actúan como operadores.

A diferencia de las redes feedforward, donde la señal de entrada es conducida de forma secuencial y procesada por las neuronas de la capa siguiente hasta alcanzar las neuronas de salida. En una red con retroalimentación hay ciclos que les permiten a las neuronas (de capas previas o de la misma capa) procesar reiteradamente la salida generada por la neurona actual antes de producir una respuesta definitiva.

1.5 Evolución histórica de las RNAs

Como se menciona al inicio de la investigación, las redes neuronales artificiales se deben al surgimiento de metodologías caracterizadas por su inspiración en los sistemas biológicos. Cabe destacar que hoy en día existe un gran número de profesionistas, investigadores y científicos, que están empleando las redes neuronales artificiales como herramienta de aplicación en la solución de tareas que requieren la capacidad innata y de bajo nivel del cerebro humano, tales como: sistemas de decisión, procesamiento de señales, visión, control en tiempo real, clasificación de patrones, etc.[11]

El origen, y por ende la existencia de aplicaciones de redes neuronales a problemas reales, son producto del recorrido hecho por los sistemas de procesamiento de información a lo largo la historia. Aún cuando hoy en día la mayoría de las aplicaciones de redes neuronales están siendo ejecutadas sobre computadoras de propósito general, existen líneas de desarrollo enfocadas en hardware especializado en procesamiento paralelo [12].

Desde la aparición de metodologías inspiradas en sistemas biológicos, las técnicas de procesamiento de información se dividen en dos corrientes: por un lado la denominada *computación hard* con un enfoque algorítmico tradicional, y por el otro, la *computación soft* donde se posicionan las teorías de lógica borrosa, redes neuronales o sistemas conexionistas, razonamiento aproximado, algoritmos

genéticos y teoría del caos [13]. Para tener un panorama más amplio de estas dos vertientes se expone en la Tabla 1.1 un comparativo entre ambas.

	Computación hard	Computación soft
Principio guía	Precisión, certeza y rigor	Explotar la tolerancia a: imprecisión, incertidumbre, verdades parciales y aproximación
Entrada	Datos completos y exactos	Datos aproximados e incompletos
Salida	Solución excesivamente exacta	Solución que es “suficientemente buena”

Tabla 1.1 Comparativo entre computación hard y computación soft. Con información de Khoury [14] y Eick [15]

Es importante mencionar que las redes neuronales artificiales nacen como una línea de investigación de la inteligencia artificial cuyo origen se remonta al movimiento científico de la cibernética en los años cuarenta. Se basa en el procesamiento simbólico y difunde la expansión del uso de computadoras en aplicaciones tales como el procesamiento de palabras, proposiciones u otras entidades.

Entre los investigadores más sobresalientes de la primera etapa en el paradigma de la inteligencia artificial a mediados de los años cuarenta y cincuenta destacan John McCarthy, Allen Newell, Hebert Simon y Marvin Minsky. En forma paralela al desarrollo de la inteligencia artificial simbólica, a mediados de los años cincuentas y mediados de los sesentas surgen grupos de investigadores dedicados a la construcción de redes neuronales o sistemas conexionistas, entre los que se encuentran, F. Rosenblatt de la Universidad de Cornell en Nueva York, C. Rosen del Instituto de Investigación en Standford California y B. Widrow del Departamento de Ingeniería Electrónica en la Universidad de Standford California, entre otros [13]. Sin embargo, los pioneros en el estudio de redes neuronales artificiales son Warren McCulloch y Waltter Pitts quienes en 1943 proponen la primera red neuronal denominada *“Red neuronal de McCulloch-Pitts”*, misma que quizá, por ser la primigenia, carecía de la posibilidad de aprendizaje, no obstante, representó el nacimiento de las redes neuronales artificiales [3].

Una importante aportación al estudio de redes neuronales artificiales es el libro *“Design for a brain”* publicado por Ross Ashby en 1952, en el cual, se recopila información respecto a la relación entre estabilidad y error en el cerebro a través del uso de un aparato que Ashby denomino homeostato, dicha publicación resultó de gran ayuda a los investigadores conexionistas [3].

Entre los trabajos posteriores sobresale el de Rosenblatt y Widrow, quienes diseñan una red neuronal con únicamente una capa de conexiones modificables. En 1958 Rosenblatt diseña el “Perceptrón”, una red neuronal de dos estratos de

conexiones donde sólo uno de los estratos es modificable. Esta red se fundamenta en el teorema de convergencia que establece, que si los parámetros del sistema tienen la habilidad de realizar una determinada clasificación, el sistema es capaz de aprender en un número finito de pasos, siempre y cuando, las conexiones se modifiquen de acuerdo con la regla de aprendizaje [16]. La regla de aprendizaje a la que se hace referencia no es más que el algoritmo de aprendizaje por corrección del error, en el cual, de manera iterativa, los pesos sinápticos son ajustados de tal suerte que la diferencia entre la respuesta generada por el sistema y la respuesta deseada se minimiza.

Por otro lado, Widrow y Hoff diseñan en 1960 una estructura similar a la del Perceptrón con un nuevo tipo de unidad de procesamiento y un mecanismo de aprendizaje que permite la entrada de información de tipo continuo. Esta nueva aportación al campo del conexionismo fue la denominada neurona ADALINE (ADAptative LINear Elements), que introduce el uso de la regla delta o regla Wodrow-Hoff como regla de aprendizaje. La regla delta mide la diferencia entre la salida actual de la red y el vector objetivo en un contexto supervisado. Esta diferencia es tratada como un error que debe ser minimizado por el ajuste de pesos. El objetivo es encontrar el mínimo de la suma de errores sobre el conjunto de entrenamiento [9]. El modelo ADALINE fue utilizado en la eliminación de ecos en las líneas telefónicas, lo que lo convirtió en la primera red neuronal aplicada a un problema real y que continuó siendo empleada durante muchas décadas [4].

De acuerdo con Muñoz Pérez [17], la decadencia del conexionismo se hace presente cuando los fieles seguidores y líderes de la inteligencia artificial, Marvin Misnky y Seymour Papert, realizan la publicación de su libro “Perceptrons” en el año de 1969, en el que hacen una fuerte crítica al modelo de Rosenblatt. En esta publicación se hace notar la limitación del perceptrón para aprender la función lógica XOR (disyunción exclusiva). Además, estos autores realizan una campaña para desacreditar la investigación en redes neuronales artificiales y desviarla a otros paradigmas. Ante este panorama algunos de los principales grupos de investigación en redes neuronales abandonan sus proyectos y dirigen sus trabajos hacia la inteligencia artificial simbólica. Entre estos grupos de investigación se destacan el de Widrow, quién aplica sus técnicas de redes neuronales artificiales a la ingeniería de telecomunicaciones. Así como el grupo de Rosen, cuyo proyecto de investigación se enfoca en la construcción de un robot móvil siguiendo el paradigma de la inteligencia artificial simbólica. Sin embargo, y contrario a la tendencia adoptada por sus contemporáneos, Rosenblatt continúa sus investigaciones en el paradigma de redes neuronales [13].

Cabe señalar que los líderes actuales en redes neuronales artificiales inician sus publicaciones en la década de los años setentas. De acuerdo con Montaño [13] los que encabezan la lista son:

- Teuvo Kohonen de Finlandia, que en 1960 desarrolla un modelo similar al de Anderson y en 1982 propone el modelo topográfico con aprendizaje autoorganizado, en el que las unidades se distribuyen según el tipo de

entrada al que responden; éste modelo es conocido como Mapas Autoorganizados de Kohonen.

- Von Der Malsburg de Alemania, quien en 1973 da a conocer un modelo con neuronas que responden a la orientación de los objetos.
- Kunihiko Fukushima de Japón, que desarrolla el "congnitrón", una red neuronal autoorganizada para el reconocimiento de patrones que es superada por el "neocognitrón". Ambos modelos especializados en la agrupación de objetos de entrada en clases (clusters) diferentes acorde a una medida de similitud.
- Stephen Grossberg de Estados Unidos, se especializa en el estudio de los mecanismos de percepción y memoria en los seres humanos, gracias a lo cual en 1967 desarrolla la red neuronal denominada "avalancha", para resolver actividades tales como el reconocimiento continuo del habla y el aprendizaje de los movimientos del brazo de un robot.

  Dentro de la gran cantidad de aportaciones de Stephen Grossberg al campo de las redes neuronales se encuentran la Teoría de la Resonancia Adaptativa (ART, por sus siglas en inglés Adaptative Resonance Theory) desarrollada al lado de Gail Carpenter, que se aplica a modelos con aprendizaje competitivo, denominados:
  a) modelo ART, define un esquema de aprendizaje no supervisado y,
  b) modelo ARTMAP denominación dada a los esquemas con aprendizaje supervisado.

  Debe hacerse notar que el aprendizaje competitivo se presenta cuando la señal de salida es generada por una única neurona que ha alcanzado su valor de respuesta máximo después de competir con las demás neuronas.
- James Anderson, quien en 1977 desarrolla un asociador lineal de patrones perfeccionado en el modelo Brain State in a Box (BSC), que alcanza la modelación de funciones arbitrariamente complejas [4].

Gracias a las investigaciones y proyectos desarrollados por los seguidores de las redes neuronales después del declive sufrido, es que en la década de los años ochentas los sistemas conexionistas emergen del olvido al que habían sido confinados. Aunque hay situaciones que favorecen el resurgimiento de las redes neuronales como paradigma de investigación. Por ejemplo, el hecho de que la inteligencia artificial simbólica se encuentra en la fase de comercialización, así como el que los científicos toman conciencia de las limitaciones de los sistemas simbólicos (la incapacidad para el reconocimiento de objetos y del lenguaje hablado o la inhabilidad para el razonamiento de sentido común). Lo anterior propicia que los científicos busquen líneas de investigación alternativas como los sistemas conexionistas.

Aún cuando los investigadores mencionados realizan grandes aportaciones al paradigma de redes neuronales, hace falta el reconocimiento de la comunidad científica, este reconocimiento llega cuando en el año de 1982 el destacado físico John Hopfield publica en la Academia Nacional de la Ciencias un importante artículo sobre redes neuronales. El artículo que presenta Hopfield contiene una

clara descripción de una variante del asociador lineal de Anderson conocida como la “Red de Hopfield” e inspirada en la minimización de la energía presente en los sistemas físicos [4]. Dicho artículo, además de lograr el reconocimiento científico de los sistemas conexionistas, impulsa la implementación de los modelos de red mediante el uso de dispositivos electrónicos que emplean tecnología VLSI (Very Large Scale of Integration) y sugiere una estrecha relación entre los sistemas físicos y las redes neuronales.

En el mismo año de la publicación hecha por Hopfield, se celebra la “U.S.-Japan Joint Conference on Cooperative/Competitive Neural Networks” [4], que reabre el paradigma de redes neuronales a gran escala. Una vez que esto sucede, de la comunidad científica surgen nuevos discípulos entre los se encuentran: Geoffrey Hinton y Terrence J. Sejnowski pertenecientes al grupo PDP (Paralell Distributed Processing) en la Universidad de San Diego California, quienes desarrollan la denominada Máquina de Boltzman; una red neuronal cuyo algoritmo de aprendizaje para la modificación de conexiones fue altamente aceptado por los conocedores.

David Rumelhart, Geoffrey Hinton y Ronald Williams pertenecientes también al grupo PDP, proponen la técnica de Backpropagation, misma que es desarrollada con anterioridad por Paul Werbos en 1974 y redescubierta por otros grupos de investigadores. Cabe destacar que Rosenblatt en 1962 ya tenía idea de ésta técnica, sin embargo, no la desarrolla de manera satisfactoria [13]. La técnica de backpropagation se considera la contribución más importante en el resurgimiento del conexionismo desatando el interés del mundo científico alrededor de las redes neuronales a mediados de los ochentas. El éxito de la técnica desarrollada por Rumelhart, Hinton y Williams se debe a que pone fin al principal problema presentado por el perceptrón multicapa de los años sesentas (la falta de conocimiento del error cometido por la red en las capas intermedias).

Finalmente, en 1985 se confirma la postura de la comunidad científica respecto a las redes neuronales artificiales con la primera reunión anual “Neural Networks for Computing” organizada por el Instituto Americano de Física. Siguiendo en 1987 la primera conferencia internacional sobre redes neuronales de la IEEE y la Sociedad Internacional de Redes Neuronales (INNS por sus siglas en inglés “International Neural Network Society) dirigida por Grossberg de Estados Unidos, Kohonen de Finlandia y Amari de Japón [4]. La evolución histórica de las redes neuronales culmina con el surgimiento, crecimiento e institucionalización de una comunidad científica enfocada al tema.

Capítulo 2
Elementos que integran el diseño de RNAs

2.1 Arquitecturas de redes neuronales artificiales

La arquitectura de las redes neuronales artificiales se puede definir como la forma en que se interconectan las neuronas dentro de la red, junto con el algoritmo de aprendizaje que se ha de utilizar. Dentro de las arquitecturas neuronales de uso común se encuentran las que a continuación se exponen:

I. *Redes Feedforward de capa simple*
   En una red de capa simple, las neuronas están organizadas en forma de capas. La forma más simple de este tipo de red toma una capa de entrada de nodos origen que son proyectados sobre una capa de neuronas de salida.

   Las redes de capa simple no son de tipo cíclico, sino de avance hacia delante (feedforward) como se muestra en la Fig. 2.1. Un ejemplo de éste tipo de arquitectura es el perceptrón.

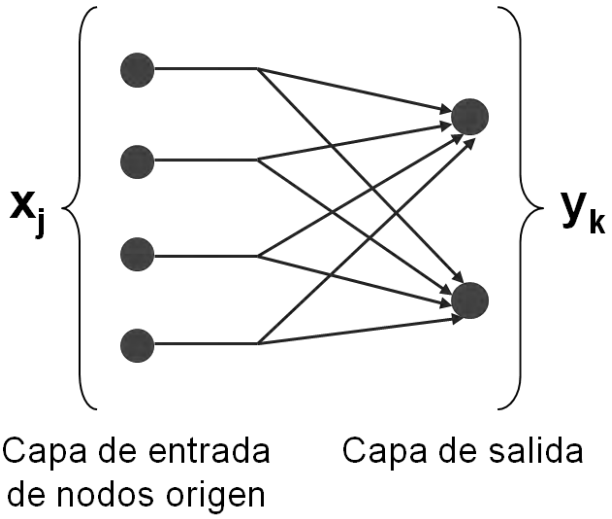


Fig. 2.1 Diagrama de una red feedforward de capa simple

II. *Redes Feedforward de capas múltiples o multicapa*
Este tipo de red se distingue del anterior debido a la presencia de capas ocultas que utilizan como señales de entrada las señales de salida generadas por la capa anterior hasta llegar a la capa de salida. Cabe señalar que las entradas en la primera capa oculta corresponden a las señales de entrada externas o nodos origen de la red. Un ejemplo gráfico de una red feedforward multicapa es el que se muestra en la Fig. 2.2.

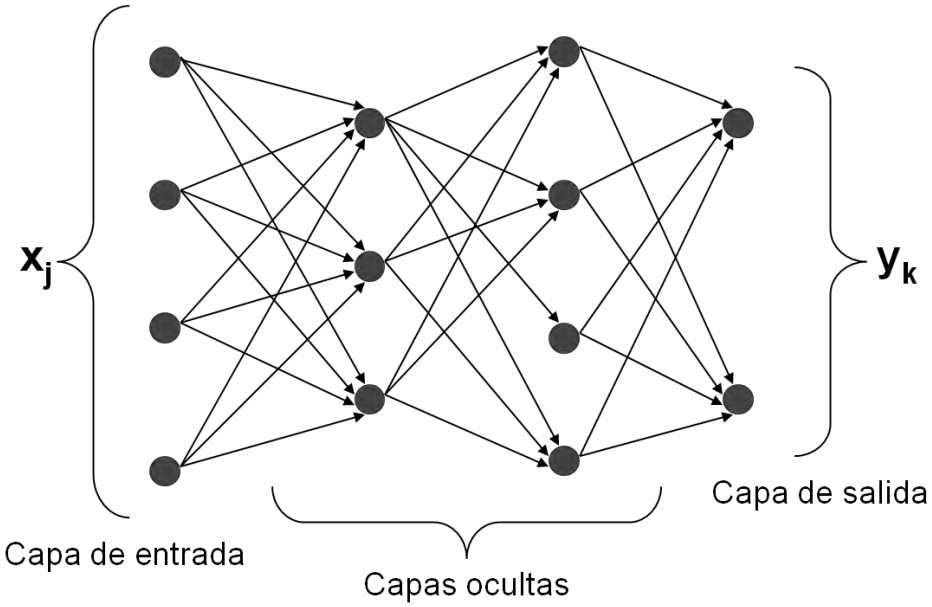


Fig. 2.2 Diagrama de una red feedforwad multicapa

Cabe señalar que en las redes feedforward el flujo de información permite que todos los estados de las neuronas se actualicen al mismo tiempo o siguiendo una secuencia determinista [18].

III. *Redes Recurrentes*
Una arquitectura de este tipo se distingue por el uso de ciclos de retroalimentación que pueden clasificarse en:

- Retroalimentación a sí misma. Cuando la salida generada por una neurona sirve a sí misma como una nueva entrada. Un ejemplo de este tipo de retroalimentación se muestra en la Fig. 2.3.

- Retroalimentación a otras neuronas. Cuando la salida generada por una neurona sirve a otras pero no a sí misma como una nueva señal de entrada. Véase un ejemplo en la Fig. 2.4.

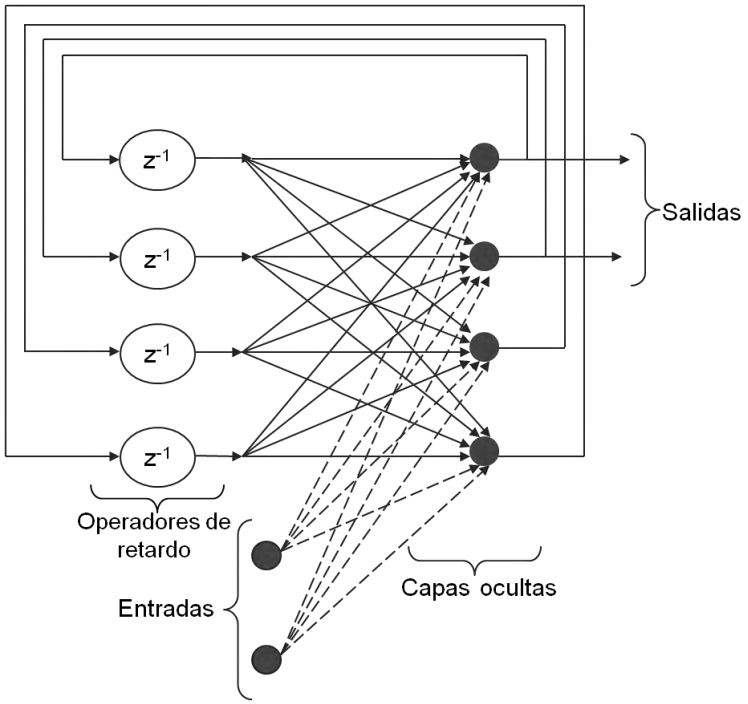


Fig. 2.3 Red recurrente con capas ocultas y retroalimentación a sí misma [2]

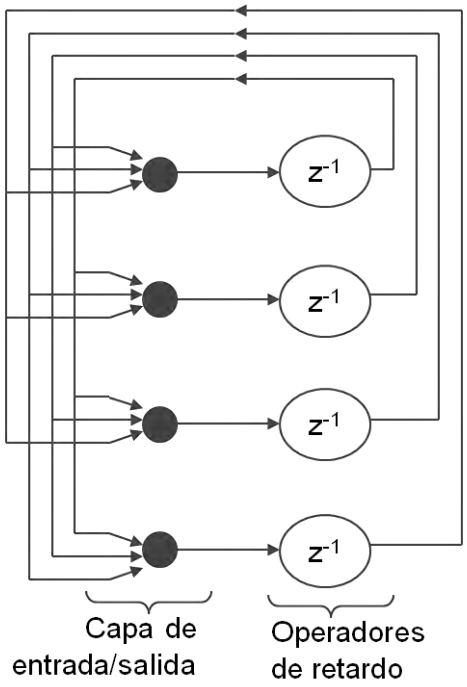


Fig. 2.4 Red recurrente sin capas ocultas ni retroalimentación a sí misma [2]

A diferencia de las redes feedforward, las redes recurrentes permiten la interconexión de neuronas en cualquier sentido y la inclusión de operadores de retardo. Generalmente, las redes recurrentes presentan un comportamiento en el cual las neuronas reciben las salidas de capas anteriores junto con las salidas de la misma capa o de capas superiores.

Una característica de las redes recurrentes es que permiten modelar la *no linealidad* y los *componentes dinámicos* de un sistema. El flujo de información en las redes recurrentes, suele ser asíncrono, es decir, el estado de cada una de las neuronas depende del comportamiento interno de sí misma o bien de otras neuronas.

2.2 Representación de la información

En la presente investigación todos los datos de entrada y salida son representados de manera abstracta a través de vectores. Estos arreglos n-dimensionales pueden ser por ejemplo: la representación vectorial de una imagen, de una agrupación de notas musicales, de una serie de síntomas, etc. En términos generales se definen como un conjunto de características que detallan el estado de un objeto o situación.

Durante la construcción de una red es importante determinar qué información está disponible y cuál es necesaria para generar la salida correcta. No hay que olvidar que estos datos sirven para el entrenamiento de la red a través de ejemplos y que además, la modificación de los parámetros libres de la red (ω_{kj} peso sináptico y b_k valor umbral) ha de reflejar su aprendizaje.

Para determinar cómo incorporar de forma grata esta información al sistema no hay una regla establecida, sin embargo, hay quienes con su experiencia han definido algunos lineamientos que son de ayuda durante la fase de diseño. A continuación se enlistan tres de ellos que han sido tomados de Haykin [2].

Regla 1. Entradas similares provenientes de clases similares deben producir representaciones similares dentro de la red y por lo tanto, deben ser clasificadas como miembros de la misma categoría. Para determinar el grado de similaridad de las entradas se puede hacer uso de alguna de las siguientes dos técnicas:

I. *Distancia euclidiana entre vectores*.
Sean los vectores

$$x_i = [x_{i1}, x_{i2}, \cdots, x_{im}]^T \quad (2.1)$$

$$x_j = [x_{j1}, x_{j2}, \cdots, x_{im}]^T \quad (2.2)$$

la distancia euclidiana entre ambos se define como:

$$d(x_i, x_j) = \|x_i - x_j\| = \left[\sum_{k=1}^{m} (x_{ik} - x_{jk})^2\right]^{\frac{1}{2}} \quad (2.3)$$

de aquí que la similaridad se defina como el recíproco de la distancia euclidiana, es decir, a menor distancia mayor grado de similaridad entre los vectores.

II. *Producto interno*.
Sean x_i y $x_j \in \mathcal{R}^m$, el producto interno entre x_i y x_j se define como:

$$\begin{aligned} \langle x_i, x_j \rangle &= x_i^T x_j \\ &= \sum_{k=1}^{m} x_{ik} x_{jk} \end{aligned} \qquad 2.4$$

Cabe señalar que el producto interno $\langle x_i, x_j \rangle$ dividido por $\|x_i\|\|x_j\|$, es el coseno del ángulo entre los vectores x_i y x_j.

Es importante recalcar que el producto interno y la distancia euclidiana entre vectores están íntimamente relacionados como se muestra en la Fig. 2.5, donde a mayor ángulo entre los vectores x_i y x_j menor producto interno. Por lo tanto, el mayor grado de similaridad posible tiende a minimizar la distancia y maximizar el producto interno.

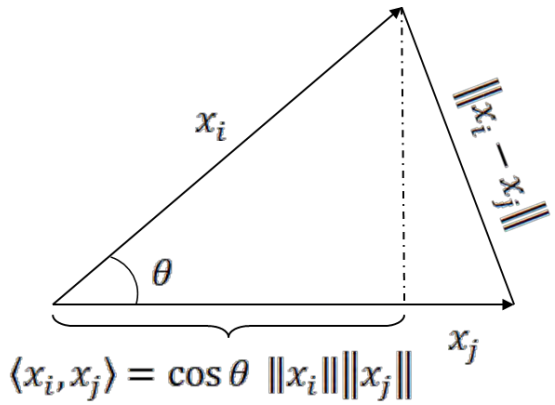


Fig. 2.5 Relación entre producto interno y distancia euclidiana

Regla 2. Entradas categorizadas como clases separadas deben dar representaciones muy diferentes en la red.

Regla 3. Si una característica es importante, entonces debe existir un número importante de neuronas involucradas en la representación de dicha característica.

2.3 Proceso de aprendizaje

El aprendizaje de la red se lleva a cabo a partir de la interacción de la red con su ambiente dando como resultado la modificación de los pesos sinápticos y sus respectivos valores umbral.

De lo anterior se desprende que entre mayor es la interacción de la red neuronal con el medio que la rodea, mayor es el nivel de aprendizaje que se genera, lo que se traduce en una mejora en la habilidad de la red para comportarse correctamente ante situaciones diferentes. A continuación se proporciona una definición formal de aprendizaje:

> "… un proceso mediante el cual los parámetros libres en una red neuronal son adaptados a través de un proceso de estimulación proveniente del ambiente en el cual la red es implantada. El tipo de aprendizaje es determinado por la forma en la que los cambios en el parámetro toman lugar" [19].

Siguiendo esta definición y de acuerdo con Haykin [2], el proceso de aprendizaje involucra la siguiente secuencia de eventos:

I. La red neuronal es estimulada por el ambiente.
II. La red neuronal efectúa cambios en sus parámetros (w_{kj}, b_k) como resultado de la estimulación.
III. La red neuronal responde de diferente manera al medio debido a los cambios ocurridos en su estructura interna.

El conjunto de reglas definidas para solucionar un problema de aprendizaje se denomina *algoritmo o regla de aprendizaje*. Existen diversos algoritmos para tal efecto, cada uno de los cuales ofrece alguna ventaja sobre los otros. Aunque la diferencia radica en la forma mediante la cual los pesos sinápticos y los valores umbral son modificados.

2.3.1 Reglas de aprendizaje

Las reglas de aprendizaje son algoritmos que sigue la red de neuronas para poder emitir una respuesta a estímulos de entrada. Existe una gran cantidad de reglas de aprendizaje, sin embargo, a continuación sólo se describen de manera breve las más comunes con el fin de completar el marco teórico de la investigación.

1. *Aprendizaje por corrección del error*

Este método consiste en ajustar los pesos sinápticos a partir del error generado por la diferencia entre la señal de salida producida por la red y la respuesta deseada.

2. *Aprendizaje basado en la memoria*

Es un método simple de aproximación de funciones que almacena datos en una memoria (base de datos). La salida es una predicción que utiliza los atributos de la entrada sobre la base de datos mediante la búsqueda de puntos similares en la memoria, el ajuste del modelo local para esos puntos y una predicción basada en el modelo. Hay cuatro componentes que definen a un sistema basado en la

memoria: una métrica de distancia, el número de vecinos más cercanos, una función de ponderación y un modelo local [20].

3. *Aprendizaje de Hebb*

La regla de Hebb plantea que si una neurona participa reiteradamente en la activación de otra, entonces, la fuerza de conexión que las une (es decir, la habilidad que posee la primera neurona de activar a la segunda) se incrementa [21]. La Fig. 2.6 ilustra el postulado de Hebb: j neuronas $(j = 1, 2, 3)$ están conectadas a la neurona k a través de sus respectivas sinapsis.

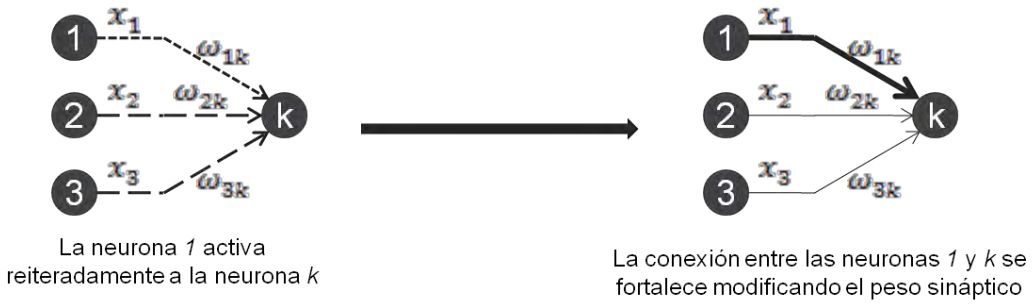


Fig. 2.6 Ilustración gráfica del postulado de Hebb

Tomando el modelo de la Fig. 2.6 y bajo el supuesto de que la neurona $j = 1$ activa reiteradamente a la neurona k, entonces la regla de Hebb establece que la conexión entre la neurona $j = 1$ y la neurona k se fortalece, lo que se traduce en el incremento del peso sináptico de la conexión.

4. *Aprendizaje competitivo*

En el aprendizaje competitivo las neuronas compiten entre sí para llegar a ser neuronas de activación. Este tipo de aprendizaje permite exclusivamente la activación de sólo una neurona de salida en cada instante. Además, cada neurona aprende a especializarse en un conjunto de patrones similares, llegando a convertirse en detectores de características para diferentes patrones de entrada.

En la forma más simple de aprendizaje competitivo, la red neuronal cuenta con una única capa de neuronas de salida, cada una de las cuales está fuertemente conectada a los nodos de entrada. La red puede incluir conexiones feedforward entre las neuronas de salida. En la Fig. 2.7, las conexiones feedforward ejecutan inhibición lateral, es decir, cada neurona tiende a inhibir a la neurona con la cual está lateralmente conectada. En contraste, las conexiones sinápticas son todas excitatorias.

5. *Aprendizaje de Boltzman*

La regla de aprendizaje de Boltzman, en honor a su creador Ludwing Boltzman, es un algoritmo de aprendizaje estocástico donde las neuronas constituyen una estructura recurrente que opera de forma binaria tomando el estado de *encendido* o *apagado*.

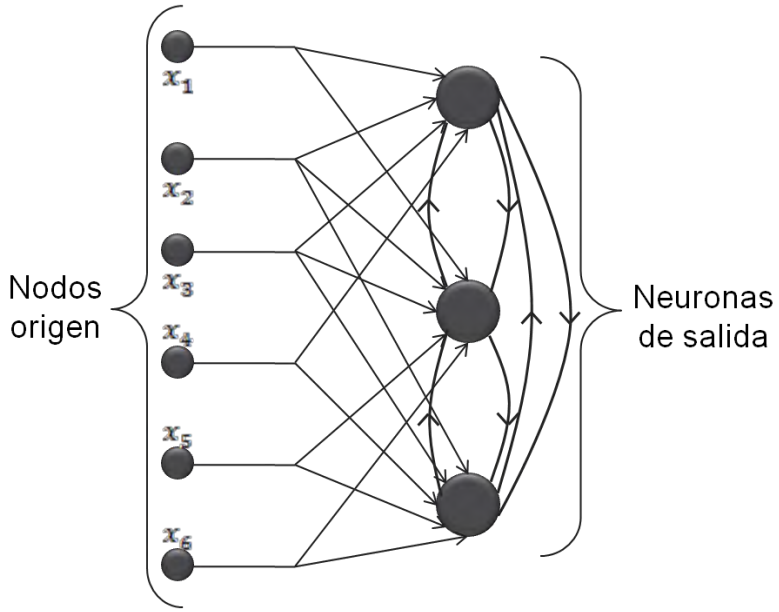


Fig. 2.7 Aprendizaje competitivo con inhibición lateral [2]

Una máquina de Boltzman se caracteriza por una *función del error o función de energía*, E, cuyo valor es determinado a partir del estado que toma cada neurona de la red. La dinámica de la máquina permite encontrar el verdadero mínimo global de la función de energía, mediante un proceso similar al templado simulado[1]. La operación consiste en elegir aleatoriamente una neurona x_k, y a continuación, cambiar su estado x_k al estado $-x_k$ bajo una condición de temperatura T que de acuerdo con Haykin [2] tiene una probabilidad de:

$$P\left(x_k \rightarrow -x_k = \frac{1}{1 + exp(-\Delta \mathrm{E_k}/\mathrm{T})}\right) \qquad 2.5$$

Donde:

T: Temperatura (no hace referencia a una temperatura física).
$\Delta \mathrm{E}_k$: Cambio en la función de energía de la máquina.

Cabe mencionar que si la regla de Boltzman es aplicada repetidamente, la máquina alcanza un estado de equilibrio térmico.

Las cinco reglas de aprendizaje descritas en párrafos anteriores son las más utilizadas en el campo de las redes neuronales artificiales y forman una gama de opciones que permiten al investigador adaptar diferentes problemas a un modelo basado en neuronas.

---

[1] El templado simulado (conocido como simulated annealig, en inglés) se inspira en el mecanismo para templar materiales al calentarlos hasta muy altas temperaturas y luego irlos enfriando paulatinamente [23].

2.3.2 Paradigmas de aprendizaje

El paradigma de aprendizaje depende de la estructura de la red y las características de los datos con los que trata. Algunas veces el término "ambiente" en lugar de "paradigma" es preferible, ya que tiene un significado más genérico [22]. Estos ambientes de aprendizaje puede clasificarse en:

1. *Aprendizaje con maestro*

También conocido como aprendizaje supervisado, se define como aquél en el cual el maestro representa el conocimiento del medio que rodea a la red y es simbolizado por un conjunto de ejemplos entrada/salida. Suponiendo que tanto el maestro como la red son expuestos a un vector de entrenamiento proveniente del ambiente, entonces el maestro es capaz de proveer a la red con la respuesta deseada para el ejemplo dado. Más aún, la respuesta dada representa la acción optima a ejecutar por la red.

Los parámetros de la red son ajustados bajo la influencia combinada del vector de entrenamiento y la señal de error, donde ésta última se define como la diferencia entre la respuesta deseada y la generada por la red. El ajuste de los parámetros se lleva a cabo de manera iterativa con el objetivo de que la red imite al maestro, dicha imitación se supone óptima bajo algún criterio estadístico. En este sentido, el conocimiento del ambiente disponible al maestro es transferido a la red neuronal a través de un entrenamiento tan completo como sea posible. Una vez que se ha alcanzado un estado óptimo de imitación la red abandona el uso del maestro y trata con el medioambiente por sí misma; a manera de ilustración véase la Fig. 2.8.

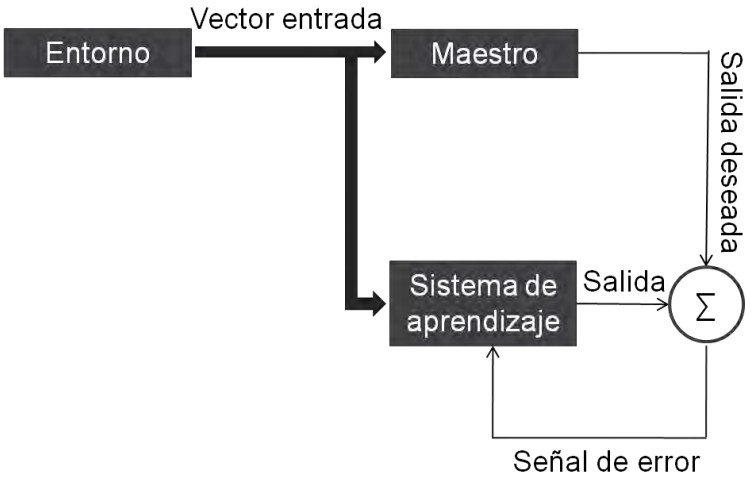


Fig. 2.8 Diagrama de bloques de un aprendizaje supervisado [2]

2. *Aprendizaje sin maestro*

En este tipo de aprendizaje no existe un tutor a cargo del proceso de aprendizaje, lo que se traduce como la falta de ejemplos. Bajo este paradigma se identifican, de acuerdo con Haykin [2] dos tipos diferentes de aprendizaje:

a) Aprendizaje por reforzamiento o programación neurodinámica.
   En este caso, el mapeo entrada/salida se ejecuta a través de una interacción continua con el ambiente. Una red que emplea este tipo de aprendizaje se construye con un *crítico*, quién se encarga de convertir una señal recibida del medioambiente denominada *señal de reforzamiento primaria,* en una señal con mayor calidad llamada *señal de reforzamiento heurístico*. La red aprende bajo un esquema de *reforzamiento retrasado*, es decir, el sistema observa una secuencia temporal de estímulos (vectores de estado) recibidos del medio que, eventualmente, tiende a la generación de una *señal de reforzamiento heurístico*. El objetivo del procedimiento es minimizar una función costo (definida como la esperanza del costo acumulado de acciones tomadas sobre una secuencia de pasos) en lugar del costo inmediato.

b) Aprendizaje no supervisado
   Este modelo, también conocido como auto-organizado, no cuenta con crítico que esté pendiente del proceso de aprendizaje; por el contrario, se define un índice que mide la calidad de las tareas independientes que la red necesita aprender para posteriormente optimizar los parámetros libres de acuerdo a este índice.

En general, para diseñar un sistema basado en redes neuronales artificiales es necesario definir componentes como: arquitectura de las neuronas, información que provee a la red del conocimiento de su medio, regla de aprendizaje que rige el comportamiento de las neuronas y paradigma de aprendizaje para tutelar la modificación en las conexiones sinápticas. A continuación se da paso a la propuesta de un modelo de memoria asociativa con redes neuronales artificiales mediante el enfoque de los sistemas dinámicos, que permite asociar un patrón previamente aprendido, con alguno que se encuentra incompleto.

Capítulo 3
Memoria Asociativa con RNAs

Los mecanismos neuronales para almacenar y recuperar la información acumulada con el paso del tiempo es al día de hoy un tema de investigación abierto. Y aunque la memoria puede considerarse como un cambio adaptativo en la respuesta del organismo basado en la experiencia [23], se basa en la modificación de los pesos sinápticos existentes entre las conexiones de una red neuronal. Tal modificación puede ser mediante la formación de una nueva sinapsis o bien la eliminación o alteración de una sinapsis existente.

De acuerdo con Anderson [23], la memoria de los seres vivos es asociativa, es decir, un estímulo se enlaza a una respuesta del tal modo que cada vez que dicho estímulo se hace presente entonces la respuesta asociada se presenta.

En términos generales se dice que un modelo tiene memoria de contenido direccional (asociativa), si el conocimiento parcial de un patrón guía, a través de un proceso dinámico, a la recuperación completa del patrón de memoria. En el lenguaje matemático el estado de n neuronas se representa por un vector x de números reales de dimensión n. Por lo que el espacio de estados del sistema se conforma de vectores $x = (x_1, x_2, \cdots, x_n)$ de números reales. Por otro lado, la función de activación $\varphi_i(x_i)$ se define como una función continuamente diferenciable y creciente para $|x_i| \leq \varepsilon$, con $\varphi_i = 0$ para $x_i \leq -\varepsilon$ y $\varphi_i = 1$ para $x_i \geq \varepsilon$, tal que ε sea cualquier número positivo menor a uno.

3.1 Definición de memoria asociativa

El objetivo de un sistema de redes neuronales con memoria asociativa consiste en asociar una respuesta a un valor de entrada dado, en otros términos, relacionar vectores de entrada con vectores de salida aprendidos previamente aún cuando se haya añadido algún tipo de ruido en las entradas.

En términos formales si se considera que la vecindad de un vector de entrada x conocido puede ser mapeada a la imagen y de x, entonces es de esperarse que la red mapee $\beta(x)$ a y, denotando a $\beta(x)$ como todos los vectores cuya distancia de x es más pequeña que alguna constante positiva ε. Siendo así que el ruido en los vectores de entrada puede ser asociado con la salida correcta [10].

3.1.1 Tipos de memoria asociativa

Una red de neuronas artificiales empleada como memoria asociativa para la recuperación de la versión correcta de un patrón de memoria a partir de una versión incompleta o corrompida, no debe confundirse con una base de datos que recupera direcciones de memoria a partir de claves o índices. Ya que en el primer caso la parte faltante está asociada con la parte existente y la información de muchas asociaciones se hace presente en las fuerzas de conexión [23].

De acuerdo con Rojas [10], este tipo de redes artificiales se clasifica en:

- *Redes heteroasociativas*. Mapean m vectores de de entrada $x^1, x^2, \cdots, x^m$ de dimensión n, sobre m vectores de salida $y^1, y^2, \cdots, y^m$, de dimensión k, de manera que $x^i \mapsto y^i$.
- *Redes autoasociativas*. Son un subconjunto de las redes heteroasociativas, en las que cada vector es asociado consigo mismo, es decir, $y^i = x^i$ para $i = 1, \cdots, m$. La función de este tipo de redes es corregir el ruido existente en los vectores de entrada.

3.1.2 Estructura de una memoria asociativa

Si se toma como punto de partida la red de la Fig. 3.1, entonces se define a ω_{ij} como el peso entre la señal de entrada i y la neurona j, $\mathbf{W}$ como la matriz de pesos $[\omega_{ij}]$ de dimensión $n \times k$ y $x = (x_1, x_2, \cdots, x_n)$ como el vector de entrada a la red. Siendo el vector de excitación e la operación

$$e = x\mathbf{W} \tag{3.1}$$

Posteriormente, cada unidad, evalúa la función de activación a partir del vector de excitación alcanzado. Por lo general se asocian m diferentes vectores fila $x^1, x^2, \cdots x^m$ de dimensión n, con m vectores fila $y^1, y^2, \cdots, y^m$ de dimensión k. Así que si se denota a $\mathbf{X}$ como la matriz de $m \times n$ cuyas filas son cada uno de los vectores de entrada, entonces, $\mathbf{Y}$ es la matriz de $m \times k$ cuyas filas son los vectores de salida, por lo que la asociación de ambas matrices con la matriz de pesos $\mathbf{W}$ conduce a

$$\mathbf{XW} = \mathbf{Y} \tag{3.2}$$

En el caso de *redes autoasociativas* cada vector es asociado consigo mismo por lo que la ecuación 3.2 se torna

$$\mathbf{XW} = \mathbf{X} \tag{3.3}$$

Si $m = n$, entonces $\mathbf{X}$ es una matriz cuadrada, y si ésta matriz es invertible, la solución de 3.2 es

$$\mathbf{W} = \mathbf{X}^{-1}\mathbf{Y} \tag{3.4}$$

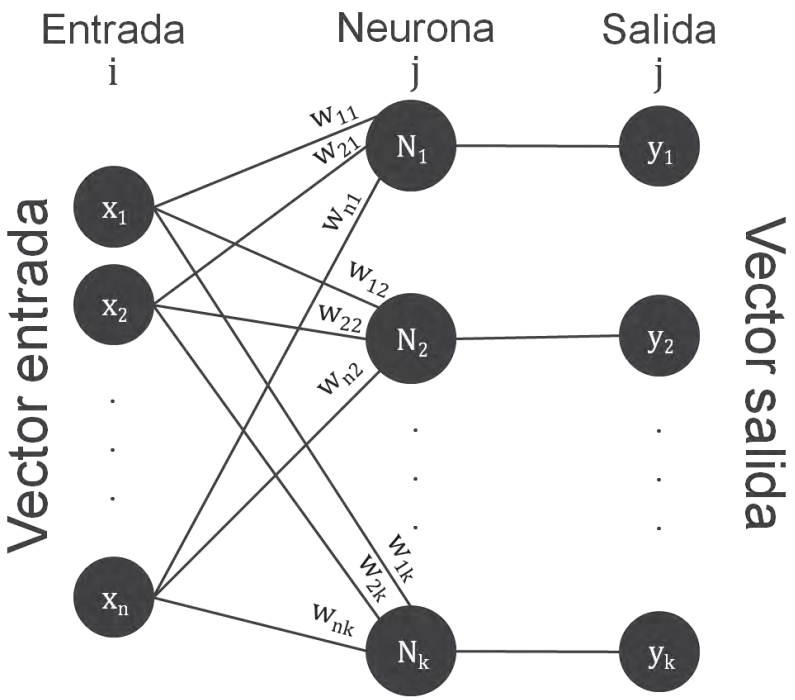


Fig. 3.1 Red con k neuronas artificiales

Lo que significa que encontrar **W** equivale a resolver el sistema de ecuaciones. En el caso de las *redes recurrentes autoasociativas*, la salida arrojada por el sistema es empleada como una nueva señal de entrada a la red. Si todas las unidades ejecutan su salida simultáneamente entonces se les denomina *redes síncronas*. Cabe señalar que en cada iteración la red es alimentada con un vector entrada $x(i)$ y produce una nueva salida $x(i+1)$. Lo importante en este tipo de redes es la existencia de un punto fijo $\vec{\xi}$ tal que

$$\vec{\xi}\mathbf{W} = \vec{\xi} \qquad 3.5$$

El vector $\vec{\xi}$ es un vector propio de **W** con valor propio de 1. En este caso la red se comporta como un sistema dinámico de primer orden, donde cada nuevo estado $x(i+1)$ está determinado por su más reciente predecesor [10]. Si la matriz de pesos es vista como un sistema dinámico entonces el principal interés reside en los puntos fijos del sistema, sin embargo, no todas las matrices de pesos convergen a un estado estable.

Las matrices cuadradas con un conjunto completo de valores propios son más útiles para propósitos de almacenamiento dado que una matriz **W** de $n \times n$ tiene a lo sumo n valores propios linealmente independientes, donde los vectores propios de **W**, $x^1, x^2, \cdots, x^n$ satisfacen 3.6.

$$x^i\mathbf{W} = \lambda_i x^i \quad \text{para } i = 1, \cdots, n \qquad 3.6$$

Siendo $\lambda_1, \cdots, \lambda_n$ cada uno de los valores propios de la matriz. Cada matriz de pesos con un conjunto completo de vectores propios define un tipo de eigenvector dominante[10].

Asúmase sin perder generalidad que λ_1 es el eigenvalor de $\mathbf{W}$ cuya magnitud es superior a la de sus semejantes, esto es, $|\lambda_1| > |\lambda_i|$ para $i \neq 1$. Ahora bien, si se admite $\lambda_1 > 0$ y se selecciona aleatoriamente un vector a_0 diferente de cero de dimensión n, entonces a_0 puede expresarse como una combinación lineal de n eigenvectores de la matriz $\mathbf{W}$:

$$a_0 = \alpha_1 x^1 + \alpha_2 x^2 + \cdots + \alpha_n x^n \qquad 3.7$$

Si se asume que todas las constantes α son diferentes de cero. Después de la primera iteración con la matriz de pesos $\mathbf{W}$ se obtiene

$$\begin{aligned} a_1 &= a_0\mathbf{W} \\ &= (\alpha_1 x^1 + \alpha_2 x^2 + \cdots + \alpha_n x^n)\mathbf{W} \\ &= \alpha_1\lambda_1 x^1 + \alpha_2\lambda_2 x^2 + \cdots + \alpha_n\lambda_n x^n \end{aligned} \qquad 3.8$$

Una vez realizadas t iteraciones el resultado es

$$a_t = \alpha_1\lambda_1^t x^1 + \alpha_2\lambda_2^t x^2 + \cdots + \alpha_n\lambda_n^t x^n \qquad 3.9$$

Se observa que el eigenvalor λ_1, cuya magnitud es superior a la de los demás, domina la expresión anterior para un t suficientemente grande. El vector a_t puede ser llevado arbitrariamente muy cerca de x^1. En cada iteración de la red asociativa, el vector x^1 atrae algún otro vector a_0 cuya componente α_1 es diferente de cero.

3.1.3 Aprendizaje asociativo

Considerar a una red de neuronas artificiales como un sistema dinámico caracterizado por un vector de estado $x = (x_1, x_2, \cdots, x_n)$, que indica el valor que toma cada neurona x_i en cualquier instante de tiempo, permite volcar sobre el modelo toda la teoría que hay detrás de los sistemas dinámicos y facilita el análisis de su comportamiento al aportar métodos gráficos y algebraicos. Además, abre la posibilidad de estudiar comportamientos cíclicos o caóticos en caso de que los haya.

En esta investigación se suponen a las redes asociativas como sistemas dinámicos cuyos atractores son aquellos vectores que han de ser aprendidos por la red. La clave en el diseño es localizar tantos atractores en el espacio de entrada como sea posible, cada uno de ellos con una región de influencia bien definida y delimitada, para evitar la presencia de un eigenvector dominante que absorba casi todo el espacio de entradas. Para lograrlo se recomienda utilizar neuronas no lineales con funciones de activación no lineal que tengan valores de salida $+1$ y

-1, en lugar de aquellas que mapean 1 y 0. Ya que vectores con componentes ± 1 tienen mayor probabilidad de ser ortogonales [10].

3.1.3.1 Aprendizaje Hebbiano

Para llevar el postulado de Hebb a una perspectiva de modelación matemática, considérese una red de capa simple con k unidades y función signo como función de activación, entonces, es necesario buscar los pesos apropiados que permitan mapear el vector entrada x con el vector salida y de dimensiones n y k respectivamente. La regla de aprendizaje a emplear en este caso es el aprendizaje de Hebb que responde a la idea de que dos neuronas que son activadas simultáneamente pueden desarrollar un grado de interacción más fuerte que aquellas neuronas cuyas actividades no se encuentran correlacionadas.

El aprendizaje de Hebb en redes asociativas consiste en la actualización del peso sináptico ω_{ij} a través del incremento $\Delta\omega_{ij}$ que mide la correlación entre la entrada x_i y la salida y_j. Rojas [10] lo define como

$$\Delta\omega_{ij} = \gamma x_i x_j \tag{3.10}$$

Donde γ es la constante que determina el índice de aprendizaje en la ecuación. Es importante señalar que la matriz de pesos $\mathbf{W}$ es la matriz cero antes de que el aprendizaje haya sido iniciado. La regla de aprendizaje se aplica a todos los pesos del vector entrada $x^1 = (x_1^1, x_2^1, \cdots, x_n^1)$ y del vector salida $y^1 = (y_1^1, y_2^1, \cdots, y_k^1)$. Por lo que la matriz de pesos actualizada corresponde a la matriz de correlación de ambos vectores y tiene la forma

$$\mathbf{W} = \left[\omega_{ij}\right]_{n\times k} = \left[x_i^1 y_j^1\right]_{n\times k} = (x^1)^T y^1$$

$$\mathbf{W} = \begin{bmatrix} \omega_{11} & \omega_{12} & \cdots & \omega_{1k} \\ \omega_{21} & \omega_{22} & & \omega_{2k} \\ \vdots & & \ddots & \vdots \\ \omega_{n1} & \omega_{n2} & \cdots & \omega_{nk} \end{bmatrix}_{n\times k} = \begin{bmatrix} x_1^1 y_1^1 & x_1^1 y_2^1 & \cdots & x_1^1 y_k^1 \\ x_2^1 y_1^1 & x_2^1 y_2^1 & & x_2^1 y_k^1 \\ \vdots & & \ddots & \vdots \\ x_n^1 y_1^1 & x_n^1 y_2^1 & \cdots & x_n^1 y_k^1 \end{bmatrix}_{n\times k} \tag{3.11}$$

La matriz $\mathbf{W}$ mapea el vector x^1 diferente de cero exactamente al vector y^1. Obsérvese que la excitación de las k unidades de salida de la red está dada por 3.12.

$$
\begin{aligned}
x^1\mathbf{W} &= [x_1^1, x_2^1, \cdots, x_n^1]_{1\times n}\begin{bmatrix} x_1^1y_1^1 & x_1^1y_2^1 & \cdots & x_1^1y_k^1 \\ x_2^1y_1^1 & x_2^1y_2^1 & & x_2^1y_k^1 \\ \vdots & & \ddots & \vdots \\ x_n^1y_1^1 & x_n^1y_2^1 & \cdots & x_n^1y_k^1 \end{bmatrix}_{n\times k} \\
&= \left[x_1^1x_1^1y_1^1 + x_2^1x_2^1y_1^1 + \cdots + x_n^1x_n^1y_1^1, \cdots, x_1^1x_1^1y_k^1 + x_2^1x_2^1y_k^1 + \cdots + x_n^1x_n^1y_k^1\right]_{1\times k} \\
&= \left(y_1^1\sum_{i=1}^{n} x_i^1x_i^1, y_2^1\sum_{i=1}^{n} x_i^1\,x_i^1, \cdots, y_k^1\sum_{i=1}^{n} x_i^1x_i^1\right) \\
&= y^1(x^1\cdot x^1)
\end{aligned}
\tag{3.12}
$$

Donde $x^1 \cdot x^1$ es el producto interno entre vectores definido como

$$x^a \cdot x^b = (x_1^a, \cdots, x_n^a)\left(x_1^b, \cdots, x_n^b\right) = x_1^a x_1^b + \cdots + x_n^a x_n^b = \sum_{i=1}^{n} x_i^a x_i^b \tag{3.13}$$

Por lo que si $x^1 \neq 0$ entonces $x^1 \cdot x^1 > 0$, y la salida de la red es

$$sgn(x^1\mathbf{W}) = (y_1^1, y_2^1, \cdots, y_n^1) = y^1 \tag{3.14}$$

Donde la función signo se aplica a cada componente del vector de excitación.

En general, si se desean asociar m vectores $x^1, x^2, \cdots, x^m$ diferentes de cero con m vectores $y^1, y^2, \cdots y^m$ de dimensiones n y k respectivamente, se debe aplicar el aprendizaje de Hebb a cada pareja de entrada-salida, por lo que el resultado de la matriz de pesos es

$$\mathbf{W} = \mathbf{W}^1 + \mathbf{W}^2 + \cdots + \mathbf{W}^m \tag{3.15}$$

Donde cada matriz $\mathbf{W^t}$ es la matriz de correlación de los vectores x^e y y^e, es decir,

$$\mathbf{W}^e = (x^e)^T y^e = [x_i^e y_i^e]_{n\times k} \tag{3.16}$$

Si la entrada de la red es el vector x^p, entonces, el vector de excitación de las neuronas es igual a y^p multiplicado por una constante positiva más un término de perturbación adicional $\sum_{e\neq p}^{m} y^e(x^e \cdot x^p)$ llamado *interferencia* (véase 3.17). Así que la red origina el vector de salida deseado y^p, cuando la interferencia es cero. Lo que es factible cuando los patrones de entrada $x^1, x^2, \cdots, x^m$ son ortogonales.

Definición: Los vectores $\vec{v}_1, \cdots, \vec{v}_k \in \mathbb{R}^n$ son ortogonales cuando para cualquier par de vectores el producto interno entre ellos es cero [24].

$$
\begin{aligned}
x^p\mathbf{W} &= x^p(\mathbf{W}^1 + \mathbf{W}^2 + \cdots + \mathbf{W}^m) \\
&= x^p\mathbf{W}^p + \sum_{e\neq p}^{m} x^p\mathbf{W}^e \\
&= x^p\mathbf{W}^p + \sum_{e\neq p}^{m} \left[x_1^p, \cdots, x_n^p\right]_{1\times n} \begin{bmatrix} x_1^e y_1^e & \cdots & x_1^e y_k^e \\ \vdots & \ddots & \vdots \\ x_n^e y_1^e & \cdots & x_n^e y_k^e \end{bmatrix}_{n\times k} \\
&= x^p\mathbf{W}^p + \sum_{e\neq p}^{m} \left[x_1^p x_1^e y_1^e + \cdots + x_n^p x_n^e y_1^e, \quad \cdots \quad , x_1^p x_1^e y_k^e + \cdots + x_n^p x_n^e y_k^e\right]_{1\times k} \\
&= y^p(x^p \cdot x^p) + \sum_{e\neq p}^{m} y^e(x^e \cdot x^p)
\end{aligned}
\tag{3.17}
$$

Aún cuando el término de perturbación sea diferente de cero, el mapeo entrada-salida puede seguir funcionando mientras la interferencia sea lo suficientemente más pequeña que $y^p(x^p \cdot x^p)$. En este caso la salida de la red es

$$
sgn(x^p\mathbf{W}) = sgn\left(y^p(x^p \cdot x^p) + \sum_{e\neq p}^{m} y^e(x^e \cdot x^p)\right) \tag{3.18}
$$

Dado que $x^p \cdot x^p$ es una constante positiva, se tiene que

$$
\begin{aligned}
sgn(x^p\mathbf{W}) &= sgn\left(y^p \frac{(x^p \cdot x^p)}{(x^p \cdot x^p)} + \sum_{e\neq p}^{m} y^e \frac{(x^e \cdot x^p)}{(x^p \cdot x^p)}\right) \\
&= sgn\left(y^p + \sum_{e\neq p}^{m} y^e \frac{(x^e \cdot x^p)}{(x^p \cdot x^p)}\right)
\end{aligned}
\tag{3.19}
$$

Ahora bien, para producir la salida deseada y^p la ecuación

$$
y^p = sgn\left(y^p + \sum_{e\neq p}^{m} y^e \frac{(x^e \cdot x^p)}{(x^p \cdot x^p)}\right) \tag{3.20}
$$

debe ser satisfecha. Esto se logra cuando el valor absoluto de todos los componentes del término de perturbación 3.21 son menores a uno.

$$
\sum_{e\neq p}^{m} y^e \frac{(x^e \cdot x^p)}{(x^p \cdot x^p)} \tag{3.21}
$$

Lo que significa que el producto escalar $x^e \cdot x^p$ debe ser pequeño en comparación con la longitud cuadrática del vector x^p. En una selección aleatoria parejas de vectores bipolares (con componentes ± 1), la probabilidad de que sean ortogonales es muy alta, siempre y cuando no muchos de ellos sean seleccionados. Además, esto incrementa la probabilidad de que la interferencia sea pequeña, conduciendo a un conjunto eficiente de pesos sinápticos para la red asociativa [10].

En el caso autoasociativo, en el que el vector x^1 es asociado consigo mismo. La matriz de pesos $\mathbf{W}^1$ también puede ser evaluada a través del aprendizaje de Hebb. Por lo que se tiene

$$\mathbf{W}^1 = (x^1)^T x^1 \tag{3.22}$$

Si un conjunto de m vectores fila $x^1, x^2, \cdots, x^m$ es autoasociado, la matriz de pesos $\mathbf{W}$ está dada por

$$\begin{aligned}
\mathbf{W} &= (x^1)^T x^1 + \cdots + (x^m)^T x^m \\
&= \begin{bmatrix} x_1^1 \\ \vdots \\ x_n^1 \end{bmatrix}_{n\times 1} [x_1^1, \cdots, x_n^1]_{1\times n} + \cdots + \begin{bmatrix} x_1^m \\ \vdots \\ x_n^m \end{bmatrix}_{n\times 1} [x_1^m, \cdots, x_n^m]_{1\times n} \\
&= \begin{bmatrix} x_1^1 x_1^1 & \cdots & x_1^1 x_n^1 \\ \vdots & \ddots & \vdots \\ x_n^1 x_1^1 & \cdots & x_n^1 x_n^1 \end{bmatrix}_{n\times n} + \cdots + \begin{bmatrix} x_1^m x_1^m & \cdots & x_1^m x_n^m \\ \vdots & \ddots & \vdots \\ x_n^m x_1^m & \cdots & x_n^m x_n^m \end{bmatrix}_{n\times n} \\
&= \begin{bmatrix} x_1^1 x_1^1 + \cdots + x_1^m x_1^m & \cdots & x_1^1 x_n^1 + \cdots + x_1^m x_n^m \\ \vdots & \ddots & \vdots \\ x_n^1 x_1^1 + \cdots + x_n^m x_1^m & \cdots & x_n^1 x_n^1 + \cdots + x_n^m x_n^m \end{bmatrix}_{n\times n}
\end{aligned} \tag{3.23}$$

Ahora bien, si se define a $\mathbf{X}$ como la matriz de $m \times n$ cuyas filas son los m vectores dados. Entonces $\mathbf{W}$ queda expresada en términos de la matriz $\mathbf{X}$ como se muestra en 3.25.

$$\mathbf{X} = \begin{bmatrix} x_1^1 & \cdots & x_n^1 \\ \vdots & \ddots & \vdots \\ x_1^m & \cdots & x_n^m \end{bmatrix}_{m\times n} \tag{3.24}$$

$$\begin{aligned}
\mathbf{W} &= \begin{bmatrix} x_1^1 & \cdots & x_1^m \\ \vdots & \ddots & \vdots \\ x_n^1 & \cdots & x_n^m \end{bmatrix}_{n\times m} \begin{bmatrix} x_1^1 & \cdots & x_n^1 \\ \vdots & \ddots & \vdots \\ x_1^m & \cdots & x_n^m \end{bmatrix}_{m\times n} \\
&= \begin{bmatrix} x_1^1 x_1^1 + \cdots + x_1^m x_1^m & \cdots & x_1^1 x_n^1 + \cdots + x_1^m x_n^m \\ \vdots & \ddots & \vdots \\ x_n^1 x_1^1 + \cdots + x_n^m x_1^m & \cdots & x_n^1 x_n^1 + \cdots + x_n^m x_n^m \end{bmatrix}_{n\times n} \\
&= \mathbf{X}^T \mathbf{X}
\end{aligned} \tag{3.25}$$

La matriz $\mathbf{W}$ es ahora la matriz de autocorrelación para el conjunto de m vectores dados. Es de esperarse que $\mathbf{W}$ pueda guiar a la reproducción de cada uno de los vectores $x^1, x^2, \cdots, x^m$ cuando es usada como matriz de pesos en una red autoasociativa. Esto significa que

$$sgn(x^i\mathbf{W}) = x^i \qquad \text{para } i = 1, \cdots, m \tag{3.26}$$

O alternativamente

$$sgn(\mathbf{XW}) = \mathbf{X} \tag{3.27}$$

Si la función $sgn(\mathbf{XW})$ se considera un operador no lineal, entonces la ecuación 3.26 expresa el hecho de que los vectores $x^1, x^2, \cdots, x^m$ son los eigenvectores del operador no lineal. El problema de aprendizaje para una red asociativa consiste en construir la matriz $\mathbf{W}$ para la cual el operador $sgn(\mathbf{XW})$ tiene a dichos eigenvectores como puntos fijos.

Ahora bien, si se toma $\mathbf{W} = \mathbf{X}^T\mathbf{X}$, entonces 3.27 se escribe como

$$sgn(\mathbf{X}\mathbf{X}^T\mathbf{X}) = \mathbf{X} \tag{3.28}$$

Al realizar la multiplicación de $\mathbf{X}\mathbf{X}^T$, se tiene

$$\begin{aligned}
\mathbf{X}\mathbf{X}^T &= \begin{bmatrix} x_1^1 & \cdots & x_n^1 \\ \vdots & \ddots & \vdots \\ x_1^m & \cdots & x_n^m \end{bmatrix}_{m\times n} \begin{bmatrix} x_1^1 & \cdots & x_1^m \\ \vdots & \ddots & \vdots \\ x_n^1 & \cdots & x_n^m \end{bmatrix}_{n\times m} \\
&= \begin{bmatrix} x_1^1x_1^1 + \cdots + x_n^1x_n^1 & \cdots & x_1^1x_1^m + \cdots + x_n^1x_n^m \\ \vdots & \ddots & \vdots \\ x_1^mx_1^1 + \cdots + x_n^mx_n^1 & \cdots & x_1^mx_1^m + \cdots + x_n^mx_n^m \end{bmatrix}_{m\times m} \\
&= \begin{bmatrix} \sum_{i=1}^{n} x_i^1x_i^1 & \cdots & \sum_{i=1}^{n} x_i^1x_i^m \\ \vdots & \ddots & \vdots \\ \sum_{i=1}^{n} x_i^mx_i^1 & \cdots & \sum_{i=1}^{n} x_i^mx_i^m \end{bmatrix}_{m\times m} = \begin{bmatrix} (x^1\cdot x^1) & \cdots & (x^1\cdot x^m) \\ \vdots & \ddots & \vdots \\ (x^m\cdot x^1) & \cdots & (x^m\cdot x^m) \end{bmatrix}_{m\times m}
\end{aligned} \tag{3.29}$$

Si los vectores $x^1, x^2, \cdots, x^m$ son bipolares y ortogonales, entonces, $\mathbf{X}\mathbf{X}^T$ es la matriz identidad (véase 3.30) multiplicada por n lo que satisface a 3.28. Si los patrones son cercanamente ortogonales entonces $\mathbf{X}\mathbf{X}^T$ se acerca a la matriz identidad y la red asociativa continua trabajando.

$$\mathbf{X}\mathbf{X}^T = \begin{bmatrix} \sum_{i=1}^{n}(\pm 1)^2 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sum_{i=1}^{n}(\pm 1)^2 \end{bmatrix}_{m\times m} = \begin{bmatrix} n & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & n \end{bmatrix}_{m\times m} = n\begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{bmatrix}_{m\times m} \quad 3.30$$

3.2 RNAs como sistemas dinámicos

El estudio de situaciones que dependen de algún parámetro (p. ej., el tiempo) y que varían de acuerdo a leyes establecidas, se considera un sistema dinámico. Conocer el estado de la situación en un momento dado, permite hacer predicciones sobre el futuro y reconstruir el pasado. De manera formal, un sistema dinámico es un modo de describir el recorrido a lo largo del tiempo de todos los puntos de un espacio dado S (espacio euclidiano o subconjunto abierto de un espacio euclidiano). Un sistema dinámico para S establece donde está $x_i \in S$, t unidades de tiempo más tarde, $x_i(t)$ [25].

Una red de neuronas artificiales puede ser representada por un sistema dinámico de tiempo continuo, a través de ecuaciones diferenciales *n*-dimensionales (véase la Tabla 3.1) que modelan la dinámica de las n neuronas. Cada neurona se define, en términos matemáticos, por un valor de estado x_i, que no es otra cosa más que un número real. Asimismo, para cada neurona $i = 1,2,\cdots,n$ se tiene asociada una *función de activación* o *función ganancia* $\varphi_i(x_i)$ y un *valor umbral* ε_i, que determinan, en conjunto, el estado de la unidad i. El nivel de actividad de n neuronas está establecido por un punto x en el espacio de estados de dimensión n, por lo que el espacio de estados queda definido por un conjunto de vectores $x = (x_1, x_2, \cdots, x_n)$ de números reales.

Sistema dinámico	Punto fijo
$\frac{dx_i}{dt} = x_i' = f_i(x,t)$	$\frac{dx_i}{dt} = f_i(x,t) = 0$

Tabla 3.1 Sistema dinámico de tiempo continuo

El principal objetivo de la red, es generar trayectorias que se aproximen asintóticamente a algún estado de equilibrio. En general, los modelos de redes neuronales reflejan precisamente a las trayectorias solución como el resultado de un proceso de decisión determinado por los datos de entrada.

Tomando la definición de Shilnikov [26], una trayectoria $x(t)$ del sistema $x_i' = f_i(x,t)$ es llamada un estado de equilibrio (también conocida como trayectoria

constante o punto fijo) si esta no depende del tiempo, es decir, $x_i(t) \equiv x_i(0) = constante$.

Es así que para localizar los puntos fijos es preciso resolver el sistema de ecuaciones diferenciales igualadas a cero. Es importante señalar que el objetivo de los sistemas dinámicos no es ubicar únicamente los puntos fijos, sino conocer cómo se comporta el sistema dentro y fuera de la región cercana a los mismos. Es por ello que una vez localizadas las trayectorias constantes es necesario un análisis de estabilidad que permita determinar su comportamiento.

La forma tradicional de realizar un análisis de estabilidad es a partir de lo que se conoce como *linealización*, misma que se consigue a través de la matriz Jacobiano, definida como una matriz cuadrada **J** de $n \times n$ donde cada elemento $\mathbf{j}_{ik}$ es la derivada parcial de $f_i(x)$ respecto a x_k evaluada en la trayectoria constante $\underline{x}$, que en su forma extendida es

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}_{\underline{x}} \qquad 3.31$$

El Jacobiano es la herramienta que brinda la pauta para determinar la estabilidad de las trayectorias constantes. El análisis se realiza evaluando el punto fijo en la matriz para posteriormente obtener los valores propios de **J** que permiten, de acuerdo con Scheinerman [27], concluir los resultados mostrados en la Tabla 3.2. Donde se aprecia que la estabilidad o inestabilidad de un punto fijo depende de la parte real de los valores propios.

Tipo de eigenvalores	Tipo de punto fijo
$Todos \quad Real(\lambda_i) < 0$	Estable
$Algún \quad Real(\lambda_i) > 0$	Inestable
$Todos \quad Real(\lambda_i) \leq 0, algún \quad Real(\lambda_i) = 0$	Desconocido

*Tabla 3.2 Clasificación de los puntos fijos según los eigenvalores de **J***

Sin embargo, una definición formal de estabilidad para una trayectoria constante está en términos de un ε y un δ que garantizan el acercamiento asintótico a $\underline{x}$. Léanse a continuación algunas definiciones de estabilidad propuestas por Haykin [2].

Definición 1. Se dice que el punto fijo $\underline{x}$ es uniformemente estable si para algún ε positivo existe un δ positivo tal que la condición

$$\|x(0) - \underline{x}\| < \delta$$

implique

$$\|x(t) - \underline{x}\| < \varepsilon$$

para todo $t > 0$.

Obsérvese que esta definición establece que una trayectoria permanece dentro de un vecindario pequeño del punto fijo $\underline{x}$ si el estado inicial $x(0)$ es cercano a $\underline{x}$.

Definición 2. El punto fijo $\underline{x}$ es convergente si existe un δ positivo tal que la condición

$$\|x(0) - \underline{x}\| < \delta$$

Implica que

$$x(t) \longrightarrow \underline{x} \quad \text{cuando} \quad t \longrightarrow \infty$$

Adviértase que si el estado inicial $x(0)$ es sumamente cercano al punto fijo $\underline{x}$, entonces la trayectoria descrita por el vector estado $x(t)$ se aproximará a $\underline{x}$ cuando t se aproxima al infinito.

Definición 3. Se dice que el punto fijo $\underline{x}$ es asintóticamente estable si $\underline{x}$ cumple las condiciones de estabilidad y convergencia.

Nótese que la estabilidad y la convergencia son propiedades independientes, es por ello que la estabilidad asintótica se presenta únicamente cuando existen ambas propiedades en el sistema.

Definición 4. El punto fijo $\underline{x}$ es asintóticamente estable o globalmente estable asintóticamente si $\underline{x}$ es estable y todas las trayectorias del sistema convergen a $\underline{x}$ cuando el tiempo t se aproxima al infinito.

Esta definición implica que el sistema no puede tener otras trayectorias constantes, lo que representa que el sistema finalmente converge a un estado constante aún cuando se presenten cambios en las condiciones iniciales.

Es importante señalar que el conjunto de todos los puntos que pueden servir como estados de inicio $x(0)$ para trayectorias que se acercan asintóticamente a una trayectoria estable es llamado región atractor o región de atracción de la trayectoria estable.

Para ilustrar lo anterior considérese el siguiente modelo general como un sistema dinámico que define una red neuronal, bajo el supuesto de que para cada $i = 1,2,\cdots,n$ existe una constante positiva k_i y una función $\rho_i(\varphi_i(x_i))$ que es continuamente diferenciable.

$$\frac{d(x_i)}{dt} = -k_i x_i + \rho_i\big(\varphi_i(x_i)\big) \tag{3.32}$$

Donde la función de activación $\varphi_i(x_i)$ es una función $\mathbb{R} \longrightarrow [0,1]$, de la forma:

$$\varphi_i(x_i) = \begin{cases} 1 & x_i \geq \varepsilon_i \\ \varphi_i(x_i) & -\varepsilon_i < x_i < \varepsilon_i \\ 0 & x_i \leq -\varepsilon_i \end{cases} \tag{3.33}$$

Entonces, el espacio de estados queda dividido en tres intervalos o regiones que satisfacen el sistema, una la *región de transición* τ_ε formada por puntos x con al menos un componente x_i que cumple $|x_i| < \varepsilon_i$ y las otras dos denominadas *ortantes relativos* que son el complemento de τ_ε, en los que $\varphi_i(x_i)$ es constante.

Siguiendo la idea de redes neuronales artificiales como modelos de sistemas dinámicos, se exponen a continuación algunas variantes del modelo 3.32 que ligan de manera más clara ambos enfoques y además, proporcionan una perspectiva del ámbito de aplicación que puede tener este campo de estudio.

3.2.1 Problemas de selección de conjuntos

Para conducir la investigación al campo de redes neuronales artificiales sujetas a los principios teóricos de los sistemas dinámicos, supóngase ahora un problema de selección de conjuntos en el que se tiene un conjunto $\boldsymbol{S}$ de n elementos con p subconjuntos de $\boldsymbol{S}$ denotados como $\boldsymbol{S}_k$ con $k = 1,2,\cdots,p$. El objetivo es encontrar un conjunto respuesta $\boldsymbol{A}$ tal que:

$$|\boldsymbol{A} \cap \boldsymbol{S}_k| = 1 \tag{3.34}$$

Es decir, la cardinalidad o número de elementos que tiene la intersección del conjunto respuesta $\boldsymbol{A}$ con cada uno de los subconjuntos $\boldsymbol{S}_k$ es uno. Para abordar el problema admítase la modificación al modelo general 3.32 propuesta por Jeffries [28] de la forma:

$$\frac{d(x_i)}{dt} = -x_i + 1 - \left(\frac{2}{n_k}\right) \sum_{\substack{S_k \\ i \in S_k}} \sum_{\substack{j \in S_k \\ j \neq i}} \varphi_j(x_j) \tag{3.35}$$

Donde n_k denota el número de subconjuntos que contienen al elemento i, asimismo la primera sumatoria es para todos aquellos subconjuntos $\boldsymbol{S}_k$ que contienen a i y la segunda para todos los elementos del subconjunto exceptuando al elemento i. Por otro lado, una trayectoria constante para 3.35 debe satisfacer

$$x_i = 1 - \left(\frac{2}{n_k}\right) \sum_{\substack{S_k \\ i \in S_k}} \sum_{\substack{j \in S_k \\ j \neq i}} \varphi_j(x_j) \tag{3.36}$$

es decir,

$$\frac{d(x_i)}{dt} = 0 \tag{3.37}$$

Ya que el sistema es de tiempo continuo. Además, considérese la función de activación $\varphi_i(x_i)$ como una función lineal con umbral definida como

$$\varphi_i(x_i) = \begin{cases} 1 & x_i \geq \varepsilon_i \\ \dfrac{x_i + \varepsilon_i}{2\varepsilon_i} & -\varepsilon_i < x_i < \varepsilon_i \\ 0 & x_i \leq -\varepsilon_i \end{cases} \tag{3.38}$$

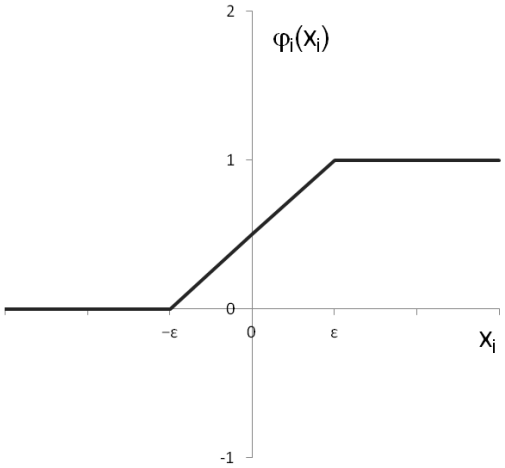


Fig. 3.2 Gráfica de la función $\varphi_i(x_i)$

Una vez definidos todos los elementos del modelo se procede a la enunciación del Teorema 1 que ha sido propuesto y demostrado por Jeffries [28] para caracterizar las trayectorias constantes estables de 3.35.

Teorema 1. Cada conjunto respuesta para el problema de selección de conjuntos 3.34 corresponde a una trayectoria constante estable x con cada $x_i = \pm 1$ para la red neuronal 3.35. También, cada trayectoria constante para la red neuronal con cada $x_i = \pm 1$ corresponde a un conjunto respuesta para el problema de selección de conjuntos.

Este teorema establece una correspondencia entre los conjuntos respuesta para el problema 3.34 y los puntos fijos estables de 3.35 con $x_i = \pm 1$. Sin embargo, 3.35 no resuelve a 3.34 en el sentido de que existen otros atractores constantes estables que no corresponden a conjuntos respuesta para 3.35. Dichos atractores se conocen como trayectorias ε*-invariantes* [28].

Para ejemplificar lo antes dicho, considérese el siguiente problema de selección de conjuntos con trayectorias constantes ε*-invariantes* propuesto por Jeffries [28]. Supóngase que se cuenta con el conjunto $\boldsymbol{S} = \{1,2,3,4,5,6\}$ que tiene los

subconjuntos $\boldsymbol{S}_1 = \{1,3\}$, $\boldsymbol{S}_2 = \{2,3\}$, $\boldsymbol{S}_3 = \{3,4\}$, $\boldsymbol{S}_4 = \{4,5\}$ y $\boldsymbol{S}_5 = \{4,6\}$. Tomando en cuenta a $\boldsymbol{S}$ y a sus respectivos subconjuntos $\boldsymbol{S}_k$, la versión asociada al problema del modelo 3.35, se resume en la Tabla 3.3.

i	n_i	$\boldsymbol{S}_k$ $i \in \boldsymbol{S}_k$	$\sum_{\substack{j \in S_k \\ j \neq i}} \varphi_j(x_j)$	$\sum_{i \in S_k} \sum_{\substack{j \in S_k \\ j \neq i}} \varphi_j(x_j)$
1	1	$\boldsymbol{S}_1 = \{1,3\}$	$\varphi_3(x_3)$	$\varphi_3(x_3)$
2	1	$\boldsymbol{S}_2 = \{2,3\}$	$\varphi_3(x_3)$	$\varphi_3(x_3)$
3	3	$\boldsymbol{S}_1 = \{1,3\}$ $\boldsymbol{S}_2 = \{2,3\}$ $\boldsymbol{S}_3 = \{3,4\}$	$\varphi_1(x_1)$ $\varphi_2(x_2)$ $\varphi_4(x_4)$	$\varphi_1(x_1) + \varphi_2(x_2) + \varphi_4(x_4)$
4	3	$\boldsymbol{S}_3 = \{3,4\}$ $\boldsymbol{S}_4 = \{4,5\}$ $\boldsymbol{S}_5 = \{4,6\}$	$\varphi_3(x_3)$ $\varphi_5(x_5)$ $\varphi_6(x_6)$	$\varphi_3(x_3) + \varphi_5(x_5) + \varphi_6(x_6)$
5	1	$\boldsymbol{S}_4 = \{4,5\}$	$\varphi_4(x_4)$	$\varphi_4(x_4)$
6	1	$\boldsymbol{S}_5 = \{4,6\}$	$\varphi_4(x_4)$	$\varphi_4(x_4)$

Tabla 3.3 Desarrollo del modelo de red neuronal 3.35 para el conjunto $\boldsymbol{S} = \{1,2,3,4,5,6\}$ y los subconjuntos $\boldsymbol{S}_1 = \{1,3\}$, $\boldsymbol{S}_2 = \{2,3\}$, $\boldsymbol{S}_3 = \{3,4\}$, $\boldsymbol{S}_4 = \{4,5\}$ y $\boldsymbol{S}_5 = \{4,6\}$.

Puede observarse que el modelo genera el sistema dinámico de dimensión seis 3.39. Con un tipo de arquitectura recurrente sin capas ocultas, con retroalimentación a otras neuronas y a sí misma, que se muestra en la Fig. 3.3.

$$\begin{aligned}
\frac{dx_1}{dt} &= 1 - x_1 - 2[\varphi_3(x_3)] \\
\frac{dx_2}{dt} &= 1 - x_2 - 2[\varphi_3(x_3)] \\
\frac{dx_3}{dt} &= 1 - x_3 - \frac{2}{3}[\varphi_1(x_1) + \varphi_2(x_2) + \varphi_4(x_4)] \\
\frac{dx_4}{dt} &= 1 - x_4 - \frac{2}{3}[\varphi_3(x_3) + \varphi_5(x_5) + \varphi_6(x_6)] \\
\frac{dx_5}{dt} &= 1 - x_5 - 2[\varphi_4(x_4)] \\
\frac{dx_6}{dt} &= 1 - x_6 - 2[\varphi_4(x_4)]
\end{aligned} \qquad 3.39$$

Para continuar con el estudio del problema es necesario determinar las trayectorias constantes o puntos fijos igualando a cero cada una de las ecuaciones como se muestra en 3.40. Nótese que $x_1 = x_2$ y $x_5 = x_6$ por lo tanto $\varphi_1 = \varphi_2$ y $\varphi_5 = \varphi_6$. Además si se toma a 3.38 como la función de activación con $\varepsilon = 1/3$ y se asumen algunos valores para φ_i sin distorsionar el comportamiento de φ, entonces es posible obtener las trayectorias constantes $\underline{x}$ que se muestran en la Tabla 3.4.

$$
\begin{aligned}
x_1 &= 1 - 2[\varphi_3(x_3)] \\
x_2 &= 1 - 2[\varphi_3(x_3)] \\
x_3 &= 1 - \frac{2}{3}[\varphi_1(x_1) + \varphi_2(x_2) + \varphi_4(x_4)] \\
x_4 &= 1 - \frac{2}{3}[\varphi_3(x_3) + \varphi_5(x_5) + \varphi_6(x_6)] \\
x_5 &= 1 - 2[\varphi_4(x_4)] \\
x_6 &= 1 - 2[\varphi_4(x_4)]
\end{aligned}
\qquad 3.40
$$

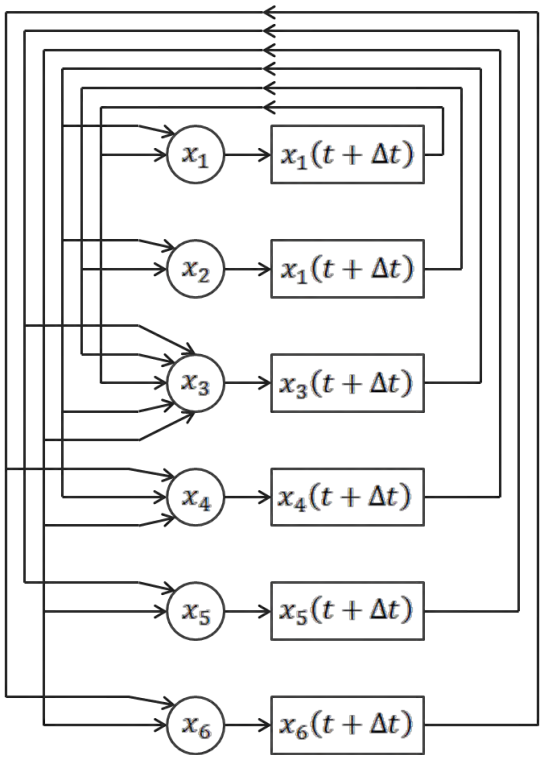


Fig. 3.3 Arquitectura recurrente sin capas ocultas para el modelo 3.39

Si $\varphi_i \Rightarrow x_j$	$\varphi^l = (\varphi_1, \varphi_2, \varphi_3, \varphi_4, \varphi_5, \varphi_6)$	$\underline{x}^r = (x_1, x_2, x_3, x_4, x_5, x_6)$
Si $\varphi_1 = \varphi_2 = 1 \Rightarrow x_3 = -1/3 - 2/3\,\varphi_4$ Si $\varphi_5 = \varphi_6 = 0 \Rightarrow x_4 = 1 - 2/3\,\varphi_3$ Si $\varphi_3 = 0 \Rightarrow x_4 = 1 \therefore \varphi_4 = 1$	$\varphi^1 = (1,1,0,1,0,0)$	$\underline{x}^1 = (1,1,-1,1,-1,-1)$
Si $\varphi_1 = \varphi_2 = 0 \Rightarrow x_3 = 1 - 2/3\,\varphi_4$ Si $\varphi_5 = \varphi_6 = 1 \Rightarrow x_4 = -1/3 - 2/3\,\varphi_3$ Si $\varphi_3 = 1 \Rightarrow x_4 = -1 \therefore \varphi_4 = 0$	$\varphi^2 = (0,0,1,0,1,1)$	$\underline{x}^2 = (-1,-1,1,-1,1,1)$
Si $\varphi_1 = \varphi_2 = 0 \Rightarrow x_3 = 1 - 2/3\,\varphi_4$ Si $\varphi_5 = \varphi_6 = 0 \Rightarrow x_4 = 1 - 2/3\,\varphi_3$ Si $\varphi_4 = 1 \Rightarrow x_3 = 1/3 \therefore \varphi_3 = 1$	$\varphi^3 = (0,0,1,1,0,0)$	$\underline{x}^3 = \left(-1,-1,\frac{1}{3},\frac{1}{3},-1,-1\right)$
Si $\varphi_1 = \varphi_2 = 1 \Rightarrow x_3 = -1/3 - 2/3\,\varphi_4$ Si $\varphi_5 = \varphi_6 = 1 \Rightarrow x_4 = -1/3 - 2/3\,\varphi_3$ Si $\varphi_4 = 0 \Rightarrow x_3 = -1/3 \therefore \varphi_3 = 0$	$\varphi^4 = (1,1,0,0,1,1)$	$\underline{x}^4 = \left(1,1,-\frac{1}{3},-\frac{1}{3},1,1\right)$

Tabla 3.4 Trayectorias constantes del modelo de red neuronal 3.39

Adviértase que los puntos fijos $\underline{x}^1 = (1,1,-1,1,-1,-1)$ y $\underline{x}^2 = (-1,-1,1,-1,1,1)$ sí son soluciones del problema de selección de conjuntos ya que sus componentes son enteros. Sin embargo, los puntos fijos $\underline{x}^3 = (-1,-1,1/3,1/3,-1,-1)$ y $\underline{x}^4 = (1,1,-1/3,-1/3,1,1)$ no son soluciones del problema porque tienen componentes fraccionarios. Lo que contradice el hecho de que los componentes de $\boldsymbol{S}$ son únicos e indivisibles. Por lo tanto, $\underline{x}^3$ y $\underline{x}^4$ se consideran atractores ε-*invariantes*.

De acuerdo a lo anterior y al Teorema 1, las trayectorias constantes estables para el problema de selección de conjuntos son $\underline{x}^1 = (1,1,-1,1,-1,-1)$ y $\underline{x}^2 = (-1,-1,1,-1,1,1)$. Donde $x_i = +1$ significa que el elemento i está presente en el conjunto y $x_i = -1$ el hecho contrario. Por consiguiente se tiene dos conjuntos respuesta $\boldsymbol{A}$,

$$\begin{aligned} \boldsymbol{A}_1 &= \{1,2,4\} \\ \boldsymbol{A}_2 &= \{3,4,5\} \end{aligned} \tag{3.41}$$

que satisfacen $|\boldsymbol{A}_j \cap \boldsymbol{S}_k| = 1$ para todo $k = 1,\cdots,6$. Y por el Teorema 1 se garantiza que $\boldsymbol{A}_1$ y $\boldsymbol{A}_2$ son trayectorias constantes estables. Sin embargo, es válido conocer el comportamiento de las otras soluciones a partir de la linealización del sistema. Por lo que la matriz Jacobiano queda definida para 3.39 como

$$\boldsymbol{J} = \begin{bmatrix} -1 & 0 & -2\varphi_3' & 0 & 0 & 0 \\ 0 & -1 & -2\varphi_3' & 0 & 0 & 0 \\ -\frac{2}{3}\varphi_1' & -\frac{2}{3}\varphi_2' & -1 & -\frac{2}{3}\varphi_4' & 0 & 0 \\ 0 & 0 & -\frac{2}{3}\varphi_3' & -1 & -\frac{2}{3}\varphi_5' & -\frac{2}{3}\varphi_6' \\ 0 & 0 & 0 & -\frac{2}{3}\varphi_4' & -1 & 0 \\ 0 & 0 & 0 & -\frac{2}{3}\varphi_4' & 0 & -1 \end{bmatrix}_{\underline{x}} \tag{3.42}$$

Si se evalúa $\boldsymbol{J}$ para $\underline{x}^3$ y $\underline{x}^4$ se obtienen los resultados de la Tabla 3.5.

$\underline{x}^l$	$\underline{x}_i^l$	φ_i^l	φ_i'
$\underline{x}^3$	-1	0	0
	-1	0	0
	$1/3$	1	0
	$1/3$	1	0
	-1	0	0
	-1	0	0

$\underline{x}^l$	$\underline{x}_i^l$	φ_i^l	φ_i'
$\underline{x}^4$	1	1	0
	1	1	0
	$-1/3$	0	0
	$-1/3$	0	0
	1	1	0
	1	1	0

Tabla 3.5 Evaluación de la matriz Jacobiano 3.42 para los atractores ε-invariantes de 3.39

Se observa que en ambos casos $\boldsymbol{J}$ es una matriz cuadrada con -1 en la diagonal principal lo que significa que $\lambda_i^3 = \lambda_i^4 = -1$ para $i = 1, \cdots, 6$. Y recordando la Tabla 3.2 se concluye que $\underline{x}^3$ y $\underline{x}^4$ son trayectorias constantes estables.

Ahora considérese un nuevo problema en el que cada elemento i del conjunto $\boldsymbol{S}$ está contenido en exactamente dos distintos subconjuntos $\boldsymbol{R}_j$ y $\boldsymbol{C}_k$. Que pueden contener al elemento i únicamente. Por ejemplo, si se define a $\boldsymbol{R}_j$ como las filas de un tablero cuadriculado y a $\boldsymbol{C}_k$ como las columnas, entonces el conjunto respuesta $\boldsymbol{A}$ es aquel que satisface

$$\begin{matrix} |\boldsymbol{A} \cap \boldsymbol{R}_j| \leq 1 & \forall \boldsymbol{R}_j \\ |\boldsymbol{A} \cap \boldsymbol{C}_k| \leq 1 & \forall \boldsymbol{C}_k \end{matrix} \tag{3.43}$$

Y además,

$$\bigcup_{|\boldsymbol{R}_j \cap \boldsymbol{A}|} \boldsymbol{R}_j \bigcup_{|\boldsymbol{C}_k \cap \boldsymbol{A}|} \boldsymbol{C}_k = \boldsymbol{S} \tag{3.44}$$

Este problema puede considerarse como una versión de cobertura de combinatorias cuyo modelo de red neuronal ha sido propuesto por Jeffries [28] como:

$$\frac{dx_i}{dt} = 1 - x_i - 2\left[\sum_{\substack{j \in \boldsymbol{R}_a \\ i \in \boldsymbol{R}_a}} \sum_{j \neq i} \varphi_j(x_j) + \sum_{\substack{j \in \boldsymbol{C}_b \\ i \in \boldsymbol{C}_b}} \sum_{j \neq i} \varphi_j(x_j) - \psi_i\left(\sum_{\substack{j \in \boldsymbol{R}_a \\ i \in \boldsymbol{R}_a}} \sum_{j \neq i} \varphi_j(x_j) \sum_{\substack{j \in \boldsymbol{C}_b \\ i \in \boldsymbol{C}_b}} \sum_{j \neq i} \varphi_j(x_j)\right)\right] \tag{3.45}$$

Con la funciones de activación 3.46. Cuyas gráficas se observan en la Fig. 3.4.

$$\varphi_i(x_i) = \begin{cases} 1 & x_i \geq \varepsilon_i \\ \dfrac{x_i + \varepsilon_i}{2\varepsilon_i} & -\varepsilon_i < x_i < \varepsilon_i \\ 0 & x_i \leq -\varepsilon_i \end{cases} \quad \text{y,} \quad \psi_i(x_i) = \begin{cases} 1 & x_i \geq \varepsilon_i \\ \dfrac{x_i}{\varepsilon_i} & 0 < x_i < \varepsilon_i \\ 0 & x_i \leq 0 \end{cases} \tag{3.46}$$

Asimismo se utiliza el método de Euler [29] para determinar los cambios de estado en el sistema con el paso del tiempo. Que corresponde a la versión actualizada de la fórmula 3.47. Donde Δt denota el incremento en el índice de tiempo.

$$x_i(t + \Delta t) = x_i(t) + \left(\frac{dx_i}{dt}\right)\Delta t \tag{3.47}$$

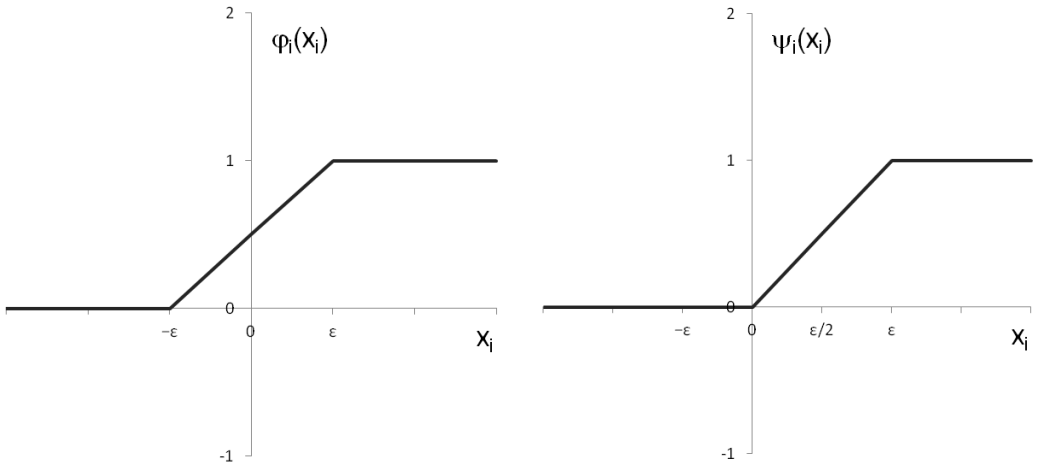


Fig. 3.4 Gráficas de las funciones $\varphi_i(x_i)$ y $\psi_i(x_i)$

Utilizar el método numérico de Euler en un problema del tipo 3.45 puede ayudar a visualizar, al menos de forma numérica, el comportamiento de este tipo de modelos. Para ello considérese el siguiente ejemplo teórico propuesto por Jeffries [28]. En una tabla cuadriculada con celdas blancas y negras se colocan sensores únicamente en los cuadros blancos. Estos sensores se ubican de acuerdo con los siguientes principios: si dos cuadrados blancos son adyacentes o si entre dos cuadrados blancos que se encuentran en la misma fila o columna sólo hay otros cuadrados blancos, entonces se dice que cada cuadrado blanco es visible al otro. Cuando un sensor es puesto en un cuadrado blanco se dice que este cuadrado está cubierto y además que cubre todos los cuadrados visibles al mismo. El problema consiste en colocar sensores de tal forma que todos lo cuadrados blancos sean cubiertos y además, sólo haya un sensor por fila y columna.

La formulación anterior permite deducir que se trata de un problema del tipo 3.43, cuyo modelo de red neuronal es 3.45. Suponiendo que la cuadrícula para la colocación de sensores toma la forma de la Fig. 3.5, entonces el sistema de ecuaciones asociado es 3.48.

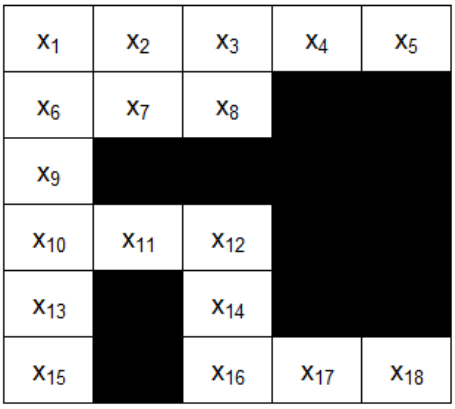


Fig. 3.5 Asociación de cuadros blancos con su respectiva neurona x_i [28]

$$
\begin{aligned}
\frac{dx_1}{dt} &= 1 - x_1 - 2[\varphi_2 + \varphi_3 + \varphi_4 + \varphi_5 + \varphi_6 + \varphi_9 + \varphi_{10} + \varphi_{13} + \varphi_{15} - \psi_1(\varphi)] \\
\frac{dx_2}{dt} &= 1 - x_2 - 2[\varphi_1 + \varphi_3 + \varphi_4 + \varphi_5 + \varphi_7 + \varphi_{11} - \psi_2(\varphi)] \\
\frac{dx_3}{dt} &= 1 - x_3 - 2[\varphi_1 + \varphi_2 + \varphi_4 + \varphi_5 + \varphi_8 + \varphi_{12} + \varphi_{14} + \varphi_{16} - \psi_3(\varphi)] \\
\frac{dx_4}{dt} &= 1 - x_4 - 2[\varphi_1 + \varphi_2 + \varphi_3 + \varphi_5 + \varphi_{17} - \psi_4(\varphi)] \\
\frac{dx_5}{dt} &= 1 - x_5 - 2[\varphi_1 + \varphi_2 + \varphi_3 + \varphi_4 + \varphi_{18} - \psi_5(\varphi)] \\
\frac{dx_6}{dt} &= 1 - x_6 - 2[\varphi_7 + \varphi_8 + \varphi_1 + \varphi_9 + \varphi_{10} + \varphi_{13} + \varphi_{15} - \psi_6(\varphi)] \\
\frac{dx_7}{dt} &= 1 - x_7 - 2[\varphi_6 + \varphi_8 + \varphi_2 + \varphi_{11} - \psi_7(\varphi)] \\
\frac{dx_8}{dt} &= 1 - x_8 - 2[\varphi_6 + \varphi_7 + \varphi_3 + \varphi_{12} + \varphi_{14} + \varphi_{16} - \psi_8(\varphi)] \\
\frac{dx_9}{dt} &= 1 - x_9 + 2\psi_9(\varphi) \\
\frac{dx_{10}}{dt} &= 1 - x_{10} - 2[\varphi_{11} + \varphi_{12} + \varphi_1 + \varphi_6 + \varphi_9 + \varphi_{13} + \varphi_{15} - \psi_{10}(\varphi)] \\
\frac{dx_{11}}{dt} &= 1 - x_{11} - 2[\varphi_{10} + \varphi_{12} + \varphi_2 + \varphi_7 - \psi_{11}(\varphi)] \\
\frac{dx_{12}}{dt} &= 1 - x_{12} - 2[\varphi_{10} + \varphi_{11} + \varphi_3 + \varphi_8 + \varphi_{14} + \varphi_{16} - \psi_{12}(\varphi)] \\
\frac{dx_{13}}{dt} &= 1 - x_{13} - 2[\varphi_{14} + \varphi_1 + \varphi_6 + \varphi_9 + \varphi_{10} + \varphi_{15} - \psi_{13}(\varphi)] \\
\frac{dx_{14}}{dt} &= 1 - x_{14} - 2[\varphi_{13} + \varphi_3 + \varphi_8 + \varphi_{12} + \varphi_{16} - \psi_{14}(\varphi)] \\
\frac{dx_{15}}{dt} &= 1 - x_{15} - 2[\varphi_{16} + \varphi_{17} + \varphi_{18} + \varphi_1 + \varphi_6 + \varphi_9 + \varphi_{10} + \varphi_{13} - \psi_{15}(\varphi)] \\
\frac{dx_{16}}{dt} &= 1 - x_{16} - 2[\varphi_{15} + \varphi_{17} + \varphi_{18} + \varphi_3 + \varphi_8 + \varphi_{12} + \varphi_{14} - \psi_{16}(\varphi)] \\
\frac{dx_{17}}{dt} &= 1 - x_{17} - 2[\varphi_{15} + \varphi_{16} + \varphi_{18} + \varphi_4 - \psi_{17}(\varphi)] \\
\frac{dx_{18}}{dt} &= 1 - x_{18} - 2[\varphi_{15} + \varphi_{16} + \varphi_{17} + \varphi_5 - \psi_{18}(\varphi)]
\end{aligned}
\qquad 3.48
$$

Al utilizar el método de Euler para encontrar las soluciones, es indispensable llevar a cabo una serie de iteraciones que conduzcan a las trayectorias constantes del sistema. Una forma sencilla de abordar el problema es trasladar el modelo a una hoja de cálculo que permita el cálculo iterativo o bien llevar el algoritmo a un lenguaje de programación. Por cuestiones de sencillez se ha optado por la primera opción, mostrándose a continuación los resultados obtenidos al ejecutar la simulación en Microsoft Office Excel 2007®.

Problema de selección de conjuntos de la forma |A intersección R_j|≤ 1 y |A intersección C_k|≤ 1 para cada R_j y C_k subconjuntos de S

	Δt	0.2000000		ε	0.1000000
Valor inicial de x_i	0.7903106	-0.5065745	-0.7181546	-0.5075867	0.1053907
	-0.5035963	-0.2844103	-0.4341036		
	-0.3040257				
x_i	0.3721364	0.0968394	0.7793675		
	-0.7604838		-0.4139585		
	-0.0892118		-0.4610841	-0.1771026	-0.9970956
Activación $\varphi_i(x_i)$	1.0000000	0.0000000	0.0000000	0.0000000	1.0000000
	0.0000000	0.0000000	0.0000000		
$\varphi_i(x_i) = \begin{cases} 1 & x_i \geq \varepsilon_i \\ \frac{x_i + \varepsilon_i}{2\varepsilon_i} & -\varepsilon_i < x_i < \varepsilon_i \\ 0 & x_i \leq -\varepsilon_i \end{cases}$	0.0000000				
	1.0000000	0.9841969	1.0000000		
	0.0000000		0.0000000		
	0.0539412		0.0000000	0.0000000	0.0000000
Suma por fila	2.0000000	2.0000000	2.0000000	2.0000000	2.0000000
	0.0000000	0.0000000	0.0000000		
$\sum_{\substack{j \in R_a \\ i \in R_a}} \varphi_j$	0.0000000				
	2.9841969	2.9841969	2.9841969		
	0.0000000		0.0000000		
	0.0539412		0.0539412	0.0539412	0.0539412
Suma por columna	2.0539412	0.9841969	1.0000000	0.0000000	1.0000000
	2.0539412	0.9841969	1.0000000		
$\sum_{\substack{j \in C_b \\ i \in C_b}} \varphi_j$	2.0539412				
	2.0539412	0.9841969	1.0000000		
	2.0539412		1.0000000		
	2.0539412		1.0000000	0.0000000	1.0000000
Diferencia por fila	1.0000000	2.0000000	2.0000000	2.0000000	1.0000000
	0.0000000	0.0000000	0.0000000		
$\sum_{\substack{j \in R_a \\ j \neq i \\ i \in R_a}} \varphi_j$	0.0000000				
	1.9841969	2.0000000	1.9841969		
	0.0000000		0.0000000		
	0.0000000		0.0539412	0.0539412	0.0539412
Diferencia por columna	1.0539412	0.9841969	1.0000000	0.0000000	0.0000000
	2.0539412	0.9841969	1.0000000		
$\sum_{\substack{j \in C_b \\ j \neq i \\ i \in C_b}} \varphi_j$	2.0539412				
	1.0539412	0.0000000	0.0000000		
	2.0539412		1.0000000		
	2.0000000		1.0000000	0.0000000	1.0000000
ψ (Diferencia por fila * Diferencia por columna)	1.0000000	1.0000000	1.0000000	0.0000000	0.0000000
	0.0000000	0.0000000	0.0000000		
$\psi_i(x_i) = \begin{cases} 1 & x_i \geq \varepsilon_i \\ \frac{x_i}{\varepsilon_i} & 0 < x_i < \varepsilon_i \\ 0 & x_i \leq 0 \end{cases}$	0.0000000				
	1.0000000	0.0000000	0.0000000		
	0.0000000		0.0000000		
	0.0000000		0.5394117	0.0000000	0.5394117
Siguiente x_i	0.4106720	-0.9989383	-1.1745237	-1.0060693	-0.1156874
	-1.0244535	-0.4212070	-0.5472829		
$x_i(t + \Delta t)$	-0.8647970				
	-0.3175461	-0.5225285	0.0298152		
	-1.2299635		-0.5311668		
	-0.6713694		-0.3746791	0.0367415	-0.8034883

Paso del tiempo (Método de Euler)

$$x_i(t + \Delta t) = x_i(t) + \left[1 - x_i(t) - 2 \left[\left(\sum_{\substack{j \in R_a \\ j \neq i \\ i \in R_a}} \varphi_j \right) + \left(\sum_{\substack{j \in C_b \\ j \neq i \\ i \in C_b}} \varphi_j \right) - \psi_i \left(\sum_{\substack{j \in R_a \\ j \neq i \\ i \in R_a}} \varphi_j \sum_{\substack{j \in C_b \\ j \neq i \\ i \in C_b}} \varphi_j \right) \right] \right] \Delta t$$

Fig. 3.6 Simulador del modelo de red neuronal 3.48

Cabe señalar que el estado de inicio para x_i es aleatorio y que además, después de aproximadamente 80 iteraciones sobre la hoja de cálculo, la trayectoria constante obtenida es

Trayectoria constante				
1.0000000	-1.0000000	-1.0000000	-1.0000000	-1.0000000
-1.0000000	**1.0000000**	-1.0000000		
-1.0000000				
-1.0000000	-1.0000000	**1.0000000**		
-1.0000000		-1.0000000		
-1.0000000		-1.0000000	**1.0000000**	-1.0000000

Fig. 3.7 Trayectoria constante para la red neuronal 3.48

Donde $+1$ denota la colocación de un sensor en el cuadrado blanco y -1 el caso contrario. Obsérvese que dicha trayectoria satisface los requerimientos del problema. Debe hacerse notar que el resultado anterior sólo representa una de las varias trayectorias solución del problema. Al realizar una serie de simulaciones con estados de inicio aleatorios para x_i se obtienen otras soluciones como las que se muestran a continuación.

Estado de inicio				
0.7903106	-0.5065745	-0.7181546	-0.5075867	0.1053907
-0.5035963	-0.2844103	-0.4341036		
-0.3040257				
0.3721364	0.0968394	0.7793675		
-0.7604838		-0.4139585		
-0.0892118		-0.4610841	-0.1771026	-0.9970956
-0.9477521	0.9676422	0.4667667	0.3035311	-0.6027059
0.3329542	0.8364206	-0.4107702		
0.8051482				
-0.1478802	-0.3122338	-0.3250630		
-0.7595933		-0.2035505		
0.5457727		0.6915230	0.3649900	0.8992224
0.9561418	-0.4892228	0.8675874	0.9414503	0.7610449
0.0759849	-0.1780056	0.9255696		
0.5729725				
-0.2627050	-0.7899273	0.4759600		
-0.7489094		-0.6014293		
0.5868147		-0.8165193	0.0947739	0.8462271
0.7126025	-0.9574149	0.7407053	0.5591181	-0.9140800
0.5514433	-0.0826734	-0.5477956		
0.6924038				
-0.2074209	0.2483060	-0.6100754		
-0.0041228		-0.4033591		
-0.3790035		-0.8068653	0.3141952	0.9853577
0.8101145	-0.6672497	-0.9914519	0.8208651	0.2233605
0.5183481	-0.0628728	0.9661629		
0.6790070				
-0.8778640	0.4383103	0.0205076		
-0.4063815		0.0219929		
0.1650253		-0.4243739	-0.3516888	0.6131908
0.2957030	-0.9487728	0.1448192	0.9052954	-0.8112968
0.0555507	0.0026860	-0.8064567		
-0.0055090				
-0.0587417	-0.0014612	-0.4672660		
-0.5208093		0.3051919		
-0.4214226		-0.0294737	-0.4497321	-0.9000005
0.7416414	0.8044387	0.4332671	-0.1636700	0.1212502
0.8100889	-0.9308951	-0.1957768		
-0.0354071				
0.7536099	-0.9466601	0.4846403		
0.1465309		0.6678970		
-0.4681318		0.1512221	-0.2478536	0.2990898

Trayectoria constante				
1.0000000	-1.0000000	-1.0000000	-1.0000000	-1.0000000
-1.0000000	**1.0000000**	-1.0000000		
-1.0000000				
-1.0000000	-1.0000000	**1.0000000**		
-1.0000000		-1.0000000		
-1.0000000		-1.0000000	**1.0000000**	-1.0000000
-1.0000000	-1.0000000	-1.0000000	**1.0000000**	-1.0000000
-1.0000000	**1.0000000**	-1.0000000		
1.0000000				
-1.0000000	-1.0000000	-1.0000000		
-1.0000000		**1.0000000**		
-1.0000000		-1.0000000	-1.0000000	**1.0000000**
-1.0000000	-1.0000000	-1.0000000	**1.0000000**	-1.0000000
-1.0000000	-1.0000000	**1.0000000**		
1.0000000				
-1.0000000	**1.0000000**	-1.0000000		
-1.0000000		-1.0000000		
-1.0000000		-1.0000000	-1.0000000	**1.0000000**
-1.0000000	-1.0000000	**1.0000000**	-1.0000000	-1.0000000
1.0000000	-1.0000000	-1.0000000		
-1.0000000				
-1.0000000	**1.0000000**	-1.0000000		
-1.0000000		-1.0000000		
-1.0000000		-1.0000000	-1.0000000	**1.0000000**
-1.0000000	-1.0000000	-1.0000000	**1.0000000**	-1.0000000
-1.0000000	-1.0000000	**1.0000000**		
1.0000000				
-1.0000000	**1.0000000**	-1.0000000		
-1.0000000		**1.0000000**		
-1.0000000		-1.0000000	-1.0000000	**1.0000000**
-1.0000000	-1.0000000	-1.0000000	**1.0000000**	-1.0000000
-1.0000000	-1.0000000	-1.0000000		
1.0000000				
-1.0000000	**1.0000000**	-1.0000000		
-1.0000000		-1.0000000		
-1.0000000		-1.0000000	-1.0000000	**1.0000000**
-1.0000000	**1.0000000**	-1.0000000	-1.0000000	-1.0000000
-1.0000000	-1.0000000	-1.0000000		
-1.0000000				
1.0000000	-1.0000000	-1.0000000		
-1.0000000		**1.0000000**		
-1.0000000		-1.0000000	-1.0000000	**1.0000000**

Fig. 3.8 Resultados de simulaciones para modelo de red neuronal 3.48

Puede observarse que sin importar el estado de inicio del sistema, la red neuronal conduce a trayectorias constantes que satisfacen de igual forma los requerimientos definidos en el problema. Para garantizar que cada trayectoria constante satisface a 3.43 para cualquier variante de 3.45, se transcribe el Teorema 2, cuya demostración se puede consultar en Jeffries [28].

Teorema 2. Cada conjunto respuesta para el problema de selección 3.43 corresponde a una trayectoria constante estable x con cada $x_i = \pm 1$ para la red neuronal 3.45. También, cada trayectoria constante para la red neuronal con cada $|x_i| > \varepsilon$ corresponde a un conjunto respuesta para el problema de selección de conjuntos con $x_i = \pm 1$.

Para una mejor comprensión de los problemas de selección de conjuntos, abordados desde la perspectiva de redes neuronales artificiales. Se presenta un ejemplo más cuyo patrón de cuadrícula toma la forma

Fig. 3.9 Rejilla para la colocación de sensores [28]

Y cuyo modelo de red neuronal y hoja de cálculo asociados son el sistema de ecuaciones 3.49 y la Fig. 3.10 respectivamente.

$$\begin{aligned}
\frac{dx_1}{dt} &= 1 - x_1 - 2[\varphi_2 + \varphi_3 + \varphi_4 - \psi_1(\varphi_2\varphi_4 + \varphi_3\varphi_4)] \\
\frac{dx_2}{dt} &= 1 - x_2 - 2[\varphi_1 + \varphi_3 - \psi_2(0)] \\
\frac{dx_3}{dt} &= 1 - x_3 - 2[\varphi_1 + \varphi_2 + \varphi_5 - \psi_3(\varphi_1\varphi_5 + \varphi_2\varphi_5)] \\
\frac{dx_4}{dt} &= 1 - x_4 - 2[\varphi_5 + \varphi_1 - \psi_4(\varphi_5\varphi_1)] \\
\frac{dx_5}{dt} &= 1 - x_5 - 2[\varphi_4 + \varphi_3 - \psi_5(\varphi_4\varphi_3)]
\end{aligned} \qquad 3.49$$

Selección de conjuntos de la forma |A intersección R_j|≤ 1 y |A intersección C_k|≤ para cada R_j y C_k subconjuntos de S			
Δt	0.2000000		
ε	0.1000000		
Entrada x_i	0.9678975	0.6315411	-0.7976464
	0.6560472		0.0987808
Activación φ_j	1.0000000	1.0000000	0.0000000
	1.0000000		0.9939039
$\sum_{\substack{j\in R_a \\ i\in R_a}} \varphi_j$	2.0000000	2.0000000	2.0000000
	1.9939039		1.9939039
$\sum_{\substack{j\in C_b \\ i\in C_b}} \varphi_j$	2.0000000	1.0000000	0.9939039
	2.0000000		0.9939039
$\sum_{\substack{j\in R_a \\ i\in R_a}} \sum_{j\neq i} \varphi_j$	1.0000000	1.0000000	2.0000000
	0.9939039		1.0000000
$\sum_{\substack{j\in C_b \\ i\in C_b}} \sum_{j\neq i} \varphi_j$	1.0000000	0.0000000	0.9939039
	1.0000000		0.0000000
$\psi_i\left(\sum_{\substack{j\in R_a \\ i\in R_a}} \sum_{j\neq i} \varphi_j \sum_{\substack{j\in C_b \\ i\in C_b}} \sum_{j\neq i} \varphi_j\right)$	1.0000000	0.0000000	1.0000000
	1.0000000		0.0000000
Cambio en x_i (método de Euler)	0.5743180	0.3052329	-1.2356787
	0.3272762		-0.1209754

$$\frac{dx_i}{dt} = 1 - x_i - 2\left[\sum_{\substack{j\in \boldsymbol{R}_a \\ i\in \boldsymbol{R}_a}} \sum_{j\neq i} \varphi_j(x_j) + \sum_{\substack{j\in \boldsymbol{C}_b \\ i\in \boldsymbol{C}_b}} \sum_{j\neq i} \varphi_j(x_j) - \psi_i\left(\sum_{\substack{j\in \boldsymbol{R}_a \\ i\in \boldsymbol{R}_a}} \sum_{j\neq i} \varphi_j(x_j) \sum_{\substack{j\in \boldsymbol{C}_b \\ i\in \boldsymbol{C}_b}} \sum_{j\neq i} \varphi_j(x_j)\right)\right]$$

Fig. 3.10 Simulador del modelo de red neuronal 3.49

Al realizar una serie de iteraciones sobre la hoja de cálculo variando los estados de inicio x_i de manera aleatoria, es posible encontrar trayectorias constantes como las que se muestran en la Fig. 3.11. Donde al igual que en el ejemplo anterior, el estado $+1$ indica la colocación de un sensor sobre el cuadrado blanco. Cabe señalar que por el Teorema 2 los conjuntos solución encontrados son trayectorias constantes estables ya a que sus componentes son ± 1.

Estado de inicio			Trayectoria constante		
-0.3087696	0.2298213	0.4442814	-1.0000000	-1.0000000	**1.0000000**
0.6344218		-0.1627335	**1.0000000**		-1.0000000
-0.0009816	0.0615802	-0.9999261	-1.0000000	**1.0000000**	-1.0000000
-0.9714327		0.2149259	-1.0000000		**1.0000000**
0.3231022	-0.4728597	-0.8616635	-1.0000000	**1.0000000**	-1.0000000
0.8862500		0.7160416	**1.0000000**		-1.0000000
0.5071144	0.3548671	-0.5937737	**1.0000000**	-1.0000000	-1.0000000
-0.5967896		-0.1599896	-1.0000000		**1.0000000**

Fig. 3.11 Resultados de simulaciones para el modelo de red neuronal 3.49

3.3 Memoria asociativa con sistemas dinámicos

En una red de neuronas artificiales que funciona como memoria de contenido direccional o de asociación, la memoria (el patrón previamente aprendido) no está localizada en un número particular de neuronas, sino que existe en asociación con otras, por lo que sólo se puede acceder a ella a través del contenido. Para modelar estas memorias se utilizan vectores binarios como representación del patrón a memorizar. Estos patrones de memoria se consideran los atractores de un sistema dinámico, tal que cada patrón de entrada supone un estado inicial que yace en un dominio de atracción de los múltiples atractores.

Con el fin de evitar confusiones, resulta conveniente, aclarar el término memoria o patrón de memoria, que hace referencia al vector que representa la imagen de alguno de los patrones reconocidos por la red. Cuando se hace referencia a un sistema dinámico, una memoria representa un estado estable del sistema.

3.3.1 Hipergrafos y redes neuronales artificiales

El uso de redes neuronales en aplicaciones de memoria asociativa o procesos de control exige garantizar que los atractores del modelo son exclusivamente aquellas trayectorias constantes construidas específicamente para un sistema.

Para abordar este dilema se han de utilizar objetos gráficos compuestos de vértices, baricentros (vértices adicionales) y arcos, a los que se definen como hipergrafos. El estudio de estos grafos ha de permitir justificar teóricamente como una trayectoria no constante se aproxima a una trayectoria constante.

Partiendo de una versión del modelo general:

$$\frac{dx_i}{dt} = -k_i x_i + \rho_i[\varphi(x)] \qquad 3.50$$

se define,

$$\frac{dx_i}{dt} = k_i(I_i - x_i) + \sum_j a_{ij_1j_2\cdots j_p}\varphi_{j_1}\varphi_{j_2}\cdots\varphi_{j_p} \qquad 3.51$$

En 3.51, φ_j es la abreviación de $\varphi_j(x_j)$, y la suma se realiza sobre ciertas permutaciones j de subconjuntos cuyos índices pertenecen al conjunto $\{1,2,\cdots,n\}$. Cada I_i es una constante no negativa.Por otro lado, la expansión de productos de φ funciones y la constante k_iI_i se denominan la *forma multiproducto de un modelo de orden superior* [28].

Como puede observarse en 3.51, $\rho_i[\varphi(x)]$ se separa en un componente constante y una suma de productos de funciones φ. Los índices para cada permutación $\{i,j_1,j_2,\cdots,j_p\}$ se asumen distintos tal que $j_1 < j_2 < \cdots < j_p$ si $p > 1$. En cada sumando, $a\_$ (la abreviación del coeficiente completo de a en 3.51) es un coeficiente constante que tiene asociado un *conjunto índice de coeficientes* de la forma $\{i,j_1,j_2,\cdots,j_p\}$. Es factible pensar en el *conjunto índice de coeficientes* como la constante que determina el signo $+$ o $-$ de $a\_$. Dado un conjunto índice de coeficientes, el conjunto ordenado $\{i,j_1,j_2,\cdots,j_p\}$ se denomina el *coeficiente simplex* asociado al conjunto.

Asimismo un sistema de la forma 3.51 con coeficientes $\{a\_\}$ equivalentes y del mismo signo se denomina una *clase signo equivalente*, donde N denota el número de coeficientes simplex en una clase signo equivalente. Para un mejor entendimiento de lo anterior, considérese el sistema 3.52 con cinco neuronas ($n = 5$) y cuatro coeficientes simplex ($N = 4$).

$$\begin{aligned}
\frac{dx_1}{dt} &= 1 - x_1 + \mathbf{a_{123}}\varphi_2\varphi_3 + \mathbf{a_{134}}\varphi_3\varphi_4 + \mathbf{a_{14}}\varphi_4 \\
\frac{dx_2}{dt} &= 1 - x_2 + a_{213}\varphi_1\varphi_3 \\
\frac{dx_3}{dt} &= 1 - x_3 + a_{312}\varphi_1\varphi_2 + a_{314}\varphi_3\varphi_4 \\
\frac{dx_4}{dt} &= 1 - x_4 + a_{413}\varphi_1\varphi_3 + a_{41}\varphi_1 + \mathbf{a_{45}}\varphi_5 \\
\frac{dx_5}{dt} &= 1 - x_5 + a_{54}\varphi_4
\end{aligned} \qquad 3.52$$

Los coeficientes simplex de 3.52 son $\{1,2,3\}$, $\{1,3,4\}$, $\{1,4\}$ y $\{4,5\}$, cuyas permutaciones asociadas a cada coeficiente son $\{213,312\},\{314,413\},\{41\}$ y $\{54\}$ respectivamente.

Asociado a cada clase signo equivalente de la forma 3.51 existe un *hipergrafo* $\mathbf{H}$ con los siguientes componentes:

- **Vértices**. Son puntos etiquetados con $\{v_1, v_2, \cdots, v_n\}$ que corresponden a cada neurona.
- **Baricentros**. Son vértices adicionales etiquetados por $\{b_1, b_2, \cdots, b_N\}$ que corresponden a un coeficiente simplex $\{i, j_1, j_2, \cdots, j_p\}$.
- **Arcos**. Son los segmentos de línea que unen a los baricentros con los vértices. Cada arista es etiquetada con su correspondiente coeficiente $\{a\_\}$ cuyo índice $\{i, j_1, j_2, \cdots, j_p\}$ determina su signo.

Se dice que un baricentro cuyos arcos son todos del mismo signo es un *baricentro del mismo signo*. Si se asume para el ejemplo 3.52 que $\{a_{123}, a_{213}, a_{312}, a_{14}, a_{41}, a_{45}, a_{54}\}$ son todos positivos mientras que $\{a_{134}, a_{341}, a_{413}\}$ son todos negativos, entonces los cuatro baricentros de 3.52 son del mismo signo. El hipergrafo asociado se muestra en la Fig. 3.12.

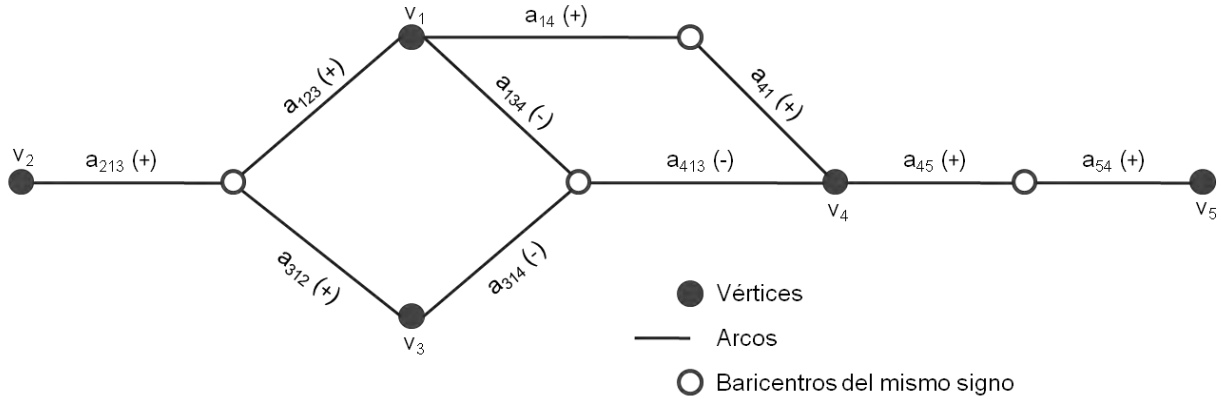


Fig. 3.12 Hipergrafo asociado al sistema dinámico 3.52 [28]

Para definir las condiciones de ciclo balanceado supóngase un hipergrafo con:

- $q + 1 \geq 2$ vértices distintos $\{v_1, \cdots, v_{q+1}\}$,
- q baricentros distintos $\{b_1, \cdots, b_q\}$, y
- $2q$ arcos de distinto signo $\{e_1, e_2, \cdots, e_{2q}\}$ con e_1 conectando a b_1 y v_1, e_2 conectando a b_1 y v_2, e_3 conectando b_2 y v_2, y así sucesivamente hasta e_{2q} conectando a b_q y v_{q+1}.

Se denomina al triple conjunto

$$[\{v_1, \cdots, v_{q+1}\}, \{b_1, \cdots, b_q\}, \{e_2, e_4, \cdots, e_{2q}\}]$$

la *trayectoria de* v_1 *a* v_{q+1}. Por supuesto, otro triple conjunto de la forma

$$[\{v_{q+1}, \cdots, v_1\}, \{b_q, \cdots, b_1\}, \{e_{2q-1}, \cdots, e_3, e_1\}]$$

constituye una segunda *trayectoria de* v_{q+1} *a* v_1 conocida como trayectoria *inversa*. Los hipergrafos asociados con cada una de las trayectorias se observan en la Fig. 3.13.

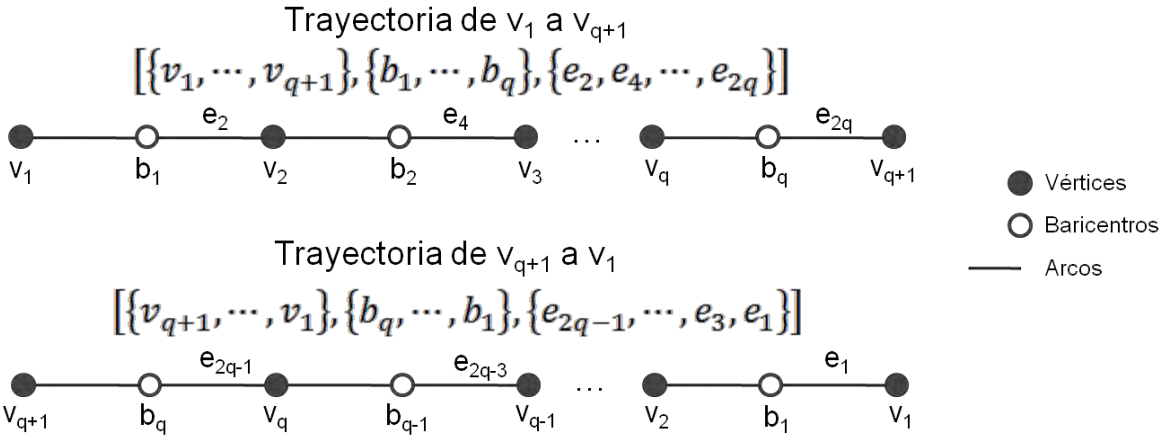


Fig. 3.13 Hipergrafos asociados a la trayectoria de v_1 *a* v_{q+1} *y su trayectoria inversa*

Se dice que un hipergrafo está *conectado* si cualesquiera dos vértices se encuentran en alguna trayectoria. Por otro lado, un ciclo con $q+1$ vértices, se define como una trayectoria de v_1 a v_{q+2} donde $v_{q+2} = v_1$, y cuyo conjunto triple asociado es $[\{v_1, \cdots, v_{q+2} = v_1\}, \{b_1, \cdots, b_{q+1}\}, \{e_2, e_4, \cdots, e_{2q+2}\}]$. Cabe señalar que para todo ciclo corresponde un ciclo inverso, que se define como la trayectoria de v_{q+2} a v_1 definida por el conjunto $[\{v_{q+2} = v_1, \cdots, v_1\}, \{b_{q+1}, \cdots, b_1\}, \{e_{2q+1}, \cdots, e_3, e_1\}]$. La Fig. 3.14 muestra un ciclo y su ciclo inverso asociado.

Cabe señalar que si cada baricentro en la trayectoria es del mismo signo, entonces el número de arcos negativos en un ciclo iguala al número de arcos negativos en su ciclo inverso. Por otro lado, el valor absoluto del producto de los $\{a\_\}$ coeficientes correspondientes a sus arcos se define como el *peso del ciclo*. Cabe destacar que como una propiedad de los hipergrafos, si un ciclo y su ciclo inverso tienen pesos iguales, entonces se dice que el ciclo es *balanceado*, lo que se denota como

$$|e_2 e_4 \cdots e_{2q+2}| = |e_{2q+1} \cdots e_3 e_1|$$

Para que los ciclos del ejemplo 3.52 sean balanceados se requiere entonces que $|a_{123}a_{314}| = |a_{134}a_{312}|$ y $|a_{14}a_{413}| = |a_{41}a_{134}|$.

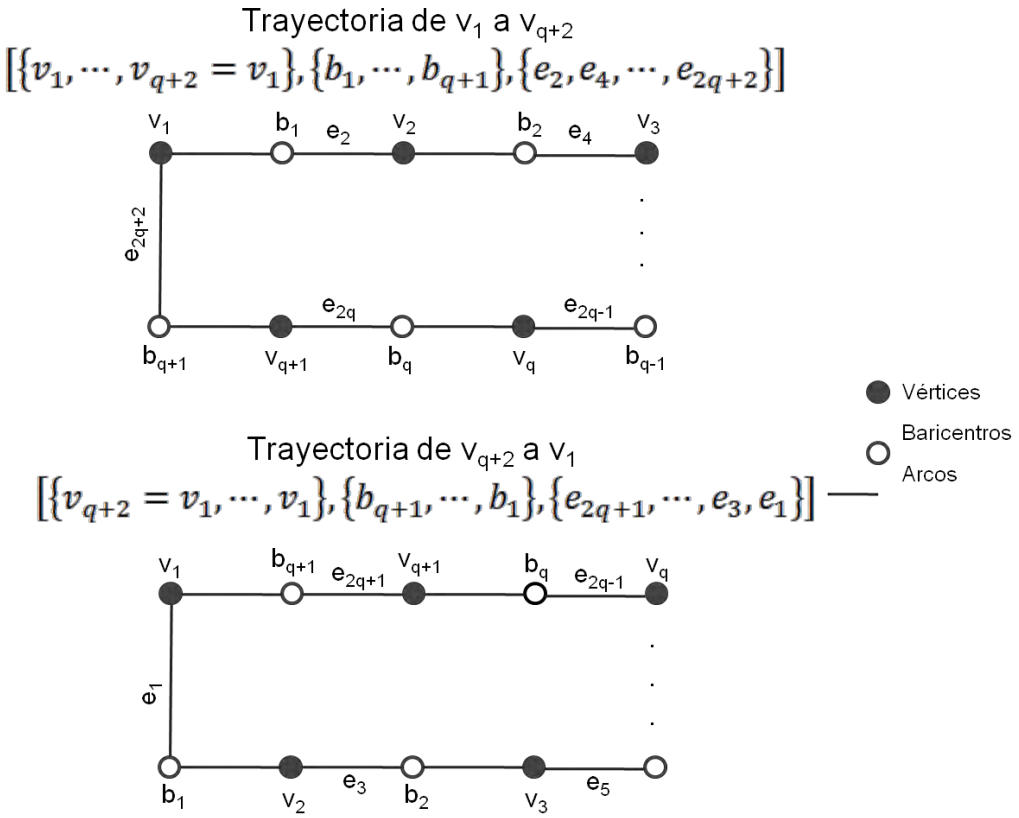


Fig. 3.14 Hipergrafo de un ciclo y su ciclo inverso asociado

Supóngase ahora que todos los baricentros del modelo 3.51 son del mismo signo, entonces, si existen constantes positivas $\{\lambda_1,\lambda_2,\cdots,\lambda_n\}$ para cada baricentro tal que se satisfaga $\lambda_i \left|a_{i\_}\right| = \lambda_j \left|a_{j\_}\right|$, con $\{i,j\}$ siendo cualesquiera índices de un coeficiente simplex, entonces las constantes $\{\lambda_i\}$ son llamadas *multiplicadores Volterra*, que en el caso específico de 3.52 deben cumplir:

$$\begin{aligned}
&\lambda_1 a_{123} = \lambda_2 a_{213} = \lambda_3 a_{312} \\
&\lambda_1 a_{134} = \lambda_3 a_{314} = \lambda_4 a_{413} \\
&\lambda_1 a_{14} = \lambda_3 a_{41} \\
&\lambda_4 a_{45} = \lambda_5 a_{54}
\end{aligned} \qquad 3.53$$

Los multiplicadores Volterra satisfacen las cuatro igualdades si $|a_{123}a_{314}| = |a_{134}a_{321}|$ y $|a_{14}a_{413}| = |a_{41}a_{134}|$ (condición de ciclo balanceado). En realidad, si cada baricentro en el hipergrafo de 3.51 es del mismo signo, entonces los multiplicadores Volterra existen sí y sólo sí cada uno de sus ciclos es balanceado. Todo lo anterior puede resumirse en los siguientes teoremas que Jeffries [28] propone y demuestra.

Lema 3.1. (Caso no lineal acíclico) Supóngase que el hipergrafo de 3.51 no tiene ciclos y que cada baricentro en el hipergrafo es del mismo signo. Entonces el sistema admite multiplicadores Volterra.

Lema 3.2. (Caso lineal) Supóngase que cada baricentro en el hipergrafo de 3.51 es del mismo signo y está conectado a exactamente dos vértices. Entonces los multiplicadores Volterra existen para 3.51 sí y sólo sí cada ciclo es balanceado.

Teorema 3.3. Supóngase que cada baricentro en el hipergrafo $\mathbf{H}$ de 3.51 es del mismo signo. Entonces los multiplicadores Volterra existen para 3.51 sí y sólo sí cada ciclo en 3.51 es balanceado.

Teorema 3.4. Si el hipergrafo de 3.51 tiene baricentros del mismo signo y ciclos balanceados (y por lo tanto multiplicadores Volterra), entonces cada trayectoria para 3.51 debe ser o debe aproximarse asintóticamente a una trayectoria constante.

Estos teoremas garantizan que un modelo del tipo 3.51 posee únicamente trayectorias constantes como atractores y no admite estados espurios. Para que los teoremas sean válidos, el modelo debe organizarse en subconjuntos de neuronas de interacción *mutuamente excitatoria* o *mutuamente inhibitoria* y cumplir las condiciones de ciclo balanceado. Además, cada subconjunto de neuronas debe representar un coeficiente simplex donde el tipo de interacción para cada uno es determinado por un baricentro del mismo signo (positivo para el caso excitatorio y negativo en el caso inhibitorio).

Retomando el modelo 3.51, es necesario encontrar alguna equivalencia entre la suma de los productos $a_{ij_1j_2\cdots j_p}\varphi_{j_1}\varphi_{j_2}\cdots\varphi_{j_p}$ con alguna otra suma de n términos cuyos multiplicandos sean φ_i o $(1-\varphi_i)$.

$$\frac{dx_i}{dt} = k_i(I_i - x_i) + \sum_j a_{ij_1j_2\cdots j_p}\varphi_{j_1}\varphi_{j_2}\cdots\varphi_{j_p} \tag{3.54}$$

Considérese que se tienen 2^n números reales $\left\{a_{ij_1j_2\cdots j_p}\right\}$, ahora fórmese una suma algebraica como un multinomio en los componentes del vector $\varphi = (\varphi_1, \varphi_2, \cdots, \varphi_n)$ de la forma:

$$\begin{aligned} A_a(\varphi) = a_0 + a_1\varphi_1 + a_2\varphi_2 + \cdots + a_n\varphi_n + a_{12}\varphi_1\varphi_2 + a_{13}\varphi_1\varphi_3 + \cdots \\ + a_{1n}\varphi_1\varphi_n + a_{23}\varphi_2\varphi_3 + a_{24}\varphi_2\varphi_4 + \cdots + a_{2n}\varphi_2\varphi_n \\ + a_{n-1\,n}\varphi_{n-1}\varphi_n + a_{i\cdots k}\varphi_i\cdots\varphi_k + a_{12\cdots n}\varphi_1\varphi_2\cdots\varphi_n \end{aligned} \tag{3.55}$$

Que en términos generales se expresa como:

$$A_a(\varphi) = a_n\varphi_n + a_{n-1\,n}\varphi_{n-1}\varphi_n + a_{i\cdots k}\varphi_i\cdots\varphi_k + a_{12\cdots n}\varphi_1\varphi_2\cdots\varphi_n \tag{3.56}$$

Por otro lado, supóngase que se cuenta con 2^n números reales $\left\{b\_\right\}$ cuyo subíndice es una cadena binomial de longitud n que a su vez establece una suma de productos de la forma:

$$\begin{aligned} B_b(\varphi) = {} & b_{00\cdots0}(1-\varphi_1)(1-\varphi_2)\cdots(1-\varphi_n) + b_{10\cdots0}\varphi_1(1-\varphi_2)\cdots(1-\varphi_n) \\ & + b_{01\cdots0}(1-\varphi_1)g_2\cdots(1-\varphi_n) + \cdots \\ & + b_{00\cdots1}(1-\varphi_1)(1-\varphi_2)\cdots(1-\varphi_n) + b_{11\cdots0}\varphi_1\varphi_2\cdots(1-\varphi_n) \\ & + \cdots + b_{10\cdots1}\varphi_1(1-\varphi_2)\cdots\varphi_n + b_{01\cdots1}(1-\varphi_1)\varphi_2\cdots\varphi_n + \cdots \\ & + b_{11\cdots1}\varphi_1\varphi_2\cdots\varphi_n \end{aligned} \qquad 3.57$$

Si se relacionan $A_a(\varphi)$ y $B_b(\varphi)$ como sigue,

$$\begin{aligned} & b_{00\cdots0} = a_0 \\ & b_{10\cdots0} = a_0 + a_1 \\ & b_{01\cdots0} = a_0 + a_2 \\ & \vdots \\ & b_{00\cdots1} = a_0 + a_n \\ & b_{110\cdots0} = a_0 + a_1 + a_2 + a_{12} \\ & b_{101\cdots0} = a_0 + a_1 + a_3 + a_{13} \\ & \vdots \\ & b_{100\cdots1} = a_0 + a_1 + a_n + a_{1n} \\ & b_{1110\cdots0} = a_0 + a_1 + a_2 + a_3 + a_{12} + a_{13} + a_{23} + a_{123} \\ & \vdots \\ & b_{11\cdots1} = a_0 + a_1 + a_2 + \cdots + a_n + a_{12} + a_{13} + a_{n-1\,n} + a_{123} + \cdots + a_{12\cdots n} \end{aligned} \qquad 3.58$$

Se puede decir que $\{b\_\}$ es derivado de $\{a\_\}$, por lo tanto $A_a(\varphi) = B_b(\varphi)$, siendo factible definir lo que se denomina *productos imagen* como la multiplicación de n elementos de la forma φ_i o $(1-\varphi_i)$ [28]. Cabe señalar que la presencia de φ_i o $(1-\varphi_i)$ en del producto imagen está determinada por los unos y ceros respectivamente de la combinación binomial en la cadena subíndice de $\{b\_\}$.

Este análisis con hipergrafos ha permitido justificar teóricamente el comportamiento de las trayectorias no constantes en el modelo general reduciéndolo a una multiplicación de productos imagen.

3.4 El modelo de memoria

Un modelo de memoria con redes neuronales artificiales puede ser considerado como un sistema que simula la recuperación de un patrón almacenado en la memoria de las redes neuronales biológicas. En términos matemáticos significa que el sistema converge a alguno de los patrones memoria (las trayectorias constantes previamente especificadas). Si las memorias se representan como vectores en el espacio n-dimensional con componentes ±1, entonces el reconocimiento matemático consiste en que el sistema converje desde cualquier vector de entrada dado a alguna de las memorias almacenadas. Cabe señalar que un modelo con n neuronas puede almacenar un número M de cadenas binomiales de tamaño n como atractores constantes tal que $1 \leq M \leq 2^n$, donde, por la regla del producto para pares ordenados, 2^n es el número máximo de maneras de ordenar cadenas binomiales de tamaño n.

Por otro lado, la función de activación del modelo tiene un rango de $[0,1]$ y puede ser una función escalón o lineal con umbral, cabe señalar que si su pendiente o ganancia es suficientemente alta, los únicos atractores constantes son las memorias dadas.

Entre las aplicaciones del modelo de memoria se encuentra la corrección de errores para la decodificación de código binario, que ha demostrado que dicho modelo es tan exacto como la distancia euclidiana en la convergencia de un estado inicial ruidoso a la memoria más cercana [28]. Este modelo puede ser considerado una memoria de contenido direccional, ya que el conocimiento parcial de una memoria guía a su recuperación completa a través de un sistema dinámico.

Si para la construcción del sistema dinámico se consideran como trayectorias atractor (trayectorias constantes o ciclos límite) a los 2^n vértices del n-cubo con coordenadas ± 1 en el espacio n-dimensional, entonces, cada trayectoria atractor cíclica está asociada con un recorrido cíclico por algunos ortantes del hiperespacio. Con un vértice del n-cubo inscrito en cada ortante asociado. Es importante hacer hincapié en que todos los atractores son disjuntos. Lo que significa que atractores diferentes corresponden a subconjuntos de vértices que no tienen ningún elemento común.

Cabe señalar que cada memoria constante (atractor) tiene asociado un ciclo de retroalimentación. Además, conforme transcurre el tiempo, el sistema es conducido a un estado particular en el que permanece hasta activar el ciclo de feedback respectivo.

Si se considera el modelo 3.59 bajo los supuestos de que para cada $i = 1,2,\cdots,n$ existe una función ρ_i continuamente diferenciable en el espacio n-dimensional y que además, k_i es una constante positiva, entonces el modelo tiene trayectorias constantes que resuelven el sistema de ecuaciones no lineales 3.60.

$$\frac{d(x_i)}{dt} = -k_i x_i + \rho_i\big(\varphi(x)\big) \qquad 3.59$$

$$x_i = k_i^{-1}\rho_i[\varphi(x)] \quad \text{donde } i = 1,\cdots,n \qquad 3.60$$

Nótese que la función de activación está presente en la dinámica del sistema de ecuaciones dividiendo el espacio de estados de 3.59 en 2^n hiperplanos que cortan sobre alguna $x_i = \pm\varepsilon$. Así, por cada x_i existen tres regiones diferentes, una para $x_i < -\varepsilon$, otra para $|x_i| \leq \varepsilon$ y una más para $x_i > \varepsilon$, por lo que el espacio n-dimensional queda dividido en 3^n regiones diferentes, conocidas como la *zona de transición* (T_ε) y los *ortantes relativos a* T_ε (O_ε). La primera región con al menos un elemento $|x_i| \leq \varepsilon$ y la segunda el complemento de la primera. Es claro que el número de ortantes relativos a T_ε por cada x_i es dos, por lo que en un espacio de dimensión n se tiene un total de 2^n ortantes en los que la función de ganancia es

constante (cero o uno), así que las trayectorias para 3.59 en O_ε son trayectorias del tipo,

$$\frac{dx_i}{dt} = k_i(x_i + c_i)$$

para algún c_i. Cabe señalar que el vector $c = (c_1, c_2, \cdots, c_n)$ puede o no permanecer en O_ε, sin embargo, si c es solución de 3.59, entonces c se encuentra en O_ε y además es una trayectoria atractor. Asumir que c_i es diferente de $\pm\varepsilon$ asegura que las trayectorias que entran a la zona de transición desde algún ortante relativo a T_ε lo hacen de forma transversal y no tangencial.

Si se asocia a cada ortante relativo una única combinación de productos imagen I con $0 \leq I \leq 1$ de la forma φ_i o $(1 - \varphi_i)$, entonces de los 2^n productos posibles existe una única forma posible de que $I = 1$ en exactamente un ortante relativo a T_ε y cero en todos los demás [28]. Así que incluir una cadena binaria de tamaño n como memoria constante implica incluir su producto imagen asociado en el modelo de memoria.

Supóngase ahora que se cuenta con M memorias diferentes formadas de n elementos, donde cada componente de la memoria se denota por $m_{Li} = 1$ o $m_{Li} = 0$ con $i = 1, \cdots, n$ y $L = 1, \cdots, M$. Entonces se dice que m_L es almacenada como una trayectoria constante $\underline{x}$ para el modelo de red neuronal 3.59 sí $\varphi_i(x_i) = m_{Li}$. Al trabajar con las M memorias dadas se obtienen *productos imagen* $\{I_L\}$ que Jeffries [28] propone como 3.61. Donde asocia con cada ortante relativo a T_ε una de las memorias dadas.

$$I_L(\varphi) = \prod_{j=1}^{n} \left\{ m_{Lj}\varphi_j(x_j) + (1 - m_{Lj})\left(1 - \varphi_j(x_j)\right) \right\} \qquad 3.61$$

Por ejemplo, si $n = 4$ y $m_1 = (0,1,0,1)$, entonces el producto imagen $I_1 = (1 - \varphi_1)\varphi_2(1 - \varphi_3)\varphi_4$ está activo en el ortante $(-,+,-,+)$ que contiene el punto $(-1,1,-1,1)$ e inactivo en los 15 ortantes restantes, lo que satisface $0 \leq I_L \leq 1$.

Si se piensa en una modificación del modelo general para almacenar las memorias $\{m_L\}$ como trayectorias atractor, entonces Jeffries [28] redefine a 3.59 como el modelo 3.62 para el que demuestra el Teorema 3.5.

$$\frac{dx_i}{dt} = -x_i + \sum_{L=1}^{M} (2m_{Li} - 1)\, I_L(\varphi) \qquad 3.62$$

Teorema 3.5. Si el estado $\underline{x}$ es una trayectoria constante para 3.62 en un ortante relativo a T_ε, entonces $\varphi(\underline{x})$ es una memoria del conjunto $\{m_L\}$. Además, si $\underline{x}$ se

define en términos de una memoria m_L por $\underline{x} = 2m_L - 1$, entonces $\underline{x}$ es una trayectoria constante estable para 3.62 con $\varphi(\underline{x}) = m_L$.

Es importante hacer notar que si un ortante relativo a T_ε no contiene alguna de las memorias del modelo 3.62, entonces cada trayectoria que pasa por O_ε se aproxima asintóticamente al origen hasta que dicha trayectoria entra a la zona de transición para cruzarla de forma transversal y alcanzar el ortante con la memoria más cercana.

Para analizar un ejemplo del comportamiento de un modelo de memoria considérese el siguiente sistema.

$$\begin{aligned} \frac{dx_1}{dt} &= -x_1 + \varphi_1(1-\varphi_2) - (1-\varphi_1)\varphi_2 \\ \frac{dx_2}{dt} &= -x_2 - \varphi_1(1-\varphi_2) + (1-\varphi_1)\varphi_2 \end{aligned} \qquad 3.63$$

Cuya función ganancia $\varphi_i(x_i)$ es una función lineal con umbral definida como

$$\varphi_i(x_i) = \begin{cases} 1 & x_i \geq \varepsilon_i \\ \dfrac{x_i + \varepsilon_i}{2\varepsilon_i} & -\varepsilon_i < x_i < \varepsilon_i \\ 0 & x_i \leq -\varepsilon_i \end{cases} \qquad 3.64$$

Con una gráfica de su comportamiento para $\varepsilon = 0.1$ como la que se muestra en la Fig. 3.15.

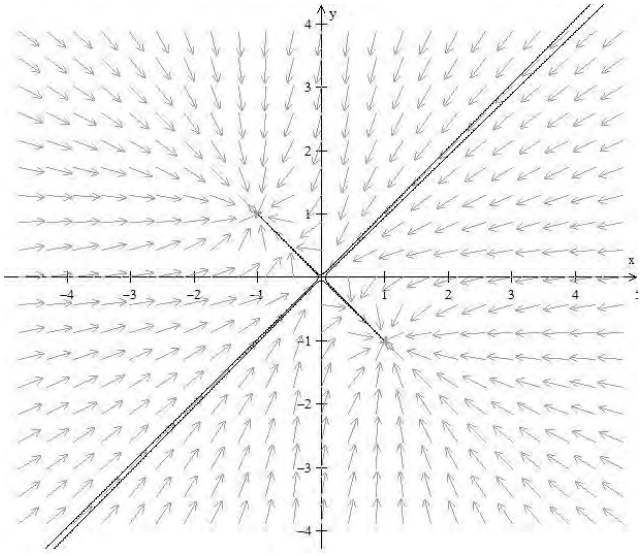


Fig. 3.15 Gráfica de las trayectorias del modelo 3.63.

Como puede observarse los estados estables del sistema son $(1,-1)$ y $(-1,1)$ que tienen asociados las memorias $(1,0)$ y $(0,1)$ respectivamente.

3.4.1 Dinámicas del modelo de memoria

Y aunque los teoremas de la sección 3.3.1 garantizan que aquellas trayectorias no constantes se acercan asintóticamente a una trayectoria constante. A continuación se realiza un análisis gráfico del comportamiento del modelo de memoria sobre todo en la región acotada por $\pm\varepsilon_i$ (se asume $\varepsilon_i = \varepsilon$ para todo ε_i). Para ello se muestran una serie de gráficas cuyos comportamientos difieren según existan o no memorias (atractores estables) en regiones adyacentes de un plano bidimensional. Se ha optado por ejemplificar la dinámica del modelo con gráficas de dos dimensiones por simplicidad.

De manera general cada punto x en T_ε puede caracterizarse por el número de coeficientes de x con $|x_i| < \varepsilon$, es decir, el número de neuronas en transición. Si exactamente una neurona está en transición, tal que $|x_1| < \varepsilon$ con $x = (x_1, x_2, \dots, x_n)$, entonces existen cuatro posibles tipos de trayectorias en los dos ortantes adyacentes (los ortantes que contienen $(1, x_2, \dots, x_n)$ y $(-1, x_2, \dots, x_n)$) dependiendo de si el ortante contiene o no una memoria.

En el caso de que ningún ortante adyacente contenga una memoria las trayectorias se comportan como las que muestra la Fig. 3.16, donde puede apreciarse que dichas trayectorias se aproximan directamente al origen hasta alcanzar alguna porción de T_ε que posea más neuronas en transición.

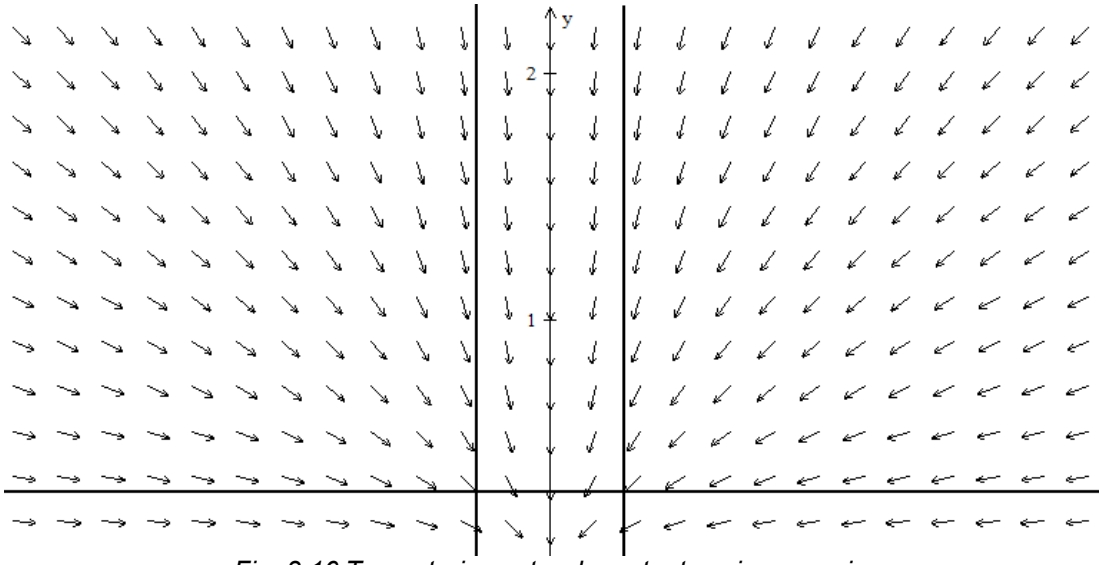


Fig. 3.16 Trayectorias entre dos ortantes sin memoria

Por otro lado, si exactamente un ortante contiene una memoria, entonces las trayectorias siguen el comportamiento de la Fig. 3.17.

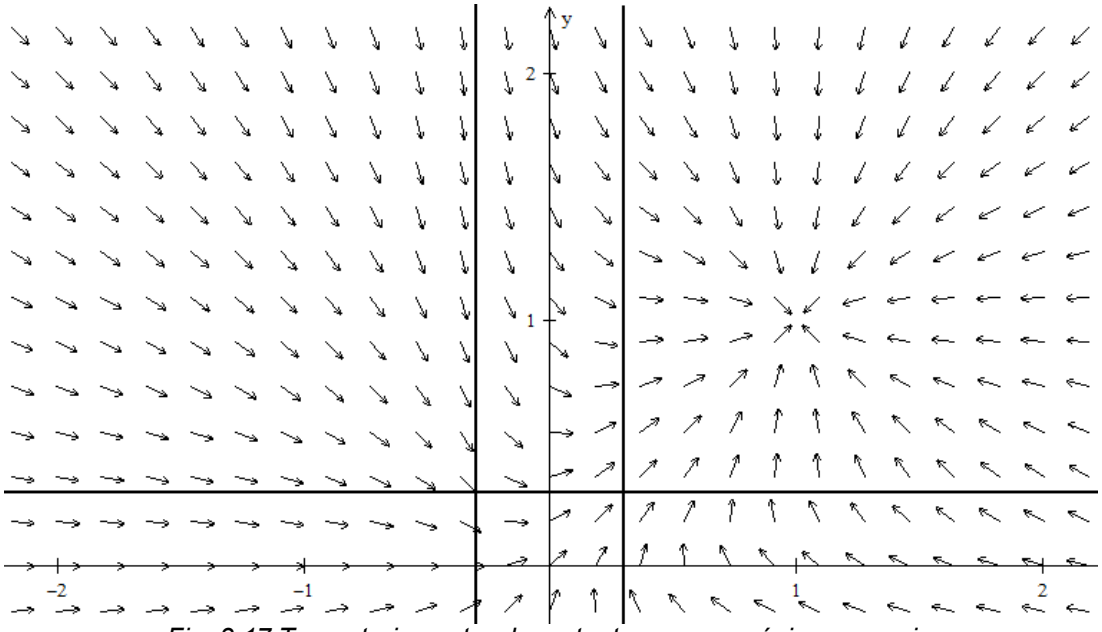


Fig. 3.17 Trayectorias entre dos ortantes con una única memoria

En cambio si ambos ortantes contienen memorias, entonces las trayectorias se comportan como las que se muestran en la Fig. 3.18.

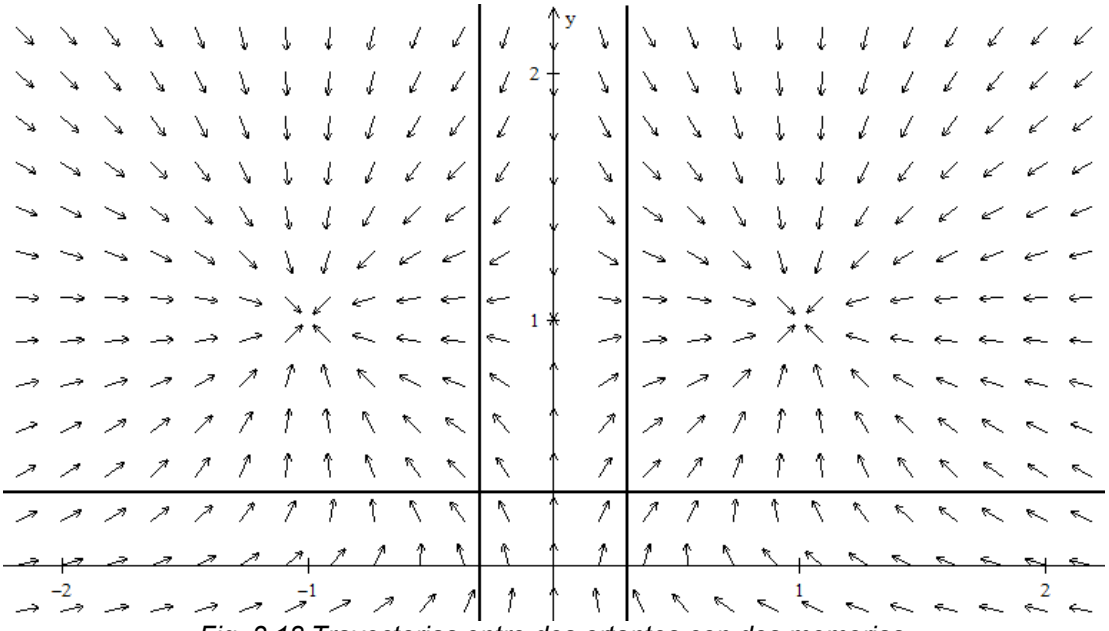


Fig. 3.18 Trayectorias entre dos ortantes con dos memorias

Considerando el caso en el que x yace en una porción de T_ε con exactamente dos neuronas en transición, entonces son posibles diferentes patrones trayectoria cerca de x y en los cuatro ortantes adyacentes dependiendo de la localización de las memorias en el plano. La Fig. 3.19 muestra algunos de los diferentes comportamientos de las trayectorias en modelos de memoria con dos, tres y cuatro memorias.

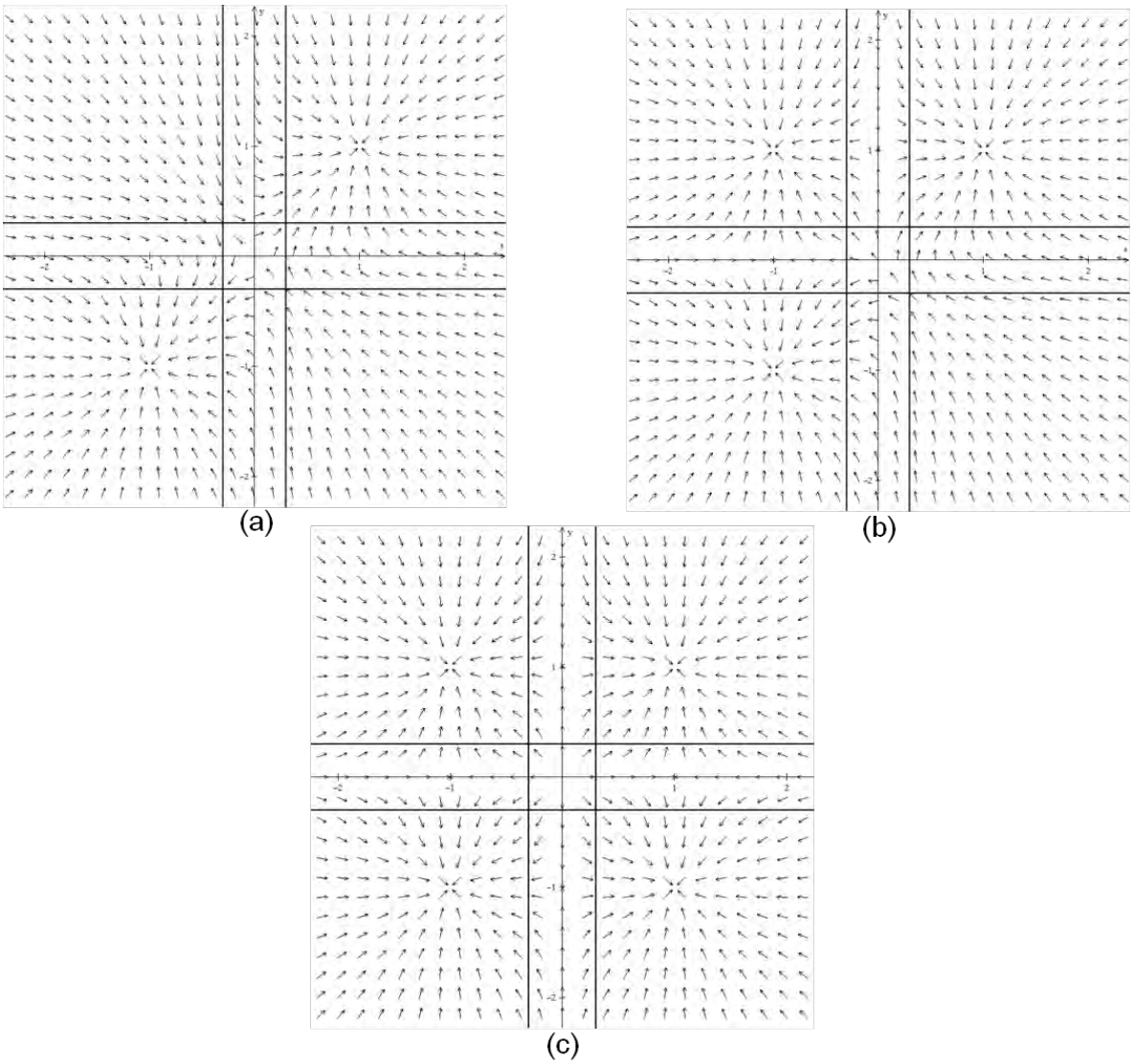


Fig. 3.19 Trayectorias con: (a) dos, (b) tres y, (c) cuatro atractores (memorias)

Las imágenes anteriores brindan una idea del comportamiento general de las trayectorias tanto en O_ε como en T_ε, y además, verifican el hecho de que aquellas trayectorias cuyo inicio está en un ortante sin memoria, se aproximan al origen hasta alcanzar la zona de transición donde se encuentran con otras neuronas en transición.

3.4.2 Aplicaciones del modelo de memoria

Supóngase a cada una de las memorias como un indicador que se activa cuando el sistema se encuentra en la memoria m_L. Si se pretende que el sistema converja rápidamente hacia determinada memoria, entonces se deben tomar muestras de los estados que conducen a esa memoria y emplearse como condiciones iniciales. Formar la cuenca de atracción de cada memoria, ya sea para conducir al sistema a un estado estable o bien a una secuencia de estados dependientes del tiempo,

representa una versión de lo que se conoce como aprendizaje de la red [28]. Siguiendo esta idea, retómese el modelo de memoria 3.62, en el que $\varphi_i(x_i)$ se define como una función lineal con umbral de la forma:

$$\varphi_i(x_i) = \begin{cases} 1 & x_i \geq \varepsilon_i \\ \dfrac{x_i + \varepsilon_i}{2\varepsilon_i} & -\varepsilon_i < x_i < \varepsilon_i \\ 0 & x_i \leq -\varepsilon_i \end{cases} \qquad 3.65$$

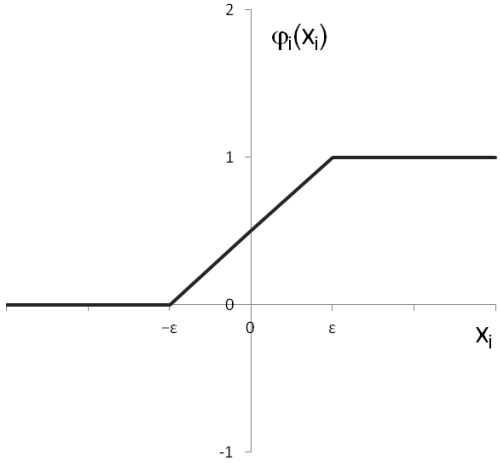


Fig. 3.20 Gráfica de la función $\varphi_i(x_i)$

Supóngase ahora que se tiene un cubo unitario del que se desean almacenar únicamente algunos de sus vértices como puntos fijos atractores. Por ejemplo, $\{(0,1,0),(1,1,0),(1,0,0),(1,0,1),(0,0,1),(0,1,1)\}$

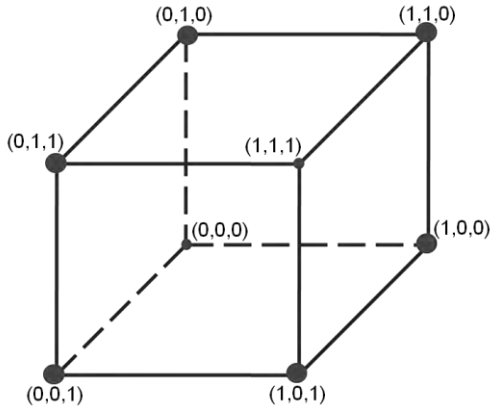


Fig. 3.21 Cubo unitario con algunos de sus vértices como memorias constantes

El sistema de ecuaciones asociado al problema es 3.66. Al igual que en los problemas de selección de conjuntos se aplica el método de Euler para encontrar las trayectorias constantes. La hoja de cálculo para la simulación se muestra en la Fig. 3.22.

$$\begin{aligned}\frac{dx_1}{dt} &= -x_1 - I_1 + I_2 + I_3 + I_4 - I_5 - I_6 \\ \frac{dx_2}{dt} &= -x_2 + I_1 + I_2 - I_3 - I_4 - I_5 + I_6 \\ \frac{dx_3}{dt} &= -x_3 - I_1 - I_2 - I_3 + I_4 + I_5 + I_6\end{aligned} \qquad 3.66$$

Modelo de memoria

Δt	0.1000000
ε	0.3000000

x_1	1.0000000	φ_1	1.0000000
x_2	-1.0000000	φ_2	0.0000000
x_3	1.0000000	φ_3	1.0000000

M_1	M_2	M_3	M_4	M_5	M_6
0.0000000	1.0000000	1.0000000	1.0000000	0.0000000	0.0000000
1.0000000	1.0000000	0.0000000	0.0000000	0.0000000	1.0000000
0.0000000	0.0000000	0.0000000	1.0000000	1.0000000	1.0000000

Elementos del producto imagen

$$m_{Lj}\varphi_j(x_j) + (1 - m_{Lj})\left(1 - \varphi_j(x_j)\right)$$

0.0000000	1.0000000	1.0000000	1.0000000	0.0000000	0.0000000
0.0000000	0.0000000	1.0000000	1.0000000	1.0000000	0.0000000
0.0000000	0.0000000	0.0000000	1.0000000	1.0000000	1.0000000

I_1	I_2	I_3	I_4	I_5	I_6

$$I_L(\varphi) = \prod_{j=1}^{n}\left\{m_{Lj}\varphi_j(x_j) + (1 - m_{Lj})\left(1 - \varphi_j(x_j)\right)\right\}$$

0.0000000	0.0000000	0.0000000	1.0000000	0.0000000	0.0000000

Coeficientes de I_L

$$2m_{Li} - 1$$

-1.0000000	1.0000000	1.0000000	1.0000000	-1.0000000	-1.0000000
1.0000000	1.0000000	-1.0000000	-1.0000000	-1.0000000	1.0000000
-1.0000000	-1.0000000	-1.0000000	1.0000000	1.0000000	1.0000000

Coeficiente de $I_L \times I_L$

$$(2m_{Li} - 1)I_L(\varphi)$$

0.0000000	0.0000000	0.0000000	1.0000000	0.0000000	0.0000000
0.0000000	0.0000000	0.0000000	-1.0000000	0.0000000	0.0000000
0.0000000	0.0000000	0.0000000	1.0000000	0.0000000	0.0000000

Siguiente x_1

x_1	1.0000000
x_2	-1.0000000
x_3	1.0000000

$$x_i(t + \Delta t) = x_i(t) + \left[-x_i(t) + \sum_{L=1}^{M}(2m_{Li} - 1)I_L(\varphi)\right]\Delta t$$

Fig. 3.22 Simulador del modelo de red neuronal 3.66

Realizando una serie de iteraciones sobre el modelo tal que el estado de inicio varié aleatoriamente, es posible observar que efectivamente para el modelo 3.66 las trayectorias de atracción constantes son aquellos vértices del cubo unitario

definidos como $\{m_L\}$ memorias. Hay que recordar que en la definición del modelo de memoria 3.62 se dice que un patrón de memoria m_L es incluido en la dinámica del sistema como una trayectoria atractor constante si $m_{Li} = \varphi_i(x_i)$, tomando esto en consideración la Fig. 3.23 muestra algunas de las simulaciones realizadas.

Estado de inicio aleatorio			Memoria atractor			
x_1	x_2	x_3	φ_1	φ_2	φ_3	m_i
0.8932829	-0.0816436	0.2825560	1.0000000	0.0000000	1.0000000	m_4
-0.1933719	0.7152680	-0.5722740	0.0000000	1.0000000	0.0000000	m_1
0.3692994	-0.3711701	-0.8939500	1.0000000	0.0000000	0.0000000	m_3
0.0061933	-0.2287183	0.2745010	0.0000000	0.0000000	1.0000000	m_5
-0.7589847	-0.3225424	-0.5408685	1.0000000	0.0000000	0.0000000	m_3
0.7484925	-0.7194477	-0.9590642	1.0000000	0.0000000	0.0000000	m_3
0.8467320	-0.6977764	-0.8605804	1.0000000	0.0000000	0.0000000	m_3
0.6444414	0.0155682	-0.1351013	1.0000000	0.0000000	0.0000000	m_3
0.6846714	0.4441935	-0.5385294	1.0000000	1.0000000	0.0000000	m_2
-0.0167184	-0.8959894	-0.2513338	1.0000000	0.0000000	0.0000000	m_3
0.9113020	0.8340097	-0.0552357	1.0000000	1.0000000	0.0000000	m_2
-0.5459597	0.6230933	0.5432406	0.0000000	1.0000000	1.0000000	m_6
0.5479302	-0.3202305	0.9519751	1.0000000	0.0000000	1.0000000	m_4

Fig. 3.23 Resultados de simulaciones para el modelo de red neuronal 3.66

Sin embargo, existen ocasiones en las que m_L no es necesariamente un punto fijo estable sino un estado de paso que guía al sistema a un nuevo estado y éste a su vez a otro más y así sucesivamente hasta que retorna al estado de inicio, es decir, el sistema se vuelve cíclico. A éste tipo de comportamiento se le conoce, en teoría de sistemas dinámicos, como *ciclos límite*, que en términos formales se define como una curva cerrada en el espacio n-dimensional. Acorde con Jeffries [28], posee las siguientes propiedades:

i. ninguna trayectoria constante está contenida dentro del ciclo límite,
ii. una trayectoria que inicia en un punto del ciclo límite debe permanecer en el ciclo límite de ahí en adelante,
iii. debe existir un número positivo ε tal que cada trayectoria que inicia dentro de ε del ciclo límite debe acercarse asintóticamente al ciclo límite y,
iv. para un número positivo ε debe haber un número positivo $\delta(\varepsilon)$ tal que se pueda garantizar que una trayectoria permanece dentro de ε del ciclo límite sí ésta inicia, dentro de ε del ciclo límite.

Siguiendo el modelo general, se define a 3.67 como una memoria que almacena $\{m_L\}$ memorias como ciclos límite [28]. Tal que $\{m_L\}$ sea una lista cíclica con $L = 1,2,\cdots,p$ y además $\{m_0 \equiv m_p, m_{p+1} \equiv m_L\}$.

$$\frac{dx_i}{dt} = -x_i + \sum_{L=1}^{M}\left(-m_{[L-1,i]} + m_{[L,i]} + 2m_{[L+1,i]} - 1\right)I_L(\varphi) \qquad 3.67$$

Asimismo se dice que $\{m_L\}$ se almacena como un ciclo límite atractor si $\{\varphi_i(x_i)\} = \{m_L\}$. Los productos imagen y la función de activación se definen como:

$$I_L(\varphi) = \prod_{j=1}^{n}\left\{m_{Lj}\varphi_j x_j + \left(1 - m_{Lj}\right)\left(1 - \varphi_j\left(x_j\right)\right)\right\} \qquad 3.68$$

$$\varphi_i(x_i) = \begin{cases} 1 & x_i \geq \varepsilon_i \\ \dfrac{x_i + \varepsilon_i}{2\varepsilon_i} & -\varepsilon_i < x_i < \varepsilon_i \\ 0 & x_i \leq -\varepsilon_i \end{cases} \qquad 3.69$$

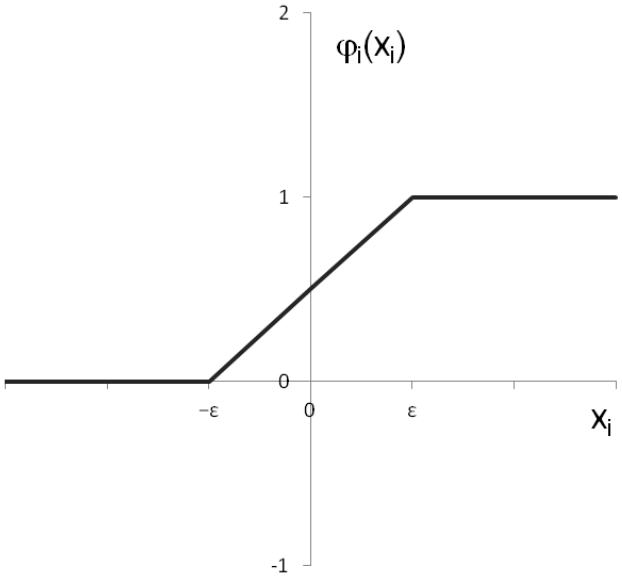


Fig. 3.24 Gráfica de la función $\varphi_i(x_i)$

Para ejemplificar el modelo de red neuronal con un ciclo límite como trayectoria atractor, supóngase que se desea obtener una red para recorrer los ocho vértices de un cubo unitario siguiendo la trayectoria $\{m_L\} = \{\rightarrow (1,1,1) \rightarrow (1,1,0) \rightarrow (1,0,0) \rightarrow (1,0,1) \rightarrow (0,0,1) \rightarrow (0,0,0) \rightarrow (0,1,0) \rightarrow (0,1,1) \rightarrow (1,1,1) \rightarrow\}$ con $L = 1,2,\cdots,8$, cuyo grafo dirigido se puede observar en la Fig. 3.25.

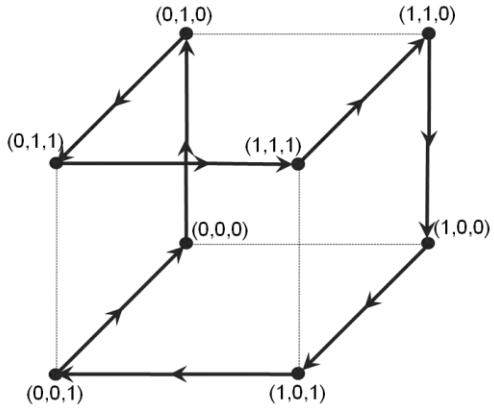


Fig. 3.25 Ciclo límite que recorre los ocho vértices de un cubo unitario

Una vez que el ciclo ha sido definido sólo queda obtener el modelo de red neuronal 3.67 asociado:

$$\begin{aligned}\frac{dx_1}{dt} &= -x_1 + 2I_1 + I_2 + I_3 - I_4 - 2I_5 - I_6 - I_7 + I_8 \\ \frac{dx_2}{dt} &= -x_2 + I_1 - I_2 - 2I_3 - I_4 - I_5 + I_6 + 2I_7 + I_8 \\ \frac{dx_3}{dt} &= -x_3 - I_1 - 2I_2 + I_3 + 2I_4 - I_5 - 2I_6 + I_7 + 2I_8\end{aligned} \qquad 3.70$$

Por otro lado, si se emplea el método de Euler para encontrar las trayectorias constantes de 3.70, es preciso desarrollar un algoritmo para efectuar los cálculos numéricos implicados. Se ha optado por crear una hoja de cálculo como la de la Fig. 3.27 que permite simulaciones del modelo.

Para corroborar que efectivamente 3.70 tenga al ciclo límite de la Fig. 3.25 como trayectoria atractor, se realizan una serie de iteraciones sobre la hoja de cálculo tal que el estado de inicio de x_i es modificado aleatoriamente. Los resultados de las simulaciones pueden ser vistos en la Fig. 3.26.

	$x_i(0)$	$\varphi_i[x_i(0)]$	$\varphi_i[x_i(t+\Delta t)]$							
x_1	0.6723602	1.0000000	1.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000	1.0000000	1.0000000
x_2	-0.8806310	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000	1.0000000	1.0000000	1.0000000	0.0000000
x_3	-0.3115262	0.0000000	1.0000000	1.0000000	0.0000000	0.0000000	1.0000000	1.0000000	0.0000000	0.0000000
x_1	-0.5035682	0.0000000	0.0000000	1.0000000	1.0000000	1.0000000	1.0000000	0.0000000	0.0000000	0.0000000
x_2	0.3101970	1.0000000	1.0000000	1.0000000	1.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000
x_3	-0.4942328	0.0000000	1.0000000	1.0000000	0.0000000	0.0000000	1.0000000	1.0000000	0.0000000	0.0000000
x_1	0.9319671	1.0000000	1.0000000	1.0000000	1.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000
x_2	0.7159764	1.0000000	1.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000	1.0000000	1.0000000
x_3	0.6426880	1.0000000	0.0000000	0.0000000	1.0000000	1.0000000	0.0000000	0.0000000	1.0000000	1.0000000

Fig. 3.26 Resultados de simulaciones para el modelo de red neuronal 3.70

Modelo que almacena $\{m_L\}$ memorias como un ciclo límite que recorre los ocho vértices de un cubo unitario

Δt	0.1000000
ε	0.0010000

x_1	0.6723602	$\varphi_1(x_1)$	1.0000000
x_2	-0.8806310	$\varphi_2(x_2)$	0.0000000
x_3	-0.3115262	$\varphi_3(x_3)$	0.0000000

M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9
0.0000000	1.0000000	1.0000000	1.0000000	1.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000
1.0000000	1.0000000	1.0000000	0.0000000	0.0000000	0.0000000	0.0000000	1.0000000	1.0000000	1.0000000
1.0000000	1.0000000	0.0000000	0.0000000	1.0000000	1.0000000	0.0000000	0.0000000	1.0000000	1.0000000

Productos imagen

$$I_L(\varphi) = \prod_{j=1}^{n} \left\{ m_{Lj}\varphi_j(x_j) + (1 - m_{Lj})\left(1 - \varphi_j(x_j)\right) \right\}$$

I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8
0	0	1	0	0	0	0	0

Coeficientes de I_L

$$-m_{[L-1,i]} + m_{[L,i]} + 2m_{[L+1,i]} - 1$$

2.000000	1.000000	1.000000	-1.000000	-2.000000	-1.000000	-1.000000	1.000000
1.000000	-1.000000	-2.00000	-1.000000	-1.000000	1.000000	2.000000	1.000000
-1.000000	-2.000000	1.000000	2.000000	-1.000000	-2.000000	1.000000	2.000000

Coeficiente de I_L x producto imagen I_L

$$\left(-m_{[L-1,i]} + m_{[L,i]} + 2m_{[L+1,i]} - 1\right) I_L(\varphi)$$

0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	-2.000000	0.000000	0.000000	0.000000	0.000000	0.000000
0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000

Siguiente x_i

x_1	0.705124
x_2	-0.992568
x_3	-0.180374

$$x_i(t + \Delta t) = x_i(t) + \left[-x_i(t) + \sum_{L=1}^{M} \left(-m_{[L-1,i]} + m_{[L,i]} + 2m_{[L+1,i]} - 1 \right) I_L(\varphi) \right] \Delta t$$

Fig. 3.27 Simulador para el modelo de red neuronal 3.70

Como puede observarse en la Fig. 3.26 es evidente que la lista cíclica $\{m_L\}$ es almacena por el modelo 3.70 como un ciclo límite atractor, ya que sin importar el estado de inicio x_i, el sistema converge al ciclo límite y permanece en él.

Si se piensa en una combinación de los modelos 3.62 y 3.67, entonces se obtiene una red neuronal de grado superior con la capacidad de almacenar $\{m_L\}$ memorias

como trayectorias atractor y ciclos límite. La adición de los modelos anteriores en una única red neuronal queda definida acorde con Jeffries [28] de la forma:

$$\frac{dx_i}{dt} = -x_i + \sum_{L=1}^{S}\left(-m_{[L-1,i]} + m_{[L,i]} + 2m_{[L+1,i]} - 1\right)I_L(\varphi) + \sum_{L=S+1}^{M}\left(2m_{Li} - 1\right)I_L(\varphi) \qquad 3.71$$

Donde tanto $I_L(\varphi)$ como $\varphi_i(x_i)$ mantienen su definición previa. En caso de presentarse un problema que involucra tanto puntos fijos constantes como ciclos límite se cuenta con un modelo de red neuronal que abarca tal caso. Supóngase ahora que se desea diseñar un sistema de dimensión cinco para almacenar las memorias descritas en la Tabla 3.6.

$\{m_L\}$	Tipo de trayectoria
$\{\rightarrow (1,1,1,1,1) \rightarrow (1,1,1,1,0) \rightarrow (1,1,1,0,0) \rightarrow (1,1,1,0,1) \rightarrow (1,1,1,1,1)\}$	Ciclo límite
$\{(\rightarrow (0,1,1,1,1)) \rightarrow (0,0,1,1,1) \rightarrow (0,0,0,1,1) \rightarrow (0,0,0,0,1) \rightarrow (0,1,0,0,1) \rightarrow (0,1,1,0,1) \rightarrow (0,1,1,1,1) \rightarrow\}$	Ciclo límite
$\{(0,0,0,0,0)\}$	Atractor
$\{(1,0,0,0,0)\}$	Atractor

Tabla 3.6 Memorias que definen dos ciclos límite y dos atractores constantes [28]

Préstese atención al hecho de que para el ejemplo anterior se tiene $S = 10$ y $M = 12$, por lo tanto existen 10 memorias que pertenecen a dos ciclos límites, el primero con 4 de ellas y el segundo con 6, así como 2 memorias atractor. Por lo que el modelo neuronal asociado al problema es 3.72. Como se ha venido haciendo en problemas anteriores el sistema se traslada a una hoja de cálculo como la que muestra la Fig. 3.29 y se aplica el método de Euler para encontrar sus trayectorias atractor.

$$\begin{aligned}
\frac{dx_1}{dt} &= -x_1 + I_1 + I_2 + I_3 + I_4 - I_5 - I_6 - I_7 - I_8 - I_9 - I_{10} - I_{11} + I_{12} \\
\frac{dx_2}{dt} &= -x_2 + I_1 + I_2 + I_3 + I_4 - I_5 - 2I_6 - I_7 + I_8 + 2I_9 + I_{10} - I_{11} - I_{12} \\
\frac{dx_3}{dt} &= -x_3 + I_1 + I_2 + I_3 + I_4 + I_5 - I_6 - 2I_7 - I_8 + I_9 + 2I_{10} - I_{11} - I_{12} \\
\frac{dx_4}{dt} &= -x_4 + 2I_1 - I_2 - 2I_3 + I_4 + 2I_5 + I_6 - I_7 - 2I_8 - I_9 + I_{10} - I_{11} - I_{12} \\
\frac{dx_5}{dt} &= -x_5 - I_1 - 2I_2 + I_3 + 2I_4 + I_5 + I_6 + I_7 + I_8 + I_9 + I_{10} - I_{11} - I_{12}
\end{aligned} \qquad 3.72$$

Al realizar una serie de iteraciones sobre la hoja de cálculo partiendo de estados aleatorios para x_i, se verifica el hecho de que el modelo 3.72 tiene a las memorias de la Tabla 3.6 como trayectorias atractor en la forma de dos ciclos límite y dos puntos fijos estables. Los resultados arrojados en diferentes simulaciones se muestran en la Fig. 3.28.

	$x_i(0)$	$\varphi_i[x_i(0)]$	$\varphi_i[x_i(t+\Delta t)]$ → Primer ciclo límite				
x_1	0.4494561	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000
x_2	0.9110711	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000
x_3	0.6649835	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000
x_4	-0.3605690	0.0000000	0.0000000	1.0000000	1.0000000	0.0000000	0.0000000
x_5	-0.0601245	0.0000000	1.0000000	1.0000000	0.0000000	0.0000000	1.0000000

	$x_i(0)$	$\varphi_i[x_i(0)]$	$\varphi_i[x_i(t+\Delta t)]$ → Primer ciclo límite						
x_1	0.0480670	1.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
x_2	-0.6043085	0.0000000	0.0000000	0.0000000	1.0000000	1.0000000	1.0000000	0.0000000	0.0000000
x_3	0.3749508	1.0000000	0.0000000	0.0000000	0.0000000	1.0000000	1.0000000	1.0000000	0.0000000
x_4	0.8709492	1.0000000	1.0000000	0.0000000	0.0000000	0.0000000	1.0000000	1.0000000	1.0000000
x_5	0.9605063	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000

	$x_i(0)$	$\varphi_i[x_i(0)]$	$\varphi_i[x_i(t+\Delta t)]$ → Primer atractor constante
x_1	-0.1706007	0.0000000	0.0000000
x_2	-0.5594596	0.0000000	0.0000000
x_3	0.3718006	1.0000000	0.0000000
x_4	-0.1435972	0.0000000	0.0000000
x_5	-0.3226400	0.0000000	0.0000000

	$x_i(0)$	$\varphi_i[x_i(0)]$	$\varphi_i[x_i(t+\Delta t)]$ → Segundo ciclo límite						
x_1	0.1726352	1.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
x_2	-0.6081021	0.0000000	0.0000000	1.0000000	1.0000000	1.0000000	0.0000000	0.0000000	1.0000000
x_3	-0.8725952	0.0000000	0.0000000	0.0000000	1.0000000	1.0000000	1.0000000	0.0000000	0.0000000
x_4	0.5226403	1.0000000	0.0000000	0.0000000	0.0000000	1.0000000	1.0000000	1.0000000	0.0000000
x_5	0.0794237	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000	1.0000000

	$x_i(0)$	$\varphi_i[x_i(0)]$	$\varphi_i[x_i(t+\Delta t)]$ → Primer atractor constante
x_1	0.7249788	1.0000000	1.0000000
x_2	-0.7242119	0.0000000	0.0000000
x_3	-0.7208712	0.0000000	0.0000000
x_4	-0.0036957	0.3152148	0.0000000
x_5	-0.8545805	0.0000000	0.0000000

Fig. 3.28 Resultados de simulaciones para el modelo de red neuronal 3.72

Después de observar los resultados se reconoce la efectividad del sistema en la convergencia a alguno de los puntos fijos estables o bien, a algún ciclo límite, independientemente del estado de inicio. Estos resultados corroboran la funcionalidad y robustez del modelo de memoria con ciclos límite.

Modelo de memoria de dimensión cinco asociado con dos ciclos límite y dos memorias constantes

Δt	0.10000
ε	0.01000

$\{m_L\}$	Memoria constante con componentes $m_{Li} \in \{1,0\}$
p	Número de memorias del ciclo límite
I_L	Producto imagen de m
$\{m_{p+1}\}$	$\{m_1\}$
$\{m_0\}$	$\{m_p\}$

x_1	0.44946	$\varphi_1(x_1)$	1.00000
x_2	0.91107	$\varphi_2(x_2)$	1.00000
x_3	0.66498	$\varphi_3(x_3)$	1.00000
x_4	-0.36057	$\varphi_4(x_4)$	0.00000
x_5	-0.06012	$\varphi_5(x_5)$	0.00000

Memorias del 1er ciclo límite ($p = 4$)

M_0	M_1	M_2	M_3	M_4	M_5
1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
0.00000	1.00000	1.00000	0.00000	0.00000	1.00000
1.00000	1.00000	0.00000	0.00000	1.00000	1.00000

Memorias del 2° ciclo límite ($p = 6$)

M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
1.00000	1.00000	0.00000	0.00000	0.00000	1.00000	1.00000	1.00000
1.00000	1.00000	1.00000	0.00000	0.00000	0.00000	1.00000	1.00000
0.00000	1.00000	1.00000	1.00000	0.00000	0.00000	0.00000	1.00000
1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000

I_1	I_2	I_3	I_4
0.00000	0.00000	1.00000	0.00000

$$I_L(\varphi) = \prod_{j=1}^{n} \left\{ m_{Lj}\varphi_j(x_j) + (1 - m_{Lj})\left(1 - \varphi_j(x_j)\right) \right\}$$

I_5	I_6	I_7	I_8	I_9	I_{10}
0.00000	0.00000	0.00000	0.00000	0.00000	0.00000

$$I_L(\varphi) = \prod_{j=1}^{n} \left\{ m_{Lj}\varphi_j(x_j) + (1 - m_{Lj})\left(1 - \varphi_j(x_j)\right) \right\}$$

Coeficientes de I_L para x_i

$$Coeficiente\ de\ I_L = -m_{[L-1,i]} + m_{[L,i]} + 2m_{[L+1,i]} - 1$$

x_1	1.00000	1.00000	1.00000	1.00000
x_2	1.00000	1.00000	1.00000	1.00000
x_3	1.00000	1.00000	1.00000	1.00000
x_4	2.00000	-1.00000	-2.00000	1.00000
x_5	-1.00000	-2.00000	1.00000	2.00000

Coeficientes de I_L para x_i

$$Coeficiente\ de\ I_L = -m_{[L-1,i]} + m_{[L,i]} + 2m_{[L+1,i]} - 1$$

x_1	-1.00000	-1.00000	-1.00000	-1.00000	-1.00000	-1.00000
x_2	-1.00000	-2.00000	-1.00000	1.00000	2.00000	1.00000
x_3	1.00000	-1.00000	-2.00000	-1.00000	1.00000	2.00000
x_4	2.00000	1.00000	-1.00000	-2.00000	-1.00000	1.00000
x_5	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000

1er atractor constante
0.00000
0.00000
0.00000
0.00000
0.00000
I_{11}
0.00000
Coeficiente I_L para x_i
-1.00000
-1.00000
-1.00000
-1.00000
-1.00000

$$I_L(\varphi) = \prod_{j=1}^{n} \left\{ m_{Lj}\varphi_j(x_j) + (1 - m_{Lj})\left(1 - \varphi_j(x_j)\right) \right\}$$

$$Coeficiente\ I_L = 2m_{Li} - 1$$

2° atractor constante
1.00000
0.00000
0.00000
0.00000
0.00000
I_{12}
0.00000
Coeficiente I_L para x_i
1.00000
-1.00000
-1.00000
-1.00000
-1.00000

Coeficiente I_L × I_L

$(Coeficiente\ I_L)(I_L)$

	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}	I_{11}	I_{12}
x_1	0.00000	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
x_2	0.00000	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
x_3	0.00000	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
x_4	0.00000	0.00000	-2.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
x_5	0.00000	0.00000	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000

Siguiente x_i	
x_1	0.50451
x_2	0.91996
x_3	0.69848
x_4	-0.52451
x_5	0.04588

$$x_i(t + \Delta t) = x_i(t) + \left[-x_i(t) + \sum_{L=1}^{S} \left(-m_{[L-1,i]} + m_{[L,i]} + 2m_{[L+1,i]} - 1 \right) I_L(\varphi) + \sum_{L=S+1}^{M} (2m_{Li} - 1)\, r_L[I_L(\varphi)] \right] \Delta t$$

Fig. 3.29 Simulador para el modelo de red neuronal 3.72

3.4.3 Simulador del modelo de memoria

Para llevar al ámbito general el modelo de memoria con atractores estables, de tal suerte que se elimine la necesidad de elaborar una hoja de cálculo para cada conjunto diferente de patrones memoria (p. ej. {A,B,C}, {1,2,3}), y además, se utilice el mismo software para redes con distinta cantidad de neuronas. Se ha elaborado un programa que simula el comportamiento de una red neuronal asociativa. La meta es ligar un estado de inicio ruidoso con su correspondiente memoria aprendida previamente. El objetivo del simulador es automatizar todos los procesos de diseño y cálculo de iteraciones necesarios para problemas diferentes.

Retomando el modelo de memoria con puntos fijos estables.

$$\frac{dx_i}{dt} = -x_i + \sum_{L=1}^{M} (2m_{Li} - 1)\, I_L(\varphi)$$

Se han diseñado dos versiones de simulador, una que emplea el método de Euler para la solución del sistema de ecuaciones diferenciales y otra que hace uso del método Runge-Kutta de cuarto orden. Esto con fines de comparación durante la convergencia.

Cabe destacar que el simulador posee los elementos característicos de una red neuronal, por ejemplo, cuenta con un conjunto de neuronas cuyo número está en función del problema (con un máximo de cuatrocientas y un mínimo de una), tiene una función de activación (función lineal con umbral) directamente relacionada al valor umbral o valor de activación (épsilon) de la neurona, además, cada neurona se retroalimenta a sí misma y a otras neuronas construyendo una red neuronal de capa simple con feedback. Por otro lado, su regla de aprendizaje sigue el modelo de memoria con puntos fijos estables.

A continuación se hace una descripción breve del funcionamiento básico del simulador y se presenta un ejemplo que hace uso del mismo.

1. Simulador: Red Neuronal con Atractores Estables (RNAE)

Sigue el modelo de memoria con atractores estables. Su funcionamiento consiste en recuperar a partir de un estado de inicio cualquiera algún patrón que haya sido aprendido previamente por la red. Como se ha venido mencionando, este objetivo se alcanza con el uso de la teoría de sistemas dinámicos y métodos numéricos, donde cada patrón aprendido por la red neuronal es considerado como un punto fijo del sistema de ecuaciones diferenciales.

A grandes rasgos, se puede decir que el modelo tiene a su cargo el diseño de la red de neuronas y la regla de aprendizaje, dejando a los métodos numéricos el resto del trabajo. Cabe destacar que aún cuando el modelo no contiene de forma

explícita la aplicación de la función de activación, ésta se encuentra implícita en los denominados *productos imagen* utilizados en el mismo.

Para un mejor entendimiento de cómo funciona el simulador, la Fig. 3.30 muestra la interfaz gráfica del programa y la Tabla 3.7 detalla cada uno de sus componentes.

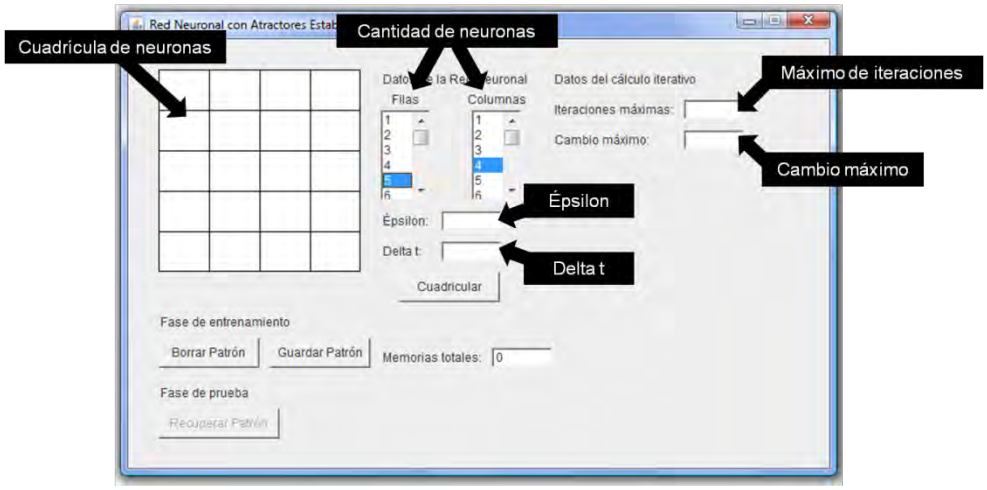


Fig. 3.30 Interfaz gráfica del simulador "Red Neuronal con Atractores Estables"

Componente	Descripción
Cuadrícula de neuronas	Representa en sentido literal el estado de cada neurona, donde una celda negra señala que la neurona se encuentra en un estado activo y, por el contrario, si la celda es blanca la neurona está inactiva.
Cantidad de neuronas	Determina la cifra total de neuronas que tiene la red al multiplicar el número de filas por columnas.
Épsilon	Establece el valor umbral de la función de activación.
Delta t	Hace alusión a la fracción de tiempo incrementada en cada iteración del método numérico.
Iteraciones máximas	Significa el máximo de iteraciones permitidas al programa antes de la convergencia.
Cambio máximo	Indica qué tan diferente debe ser el valor anterior del actual para que el sistema converja a un estado estable (error medio admitido).

Tabla 3.7 Componentes principales del simulador "Red Neuronal con Atractores Estables"

Básicamente el simulador *"Red Neuronal con Atractores Estables"* trabaja de la siguiente manera:

- primeramente, se deben definir los parámetros de la Tabla 3.7,

- a continuación, se inicia con la denominada fase de entrenamiento durante la cual el programa obtiene desde de la cuadrícula de neuronas cada una de las memorias a aprender,
- para seguir con la fase de prueba, donde a partir de un patrón cualquiera dibujado sobre la cuadrícula de neuronas el sistema captura el estado de inicio de la red y empieza a trabajar con el modelo,
- por último, el programa muestra sobre la cuadrícula el estado final de cada neurona, el cual, según el Teorema 3.5, debe corresponder a una de las memorias previamente aprendidas por la red.

Cabe señalar que la fase de prueba pone en marcha el diseño de la red neuronal y el cálculo iterativo evaluando en cada iteración los *productos imagen*. Además de realizar el proceso de recuperación de la memoria. Es importante hacer hincapié en el hecho de que la fase de entrenamiento puede continuar aún después de la fase de prueba.

A continuación se muestra un ejemplo que hace uso del simulador, bajo el supuesto de que es capaz de aprender los números arábigos. Las características primarias de la red se muestran en la Tabla 3.8. Mientras que los patrones de memoria se ilustran el la Fig. 3.31.

Característica	Valor
Total de neuronas:	20
Número de filas:	5
Número de columnas:	4
Épsilon	0.00001
Delta t	0.1
Iteraciones máximas	500
Cambio máximo	$1.0\ E-10$

Tabla 3.8 Características de la red neuronal que aprende los números arábigos

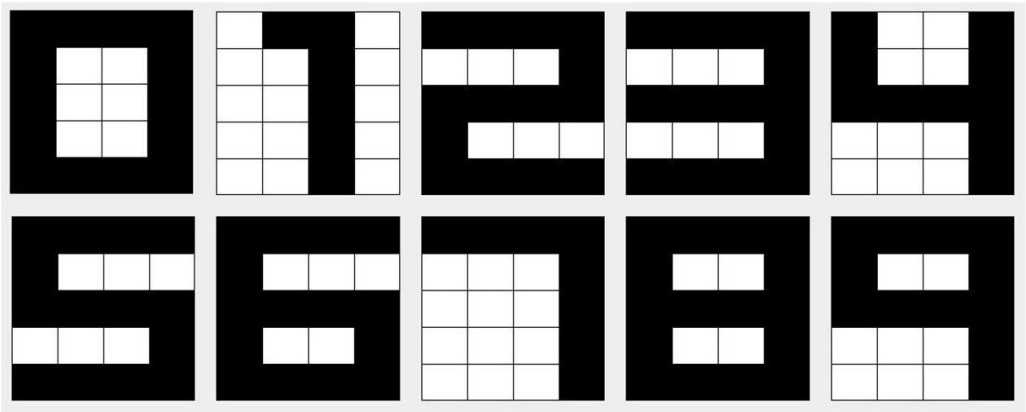


Fig. 3.31 Fase de entrenamiento para el simulador RNAE

En este ejemplo no importa el orden en que los patrones de memoria son aprendidos por la red, ya que la meta es simplemente reconocer dichos patrones y no así su orden de precedencia (como ocurre en un ciclo límite). Una vez señalado lo anterior se da paso a la fase de prueba de la que se obtienen los resultados de la Tabla 3.9.

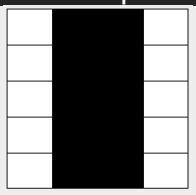
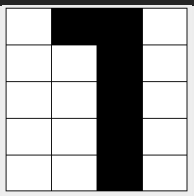
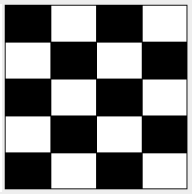
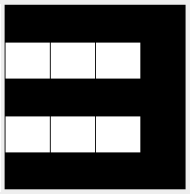
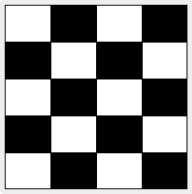
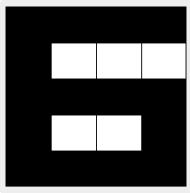
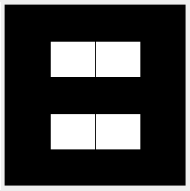
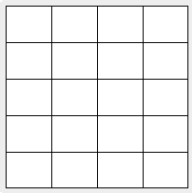
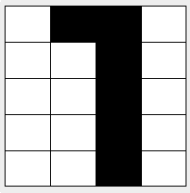
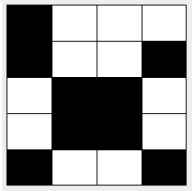
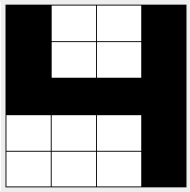
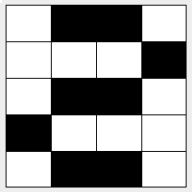
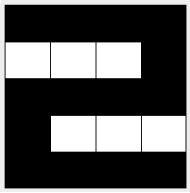
No. prueba	Patrón de prueba	Respuesta del simulador
P_1		
P_2		
P_3		
P_4		
P_5		
P_6		
P_7		

Tabla 3.9 Resultados de la fase de prueba con el simulador RNAE

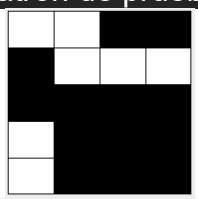
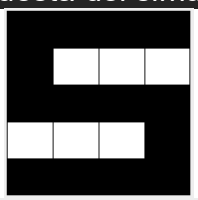
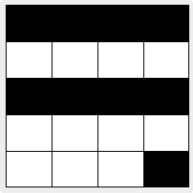
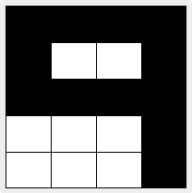
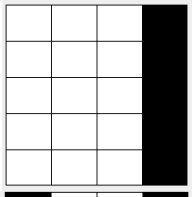
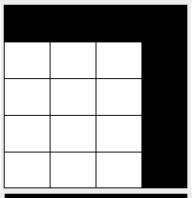
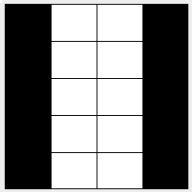
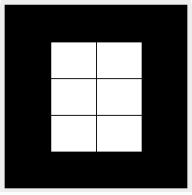
No. prueba	Patrón de prueba	Respuesta del simulador
P_8		
P_9		
P_{10}		
P_{11}		

Tabla 3.9 Resultados de la fase de prueba con el simulador RNAE (Continuación)

Como puede observarse, el simulador *"Red Neuronal con Atractores Estables"* bajo las especificaciones de la Tabla 3.8 converge a un estado estable con cada patrón de entrada durante la fase de prueba. Es sumamente importante recalcar el hecho de que el simulador está diseñado para funcionar únicamente con patrones que siguen la escala especificada por la cuadrícula de neuronas, es decir, cada patrón que se desee sea aprendido y recuperado por la red ha de trasladarse a la escala planteada al inicio de la ejecución del programa.

Por otro lado, ha de señalarse la trascendencia de encontrar el valor apropiado de épsilon tal que el comportamiento de la red cumpla las expectativas en cada caso de estudio. En el ejemplo actual un valor más grande conduce al sistema a un estado de caos.

3.4.4 Análisis de resultados

Una sencilla forma de corroborar que efectivamente el patrón recuperado por el simulador corresponde a la memoria más cercana, es a través de lo que se conoce como distancia de Hamming[2], que en éste caso, mide la diferencia de estados entre dos conjuntos de neuronas. Para calcular la distancia de Hamming

[2] La distancia de Hamming entre dos cadenas binomiales es el número de coeficientes en los que las cadenas difieren.[28]

obsérvese primeramente la Fig. 3.32 que contiene la representación vectorial de las memorias y los patrones de prueba. Donde los estados de activo e inactivo son simbolizados por 1 y 0 respectivamente.

Representación neuronal de la memoria																				
Memoria	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	x_{16}	x_{17}	x_{18}	x_{19}	x_{20}
0	1	1	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	1	1	1
1	0	1	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0
2	1	1	1	1	0	0	0	1	1	1	1	1	1	0	0	0	1	1	1	1
3	1	1	1	1	0	0	0	1	1	1	1	1	0	0	0	1	1	1	1	1
4	1	0	0	1	1	0	0	1	1	1	1	1	0	0	0	1	0	0	0	1
5	1	1	1	1	1	0	0	0	1	1	1	1	0	0	0	1	1	1	1	1
6	1	1	1	1	1	0	0	0	1	1	1	1	1	0	0	1	1	1	1	1
7	1	1	1	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
8	1	1	1	1	1	0	0	1	1	1	1	1	1	0	0	1	1	1	1	1
9	1	1	1	1	1	0	0	1	1	1	1	1	0	0	0	1	0	0	0	1

Representación neuronal del patrón de prueba																				
Patrón de prueba	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	x_{16}	x_{17}	x_{18}	x_{19}	x_{20}
P_1	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
P_2	1	0	1	0	0	1	0	1	1	0	1	0	0	1	0	1	1	0	1	0
P_3	0	1	0	1	1	0	1	0	0	1	0	1	1	0	1	0	0	1	0	1
P_4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
P_5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P_6	1	0	0	0	1	0	0	1	0	1	1	0	0	1	1	0	1	0	0	1
P_7	0	1	1	0	0	0	0	1	0	1	1	0	1	0	0	0	0	1	1	0
P_8	0	0	1	1	1	0	0	0	1	1	1	1	0	1	1	1	0	1	1	1
P_9	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
P_{10}	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
P_{11}	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1

Fig. 3.32 Representación numérica de los conjuntos de neuronas

Con esta información es posible calcular el número de estados diferentes entre los patrones de prueba y las memorias aprendidas. Los resultados pueden ser vistos en la Fig. 3.33 que enfatiza la distancia más pequeña entre cada par comparado. Asimismo se confirma que los patrones recuperados por el simulador concuerdan con aquellos que se obtienen si en lugar de utilizar el modelo de memoria se emplea la distancia de Hamming.

		Patrones de prueba										
		P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}
Memorias aprendidas	M_1	16	10	10	6	14	13	10	9	9	9	4
	M_2	4	10	10	14	6	11	6	11	9	11	16
	M_3	12	10	10	6	14	11	6	9	5	11	8
	M_4	12	8	12	6	14	11	8	7	5	9	8
	M_5	16	10	10	10	10	7	12	7	5	5	4
	M_6	12	10	10	6	14	11	10	5	5	11	8
	M_7	13	11	9	5	15	12	9	6	6	12	7
	M_8	14	10	10	12	8	11	10	11	5	3	6
	M_9	14	10	10	4	16	11	8	7	7	11	6
	M_{10}	14	10	10	8	12	9	10	7	3	7	6

Fig. 3.33 Distancia de Hamming entre los patrones de prueba y las memorias

Puede pensarse que es más sencillo emplear la distancia de Hamming que el modelo de memoria, sin embargo, como se observa en la Fig. 3.33 el número de valores diferentes entre pares comparados puede repetirse (véase el caso del patrón de prueba número siete) y, decidir cuál de todos es, sino el correcto al menos el mejor como respuesta, no es una tarea simple y mucho menos si se pretende llevar a un algoritmo de cómputo. Por ello las redes neuronales artificiales resultan de gran ayuda puesto que ofrecen esa habilidad inherente al cerebro humano, ya que si se visualiza de manera subjetiva el patrón de prueba siete, puede corroborarse que la forma de éste se acerca más al dígito dos que al dígito uno.

El Apéndice A muestra un comparativo entre dos variantes del sistema, una con el método de Euler y otra con el método Runge-Kutta de cuarto orden, ambas utilizadas en el aprendizaje de los números arábigos y bajo las características establecidas en la Tabla 3.8. Al cotejar los resultados obtenidos en ambos sistemas se puede concluir que en este caso el método de Euler converge más rápido que el Runge-Kutta, sin embargo, el costo por la pérdida de información con valores muy pequeños tiende a reducir ésta ventaja. Ya que al utilizar umbrales cercanos a cero es preferible la exactitud en el manejo de la información que la velocidad con que se alcanza el punto de equilibrio.

Este problema de reconocimiento de caracteres (los números arábigos, en este caso) es una aplicación que ya ha sido abordada con anterioridad utilizando redes neuronales artificiales. Por ejemplo, Haykin[2] se vale de una red de Hopfield donde, al igual que en el modelo de memoria de Jeffries [28], los puntos fijos son los patrones a memorizar (memorias fundamentales) por la red y las neuronas pueden tomar valores de ± 1 (encendida y apagada respectivamente).

El modelo de Haykin [2] tiene una fase de aprendizaje en la que calcula el peso de las conexiones para un conjunto de memorias fundamentales, utilizando la regla del producto externo. Posteriormente, durante la fase de reconocimiento, las neuronas adquieren el estado del patrón de prueba como estado de inicio. Este estado inicial se modifica iterativamente acorde a la función signo que compara el potencial de acción (la suma ponderada del estado de la neurona) con el valor umbral (cero para el ejemplo). La modificación continúa hasta que el estado de las neuronas alcanza un estado estable, es decir, permanece sin cambio o hasta un error permitido. Siendo este punto fijo la salida de la red.

Aunque los resultados son prometedores, las dificultades, como el autor lo reconoce, se hacen presentes cuando la red converge a un patrón erróneo partiendo de la versión corrupta de una memoria fundamental (patrón memoria). Además, a diferencia del modelo de memoria con sistemas dinámicos, admite la presencia de estados espurios (que no corresponden a ninguna de las memorias fundamentales). Estos estados degenerados suponen un problema, ya que en algunos casos, la salida de la red resulta en una combinación lineal de un número singular de patrones almacenados, y en otros, la correlación con alguna de las memorias fundamentales es nula.

Por otro lado, Anderson [23], utiliza un modelo al que define como *modelo de vector sumado*. Este modelo supone que la fuerza sináptica global es la suma de la historia de su activación sináptica, donde los miembros del conjunto de aprendizaje se forman al sumar los M vectores memoria de dimensión n, aprendidos por la red. Aunque el resultado final de la suma de las entradas parece a primera vista ruido aleatorio, la información aún está presente. El problema surge cuando se quiere recuperar cualquier vector almacenado, ya que, tal como el autor lo afirma, se ha perdido tanta información que sacar lo que queda es todo un reto.

El modelo del vector sumado, sirve más como un filtro para detectar la presencia o ausencia de un vector de memoria, bajo el supuesto de que si el patrón está presente, el producto interno es positivo y grande.

El modelo de vector sumado no permite conocer los valores utilizados para almacenar la memoria, a diferencia del modelo de memoria con sistemas dinámicos, donde, a partir de un análisis de las ecuaciones del sistema, es posible obtener sus memorias. Sin embargo, ambos modelos suponen que el patrón de entrada contiene una gran cantidad de información que determina el patrón salida.

Este estudio del modelo de Hopfield y del vector sumado, permite advertir que el modelo de memoria asociativa con sistemas dinámicos propuesto por Jeffries [28], aporta considerables ventajas sobre las otras técnicas. Y aunque conforme hay un incremento en el número de neuronas el sistema pierde estabilidad debido a la cantidad de iteraciones necesarias para alcanzar el estado de equilibro, resulta adecuado para los propósitos de esta tesis.

Conclusiones

El objetivo principal de esta investigación ha sido dar un enfoque de sistemas dinámicos al modelo de redes neuronales artificiales de memoria asociativa y mostrar su efectividad mediante su simulación en software. Bajo el supuesto de que las memorias corresponden a estados de equilibrio (atractores estables) en el espacio fase del sistema. Donde la convergencia al atractor apropiado es llamada reconocimiento.

Al medir los resultados del modelo por su habilidad para reconocer el patrón correcto aprendido previamente (*content-addressability* [30]), puede decirse que su desempeño tuvo el comportamiento esperado. Al asociar cada patrón de prueba con el número arábigo con mayor grado de similitud (menor distancia de hamming). Además, en ningún caso se alcanzo un estado espurio que no correspondiera a alguno de los patrones memorizados con anterioridad. Lo que habla de un buen grado de generalización.

Sin embargo, si el desempeño se mide por su capacidad de memoria (número de patrones que pueden ser aprendidos y correctamente reconocidos por la red) hay que reconocer las desventajas del modelo. Aunque teóricamente para el ejemplo el sistema tiene la capacidad de aprender 2^{20} memorias diferentes, un sistema dinámico de tales magnitudes resulta difícil de manipular. Ya que conforme aumenta el número de memorias también se incrementa el número de cálculos. Siendo necesario calcular en cada iteración tantos productos imagen por neurona como patrones de aprendizaje existan. Además, cada uno de los 2^{20} ortantes del hiperespacio tendría asociado un atractor estable, por lo que todo patrón de prueba permanecería en el ortante de origen. Lo que se traduce en un sistema tan generalizado que hace innecesario el análisis del comportamiento de las trayectorias y elimina la ventaja de tratar con información parcialmente corrompida, al abarcar todas las posibles combinaciones de estados de inicio (nótese que esto último sólo aplica para el software de reconocimiento de patrones).

No obstante, el modelo muestra características propias de una memoria asociativa. Ya que se construye a partir del aprendizaje (los números arábigos en el ejemplo). Es de contenido direccional, al no responder a una medida de similitud (distancia de hamming o distancia euclidiana) entre el patrón de prueba y las memorias aprendidas. Sino que una vez que la entrada perturba el estado del

sistema, las neuronas comienzan a interactuar unas con otras hasta alcanzar nuevamente un estado estable al que asocian la salida. Asimismo, tiene la capacidad de trabajar con información parcialmente corrompida al reconocer correctamente todos los patrones distorsionados o incompletos utilizados como estados de inicio en el ejemplo.

Por otro lado, el enfoque de sistemas dinámicos abre la pauta para posibles trabajos futuros donde se profundice sobre que tanto se acercan estos modelos a los procesos cognitivos. Ya que de acuerdo con Izhikevich [31], los neurocientíficos han encontrado constantes fenómenos de equilibro, estabilidad, atractores ciclos límite y bifurcaciones en el comportamiento de agrupaciones locales de neuronas.

Además, queda pendiente la automatización de los modelos de selección de conjuntos, ciclos límite y ciclos límite con atractores estables. Cabe la posibilidad de que estos modelos puedan incluirse en un mismo software que permita adaptarlos (p. ej. modificar las funciones de activación, construir redes con capas ocultas o utilizar la salida de un modelo con atractores estables como entrada a uno con ciclos límite) para observar su comportamiento y trasladar el modelo a problemas que muestren conductas semejantes.

Y ya que es trabajo de los matemáticos aplicados y computólogos, estudiar las propiedades de nuevos sistemas utilizando técnicas empleadas en otras áreas del conocimiento con el apoyo de herramientas de cómputo. Se ha puesto en evidencia que la fusión de las redes neuronales artificiales, los sistemas dinámicos, las neurociencias y el desarrollo computacional, estimula el replanteamiento de viejos paradigmas desde un nuevo enfoque y fomenta un trabajo conjunto interdisciplinario que conduce a nuevos horizontes.

Bibliografía

[1] J. A. Escamilla Reyna. (1994). *Modelos matemáticos para el estudio de la activación de la corteza cerebral* [En línea]. Disponible: http://www.red-mat.unam.mx/foro/volumenes/vol022/TesisEscamilla-f.pdf (Último acceso: 19 de julio de 2016).

[2] S. O. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. New Jersey: Prentice Hall, 1999.

[3] R. Lahoz-Beltra, *Bioinformática: Simulación, Vida Artificial e Inteligencia Artificial*, España: Ediciones Díaz de Santos, 2004.

[4] R. Flores López y J. M. Fernández Fernández, *Las Redes Neuronales Artificiales Fundamentos Teóricos y Aplicaciones Prácticas*, La Coruña: Netbiblo, 2008.

[5] J. R. Hilera González, *Redes Neuronales Artificiales Fundamentos, Modelos y Aplicaciones*, Madrid: RA-MA, 1994.

[6] A. Córdova Martínez y A. Chicote, *Fisiología Dinámica*, España: Elsevier, 2003.

[7] F.C. Hoppensteadt, *et al*., *Weakly Connected Neural Networks*, New York: Springer Science, 1997.

[8] R. Gómez Nesterkín. (1996). *Modelación y predicción mediante redes funcionales* [En línea]. Disponible: http://www.red-mat.unam.mx/foro/volumenes/vol002/redfun.ps (Último acceso: 19 de julio de 2016).

[9] M. T. Hagan, *et al*. *Neural Network Design* (2 ed.) [En línea]. Disponible: http://hagan.okstate.edu/NNDesign.pdf (Último acceso: 19 de julio de 2016).

[10] R. Rojas, *Neural Networks a Systematic Introduction*, New York: Springer, 1996.

[11] N. De Abajo Martínez y A. Gómez Gómez, *Introducción a la Inteligencia Artificial: Sistemas Expertos, Redes Neuronales Artificiales y Computación Evolutiva*, España: Universidad de Oviedo, 2001.

[12] B. D. Ripley, *Patter Recognititon and Neural Networks*, Massachusetts: Cambridge University Press, 2005.

[13] J. J. Montaño Moreno. (2002). *Redes neuronales artificiales aplicadas al análisis de datos* [En línea]. Disponible: http://www.tesisenred.net/bitstream/handle/10803/9441/tjjmm1de1.pdf (Último acceso: 19 de julio de 2016).

[14] R. Khoury. (2007). *Introduction to Soft Computing* [En línea]. Disponible: http://pami.uwaterloo.ca/~khoury/ece457s07/Lecture12.pdf (Último acceso: 19 de julio de 2016).

[15] C. F. Eick. (2008). *What is soft computing Techniques used in soft computing* [En línea]. Disponible: http://www2.cs.uh.edu/~ceick/6367/Soft-Computing.pdf (Último acceso: 19 de julio de 2016).

[16] L. Fausett, *Fundamentals of Neural Networks: Architectures, Algorithms and Applications*, New Jersey: Prentice-Hall, 1994.

[17] J. Muñoz Pérez, *Inteligencia Computacional Inspirada en la Vida*, Málaga: SPICUM, 2010.

[18] J. F. Araya. (2004). *Acerca de modelos neuronales: redes neuronales estáticas y recurrentes* [En línea]. Disponible: http://www2.ing.puc.cl/~jfaraya/docs/NeuralGuide.pdf (Último acceso: 10 de abril de 2010).

[19] J. M. Mendel y K. Fu, *Adpatative Learning and Pattern Recognition Systems*, vol. 66, USA: Academic Press, 1970.

[20] J. Schneider y A. W. Moore. (1997, Feb 7). *A Locally Weighted Learning Tutorial using Vizier 1.0* [En línea]. Disponible: https://www.cs.cmu.edu/~schneide/tut5/node9.html (Último acceso: 19 de julio de 2016).

[21] V. Weisz, *et al.*, “Modelizando el cerebro: redes neuronales artificiales,” *Revista del Hospital Italiano de Buenos Aires,* vol. 25, nº 3, pp. 130-136, 2005.

[22] J. K. Wu, *Neural Networks and Simulation Methods*, New York: Marcel Dekker, 1994.

[23] J. A. Anderson, *Redes Neurales*, México: Alfaomega, 2007.

[24] J. Hefferon. (2008). *Linear Algebra* [En línea]. Disponible: http://joshua.smcvt.edu/linearalgebra/book.pdf (Último acceso: 19 de julio de 2016).

[25] J. Rosales Ortega. *Sistemas Dinámicos Elementales* [En línea]. Disponible: https://tecdigital.tec.ac.cr/revistamatematica/MundoMatematicas/Sistemas_dinamicos_elementales/Sisdinamicos.pdf (Último acceso 19 de julio de 2016).

[26] L. P. Shilnikov, *et al*., *Methods of Qualitative Theory in Nonlinear Dynamics Part I,* vol. IV, New Jersey: World Scientific, 1998.

[27] E. R. Scheinerman, *Invitation to Dynamical Systems*, USA: Johns Hopkins University Press, 1996.

[28] C. Jeffries, *Code Recognition and Set Selection with Neural Networks, Boston*, USA: Birckhäuser, 1991.

[29] G. Teschl, *Ordinary Differential Equations and Dynamical Systems, Graduate Studies in Mathematics*, vol. 140, Rhode Island: American Mathematical Society, 2012.

[30] K. Prasad, *et al.*, "A Study on Associative Neural Memories," *IJACSA. International Journal of Advanced Computer Science and Applications,* vol. 1, nº 6, pp. 124-133, Dic 2010.

[31] E. M. Izhikevich, *Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting*, Massachusetts: MIT Press, 2007.

Apéndices

Apéndice A

Comparativo en la convergencia del método numérico de Euler y Runge-Kutta de cuarto orden en la solución del sistema de ecuaciones para un modelo de memoria durante el reconocimiento de los dígitos de Sistema Métrico Decimal implementado sobre el Simulador Red Neuronal con Atractores Estables bajo las condiciones establecidas en la Tabla 3.8.

	Análisis de convergencia para el método de Euler y Runge-Kutta de cuarto orden sobre el simulador Red Neuronal con Atractores Estables -RNAE-				
	Método de Euler	Método Runge Kutta 4^{o} orden	Diferencia en valor absoluto	Método que se acerca más a la convergencia	Patrón de prueba
Iteración:	311	325	14		
X_1	-0.99999999912910500	-0.99999999907391400	0.00000000005519096	Euler	P_1
X_2	0.99999999912910500	0.99999999907391400	0.00000000005519096	Euler	
X_3	0.99999999912910500	0.99999999907391400	0.00000000005519096	Euler	
X_4	-0.99999999912910500	-0.99999999907391400	0.00000000005519096	Euler	
X_5	-0.99999999912910500	-0.99999999907391400	0.00000000005519096	Euler	
X_6	-0.99999999912909400	-0.99999999907389900	0.00000000005519496	Euler	
X_7	0.99999999912910500	0.99999999907391400	0.00000000005519096	Euler	
X_8	-0.99999999912910500	-0.99999999907391400	0.00000000005519096	Euler	
X_9	-0.99999999912910500	-0.99999999907391400	0.00000000005519096	Euler	
X_{10}	-0.99999999912909400	-0.99999999907389900	0.00000000005519496	Euler	
X_{11}	0.99999999912910500	0.99999999907391400	0.00000000005519096	Euler	
X_{12}	-0.99999999912910500	-0.99999999907391400	0.00000000005519096	Euler	
X_{13}	-0.99999999912910500	-0.99999999907391400	0.00000000005519096	Euler	
X_{14}	-0.99999999912909400	-0.99999999907389900	0.00000000005519496	Euler	
X_{15}	0.99999999912910500	0.99999999907391400	0.00000000005519096	Euler	
X_{16}	-0.99999999912910500	-0.99999999907391400	0.00000000005519096	Euler	
X_{17}	-0.99999999912910500	-0.99999999907391400	0.00000000005519096	Euler	
X_{18}	-0.99999999912909400	-0.99999999907389900	0.00000000005519496	Euler	
X_{19}	0.99999999912910500	0.99999999907391400	0.00000000005519096	Euler	
X_{20}	-0.99999999912910500	-0.99999999907391400	0.00000000005519096	Euler	

	Análisis de convergencia para el método de Euler y Runge-Kutta de cuarto orden sobre el simulador Red Neuronal con Atractores Estables -RNAE-				
	Método de Euler	Método Runge Kutta 4° orden	Diferencia en valor absoluto	Método que se acerca más a la convergencia	Patrón de prueba
Iteración:	320	334	14		
X_1	0.99999999912906300	0.99999999908930000	0.00000000003976297	Euler	
X_2	0.99999999912905700	0.99999999908929300	0.00000000003976397	Euler	
X_3	0.99999999912906200	0.99999999908929900	0.00000000003976297	Euler	
X_4	0.99999999912905800	0.99999999908929300	0.00000000003976497	Euler	
X_5	-0.99999999912903000	-0.99999999908929300	0.00000000003973699	Euler	
X_6	-0.99999999912905800	-0.99999999908929300	0.00000000003976497	Euler	
X_7	-0.99999999912906300	-0.99999999908930000	0.00000000003976297	Euler	
X_8	0.99999999912905000	0.99999999908929700	0.00000000003975298	Euler	
X_9	0.99999999912906200	0.99999999908929900	0.00000000003976297	Euler	
X_{10}	0.99999999912905400	0.99999999908929200	0.00000000003976197	Euler	P_2
X_{11}	0.99999999912905900	0.99999999908929900	0.00000000003975997	Euler	
X_{12}	0.99999999912905800	0.99999999908929300	0.00000000003976497	Euler	
X_{13}	-0.99999999912903700	-0.99999999908929500	0.00000000003974199	Euler	
X_{14}	-0.99999999912905800	-0.99999999908929300	0.00000000003976497	Euler	
X_{15}	-0.99999999912906300	-0.99999999908930000	0.00000000003976297	Euler	
X_{16}	0.99999999912905600	0.99999999908929800	0.00000000003975797	Euler	
X_{17}	0.99999999912905800	0.99999999908929900	0.00000000003975897	Euler	
X_{18}	0.99999999912905400	0.99999999908929200	0.00000000003976197	Euler	
X_{19}	0.99999999912905800	0.99999999908929900	0.00000000003975897	Euler	
X_{20}	0.99999999912905800	0.99999999908929300	0.00000000003976497	Euler	
Iteración:	321	335	14		
X_1	0.99999999912902500	0.99999999907391600	0.00000000005510903	Euler	
X_2	0.99999999912902800	0.99999999907392200	0.00000000005510603	Euler	
X_3	0.99999999912902400	0.99999999907391600	0.00000000005510803	Euler	
X_4	0.99999999912902900	0.99999999907392200	0.00000000005510703	Euler	
X_5	0.99999999912902200	0.99999999907391400	0.00000000005510803	Euler	
X_6	-0.99999999912902900	-0.99999999907392200	0.00000000005510703	Euler	
X_7	-0.99999999912902500	-0.99999999907391600	0.00000000005510903	Euler	
X_8	-0.99999999904591700	-0.99999999907386700	0.00000000002795009	Runge-Kutta	
X_9	0.99999999912902400	0.99999999907391600	0.00000000005510803	Euler	
X_{10}	0.99999999912902400	0.99999999907391700	0.00000000005510703	Euler	P_3
X_{11}	0.99999999912902000	0.99999999907391100	0.00000000005510903	Euler	
X_{12}	0.99999999912902900	0.99999999907392200	0.00000000005510703	Euler	
X_{13}	0.99999999912901400	0.99999999907389500	0.00000000005511902	Euler	
X_{14}	-0.99999999912902900	-0.99999999907392200	0.00000000005510703	Euler	
X_{15}	-0.99999999912902500	-0.99999999907391600	0.00000000005510903	Euler	
X_{16}	0.99999999912902100	0.99999999907391100	0.00000000005511003	Euler	
X_{17}	0.99999999912902000	0.99999999907391400	0.00000000005510603	Euler	
X_{18}	0.99999999912902400	0.99999999907392000	0.00000000005510403	Euler	
X_{19}	0.99999999912902000	0.99999999907391400	0.00000000005510603	Euler	
X_{20}	0.99999999912902900	0.99999999907392200	0.00000000005510703	Euler	

	Análisis de convergencia para el método de Euler y Runge-Kutta de cuarto orden sobre el simulador Red Neuronal con Atractores Estables -RNAE-				
	Método de Euler	Método Runge Kutta 4° orden	Diferencia en valor absoluto	Método que se acerca más a la convergencia	Patrón de prueba
Iteración:	311	325	14		
X_1	0.99999999912915300	0.99999999907395500	0.00000000005519807	Euler	
X_2	0.99999999912915300	0.99999999907395500	0.00000000005519807	Euler	
X_3	0.99999999912915300	0.99999999907395500	0.00000000005519807	Euler	
X_4	0.99999999912915300	0.99999999907395500	0.00000000005519807	Euler	
X_5	0.99999999912915200	0.99999999907395500	0.00000000005519707	Euler	
X_6	-0.99999999912914100	-0.99999999907394000	0.00000000005520095	Euler	
X_7	-0.99999999912914100	-0.99999999907394000	0.00000000005520095	Euler	
X_8	0.99999999912914000	0.99999999907395500	0.00000000005518497	Euler	
X_9	0.99999999912915300	0.99999999907395500	0.00000000005519807	Euler	
X_{10}	0.99999999912915300	0.99999999907395500	0.00000000005519807	Euler	P_4
X_{11}	0.99999999912915300	0.99999999907395500	0.00000000005519807	Euler	
X_{12}	0.99999999912915300	0.99999999907395500	0.00000000005519807	Euler	
X_{13}	0.99999999912915200	0.99999999907395500	0.00000000005519707	Euler	
X_{14}	-0.99999999912914100	-0.99999999907394000	0.00000000005520095	Euler	
X_{15}	-0.99999999912914100	-0.99999999907394000	0.00000000005520095	Euler	
X_{16}	0.99999999912915300	0.99999999907395500	0.00000000005519807	Euler	
X_{17}	0.99999999912915300	0.99999999907395500	0.00000000005519807	Euler	
X_{18}	0.99999999912915300	0.99999999907395500	0.00000000005519807	Euler	
X_{19}	0.99999999912915300	0.99999999907395500	0.00000000005519807	Euler	
X_{20}	0.99999999912915300	0.99999999907395500	0.00000000005519807	Euler	
Iteración:	315	329	14		
X_1	-0.99999999912903300	-0.99999999908929500	0.00000000003973799	Euler	
X_2	0.99999999912902500	0.99999999908928500	0.00000000003973999	Euler	
X_3	0.99999999912902500	0.99999999908928500	0.00000000003973999	Euler	
X_4	-0.99999999912903300	-0.99999999908929500	0.00000000003973799	Euler	
X_5	-0.99999999912903300	-0.99999999908929500	0.00000000003973799	Euler	
X_6	-0.99999999912903300	-0.99999999908929500	0.00000000003973799	Euler	
X_7	0.99999999912902500	0.99999999908928500	0.00000000003973999	Euler	
X_8	-0.99999999912903300	-0.99999999908929500	0.00000000003973799	Euler	
X_9	-0.99999999912903300	-0.99999999908929500	0.00000000003973799	Euler	
X_{10}	-0.99999999912903300	-0.99999999908929500	0.00000000003973799	Euler	P_5
X_{11}	0.99999999912902500	0.99999999908928500	0.00000000003973999	Euler	
X_{12}	-0.99999999912903300	-0.99999999908929500	0.00000000003973799	Euler	
X_{13}	-0.99999999912903300	-0.99999999908929500	0.00000000003973799	Euler	
X_{14}	-0.99999999912903300	-0.99999999908929500	0.00000000003973799	Euler	
X_{15}	0.99999999912902500	0.99999999908928500	0.00000000003973999	Euler	
X_{16}	-0.99999999912903300	-0.99999999908929500	0.00000000003973799	Euler	
X_{17}	-0.99999999912903300	-0.99999999908929500	0.00000000003973799	Euler	
X_{18}	-0.99999999912903300	-0.99999999908929500	0.00000000003973799	Euler	
X_{19}	0.99999999912902500	0.99999999908928500	0.00000000003973999	Euler	
X_{20}	-0.99999999912903300	-0.99999999908929500	0.00000000003973799	Euler	

	Análisis de convergencia para el método de Euler y Runge-Kutta de cuarto orden sobre el simulador Red Neuronal con Atractores Estables -RNAE-				
	Método de Euler	Método Runge Kutta 4° orden	Diferencia en valor absoluto	Método que se acerca más a la convergencia	Patrón de prueba
Iteración:	318	332	14		
X_1	0.99999999912905200	0.99999999908976600	0.00000000003928602	Euler	
X_2	-0.99999999912904600	-0.99999999908976400	0.00000000003928202	Euler	
X_3	-0.99999999912904600	-0.99999999908976400	0.00000000003928202	Euler	
X_4	0.99999999912904700	0.99999999908975900	0.00000000003928802	Euler	
X_5	0.99999999912905200	0.99999999908976600	0.00000000003928602	Euler	
X_6	-0.99999999912905200	-0.99999999908976600	0.00000000003928602	Euler	
X_7	-0.99999999912905200	-0.99999999908976600	0.00000000003928602	Euler	
X_8	0.99999999912905200	0.99999999908976600	0.00000000003928602	Euler	
X_9	0.99999999912904600	0.99999999908975900	0.00000000003928702	Euler	
X_{10}	0.99999999912905200	0.99999999908976600	0.00000000003928602	Euler	P_6
X_{11}	0.99999999912905200	0.99999999908976600	0.00000000003928602	Euler	
X_{12}	0.99999999912904700	0.99999999908975900	0.00000000003928802	Euler	
X_{13}	-0.99999999912905200	-0.99999999908976600	0.00000000003928602	Euler	
X_{14}	-0.99999999912904700	-0.99999999908975900	0.00000000003928802	Euler	
X_{15}	-0.99999999912904700	-0.99999999908975900	0.00000000003928802	Euler	
X_{16}	0.99999999912904600	0.99999999908975900	0.00000000003928702	Euler	
X_{17}	-0.99999999912904600	-0.99999999908975900	0.00000000003928702	Euler	
X_{18}	-0.99999999912905200	-0.99999999908976600	0.00000000003928602	Euler	
X_{19}	-0.99999999912905200	-0.99999999908976600	0.00000000003928602	Euler	
X_{20}	0.99999999912905200	0.99999999908976600	0.00000000003928602	Euler	
Iteración:	316	330	14		
X_1	0.99999999912906900	0.99999999910122900	0.00000000002783995	Euler	
X_2	0.99999999912907900	0.99999999910123900	0.00000000002783995	Euler	
X_3	0.99999999912907900	0.99999999910123900	0.00000000002783995	Euler	
X_4	0.99999999912906900	0.99999999910122900	0.00000000002783995	Euler	
X_5	-0.99999999912907900	-0.99999999910123900	0.00000000002783995	Euler	
X_6	-0.99999999912907900	-0.99999999910123900	0.00000000002783995	Euler	
X_7	-0.99999999912907600	-0.99999999910123800	0.00000000002783795	Euler	
X_8	0.99999999912907600	0.99999999910123800	0.00000000002783795	Euler	
X_9	0.99999999912906900	0.99999999910122900	0.00000000002783995	Euler	
X_{10}	0.99999999912907600	0.99999999910123800	0.00000000002783795	Euler	P_7
X_{11}	0.99999999912907900	0.99999999910123900	0.00000000002783995	Euler	
X_{12}	0.99999999912906900	0.99999999910122900	0.00000000002783995	Euler	
X_{13}	0.99999999912907400	0.99999999910123800	0.00000000002783596	Euler	
X_{14}	-0.99999999912907900	-0.99999999910123900	0.00000000002783995	Euler	
X_{15}	-0.99999999912907600	-0.99999999910123800	0.00000000002783795	Euler	
X_{16}	-0.99999999912907700	-0.99999999910123900	0.00000000002783795	Euler	
X_{17}	0.99999999912906900	0.99999999910122900	0.00000000002783995	Euler	
X_{18}	0.99999999912907600	0.99999999910123800	0.00000000002783795	Euler	
X_{19}	0.99999999912907900	0.99999999910123900	0.00000000002783995	Euler	
X_{20}	0.99999999912906900	0.99999999910122900	0.00000000002783995	Euler	

	Análisis de convergencia para el método de Euler y Runge-Kutta de cuarto orden sobre el simulador Red Neuronal con Atractores Estables -RNAE-				
	Método de Euler	Método Runge Kutta 4° orden	Diferencia en valor absoluto	Método que se acerca más a la convergencia	Patrón de prueba
Iteración:	313	329	16		
X_1	0.99999999913235400	0.99999999911864500	0.00000000001370903	Euler	
X_2	0.99999999913235400	0.99999999911864500	0.00000000001370903	Euler	
X_3	0.99999999913236300	0.99999999911865500	0.00000000001370803	Euler	
X_4	0.99999999913236400	0.99999999911865500	0.00000000001370903	Euler	
X_5	0.99999999913236300	0.99999999911865500	0.00000000001370803	Euler	
X_6	-0.99999999913236400	-0.99999999911865500	0.00000000001370903	Euler	
X_7	-0.99999999913236400	-0.99999999911865500	0.00000000001370903	Euler	
X_8	-0.99999999913236300	-0.99999999911865400	0.00000000001370903	Euler	
X_9	0.99999999913236400	0.99999999911864500	0.00000000001371903	Euler	
X_{10}	0.99999999913236400	0.99999999911865500	0.00000000001370903	Euler	P_8
X_{11}	0.99999999913236400	0.99999999911865500	0.00000000001370903	Euler	
X_{12}	0.99999999913236400	0.99999999911865500	0.00000000001370903	Euler	
X_{13}	-0.99999999913236200	-0.99999999911864600	0.00000000001371603	Euler	
X_{14}	-0.99999999913235400	-0.99999999911864500	0.00000000001370903	Euler	
X_{15}	-0.99999999913235400	-0.99999999911864500	0.00000000001370903	Euler	
X_{16}	0.99999999913236400	0.99999999911865500	0.00000000001370903	Euler	
X_{17}	0.99999999913235400	0.99999999911864500	0.00000000001370903	Euler	
X_{18}	0.99999999913236300	0.99999999911865500	0.00000000001370803	Euler	
X_{19}	0.99999999913236300	0.99999999911865500	0.00000000001370803	Euler	
X_{20}	0.99999999913236400	0.99999999911865500	0.00000000001370903	Euler	
Iteración:	310	324	14		
X_1	0.99999999912989300	0.99999999907391700	0.00000000005597600	Euler	
X_2	0.99999999912989300	0.99999999907391700	0.00000000005597600	Euler	
X_3	0.99999999912989300	0.99999999907391700	0.00000000005597600	Euler	
X_4	0.99999999912989300	0.99999999907391700	0.00000000005597600	Euler	
X_5	0.99999999912988000	0.99999999907390000	0.00000000005598000	Euler	
X_6	-0.99999999912989300	-0.99999999907391700	0.00000000005597600	Euler	
X_7	-0.99999999912989300	-0.99999999907391700	0.00000000005597600	Euler	
X_8	0.99999999912988000	0.99999999907390000	0.00000000005598000	Euler	
X_9	0.99999999912989300	0.99999999907391700	0.00000000005597600	Euler	
X_{10}	0.99999999912989300	0.99999999907391700	0.00000000005597600	Euler	P_9
X_{11}	0.99999999912989300	0.99999999907391700	0.00000000005597600	Euler	
X_{12}	0.99999999912989300	0.99999999907391700	0.00000000005597600	Euler	
X_{13}	-0.99999999912989300	-0.99999999907391700	0.00000000005597600	Euler	
X_{14}	-0.99999999912989300	-0.99999999907391700	0.00000000005597600	Euler	
X_{15}	-0.99999999912989300	-0.99999999907391700	0.00000000005597600	Euler	
X_{16}	0.99999999912988000	0.99999999907390000	0.00000000005598000	Euler	
X_{17}	-0.99999999912989300	-0.99999999907391700	0.00000000005597600	Euler	
X_{18}	-0.99999999912989300	-0.99999999907391700	0.00000000005597600	Euler	
X_{19}	-0.99999999912989300	-0.99999999907391700	0.00000000005597600	Euler	
X_{20}	0.99999999912989300	0.99999999907391700	0.00000000005597600	Euler	

	Análisis de convergencia para el método de Euler y Runge-Kutta de cuarto orden sobre el simulador Red Neuronal con Atractores Estables -RNAE-				
	Método de Euler	Método Runge Kutta 4° orden	Diferencia en valor absoluto	Método que se acerca más a la convergencia	Patrón de prueba
Iteración:	310	324	14		
X_1	0.99999999912987600	0.99999999907390000	0.00000000005597600	Euler	
X_2	0.99999999912987600	0.99999999907390000	0.00000000005597600	Euler	
X_3	0.99999999912987600	0.99999999907390000	0.00000000005597600	Euler	
X_4	0.99999999912988900	0.99999999907391700	0.00000000005597200	Euler	
X_5	-0.99999999912988900	-0.99999999907391700	0.00000000005597200	Euler	
X_6	-0.99999999912988900	-0.99999999907391700	0.00000000005597200	Euler	
X_7	-0.99999999912988900	-0.99999999907391700	0.00000000005597200	Euler	
X_8	0.99999999912988900	0.99999999907391700	0.00000000005597200	Euler	
X_9	-0.99999999912988900	-0.99999999907391700	0.00000000005597200	Euler	
X_{10}	-0.99999999912988900	-0.99999999907391700	0.00000000005597200	Euler	P_{10}
X_{11}	-0.99999999912988900	-0.99999999907391700	0.00000000005597200	Euler	
X_{12}	0.99999999912988900	0.99999999907391700	0.00000000005597200	Euler	
X_{13}	-0.99999999912988900	-0.99999999907391700	0.00000000005597200	Euler	
X_{14}	-0.99999999912988900	-0.99999999907391700	0.00000000005597200	Euler	
X_{15}	-0.99999999912988900	-0.99999999907391700	0.00000000005597200	Euler	
X_{16}	0.99999999912988900	0.99999999907391700	0.00000000005597200	Euler	
X_{17}	-0.99999999912988900	-0.99999999907391700	0.00000000005597200	Euler	
X_{18}	-0.99999999912988900	-0.99999999907391700	0.00000000005597200	Euler	
X_{19}	-0.99999999912988900	-0.99999999907391700	0.00000000005597200	Euler	
X_{20}	0.99999999912988900	0.99999999907391700	0.00000000005597200	Euler	
Iteración:	313	326	13		
X_1	0.99999999913139200	0.99999999907714900	0.00000000005424305	Euler	
X_2	0.99999999913129000	0.99999999907712500	0.00000000005416501	Euler	
X_3	0.99999999913129000	0.99999999907712500	0.00000000005416501	Euler	
X_4	0.99999999913139200	0.99999999907714900	0.00000000005424305	Euler	
X_5	0.99999999913139200	0.99999999907714900	0.00000000005424305	Euler	
X_6	-0.99999999913139200	-0.99999999907714900	0.00000000005424305	Euler	
X_7	-0.99999999913139200	-0.99999999907714900	0.00000000005424305	Euler	
X_8	0.99999999913139200	0.99999999907714900	0.00000000005424305	Euler	
X_9	0.99999999913139200	0.99999999907714900	0.00000000005424305	Euler	
X_{10}	-0.99999999913039200	-0.99999999907713900	0.00000000005325296	Euler	P_{11}
X_{11}	-0.99999999913039200	-0.99999999907713900	0.00000000005325296	Euler	
X_{12}	0.99999999913139200	0.99999999907714900	0.00000000005424305	Euler	
X_{13}	0.99999999913129000	0.99999999907713900	0.00000000005415102	Euler	
X_{14}	-0.99999999913139200	-0.99999999907714900	0.00000000005424305	Euler	
X_{15}	-0.99999999913139200	-0.99999999907714900	0.00000000005424305	Euler	
X_{16}	0.99999999913139200	0.99999999907714900	0.00000000005424305	Euler	
X_{17}	0.99999999913129000	0.99999999907713900	0.00000000005415102	Euler	
X_{18}	0.99999999913128100	0.99999999907712500	0.00000000005415601	Euler	
X_{19}	0.99999999913128100	0.99999999907712500	0.00000000005415601	Euler	
X_{20}	0.99999999913139200	0.99999999907714900	0.00000000005424305	Euler	

Apéndice B

Código fuente del Simulador para la Red Neuronal con Atractores Estables desarrollado en NetBeans Integrated Development Enviroment 8.0.2, cuyo desempeño como memoria asociativa en el reconocimiento de patrones se muestra en el apartado 3.4.3.

```
/* @author Aide Guadalupe Villafranco Ramírez */

/* Programa que simula el comportamiento de un modelo de Memoria Asociativa
 * con el uso de la teoría de Sistemas Dinámicos y el concepto de Atractores Estables.
 * Donde cada memoria representa un punto fijo estable del Sistema de Ecuaciones Diferenciales
 * y se utiliza el método numérico de Euler para su solución. */

/* Zona de importación de librerias.*/

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import javax.swing.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

/* Clase principal del programa que agrupa todos los elementos en la ventana principal
 * y crea una instancia de sí misma. */

public class JavaAtractoresEstables {
    Button botonRecuperarPatron;
    CanvasAtractoresEstables canvas;
    TextField textoEpsilon;
    TextField textoDeltaT;
    TextField textoIteracionesMaximas;
    TextField textoCambioMaximo;
    Frame ventana;

    public static void main ( String args[] ){
        /* Instancia de la clase principal JavaAtractoresEstables. */
        new JavaAtractoresEstables();
    }

    /* Constructor de la clase principal JavaAtractoresEstables. */
    public JavaAtractoresEstables () {

    /* Instanciación  y asignación de características de la ventana principal. */
    ventana = new Frame( "Red Neuronal con Atractores Estables" );
    ventana.setLayout( new BorderLayout() );
    ventana.setBackground( Color.gray );
    ventana.setSize( 700, 450 );
```

```
/* Instancia del JPanel que agrupa todos los componentes de la ventana principal. */
JPanel panel = new JPanel();
panel.setLayout( null );

/* Instancia de la clase CanvasAtractores encargada de todos los pormenores
 * ocurridos sobre la cuadrícula de patrones. */
canvas = new CanvasAtractoresEstables();
canvas.ancho = canvas.alto = 200;  /* Dimensiones de la cuadrícula de patrones. */
canvas.setBounds( 30, 30, canvas.ancho, canvas.alto );

/* Instanciación y asignación de las características que acompañan a los objetos que permiten
 * la manipulación del número de filas y columnas en la cuadrícula de patrones. */
Label etiquetaFilas = new Label( "Filas" );
etiquetaFilas.setBounds( 250, 50, 50, 20 );
etiquetaFilas.setAlignment( Label.CENTER );

List listaFilas = new List();
for ( int i = 1; i <= 20; i++ ){ listaFilas.add( "" +i ); } /* Adición de hasta 20 filas. */
listaFilas.setMultipleMode( false );
listaFilas.select( 0 );

canvas.filas = listaFilas.getSelectedIndex() + 1; /* Número de filas de la cuadrícula de patrones.*/
listaFilas.setBounds( 250, 70, 50, 90 );

Label etiquetaColumnas = new Label( "Columnas" );
etiquetaColumnas.setBounds( 335, 50, 60, 20 );
etiquetaColumnas.setAlignment( Label.CENTER );

List listaColumnas = new List();
for (int i = 1; i <= 20; i++){ listaColumnas.add( "" +i ); } /* Adición de hasta 20 columnas. */
listaColumnas.setMultipleMode( false );
listaColumnas.select( 0 );

/* Número de columnas en la cuadrícula de patrones. */
canvas.columnas = listaColumnas.getSelectedIndex() + 1;
listaColumnas.setBounds( 340, 70, 50, 90 );

/* Ancho de cada uno de los cuadros de la cuadrícula de patrones. */
canvas.deltaX = canvas.ancho / canvas.columnas;
/* Alto de cada uno de los cuadros de la cuadrícula de patrones. */
canvas.deltaY = canvas.alto / canvas.filas;

 /* Punto de inicio (x,y) para dibujar la cuadrícula de patrones
  * de tal suerte que aparezca lo mejor distribuido posible. */
canvas.x = ( canvas.ancho % canvas.columnas ) / 2;
canvas.y = ( canvas.alto % canvas.filas ) / 2;

/* Instancia del arreglo unidimensional que contiene los valores en el eje X
 * de las líneas que separan cada columna en la cuadrícula de patrones. */
canvas.ejeX = new int[ canvas.columnas + 1 ];
canvas.ejeX[ 0 ] = 0;
canvas.ejeX[ canvas.columnas ] = canvas.ancho;

/* Instancia del arreglo unidimensional que contiene los valores en el eje Y
 * de las líneas que separan cada fila en el cuadrícula de patrones. */
canvas.ejeY = new int[ canvas.filas + 1 ];
```

```
canvas.ejeY[ 0 ] = 0;
canvas.ejeY[ canvas.filas ] = canvas.alto;

/* Instancia del arreglo bidimensional que contiene
 * los valores de la cuadrícula de patrones (encendido/apagado). */
canvas.cuadricula = new boolean[ canvas.columnas ][ canvas.filas ];

/* Instancia del arreglo bidimensional que contiene los valores
 * de cada una de las neuronas asociados a la cuadrícula de patrones. */
canvas.neuronas = new double[ canvas.columnas ][ canvas.filas ];

/* Inicialización del arreglo cuadrícula y neuronas. */
for ( int i = 0; i < canvas.columnas; i++ )
   for ( int j = 0; j  < canvas.filas; j++ ) {
      canvas.cuadricula[ i ][ j ] = false;
      canvas.neuronas[ i ][ j ] = -1.0;
   }

/* Instanciación y asignación de las características que acompañan a los
 * objetos que permiten el entrenamiento de la red. */
Label etiquetaFaseEntrenamiento = new Label ( "Fase de entrenamiento" );
etiquetaFaseEntrenamiento.setBounds( 30, 270, 150, 20 );

Button botonBorrar = new Button( "Borrar Patrón" );
botonBorrar.setBounds( 30, 295, 100, 30 );
botonBorrar.addActionListener( new BorrarPatronAtractoresEstables( canvas ) );

Button botonGuardar = new Button( "Guardar Patrón" );
botonGuardar.setBounds( 140, 295, 100, 30 );

Label totalMemorias = new Label( "Memorias totales: " );
totalMemorias.setBounds( 250, 305, 100, 20 );

TextField textoTotalMemorias = new TextField( );
textoTotalMemorias.setBounds( 360, 305, 60, 20 );
textoTotalMemorias.setBackground( Color.white );
textoTotalMemorias.setEditable( false );
textoTotalMemorias.setText( Integer.toString( canvas.totalMemorias ) );

/* Instanciación y asignación de las características que acompañan a los objetos que permiten
 * la manipulación de los parámetros de entrada de la red neuronal, tales como:
 * el valor de epsilón y delta t. */
Label etiquetaDatosRed = new Label( "Datos de la Red Neuronal" );
etiquetaDatosRed.setBounds( 250, 30, 150, 20 );

Label etiquetaEpsilon = new Label( "Épsilon:" );
Label etiquetaDeltaT = new Label( "Delta t:" );
etiquetaEpsilon.setBounds( 250, 170, 50, 20 );
etiquetaDeltaT.setBounds( 250, 200, 50, 20 );

textoEpsilon = new TextField( );
textoDeltaT = new TextField( );
textoEpsilon.setBounds( 310, 170, 60, 20 );
textoDeltaT.setBounds( 310, 200, 60, 20 );
textoEpsilon.setBackground( Color.white );
textoDeltaT.setBackground( Color.white );
```

```
    textoEpsilon.setEditable( true );
    textoDeltaT.setEditable( true );
    textoEpsilon.addActionListener(  new EventosEntradaAtractoresEstables( textoEpsilon, ventana )
);
    textoDeltaT.addActionListener( new EventosEntradaAtractoresEstables( textoDeltaT, ventana ) );
    textoEpsilon.setText( "" );
    textoDeltaT.setText( "" );

    /* Instancia del botón que dibuja la cuadrícula sobre el objeto de la clase CanvasAtractores. */
    Button botonCuadricular = new Button( "Cuadricular" );
    botonCuadricular.setBounds( 268, 230, 100, 30 );

    /* Instanciación y asignación de las características acompañan a los objetos que permiten
     * la manipulación de los prámetros de entrada para un cálculo iterativo, tales como:
     * número máximo de iteraciones y cambio máximo entre los valores. */
    Label etiquetaCalculoIterativo = new Label( "Datos del cálculo iterativo" );
    etiquetaCalculoIterativo.setBounds( 420, 30, 150, 20 );

    Label etiquetaIteracionesMaximas = new Label( "Iteraciones máximas:" );
    etiquetaIteracionesMaximas.setBounds( 420, 60, 120, 20 );

    Label etiquetaCambioMaximo = new Label( "Cambio máximo:" );
    etiquetaCambioMaximo.setBounds( 420, 90, 120, 20 );

    textoIteracionesMaximas = new TextField( );
    textoIteracionesMaximas.setBounds( 550, 60, 60, 20 );
    textoIteracionesMaximas.setBackground( Color.white );
    textoIteracionesMaximas.setEditable( true );
    textoIteracionesMaximas.setText( "" );
    /* Registra un receptor de eventos de tipo acción que evalúa los valores introducidos en
     * el campo de texto y rechaza aquellos que no sean numeros enteros. */
    textoIteracionesMaximas.addActionListener(    new    EventosEntradaEnterosAtractoresEstables(
textoIteracionesMaximas, ventana ) );

    textoCambioMaximo = new TextField( );
    textoCambioMaximo.setBounds( 550, 90, 60, 20 );
    textoCambioMaximo.setBackground( Color.white );
    textoCambioMaximo.setEditable( true );
    textoCambioMaximo.setText( "" );
    textoCambioMaximo.addActionListener(          new          EventosEntradaAtractoresEstables(
textoCambioMaximo, ventana ) );

    /* Instanciación y asignación de las características acompañan a los objetos que permiten
     * evaluar el comportamiento de la red neuronal. */
    Label etiquetaFasePrueba = new Label( "Fase de prueba" );
    etiquetaFasePrueba.setBounds( 30, 340, 150, 20 );

    botonRecuperarPatron = new Button( "Recuperar Patrón" );
    botonRecuperarPatron.setBounds( 30, 365, 120, 30 );
    botonRecuperarPatron.setEnabled( false );

    /* Adición de los receptores de eventos que manipulan filas y las columnas
     * en la cuadrícula de patrones. */
    listaFilas.addActionListener(    new    CuadricularAtractoresEstables(    canvas,    listaFilas,
listaColumnas, botonGuardar, textoTotalMemorias ));
```

```
    listaColumnas.addActionListener( new CuadricularAtractoresEstables( canvas, listaFilas, listaColumnas, botonGuardar, textoTotalMemorias ));
    botonCuadricular.addActionListener( new CuadricularAtractoresEstables( canvas, listaFilas, listaColumnas, botonGuardar, textoTotalMemorias ));

    /* Adición del receptor de eventos que almacena y manipula las memorias del modelo. */
    botonGuardar.addActionListener( new AlmacenarMemoriasAtractoresEstables( canvas, botonRecuperarPatron, textoTotalMemorias, ventana ) );
    botonGuardar.setEnabled( false);

    /* Adición del receptor de eventos que evalúa el comportamiento de la red
     * a través de la convergencia a un punto fijo estable. */
    botonRecuperarPatron.addActionListener( new FasePruebaAtractoresEstables( canvas, textoEpsilon, textoDeltaT, textoIteracionesMaximas, textoCambioMaximo, ventana ) );

    /* Adición del receptor de eventos que manipula el encendido/apagado
     * sobre la cuadrícula de patrones. */
    canvas.addMouseListener( new ActivarDesactivarAtractoresEstables( canvas ) );
    canvas.setEnabled( false );

    /* Adición al JPanel de todos los componentes antes definidos. */
    panel.add( canvas );
    panel.add( etiquetaFilas );
    panel.add( listaFilas );
    panel.add( etiquetaColumnas );
    panel.add( listaColumnas );
    panel.add( etiquetaFaseEntrenamiento );
    panel.add( botonBorrar );
    panel.add( botonGuardar );
    panel.add( totalMemorias );
    panel.add( textoTotalMemorias );
    panel.add( etiquetaDatosRed );
    panel.add( etiquetaEpsilon );
    panel.add( etiquetaDeltaT );
    panel.add( textoEpsilon );
    panel.add( textoDeltaT );
    panel.add( botonCuadricular );
    panel.add( etiquetaCalculoIterativo );
    panel.add( etiquetaIteracionesMaximas );
    panel.add( etiquetaCambioMaximo );
    panel.add( textoIteracionesMaximas );
    panel.add( textoCambioMaximo );
    panel.add( etiquetaFasePrueba );
    panel.add( botonRecuperarPatron );

    /* Adición del JPanel a la ventana principal y establecimiento de sus características. */
    ventana.add( panel, "Center" );
    ventana.setResizable( false );
    ventana.setLocationRelativeTo( null );
    ventana.setVisible( true );

    /* Adición del receptor de eventos de ventana (cerrar). */
    ventana.addWindowListener( new ConclusionAtractoresEstables() );

    }
}
```

```
 /* Clase que implementa un receptor de eventos de tipo acción para extraer
  * valores de tipo double sobre un campo de texto enviado como argumento.*/
class EventosEntradaAtractoresEstables implements ActionListener {
   TextField texto;
   Frame ventanaPrincipal;

   /* Constructor de la clase. */
   EventosEntradaAtractoresEstables( TextField campoTexto, Frame ventana ){
      texto = campoTexto;
      ventanaPrincipal = ventana;
   }

   /* Método que ejecuta la acción de extraer de un campo de texto un valor de tipo double,
    * que en caso de encontrar null o su equivalente lanza una advertencia. */
   public void actionPerformed ( ActionEvent evt ) {
      double valor;
      /* Sentencia try/catch para el método Double.valueOf( string )
       * que lanza una excepción de tipo NumberFormatException. */
      try {
         valor = Double.valueOf( texto.getText() );
         /* Compara si el valor contenido en el campo de texto es negativo. */
         if ( valor < 0.0 ) {
            /* Sentencia que lanza a la ventana un diálogo no modal de tipo advertencia. */
            JOptionPane.showMessageDialog( ventanaPrincipal,
               "Imposible usar la entrada escrita, es necesario escribir un número entero o decimal
positivo",
               "Red Neuronal",
               JOptionPane.WARNING_MESSAGE );
            texto.setText( "" );
         }
         texto.setText( Double.toString( valor ) );
      } catch ( NumberFormatException exepcion ) {
         /* Sentencia que lanza a la ventana un diálogo no modal de tipo advertencia. */
         JOptionPane.showMessageDialog( ventanaPrincipal,
               "Imposible usar la entrada escrita, es necesario escribir un número entero o decimal
positivo",
               "Red Neuronal",
               JOptionPane.WARNING_MESSAGE );
         texto.setText( "" );
      }
   }
}

/* Clase que implementa un receptor de eventos de tipo acción para extraer
 * valores de tipo entero sobre un campo de texto enviado como argumento. */
class EventosEntradaEnterosAtractoresEstables implements ActionListener {
   TextField texto;
   Frame ventanaPrincipal;

   /* Constructor de la clase. */
   EventosEntradaEnterosAtractoresEstables( TextField campoTexto, Frame ventana ){
      texto = campoTexto;
      ventanaPrincipal = ventana;
   }
```

```
   /* Método que ejecuta la acción de extraer de un campo de texto un valor de tipo int,
    * que en caso de encontrar un null o su equivalente lanza una advertencia. */
   public void actionPerformed ( ActionEvent evt ) {
      int valor;
      /* Sentencia try/catch para el método Integer.valueOf( string )
       * que lanza una excepción de tipo NumberFormatException. */
      try {
         valor = Integer.valueOf( texto.getText() );
         /* Compara si el valor contenido en el campor de texto es negativo. */
         if ( valor < 0 ) {
            /* Sentencia que lanza a la ventana un diálogo no modal de tipo advertencia. */
            JOptionPane.showMessageDialog( ventanaPrincipal,
               "Imposible usar la entrada escrita, es necesario escribir un número entero o decimal
positivo",
               "Red Neuronal",
               JOptionPane.WARNING_MESSAGE );
            texto.setText( "" );
         }
         texto.setText( Integer.toString( valor ) );
      } catch ( NumberFormatException exepcion ) {
         /* Sentencia que lanza a la ventana un diálogo no modal de tipo advertencia. */
         JOptionPane.showMessageDialog( ventanaPrincipal,
               "Imposible usar la entrada escrita, es necesario escribir un número entero o decimal
positivo",
               "Red Neuronal",
               JOptionPane.WARNING_MESSAGE );
         texto.setText( "" );
      }
   }
}

/* Clase que implementa un receptor de eventos de tipo acción para
 * restablecer los valores de las variables de un objeto de tipo CanvasAtractores
 * de tal suerte que la cuadrícula sea redibujada por el método paint() de ésa clase. */
class CuadricularAtractoresEstables implements ActionListener {
   CanvasAtractoresEstables canvasRef;
   List listaFilas, listaColumnas;
   Button guardar;
   TextField textoTotalMemorias;

   /* Constructor de la clase */
   CuadricularAtractoresEstables ( CanvasAtractoresEstables canvas, List ListaFilas, List
ListaColumnas, Button botonGuardar, TextField totalMemorias ){
      canvasRef = canvas;
      listaFilas = ListaFilas;
      listaColumnas = ListaColumnas;
      guardar = botonGuardar;
      textoTotalMemorias = totalMemorias;
   }

   /* Método que restablece los valores de todas las variables asociadas
    * a la cuadrícula de patrones. */
   public void actionPerformed ( ActionEvent evt ){

      canvasRef.filas = listaFilas.getSelectedIndex() + 1; /* Número de filas. */
      canvasRef.columnas = listaColumnas.getSelectedIndex() + 1; /* Número de columnas. */
```

```
        canvasRef.deltaX = canvasRef.ancho / canvasRef.columnas; /* Ancho de columnas. */
        canvasRef.deltaY = canvasRef.alto / canvasRef.filas; /* Ancho de filas. */

        /* Valor en x del punto de inicio para dibujar las columnas. */
        canvasRef.x = ( canvasRef.ancho % canvasRef.columnas ) / 2;
        /* Valor en y del punto de inicio para dibujar las filas. */
        canvasRef.y = ( canvasRef.alto % canvasRef.filas ) / 2;

        /* Instancia de los arreglos unidimendionales que contienen los valores de los ejes x,y
         * respectivamente, para el dibujado de las líneas de división de la cuadrícula de patrones. */
        canvasRef.ejeX = new int[ canvasRef.columnas + 1 ];
        canvasRef.ejeY = new int[ canvasRef.filas + 1 ];
        canvasRef.ejeX[ 0 ] = 0;
        canvasRef.ejeY[ 0 ] = 0;
        canvasRef.ejeX[ canvasRef.columnas ] = canvasRef.ancho;
        canvasRef.ejeY[ canvasRef.filas ] = canvasRef.alto;

        /* Instancia de los arreglo bidimensionales que contienen los valores
         * del patrón de cuadrícula (boolean) y de las neuronas asociadas a dicho patrón (double). */
        canvasRef.cuadricula = new boolean[ canvasRef.columnas ][ canvasRef.filas ];
        canvasRef.neuronas = new double[ canvasRef.columnas ][ canvasRef.filas ];

        for( int i = 0; i < canvasRef.columnas; i++ )
           for ( int j = 0; j < canvasRef.filas; j++ ) {
              canvasRef.neuronas[ i ][ j ] = -1.0;
              canvasRef.cuadricula[ i ][ j ] = false;
           }
        canvasRef.totalMemorias = 0; /* Total de memorias almacenadas en la red. */

        /* Sentencia que coloca en el cuadro de texto textoTotalMemorias el valor de totalMemorias. */
        textoTotalMemorias.setText( Integer.toString( canvasRef.totalMemorias ) );
        canvasRef.setEnabled( true ); /* Se activa la cuadrícula de patrones para su uso. */
       /* Se activa el botón que permite guardar los patrones dibujados sobre la cuadrícula. */
        guardar.setEnabled( true );
        canvasRef.repaint(); /* Repintado. */
    }
}

/* Clase encargada de almacenar los patrones dibujados sobre la
 * cuadrícula en un arreglo bidimensional. */
class AlmacenarMemoriasAtractoresEstables implements ActionListener{
    CanvasAtractoresEstables canvasRef;
    int[][] arregloTemporal;
    Button recuperarPatron;
    TextField totalMemorias;
    Frame ventanaPrincipal;

    /* Contructor de la clase. */
    AlmacenarMemoriasAtractoresEstables ( CanvasAtractoresEstables canvas, Button
botonRecuperarPatron, TextField textoTotalMemorias, Frame ventana ){
        canvasRef = canvas;
        recuperarPatron = botonRecuperarPatron;
        totalMemorias = textoTotalMemorias;
        ventanaPrincipal = ventana;
    }
```

```
   /* Método que almacena los patrones dibujados sobre la cuadrícula
    * en el arreglo bidimensional matrizMemorias. */
   public void actionPerformed( ActionEvent evt ){

      int g, h, sumaDiferencias;

      canvasRef.totalMemorias++; /* Incrementa el contador totalMemorias. */
     /* Variable bandera que indica si el nuevo patrón de entrada no está siendo repetido. */
      sumaDiferencias = 0;

      /* Instancia del arreglo que almacena de forma temporal
       * los valores contenidos en la matriz de memorias
       * para permitir la propiedad dinámica de la misma. */
      arregloTemporal = new int[ canvasRef.columnas * canvasRef.filas ][ canvasRef.totalMemorias
- 1 ];

      g = 0;

      /* Copiado de la matriz de memorias al arreglo temporal. */
      for ( int j = 0; j < canvasRef.totalMemorias - 1; j++ )
         for ( int i = 0; i < canvasRef.columnas * canvasRef.filas; i++ )
            arregloTemporal[ i ][ j ] = canvasRef.matrizMemorias[ i ][ j ];

      /* Re-instanciación de la matriz de memorias */
      canvasRef.matrizMemorias = new int[ canvasRef.columnas * canvasRef.filas ][
canvasRef.totalMemorias ];

      /* Bloque de sentencias for que compara el nuevo patrón de memoria con los
       * almacenados anteriormente para evitar su repetición. */
      for ( int k = 0; k < canvasRef.totalMemorias - 1; k++ ) {
         for ( int j = 0; j < canvasRef.filas; j++ )
            for ( int i = 0; i < canvasRef.columnas; i++ ) {
               if ( canvasRef.cuadricula[ i ][ j ] )
                  sumaDiferencias = sumaDiferencias + Math.abs( 1 - arregloTemporal[ g ][ k ] );
               else
                  sumaDiferencias = sumaDiferencias + Math.abs( 0 - arregloTemporal[ g ][ k ] );
               g++;
            }
            /* Sentencia que activa la bandera sumaDiferencias
             * para indicar que un patrón está siendo repetido. */
            if ( sumaDiferencias == 0 ) {
             sumaDiferencias = -1;
             break;
           }
           sumaDiferencias = 0;
           g = 0;
      }

      /* Copia de los valores del arreglo temporal a la matriz de memorias
       * una vez que ha sido re-dimensionada. */
      for ( int j = 0; j < canvasRef.totalMemorias - 1; j++ )
         for ( int i = 0; i < canvasRef.columnas * canvasRef.filas; i++ )
            canvasRef.matrizMemorias[ i ][ j ] = arregloTemporal[ i ][ j ];
```

```
      /* Inicialización de las variables que permiten acceder a arreglos de dimensiones diferentes
       * a través un mismo ciclo. */
      g = h = 0;

      /* Sentencia if/else que evalúa la bandera sumaDiferencias. */
      if ( sumaDiferencias == -1 ) { /* Bloque para el caso patrón repetido. */
         canvasRef.totalMemorias--; /* Decremento del total de memorias. */
         /* Sentencia que envía a la pantalla un diálogo no modal para advertir que el patrón
          * está siendo repetido. */
         JOptionPane.showMessageDialog( ventanaPrincipal,
               "El patron de memoria está siendo repetido",
               "Red Neuronal con Atractores Estables",
               JOptionPane.WARNING_MESSAGE );
      }

      else { /* Bloque para el caso patrón no repetido. */
         /* Alamecenamiento del nuevo patrón de memoria en la matriz de memorias. */
         for ( int i = 0; i < canvasRef.columnas * canvasRef.filas; i++ ) {
            for ( int j = 0; j < canvasRef.totalMemorias - 1; j++ )
               canvasRef.matrizMemorias[ i ][ j ] = arregloTemporal[ i ][ j ];

            if ( canvasRef.cuadricula[ g ][ h ] )
               canvasRef.matrizMemorias[ i ][ canvasRef.totalMemorias - 1] = 1;
            else canvasRef.matrizMemorias [ i ][ canvasRef.totalMemorias - 1] = 0;

            g++;
            if ( g == canvasRef.columnas ) {    g = 0;    h++;     }
         }
      }

      /* Bloque de ciclos for para limpiar el patrón de cuadrícula. */
      for( int i = 0; i < canvasRef.columnas; i++ )
         for ( int j = 0; j < canvasRef.filas; j++ )
            canvasRef.cuadricula[ i ][ j ] = false;

      /* Sentencia que muestra en un cuadro de texto el total de memorias almacenadas. */
      totalMemorias.setText( Integer.toString( canvasRef.totalMemorias ) );
      recuperarPatron.setEnabled( true ); /* Habilita el botón recuperarPatrón. */
      canvasRef.repaint();

   }
}

/* Clase encargada de probar el funcionamiento de la red neuronal
 * sobre un patrón cualquiera dibujado en la cuadrícula. */
class FasePruebaAtractoresEstables implements ActionListener {
   CanvasAtractoresEstables canvasRef;
   TextField textoCambioMaximo, textoIteracionesMaximas, textoEpsilon, textoDeltaT;
   Frame ventanaPrincipal;

   /* Constructor de la clase. */
   FasePruebaAtractoresEstables ( CanvasAtractoresEstables canvas, TextField epsilonTexto,
TextField deltatTexto, TextField iteracionesMaximasTexto, TextField cambioMaximoTexto, Frame
ventana ){
      canvasRef = canvas;
```

```
      textoEpsilon = epsilonTexto;
      textoDeltaT = deltatTexto;
      textoIteracionesMaximas = iteracionesMaximasTexto;
      textoCambioMaximo = cambioMaximoTexto;
      ventanaPrincipal = ventana;
   }

   /* Método encargado de diseñar la red neuronal siguiendo el modelo de memoria y
    * recuperar a partir de un patrón cualquiera un patrón almacenado como memoria
    * haciendo uso del método numérico de Euler para la solución del sistema */
   public void actionPerformed ( ActionEvent evt ) {
      double fi, epsilon, deltaT, cambioMaximo, suma, neuronaNueva, sumaErrores, multiplicacion;
      double[] productoImagen;
      int iteracionesMaximas, g, h, contador;

      /* Sentencia try/catch para obtener a partir de cuadros de texto
       * valores de tipo integer y double. */
      try {
         /* Diferencia entre iteraciones para detener el método numérico. */
         cambioMaximo = Double.valueOf( textoCambioMaximo.getText() );
         /* Número máximo de iteraciones */
         iteracionesMaximas = Integer.valueOf( textoIteracionesMaximas.getText() );
         /* Valor umbral del modelo de red neuronal que delimita la zona de transición. */
         epsilon = Double.valueOf( textoEpsilon.getText() );
         /* Fracción de tiempo establecida entre cada iteración del método numérico. */
         deltaT = Double.valueOf( textoDeltaT.getText() );

         /* Bloque if/else para evitar valores negativos en los cuadros de texto. */
         if ( cambioMaximo < 0.0 || iteracionesMaximas < 0 || epsilon < 0.0 || deltaT < 0.0 ) {
            /* Bloque para el caso de valores negativos. */
            JOptionPane.showMessageDialog( ventanaPrincipal,
               "Imposible usar la entrada escrita, es necesario escribir un número entero o decimal
positivo",
               "Red Neuronal con Atractores Estables",
               JOptionPane.WARNING_MESSAGE );
            if ( cambioMaximo < 0.0 ) textoCambioMaximo.setText( "" );
            else if ( iteracionesMaximas < 0 ) textoIteracionesMaximas.setText( "" );
            else if ( epsilon < 0.0 ) textoEpsilon.setText( "" );
            else textoDeltaT.setText( "" );
         }

         else { /* Bloque para el caso de valores positivos. */

            canvasRef.valorUmbral = epsilon; /* Valor que activa o desactiva las neuronas. */

            /* Inicialización del arreglo de neuronas que sirve como entrada
             * al modelo de red neuronal. */
            for( int i = 0; i < canvasRef.columnas; i++ )
               for ( int j = 0; j < canvasRef.filas; j++ ) {
                  if ( canvasRef.cuadricula [ i ][ j ] )
                     canvasRef.neuronas[ i ][ j ] = 1.0;
                  else canvasRef.neuronas[ i ] [ j ] = -1.0;
               }
```

```
            /* Instancia del arreglo unidimensional que almacena
             * los valores de los productos imagen.*/
            productoImagen = new double[ canvasRef.totalMemorias ];
            for ( int i = 0; i < canvasRef.totalMemorias; i++ )
               productoImagen[ i ] = 0.0;

            /* Contador para terminar con el método numérico
             * si es mayor al máximo número de iteraciones establecido. */
            contador = 0;

            /* suma: variable que almacena la sumatoria del modelo de memoria. */
            /* sumaErrores: variable que almacena la diferencia
             * entre cada iteración del método numérico y permite su terminación
             * si es menor al cambio máximo establecido. */
            suma = sumaErrores = 0.0;

            /* Ciclo do/while que obtiene el modelo de red neuronal
             * según los patrones de memoria almacenados
             * y aplica el método numérico de Euler. */
            do {

               /* Variables que permiten acceder a arreglos de dimensiones diferentes
                * a partir de un mismo ciclo. */
               g = h = 0;

               /* Variable que almacena de forma temporal el producto imagen
                * de cada patrón de memoria almacenado. */
               multiplicacion = 1.0;

               /* Bloque de ciclos for para el cálculo del producto imagen correspondiente
                * a cada memoria almacenada en la matriz de memorias. */
               for ( int k = 0; k < canvasRef.totalMemorias; k++ ){
                  for ( int j = 0; j < canvasRef.filas; j ++ ){
                     for ( int i = 0; i < canvasRef.columnas; i++ ){
                        if ( canvasRef.neuronas[ i ][ j ] >= epsilon )
                           fi= 1.0;
                        else if ( canvasRef.neuronas[ i ][ j ] <= -epsilon )
                           fi= 0.0;
                        else fi= ( canvasRef.neuronas[ i ][ j ] + epsilon ) / ( 2 * epsilon );

                        multiplicacion = multiplicacion * ( canvasRef.matrizMemorias[ g ][ k ] * fi + ( 1 -
canvasRef.matrizMemorias[ g ][ k ] )* ( 1 - fi ) );
                        g++;
                     }
                  }
                  productoImagen[ k ] = multiplicacion;
                  multiplicacion = 1.0;
                  g = 0;
               }

               g = 0;
               sumaErrores = 0.0;
```

```
                /* Bloque de ciclos for que por un lado genera el modelo de red neuronal
                 * siguiendo el modelo de memoria y los patrones almacenados,
                 * y por el otro aplica el método numérico de Euler a la red neuronal obtenida. */
                for ( int j = 0; j < canvasRef.filas; j++ )
                    for ( int i = 0; i < canvasRef.columnas; i++ ) {

                        for ( int k = 0; k < canvasRef.totalMemorias; k++ )
                            suma = suma + ( 2 * canvasRef.matrizMemorias[ g ][ k ] - 1 ) * productoImagen[
k ];

                        g++;

                        neuronaNueva = canvasRef.neuronas[ i ][ j ] + ( -canvasRef.neuronas[ i ][ j ] +
suma )* deltaT;

                        sumaErrores = sumaErrores + Math.abs( canvasRef.neuronas[ i ][ j ] -
neuronaNueva );
                        canvasRef.neuronas[ i ][ j ] = neuronaNueva;

                        suma = 0.0;
                    }

                contador++;

            } while ( sumaErrores / ( canvasRef.columnas * canvasRef.filas ) > cambioMaximo &&
contador < iteracionesMaximas );

            /* Bloque de ciclos for que limpia la cuadrícula. */
            for( int i = 0; i < canvasRef.columnas; i++ )
                for ( int j = 0; j < canvasRef.filas; j++ )
                    canvasRef.cuadricula[ i ][ j ] = false;

            /* Se activa la bandera que permite el repintado de la cuadrícula
             * según los nuevos valores obtenidos para cada neurona,
             * sin olvidar que cada cuadro representa una neurona del modelo. */
            canvasRef.pintarFocos = true;
            canvasRef.repaint();
        }

    } catch ( NumberFormatException exepcion ) {
        /* Bloque que captura las excepciones generadas por los cuadros de texto,
         * en caso de que el valor introducido no corresponda a un valor numérico. */
        textoEpsilon.dispatchEvent( evt );
        textoDeltaT.dispatchEvent( evt );
        textoIteracionesMaximas.dispatchEvent( evt );
        textoCambioMaximo.dispatchEvent( evt );
    }

  }
}
```

```
/* Clase encargada de recibir los eventos del mouse ocurridos en la cuadrícula. */
class ActivarDesactivarAtractoresEstables extends MouseAdapter {

    CanvasAtractoresEstables canvasRef;

    /* Constructor de la clase. */
    ActivarDesactivarAtractoresEstables ( CanvasAtractoresEstables canvas ){
        canvasRef = canvas;
    }

    @Override
    /* Método receptor de eventos del mouse para activar el indicador de que el mouse
     * ha sido presionado sobre el patrón de cuadrícula,
     * y obtener las coordenadas donde ha sucedido el evento. */
    public void mousePressed( MouseEvent e ) {
        canvasRef.mouseClick = true;
        canvasRef.puntoX = e.getX();
        canvasRef.puntoY = e.getY();
        canvasRef.repaint();
    }

    @Override
    /* Método receptor de eventos del mouse para activar el indicador de que el mouse
     * ha sido presionado dos veces sobre el patrón de cuadrícula,
     * y obtener las coordenadas en donde ha sucedido el evento. */
    public void mouseClicked ( MouseEvent e ){
        if ( e.getClickCount() == 2 ){
            canvasRef.mouseDobleClick = true;
            canvasRef.puntoX = e.getX();
            canvasRef.puntoY = e.getY();
            canvasRef.repaint();
        }
    }
}

/* Clase encargada de cerrar la ventana y salir del programa. */
class ConclusionAtractoresEstables extends WindowAdapter {
    @Override
  public void windowClosing( WindowEvent evt ) {
    System.exit( 0 );
    }
}

/* Clase encargada de limpiar el patrón de cuadrícula en caso de que así lo desee el usuario. */
class BorrarPatronAtractoresEstables implements ActionListener {
    CanvasAtractoresEstables canvasRef;

    /* Constructor de la clase. */
    BorrarPatronAtractoresEstables ( CanvasAtractoresEstables canvas ){
        canvasRef = canvas;
    }
```

```
    /* Método receptor de eventos de tipo acción que restablece
     * los valores de la cuadrícula a false. */
    public void actionPerformed ( ActionEvent evt ){
        for( int i = 0; i < canvasRef.columnas; i++ )
            for( int j = 0; j < canvasRef.filas; j++ )
                canvasRef.cuadricula[ i ][ j ] = false;
        canvasRef.repaint();
    }
}

/* Clase encargada de crear objetos de tipo cuadrícula y responder
 * a todas las indicaciones de pintado y repintado. */
class CanvasAtractoresEstables extends Canvas {

  boolean mouseClick, mouseDobleClick, pintarFocos;
  int ancho, alto, totalMemorias;
  int filas, columnas, deltaX, deltaY, x, y, puntoX, puntoY;
  double valorUmbral;
  int[] ejeX, ejeY;
  double[][]  neuronas;
  boolean[][] cuadricula;
  int[][] matrizMemorias;

  /* Constructor de la clase. */
  public CanvasAtractoresEstables () {
    this.setBackground( Color.white );
    totalMemorias = 0;
  }


  @Override
  /* Método receptor de eventos de pintado y repintado. */
  public void paint( Graphics g ){

    g.setColor( Color.black ); /* Establece el color por defecto a usar. */
    g.drawRect( 0, 0, ancho - 1, alto - 1 ); /* Dibuja el rectángulo de la cuadrícula. */

    /* Ciclo for que dibuja las líneas verticales de la cuadrícula
     * según el número de columnas establecido. */
    for ( int i = 0; i < filas - 1; i++ ){
        y = y + deltaY;
        g.drawLine( 0, y, 200, y );
        ejeY[ i + 1 ] = y;
    }

    /* Ciclo for que dibuja las líneas horizontales de la cuadrícula
     * según el número de filas establecido. */
    for ( int i = 0; i < columnas - 1; i++ ){
        x = x + deltaX;
        g.drawLine( x, 0, x, 200 );
        ejeX[ i + 1 ] = x;
    }

    x = ( ancho % columnas ) / 2;
    y = ( alto % filas ) / 2;
```

```
/* Bloque de ciclos for para pintar de color negro aquellos cuadros
 * que han sido activados en la cuadrícula. */
for ( int j = 0; j < filas; j++ )
   for ( int i = 0; i  < columnas; i++ )
      if ( cuadricula [ i ][ j ] )
         g.fillRect( ejeX[ i ] + 1 , ejeY[ j ] + 1 ,
               ejeX[ i + 1 ] - ejeX[ i ] - 1 , ejeY[ j + 1 ] - ejeY[ j ] - 1 );

/* Sentencia if/else que evalúa las banderas activadas por los clicks del mouse
 * dentro de la cuadrícula. */
if ( mouseDobleClick ){ /* Bloque para el caso de doble click (desactivar celda). */
   mouseDobleClick = false; /* Se desactiva la bandera de doble click. */
   int i, j;

   /* Sentencias for que determinan la celda de la cuadrícula donde ha sido recibido
    * el evento de doble click. */
   for ( i = 1; i <= columnas + 1; i++ ){
      if ( puntoX  < ejeX[ i ] ) break;
   }
   for ( j = 1; j <= filas + 1; j ++ ) {
      if ( puntoY < ejeY[ j ] ) break;
   }
   cuadricula[ i - 1 ][ j - 1 ] = false; /* Se desactiva la celda indicada. */
   /* Se limpia el rectángulo que ocupa la celda señalada. */
   g.clearRect( ejeX[ i - 1 ] + 1, ejeY[ j - 1 ] + 1,
         ejeX[ i ] - ejeX[ i - 1 ] - 1, ejeY[ j ] - ejeY[ j - 1 ] - 1 );
   g.drawRect( 0, 0, ancho - 1, alto - 1 );
}

else if ( mouseClick ){ /* Bloque para el caso click (activar celda). */
   mouseClick = false; /* Desactiva la bandera de click. */
   int i, j;

   /* Sentencias for que determinan la celda de la cuadrícula
    * donde ha sido recibido el evento del mouse. */
   for ( i = 1; i <= columnas + 1; i++ ){
      if ( puntoX  < ejeX[ i ] ) break;
   }
   for ( j = 1; j <= filas + 1; j ++ ) {
      if ( puntoY < ejeY[ j ] ) break;
   }
   cuadricula[ i - 1 ][ j - 1 ] = true; /* Se activa la celda indicada. */
   /* Se rellena el rectángulo que ocupa la celda señalada. */
   g.fillRect( ejeX[ i - 1 ] + 1 , ejeY[ j - 1 ] + 1 ,
         ejeX[ i ] - ejeX[ i - 1 ] - 1 , ejeY[ j ] - ejeY[ j - 1 ] - 1 );
}

/* Sentencia if que evalúa la bandera encargada de pintar sobre la cuadrícula el patrón obtenido
 * después de aplicar el método numérico a la red neuronal. */
if ( pintarFocos ) {

   pintarFocos = false; /* Se desactiva la bandera de pintado. */
   g.setColor( Color.black );
```

```
        /* Bloque de ciclos for para evaluar el valor del arreglo de neuronas,
         * y rellenar las celdas correspondientes a las neuronas activadas,
         * es decir, que han sobrepasado el valor umbral. */
        for ( int j = 0; j < filas; j++ )
            for ( int i = 0; i < columnas; i++ )
                if ( neuronas[ i ][ j ] >= valorUmbral ) {
                    g.fillRect( ejeX[ i ] + 1 , ejeY[ j ] + 1 ,
                            ejeX[ i + 1 ] - ejeX[ i ] - 1 , ejeY[ j + 1 ] - ejeY[ j ] - 1);
                    cuadricula[ i ][ j ] = true;
                }
    }

  }

}
```