UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

UN SISTEMA DE PATRONES DE SOFTWARE PARA REDES NEURONALES ARTIFICIALES

T E S I S

QUE PARA OBTENER EL TÍTULO DE :

LICENCIADA EN CIENCIAS DE LA COMPUTACIÓN

P R E S E N T A :

BEATRIZ PERALTA CORTÉS

DIRECTOR DE TESIS:
M. EN. C. JORGE LUIS ORTEGA ARJONA

FACULTAD DE CIENCIAS
UNAM

2005

Agradecimientos

Agradezco el apoyo del personal académico y administrativo que labora en el Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas de la UNAM, en especial a los miembros del Departamento de Ingeniería de Sistemas Computacionales y Automatización por brindarme su apoyo académico y moral para llevar a buen termino la tesis que presento.

Una mención muy especial merecen Héctor Benítez Pérez y Jorge Ortega Arjona quienes, además de brindarme su amistad, me acompañaron en la realización de mi tesis como asesores. Les doy gracias porque siempre tuvieron la paciencia y la comprensión necesaria para responder a mis dudas, cuestionamientos e inquietudes emanados durante la realización de esta tesis.

Asimismo, quiero expresar mi gratitud a los doctores Pedro Miramontes Vidal, Enrique Ruiz-Velasco Sánchez, y a la doctora Hanna Oktaba por la lectura, los cometario y las correcciones a este trabajo de investigación.

Gracias a mi mamá y a mi papá por la confianza que han depositado en mi, por su apoyo incondicional en todo lo que he realizado a lo largo de mi vida, por los esfuerzos que han realizado para sacarnos adelante a mis hermanos y a mí en particular, por estar a mi lado y acompañarme en los buenos y en los malos momento. En especial ahora que concluyo una etapa en mi vida profesional.

También agradezco a mis amigas, amigos y familiares por estar siempre conmigo, apoyarme acompañarme no sólo en este proceso sino en otros y de quienes siempre he recibido una palabra de aliento que me motiva y me impulsa a seguir cuando lo he necesitado. Sin embargo, no quiero mencionar a nadie en particular por temor a omitir algún nombre y porque todos ustedes siempre están en mis pensamientos.

Para terminar, agradezco a Dios por permitirme llegar a este momento y por ser un guía que ha iluminado mi andar en esta vida.

Índice General

Capítulo 1

Introducción

La presente tesis, "*Un Sistema de Patrones de Software para Redes Neuronales Artificiales*", describe un panorama general de las Redes Neuronales Artificiales y los Patrones de Software, con la finalidad de proponer un Sistema de Patrones de Software como guía para la solución de problemas usando Redes Neuronales Artificiales.

1.1 Resumen

Las Redes Neuronales Artificiales son un campo de investigación que estudia el desarrollo de modelos matemáticos a partir de elementos simples y no lineales [Corchado et al., 2000, Hilera and Martínez, 1995, Fausett, 1994]. Estos modelos son viables para resolver una amplia variedad de problemas como clasificación de patrones, reconocimiento y síntesis de lenguaje, funciones de aproximación, modelado de sistemas no lineales y de control, compresión de imagen, memoria asociativa, agrupamientos, entre otros [Corchado et al., 2000].

Existen diferentes formas de clasificar las Redes Neuronales Artificiales. Estas formas toman en cuenta criterios básicos como la *topología* de la red, el *tipo de asociación* de las señales de entrada y salida, así como los *mecanismos de aprendizaje* [Corchado et al., 2000, Hassoon, 1995]. Sin embargo, estas clasificaciones no establecen criterios para seleccionar el tipo de red a utilizar en un problema dado, considerando que un sistema de clasificación sirve comúnmente para la selección. Es por ello que se plantea el uso de Patrones de Software como guía para la selección de Redes Neuronales a partir de la experiencia obtenida en problemas ya conocidos.

Los Patrones (*patterns*) se proponen como elementos para la selección de configuraciones [Alexander et al., 1977, Buschmann et al., 1996, Coplien, 2001]. Un Patrón se define como "la relación entre un problema, descrito en términos de una función requerida, que ocurre dentro de un contexto determinado, y una solución descrita mediante una configuración genérica de componentes" [Alexander et al., 1977]. En general, el objetivo de los Patrones es capturar la experiencia de diseño, al conjuntar y describir una forma o configuración con la función que ésta realiza, y que se observa en sistemas ya existentes. Por medio de observación de casos, la forma puede ser parte del razonamiento predictivo alrededor de la función.

1.2 Objetivo

La presente tesis pretende analizar un conjunto de problemas resueltos con Redes Neuronales Artificiales, a fin de proponer un Sistema de Patrones de Software como una heurística para la selección de Redes Neuronales Artificiales. Es decir, partiendo del análisis de los problemas que se resuelven con Redes Neuronales, se busca identificar las características más relevantes que impliquen su uso como solución.

Las características del problema y la Red Neuronal que se utiliza al resolverlo son elementos importantes para considerar la existencia de un Patrón de Software. Sin embargo, es necesario verificar si tal relación problema-solución se presenta en diferentes contextos y repetidamente, para decir que realmente se trata de un Patrón de Software.

Este trabajo es una guía para el diseño de nuevas soluciones tomando en cuenta lo que ya se conoce. Por lo tanto el objetivo es proponer una aproximación de Patrones de Software a partir de las soluciones que existen y las características más relevantes para el uso de una Red Neuronal Artificial.

1.3 Estructura de la tesis

La estructura del trabajo se desarrolla de la siguiente manera:

- En el capítulo 2 se establecen los conceptos básicos de Patrones de Software, su clasificación y representación como descripciones de configuraciones de componentes de software.

- En el capítulo 3 se introducen los conceptos generales de Redes Neuronales Artificiales, sus diferentes clasificaciones y caracteríticas relevantes.

- En el capítulo 4 se presenta la propuesta de un Sistema de Patrones de Software para Redes Neuronales, como una descripción que captura la experiencia de diseño mediante la relación entre los problemas y las soluciones descritas como Redes Neuronales. Se consideran cuatro patrones de software [Hassoon, 1995, Hilera and Martínez, 1995, Corchado et al., 2000]: los *Mapas Auto Organizados de Kohonen* (SOM), el *Perceptrón Multicapa* (MLP), las *Redes de Funciones de Base Radial* (RBF) y la *Red de Hopfield*; este sistema no abarca todas las redes neuronales que existen, pero se propone como base para el uso de Patrones de Software en el área de Redes Neuronales.

- En el capítulo 5 se describe un problema de clasificación de fallas. A partir de las características del problema, se selecciona una configuración de Red Neuronal basada en un Patrón de Software que tentativamente lo resuelve.

- En el capítulo 6 se describen y comentan los resultados obtenidos, así como algún trabajo futuro en el área.

Capítulo 2

Patrones de Software

2.1 Definición de Patrón

Un Patrón es *la descripción de la relación entre un problema recurrente, que sucede en un contexto dado, con una solución que lo resuleve* [Alexander et al., 1977, Buschmann et al., 1996]. Los Patrones se utilizan para construir soluciones considerando la experiencia obtenida en la solución de problemas ya conocidos. Un patrón describe parte de una solución de acuerdo a un problema que esta presente en un contexto. Es decir, un patrón propone cómo hacer las cosas bajo ciertas condiciones, después del análisis de problemas ya resueltos.

Cuando se trata de resolver un problema nuevo, primero se identifica algún parecido a un problema ya resuelto, de tal manera que se pueda utilizar parte de su solución para desarrollar una nueva solución. Esta es una estrategia común en diferentes dominios de aplicación, como en arquitectura, economía, ingeniería de software, etc. [Buschmann et al., 1996]

Christopher Alexander [1] formaliza el término de Patrón en la arquitectura civil y el diseño urbano. En su libro *The Timeless Way of Building* [Alexander, 1979] desarrolla formalmente una aproximación a una síntesis basada sobre la composición de Patrones. Un patrón es una estructura recurrente dentro de un dominio de diseño. Puede ser bastante concreto o relativamente abstracto. Sin embargo debe estar compuesto de un problema para una aplicación y su solución.

[1] Arquitecto inglés y profesor de la Universidad de Stanford, California.

James O. Coplien define un patrón como *una pieza de literatura que describe el diseño de un problema y una solución general para problemas en contextos particulares* [Coplien, 2001]. Christopher Alexander define el término de patrón de la siguiente manera [Alexander, 1979]:

> *Cada patrón es una regla de tres partes, la cual expresa una relación entre cierto contexto, un problema y una solución.*
>
> *Como un elemento en el mundo, cada patrón es una relación entre un contexto, un cierto sistema de fuerzas que ocurre repetidamente en ese contexto, y una cierta configuración espacial que permite a esas fuerzas resolverse por sí mismas.*
>
> *Como un elemento del lenguaje, un patrón es una instrucción, la cual muestra cómo esta configuración espacial puede usarse una y otra vez para resolver el sistema dado de fuerzas, donde quiera que el contexto lo haga relevante.*
>
> *Un patrón es, en resumen, al mismo tiempo una cosa que sucede en el mundo y una regla que dice cómo crear esa cosa, y cuándo debe ser creada. Es tanto un proceso como una cosa: tanto una descripción de la cosa animada y una descripción del proceso con el cual se generará esa cosa.*

En el libro *Pattern-Oriented Software Architecture A System of Pattern* [Buschmann et al., 1996] un patrón se describe mediante un esquema formando por un contexto, un problema y una solución.

El *Contexto* describe la situación en la cual el problema se presenta. El contexto comprende lo que se va a resolver y las características que se deben cubrir. En esta parte, se determina la situación en la que el patrón se puede aplicar. Para ello se consideran las situaciones donde el patrón está presente. Esto no garantiza cubrir todas las situaciones, sin embargo propone una guía para su uso [Buschmann et al., 1996].

El *Problema* es una tensión entre los elementos del contexto. Se presenta repetidamente asociado al contexto. Su descripción especifica de manera general en qué consiste el problema y qué es "lo escencial" del mismo. El objetivo es definir lo que se va a resolver. Comúnmente, la especificación del problema está formada por un conjunto de *fuerzas* [2], por ejemplo:

[2]Esté término es utilizado por la comunidad de Patrones para denotar cualquier elemento que ocurre en el contexto y que define al problema.

- Los requisitos que la solución debe satisfacer.
- Las restricciones que se deben considerar.
- Las propiedades deseables que la solución puede tener.

Las fuerzas son la clave para resolver el problema. Analizarlas desde diferentes puntos de vista ayuda a entender los detalles. En ocasiones estas fuerzas pueden ser contradictorias o complementarias.

La *Solución* es la descripción de una organización de componentes que resuelve el problema. Describe cómo balancear las fuerzas asociadas al problema y al contexto. En general, no necesariamente se resuelven todas las fuerzas. Sin embargo, la solución propone cómo resolverlas o complementarlas, sobre todo en el caso en que las fuerzas sean contradictorias.

2.2 ¿Qué es un Patrón de Software?

En Ingeniería de Software, los patrones se utilizan para construir sistemas de software con propiedades especificas. Intentan capturar la experiencia desarrollada por expertos y promueven la práctica de diseño.

Un Patrón de Software *describe un problema de diseño particular, que proviene de un contexto de diseño específico y presenta un esquema general bien probado cómo solución* [Buschmann et al., 1996]. Este esquema de solución está formado por abstracciones como procedimientos u objetos que, al combinarse con otros, forman parte del sistema y específican como se construyen los componentes y las relaciones entre ellos. De esta manera describen sistemas de software probado y utilizado por diseñadores y programadores expertos.

Los Patrones de Software son una guía para proponer la solución parcial a problemas de diseño. En ellos se considera lo que está presente en el contexto y captura los aspectos escenciales para proponer una solución.

2.2.1 Descripción de Patrones de Software

Los Patrones de Software se describen a través de platillas con determinadas características. El objetivo es comprender "lo esencial" del patrón, es decir, en qué consiste el problema, cómo puede resolverse, bajo qué condiciones puede aplicarse,

cómo es la propuesta de solución, etc. Además debe proporcionar los detalles necesarios para su implementación así como las ventajas y desventajas de su aplicación.

La estructura básica de Contexto-Problema-Solución es un buen punto de inico para describir los patrones. Sin embargo, describir patrones basados exclusivamente en este esquema no es suficiente, ya que no considera aspectos necesarios para el desarrollo de la solución en forma de un sistema de software, que deben incluirse como parte de una descripción más completa.

Existen diferentes formas para describir los Patrones de Software: la forma de GoF (*Gang of Four*)[Gamma et al., 1995], la forma Portland, la forma de Coplien [Coplien, 2001], la forma POSA [Buschmann et al., 1996], y otras. Todas ellas coinciden en varios puntos que describen al patrón, como por ejemplo, un patrón debe tener un nombre intuitivo que encierre su esencia, así como una descripción del problema y las fuerzas asociadas, además de incluir guías para su implementación y los detalles que lo originaron.

Particularmente, esta tesis utiliza la forma de POSA para describir los patrones que se proponen. La forma POSA contiene las siguientes secciones:

Nombre El nombre identifica al patrón. Usualmente, el nombre es corto, con una palabra o una frase se indica la solución del problema o el contexto. Esta acompañado de un breve resumen del patrón [Coplien, 1994, Appleton, 1997, Lea, 1993].

También conocido como Incluye otros nombres, sinónimos o alias que identifican al patrón.

Ejemplos Los ejemplos ilustran la aplicación del patrón. En ellos se especifica el contexto inicial, cómo se aplica el patrón, la transformación y el resultado del contexto. Es necesario incluir uno o más ejemplos.

Contexto El contexto describe la situación en la cual el patrón puede aplicarse, incluye las precondiciones del problema y los requisitos que se deben satisfacer en la solución. El contexto se propone como una configuración inicial de un sistema antes de aplicar el patrón [Lea, 1993].

Problema El problema especifica de manera general lo que el patrón resuelve, incluye una discusión de las fuerzas asociadas al problema y los objetivos que se desean alcanzar.

Solución La solución describe cómo resolver de manera parcial el problema. Es equivalente a dar instrucciones para llevar a cabo un producto. La solución incluye la estructura y el funcionamiento necesarios para realizar el objetivo deseado.

Estructura La estructura describe detalladamente los aspectos estructurales de la solución. Incluye diagramas de clases, modelos de objetos, modelos dinámico o modelos funcionales.

Dinámica La dinámica describe el funcionamiento de la solución durante el tiempo de ejecución. Incluye diagramas de secuencia o diagramas de flujo.

Implementación La implementación describe cómo llevar a cabo la solución. Es equivalente a una guía que indica cómo implementar el patrón.

Ejemplo Resuelto El ejemplo resuelto se propone para realizar una discusión de los aspectos que ayuden a resolver un ejemplo que aún no es cubierto con la solución del patrón.

Variantes Las variantes describen especificaciones de soluciones semejantes al patrón.

Usos Conocidos Los usos conocidos describen ejemplos de sistemas que emplean la solución que el patrón propone. Se describen por lo menos tres ejemplos.

Consecuencias Las consecuencias describen las ventajas y desventajas de aplicar el patrón.

Ver también Ver también incluye referencias a patrones que resuelven problemas semejantes.

2.2.2 Sistema de Patrones de Software

Un Sistema de Patrones de Software es una colección estructurada de patrones. En Ingenieria de Software, un Sistema de Patrones sirve como una guía de implementación, combinación y uso práctico de patrones en el desarrollo de software [Buschmann et al., 1996]. El objetivo es considerar la composición de patrones para proponer parte de la solución a un objetivo deseado.

El Sistema de Patrones tiene una estructura que describe cada Patrón dentro del conjunto, como una forma posible de ordenar a sus componentes para satisfacer un objetivo, indicando su relación con otros patrones.

De acuerdo con Alexander, el arquitecto consulta un conjunto de Patrones para escoger de entre ellos, los Patrones que reunan elementos deseados en un proyecto. Así construye bloques para una síntesis, o sugiere elementos que podrían presentarse en la construcción. Cada patrón propone instrucciones para soluciones estructuradas, que son mezclados para producir el diseño de un sistema [Coplien, 2001].

2.3 Clasificación de Patrones

En los Sistemas de Patrones de Software, los patrones se clasifican de tal forma que se soporte el desarrollo de software y se cumpla con las siguientes propiedades:

- Ser simples y fáciles de aprender.
- Tener criterios de clasificación que no sean multidimensionales y que se organicen de acuerdo con las propiedades de los patrones.
- Cada criterio de clasificación debe reflejar propiedades del patrón, por ejemplo el tipo de problema al que están dirigidos.
- Proporcionar un itinerario del conjunto de Patrones aplicables potencialmente y no un esquema rígido que intente soportar o encontrar Patrones "correctos".
- El esquema debe estar abierto para la integración de nuevos Patrones de acuerdo a las necesidades para refabricar la clasificación existente.

Los Patrones están orientados a aspectos específicos dentro de un mismo dominio. Se clasifican para formar categorías.

2.3.1 Categoría de Patrones de Software

Los Patrones se clasifican de acuerdo a su escala o nivel de abstracción respecto a un sistema de software. Sirven para describir la estructura total de sistemas de software, o soportan relaciones entre sus subsistemas y sus componentes o ayudan en aspectos de su implementación usando un lenguaje de programación específico. De esta forma los patrones se agrupan en tres categorías [Buschmann et al., 1996]:

- Los *Patrones Arquitectónicos* describen la organización fundamental o el esquema general para un sistema de software. Son descripciones de las propiedades estructurales de un sistema. Estos Patrones proporcionan un conjunto de subsistemas predefinidos, especifican las responsabilidades entre ellos e incluyen reglas y guías para la organización de sus relaciones. Se utilizan al empezar la especificación y la estructura fundamental de un sistema.

- Los *Patrones de Diseño* proporcionan un esquema para refinar los subsistemas o componentes y las relaciones entre ellos. Estos Patrones describen comúnmente estructuras recurrentes de componentes comunicantes, que solucionan un problema de diseño general dentro de un contexto particular. Muchos Patrones de Diseño proporcionan estructuras para la descomposición de servicios o componentes complejos. Otros están dirigidos a la cooperación entre ellos y son idependientes del lenguaje o los paradigmas de programación.

- Los *Patrones Idiomáticos, (Idioms)*, algunas veces llamados patrones de código [Appleton, 1997], están orientados para un lenguaje de programación específico. Estos patrones describen cómo implementar aspectos particulares de componentes o las relaciones entre ellos usando las características de un lenguaje dado. Los patrones idiomáticos son usados en la fase de implementación para transformar la arquitectura de software en programación escrita con un lenguaje específico.

Nótese que la diferencia entre estas categorías de patrones radica en el nivel de abstracción de cada uno y sus detalles. Los Patrones Arquitectónicos son estrategias de alto nivel que conciernen a componentes de gran escala, como las propiedades globales y los mecanismos del sistema. Su uso afecta la estructura y organización del sistema como un todo. Por otra parte, los Patrones de Diseño son de nivel medio, es decir, consisten en tácticas para estructurar las entidades y sus relaciones. Sin embargo, estos patrones no afectan la estructura del sistema. Y los Patrones Idiomáticos son paradigmas y técnicas de programación específicas que constituyen un bajo nivel o los detalles de la estructura de sus componentes [Appleton, 1997].

2.3.2 Categoría de Problemas

La Categoría de Problemas clasifica a los patrones respecto a situaciones de diseño concreto. Algunas Categorias de Problemas son [Buschmann et al., 1996]:

- *De la nada a una estructura* incluye patrones que soportan una adecuada descomposición de las tareas del subsistema cooperando dentro de subtareas.
- *Sistemas distribuidos* incluye patrones que proporcionan infraestructura para sistemas con componentes localizados en diferentes procesadores o en algunos subsistemas y componentes.
- *Sistemas interactivos* incluye patrones que ayudan a la estructura del sistema con interacción hombre-computadora.
- *Sistemas Adaptables* incluye patrones que proporcionan infraestructura para la extensión y adaptación de aplicaciones en respuesta a la evolución y cambios funcionales requeridos.
- *Descomposición estructural* incluye patrones que soportan una adecuada descomposición de subsistemas y componentes complejos.
- *Organización de tareas* incluye patrones que definen como colaboran los componentes para proporcionar un servicio complejo.
- *Control de Acceso* incluye patrones que guardan y controlan el acceso a servicios o componentes.
- *Manejadores* incluye patrones para soportar colecciones de objetos homogéneos, servicios y componentes dentro de una entidad.
- *Comunicación* incluye patrones que ayudan a organizar la comunicación entre componentes.
- *Reutilización* incluye patrones que colaboran en el manejo de partes entre componentes y objetos.

Existen algunos Patrones que pueden ser asignados a más de una Categoría de Problemas.

2.4 Selección de Patrones

Un Sistema de Patrones define un procedimiento para seleccionar un patrón. Incluye siete pasos [Buschmann et al., 1996]:

1. *Especificar el problema* consiste en entender el problema a resolver, es decir, ¿cuál es el problema?, ¿cuáles son sus fuerzas?, si existen algunos aspectos que influyen sobre la estructura básica del sistema, si es distribuido o iterativo, si es necesario dividir el problema en subproblemas, etc.

2. *Seleccionar la categoría del Patrón* corresponde a proponer un nivel de escala o abstracción de acuerdo al problema de diseño que se desarrolla. Es necesario decidir si el patrón es arquitectonico, de diseño o idiomático.

3. *Seleccionar la categoría del Problema* corresponde a proponer una situación de diseño concreto de acuerdo al tipo de problema. En la categoría de problemas, cada categoría resume el tipo de problema al que está dirigido. Si no es posible asociar el problema de diseño concreto con alguna categoría de problemas, entonces se selecciona una categoría de problema alternativo.

4. *Comparar el problema descrito* consiste en proponer un patrón que ayude a resolver el problema concreto. A partir de la descripción del problema y las fuerzas que mejor se ajusten al patrón seleccionado en la categoría de problemas, se escoge aquél que satisfaga los aspectos particulares del problema a resolver. Así, un patrón, o la combinación con otros patrones, puede ayudar a resolver el problema concreto. En este paso se requiere conocimiento específico acerca del problema de diseño.

5. *Comparar los beneficios y desvejas* consiste en describir las ventajas y desventajas de aplicar el patrón seleccionado. Se escoge el patrón que proporcione más beneficios y menos desventajas.

6. *Seleccionar las variantes* consiste en escoger de la documentación de patrones, los patrones que implementen la solución o que especifiquen alguna arquitectura semejante a la del problema.

7. *Seleccionar una categoría de problema alternativo* consite en escoger otra categoría de problema. Esto se hace en el caso de que no haya una categoría de problemas que proponga parte de la solución o cuándo la categoría de problemas no incluye patrones que ayuden en la solución del problema.

Los pasos anteriores sirven para seleccionar un patrón que contribuya en la construcción de soluciones a un problema de diseño específico. Algunos pasos, como 2, 3 y 4, no proporcionan propiamente ningún resultado, en el sentido de construir parte de la solución. Sin embargo, son útiles para seleccionar una categoría de

problema alternativo cuando el sistema de patrones no incluye ningún patrón que ayude a resolver el problema de diseño.

2.5 Resumen

Los Patrones se proponen como guías para construir soluciones a partir de soluciones previas a problemas resueltos, considerando los elementos que se presentan repetidamente en un problema dentro de un contexto determinado. El objetivo de los patrones es capturar lo esencial del problema para proponer una solución. Los patrones se describen a través de plantillas que proporcionan los elementos fundamentales para construir una solución. En esta tesis, se utiliza la forma POSA para describir los patrones.

Los patrones se clasifican de acuerdo a su nivel de abstracción. Pueden ser patrones arquitectónicos, de diseño o idiomáticos. Otra clasificación considera el tipo de problema que resuelven. En este caso los patrones están dirigidos a sistemas distribuidos, iterativos, adaptables, o a organización de tareas, de comunicación, etc.

Por último, para resolver un problema utilizando patrones se describe un esquema de selección formado por siete pasos. Estos pasos contribuyen a elegir un patrón que proponga parte de la solución al problema de diseño.

Capítulo 3

Redes Neuronales Artificiales

Una Red Neuronal Artificial es un sistema de procesamiento de información que está formado por elementos no-lineales llamados neuronas. Son modelos matemáticos viables para resolver una amplia variedad de problemas, tales como clasificación de patrones, reconocimiento y síntesis de lenguaje, funciones de aproximación, modelado de sistemas no lineales y de control, compresión de imagen, memoria asociativa, agrupamientos, etc. [Luo and Unbehauen, 1997, Hassoon, 1995, Caudill and C., 1992, Hilera and Martínez, 1995].

Las Redes Neuronales Artificiales intentan reproducir de forma simplificada el funcionamiento de un cerebro [Corchado et al., 2000]. No tratan de reproducir el cerebro humano, se centran en mecanismos de resolución de problemas individuales. Por ejemplo, tratan de imitar el proceso de almacenar información en patrones y utilizar tal información para resolver cierto tipo de problemas.

Las Redes Neuronales Artificales proporcionan un esquema de procesamiento alternativo basado en la operación de un número determinado de neuronas que se conectan entre sí. Un atributo significativo de las neuronas es su habilidad para aprender, interactuando con el medio en el que se encuentran o con información de entrada.

En este capítulo se presentan diferentes formas de clasificar las redes neuronales, donde se consideran algunos criterios básicos: la *arquitectura* de la red, *el tipo de asociación* de las señales de entrada y salida, y los *mecanismos de aprendizaje*.

En el caso de los mecanismos de aprendizaje se considera la clasificación de Redes Neuronales Artificiales en tres clases de mecanismos [Hassoon, 1995]:

- *Aprendizaje supervisado.* En el aprendizaje supervisado, cada entrada que recibe la red en la fase de entrenamiento se compara con un objetivo específico. De este modo, la red modifica su comportamiento en función de los resultados que se van obteniendo al hacer tal comparación, a fin de producir una salida.

- *Aprendizaje no supervisado.* El aprendizaje no supervisado involucra el agrupamiento o detección de similitudes entre elementos de un conjunto de entrenamiento a su entrada. En la fase de entrenamiento, la red modifica su comportamiento en base a mecanismos probabilísticos, lo que determina la salida de la red.

- *Aprendizaje reforzado (Reinforcement).* En el aprendizaje reforzado, a cada neurona se asocia un valor o peso determinado. Durante la fase de entrenamiento, la red toma el error entre la entrada y la salida para maximizar los valores esperados de una función criterio (llamada *señal reforzadora*) [Hassoon, 1995]. El comportamiento de la red se modifica, actualizando los pesos de cada neurona de acuerdo a la función criterio. Esto modifica a la vez su salida, repitiéndose el ciclo hasta obtener un error mínimo.

3.1 La Neurona Artificial

La *Neurona Artificial* es la unidad de procesamiento básico de una Red Neuronal. En la figura 3.1 se muestra la estructura de una neurona artificial genérica, la cual tiene asociado un conjunto de conexiones caracterizadas por un valor llamado *peso* (w). A través de las conexiones, la neurona recibe valores de entrada x y produce una salida Y. La relación entre las entradas y la salida está determinada por una función que se conoce como *función de activación* o *función de transferencia* F.

La neurona artificial se comporta de la siguiente manera:

1. Una neurona k recibe un conjunto de entradas que dependen de sus conexiones con otras neuronas o del exterior. De esta forma, a cada entrada x se le multiplica por el peso w_i, $(1 \leq i \leq n)$ correspondiente a su conexión de entrada.

2. A partir de las entradas a la neurona, se determina el *nivel de activación* o *nivel de excitación* que dispara la propagación de actividad en la neurona k. El nivel de activación se determina mediante una *regla de propagación*, denotada

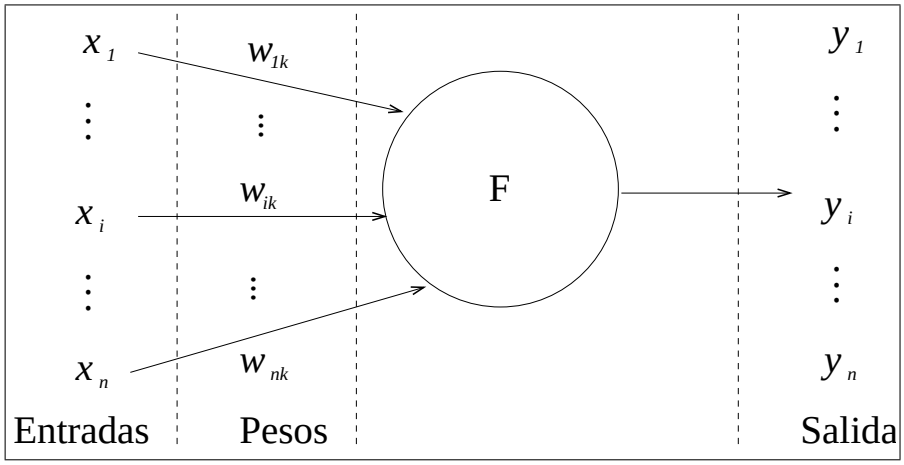

Figura 3.1: Neurona Artificial.

por s_k. Esta regla considera la suma de todas las entradas x a la neurona k, ponderadas por los pesos correspondientes w_i.

3. La salida de la neurona k denotada por Y_k, se determina a partir de su nivel de excitación y de la función de activación F_k. El objetivo de esta función es limitar la amplitud de la salida dentro de un rango de valores normalizados. Generalmente, la salida se encuentra en el intervalo de [0,1] o [-1,1].

La neurona artificial representa un algoritmo que realiza la suma ponderada de las diferentes entradas que recibe de otras neuronas o del exterior, y produce una salida según sea el resultado de la suma con respecto al umbral o nivel de disparo. La función de transferencia para la activación de la neurona depende del problema que se quiera resolver.

El modelo básico de la neurona se describe mediante las siguiente ecuación:

$$s_k = \sum_{i=1}^{n} w_{ik} x_i + b_k \tag{3.1}$$

donde b_k denota la polarización o "bias", con el fin de aumentar o disminuir el nivel de excitación de la neurona, dependiendo de su valor. Si se considera a b_k como el peso de una entrada adicional siempre igual a 1, entonces la ecuación 3.1 se reescribe

de la siguiente manera.

$$s_k = \sum_{i=1}^{n} w_{ik} x_i \tag{3.2}$$

3.2 Tipos de Funciones de Activación

La función de activación **F(s)** de una neurona artificial define los diferentes estados de la red en términos del nivel de excitación, donde $\mathbf{s} = \mathbf{wx}$, $\mathbf{w} = (b_k, w_{1k}, w_{2k}, \ldots, w_{nk})$ es el vector de pesos y $\mathbf{x} = (1, x_1, x_2, \ldots, x_n)^T$ es el vector de entradas. Dependiendo de su representación matemática, tal función de activación puede ser de compuerta de umbral lineal, de compuerta de umbral cuadrático, de compuerta de umbral polinomial [Hassoon, 1995], o sigmoidal [Corchado et al., 2000].

- Función de Umbral Lineal

  Esta función de activación, también es conocida como *Lineal Threshold Gates o LTG*, es la primera función utilizada en las redes neuronales artificiales. Matemáticamente se define de la siguiente forma:

$$Y_k = \mathbf{F}_k(\mathbf{wx}) = \begin{cases} 1 & \text{si } \mathbf{wx} \geqslant 0 \\ 0 & \text{si } \mathbf{wx} < 0 \end{cases} \tag{3.3}$$

  con $x \in \{0,1\}^n$ y $w \in \mathbb{R}^n$ y $Y \in \{0,1\}$.

  Las neuronas que emplean esta función clasifican los puntos del espacio de entrada en subespacios, cuando los datos son linealmente separables. Esto se lleva a cabo mediante la adaptación adecuada de los pesos.

- Función de Compuerta de Umbral Cuadrático

  La función de Compuerta de Umbral Cuadrático (*Quadratic Threshold Gates*, o QTG) se utiliza para clasificar los puntos de un espacio de entrada en subespacios, cuando los datos no son linealmente separables. Esta función se define mediante la siguiente expresión:

$$Y_k = F_k(\mathbf{wx}) = \begin{cases} 1 & \text{si } \sum_{i=1}^{n} \mathbf{w}_i \mathbf{x}_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} w_{ij} \mathbf{x}_i \mathbf{x}_j \geqslant T \\ 0 & \text{en otro caso} \end{cases} \tag{3.4}$$

donde T es un valor de umbral, $T \in \mathbb{R}$.

- Función de Compuerta de Umbral Polinomial

  La función de Compuerta de Umbral Polinomial (*Polynomial Threshold Gates*, o PTG) expande el concepto de las funciones anteriores mediante un polinomio de orden r. Así, una compuerta de umbral polinomial se denota por PTG(r). Nótese que la función umbral lineal (LTG) y la función de umbral cuadrático (QTG) son casos particulares de la compuerta de umbral polinomial, la función LTG = PTG(1), y QTG = PTG(2). La compuerta de umbral polinomial PTG(r) se define de la siguiente manera:

$$Y_k = F_k(\mathbf{wx}) = \begin{cases} 1 & \text{si } \sum_{i_1=1}^{n} \mathbf{w}_{i_1} \mathbf{x}_{i_1} + \sum_{i_1=1}^{n} \sum_{i_2=i_1}^{n} w_{i_1 i_2} \mathbf{x}_{i_1} \mathbf{x}_{i_2} + \ldots + \sum_{i_1=1}^{n} \sum_{i_2=i_1}^{n} \\ & \qquad \ldots \sum_{i_r=i_{r-1}}^{n} w_{i_1 i_2 \ldots i_r} \mathbf{x}_{i_1} \mathbf{x}_{i_2} \ldots \mathbf{x}_{i_r} \geq T \\ 0 & \text{en otro caso} \end{cases} \tag{3.5}$$

  Las neuronas que emplean esta función se utilizan para clasificar los puntos de un espacio de entrada en subespacios, de tal manera que esta función tiene mayor grado de libertad.

- Función Sigmoidal

  La función sigmoidal (*sigmoid function*) es estrictamente creciente y presenta un equilibrio entre el comportamiento lineal y no lineal de los datos. Un ejemplo de esta función es la función logística, definida como:

$$Y_k = F_k(\mathbf{wx}) = \frac{1}{1 + e^{-a\mathbf{wx}}} \tag{3.6}$$

  donde el parámetro a es la *pendiente de la curva*. Por ejemplo si a tiende a infinito, la función tiende a alcanzar la forma de la función umbral lineal. Esta función varía en el intervalo [0,1].

- Función Signo

  La función signo es simétrica a la función umbral respecto al origen y varía en el intervalo [-1,1], es decir:

$$Y_k = F_k(\mathbf{wx}) = \begin{cases} 1 & \text{si } \mathbf{wx} > 0 \\ 0 & \text{si } \mathbf{wx} = 0 \\ -1 & \text{si } \mathbf{wx} < 0 \end{cases} \tag{3.7}$$

3.3 Redes Neuronales Artificiales

Las Redes Neuronales Artificiales son sistemas de procesamiento de información, formados por elementos no lineales conocidos como neuronas. El conjunto de neuronas que conforman la red neuronal se conectan de tal manera que la salida de una neurona cualquiera sirve, generalmente, como entrada de otras neuronas. El número de neuronas, la disposición de las mismas y las conexiones entre ellas determinan su estructura, denominada arquitectura o topología de la red. La figura 3.2 muestra la topología general de una red neuronal, formada por tres capas de neuronas: la capa de entrada, la capa intermedia u oculta y la capa de salida. Las conexiones entre estas capas también se caracterizan por valores llamados pesos.

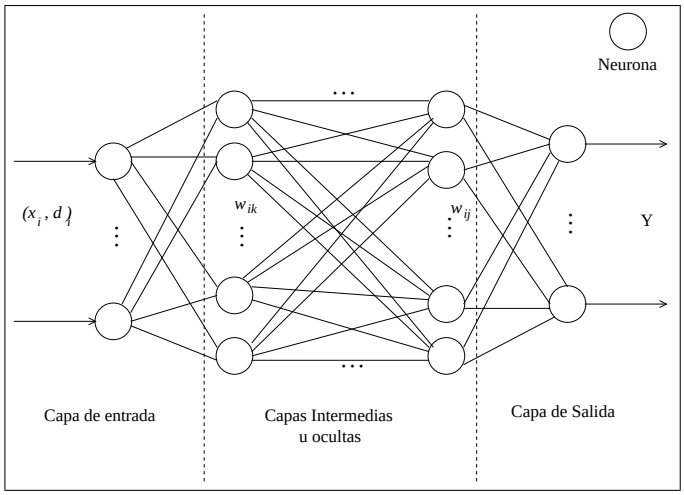

Figura 3.2: Topología de una Red Neuronal Artificial.

La *capa de entrada* recibe del exterior los datos que procesa la red. Esta capa toma los valores de entrada, también conocidos como conjunto de entrenamiento. Generalmente, estos valores son vectores que representan las características de un problema y son envíados a la capa intermedia o a la capa de salida dependiendo de la topología que tiene la red.

La *capa intermedia* u *oculta* recibe la información de la capa de entrada y evalúa la función que determina como modificar los pesos de sus conexiones. Además proporciona el resultado que se envía a otra capa oculta o a la capa de salida. Esto depende de la topología de la red, porque la red puede tener más de una o ninguna capa oculta.

La *capa de salida* proporciona el resultado final del procesamiento que lleva a cabo la red. Esta capa recibe la información de la capa de entrada o de la última capa oculta y realiza las operaciones para modificar y ajustar los pesos entre las conexiones, de tal manera que proporciona una solución.

La fase de entrenamiento consiste en determinar los valores de los pesos asociados a cada conexión. Al iniciar el entrenamiento, los pesos se asignan de manera aleatoria o a través de algún algoritmo matemático. Después, se determinan los parámetros necesarios para ajustar los pesos entre las neuronas de acuerdo a una regla, llamada *regla de aprendizaje*. Así, los pesos asociados a cada una de las conexiones representan la configuración de la red. Está configuración se modifica en el tiempo a través del entrenamiento, el cual determina la capacidad aprendizaje o adaptación de la red. Después de determinar los pesos de las conexiones, la fase de aprendizaje consiste en presentar nuevos datos de entrada y la red es capaz de proporcionar un resultado. A través del entrenamiento y aprendizaje de la red se detemina el resultado final.

3.4 Clasificación de Redes Neuronales

En esta sección se presentan diferentes formas de clasificar las redes neuronales, donde se consideran criterios básicos como la forma en que están dispuestas las neuronas, cómo son los datos de entrada y salida o la manera en que aprende la red.

La manera en que están dispuestas las neuronas definen la *arquitectura* o *topología* de la red, es decir, cuántas capas de neuronas conforman la red, cómo envían la información a otras neuronas, y si solo cuenta con conexiones hacia las capas superiores (conexiones hacia adelante) o también tiene conexiones con las capas inferiores (conexiones hacia atrás).

El *tipo de asociación* de los datos de entrada y salida se refiere a los datos que utiliza la red para el aprendizaje, por ejemplo, si la información es binaria o analógica, o recibe un tipo de información y regresa otro.

La forma en que aprenden las redes neuronales se refiere a los *mecanismos de aprendizaje* que emplea la red para modificar los pesos ente sus conexiones.

Los aspectos anteriores definen la clasificación clásica de las Redes Neuronales Artificiales, la cual considera las siguientes clasificaciones: (a) Clasificación por la arquitectura de la red, (b) Clasificación por datos de entrada y salida y (c) Clasificación por forma de aprendizaje.

3.4.1 Clasificación por arquitectura de la red

En esta clasificación las redes neuronales se dividen de acuerdo con el número de capas o niveles de neuronas y el tipo de conexiones entre las mismas. Una distinción entre ellas son las redes *monocapa* y *multicapa*.

Las *redes monocapa* están formadas por una capa de neuronas que intercambia señales con el exterior. Estas señales constituyen la entrada y salida del sistema. En algunos casos las neuronas establecen conexiones laterales con las neuronas, dando origen a conexiones autorrecurrentes, es decir, la salida de la neurona se conecta con su propia entrada. Un ejemplo de red neuronal con este esquema es la red de Hopfield.

Las *redes multicapa* están formadas por conjuntos de neuronas jerárquicas en distintos niveles o capas, con al menos una capa de entrada y otra de salida, en algunos casos una o varias capas intermedias u ocultas.

Normalmente todas las neuronas de una capa reciben datos de una capa anterior y envían datos a una capa posterior. Este tipo de conexiones se le conoce como conexiones "hacia adelante" (o *feedforward*). Si una red sólo dispone de este tipo de conexiones se llama red *feedforward*. Algunas redes que cuentan con este tipo de conexiones son: el perceptrón simple, el perceptrón multicapa, las redes de funciones de base radial, y la red Mapas auto-organizados de Kohonen [Pal and Mitra, 1999].

Sin embargo, existen otras redes en las cuales las neuronas tienen conexiones con neuronas de capas anteriores. A estas conexiones se les denomina conexiones "hacia atrás" (o *feedback*). En tal caso, se dice que se trata de una red *feedback*. Entre estas redes, destacan las redes Adaline y Madaline, y las redes de retropropagación (*backpropagation*) [Pal and Mitra, 1999].

3.4.2 Clasificación por Datos de Entrada y Salida

En esta clasificación se considera la naturaleza de los datos de entrada y salida que emplea la red neuronal para llevar a cabo el entrenamiento y aprendizaje. Los datos son analógicos, discretos o híbridos.

Las redes neuronales analógicas procesan datos de entrada cuyos valores son reales, continuos y acotados, normalmente entre los intervalos de [-1,1] o [0,1], para dar una respuesta analógica. Estas redes presentan normalmente funciones de activación continuas lineales o sigmoides. Entre ellas destacan las redes de retropropagación, los mapas de Kohonen, y la memoria lineal asociativa, entre otras.

Las redes neuronales discretas procesan datos de entrada discretos, generalmente binarios {0,1}, dando como respuesta datos discretos. Entre ellas destacan: la red discreta de Hopfield y la Máquina de Boltzman.

Las redes neuronales híbridas procesan datos de entrada analógica y como respuesta regresan datos binarios o discrtetos. Entre ellas se encuentran el perceptrón, y las redes Adaline y Madaline.

3.4.3 Clasificación por forma de aprendizaje

Las redes neuronales artificiales llevan a cabo un proceso mediante el cual mo-difican y actualizan los pesos entre las conexiones de las neuronas, de acuerdo a una regla establecida. A este proceso se le conoce como fase de *entrenamiento*. El entrenamiento es visto como una búsqueda en un espacio parametrizado multidimensional llamado *pesos*, para dar una solución y optimizar gradualmente la *regla de aprendizaje* [Hassoon, 1995]. Después de un cierto período de tiempo, la red ya no modifica los pesos entre sus conexiones, es decir, la fase de entrenamiento ha concluido y ahora la red es capaz de proporcionar una repuesta ante un dato de entrada nuevo. A esta fase se la llama fase de *aprendizaje*.

La forma en que se lleva a cabo el entrenamiento y el aprendizaje son fundamentales para el desempeño de la red, y determinan la siguiente clasificación. Esta clasificación se divide en tres grupos: *aprendizaje supervisado*, *aprendizaje no supervisado* y *aprendizaje reforzado (Reinforcement)*.

En el **aprendizaje supervisado** cada patrón o dato de entrada que recibe la red es comparado con un dato específico, de tal manera que la red modifica los pesos gradualmente. En cada paso de la fase de entrenamiento se actualizan los pesos hasta que el error entre la salida de la red y los datos establecidos se reduce.

En este caso los pesos se obtienen minimizando alguna función de error, la cual mide la diferencia entre los valores establecidos y los valores calculados por la red.

Las redes neuronales con este tipo de aprendizaje reciben como entradas dos vectores $\{\mathbf{x}, \mathbf{d}\}$. El primer vector $\mathbf{x}$ es un *conjunto de entrenamiento o entrada*, que generalmente son valores utilizados para representar las características de un problema y en este caso son usados para el entrenamiento de la red. El segundo vector $\mathbf{d}$ es el conjunto con el cual se compara la salida de la red, *los valores esperados* o los valores que se desea obtener, los cuales se establecen al inicio de la fase de entrenamiento.

La red neuronal procesa $\mathbf{x}$ y compara los resultados obtenidos durante el entrenamiento con $\mathbf{d}$. La diferencia entre ambos datos se conoce como el *error* que comete la red. Una vez obtenido el error, el objetivo es minimizarlo. Para ello se emplean diferentes algoritmos. Estos algoritmos calculan la diferencia entre la salida de la red y la salida esperada, utilizando este error para ajustar los pesos entre las conexiones de las neuronas. Entre los algoritmos empleados para este objetivo se encuentran *las reglas de corrección de error*.

Las *reglas de corrección de error* son manipulaciones sobre los datos a comparar que originalmente se proponen como unidades de entrenamiento simples. Estas reglas manejan el error de la salida hasta aproximarlo a cero. Algunos ejemplos de estas reglas son la regla del perceptrón, la regla de mínimos cuadrados (o *Least Mean Square*, LMS) y la regla delta generalizado [Hassoon, 1995].

La regla de aprendizaje del perceptrón considera una compuerta de umbral lineal, que mapea un vector de entrada $\mathbf{x} = [x_0 x_1 x_2 \dots x_n]^T$ a una salida binaria y. El valor de entrada x_0 usualmente es igual a uno y juega el papel de *bias* o umbral (véase la sección 3.2); $\mathbf{w} = [w_0 w_1 w_2 \dots w_n]^T \in \mathbb{R}^n$ es el vector de *pesos*. La relación entre la entrada y la salida del perceptrón está determinada por la función de activación, en este caso $y = sgn(\mathbf{x}^T\mathbf{w})$ donde *sgn* es la función signo, que regresa $+1$ ó -1 dependiendo del signo del argumento escalar.

En la fase de entrenamiento se busca que el perceptrón identifique los pares de entrenamiento $\{\mathbf{x}_1, d_1\}, \{\mathbf{x}_2, d_2\}, \dots, \{\mathbf{x}_m, d_m\}$, donde $\mathbf{x}_k \in \mathbb{R}^n$ es el k-ésimo vector de entrada y $d_k \in \{-1, +1\}$ es el valor esperado para ese vector de entrada, con $k = 1, 2, \dots, m$. El orden en que se presentan los vectores de entrenamiento siempre es alterado. El objetivo es diseñar un perceptrón tal que su salida Y_k, para cada vector de entrada $\mathbf{x}_k$ coincida con d_k. Para ello se requiere que $Y = sgn(\mathbf{x}^T\mathbf{w}_k) = d_k$ para cada $k = 1, 2, \dots, m$. Si esto se cumple, entonces el perceptrón clasifica correctamente el conjunto de entrenamiento y satisface las siguientes desigualdades:

$$\begin{cases} (\mathbf{x}_k)^T\mathbf{w}^* > \theta & \text{si } d_k = +1 \\ (\mathbf{x}_k)^T\mathbf{w}^* < \theta & \text{si } d_k = -1 \end{cases} \tag{3.8}$$

$\theta = 0$ es el valor del umbral para este caso.

El objetivo es encontrar un vector de solución $\mathbf{w}^*$ para la ecuación 3.8. Es decir, un hiperplano que clasifique correctamente todos los vectores $x_k, k = 1, 2, \ldots, m$, tal que $\mathbf{x}_k^T\mathbf{w}^* = 0$. Esta solución divide el espacio de entrada en dos regiones; una de las regiones contiene todos los puntos x_k con $d_k = +1$ y la otra región contiene todos los puntos x_k con $d_k = -1$.

La regla de aprendizaje que utiliza el perceptrón para encontrar la solución $\mathbf{w}^*$, se desarrolla de la siguiente manera: al iniciar la fase de entrenamiento, se inicia el vector de pesos $\mathbf{w}_1$ de manera aleatoria. Después se calcula la suma de los pesos por las entradas y compara el resultado con un valor umbral θ, que es la actividad mínima requerida para que el perceptrón genere una salida positiva. Si el resultado de la red es mayor o igual que el umbral, la salida es $+1$, sino es -1 y se aplica la función de transferencia $Y = sgn(\mathbf{x}^T\mathbf{w})$. Esto se muestra en la figura 3.3.

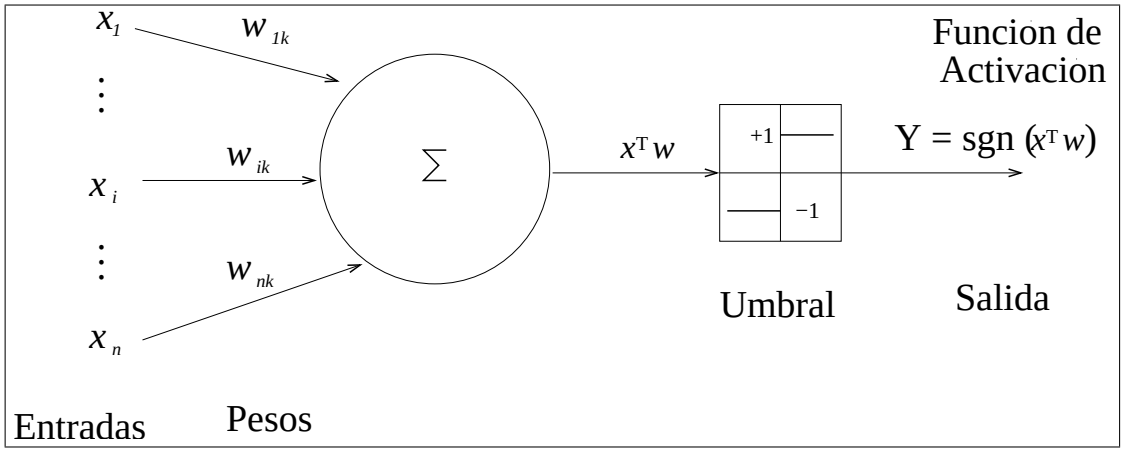

Figura 3.3: El perceptrón.

Así se utilizan los m pares $\{\mathbf{x}_k, d_k\}$, para actualizar sucesivamente el vector de pesos, hasta que se encuentre una solución $\mathbf{w}^*$ que clasifique correctamente el conjunto de entrenamiento. Esto se expresa en la siguiente ecuación:

$$\begin{cases} \mathbf{w}_0 & \text{arbitrario} \\ \mathbf{w}_k = \mathbf{w}_{k-1} + \rho(d_{k-1} - y_{k-1})\mathbf{x}_{k-1}, & k = 1, 2, \ldots \end{cases} \tag{3.9}$$

donde ρ es una constante positiva llamada *razón de aprendizaje*.

Otra regla de corrección de error es la regla LMS que emplea una función de activación cuadrática. Esta regla originalmente se emplea para entrenar una unidad lineal, conocida como ADALINE (*adaptive linear combiner element*) figura 3.4.

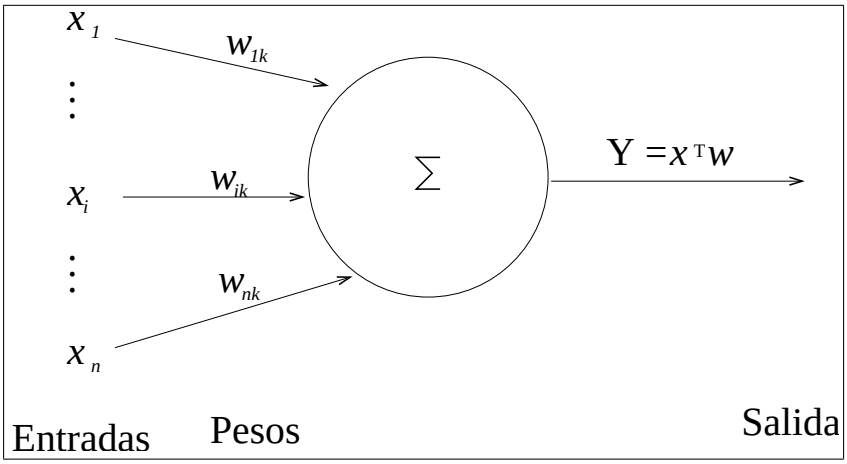

Figura 3.4: ADALINE.

En este caso, la salida de la neurona Y_k ante la entrada $\mathbf{x_k}$ es simplemente $Y_k = (\mathbf{x}_k)^T\mathbf{w}$. Esta regla converge a la solución $\mathbf{w}^*$ por medio de mínimos cuadrados LMS (*least-mean-square*) que corresponde al error de salida sobre los patrones de la misma longitud, es decir, la norma $||\mathbf{x}_k||$ es la misma para toda k.
La regla LMS es:

$$\begin{cases} \mathbf{w}_1 = 0 \text{ ó arbitrario} \\ \mathbf{w}_{k+1} = \mathbf{w}_k + \alpha(d_k - y_k)\frac{\mathbf{x}_k}{||\mathbf{x}_k||_2} \end{cases} \tag{3.10}$$

donde $d_k \in \mathbb{R}$ es la salida deseada y 0 ¡ $\alpha < 1$ es la constante que controla la velocidad de convergencia y no depende de la magnitud de los vectores de entrada. Por otro lado, α se utiliza para distinguir esta regla de otras reglas similares.

LMS es otra regla de aprendizaje que se utiliza para encontrar la solución $\mathbf{w}^*$ y se desarrolla de la siguiente manera: al iniciar la fase de entrenamiento, se inicializa el vector de pesos $\mathbf{w}_1$ de manera aleatoria ó igual a cero. Después se calcula la suma de los pesos por las entradas, se compara el resultado con el valor esperado y se calcula el error. El error es igual a la salida deseada menos la salida de la red. Una vez calculado el error, se utiliza para ajustar los pesos de las conexiones entre la neuronas.

El algoritmo consiste en moverse en el espacio de los pesos, en el sentido de disminución del gradiente y en la dirección del mismo. El objetivo es modificar los pesos w_{ik} de tal manera que se minimice la suma de los cuadrados de los errores, mediante un incremento denotado por Δw_{ik} proporcional al gradiente del error, hasta obtener la respuesta esperada. Si el resultado de la red es diferente del valor esperado entonces se evalúa la ecuación 3.10 repetidamente hasta obtener la respuesta esperada. En seguida se presenta otro patrón del conjunto de entrenamiento. Si después de su evaluación, el resultado no es el deseado, se calcula el error y nuevamente se aplica la ecuación 3.10 hasta obtener el mínimo error. Antes de presentar un nuevo patrón se verifica que, al evaluar el patrón anterior se obtenga la respuesta esperada. Si no es así, se debe modificar nuevamente los pesos. Este proceso se repite constantemente hasta ajustar los pesos, de tal manera que al evaluar los patrones previos se obtenga la respuesta esperada.

Hasta ahora sólo se han tratado reglas de correción de error para redes con una sola capa de neuronas. Sin embargo, existen redes, como el perceptrón multicapa, que está formado por una capa de neuronas de entrada, una capa de neuronas de salida, y una o más capas de neuronas ocultas o intermedias. Esta red también emplea un algoritmo de aprendizaje que minimiza el error entre la salida de la red y la salida deseada. Para ello utiliza la *regla delta generalizada* ó *backpropagation*.

La regla delta generalizada se divide en tres fases: (a) recibe los datos de entrenamiento y los envía a las capas ocultas, (b) evalúa la función de activación y calcula el error de la red, y (c) propaga el error hacia las capas intermedias y ajusta los pesos [Fausett, 1994]. Esto se muestra en la figura 3.5.

El perceptón multicapa recibe un vector de entrada x_k y lo envía a la primer capa oculta, en donde se calcula la función de activación. El resultado z_k se propaga a través de las capas ocultas hasta la capa de salida. En la capa de salida se calcula la respuesta de la red Y. Durante el entrenamiento, cada neurona de salida compara el resultado de la red y_k con el valor esperado d_k para determinar el error asociado al patrón de entrada x_k. Si la salida de la red no coincide con la salida esperada, se calcula el error con δ_k, que es la derivada del error con respecto a la función

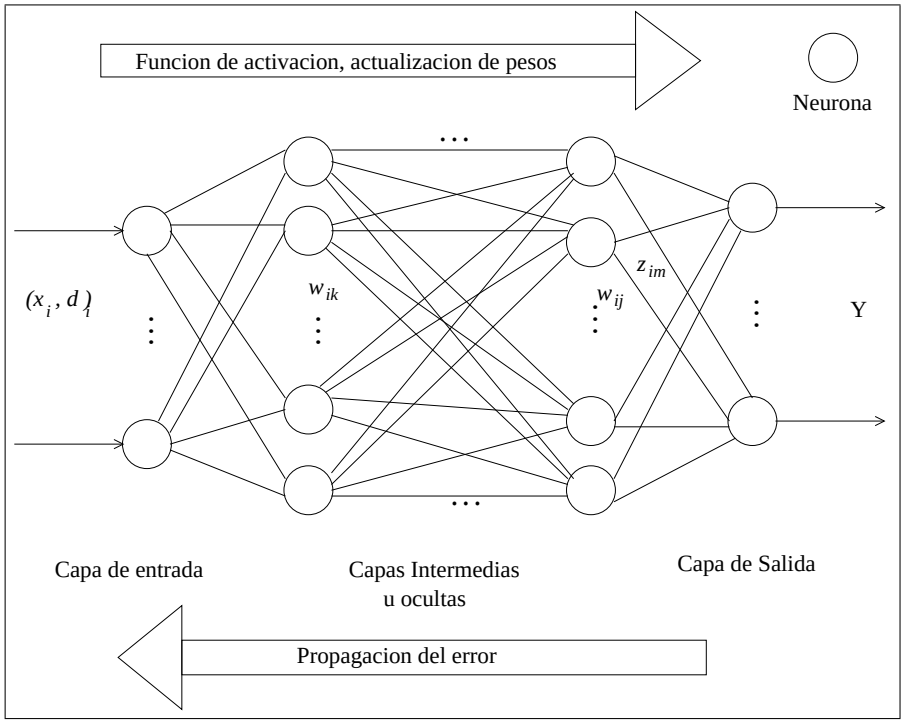

Figura 3.5: Perceptrón Multicapa.

de activación y representa la cantidad que se necesita para ajustar los pesos w_{ik} para cada entrada x_k. Este error se propaga hacia las capas anteriores, es decir, las neuronas de salida envían el error a las neuronas de las capas intermendias y éstas a su vez, propagan el error hacia las neuronas de entrada. Después se ajustan los pesos entre las neuronas de salida y las neuronas intermerdias utilizando δ_k. De manera similar, se ajustan los pesos entre las neuronas de entrada y las neuronas ocultas utilizando δ_k y la función de activación de x_k, donde

$$\delta_k = (d_k - y_k)F'(\mathbf{wx}) \tag{3.11}$$

y $F'(\mathbf{wx})$ es la derivada del error con respecto a la función de activación F.

Los pesos se actualizan mediante la siguiente ecuación:

$$\begin{cases} \mathbf{w}_1 = 0 \text{ ó arbitrario} \\ \mathbf{w}_{k+1} = \mathbf{w}_k + \eta\delta(k)\mathbf{x}_k \end{cases} \tag{3.12}$$

donde η es la tasa de aprendizaje.

A este algoritmo también se le conoce como el método del gradiente descendiente por pasos, porque se calcula la derivada del error y tiene como objetivo mover el vector de pesos de tal manera que el error tienda a cero.

El **Aprendizaje no supervisado** consiste en agrupar o detectar similitudes entre los datos de entrada. En este aprendizaje no hay ningún valor esperado. La red es capaz de extraer las características más significativas de los datos de entrada, o en base a la similitud entre los datos, hace agrupamientos en vecindades, sintetizando gradualmente los pesos de sus conexiones en cada paso del proceso de entrenamiento. Los datos de este aprendizaje pueden contener valores de entrada y valores de salida, pero a diferencia del aprendizaje supervisado, no hay información acerca de cuáles salidas corresponden a cada una de las entradas.

Las técnicas de aprendizaje no supervisado, además de extraer información a cerca de la distribución de los datos de entrada, son utilizadas como herramientas de preprocesamiento para el aprendizaje supervisado, porque mapean las características extraídas a espacios de menor dimensión. Estas técnicas se pueden dividir en dos categorías [Nelles, 2001]:

- *Análisis de componentes principales (Principal Component Analysis ó PCA). Esta técnica reduce la dimensión del conjunto de entrenamiento. El objetivo es obtener la información más significativa de los datos de entrada. Es decir, el análisis de componentes principales se usa para transformar los ejes de entrada y adaptarlos a los ejes originales para representar los datos. Sin embargo, esta técnica no siempre es conveniente, porque al reducir la dimensión del espacio se corre el riesgo de perder información relevante, ya que sólo se considera una distribución de los datos de entrada.*

- *Técnicas de agrupamiento (Clustering). Estas técnicas son utilizadas para encontrar grupos de patrones o muestras de datos similares que permitan incorporar información acerca de los datos de entrada. Las técnicas más usadas incluyen algunas redes neuronales que sirven para hacer agrupamientos. Por ejemplo, los Mapas Auto-organizados de Kohonen, la red de gas natural, la red de la teoría de resonancia adaptativa, y algunos algoritmos de agrupamiento como los k-means, el fuzzy c-means, o el algoritmo Gustafson-Kessel.*

El aprendizaje no supervisado maneja un conjunto de vectores de entrenamiento $\{\mathbf{x}_i : i = 1, 2, \ldots, m\}$ en $\mathbb{R}^n$. Estos vectores representan la características del problema que se quiere resolver y no están etiquetados. El objetivo de este aprendizaje es encontrar las similitudes o regularidades en el conjunto de entrenamiento para

formar grupos. Otra aplicación consiste en mapear los vectores de entrada $\mathbf{x}_i$ en conjuntos de menor dimensión, de tal manera que la relación topológica que existe en el conjunto de entrenamiento se preserva en el nuevo conjunto de solución. Así los pesos son optimizados con respecto a una función de criterio o de similitud.

En este aprendizaje se consideran técnicas de auto-organización o técnicas para descubrir estructuras en los datos de entrada. Algunas reglas de este tipo de aprendizaje son: *aprendizaje Hebbian* (utilizado en las redes de Hopfield) y el *aprendizaje de los Mapas de Características Auto-organizado* (que se usa en las redes de Kohonen) [Hassoon, 1995].

La regla de aprendizaje Hebbian consiste en modificar los pesos de acuerdo con algún criterio de correlación entre la actividad de las neuronas. Esta regla se basa en la hipótesis que supone que la eficiencia sináptica biológica cambia en proporción a la correlación entre la información pre- y post-sináptica, respectivamente [Hassoon, 1995]. Una red neuronal que emplea esta regla de aprendizaje es la red de Hopfield.

La red de Hopfield está formada por una capa con n neuronas conectadas completamente. La fase de entrenamiento de esta red consiste en almacenar un conjuto de vectores de entrada mediante la presentación repetida del mismo y la adaptación de los pesos. Posteriormente, en la fase de aprendizaje se presenta un vector ya almacenado y la red es capaz de recuperar el vector original. Si la información de entrada no coincide con ningún vector ya almacenado por estar incompleto o deformado, la red genera la salida más parecida. Debido a la forma de entrenamiento y aprendizaje de esta red, se dice que es una red auto-asociativa.

En la red de Hopfield cada neurona se caracteriza por un vector de pesos $\mathbf{w}_i = (w_{i1}, \ldots, w_{in})^T$ de dimensión n y los pesos de las conexiones son simétricos, es decir, el peso de la conexión entre la neurona i a la neurona j tienen el mismo valor $w_{ij} = w_{ji}$ para $i \neq j$. En otro caso si $i = j$ entonces $w_{ij} = 0$. Cada neurona recibe la misma entrada $\mathbf{x} \in \mathbb{R}^n$ y la salida de una neurona retroalimenta a las demás neuronas [Hilera and Martínez, 1995].

El entrenamiento de la red de Hopfield consiste en almacenar un conjunto de patrones de entrada y proyectar el espacio de entrada sobre la salida. El objetivo es reconstruir los patrones de entrada a partir de la información almacenada.

Durante el entrenamiento, el estado de la red de Hopfield en el tiempo t está determinado por el estado de todas la neuronas en ese instante, y se representa por el vector $Y^{(t)} = (y_1^{(t)}, \ldots, y_n^{(t)})$. Cuando se presenta el vector de entrada $\mathbf{x} = (x_1, \ldots, x_n)$ a la red, éste es considerado como estado inicial. Es decir, $Y^{(0)} =$

$(x_1, \ldots, x_n)^T$. Entonces, el nivel de activación de la neurona k en el tiempo $t+1$ se calcula como la suma del producto punto de los pesos y el estado anterior de la red. Se evalúa la función de activación y se obtiene la salida correspondiente. Este proceso se repite hasta que la red alcance un estado estable, es decir, cuando el estado de todas las neuronas no se modifique. Después de esto, la fase de aprendizaje consiste en presentar una nueva entrada a la red, y ésta debe ser capaz de asociar esta entrada a un patrón ya almacenado.

Otra regla de aprendizaje no supervisado es la regla que utilizan los mapas auto-organizados de Kohonen (*Self Organizing Map* o SOM). Esta regla se emplea para hacer agrupamientos basándose sólo en las características de los datos de entrada. Esta regla se considera como competitiva, ya que las neuronas compiten para ser activadas.

El objetivo de los mapas auto-organizados es formar mapas de características estableciendo una función de semejanza entre un conjunto de entrada y un espacio (normalmente de 1 ó 2 dimensiones). Es decir, que para vectores de entrada con características comunes se deben activar las neuronas situadas en zonas próximas. De esta manera, las neuronas que son vecinas en la topología de la red tienen un vector de pesos similar. Con ellos se garantiza que las neuronas vecinas representen regiones similares en el espacio de entrada. El aprendizaje de esta red se lleva a cabo mediante la adaptación y modificación de los pesos entre las conexiones adyacentes a las neuronas con la distancia mínima [Hassoon, 1995, Caudill and C., 1992, Fausett, 1994, Corchado et al., 2000].

Las redes SOM están formadas por una capa con n neuronas de entrada y otra capa con m neuronas de salida. Se le presenta un vector de entrada o patrón de entrenamiento $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ con n características. La red proyecta el espacio de entrada sobre la salida y en el entrenamiento realiza la proyección conservando el orden topológico. El objetivo es utilizar la regla de aprendizaje competitivo con respecto al conjunto de neuronas vecinas, de tal manera que el proceso de aprendizaje no sea global sino local.

Cada neurona se caracteriza por un vector de pesos $\mathbf{w}_i = (w_{i1}, w_{i2}, \ldots, w_{in})$ de dimensión n, y recibe la misma entrada $\mathbf{x} \in \mathbb{R}^n$ en la capa de entrada. Este vector pasa a la capa de salida donde se lleva a cabo el entrenamiento de la red. En la capa de salida, las neuronas compiten por activarse comparando la información de entrada con su vector de pesos asociado. Así, el vector más parecido a la información de entrada determina las neuronas ganadoras, y son las únicas que modifican los pesos asociados a sus conexiones.

Los pesos se actualizan mediante la siguiente ecuación:

$$w_k(t+1) = w_k(t) + \eta g(k,i)(x(t) - w_k(t)) \tag{3.13}$$

donde η es la tasa de aprendizaje y $g(k,i)$ es una función decreciente entre las neuronas k e i tal que $g(i,i) = 1$. Por ejemplo, para esta función comúnmente se utiliza la función gaussiana de la siguiente forma:

$$g(k,i) = e^{-||k-i||^2} \tag{3.14}$$

El **aprendizaje reforzado** es un proceso de prueba y error. Consiste en ajustar los pesos de la red en respuesta a la información evolutiva de los datos establecidos. Las reglas de este aprendizaje son mecanismos de búsqueda estocástica que maximizan la probabilidad de reforzar los extremos positivos de un conjunto de entrenamiento [Pal and Mitra, 1999].

El aprendizaje reforzado se emplea para maximizar los valores esperados de una función criterio llamada *señal reforzadora*. La idea básica tiene su origen en la psicología, en estudios realizados sobre el aprendizaje de animales. Si una acción es seguida de un premio tiende a ser reforzada; pero si esta acción es castigada, tiende a ser reprimida [Hassoon, 1995].

Este aprendizaje es similar al aprendizaje supervisado excepto que la salida deseada no se proporciona. En este caso, el valor esperado d_k se reduce a indicar mediante la señal de refuerzo r_k si la salida obtenida por la red y_k se ajusta a la deseada (éxito o fracaso). En función de ello se ajustan los pesos utilizando métodos de probabilidades. [Hilera and Martínez, 1995].

Existen diferentes formas de determinar un éxito o fracaso, es decir, que la salida sea la "correcta" o no. Estos métodos utilizan funciones de probabilidad y tienen como objetivo mapear el conjunto de entrada a la salida para maximizar el desempeño de la señal reforzadora.

Dado un conjunto de entrenamiento de la forma $\{\mathbf{x}, \mathbf{r}_k\}, k = 1, 2, \ldots, m$, donde $\mathbf{x} \in \mathbb{R}^n$ y $\mathbf{r}_k$ son los datos de evaluación o la señal de evaluación. Normalmente $\mathbf{r}_k \in \{-1, +1\}$ y reemplaza la función criterio. El objetivo no es asociar $\mathbf{x}$ con $\mathbf{r}_k$ como en el aprendizaje supervisado, sino que $\mathbf{r}_k$ indica a la neurona que está siendo entrenada con una entrada $\mathbf{x}$. Además $\mathbf{r}_k$ evalúa la asignación para las unidades de salida y_k correspondientes a la entrada $\mathbf{x}$ [Hassoon, 1995].

La regla de aprendizaje reforzado se presenta para unidades estocásticas y se expresa de la siguiente manera:

$$w_{k+1} = w_k + \rho(r_k)(d_k - < y_k >)(1 - < y_k >^2)x_k \tag{3.15}$$

donde

$$d_k = \begin{cases} y_k & \text{si } r_k = +1 \text{ (éxito)} \\ -y_k & \text{si } r_k = -1 \text{ (fracaso)} \end{cases} \tag{3.16}$$

$$\rho(r_k) = \begin{cases} \rho^+ & \text{si } r_k = +1 \\ -\rho^- & \text{si } r_k = -1 \end{cases} \tag{3.17}$$

con $\rho^+ >> \rho^- > 0$ y

$$< y_k >= (+1)P(y = +1) + (-1)P(y = -1) \tag{3.18}$$

donde P indica la probabilidad de que suceda y.

3.5 Resumen

Las Redes Neuronales Artificiales son sistemas de procesamiento de información, formadas por elementos no lineales llamados neuronas. El conjunto de neuronas que conforman la red neuronal se conectan de tal manera que la salida de una neurona cualquiera sirve, generalmente, como entrada de otras neuronas. El número de neuronas, la disposición de las mismas y las conexiones entre ellas determinan la arquitectura o topología de la red. La topología general de una red neuronal está formada por tres tipos de capas de neuronas: la capa de entrada, la capa intermedia u oculta y la capa de salida. Las conexiones entre estas capas también se caracterizan por valores llamados pesos.

Las Redes Neuronales Artificiales se clasifican de tal manera que consideran criterios básicos como la forma en que están dispuesta las neuronas, cómo son los datos de entrada y salida o la manera en que aprende la red.

El entrenamiento y aprendizaje de la red detemina el resultado final. La fase de entrenamiento consiste en determinar los valores de los pesos asociados a cada conexión. La fase de aprendizaje consiste en presentar nuevos datos de entrada y verificar que la red sea capaz de proporcionar un resultado.

Los aspectos anteriores definen la clasificación clásica de las Redes Neuronales Artificiales, la cual considera las siguientes clasificaciones: (a) Clasificación por la arquitectura de la red, (b) Clasificación por datos de entrada y salida y (c) Clasificación por forma de aprendizaje.

Capítulo 4

Sistema de Patrones de Software para Redes Neuronales Artificiales

En los capítulos anteriores se han tratado los conceptos básicos de Patrones de Software (capítulo 2) y Redes Neuronales Artificiales (capítulo 3), de manera separada. En este capítulo se describe la conjunción de estos temas, dando origen a un Sistema de Patrones Software como propuesta para resolver problemas utilizando Redes Neuronales Artificiales.

Anteriormente, el capítulo 3 describe las principales caracteríśticas que componen una Red Neuronal Artificial: sus componentes, los datos de entrada, la forma en que realiza una tarea, y su entrenamiento y aprendizaje. Estos criterios proporcionan una forma de clasificar a las redes neuronales. Sin embargo, esta clasificación no proporciona criterios para seleccionar una red neuronal.

El objetivo principal de esta tesis es proporcionar un Sistema de Patrones de Software para Redes Neuronales Artificiales que sirva como criterio de selección de redes neuronales. En este sistema, cada patrón toma en cuenta las características del problema y la red neuronal que se emplea para resolverlo. El sistema está compuesto por un conjunto representativo de redes neuronales, cada una de las cuales se describe utilizando la forma POSA [Buschmann et al., 1996]. Las redes que se describen como patrones de software en este capítulo son las siguientes:

- **La red de Mapas Auto-Organizados o SOM.** Este tipo de redes se utiliza para la clasificación de datos.
- **El Perceptrón Multicapa o MLP.** Estas redes utilizan la salida deseada

para establecer una relación entre los datos de entrada, cuando es importante la precisión de la respuesta.

- **Las redes de Funciones de Base Radial o RBF.** Estas redes utilizan la salida esperada para establecer una relación entre los datos de entrada, cuando no es importante el tiempo de respuesta.
- **Las redes de Hopfield.** Este tipo de redes se utiliza para la clasificación o reconstrucción de patrones a partir de la información almacenada.

4.1 El Patrón *Mapas Auto-organizados o SOM*

- **Nombre:** Mapas autoorganizados o SOM (*Self-Organizing Maps*).
- **Resumen:** El Patrón SOM describe una red neuronal artificial que se utiliza para la clasificación de patrones de datos [1].
- **También conocido como:** Redes de Kohonen o Mapas de Características Autorganizados.
- **Ejemplo:** En sistemas de reconocimiento de caracteres, se requiere identificar patrones de entrada que, agrupados, representen letras con diferentes fuentes. El problema es ¿cómo detectar automáticamente cada letra por un conjunto de puntos, de tal manera que en su conjunto se reconozcan como la letra que se trata (sobre una superficie bidimiensional)? Además, puede darse que algunos puntos dentro del conjunto no se encuentren propiamente en un sitio adecuado, o estén fuera del conjunto. Los puntos se presentan en diferentes cantidades y densidades [Fausett, 1994].
- **Contexto:** El patrón SOM se utiliza cuando los datos no están bien definidos o se tienen muchos datos de entrada.
- **Problema:** Encontrar características similares entre los datos de entrada sin supervisión.

  Las *Fuerzas* asociadas a este problema son:

  - Los datos representan la información obtenida de varias dimensiones.
  - Los datos no están bien definidos.

[1] Estos patrones representan las características de los datos a través de vectores

  - Se tiene una gran cantidad de datos en los que pueden existir dependencias entre ellos.
  - Se requiere obtener grupos de datos que representen toda la información.
  - No se requiere supervisión para llevar a cabo la clasificación [2].

- **Solución:** La Red SOM forma mapas de características de manera auto-organizada. El objetivo es establecer una correspondencia entre los datos de entrada y un espacio (normalmente de 1 ó 2 dimensiones) a través de una función de semejanza, de tal manera que ante vectores de entrada con características comunes se deben activar neuronas situadas en zonas próximas.

  El entrenamiento de la red consiste en la adaptación y modificación de los pesos entre las conexiones adyacentes a las *neuronas ganadoras*, es decir, las neuronas que hayan tenido la distancia mínima. De esta manera, vectores de datos de entrada similares activan y generan un orden global.

  El aprendizaje de la red termina cuando se reducen las zonas de neuronas activadas por vectores de datos de entrada parecidos. Es necesario repetir el proceso de entrenamiento hasta refinar el mapa topólogico.

- **Estructura:** Las redes SOM poseen dos capas de neuronas con conexiones entre ellas: n neuronas en la capa de entrada y m neuronas en la capa de salida, con $n << m$ y $n * m$ conexiones. La figura 4.1 muestra la estructura de una red SOM.

  Participantes:

  - Neuronas de entrada: Estas neuronas pertenecen a la capa de entrada y sólo reciben los datos para el entrenamiento de la red. Después pasan los datos a las neuronas de la capa de salida.
  - Neuronas de salida: Estas neuronas se encargan de llevar a cabo (a) el entrenamiento de la red, identificando las diferencias entre las distancias euclidianas y modificando los pesos de las conexiones adyacentes a las neuronas ganadoras y (b) el aprendizaje de la red, que consiste en establecer las diferentes categorías entre los datos en función de su semejanza.

- **Dinámica:** Suponemos un vector de entrada o patrón de entrenamiento con n características y representado por el vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ en el espacio

[2]Esta red también trabaja con aprendizaje supervisado. Sin embargo, en esta tesis no se considera tal caso. Para mayor referencia consultar el libro de Kohonen [Kohonen, 1990]

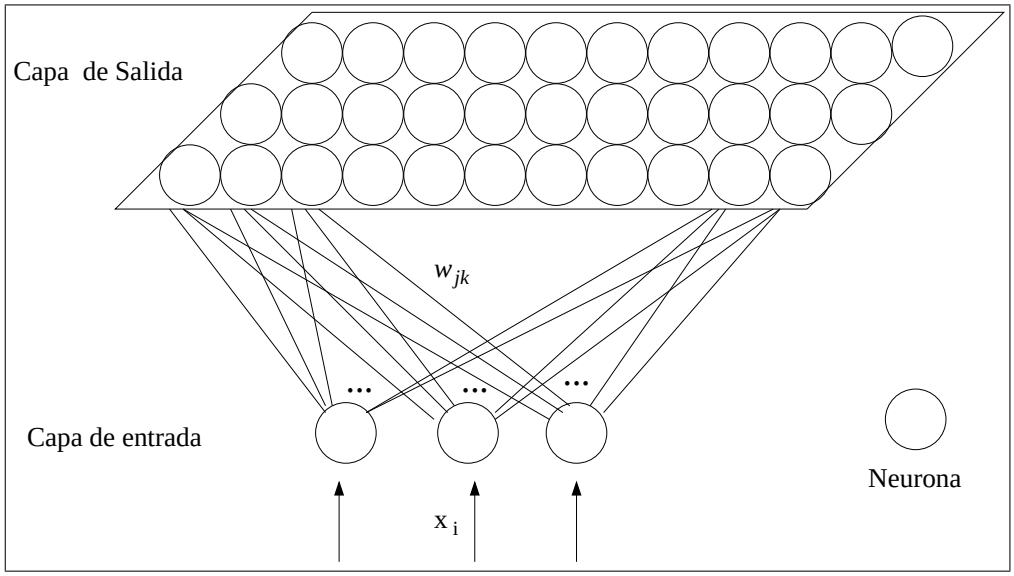

Figura 4.1: Estructura de la red SOM

de dimensión n. La red proyecta el espacio de entrada sobre la salida, y en el entrenamiento realiza la proyección conservando el orden topológico. Para ello se utiliza la regla de aprendizaje competitivo [véase sección 3.4.3, pág. 37] con respecto al conjunto de neuronas vecinas, de tal manera que el proceso de aprendizaje no sea global sino local.

Cada neurona k se caracteriza por un vector de pesos $\mathbf{w}_k = (w_{k1}, w_{k2}, \ldots, w_{kn})$ de dimensión n, y recibe la misma entrada $\mathbf{x} \in \mathbb{R}^n$ en la capa de entrada. Este vector pasa a la capa de salida en donde se lleva a cabo el entrenamiento de la red.

En la capa de salida, las neuronas compiten por activarse comparando los datos de entrada con su vector de pesos asociado. Así, el vector de pesos más parecido al vector de datos de entrada determina las neuronas ganadoras y son las únicas que modifican los pesos asociados a sus conexiones. La figura 4.2 describe el diagrama de secuencia que emplea el algoritmo SOM.

El algoritmo de aprendizaje que utiliza la red SOM se describe a continuación:

1. Crear el vector de pesos $\mathbf{w}_k = (w_{k1}, w_{k2} \ldots, w_{kn})$ para cada conexión entre las neuronas.
2. Fijar la zona inicial de vecindad entre las neuronas de la capa de salida.

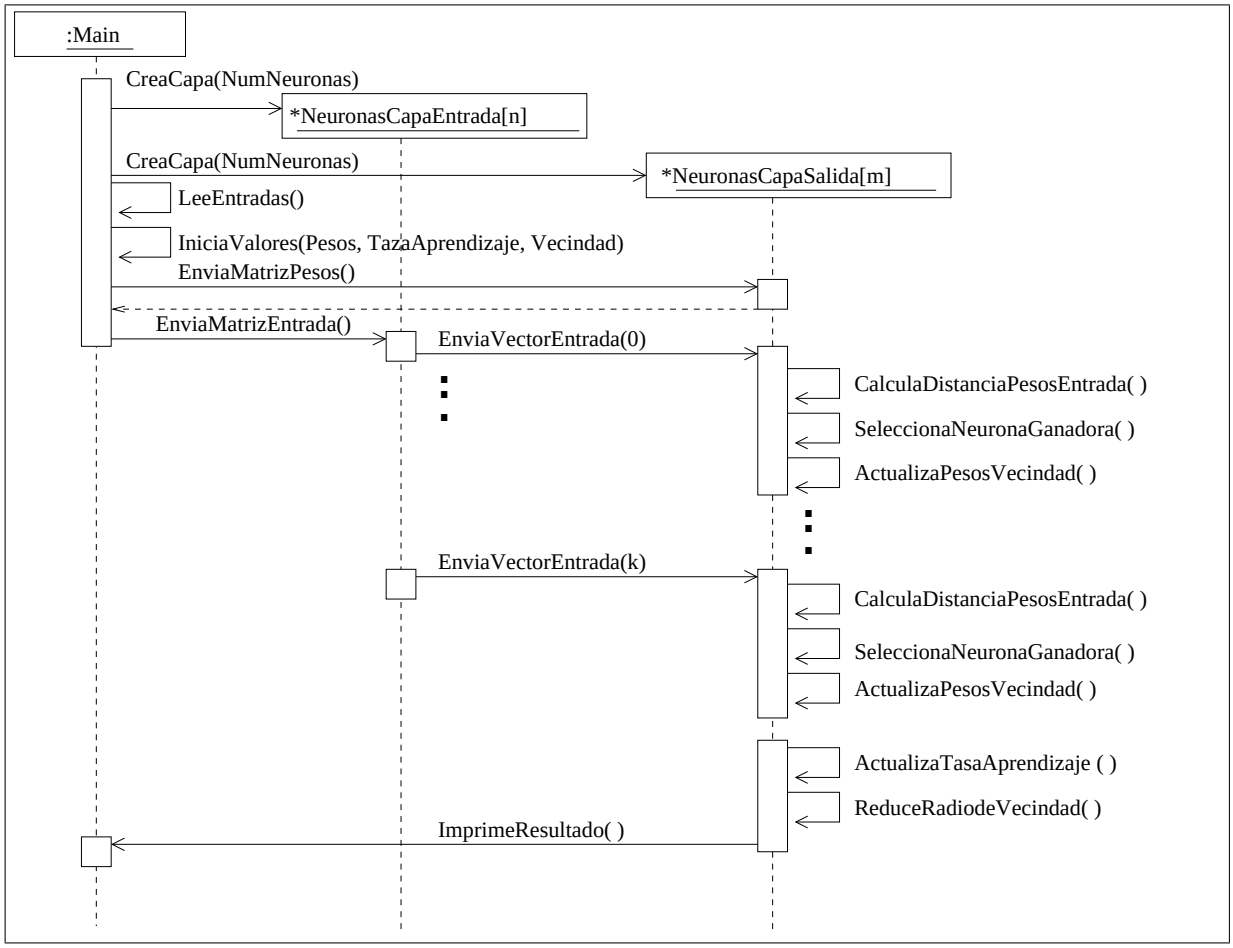

Figura 4.2: Diagrama de secuencia del algoritmo SOM

3. Presentar el vector de entrada $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$ en la capa de entrada.
   (a) Para cada neurona de salida k caracterizada por su vector de pesos $\mathbf{w}_k$, determinar su estado de activación Y_k, definido por la distancia euclidiana entre $\mathbf{x}$ y $\mathbf{w}_k$: $Y_k = \mathbf{w}_k\mathbf{x}$.
   (b) Seleccionar como neurona ganadora, aquélla neurona cuyo resultado entre la distancia del vector de datos de entrada y el vector de pesos haya sido el menor.
   (c) Una vez seleccionada la neurona ganadora i, se actualizan los pesos de las conexiones entre la neurona ganadora y la neurona de entrada con la siguiente regla de aprendizaje.

$$\mathbf{w}_k(t+1) = \mathbf{w}_k(t) + \eta g(k,i)(\mathbf{x}(t) - \mathbf{w}_k(t)) \tag{4.1}$$

   donde η es la tasa de aprendizaje y $g(k,i)$ es una función entre las neuronas k e i tal que $g(i,i) = 1$.
   Generalmente se utiliza la función gaussina para $g(k,i)$ de forma que:

$$g(k,i) = e^{-||k-i||^2} \tag{4.2}$$

   A esta función también se le conoce como malla, ya que define cómo quedan agrupados los datos de entrenamiento.
4. Actualizar la tasa de aprendizaje.
5. Reducir el radio de vecindad.
6. El proceso se repite, a partir del paso 3, volviendo a presentar todo el conjunto de entrenamiento t veces.

Las propiedades más importantes de esta red son: (a) el concepto de aprendizaje en un vecindario cercano a la neurona ganadora, y (b) la forma en que se modifican los pesos para que destaquen propiedades topológicas en la proyección de características.

- **Implementación:**

  Para la implementación de una red SOM se consideran los siguientes aspectos:

  – La estructura de la red SOM consta de dos capas de neuronas y recibe un vector de entrada.
  – La capa de entrada sólo proporciona el almacenamiento del vector de entrada.

- El proceso es secuencial. Se proponen los siguientes tipos de varibles: renglones, columnas, número de entradas, número de salidas.
- Para cada neurona de la capa de salida, se calcula la distancia entre el vector de salida de la capa de entrada y el vector de pesos.
- Después de calcular la distancia, ésta se almacena. Cuando todas las neuronas de salida hayan calculado su distancia, se obtiene la matriz de distancias.
- En el entrenamiento se almacenan los índices para localizar la neurona ganadora. Es decir, se identifica la neurona con distancia mínima y su índice se utiliza para identificarla.

El aprendizaje de la red SOM se divide en dos fases. La primer fase consiste en entrenar a la red presentando los vectores de entrada hasta que la vecindad se haya reducido, es decir, que gane la misma neurona. En la segunda fase, se prueba la red mediante comprobar que después de cierto número de ciclos las neuronas ganadoras siempre sean las mismas, es decir, que todos los vectores que han sido asociados a su neurona ganadora (aunque no necesariamente a neuronas diferentes) continúen ejecutándose durante varios ciclos para garantizar la estabilidad de la red.

En la fase de entrenamiento se inicializa la matriz de pesos de manera aleatoria. Estos valores corresponden a las conexiones entre las neuronas de la capa de entrada y las neuronas de la capa de salida. Durante el entrenamiento, la red recibe los vectores de entrada y tiene que auto-adaptarse, modificando los valores de su matriz de pesos cada vez que recibe otro vector de entrada. Para ello se propone combinar dos rutinas [Freeman and Skapura, 1992]. La primer rutina calcula la distancia entre el vector de entrada y el vector de pesos de una neurona específica de la capa de salida. La segunda rutina se utiliza para llamar a la primer rutina, pero para cualquier neurona de la capa de salida.

Para determinar la vecindad se consideran dos funciones: una función que regrese un valor falso o verdadero para indicar si una neurona está dentro de la vecindad de la neurona ganadora, y la otra función actualiza los valores de las conexiones entre las neuronas de salida.

- **Ejemplo resuelto:**

  En el problema de reconocimiento de caracteres se requiere identificar patrones de entrada que, agrupados, representan letras con diferentes fuentes. Este problema consiste en detectar automáticamente cada letra por un conjunto

de puntos, de tal manera que en su conjunto se reconozcan como la letra que se trate.

Se seleccionan siete letras (A, B, C, D, E, J y K) con 3 fuentes diferentes. Cada letra se representa por una matriz de 7×9 puntos y está formada por valores binarios. Dentro de esta matriz se puede dar que algunos puntos no estén en un sitio adecuado, o estén fuera del conjunto. Para resolver el problema, se utiliza una red SOM que agrupa los datos de entrada sin supervisión [Fausett, 1994].

La red SOM está formada por: *(a)* 63 neuronas de entrada, que reciben los datos de cada letra; *(b)* 63 neuronas de salida, tomando en cuenta que la red SOM tiene una estructura lineal. Es decir, que la vecindad de la neurona ganadora k consiste en actualizar los pesos de sus vecinas $k-1$ y $k+1$. Para este problema se utiliza una tasa de aprendizaje igual a 0.6. Este valor va decreciendo durante el entrenamiento de la red hasta obtener un valor final de 0.01 [Fausett, 1994].

Al iniciar el proceso de entrenamiento, únicamente las neuronas vecinas a la neurona ganadora actualizan sus pesos. Sin embargo, es muy posible que estas neuronas no tengan vectores de pesos cercanos a los valores del vector de entrada. Es por esto que durante el proceso de entrenamiento los pesos se van ajustando, de tal manera que identifiquen 21 grupos. Se considera que cada grupo representa una letra [Fausett, 1994].

- **Variantes:** LVQ (*Learning Vector Quantization*) es una red neuronal SOM utilizada para clasificar. Se basa en el principio de formación de mapas topológicos para establecer características comunes en la información de entrada. A diferencia de la red SOM tradicional, esta red utiliza el aprendizaje supervisado para actualizar el peso de las neuronas ganadoras. Durante el entrenamiento, se proporciona un conjunto de vectores de entrenamiento con una clasificación conocida, de tal manera que las neuronas compiten por activarse y las neuronas ganadoras son las únicas que ajustan sus pesos. En la fase de aprendizaje, la red LVQ clasifica un vector de entrada asignándolo a la clase de la neurona cuyo vector de pesos sea el más cercano al vector de entrada. El objetivo de LVQ es dividir el espacio de entrada en regiones que representan una o más neuronas de salida y cada una de ellas forma una clase o categoría [Hilera and Martínez, 1995, Fausett, 1994].

- **Usos conocidos:** En general, la red SOM se utiliza en agrupamientos (*clustering*) cuando se desea determinar a qué grupo pertenecen los datos de entrada. Comúnmente se usa en la extracción o identificación de características relevantes de una señal. También en la reducción de dimensiones para proyectar

y visualizar espacios de señales de n dimensiones a espacios de uno o dos dimensiones [Corchado et al., 2000].

- Las redes SOM se han empleado en el **procesamiento de lenguaje**. El objetivo es convertir el habla a texto escrito en tiempo real, independientemente de la persona que hable y del ruido del ambiente. El sistema inicia con una señal procesada a través de un micrófono y un preprocesador analógico/digital. La señal se convierte en la representación de 15 canales que cubren un rango de frecuencia de 200 Hz a 5 kHz. Estos canales constituyen un vector de entrada $\mathbf{x}(t)$ de dimensión 15 y representa los diferentes fonemas del lenguaje, para un mapa de características. Se entrena a la red de tal manera que para fonemas muy parecidos se activen neuronas de salida próximas creando un *mapa fonotópico* [Hassoon, 1995].
- Las redes SOM se han utilizado en la **codificación de datos**. El objetivo es diseñar códigos para reducir o comprimir el tamaño de la información. Por ejemplo, en la compresión de imágenes en color se trata de averiguar qué pixeles de la imagen pueden representarse por el mismo color, con el fin de utilizar menos bits por pixel. Generalmente esta aplicación se utiliza en la conversión de imágenes originales con pixeles de 2^{24} diferentes colores a imágenes para monitores VGA con pixeles de 2^8 colores (8 bits por pixel y el color de un pixel se codifica con 24 bits). En este problema se usa una red SOM para establecer los 2^8 colores más frecuentes que están presentes en la imagen original. Se divide cada imagen en tres imágenes, de tipo monocolor que representan las versiones en rojo, verde y azul. Estas tres combinaciones dan lugar a la imagen original. Cada pixel de estas versiones se codifica con 8 bits, por lo que existen 256 posibles grados de intensidad de rojo, verde y azul.

  Se usa una red SOM con tres neuronas en la capa de entrada y 256 neuronas en la capa de salida. Estas neuronas corresponden a los 256 colores que deben identificar la red. En la capa de entrada, la red recibe los valores correspondientes al grado de intensidad de los colores básicos: rojo, verde y azul de cada píxel de la imagen. Después del entrenamiento, los pesos de las conexiones de cada neurona en la capa de salida representan la combinación de los colores verde, rojo y azul para cada uno de los 256 colores, de manera que se obtiene un *mapa cromato-tópico* de la imagen aprendida [Hilera and Martínez, 1995].
- Las redes SOM se han utilizado para simular la formación de *mapas*

somatosensoriales entre los receptores tácticos de la superficie de una mano y una corteza artificial formada por un capa de 30×30 neuronas. El conjunto de entrenamiento consiste en la actividad de los patrones de un conjunto de receptores tácticos cubriendo la superficie de una mano. El mapa de características se inicia de manera aleatoria y los puntos se seleccionan de acuerdo a su distribución de probabilidad definida por cinco regiones que corresponden a los dedos [Hassoon, 1995].

- **Consecuencias:**

  Ventajas

  - La red SOM aprende rápidamente, es decir, ante un dato de entrada nuevo, la red puede clasificarlo en un tiempo relativamente corto.
  - Cuando no se cuenta con la información suficiente de entrada, la red SOM es todavía capaz de llevar a cabo una clasificación.
  - La red SOM agrupa la información de tal manera que reduce la dimensión de los datos de entrada.
  - La red SOM preserva la topología de los datos de entrada, es decir, no modifica la información que recibe, solo la etiqueta.

  Desventajas

  - La red SOM que utiliza aprendizaje sin supervisión necesita cada vez repetir todo el proceso de aprendizaje para clasificar un nuevo dato de entrada.

4.2 Patrón *Perceptrón Multicapa o MLP*

- **Nombre:** Perceptrón Multicapa o MLP (*Multi–Layer Perceptron*)
- **Resumen:** El Patrón MLP describe una red neuronal artificial que utiliza la salida deseada para establecer una relación entre los datos de entrada.
- **También conocido como:** Retropropagación (*Backpropagation*).
- **Ejemplo:** En medicina, se requiere detectar la oclusión coronaria o infarto al miocardio agudo, la cual es una enfermedad difícil de diagnosticar. El

problema consiste en averiguar si un paciente puede sufrir esta enfermedad comparando su historial clínico con el historial clínico de pacientes que sí la han padecido. El resultado debe ser lo más preciso posible, considerando a menudo historiales muy similares que no se relacionan con la enfermedad del paciente [Hassoon, 1995].

- **Contexto:** El patrón MLP se utiliza cuando se tiene grandes cantidades de datos que están relacionados y se conoce de antemano la salida. Además es importante la precisión de la respuesta pero no el tiempo de ejecución.

- **Problema:** Establecer la relación entre los datos de entrada y la salida esperada.

  Las *Fuerzas* asociadas a este problema son:

  - Existe una gran cantidad de datos con dependecias entre ellos.
  - Los datos pueden tener información redundante.
  - Es necesario conocer la salida esperada para llevar a cabo el entrenamiento de la red.
  - Se requiere que la solución sea precisa.
  - No es importante el tiempo de respuesta.

- **Solución:** La Red MLP establece una relación entre los datos de entrada y los datos de salida a partir de ciertos valores esperados. Esta red está formada por neuronas artificiales que calculan el producto punto de los pesos por las entradas más unos pesos extras llamados umbrales o polarización (*bias*). Con el resultado que se obtiene del producto punto se evalúa la función de activación. Las redes MLP están formadas por varias capas de neuronas, así que la salida de las neuronas en una capa forma la entrada de la capa siguiente. Durante el aprendizaje, la red calcula los pesos usando el algoritmo de propagación hacia atrás. El objetivo es utilizar el error que existe entre la salida de la red y el valor esperado para ajustar la matriz de pesos y minimizar el error.

- **Estructura:** Las redes MLP están formadas por tres tipos de neuronas: n neuronas de entrada, m neuronas ocultas o intermedias y k neuronas de salida, $n \leq m \leq k$. En la figura 4.3 se muestra la estructura general de las redes MLP.

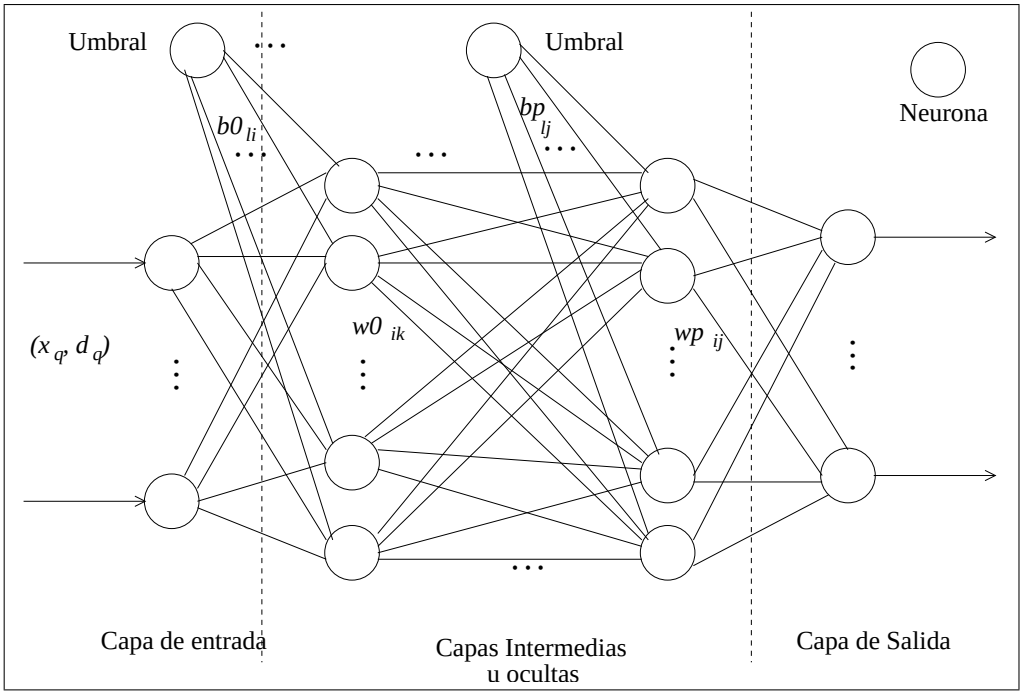

Figura 4.3: Estructura general de las redes MLP. Los pares (x_q, d_q) son los valores de entrada que reciben las redes MLP. x_q representa el vector de entrada q y d_q es la salida esperada para el vector de entrada x_q. $w0_{ik}$ es el peso de la conexión entre las neuronas i y k en la capa oculta 0 y wp_{ij} es el peso de la conexión entre las neuronas i y j en la capa oculta p. Así $b0_{li}$ es el umbral de la conexión entre las neuronas i y l en la capa oculta 0 y bp_{lj} es el umbral de la conexión entre las neuronas j y l en la capa oculta p.

Participantes:

- Neuronas de entrada: Estas neuronas pertenecen a la capa de entrada y se encargan de recibir los datos para el entrenamiento de la red. Es decir, los valores de entrada y los valores de salida deseados, además de enviar estos datos a las neuronas de la capa oculta y de la capa de salida respectivamente.
- Neuronas ocultas: Estas neuronas calculan el producto punto de las entradas y los pesos asociados a sus conexiones, y utilizan ese valor para obtener la función de similitud o función de activación. Su resultado se envía a la siguiente capa oculta o a la capa de salida.
- Neuronas de salida: Estas neuronas reciben el valor de la salida esperada y el resultado de las capas ocultas y proporcionan el resultado de la red. El resultado se obtiene de comparar el valor obtenido por la red con un valor esperado. Si el resultado es diferente del valor esperado, se calcula el error que comete la red y se envía ese valor a las neuronas ocultas para modificar los pesos asociados a cada neurona, así como el valor de los umbrales.

- **Dinámica:** Suponemos que un vector de entrada o patrón de entrenamiento tiene n características y está representado por el vector $x = (x_1, x_2, \ldots, x_n)$ en el espacio de dimensión n. La red proyecta el espacio de entrada sobre la salida a través de las neuronas ocultas. En el entrenamiento se lleva a cabo la modificación de los pesos entre las conexiones. El objetivo es ajustar la matriz de pesos para minimizar el error entre la salida de la red y los valores esperados. Existen varios métodos para resolver este problema. Por ejemplo: el método del gradiente descendiente por pasos, el método de Newton y el método del gradiente conjugado [Hu and Hwang, 2002]. En este trabajo se considera el método del gradiente descendiente por pasos, que consiste en utilizar el error entre la salida de la red y la salida esperada para modificar los pesos de las capas anteriores [véase sección 3.4.3, pág. 33]. Después del entrenamiento, la red MLP solo utiliza la fase de propagación hacia adelante para realizar la prueba del algoritmo y utilizar los valores obtenidos en el entrenamiento.

  Una neurona k se caracteriza por un vector de pesos $\mathbf{w}_k = (w_{k1}, w_{k2}, \ldots, w_{kn})$ de dimensión n, recibe el vector de entrada $\mathbf{x}$ y tiene la misma función de activación, la cual generalmente es una función sigma o una función tangente hiperbólica. La salida de las neuronas de una capa forma la entrada de la capa siguiente.

En la capa de salida, las neuronas calculan el resultado de la red considerando la diferencia entre la salida de las neuronas y el valor esperado. De ello depende que se modifiquen los pesos, de tal manera que cuando los valores son diferentes se obtiene el error que comete la red, regresándolo a las neuronas ocultas para actualizar los pesos y los umbrales. Esto se muestra en la figura 4.4.

El algoritmo de aprendizaje que utiliza la red MLP para el entrenamiento se describe a continuación:

1. Crear el vector de pesos $\mathbf{w}_k = (w_{k1}, w_{k2}, \ldots, w_{kn})^T$ para cada conexión entre las neuronas.
2. Presentar el vector de entrada $\mathbf{x}_q = (x_1, x_2, \ldots, x_n)$ y la salida deseada $\mathbf{d}_q = (d_1, d_2, \ldots, d_n)$ en la capa de entrada.
3. Cada neurona en la capa de entrada recibe los valores de entrada y los pasa a las neuronas de la capa oculta.
   (a) Las neuronas de la capa oculta calculan la suma del producto del vector de entrada por su vector de pesos y evalúa la función de activación para calcular la salida, que se envía a la siguiente capa.
   (b) El paso anterior se repite para todas las neuronas de las capas ocultas. La última capa oculta envía el resultado a las neuronas de la capa de salida.
   (c) Las neuronas en la capa de salida reciben el valor esperado correspondiente al vector de entrada y calcula el error que comete la red. La diferencia entre la salida de la red y el valor esperado se denomina *delta* y se utiliza para actualizar los pesos y los umbral de la conexiones en las capas ocultas.
   (d) Las neuronas en las capas ocultas reciben el valor delta de la capa de salida, obtienen la suma de delta multiplicado por la derivada de su función de activación para calcular el error que comente la red y ajustar los pesos y los umbrales para cada conexión.
   (e) Las neuronas de salida actualizan los pesos y los umbrales de sus conexiones.
4. El proceso se repite t veces.

Para la fase de prueba, que se lleva a cabo después del entrenamiento de la red MLP, sólo se utiliza la propagación hacia adelante del algoritmo de entrenamiento. Esto se muestra en la figura 4.5:

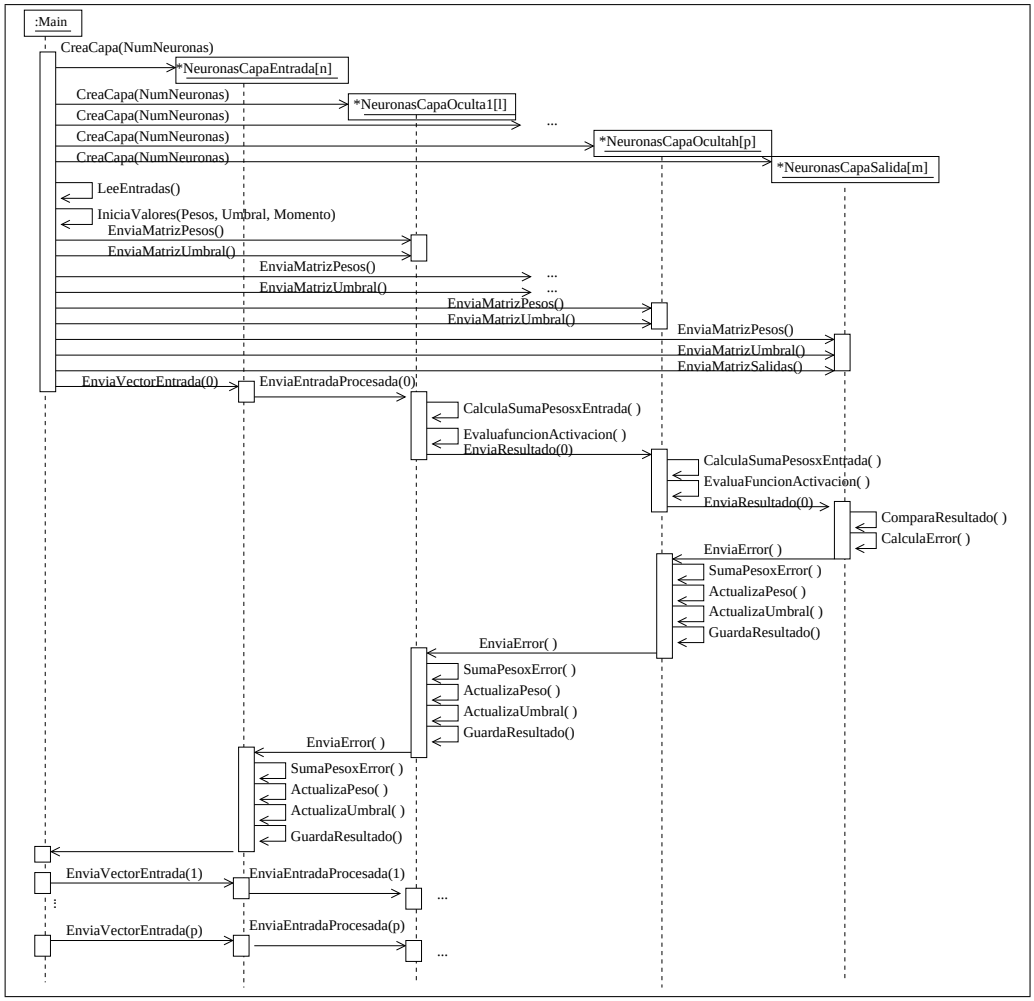

Figura 4.4: Diagrama de secuencia para fase de entrenamiento del MLP

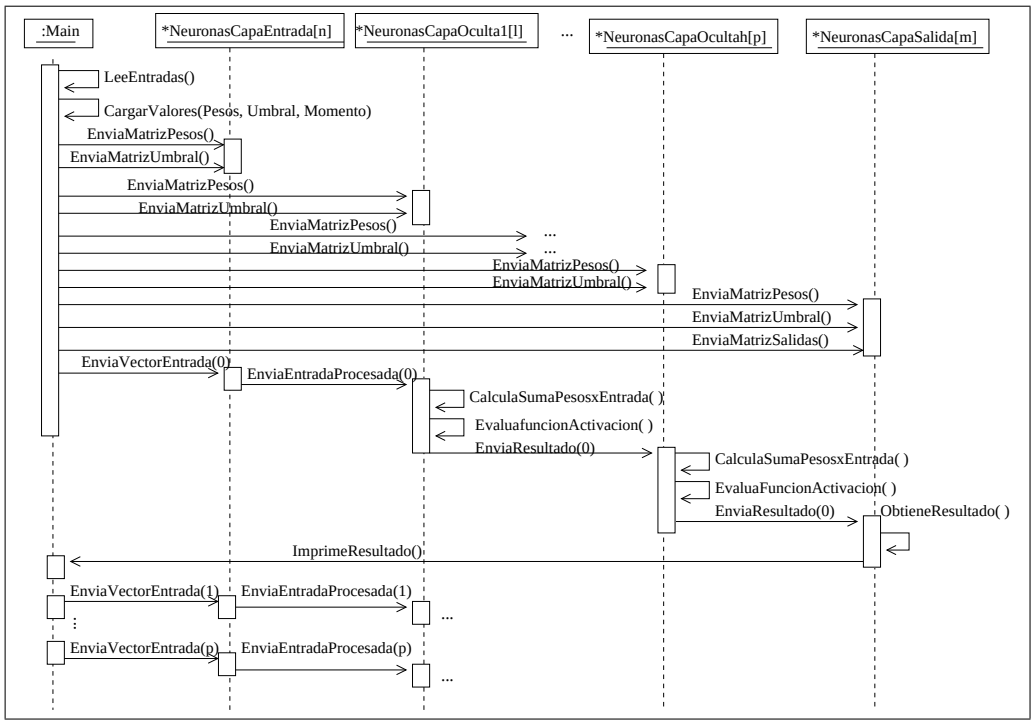

Figura 4.5: Diagrama de secuencia para la fase de prueba del MLP

1. Iniciar los pesos y los umbrales con el resultado del algoritmo de entrenamiento.
2. Para cada vector de entrada:
   (a) Obtener la suma de los pesos por las entrada más los umbrales.
   (b) Enviar el resultado a la siguiente capa oculta.
   (c) Aplicar la función de activación.
3. Repertit el paso 2 en cada capa hasta la capa de salida.

- **Implementación:** Para la implementación de una red MLP se consideran los siguientes aspectos [Freeman and Skapura, 1992]:

  1. La estructura de la red MLP consta de una capa de neuronas de entrada, m capas de neuronas ocultas y una capa de neuronas de salida. Sin embargo, se sabe que cualquier función continua puede resolverse utilizando un MLP con tres capas [Luo and Unbehauen, 1997]: una capa de entrada, una capa oculta y una capa de salida. Por lo tanto, en esta implementación solo se considera una capa oculta ($m = 1$).
  2. La información en la red MLP fluye en dos direcciones, pero en una sola dirección a la vez. Es decir, durante el entrenamiento la red recibe los datos en la capa de entrada y los envía a la capa de salida a través de las neuronas de la capa oculta. En la capa de salida se calcula el error que comete la red y este valor se envía hacia la capa de entrada. Después del entrenamiento, en la etapa de prueba, los datos sólo se envían hacia la capa de salida.
  3. Todas las neuronas de la capa oculta y de la capa de salida tienen la misma función de evaluación. Por ejemplo una función sigmoide.
  4. Es necesario utilizar una variable *momento*, la cual sirve para controlar en qué iteración se actualizan los pesos.
  5. El entrenamiento de la red MLP empieza cuando se presentan los datos de entrada. La red recibe los datos en la capa de entrada y realiza las siguientes operaciones:
     (a) Localizar la primer neurona de la capa oculta.
     (b) Iniciar el valor de los pesos, los umbrales y el momento igual a cero.
     (c) Calcula el producto del vector de entrada por el peso de la conexión de la neurona en la capa oculta.
     (d) Guardar el producto total acumulado.

(e) Repite los pasos 5c y 5d para cada conexión de entrada.

(f) Calcula la salida de las neuronas evaluando la función de activación:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{4.3}$$

donde x = entrada total.

(g) Repite los pasos 5b a 5f para cada neurona en esta capa.

(h) Repite los pasos 5a a 5g para cada capa en la red.

6. Con el proceso anterior se obtiene la salida de la red. Esta salida se compara con el valor esperado correspondiente al vector de entrada y se obtiene el error que comete la red. Ese error se envía a las neuronas de la capa oculta, a fin de actualiazar los pesos de sus conexiones mediante los siguientes pasos:

   (a) Localizar la primer neurona oculta de la capa anterior a la capa de salida.

   (b) Establecer el error total igual a cero.

   (c) Calcular el producto del peso de la conexión de salida y el error proporcionado por la neurona de salida o de la capa superior.

   (d) Sumar el producto del error acumulado.

   (e) Repetir los pasos 6c y 6d para cada conexión de salida.

   (f) Multiplicar el error acumulado. Este valor es la salida de la neurona en la capa oculta producida durante la operación hacia adelante.

   (g) Calcular el valor de los pesos para la conexión de la neurona y agregar una fracción del error acumulado.

   (h) Actualizar los pesos de cada conexión.

   (i) Actualizar los umbrales de cada neurona.

   (j) Guardar el valor de los pesos (el nuevo valor de los pesos y el viejo valor de cada conexión).

   (k) Repetir los pasos 6h a 6j para cada conexión de la neurona.

   (l) Repetir los pasos 6h a 6k para cada neurona de esta capa.

   (m) Repetir los pasos 6h a 6l para cada capa en la red.

7. Se utiliza una estructura para las conexiones que contenga el valor de los pesos y los umbrales, de tal manera que facilite el acceso a esos valores durante la fase propagación hacia adelante y otra estructura que contenga el valor de los errores para la fase de propagación hacia atrás.

8. Para la progación hacia adelante se proponen las siguientes funciones:
   (a) Una función que recibe los valores de entrada.
   (b) Una función que calcule la suma del peso por la entrada, evalúe la función de activación y guarde el resultado.
   (c) Una función que propague el resultado de la neruona en una capa a la capa siguiente.
   (d) Una función que obtenga y guarde el resultado de la red.
9. Para la propagación del error hacia atrás se proponen tres funciones:
   (a) Una función que calcule el error de la neuronas en la capa de salida.
   (b) Una función que propague el error de una capa hacia la capa anterior.
   (c) Una función que utilice el error para actualizar el valor de los pesos.
10. Considerar una función que calcule el valor *delta* para actualizar los pesos (véase la ecuación 3.11 del capítulo anterior).
11. Para concluir la implementación se propone una función principal que realice las llamada a las funciones anteriores.

- **Ejemplo resuelto:** El problema consiste en diagnosticar si un paciente que ha ingresado a la unidad de cuidado coronario puede sufrir un infarto. Para ello se compara el historial clínico de pacientes que han sufrido esta enfemedad con el historial del paciente que no ha padecido un infarto. El resultado debe ser lo más preciso posible, considerando que a menudo historiales muy similares no se relacionan con la enfermedad que puede sufrir un paciente [Hassoon, 1995].

  Para este problema se utiliza una red MLP con el objetivo de obtener mejores resultados que los diagnósticos clínicos o las aproximaciones de sistemas expertos tradicionales. Esta red está formada por cuatro capas de neuronas conectadas completamente. La red se implementa mediante 10 neuronas en la capa de entrada, que se conectan con 10 neuronas de la primer capa oculta. Las neuronas de la primer capa oculta se conectan con 10 neuronas de la segunda capa oculta, y las neuronas de la segunda capa oculta se conectan con las neuronas de la capa de salida. Para el aprendizaje de la red, todas la neuronas utilizan una función de activación sigmoide y el algoritmo de propagación hacia atrás [Hassoon, 1995].

  El conjunto de entrenamiento contiene la información de 356 pacientes que han sido admitidos en la unidad de cuidado coronario. De los 356 pacientes, 236 no han sifrido un infarto y 120 lo han padecido. Para el entrenamiento de la red

se selecciona aleatoriamente un conjunto de datos tomado de la mitad de los pacientes que han sufrido un infarto (118 pacientes) y la mitad de los pacientes que no lo han sufrido (60 pacientes). El resto de la información se utiliza para verificar el funcionamento de la red en la fase de prueba [Hassoon, 1995].

Los datos de cada paciente se integran por 20 variables que son utilizadas para predecir un infarto. Algunas de estas variables son edad, sexo, náuseas y vómito, respiración agitada, diabetes, hipertensión, angina, entre otras. Estas variables se eligen de un conjunto de 41 variables registradas en los pacientes que ingresaron a la unidad de emergencia de cuidado coronario. La mayoría de la variables pueden ser codificadas en binario, de tal manera que 1 representa la presencia de un síntoma y 0 representa su ausencia. Otras variables, como la edad del paciente, se pueden codificar con valores entre 0.0 y 1.0. La salida de la red consiste de valores binarios: 1 confirma la presencia de un infarto y 0 la ausencia del infarto [Hassoon, 1995].

Después de entrenar a la red, se utiliza la información de los 178 pacientes restantes del conjunto de entrenamiento (118 pacientes no han sufrido un infarto y 60 si lo han padecido). Algunos resultados de la red identificaron correctamente la presencia de un infarto con una tasa de 92% y su ausencia con un tasa de 96%. Este resultado mejora el 88% de certidumbre que proporcionan los diagnósticos tradicionales; el 26% restante se considera de falsa alarma [Hassoon, 1995].

- **Variantes:**

  Algunas variantes del algoritmo MLP para introducir más velocidad de convergencia son: Quickprop (o *quickpropagation*) y Rprop (o *resilient backpropagation*). Quickprop utiliza el error que comente la red para calcular el cambio de los pesos de manera local utilizando mínimos cuadrados. Rprop considera el error producido en la vecindad de la neurona local para hacer el cambio de los pesos [Pal and Mitra, 1999].

- **Usos conocidos:** Generalmente las redes MLP se utilizan en el reconocimiento de patrones, procesamiento de señales, reconocimiento de lenguaje, diagnósticos médicos, modelado de sistemas no lineales, etc.[Hassoon, 1995].

  – Una de las primeras aplicaciones de la red MLP es en el **procesamiento del lenguaje** para convertir texto en inglés a habla: NETtalk [Hassoon, 1995]. NETtalk es un sistema que consiste de 2 módulos: un módulo para el mapeo de la red y el otro módulo está formado por un sintetizador de habla comercial. Para este problema se utiliza una red

neuronal MLP que tiene 80 neuronas en la capa oculta y 26 neuronas en la capa de salida. Las neuronas de la capa de salida forman un código de uno de los 26 fonemas y la salida de la red maneja un sintetizador de lenguaje que transforma el sonido generado asociado a los fonemas de entrada.

La entrada de la red es un vector binario de dimensión 203 (que codifica una ventana de 7 caracteres consecutivos, 29 bits para cada 7 caracteres, incluyendo la puntuación; cada caracter es codificado usando un código binario de uno de los 29 fonemas). La salida deseada es el código de un fonema dado para la pronunciación de las letras de entrada.

El conjunto de entrenamiento para la red MLP está formado por 1024 palabras de un conjunto de fonemas en inglés. NETtalk es capaz de identificar un lenguaje ilegible después de 10 ciclos de entrenamiento con una exactitud de 95%. Cuando se prueba con un texto nuevo, la red puede distinguir entre vocales y consonantes con una exactitud de 78%.

– Otra aplicación de la red MLP se lleva a cabo en el **reconocimiento de escritura**. En Estados Unidos, el servicio postal está interesado en reconocer la escritura de los dígitos que aparecen en los correos. Para este problema se diseña una red MLP que reconoce segmentos de números digitalizados en la escritura de los códigos ZIP [Hassoon, 1995].

  La red MLP se entrena con un conjuto de 7291 ejemplares, los cuales contienen números ambiguos, inclasificables, o con algún error de clasificación. Los ejemplos se presentan a través de una transformación lineal que hace que el segmento de un dígito se ajuste en una imagen de 16×16 niveles de grises (los niveles de grises en cada imagen son tranformados y escalados a un rango de 1 a -1).

  La red se compone de una capa de neuronas de entrada, tres capas de neuronas ocultas H1, H2 y H3, y una capa de neuronas de salida. La capa oculta H1 se conecta a la imagen de entrada y la capa oculta H3 envía el resultado a las neuronas de la capa de salida. La capa H3 está formada por 30 neuronas conectadas completamente a las neuronas de la capa H2 y a la neuronas de la capa de salida.

  Para el entrenamiento de la red se utilizan conexiones controladas por un peso para reducir el número de párametros libres en la red. Estas conexiones se establecen entre la capa de entrada y las capas ocultas. El objetivo es desarrollar propiedades que seleccionen las características relevantes, de tal manera que se simplifique la tarea de clasificación en la capa oculta H3 y la capa de salida.

La primer capa oculta H1 está compuesta de un "mapa de características", donde todas las neuronas en una parte del mapa tiene el mismo conjunto de pesos y algunas neuronas difieren en el conjunto de umbral. Existe 12 grupos de 8×8 mapas de características (64 neuronas por mapa). Cada neurona en un mapa recibe como entrada una imagen de 5×5, donde la posición exacta de una característica no se determina con mucha precisión. Cada una de las 64 neuronas en un mapa ejecuta las mismas operaciones sobre parte de la imagen. Los mapas se utilizan para detectar la presencia de una de las 12 posibles microcaracterísticas en una posición arbitraria en la imagen de entrada.

La capa H2 está compuesta por 12 mapas de características, cada uno de 4×4 neuronas. Cada neurona en la capa H2 recibe como entrada un subconjunto de 8 mapas, de los 12 mapas que se utilizan en la capa oculta H1. Para esta estructura, la red tiene 1,000 neuronas con 64,660 conexiones y 9,760 pesos independientes. Todas las neuronas usan una función de activación tangente hiperbólica y, antes del entrenamiento, los pesos se inician con una distribución aleatoria uniforme entre -2.4 y +2.4.

La red se entrena durante 23 ciclos a través de un conjunto de entrenamiento que require de 3 días de tiempo de CPU sobre una Sun SparcStarion 1. El porcentaje de patrones mal clasificado es de 0.14% sobre el conjunto de entrenamiento y 5% sobre el conjunto de prueba (con 2,007 nuevas muestras).

– La red MLP se emplea en la **compresión de imágenes**. El objetivo es explotar la redundancia que existe en las imágenes para almacenarlas o transmitirlas, de tal manera que se utilice un número menor de bits [Hassoon, 1995].

La compresión de imágenes es un problema de codificación y decodificación para optimizar la calidad de una imagen codificada, de tal forma que sea posible recuperarla de tamaño original. Para este problema se emplea la red MLP formada por el mismo número de neuronas en la capa de entrada como en la capa de salida y cuenta con una capa de neuronas ocultas. Las neuronas de la capa oculta evalúan una función sigmoide bipolar y las neuronas de la capa de salida tienen una función lineal. Esta red se entrena con un conjunto de vectores de dimensión n formado por números reales y actúa como un codificador.

La red recibe como entrada regiones de 8×8 pixeles de una imagen ($n = 64$) y cuenta con 16 neuronas ocultas. La propagación del error

hacia atrás se usa para que la red autoasocie aleatoriamente ventanas selecciondas de 8×8 pixeles de la imagen dada. Después del entrenamiento, la red se utiliza para descomprimir la imagen de tal manera que se pueda recuperar la imagen total, parte por parte.

- **Consecuencias:**

  Ventajas

  - La red MLP proporciona resultados más precisos si se utiliza un número grande de neuronas en sus capas ocultas.
  - Esta red es poco sensitiva al ruido, ya que los datos para el entrenamiento son procesados muchas veces.
  - La red MLP es capaz de proporcionar un resultado cuando la información es redundante, o no se cuenta con la información completa.

  Desventajas

  - La velocidad para el entrenamiento de la red es muy lenta debido a que tiene que propagar el error hacia las capas ocultas.
  - A pesar de que la red ya ha sido entrenada, la evaluación para un dato nuevo es muy lenta.
  - En ocasiones las red no converge a una solución determinada.

4.3 Patrón *Funciones de Base Radial o RBF*

- **Nombre:** Redes de Funciones de Base Radial o RBF (*Radial Basis Function*)
- **Resumen:** El Patrón RBF describe una red neuronal artificial que utiliza la salida esperada para establecer una relación entre los datos de entrada cuando es importante el tiempo de respuesta.
- **También conocido como:** Aproximador Universal.
- **Ejemplo:** En la comunicación entre sistemas de cómputo se requiere reconstruir señales binarias, de tal manera que apartir de una señal distorsionada o con ruido, pueda recuperarse la señal original en tiempo real [Luo and Unbehauen, 1997].

- **Contexto:** El patrón RBF se utiliza cuando se tienen pocos datos relacionados, se conoce de antemano la salida, y es importante el tiempo de respuesta. Sin embargo no se requiere mucha precisión en la respuesta.

- **Problema:** Establecer la relación entre los datos de entrada y la salida esperada.

  Las *Fuerzas* asociadas a este problema son:

  - Se cuenta con poca información de entrada.
  - No siempre hay dependencia clara entre los datos.
  - Se conoce la salida esperada.
  - Es importante el tiempo de ejecución.
  - Se requiere poca precisión en la respuesta.

- **Solución:** La Red RBF establece una relación entre los datos de entrada y los datos de salida a partir de ciertos valores esperados. Esta red está formada por neuronas artificiales que calculan la distancia euclidiana entre el vector de entrada y los centros de las funciones base en las neuronas ocultas. En el entrenamiento se lleva a cabo la modificación de los pesos entre las conexiones y se actualizan los parámetros necesarios para las funciones base. El objetivo es establecer la relación que existe entre los datos de entrada y salida esperada por medio de la combinación lineal de funciones base [Hu and Hwang, 2002] y limitar la respuesta de la red en una región local dentro del espacio de entrada para cada función base.

- **Estructura:** Las redes RBF están formadas por tres tipos de neuronas: n neuronas en la capa de entrada, k neuronas en la capa oculta o intermedia y m neuronas en la capa de salida, $n \leq k \leq m$, con $(n \times k) + (k \times m)$ conexiones. La figura 4.6 muestra la estructura general de las redes RBF.

  Participantes:

  - Neuronas de entrada: Estas neuronas reciben los datos para el entrenamiento de la red, es decir, los vectores de entrada, así como los valores de salida esperados. Se encargan de comunicar estos datos a las neuronas de la capa oculta y a las neuronas de la capa de salida.
  - Neuronas ocultas: Estas neuronas contienen los parámetros necesarios para las funciones base, como son los centros y la amplitud de dichas funciones. Reciben el vector de entrada y calculan la distancia de este

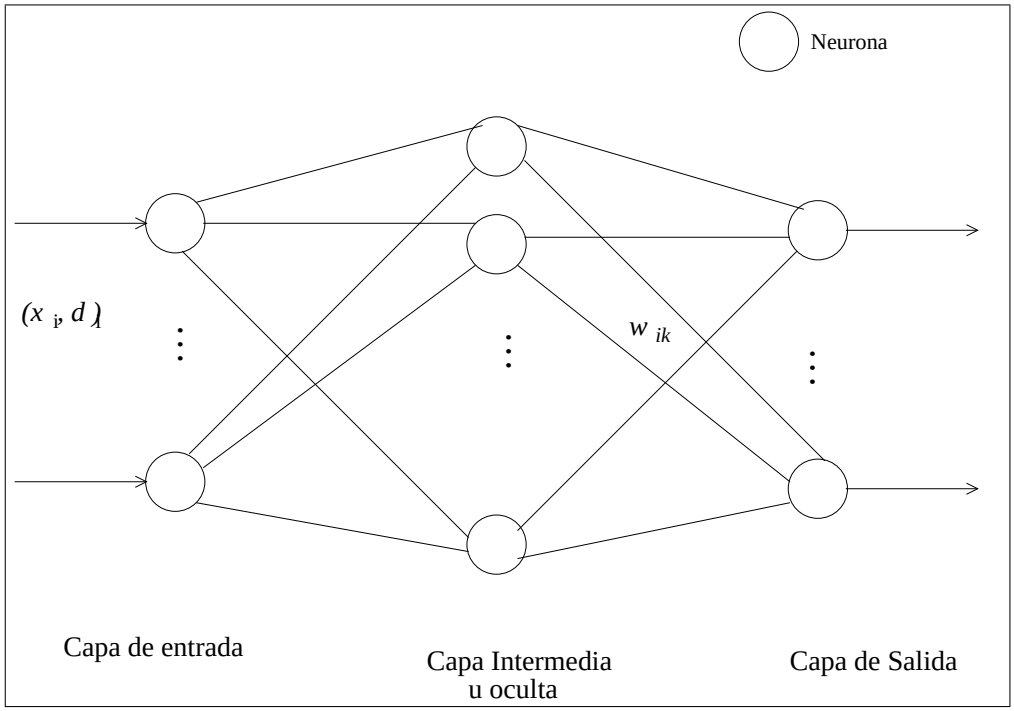

Figura 4.6: Estructura de la red RBF

vector a los centros asociados a cada neurona. Evalúan las funciones de activación para actualizar los parámentros necesarios (centros, amplitud de las funciones base y los pesos) y envían el resultado a las neuronas de salida.

– Neuronas de salida: Estas neuronas reciben el resultado de la capa oculta y lo comparan con la salida esperada, obteniendo el error y tratando de mínimizarlo usando el algoritmo de mínimos cuadrados.

- **Dinámica:** Suponemos que un vector de entrada o patrón de entrenamiento tiene n características y está representado por el vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ en el espacio de dimensión n. Sea $\mathbf{c} = (c_1, c_2, \ldots, c_n)$ el vector que contiene los centros de las funciones base en el espacio de dimensión n. La red proyecta el espacio de entrada sobre la salida a través de la capa oculta.

  Cada neurona k en la capa oculta se caracteriza por un vector de pesos $\mathbf{w}_k = (w_{k1}, w_{k2}, \ldots, w_{kn})$ de dimensión n, recibe la misma entrada $\mathbf{x} \in \mathbb{R}^n$

y tiene diferente función de activación. En la capa oculta, la red calcula la distancia euclidiana entre el vector de entrada y el vector de centros, evalúa la función de activación y manda tal resultado a la capa de salida.

Durante el aprendizaje, la red calcula los pesos usando el algoritmo de mínimos cuadrados (*Least Mean Square, LMS*). Es decir, considera el error que existe entre la salida de la red y el valor esperado para ajustar la matriz de pesos, los centros y la amplitud de las funciones base minimizando, el error para modelar los datos en un sentido local.

En la capa de salida, las neuronas calculan el resultado utilizando el algoritmo de mínimos cuadrados. De ello depende que se modifique los parámentros de la red. La figura 4.7 muestra el diagrama de secuencia para la fase de entrenamiento de la red RBF.

El algoritmo de entrenamiento que utiliza la red RBF se describe a continuación:

1. Crear el vector de pesos $\mathbf{w}_k = (w_{k1}, w_{k2}, \ldots, w_{kn})^T$ para cada conexión entre las neuronas de la capa oculta y la capa de salida.
2. Determinar el vector de centros $\mathbf{c}_k = (c_1, c_2, \ldots, c_n)^T$ para cada función base. Existen diferentes métodos que emplean los datos de entrada para escoger los centros de las funciones base [Hu and Hwang, 2002].
3. Presentar el vector de entrada $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ en la capa de entrada. Y la salida deseada $\mathbf{d}_i = (d_1, d_2, \ldots, d_n)$ en la capa de salida.
4. Cada neurona en la capa de entrada recibe los vectores del conjunto de entrenamiento y los envía a las neuronas de la capa oculta. En la capa oculta:
   (a) Se calcula el producto punto entre el vector de entrada $\mathbf{x}$ y los centro $\mathbf{c}$ de las funciones base.
   (b) Se evalúa la función base y el resultado se envía a las neuronas de la capa de salida.

   En la capa de salida:
   (a) Se calcula el error que comete la red comparando el resultado de la red y la salida esperada. El error se minimiza utilizando el algoritmo de mínimos cuadrados.
   (b) Se actualizan los centros, la amplitud de las funciones base y los pesos.

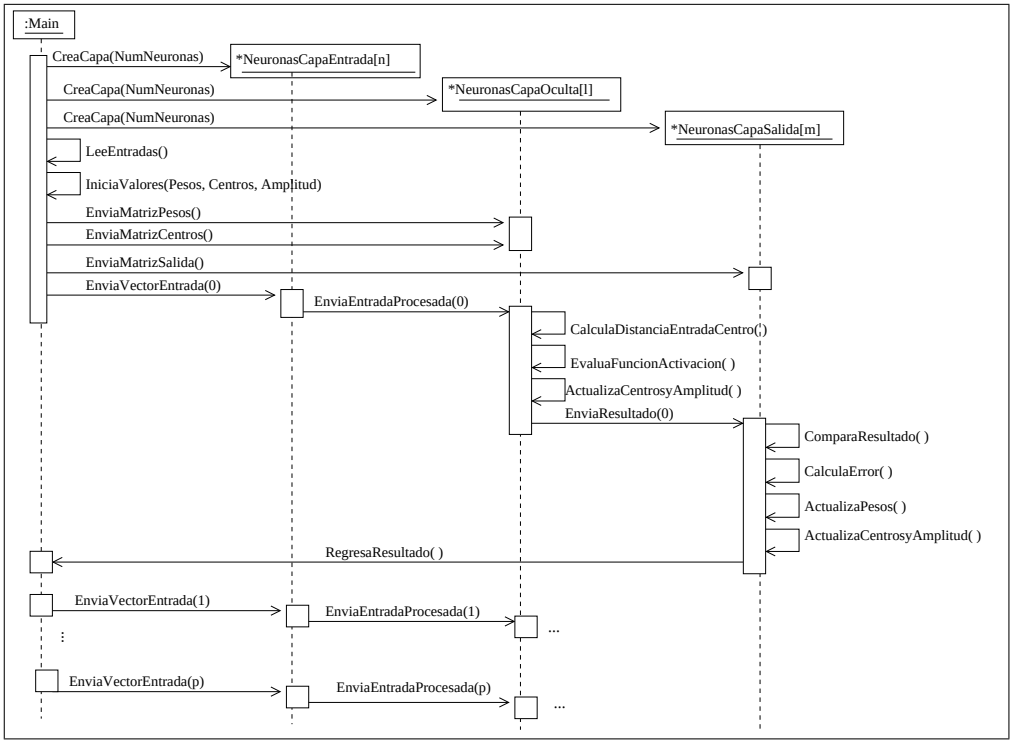

Figura 4.7: Diagrama de secuencia para el entrenamiento del algortimmo RBF

5. El proceso se repite t veces.

- **Implementación:** Para la implementación de una red RBF se consideran los siguientes aspectos:
  - La estructura de la red RBF consta de una capa de neuronas de entrada, una capa de neuronas ocultas y una capa de neuronas de salida.
  - La información en la red RBF fluye en una sola dirección. Es decir, la red RBF recibe los datos en la capa de entrada y estos datos se envían a las neuronas de la capa oculta. En la capa oculta se realizan la operaciones necesarias para actualizar los pesos de las conexiones entre las neuronas y los centros de las funciones base de cada neurona. El resultado se envía a las neuronas de la capa de salida.
  - El centro de las funciones base en las neuronas de la capa oculta es diferente para cada neurona.
  - El entrenamiento de la red RBF empieza cuando la red recibe los datos de entrada y éstos se envían a las neuronas de la capa oculta, en donde se lleva a cabo las siguientes operaciones:
    1. Inicializar el vector de pesos $\mathbf{w}_k = (w_{k1}, w_{k2}, \ldots, w_{kn})^T$ para cada conexión entre las neuronas de la capa oculta y la capa de salida.
    2. Determinar el vector de centros $\mathbf{c}_k = (c_1, c_2, \ldots, c_n)^T$ para cada función base. Existen diferentes métodos que emplean los datos de entrada para escoger los centros de las funciones base [Hu and Hwang, 2002]. Por ejemplo, escoger el centro de las funciones base de manera aleatoria, o utilizar el algoritmo k-means con los datos de entrada.
    3. Presentar el vector de entrada $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ en la capa de entrada. Y la salida deseada $\mathbf{d}_i = (d_1, d_2, \ldots, d_n)$ en la capa de salida.
    4. En la capa oculta, calcular el producto punto entre el vector de entrada $\mathbf{x}$ y los centro $\mathbf{c}$ de las funciones base.
    5. Evaluar la funciones base y el resultado se envía a las neuronas de la capa de salida.
    6. En la capa de salida, calcular el error que comete la red comparando el resultado de la red con la salida esperada.
    7. Minimizar el error utilizando el algoritmo de mínimos cuadrados.

8. Actualizar los centros, la amplitud de las funciones base y los pesos.

- **Ejemplo resuelto:** El problema de reconstruir señales binarias en los sistemas de comunicación en tiempo real se resuelve con un ecualizador no lineal que se basa en redes RBF [Luo and Unbehauen, 1997].

  Este ecualizador está formado por tres capas de neuronas. Las neuronas de la capa de entrada evalúan una función lineal y el resultado alimenta la entrada de las neuronas en la capa oculta. Para este problema las conexiones entre la capa de entrada y la capa oculta no tienen pesos. Sin embargo en la capa oculta, cada neurona recibe el valor de entrada correspondiente sin alterar y evalúa las funciones de base radial [Luo and Unbehauen, 1997].

  Para este ejemplo se utiliza la función Gausiana y el resultado se denota por $h_i(n)$.

$$h_i(n) = f_i(||Y(n) - \mathbf{c_i}||) \tag{4.4}$$

  La capa de salida está formada por una neurona que calcula la siguiente función de activación:

$$g(x) = \frac{1 - e^{\alpha x}}{1 + e^{\alpha x}} \tag{4.5}$$

  El resultado de la red es el equalizador no lineal que funciona como un umbral. Las conexiones entre las neuronas de la capa oculta y la neurona de la capa de salida sí tienen pesos, denotados por el vector $\mathbf{w}_k$ que representa el pesos de la conexión entre la $k-$ésima neurona en la capa oculta y la neurona de salida [Luo and Unbehauen, 1997].

  El parámetro de escalamiento σ, el vector de centros $\mathbf{c}$, y el vector de pesos $\mathbf{w}_k$, $(k = 1, 2, \ldots, n)$ se calculan antes que la salida de la red. Después se estiman los valores de la señal que se transmite. Esto depende del canal de comunicación, que se determina con los datos disponibles y algunos ejemplares conocidos de la señal transmitida [Luo and Unbehauen, 1997].

- **Variantes:** Moody y Darken proponen una variante de la red RBF para mejorar las propiedades de aproximación. La propuesta consiste en que la activación de neuronas ocultas sea normalizada. El objetivo es utilizar una suma igual a uno en las neuronas ocultas para todas las entradas. Así la red realiza la partición de la unidad para la descomposición y aproximación de funciones [Hassoon, 1995].

Otra variante de las redes RBF consiste en utilizar una competencia suave (*soft competition*) para localizar los centros de las funciones gaussianas. Este algoritmo es equivalente a realizar una competencia entre las neuronas donde la neurona ganadora toma todo. Es decir, en este caso las neuronas RBF con la salida mayor actualizan sus pesos [Hassoon, 1995].

- **Usos conocidos:** En general las redes RBF se utilizan en funciones de aproximación, clasificación y modelado de sistemas dinámicos, en series de tiempo, en procesamiento de señales, etc.

  - Una aplicación de las redes RBF se lleva a cabo en la **clasificación de sonidos**. El problema consiste en clasificar el sonido de 10 palabras pronunciadas por 67 personas (hombres, mujeres y niños) [Hassoon, 1995].
    Los datos se obtienen haciendo el análisis espectrográfico de las vocales contenidas en las palabras pronunciadas. Para ello se considera la primera y segunda frecuencia de la región de resonancia. Las palabras utilizadas en esta aplicación son: *heed, hid, head, had, hud, hod, heard, hood, who'd,* y *hawed*. La información obtenida se divide aleatoriamente en dos conjuntos, uno con 338 datos para el entrenamiento de la red y el otro con 333 para la etapa de prueba.
    La red RBF que se utiliza está formada por 100 neuronas ocultas, con una función de activación gaussiana. Después de la fase de entrenamiento, esta red RBF clasifica correctamente alrededor del 87.1% de los datos del conjunto de pruba (333 ejemplares).
  - En el **Mercado Finaciero**, se usan las redes RBF como una solución alternativa para la opción de precios. La fórmula Black-Scholes se emplea para derivar valores iniciales bajo condiciones muy especificas, pero en circunstancias donde el modelo es no lineal y no existe una solución análitica. Entonces se utilizan las redes RBF. El objetivo es que la red RBF aprenda la fórmula de Black-Scholes y simule los datos. Se supone que esta fórmula determina la opción de precios y emplea el método de Monte Carlo para generar datos de precios artificiales. El resultado de este trabajo muestra que la red RBF es capaz de aprender la opción de precios con una precisión adecuada [Hu and Hwang, 2002].
  - Las redes RBF se utilizan para modelar **series de tiempo**. Las series de tiempo requieren estimaciones de un modelo de la forma $y(t) = F(x(t))$ donde F es la función que se desea mapear $F : \mathbb{R}^m \rightarrow \mathbb{R}$.
    Un problema típico en las series de tiempo consiste en predecir la serie de Mackey-Glass, observando los datos generados por el retraso de la

ecuación diferencial Mackey-Glass:

$$\frac{\delta x(t)}{\delta t} = -bx(t) + a\frac{x(t-\tau)}{1+x(t-\tau)^{10}} \tag{4.6}$$

con $\tau = 17$, $a = 0.2$, y $b = 0.1$.

En este problema se emplea las redes RBF para predecir series de tiempo generadas por una aproximación de la ecuación 4.6. La red RBF que se utiliza contiene entre 100 y 1000 neuronas. Para seleccionar los centros de las funciones base se consideran los vecinos más cercanos usando un centro como punto base. Sin embargo sólo se aplica para un conjunto finito de datos. Otra forma de seleccionar los centros consiste en utilizar el algoritmo de agrupamiento k-means.

Computacionalmente, la red RBF para este problema es 16 veces más eficiente que la red MLP, pero requiere alrededor de 27 veces más datos para obtener una precisión similar [Hu and Hwang, 2002].

- **Consecuencias:**

  Ventajas

  - La velocidad en el entrenamiento de la red es rápida con respecto a la del red Perceptrón Multicapa (MLP).
  - La red RBF no requiere muchas neuronas para proporcionar el resultado.
  - Se puede controlar la precisión a través de los parámetros de las funciones de activación.
  - Requiere poca memoria para la ejecución del programa.

  Desventajas

  - Necesita un mayor número de información con respecto a MLP para obtener la misma precisión.
  - Para obtener resultados más precisos se requiere calcular un número elevado de centros para las funciones de base.
  - En ocasiones no converge a una solución y puede caer en algún mínimo local.

4.4 Patrón *Redes de Hopfield*

- **Nombre:** Patrón Redes de Hopfield.
- **Resumen:** El Patrón Redes de Hopfield describe una red neuronal artificial que se utiliza para la clasificación o reconstrucción de patrones a partir de la información almacenada.
- **También conocido como:** Redes de Memoria Asociativa.
- **Ejemplo:** En reconocimiento de imágenes, se requiere reconstruir imágenes que representan cuatro letras. Las letras están distorsionadas o incompletas. En este caso los datos no definen completamente la imagen y la clasificación debe ser automática [Hilera and Martínez, 1995].
- **Contexto:** El patrón Redes de Hopfield se utiliza cuando se tiene una gran cantidad de datos con mucho ruido.
- **Problema:** Encontrar características similares entre los datos de entrada sin supervisión.

  Las *Fuerzas* asociadas a este problema son:

  - Los datos representan un gran número de características.
  - Los datos pueden tener dependencias entre ellos.
  - Los datos puede representar solo una parte de la información a clasificar.
  - No se requiere supervisión para llevar a cabo la clasificación.

- **Solución:** La red de Hopfield encuentra características similares entre los datos de entrada. El objetivo es clasificar o reconstruir la información de entrada a partir de la información ya almacenada. El entrenamiento de la red consiste en almacenar un conjuto de información mediante la presentación repetida de los datos de entrada y la adaptación de los pesos. En la fase de prueba, la red es capaz de recuperar la información original, si se presenta un dato ya almacenado. Sin embargo, si la información de entrada no coincide con ningún dato almacenado por estar incompleto o deformado, la red genera la salida más parecida.
- **Estructura:** Las redes de Hopfield poseen una capa formada por n neuronas. Estas neuronas se conectan entre sí, de tal manera que la red tiene $(n \times n)$

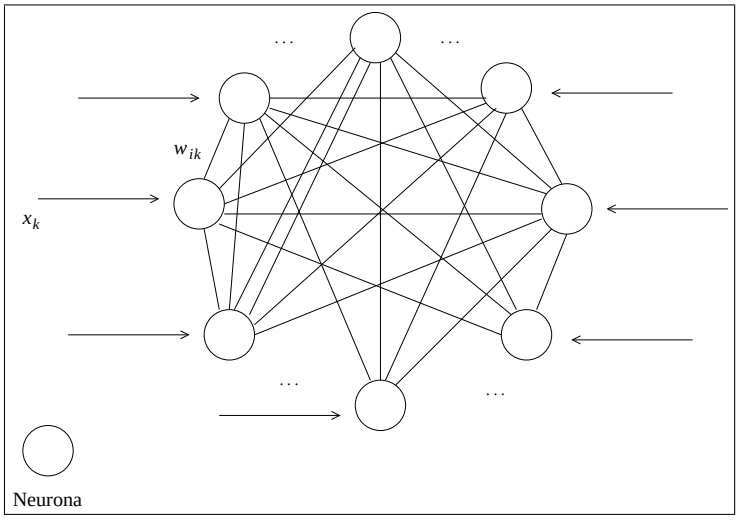

Figura 4.8: Topología de la red de Hopfield.

conexiones. En la figura 4.8 se muestra la estructura general de las redes de Hopfield.

Participantes:

- Neuronas de entrada y salida: En las redes de Hopfield todas las neuronas tienen la misma función. Reciben el mismo vector de entrada, calculan los pesos entre sus conexiones por medio de la multiplicación de matrices y se actualizan de manera asíncrona dependiendo del resultado de la función de activación.

- **Dinámica:** Suponemos que un vector de entrada o patrón de entrenamiento tiene n características y está representado por el vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ en el espacio de dimensión n. Durante el entrenamiento, la red almacena un conjunto de vectores de entrada y proyecta el espacio de entrada sobre la salida. El objetivo es reconstruir la información de entrada a partir de la información almacenada.

  Las redes de Hopfield están formadas por una capa con n neuronas conectadas completamente. Cada neurona k se caracteriza por un vector de pesos $\mathbf{w}_k = (w_{1k}, w_{2k}, \ldots, w_{kn})$ de dimensión n. Los pesos de las conexiones son simétricos, es decir, el peso de la conexión entre la neurona k a la neurona j tienen el mismo valor $w_{kj} = w_{jk}$ para $k \neq j$ en otro caso $w_{jk} = 0$.

Cada neurona recibe la misma entrada $\mathbf{x} \in \mathbb{R}^n$, evalúa la misma función de activación y su salida retroalimenta al resto de las neuronas.

La red de Hopfield está diseñada para trabajar con valores binarios -1 y 1. Sin embargo, para el ajuste de los pesos también se consideran los valores de 0 y 1.

Al iniciar la fase de entrenamiento se presenta el vector de entrada $\mathbf{x}$ a la única capa que tiene esta red. Cada neurona recibe como entrada la salida de cada una de las otras neuronas. Estos valores inicialmente coinciden con los datos de entrada, multiplicados por los pesos de sus conexiones correspondientes. La suma de todos estos valores corresponde a la entrada de otra neurona, la cual evalúa la función de transferencia. Este procedimiento es el primer paso del algoritmo de entrenamiento y se repite hasta que la salida de las neuronas se estabilicen, es decir, la salida de las neuronas no cambie su valor. Cuando esto sucede se genera la salida de la red.

El funcionamiento de la red de Hopfield se basa en la regla de Hebb [véase sección 3.4.3, pág. 36]. Esta regla es un método para determinar los pesos de la red utilizando la información de entrada [Fausett, 1994]. La figura 4.9 muestra el diagrama de secuencia para el entrenamiento de la red de Hopfield. El objetivo es establecer el cambio de los pesos entre dos neuronas de manera proporcional. A esta proporción se le denomina tasa de aprendizaje. Para obtenerla se localizan los minímos locales de la función de activación, de tal manera que al presentarse un nuevo dato de entrada, este valor sea asociado a un dato ya almacendado como un minímo local.

El algoritmo de aprendizaje que utiliza la red de Hopfield se describe a continuación:

1. Iniciar el vector de pesos $\mathbf{w}_k = (w_{k1}, w_{k2} \ldots, w_{kn})^T$ para las conexiones entre las neuronas.
2. Presentar el vector de entrada $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$ a cada neurona.
3. El estado inicial de la red, en el tiempo $t = 0$, es igual al vector de entrada $y(0) = x(0)$.
   (a) Calcular el nivel de activación de la neurona k en el tiempo $t + 1$ como la suma del producto punto de los pesos por el estado anterior de la red. Este valor se denota por $h_k(t + 1)$

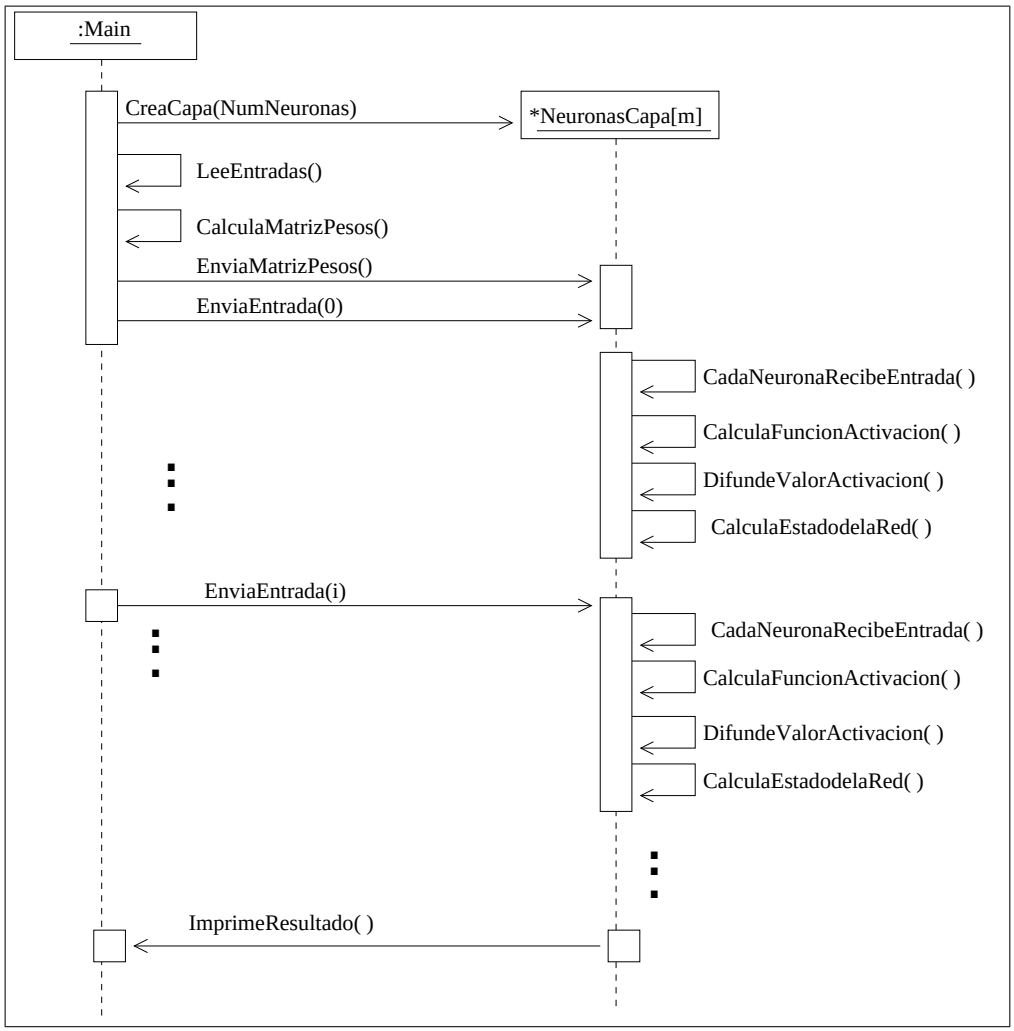

Figura 4.9: Diagrama de secuencia para el entrenamiento de la red de Hopfield

(b) Determinar la activación de cada neurona en el tiempo t+1 evaluando la función de activación. Generalmente se utliliza la función signo (sgn) o escalón, sobre el nivel de excitación $h_k(t+1)$ determinado por:

$$y_k(t+1) = sgn(h_k(t+1)) = \begin{cases} +1 & \text{si} \quad h_k(t+1) > 0 \\ -1 & \text{si} \quad h_k(t+1) < 0 \\ y_k(t) & \text{si} \quad h_k(t+1) = 0 \end{cases} \tag{4.7}$$

(c) Difundir el resultado a todas las neuronas y actualizar su estado.

(d) El proceso se repite hasta alcanzar un estado estable. Es decir, que el estado de todas las neuronas no se modifique.

$$y_k(t) = y_k(t+1) \tag{4.8}$$

- **Implementación:**

Para la implementación de una red de Hopfield se consideran los siguientes aspectos:

1. La estructrua de la red de Hopfield consta de una capa con n neuronas.
2. Cada neurona recibe datos del resto de las neuronas y envía su salida a todas las neuronas.
3. Los pesos entre las conexiones son simétricos. Es decir, el peso de la conexión entre la neurona k a la neurona j tienen el mismo valor $w_{kj} = w_{jk}$ para $k \neq j$ en otro caso $w_{jk} = 0$.
4. El algoritmo de entrenamiento y aprendizaje que emplea la red de Hopfield se describe a continuación:
   (a) Iniciar el vector de pesos $\mathbf{w}_k = (w_{k1}, w_{k2} \ldots, w_{kn})^T$ para las conexiones.
   (b) Presentar el vector de entrada $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$ a cada neurona.
   (c) Calcular el estado de la red. En el estado inicial $t = 0$, se trata del vector de entrada, de tal modo que $y(0) = x(0)$.
   (d) Calcular el nivel de activación de la neurona k en el tiempo $t + 1$ como la suma del producto punto de los pesos por el estado anterior de la red.
   (e) Evaluar la función de activación.
   (f) Enviar el resultado a todas las neuronas y actualizar su estado.

(g) Repetir los pasos 4.a a 4.c hasta que el estado de todas las neuronas no se modifique.

$$y_k(t) = y_k(t+1) \tag{4.9}$$

- **Ejemplo resuelto:** En el sistema de reconocimiento de imágenes se consideran únicamente la reconstrucción de cuatro letras (A, B, C y D). Estas letras se codifican como matrices de 7×6 pixeles y cada letra puede estar distorcionada o incompleta [Hilera and Martínez, 1995].

  Para este problema se emplea una red de Hopfield capaz de recuperar la información de entrada apartir de la información parcial o deformada de la misma. Para ello se emplea un red formada por una capa con 42 neuronas, considerando una neurona por pixel, con $42 \times 42 = 1722$ conexiones. Se utilizan valores binarios: negro = +1 y el blanco =-1 [Hilera and Martínez, 1995].

  Durante la fase de entrenamiento, se almacenan los datos correspondientes a las cuatro letras. Después se calculan los 1,722 pesos que tiene la red utilizando los datos de entrada. Posteriormente en la fase de prueba, la red es capaz de reconocer una imagen de entrada distorsionada y genera la salida más parecida a los datos de entrada [Hilera and Martínez, 1995].

- **Variantes:**

  - *El modelo contínuo de la red de Hopfield.* Esta red se caracteriza por tener variables de estado contínuas. El aprendizaje se lleva a cabo mediante un sistema de ecuaciones diferenciales no lineales. La topología de la red es la misma que la del modelo discreto. Sin embargo, considera la polarización o *bias* de cada neurona. Asimismo, para la k-ésima neurona se considera una entrada adicional con valor 1 ($y_0 = 1$) y con un peso asociado $w_{0k} = b_k$, y la salida de la neurona puede tomar valores reales [Corchado et al., 2000].
  - *La Máquina de Boltzman* es una generalización de la red de Hopfield que incluye neuronas ocultas y la forma en que se actualiza las neuronas se basa en un concepto de termodinámica conocido como *simulated annealing* [Pal and Mitra, 1999].

- **Usos conocidos:** Las redes de Hopfield generalmente se emplean en reconocimiento de imágenes y resolución de problemas de optimización.

- En el **reconocimiento de imágenes** se utiliza la red de Hopfield para reconstruir versiones distorcionadas o parciales de imágenes almacenadas. Se diseña una red con 234000 neuronas para reconocer imágenes de 130×180 pixeles. Durante el entrenamiento, la red almacena siete imágenes. En la etapa de prueba, se presentan imágenes distorsionadas e incompletas y la red las asocia al patrón más parecido que haya almacenado antes [Hilera and Martínez, 1995].

- La red de Hopfield se ha empleado para resolver **problemas de optimización**. El objetivo es expresar un problema mediante una expresión matemática, llamada función de costo, de tal manera que se busca minimizarla. Por ejemplo para desarrollar conversores analógicos-digitales (A/D), el diseño se plantea como la solución a un problema de optimización en términos de una función de energía. Los conversores A/D son circuitos electrónicos que reciben como entrada una señal analógica (por ejemplo, una señal de audio) y genera una serie de configuraciones binarias o palabras con un cierto número de bits. Mientras mayor sea el número de bits, mayor es la precisión en la conversión. Un conversor A/D se caracateriza por el número de dígitos binarios que componen cada palabra generada: es posible tener 2^n palabras diferentes, donde n es el número de bits de cada palabra. El objetivo es reconstruir una señal analógica a partir de los valores de las entradas y obtener una señal semejante a la original. Para el diseño del conversor A/D se utiliza la red de Hopfield: primero se define la función que se quiere minimizar y se ajusta mediante un término extra que garantice la estabilidad; segundo, se obtiene los pesos; finalmente, la implementación física se hace usando una red con 4 neuronas [Hilera and Martínez, 1995].

- Otra aplicación de las redes de Hopfield se lleva a cabo en la solución del problema del **agente viajero**. Un viajero tiene que visitar n ciudades, de tal manera que saliendo de una ciudad visite todas las ciudades sin pasar más de una vez por cada una y además recorra la distancia menor entre cada ciudad. Al final debe regresar a la ciudad de partida [Corchado et al., 2000, Hilera and Martínez, 1995]. Para este problema se utiliza una red de Hopfield con $n \times n$ neuronas. Si se visitan cinco ciudades, la red de Hopfield está formada por 25 neuronas conectadas completamente. Se inicializan los pesos y se establece una función de optimización de tal manera que después de entrenar a la red, es capaz de presentar la ruta con la distancia mínima.

- **Consecuencias:**

  Ventajas

  - La red Hopfield reconstruye imágenes deformadas a partir de la información almacenada.
  - La red asocia datos de entrada a patrones ya identificados.

  Desventajas

  - El número máximo de patrones que puede almacenar la red es limitado.
  - Requiere mucho tiempo de procesamiento para converger a una solución estable.
  - Puede caer en mínimos locales y no converger a una solución.
  - La red no puede recuperar correctamente un patrón ya almacenado si ha sufrido una rotación o traslación [Hilera and Martínez, 1995].

4.5 Resumen

El Sistema de Patrones de Software para Redes Neuronales Artificiales proporciona criterios de selección a partir de las caractísticas del problema y la experiencia obtenida en problemas ya conocidos, de tal manera que se propone una red neuronal artificial como solución.

Este sistema esta formado por cuatro redes neuronales artificiales, las cuales se describen mediante la forma POSA: (a) La red de Mapas Auto-Organizados, (b) El Perceptrón Multicapa, (c) Las redes de Funciones de Base Radial y (d) Las redes de Hopfield. Estas redes se utilizan para la clasificación de los datos de entrada. Como lo considera la teoría de patrones de software, no son un conjunto exahustivo de redes neuronales. Sin embargo, proponen algunos criterios para su selección. Nótese que el presente sistema se considera más bien como un conjunto inicial de patrones de software para redes neuronales, que puede ser complementado y modificado a futuro.

Cada uno de los patrones describe la red neuronal a través de diagramas y proporcionan algunas aplicaciones de estas redes, así como una descripción general de su implementación.

Capítulo 5

Selección de una Red Neuronal utilizando Patrones de Software

En el capítulo anterior se describen cuatro Patrones de Software como descripciones de soluciones parciales a un problema, que se resuelve utilizando Redes Neuronales Artificiales. Estos Patrones de Software consideran las características presentes en un contexto, y se describen como guías para la selección de Redes Neuronales.

En este capítulo se analiza un problema de clasificación de fallas de tal manera que se usan los Patrones de Software del capítulo anterior para seleccionar una red neuronal artificial como solución al problema.

En el problema de clasificación de fallas, éstas se detectan en el sensor de inercia en la simulación de un modelo de avión ADMIRE (*Aero Data Model in a Research Environment*) [Gateour, 2001]. A partir de la descripción del problema, se selecciona un patrón de software que describe una solución para la clasificación de fallas. El objetivo es identificar las características más relevantes en el problema de clasificación de fallas, y a partir de ellas, utilizar las descripciones desarrolladas en el capítulo anterior para seleccionar una Red Neuronal.

5.1 Descripción del problema

Modelar el sistema de un avión consiste en resolver un modelo matemático que represente los diferentes estados de operación del avión. Comúnmente, tal modelo matemático no se puede expresar por medio de ecuaciones algebraicas lineales, de-

bido a que es un sistema dinámico no lineal. Es por esto que para la modelación se proponen distintos puntos de operación, en los cuales el modelo matemático se linealiza para su análisis.

El modelo matemático linealizado se genera usando un paquete de simulación a partir de un punto de operación. En tal paquete, el modelo linealizado se forma por diferentes variables de estado, que incluyen presión, razones de flujo en la masa de gas y de aire, velocidad del eje, temperatura absoluta, presión estática, y otras. Un punto de operación define el comportamiento dinámico del sistema alrededor de ciertas condiciones iniciales. De hecho el modelo es linealizado alrededor de este punto de operación. Así, el modelo linealizado que se obtiene utiliza diferentes puntos de operación para probar y evaluar los diferentes estados de operación del modelo [Patton et al., 2000].

En este capítulo se considera el modelo de un avión basado en el *Aero Data Model in a Research Environment* ADMIRE. Este modelo describe un avión pequeño, desarrollado bajo el ambiente de Matlab/Simulink, el cual se utiliza para realizar simulaciones de vuelo. Para su ejecución se necesita Matlab5.3/Simulink y un compilador de C que soporte Matlab [Gateour, 2001].

La simulación del modelo ADMIRE se desarrolla a Mach 1.2[1] a 6000m de altitud. Sin embargo puede trabajar para altitudes más altas. ADMIRE modela el sistema principal del avión, actuadores, sensores y parámetros de incertidumbre. Cuenta con un sistema de control de vuelo longitudinal que controla la tasa de lanzamiento a baja velocidad y con seis grados de libertad, esto se muestra en la figura 5.1 [Gateour, 2001].

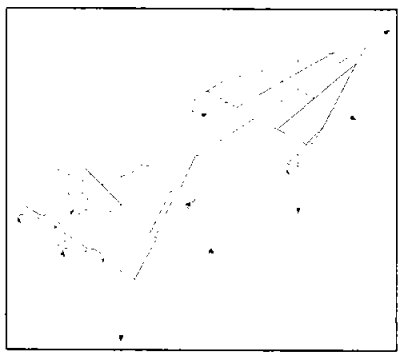

Figura 5.1: Modelo del avión tomado de la página de ADMIRE.

[1]Mach es una medida que representa la velocidad del sonido

El sistema de control de ADMIRE se representa con 30 puntos de operación, diseñados como se muestra en la figura 5.2 [Gateour, 2001]. Las características del sistema de control están dadas entre los puntos por interpolación lineal. Cada punto sobre el perímetro de la figura representa un punto de operación en el movimiento del avión. Para ellos, se consideran las siguientes altitudes: 20m, 3000m y 6000m a un Mach 0.2200, 0.3500, 0.4500, 0.5500, 0.8000, 0.9000, 0.9500, 1.0000, 1.0500, 1.1000 y 1.2000.

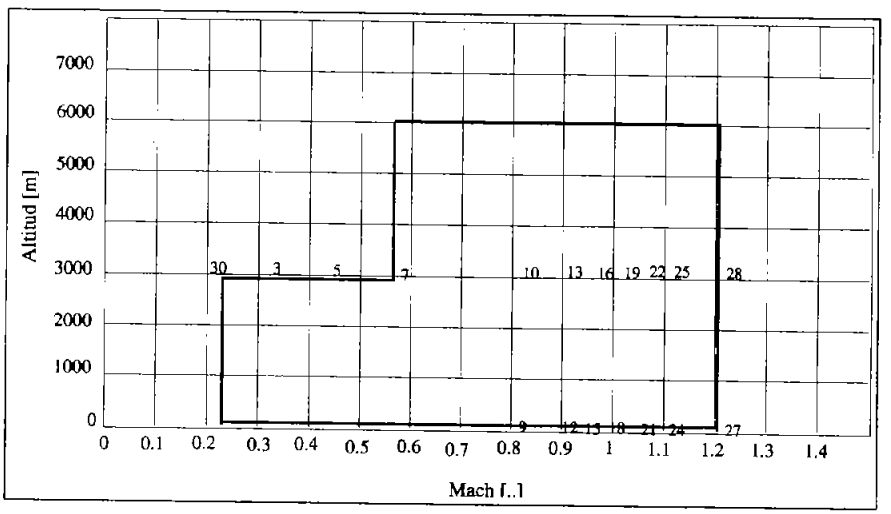

Figura 5.2: Puntos del Diseño del Sistema de Control tomado de la página de ADMIRE.

El problema para propósitos de esta tesis es seleccionar una red neuronal para clasificar fallas utilizando el modelo de ADMIRE. Las fallas tipo corrimiento en amplitud [Chen and Patton, 1999] se aplican en dos sensores diferentes para crear situaciones de incertidumbre, que deben ser detectadas. Nótese que las fallas tipo corrimiento en amplitud son particularmente difíciles de detectar y presentan un desafío para teoría de detección de fallas [Benítez et al., 2005].

La clasificación de fallas en el modelo ADMIRE se lleva a cabo en dos etapas [Benítez et al., 2005]: la primer etapa se desarrolla fuera de línea, y consiste en obtener la información para la base de datos utilizando PCA (*Principal Component Analysis*) [2] de acuerdo al funcionamiento normal del modelo; la segunda etapa se

[2]PCA es un técnica para reducir la dimensión de los datos en terminos de una proyección ortonormal.

desarrolla en línea, y consiste en la clasificación de fallas utilizando algún tipo de red neuronal que realice la comparación de la información almacenada y los datos obtenidos al alterar el funcionamiento del modelo. En esta etapa se contruye la matriz de datos histórica y se evalúa utilizando PCA. Esto se muestra en la figura 5.3.

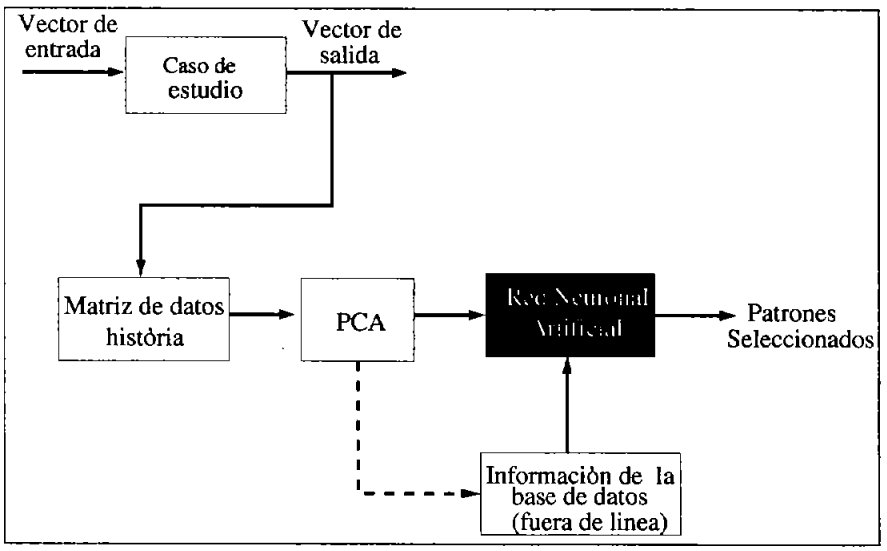

Figura 5.3: Diagrama del problema de clasificación de fallas en el modelo ADMIRE.

La matriz de datos histórica se utiliza para recolectar suficiente información y determinar el funcionamiento del sistema en un punto de operación. Esta matriz se forma con muestras de datos recolectados durante el tiempo total de una ventana de tiempo. Aquí, el tiempo juega un papel crucial en términos de la frecuencia de las fallas. Después de construir la matriz de datos histórica, se altera el funcionamiento de ADMIRE aplicando las fallas antes mencionadas. Se consideran dos ángulos, el ángulo de ataque (ángulo alpha) y el ángulo de desplazamiento (ángulo beta) de los sensores de inercia. El modelo de ADMIRE incluye un ángulo de ataque de -10 grados hasta 30 grados y un ángulo de deslizamiento lateral de ± 20 grados. Los actuadores son estabilizadores izquierdo y derecho del ala principal y contienen cuatro alerones, un timón y equipo con capacidad vertical de empuje y ajuste. Además, están equipados para usar a la turbulencia atmosférica como una fuerza de perturbación externa. Por tanto se toman algunas muestras que permitan visualizar el funcionamiento anormal con diferentes frecuencias, con la finalidad de establecer escenarios desconocidos con límites en los tiempos de muestreo durante una ventana

de tiempo total. Así, con la información que se obtiene, se puede decidir la presencia o ausencia de una falla [Benítez et al., 2005].

Para el experimento se consideran cinco escenarios: un escenario sin fallas, cuatro escenarios con fallas y un solo punto de operación.

Los cinco escenarios se describen a continuación:

- En el escenario libre de fallas se consideran tres varables: la relación del cuerpo fijo, la relación de rotación fija y la relación de la posición del cuerpo fijo. Para este escenario se realiza una simulación durante un período de 60 segundos.
- El primer escenario con fallas considera las mismas variables que el escenario anterior y consiste en aplicar una falla a la deriva en el sensor de ataque (ángulo alpha). Esta falla se aplica cada 20 segundos por un período de 20 segundos. La respuesta del sistema es casi inmediata.
- El segundo escenario con fallas consiste en aplicar la falla a la deriva en el sensor de desplazamiento (ángulo beta). La respuesta aquí es diferente al escenario previo.
- Para el caso del tercer y cuarto escenarios con fallas, se realiza una saturación en los sensores alpha y beta. En este caso, el sistema no presenta ninguna alteración en su funcionamiento dinámico.

Basándose en la descripción dada de la simulación para detección de fallas utilizando el modelo ADMIRE, el objetivo es seleccionar una red neuronal que realice la clasificación de fallas a partir de los datos de la simulación y durante la fase en línea (figura 5.3).

5.2 Selección de una Red Neuronal Artificial

En esta sección se analizan los Patrones de Software para Redes Neuronales Artificiales descritos en el capítulo anterior, como guías de selección para una red neuronal que se adecúe al problema de clasificación de fallas. A través del análisis del problema se busca identificar las características que implique el uso más específico de una red neuronal en particular.

Para llevar a cabo el análisis de cada patrón, inicialmente se revisa su contexto [véase sección 2.4, pág 16]:

- **Contexto de SOM**: El patrón SOM se utiliza cuando los datos no están bien definidos o se tienen muchos datos de entrada [véase sección 4.1, pág. 40].

- **Contexto de MLP**: El patrón MLP se utiliza cuando se tiene grandes cantidades de datos que están relacionados y se conoce de antemano la salida. Además es importante la precisión de la respuesta pero no el tiempo de ejecución [véase sección 4.2, pág. 48].

- **Contexto de RBF**: El patrón RBF se utiliza cuando se tienen pocos datos relacionados, se conoce de antemano la salida, y es importante el tiempo de respuesta. Sin embargo no se requiere mucha precisión en la respuesta [véase sección 4.3, pág.61].

- **Contexto de Hopfield**: El patrón Redes de Hopfield se utiliza cuando se tiene una gran cantidad de datos con mucho ruido [véase sección 4.4, pág. 70].

Para el contexto de SOM, se observa que para el problema de clasificación de fallas utilizando el modelo ADMIRE, se tienen una gran cantidad de datos de entrada (los valores que resultan de aplicar PCA). Además, estos datos no proveen una definición precisa para resolver el problema.

En los contextos de MLP y RBF, se hace notorio que es necesario conocer de antemano la salida deseada. En el ejemplo de clasificación de fallas no contamos con tal información, es decir, no existe una clasificación esperada, debido a que no tenemos grupos de datos que definan la presencia de una falla. A pesar de tener los datos de la matriz de datos histórica, no es suficiente dado que se requiere más datos adicionales para que se defina la presencia de una falla. Por lo tanto, es difícil ubicar el problema de clasificación de fallas en estos dos contextos, ya que no se cuenta previamente con una salida deseada.

En el contexto de Hopfield, se observa que se requiere una gran cantidad de datos con mucho ruido. En este contexto podría ubicarse el problema de clasificación de fallas, ya que la matriz de datos histórica está formada por una gran cantidad de datos (procesados por PCA) y el resultado pueden tener mucho ruido.

Tras el análisis anterior, se observa que potencialmente es posible utilizar los patrones de la red SOM o de la red de Hopfield para el problema de clasificación de fallas. Sin embargo, aún no es suficiente con esta infomación para decidir cuál de estos dos patrones resulta conveniente para la red neuronal dentro del problema. Para ello, a continuación se toma en consideración la descripción de los problemas de estos dos patrones, así como sus fuerzas asociadas.

- **Problema del patrón SOM**: Encontrar características similares entre los datos de entrada sin supervisión [véase sección 4.1 pág. 40].

  Las *Fuerzas* asociadas a este problema son:

  - Los datos representan la información obtenida de varias dimensiones.
  - Los datos no están bien definidos.
  - Se tiene una gran cantidad de datos en los que pueden existir dependencias entre ellos.
  - Se requiere obtener grupos de datos que representen toda la información.
  - No se requiere supervisión para llevar a cabo la clasificación.

- **Problema del patrón de Hopfield**: Encontrar características similares entre los datos de entrada sin supervisión [véase sección 4.4, pág 70].

  Las *Fuerzas* asociadas a este problema son:

  - Los datos representan un gran número de características.
  - Los datos pueden tener dependencias entre ellos.
  - Los datos puede representar solo una parte de la información a clasificar.
  - No se requiere supervisión para llevar a cabo la clasificación.

Aparentemente, el problema que resuelven estos dos patrones es el mismo. Sin embargo, las fuerzas asociadas a cada problema son diferentes.

Para la clasificación de fallas en la simulación del modelo ADMIRE, la información que se tiene es el resultado de aplicar PCA a la matriz de datos histórica. Esto significa que los datos representan la información obtenida en varias dimensiones. Además, estos datos no definen bien el problema, son una una gran cantidad y pueden tener interdependencias entre ellos. El objetivo es hacer agrupamientos basándose sólo en las características de los datos, de manera que la clasificación se realice automáticamente. De acuerdo a estas características, es notorio que la red descrita por el patrón SOM parece ser la más apropidada para el problema de clasificación de fallas, basándose en la descripción del propio problema. Por otro lado, la red descrita por el patrón de Hopfield tiene el inconveniente de que no expresa la dimensión de los datos. Es por esto que se decide que el problema de clasificación de fallas se adecúa más directamente con las fuerzas del patrón de SOM, y por lo tanto, se selecciona este patrón como la solución más apropiada para el problema de clasificación de fallas.

Hasta aquí, se ha mostrado el uso de los Patrones de Software para Redes Neuronales Artificiales como una guía o criterio de selección de redes neuronales, basándose en las características encontradas en la descripción del contexto y el problema a resolver. En la siguiente sección, se muestra cómo el patrón seleccionado sirve como base para el desarrollo de la solución al problema.

5.3 Uso del Patrón de Software para resolver el Problema

En la sección anterior, se selecciona la red neuronal SOM para resolver el problema de clasificación de fallas en el modelo ADMIRE. A continuación se describe el patrón SOM como guía para resolver tal problema.

Basándose en la descripción del patrón SOM [véase sección 4.1, pág. 40] se tiene que:

- **Solución:** Las redes SOM forman mapas de características de manera autoorganizada y establecen una correspondencia entre los datos de entrada y un espacio de 1 ó 2 dimensiones, a través de una función de semejanza. Esta función de semajanza se le conoce como malla, dado que presenta una respuesta homogénea a los datos de entrada. De tal manera que ante vectores de entrada con características comunes se activan neuronas situadas en zonas próximas.

  El entrenamiento de la red SOM consiste en la adaptación y modificación de los pesos entre las conexiones adyacentes a las *neuronas ganadoras*. De esta manera, vectores de datos de entrada similares activan y generan un orden global.

  El aprendizaje de la red termina cuando se reducen las zonas de neuronas activadas por vectores de datos de entrada parecidos. Es necesario repetir el proceso de entrenamiento hasta refinar el mapa topólogico.

- **Estructura:** La red SOM para el problema de clasificación de fallas en el modelo ADMIRE está formada por dos capas de neuronas con conexiones entre ellas: 9 neuronas en la capa de entrada y 500 neuronas en la capa de salida, con $9 * 500$ conexiones. La figura 5.4 muestra la estructura de tal red SOM.

  Participantes:

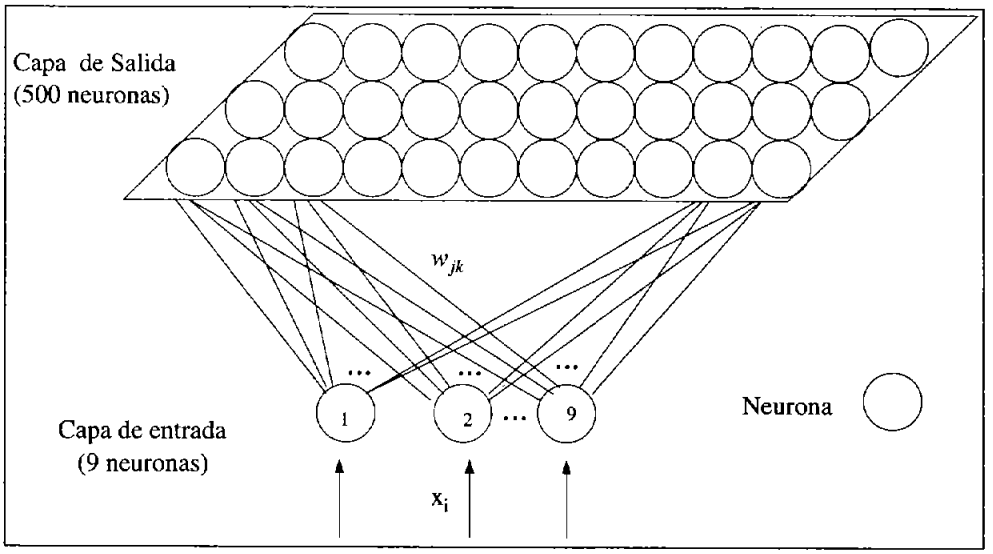

Figura 5.4: Estructura de la red SOM

- Neuronas de entrada (9): Estas neuronas pertenecen a la capa de entrada y sólo reciben los datos que le provee el componente PCA (*Principal Component Analysis*) para el entrenamiento de la red dentro de la solución para la clasificación de fallas. Estas neuronas pasan los datos a las neuronas de la capa de salida.
- Neuronas de salida (500): Estas neuronas se encargan de llevar a cabo (a) el entrenamiento de la red, identificando las diferencias entre las distancias euclidianas y modificando los pesos de las conexiones adyacentes a las neuronas ganadoras y (b) el aprendizaje de la red, que consiste en establecer las diferentes categorías entre los datos en función de su semejanza.

- **Dinámica:** Suponemos que un vector de entrada o patrón de entrenamiento tiene 9 características y está representado por el vector $\mathbf{x} = (x_1, x_2, \ldots, x_9)$ en el espacio de dimensión 9. La red proyecta el espacio de entrada sobre la salida, y en el entrenamiento realiza la proyección conservando el orden topológico. Para ello se utiliza la regla de aprendizaje competitivo [véase sección 3.4.3, pág. 35] con respecto al conjunto de neuronas vecinas, de tal manera que el proceso de aprendizaje no sea global sino local. Cada neurona k se caracteriza por un vector de pesos $\mathbf{w}_k = (w_{k1}, w_{k2}, \ldots, w_{k9})$ de dimensión 9, y recibe la

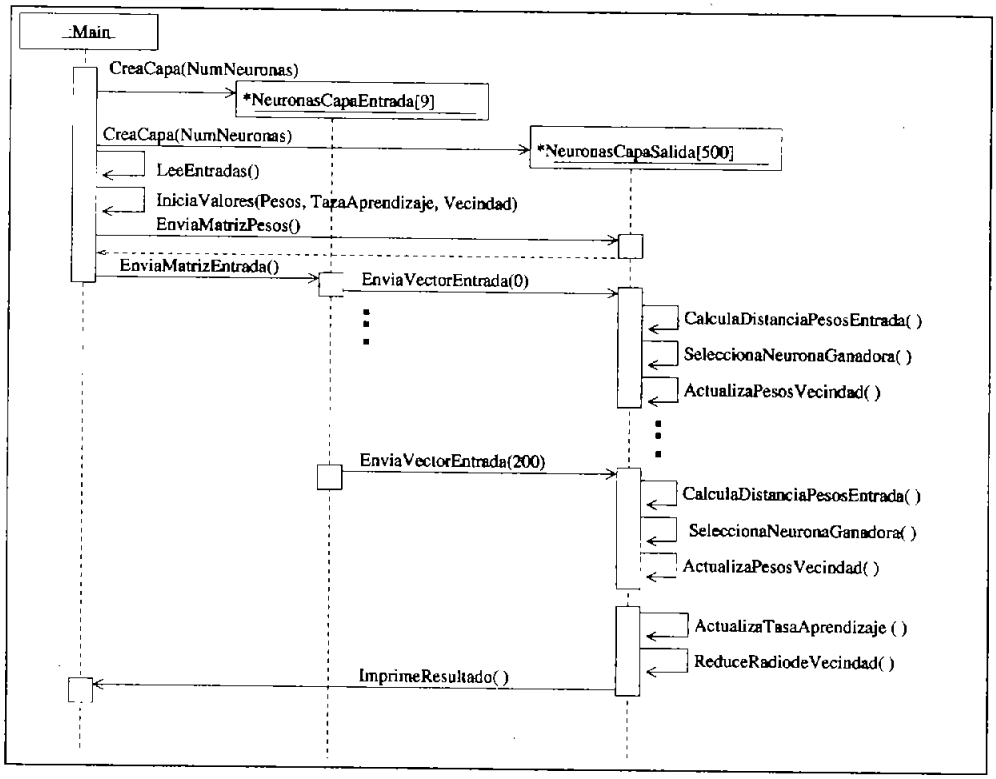

Figura 5.5: Diagrama de secuencia del algoritmo SOM para el problema de clasificación de fallas.

misma entrada $\mathbf{x} \in \mathbb{R}^9$ en la capa de entrada. Este vector pasa a la capa de salida en donde se lleva a cabo el entrenamiento de la red.

En la capa de salida, las neuronas compiten por activarse comparando los datos de entrada con su vector de pesos asociado. Así, el vector de pesos más parecido al vector de datos de entrada determina las neuronas ganadoras y son las únicas que modifican los pesos asociados a sus conexiones. La figura 5.5 describe el diagrama de secuencia que emplea el algoritmo SOM.

El algoritmo de aprendizaje que utiliza la red SOM se describe a continuación:

1. Crear el vector de pesos $\mathbf{w}_k = (w_{k1}, w_{k2} \ldots, w_{k9})$ para cada conexión entre las neuronas.
2. Fijar la zona inicial de vecindad entre las neuronas de la capa de salida.

3. Presentar el vector de entrada $\mathbf{x} = (x_1, x_2, \ldots, x_9)^T$ en la capa de entrada.
   (a) Para cada neurona de salida k caracterizada por su vector de pesos $\mathbf{w}_k$, determinar su estado de activación Y_k, definido por la distancia euclidiana entre $\mathbf{x}$ y $\mathbf{w}_k$: $Y_k = \mathbf{w}_k\mathbf{x}$.
   (b) Seleccionar como neurona ganadora, aquélla neurona cuyo resultado entre la distancia del vector de datos de entrada y el vector de pesos haya sido el menor.
   (c) Una vez seleccionada la neurona ganadora i, se actualizan los pesos de las conexiones entre la neurona ganadora y la neurona de entrada con la siguiente regla de aprendizaje.

$$\mathbf{w}_k(t+1) = \mathbf{w}_k(t) + \eta g(x_k, x_i)(\mathbf{x}(t) - \mathbf{w}_k(t)) \tag{5.1}$$

   donde η es la tasa de aprendizaje, representa un valor constante igual a 0.7 y $g(x_k, x_i)$ es una función entre las neuronas x_k e x_i tal que $g(x_i, x_i) = 1$ [Benítez et al., 2005].
   Generalmente se utiliza la función gaussina. En este caso $g(x_k, x_i)$ es de la forma [Benítez et al., 2005]:

$$g(x_k, x_i) = e^{-0.5*\frac{||x_k - x_i||^2}{\sigma^2}} \tag{5.2}$$

   donde k e i son los índices de las neuronas y σ es la desviación estándar. Esta función se conoce como *malla*, ya que define cómo quedan agrupados los datos de entrenamiento.
4. Actualizar la tasa de aprendizaje.
5. Reducir el radio de vecindad.
6. El proceso se repite, a partir del paso 3, volviendo a presentar todo el conjunto de entrenamiento t veces.

- **Implementación:**

  En la implementación de una red SOM para el problema de clasificación de fallas en el modelo ADMIRE se consideran los siguientes aspectos [Benítez et al., 2005]:

  - La estructura de la red SOM consta de dos capas de neuronas: (a) La capa de entrada de la red SOM sólo proporciona el almacenamiento del vector de entrada. (b) En la capa de salida, cada neurona calcula la distancia entre el vector de salida de la capa de entrada y el vector de pesos.

- Los datos para el entrenamiento de la red SOM son el resultado de aplicar PCA a la matriz de datos histórica y se establecen en la etapa fuera de línea.
- En el entrenamiento se almacenan los índices para localizar la neurona ganadora. Es decir, se identifica la neurona con distancia mínima y su índice se utiliza para identificarla.
- Los datos para probar el funcionamiento de la red SOM se obtienen de la etapa en línea y son comparado con los resultados obtenidos fuera de línea.
- El proceso completo permite la clasificación en línea de los datos basándose en una ventana de tiempo definida por el comportamiento del sistema.

El uso de la red SOM para resolver el problema de clasificación de fallas se justifica por los siguientes motivos:

- La red SOM hace distinciones entre los mapas de diferentes escenarios [Benítez et al., 2005].
- La red SOM localiza las fallas variantes de acuerdo a la clasificación de datos [Benítez et al., 2005].
- El uso de datos multidimensionales permite localizar una falla local [Benítez et al., 2005].
- La red SOM preserva la topología de los datos de entrada, es decir, no modifica la información que recibe, sólo la etiqueta.
- La red SOM trabaja con grandes cantidades de datos.

5.4 Resumen

En este capítulo se analiza un problema de clasificación de fallas de tal manera que se usan los Patrones de Software descritos en el capítulo anterior para seleccionar una red neuronal artificial como solución al problema.

En el problema de clasificación de fallas se considera la simulación de un modelo de avión ADMIRE (*Aero Data Model in a Research Environment*) [Gateour, 2001].

En este modelo se detectan las fallas en el sensor de inercia. A partir de la descripción del problema, se selecciona un patrón de software que describe una solución para la clasificación de fallas. El objetivo es identificar las características más relevantes en el problema de clasificación de fallas, y a partir de ellas, utilizar las descripciones desarrolladas en el capítulo anterior para seleccionar una Red Neuronal.

Para el problema actual, se selecciona la red neuronal SOM, utilizando el Patrón de Software que describe la solución al problema como una la red neuronal de este tipo. Este Patrón de Software considera las características presentes en su contexto, y se describen como guía para la selección de Redes Neuronales SOM.

Capítulo 6

Conclusiones

6.1 Resumen

La presente tesis propone un Sistema de Patrones de Software como guía para la solución de problemas usando Redes Neuronales Artificiales, tomando en cuenta lo que ya se conoce en este campo. Es una propuesta para el uso de Patrones de Software considerando la relación entre los problemas y las soluciones descritas mediante una Red Neuronal Artificial.

Se describen cuatro Patrones de Software para la selección de Redes Neuronales Artificiales: los *Mapas Auto Organizados de Kohonen* (SOM), el *Perceptrón Multicapa* (LMP), las *Redes de Funciones de Base Radial* (RBF) y la *Red de Hopfield*. Estos Patrones de Software son utilizados para la clasificación y asociación de información. Claramente, no abarcan un conjunto exhaustivo de Redes Neuronales Artificiales. Sin embargo se presentan en este trabajo como una base inicial en la descripción de soluciones basadas en Redes Neuronales Artificiales a través del uso de Patrones de Software.

Para llevar a cabo este trabajo se analizó un conjunto de problemas resueltos con Redes Neuronales Artificiales y se propone un Sistema de Patrones de Software como una heurística para su selección. Partiendo del análisis de los problemas, se identifica las características más relevantes que impliquen el uso de una Red Neuronal Artificial como solución.

6.2 Objetivo

La presente tesis analiza un conjunto de problemas resueltos con Redes Neuronales Artificiales, y propone el Sistema de Patrones de Software como una heurística para la selección de Redes Neuronales Artificiales. Partiendo del análisis de los problemas que se resuelven con Redes Neuronales, se identifica las características más relevantes que impliquen su uso como una solución.

La selección de un Patrón de Software para Redes Neuronales se basa en las características del problema y la propia Red Neuronal que se utiliza como solución. En la presente tesis, se verifica la presencia de soluciones basadas en tipos particulares de redes neuronales en diferentes contextos y repetidamente, de tal modo que se puede afirmar que efectivamente representan un Patrón de Software.

Este trabajo es una guía para el diseño de nuevas soluciones tomando en cuenta lo que ya se conoce. Por lo tanto se propone una aproximación de Patrones de Software a partir de las soluciones que existen y las características más relevantes para el uso de una Red Neuronal Artificial.

6.3 Trabajo Futuro

Para la descripción de los patrones desarrollados en esta tesis, se considera un grupo representativo de problemas resueltos con redes neuronales. De tal manera que se deja como trabajo futuro desarrollar ampliamente las fuerza asociadas a cada patrón.

En Ingenieria de Software los patrones son más conocidos que en Redes Neuronales Artificiales. Por lo que se espera que este trabajo sirva de guía para desarrollar soluciones descritas como una red neuronal a partir de las características de los problemas.

Un conjuto de patrones que proporciona soluciones a problemas de diseño presentes en un dominio específico se llama lenguaje de patrones. En el caso de redes neuronales artificiales este trabajo podría complementarse de tal manera que se integre un Lenguaje de Patrones para Redes Neuronales Artificiales que describa aspectos de diseño.

Existen Patrones idiomáticos que están orientados a implementar aspectos específicos de diseño utilizando un lenguaje de programación determinado. En esta tesis se deja como trabajo futuro la implementación y programación de los algorit-

mos básicos de una red neuronal utilizando un lenguaje de programación dado. De tal manera que se construyan patrones idiomáticos para redes neuronales artificiales.

En general los patrones son flexibles a cambios. Es decir, a través del tiempo lo patrones pueden ser modificados de acuerdo a los cambios y evolución de la tecnología o las nuevas formas de trabajar. De tal manera que este sistema queda abierto para ser actualizado o complementado.

Por lo tanto, se espera que este trabajo se un buen punto de partida para desarrollar patrones de software en el área de Redes Neuronales a fin de contar con guías para describir soluciones que capturen la experiencia de desarrolladores y programadores.

Bibliografía

[Alexander, 1979] Alexander, C. (1979). *The Timeless Way of Building.* Oxford University Press, New York.

[Alexander et al., 1977] Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., and Angel, S. (1977). *A Pattern Language.* Oxford University Press, New York.

[Appleton, 1997] Appleton, B. (1997). Patterns and software: Essential concepts and terminology. http://www.enteract.com/˜bradapp.

[Benítez et al., 2005] Benítez, P. H., García, N. F., and Thompson, H. (2005). Fault classification based upon self organizing feature maps and principal component analysis for inertial sensor drift.

[Buschmann et al., 1996] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., and Stal, M. (1996). *Pattern-Oriented Software Architecture A System of Patterns.* John Wiley and Sons Ltd, Chichester.

[Caudill and C., 1992] Caudill, M. and C., B. (1992). *Understanding Neural Networks: Computer Explorations*, volume 1: Basic Networks. The MIT Press, London, England.

[Chen and Patton, 1999] Chen, J. and Patton, R. (1999). *Robust model-based fault diagnosis for dynamic systems.* Kluwer academic.

[Coplien, 2001] Coplien, J. (2001). *Tutorial: Foundation of Pattern Concepts and Pattern Writing.* Bell Labs/USA and University of Manchester UK.

[Coplien, 1994] Coplien, J. O. (1994). Software desing pattern: Common questions ans answers. *in Proc. OBJECT EXPO'94 Conf.*

[Corchado et al., 2000] Corchado, J. M., Díaz, F., Borrajo, L., and Fernández, F. (2000). *Redes Neuronales Artificiales, Un enfoque práctico*. Servicio de Publicacións de Univerdidade de Vigo, España.

[Fausett, 1994] Fausett, L. (1994). *Fundamentals of neural networks Architectures, algorithms, and applications*. Prentice-Hall, United States of America.

[Freeman and Skapura, 1992] Freeman, J. A. and Skapura, D. M. (1992). *Neural Networks. Algorithms, Applications and Progamming Techniques*. Addison Wesley, United States of American.

[Gamma et al., 1995] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns, Elements of Reusable Object Oriented Software*. Addison Wesley.

[Gateour, 2001] Gateour (2001). Página del modelo admire 2004. htth://www.foi.se/admire/.

[Hassoon, 1995] Hassoon, M. H. (1995). *Fundamentals of Artificial neural network*. The MIT Press, London England.

[Hilera and Martínez, 1995] Hilera, J. R. and Martínez, V. J. (1995). *Redes Neuronales Artificiales Fundamentos, modelos y aplicaciones*. RA-MA, Madrid, España.

[Hu and Hwang, 2002] Hu, Y. H. and Hwang, J. (2002). *Handbook of Neural Network Signal Processing*. CRC Press, New York Washington, D.C.

[Kohonen, 1990] Kohonen, T. (1990). *The Self-Organizing Map*, volume 78. Proceeding of the IEEE.

[Lea, 1993] Lea, D. (1993). Christopher alexander: An introduction for object-oriented designers. SUNY Oswego/ NY CASE Center.

[Luo and Unbehauen, 1997] Luo, F. and Unbehauen, R. (1997). *Applied Neural Networks for Signal Processing*. Cambridge University Press, United States of America.

[Nelles, 2001] Nelles, O. (2001). *Nonlinear System Identifation. From classical Approaches to Neural Networks and Fuzzy Models*. Springer-Verlay Berling Heidelberng, Germany.

[Pal and Mitra, 1999] Pal, S. K. and Mitra, S. (1999). *Neuro-Fuzzy Pattern Recognition, Methods in Soft Computing*. Wiley Series on Intelligent Systems. John Wiley & Sons inc., United States of America.

[Patton et al., 2000] Patton, R. J., Frank, P. M., and Clark, R. N. E. (2000). *Issues of Fault Diagnosis for Dynamic Systems*. Springer, Great Britain.