Optimización en el entrenamiento del Perceptrón Multicapa

Hevert Vivas

Universidad del Cauca
Facultad de Ciencias Naturales, Exactas y de la Educación
Departamento de Matemáticas
Maestría en Ciencias Matemáticas
Popayán, 2014

Optimización en el entrenamiento del Perceptrón Multicapa

Hevert Vivas

Trabajo de investigación presentado como requisito parcial para optar al título de Magíster en Ciencias Matemáticas

Directora
Dra. Rosana Pérez Mera
Profesora de la Universidad del Cauca

Codirector
Dr. Héctor Jairo Martínez Romero
Profesor de la Universidad del Valle

Universidad del Cauca
Facultad de Ciencias Naturales, Exactas y de la Educación
Departamento de Matemáticas
Maestría en Ciencias Matemáticas
Popayán
2014

Agradecimientos

A mi esposa *Ximena* y a mis hijos *Camilo* y *Juliana*, porque fueron mi inspiración; a mi profesora, directora de tesis y amiga *Rosana*, por la paciencia que me tuvo y el ánimo que me dio para que no renunciara a este propósito; a mi codirector *Héctor Jairo Martínez*, por sus sabios consejos; al profesor y amigo *Diego Correa* y a mis compañeros y amigos *Mauricio* y *Favián*, por su colaboración incondicional.

Hevert V.

Tabla de Contenidos

Índice de figuras

Capítulo 1

Introducción

Cuando leemos estas palabras, estamos usando una red neuronal biológica compleja, formada por un gran conjunto de interconexiones de neuronas que nos facilitan la lectura, respiración, movimiento y pensamiento. Cada una de nuestras neuronas biológicas, un rico ensamble de tejido y química, tiene la complejidad, aunque no la velocidad, de un microprocesador [15].

Se calcula que los seres humanos tenemos alrededor de 100 mil millones de neuronas en nuestro cerebro y esas neuronas, que se encuentran conectadas entre sí, forman una red amplia y expandida, denominada *red neuronal*. Gracias al trabajo de estas neuronas, somos capaces de asimilar lo que percibimos en nuestro entorno y de adquirir, a través de esa información, el conocimiento de lo que nos rodea. Dichos conocimientos quedan atrapados en esta red, pero para ello es necesario un proceso de entrenamiento o aprendizaje [4].

El hombre siempre ha buscado imitar las buenas características de algunos animales y de él mismo mediante la construcción de máquinas. Las *Redes Neuronales Artificiales* (RNA) son sistemas de procesamiento de información que funcionan de manera similar a las redes neuronales biológicas. Estas redes tienen en común con el cerebro humano la distribución de las operaciones a realizar en una serie de elementos básicos que, por analogía con los sistemas biológicos, se denominan *neuronas artificiales*, las cuales están relacionadas entre sí, mediante una serie de conexiones que se conocen como *pesos sinápticos*. Estos pesos y conexiones varían con el tiempo mediante un proceso, usualmente iterativo, conocido como *aprendizaje o entrenamiento* de la red.

En el **entrenamiento** o **aprendizaje** de una red, la determinación de dichos pesos se hace de tal manera que al final de un buen proceso de entrenamiento, la información *relevante* de los datos de entrada queda incorporada en la estructura de la red.

Dos tipos básicos de **aprendizaje** son el *supervisado* y el *no supervisado*. En el primero, se presenta a la red un conjunto de *patrones*, junto con la *salida deseada*, e iterativamente, la red ajusta sus pesos hasta que su salida tiende a ser la deseada utilizando para ello, información del *error* que comete en cada paso. En el segundo tipo de aprendizaje, se presenta a la red un conjunto de patrones sin tener en cuenta la respuesta deseada. La red, mediante alguna regla de aprendizaje, estima una *función de densidad de probabilidad* que permite reconocer regularidades en el conjunto de entradas, extraer rasgos y agrupar patrones según su similitud. Las reglas de aprendizaje supervisadas suelen ser computacionalmente más complejas, pero sus resultados suelen ser más exactos [14].

Dentro del grupo de modelos de redes neuronales artificiales están *el perceptrón simple, la adalina y el perceptrón multicapa*. Estos modelos presentan un gran interés histórico, pues su evolución representa la historia misma de las redes neuronales artificiales [14].

El perceptrón multicapa es una red neuronal unidireccional constituida por tres capas o más: una capa de entrada, otra de salida y el resto de las capas intermedias denominadas capas ocultas. El tipo de aprendizaje de esta red es *supervisado* y se hace mediante el algoritmo denominado *retropropagación* de errores (*backpropagation*). Esta red, junto con su algoritmo de aprendizaje, es el modelo más empleado en aplicaciones prácticas (se estima que un $70\,\%$ de los desarrollos con redes neuronales hacen uso de alguna de sus variantes [14]).

Como mencionamos anteriormente, en el proceso de aprendizaje de una red neuronal artificial está inmersa una *función error*, la cual depende explícitamente de los pesos *sinápticos* y proporciona el error que comete la red, comparando en cada iteración del proceso de entrenamiento, la salida obtenida con la salida deseada. Matemáticamente, la *función error* es un campo escalar; es decir, una función de valor real y variable vectorial,

$$\begin{aligned} E: \ \mathbb{R}^n &\longrightarrow \mathbb{R} \\ \boldsymbol{w} &\longmapsto E(\boldsymbol{w}), \end{aligned}$$

donde el vector $\boldsymbol{w}$ tiene como componentes los *pesos sinápticos* de la red. La gráfica de este campo escalar es una hipersuperficie. El proceso de aprendizaje en una red neuronal artificial consiste en encontrar un vector $\boldsymbol{w}$ (es decir una configuración de pesos) que *minimice* la función E. Así, asociado al rendimiento de una red neuronal, tenemos el

siguiente problema de minimización,

$$\begin{array}{c} Minimizar \quad E(\boldsymbol{w}) \\ \boldsymbol{w} \in \mathbb{R}^n \end{array} \tag{1.1}$$

donde

$$E(\boldsymbol{w}) = \frac{1}{2} \sum_{\mu=1}^{p} \sum_{i=1}^{m} (\boldsymbol{t}_i^{\mu} - \boldsymbol{z}_i^{\mu}(\boldsymbol{w}))^2 \equiv \frac{1}{2} \sum_{\mu=1}^{p} \| \boldsymbol{t}^{\mu} - \boldsymbol{z}^{\mu}(\boldsymbol{w}) \|_2^2 \tag{1.2}$$

y $\boldsymbol{z}_i^{\mu}(\boldsymbol{w})$ y $\boldsymbol{t}_i^{\mu}$ son, respectivamente, las salidas obtenidas por la red y las salidas deseadas, para un conjunto de p patrones de entrenamiento y unos pesos $\boldsymbol{w}$ determinados.

El problema (1.1), con E definida por (1.2), es un *problema de mínimos cuadrados no lineales*. Algoritmos para resolver este tipo de problemas en un contexto general, han sido ampliamente estudiados [5].

En este trabajo de investigación, proponemos e implementamos por primera vez, el *método secante estructurado* para el entrenamiento del perceptrón multicapa y analizamos su desempeño numérico comparándolo con métodos ampliamente usados con el mismo propósito, tales como, *Gauss-Newton* y *Levenverg-Marquardt* [14]. Pruebas numéricas preliminares muestran un buen desempeño numérico del método propuesto.

La presentación de este documento la organizamos de la siguiente forma.

En el **Capítulo 2**, presentamos el problema de *Mínimos Cuadrados No Lineales (MCNL)* como un caso particular de *minimización sin restricciones*. Analizamos diferentes métodos de solución, sus algoritmos y propiedades de convergencia. En primer lugar, describimos métodos de propósito general, como el *método de Newton* y los *métodos secantes*, y en segundo lugar, métodos que aprovechan la estructura del problema MCNL, como el *método de Gauss Newton*, *Levenberg-Marquardt* y *secantes estructurados*.

En el **Capítulo 3**, presentamos en forma descriptiva las redes neuronales artificiales, su evolución histórica, estructura y funcionamiento, profundizando en el *perceptrón multicapa* y en su algoritmo de entrenamiento denominado *retropropagación* de errores.

En el **Capítulo 4**, proponemos e implementamos por primera vez, el método *secante estructurado* para el entrenamiento del perceptrón multicapa y analizamos numéricamente su desempeño para diferentes actualizaciones secantes. Además, comparamos su desempeño con los métodos de *Gauss-Newton* y *Levenverg-Marquardt*, ampliamente utilizados

con el mismo propósito.

Finalmente, en el **Capítulo 5**, hacemos algunos comentarios finales y propuestas de trabajos futuros sobre el tema.

Capítulo 2

Mínimos cuadrados no lineales

El método de mínimos cuadrados tiene una larga historia que se remonta a los principios del siglo XIX. En Junio de 1801, ***Zach****, un astrónomo que* ***Gauss*** *había conocido dos años antes, publicaba las posiciones orbitales del cuerpo celeste* ***Ceres****, un nuevo pequeño planeta descubierto por el astrónomo italiano* ***Giuseppe Piazzi*** *en ese mismo año. Desafortunadamente,* ***Piazzi*** *sólo pudo observar 9 grados de su órbita antes de que este cuerpo desapareciera detrás del sol.* ***Zach*** *publicó varias predicciones de su posición, incluyendo una de* ***Gauss*** *que difería bastante del resto. Cuando* ***Ceres*** *fue redescubierto por* ***Zach*** *en diciembre de 1801, estaba casi exactamente en donde* ***Gauss*** *había predicho. Aunque todavía no había revelado su método,* ***Gauss*** *había descubierto el método de mínimos cuadrados. En un trabajo brillante, logró calcular la órbita de* ***Ceres*** *a partir de un número reducido de observaciones* [2].

En este capítulo, abordamos un problema especial de *minimización sin restricciones* que surge con frecuencia en problemas prácticos, tales como *ajuste de curvas, reconocimiento* y *clasificación de patrones* y en *redes neuronales artificiales*, entre otros. Por su estructura particular, el problema es, por sí mismo, un tema de investigación en el área de optimización. Nos referimos al problema de *Mínimos Cuadrados No Lineales (MCNL)*. En general, los métodos usados para resolver este problema son iterativos; es decir, a partir de una aproximación inicial de la solución, generan una sucesión de aproximaciones que bajo ciertas hipótesis, se espera converja a la solución del problema.

2.1. Planteamiento del problema

Dada una función no lineal y dos veces continuamente diferenciable definida por

$$\begin{array}{rcl} R: \ \mathbb{R}^n & \longrightarrow & \mathbb{R}^m, \ m \geq n, \\ \boldsymbol{x} & \mapsto & R(\boldsymbol{x}) = \begin{pmatrix} r_1(\boldsymbol{x}) \\ \vdots \\ r_m(\boldsymbol{x}) \end{pmatrix}, \end{array}$$

el problema de *Mínimos Cuadrados No Lineales (MCNL)* consiste en resolver el *problema de minimización sin restricciones*

$$\begin{array}{ll} Minimizar & f(\boldsymbol{x}) = \dfrac{1}{2} R(\boldsymbol{x})^T R(\boldsymbol{x}) \cdot \\ \boldsymbol{x} \in \mathbb{R}^n & \end{array} \tag{2.1}$$

Observemos que

$$f(\boldsymbol{x}) = \frac{1}{2} R(\boldsymbol{x})^T R(\boldsymbol{x}) = \frac{1}{2} \sum_{i=1}^{m} r_i(\boldsymbol{x})^2 \cdot$$

En algunas ocasiones, el sistema de ecuaciones no lineales

$$r_i(\boldsymbol{x}) = 0, \quad i = 1, \ldots, m, \tag{2.2}$$

es *sobredeterminado* y por lo general, no tiene solución, por lo cual, se hace necesario "suavizar" el sentido (o significado) de solución, buscando un vector $\boldsymbol{x}$ tal que

$$r_i(\boldsymbol{x}) \approx 0, \quad i = 1, \ldots, m, \tag{2.3}$$

lo cual se hace, resolviendo el problema (2.1).

La estructura particular del problema (2.1), se observa claramente en las expresiones para el *vector gradiente* y *la matriz hessiana* de f, en $\boldsymbol{x}$. En efecto, después de realizar algunos cálculos algebraicos, tenemos

$$\nabla f(\boldsymbol{x}) = J(\boldsymbol{x})^T R(\boldsymbol{x}) \quad \text{y} \quad \nabla^2 f(\boldsymbol{x}) = J(\boldsymbol{x})^T J(\boldsymbol{x}) + S(\boldsymbol{x}), \tag{2.4}$$

donde, $J(\boldsymbol{x}) \in \mathbb{R}^{m\times n}$ es la *matriz jacobiana* de R en $\boldsymbol{x}$, dada por

$$J(\boldsymbol{x}) = \begin{pmatrix} \nabla r_1(\boldsymbol{x})^T \\ \nabla r_2(\boldsymbol{x})^T \\ . \\ . \\ . \\ \nabla r_m(\boldsymbol{x})^T \end{pmatrix} \tag{2.5}$$

y

$$S(\boldsymbol{x}) = \sum_{i=1}^{m} r_i(\boldsymbol{x})\nabla^2 r_i(\boldsymbol{x}). \tag{2.6}$$

Observemos que la matriz $J(\boldsymbol{x})$ contiene solamente información de *primer orden* (primeras derivadas parciales) y $S(\boldsymbol{x})$ contiene información de *segundo orden* (es una combinación lineal de matrices hessianas). Esta estructura especial de la matriz hessiana de f es la que se aprovecha en algunos de los métodos usados para resolver el problema (2.1) y es la razón por la cual no se usan métodos de propósito general para resolver el mismo.

Es importante mencionar que la información de primer orden es relativamente fácil de calcular, mientras que la de segundo orden es numéricamente difícil de calcular, ya que involucra el cálculo de m hessianos, lo que implica un alto costo computacional [5],[20],[8].

Si consideramos que la función R definida en (2.1) es lineal; es decir, $R(\boldsymbol{x}) = A\boldsymbol{x} + \boldsymbol{b}$, donde $A \in \mathbb{R}^{m\times n}$ y $b \in \mathbb{R}^m$, entonces el problema (2.1) se convierte en un problema de *mínimos cuadrados lineales*, el cual es igual a resolver el siguiente problema

$$\begin{array}{ll} Minimizar & g(\boldsymbol{x}) = \dfrac{1}{2} \parallel A\boldsymbol{x} + \boldsymbol{b} \parallel_2^2 . \\ \boldsymbol{x} \in \mathbb{R}^n & \end{array} \tag{2.7}$$

Debido a que la función g es *diferenciable* en $\mathbb{R}^n$, se cumple que en cualquier *minimizador,* el gradiente de g se anula. Teniendo en cuenta que $g(\boldsymbol{x})$ se puede escribir como $g(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T A^T A\boldsymbol{x} + \boldsymbol{b}^T A\boldsymbol{x} + \frac{1}{2}\boldsymbol{b}^T\boldsymbol{b}$, entonces resolver el problema (2.7) es equivalente a resolver el sistema de ecuaciones lineales

$$\nabla g(\boldsymbol{x}) = A^T A\boldsymbol{x} + A^T\boldsymbol{b} = \boldsymbol{0}, \tag{2.8}$$

donde si A es una matriz de rango completo, se tiene que

$$\boldsymbol{x} = -[A^T A]^{-1} A^T\boldsymbol{b}. \tag{2.9}$$

Para más detalle del caso lineal, se puede consultar [20],[5] o [8]. Sin embargo, dado que nuestro interés es el caso *no lineal*, en lo que sigue nos enfocaremos en algunos de los métodos que resuelven problemas de este tipo.

2.2. Métodos de solución

En esta sección, describimos algunos métodos de solución del problema de *mínimos cuadrados no lineales* (2.1). Incluimos, dada su popularidad e importancia, métodos de propósito general para resolver problemas de minimización tales como el *método de Newton* y los *métodos secantes*, los cuales debido a que no aprovechan la estructura particular del problema (2.1), no son apropiados para resolverlo y describimos métodos especialmente diseñados para resolver el problema, los cuales sí aprovechan su estructura, estos son: *Gauss Newton*, *Levenberg-Marquardt* y *secante estructurados*. Todos estos métodos coinciden en que una *iteración básica* incluye la *solución de un sistema de ecuaciones lineales* para posteriormente, generar *la iteración siguiente*.

Los métodos mencionados en el párrafo anterior pueden considerarse como de *búsqueda direccional* ya que, a partir de un punto $\boldsymbol{x}_k$, determinan una *dirección de descenso*[1] tal que, al movernos en esa dirección, encontramos otro punto $\boldsymbol{x}_{k+1}$ en el cual el valor de f disminuye; es decir, $f(\boldsymbol{x}_{k+1}) \leq f(\boldsymbol{x}_k)$.

En la discusión de los diferentes métodos para resolver (2.1), es importante distinguir entre problemas de *residuo cero*, *residuo pequeño* y *residuo grande*. Estos términos se refieren al valor del residuo en el minimizador $\boldsymbol{x}_*$ de (2.1).

En lo que sigue, para cada uno de los métodos, presentaremos su deducción junto con su algoritmo respectivo, sus propiedades de convergencia, ventajas y desventajas. Los métodos que veremos en este capítulo son locales; es decir, que sus buenas propiedades de convergencia dependen en gran parte de la buena elección del punto inicial, por tal motivo, es necesario incorporar estrategias de globalización (estrategias que permitan la convergencia a partir de cualquier punto inicial). Dos de estas estrategias son *búsqueda lineal* y *región de confianza*.

Métodos que usan la estrategia de *búsqueda lineal* generan, inicialmente, una *dirección de búsqueda* y después, buscan un *tamaño de paso* adecuado a lo largo de esa dirección, en

[1]Cuando f es continuamente diferenciable, decimos que $\boldsymbol{d}_k$ es una *dirección de descenso* para f a partir de $\boldsymbol{x}_k$, si se cumple que $\nabla f(\boldsymbol{x}_k)^T \boldsymbol{d}_k < 0$.

tanto que, los métodos que usan la estrategia de *región de confianza*, definen una región alrededor de una aproximación de la solución, dentro de la cual, el modelo de la función objetivo es una buena aproximación de esta. En dicha región, escogen la *aproximación siguiente* (*el paso*) como el minimizador del modelo. Si un *paso* no es aceptado, se reduce el tamaño de la región y se encuentra un nuevo minimizador [5] [20].

2.2.1. Método de Newton

La idea básica del método de *Newton* para resolver un problema de *minimización sin restricciones* consiste en construir, en cada iteración, $\boldsymbol{x}_k$, *un modelo local* de la función a minimizar para posteriormente, resolver el problema para este modelo. El *modelo local* que se considera es un modelo cuadrático basado en la serie de *Taylor* de la función f alrededor del punto $\boldsymbol{x}_k$,

$$m_k(\boldsymbol{x}) = f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)^T(\boldsymbol{x} - \boldsymbol{x}_k) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_k)^T\nabla^2 f(\boldsymbol{x}_k)^T(\boldsymbol{x} - \boldsymbol{x}_k)\cdot \tag{2.10}$$

Puesto que $m_k(\boldsymbol{x}) \approx f(\boldsymbol{x})$ en un entorno de $\boldsymbol{x}_k$, entonces para resolver el problema (2.1) resolvemos, en cada iteración, el problema

$$\begin{aligned} Minimizar \quad & m_k(\boldsymbol{x})\cdot \\ \boldsymbol{x} \in \mathbb{R}^n & \end{aligned} \tag{2.11}$$

La función modelo m_k es cuadrática; así, resolver el problema (2.11) es equivalente a resolver el sistema de ecuaciones lineales $\nabla m_k(\boldsymbol{x}) = \boldsymbol{0}$; es decir,

$$\nabla^2 f(\boldsymbol{x}_k)(\boldsymbol{x} - \boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k) = \boldsymbol{0}. \tag{2.12}$$

Equivalentemente,

$$\begin{aligned} \nabla^2 f(\boldsymbol{x}_k)\boldsymbol{d}_k &= -\nabla f(\boldsymbol{x}_k) \\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \boldsymbol{d}_k. \end{aligned} \tag{2.13}$$

Las igualdades (2.13) se conocen como una *iteración de Newton*. Si $\nabla^2 f(\boldsymbol{x}_k)$ es una matriz *definida positiva* [5], por tanto no singular, se tiene que

$$\boldsymbol{d}_k = -[\nabla^2 f(\boldsymbol{x}_k)]^{-1}\nabla f(\boldsymbol{x}_k), \tag{2.14}$$

la cual es una dirección de descenso. En efecto,

$$\nabla f(\boldsymbol{x}_k)^T \boldsymbol{d}_k = -\nabla f(\boldsymbol{x}_k)^T [\nabla^2 f(\boldsymbol{x}_k)]^{-1} \nabla f(\boldsymbol{x}_k) \tag{2.15}$$

y, puesto que $\nabla^2 f(\boldsymbol{x}_k)$ es una matriz *definida positiva*, concluimos que si $\nabla f(\boldsymbol{x}_k) \neq \mathbf{0}$, entonces

$$\nabla f(\boldsymbol{x}_k)^T [\nabla^2 f(\boldsymbol{x}_k)]^{-1} \nabla f(\boldsymbol{x}_k) > 0, \tag{2.16}$$

por lo cual, de (2.15) y (2.16), $\nabla f(\, \boldsymbol{x}_k)^T \boldsymbol{d}_k < 0$ y en consecuencia, $\boldsymbol{d}_k$ es, efectivamente, una dirección de descenso. Dicha dirección es llamada *dirección de Newton* o *paso de Newton*. Así, Si $\nabla^2 f(\boldsymbol{x}_k)$ es *definida positiva*, una iteración del *método de Newton* para el problema (2.1) está dada por

$$\boldsymbol{x}_{k+1} \;=\; \boldsymbol{x}_k - [J(\boldsymbol{x}_k)^T J(\boldsymbol{x}_k) + S(\boldsymbol{x}_k)]^{-1} J(\boldsymbol{x}_k)^T R(\boldsymbol{x}_k), \tag{2.17}$$

donde $J(\boldsymbol{x})$ y $S(\boldsymbol{x})$ fueron definidas anteriormente por (2.5) y (2.6), respectivamente.

La convergencia de este método es q-cuadrática, siempre y cuando $\nabla^2 f$ sea *Lipschitz continua* alrededor de $\boldsymbol{x}_k$ y $\nabla^2 f(\boldsymbol{x}_k)$ sea definida positiva [5].

Podemos decir que las propiedades de convergencia del *método de Newton* son muy buenas comparadas con otros métodos, sin embargo, como ya se mencionó antes, el término $S(\boldsymbol{x})$ es, computacionalmente, costoso de calcular.

2.2.2. Método Gauss-Newton

Como mencionamos en la Sección 2.1, cuando el sistema $R(\boldsymbol{x}) = \mathbf{0}$ no tiene solución, buscamos un vector $\boldsymbol{x}$ tal que $R(\boldsymbol{x}) \approx \mathbf{0}$. Para ello, construimos *un modelo afín* [5] de $R(\boldsymbol{x})$ al rededor de $\boldsymbol{x}_k$,

$$R(\boldsymbol{x}) \approx M_k(\boldsymbol{x}) = R(x_k) + J(x_k)(x - x_k), \tag{2.18}$$

donde $M_k : \mathbb{R}^n \longrightarrow \mathbb{R}^m$, $m > n$, y resolvemos el problema $M(\boldsymbol{x}) \approx \mathbf{0}$ como el problema de *mínimos cuadrados lineales*

$$\begin{array}{ll} Minimizar & \dfrac{1}{2} \parallel R(x_k) + J(x_k)(x - x_k) \parallel_2^2 \\ \boldsymbol{x} \in \mathbb{R}^n & \end{array} \tag{2.19}$$

el cual, es equivalente a (2.7). Para el caso en el cual la matriz $J(\boldsymbol{x}_k)$ es de rango completo, la solución teórica ya la analizamos en (2.9).

Una manera alternativa de ver el *método de Gauss-Newton* es como una variación del *método de Newton*, en la cual se ha hecho la aproximación $\nabla^2 E(\boldsymbol{w}_k) \approx J(\boldsymbol{w})^T J(\boldsymbol{w}_k)$.

Si, de nuevo suponemos que la matriz $J(\boldsymbol{x}_k)$ es de rango completo, la solución a (2.19) está dada por

$$\boldsymbol{x} - \boldsymbol{x}_k \;=\; -[J(\boldsymbol{x}_k)^T J(\boldsymbol{x}_k)]^{-1} J(\boldsymbol{x}_k)^T R(\boldsymbol{x}_k) = \boldsymbol{s}_G, \tag{2.20}$$

donde $\boldsymbol{s}_G$ se conoce como *paso de Gauss-Newton*. Sin embargo, puede ser más práctico resolver el problema (2.19) vía la factorización QR de la matriz $J(\boldsymbol{x}_k)$ [20] [5] [8].

Una iteración básica del *método de Gauss-Newton* está dada por

$$\begin{aligned} J(\boldsymbol{x}_k)^T J(\boldsymbol{x}_k)\boldsymbol{s}_G &= J(\boldsymbol{x}_k)^T R(\boldsymbol{x}_k) \\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \boldsymbol{s}_G. \end{aligned}$$

o equivalentemente,

$$\boldsymbol{x}_{k+1} \;=\; \boldsymbol{x}_k - [J(\boldsymbol{x}_k)^T J(\boldsymbol{x}_k)]^{-1} J(\boldsymbol{x}_k)^T R(\boldsymbol{x}_k) = \boldsymbol{x}_k + \boldsymbol{s}_G. \tag{2.21}$$

Al igual que en la dirección de *Newton* y dado que la matriz $J(\boldsymbol{x}_k)^T J(\boldsymbol{x}_k)$ es *definida positiva*, no es difícil demostrar que $\boldsymbol{s}_G$ es una dirección de descenso, por lo cual la iteración de *Gauss-Newton* está bien definida.

El siguiente teorema, cuya demostración se puede encontrar en [5], permite hacer un análisis de la convergencia del *método de Gauss-Newton*.

Teorema 2.1. [5] *Sean* $R : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ *y* $f(\boldsymbol{x}) = \frac{1}{2} R(\boldsymbol{x})^T R(\boldsymbol{x})$ *dos veces continuamente diferenciable en un conjunto abierto y convexo* $D \in \mathbb{R}^n$. *Supongamos que* $J(\boldsymbol{x}) \in Lip_\gamma(D)$ *con* $\|J(\boldsymbol{x})^{-1}\|_2 \leq \alpha$ *para todo* $\boldsymbol{x} \in D$, *y que existen* $\boldsymbol{x}_* \in D$ *y* $\lambda, \sigma \geq 0$, *tales que* $J(\boldsymbol{x}_*)^T R(\boldsymbol{x}_*) = 0$, *donde* λ *es el valor propio más pequeño de* $J(\boldsymbol{x}_*)^T J(\boldsymbol{x}_*)$ *y*

$$\| \, (J(\boldsymbol{x}) - J(\boldsymbol{x}_*))^T R(\boldsymbol{x}_*) \, \|_2 \;\leq\; \sigma \;\| \, \boldsymbol{x} - \boldsymbol{x}_* \, \|_2, \tag{2.22}$$

para todo $\boldsymbol{x} \in D$. *Si* $\sigma < \lambda$, *entonces para cualquier* $c \in (1, \lambda/\sigma)$, *existe* $\epsilon > 0$ *tal que para todo* $\boldsymbol{x}_0 \in N(\boldsymbol{x}_*, \epsilon)$ *la sucesión generada por el método de Gauss-Newton*

$$\boldsymbol{x}_{k+1} \;=\; \boldsymbol{x}_k - [J(\boldsymbol{x}_k)^T J(\boldsymbol{x}_k)]^{-1} J(\boldsymbol{x}_k)^T R(\boldsymbol{x}_k),$$

está bien definida, converge a $\boldsymbol{x}_*$ *y satisface que*

$$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_*\|_2 \;\leq\; \frac{c\sigma}{\lambda}\|\boldsymbol{x}_k - \boldsymbol{x}_*\|_2 \;+\; \frac{c\alpha\gamma}{2\lambda}\|\boldsymbol{x}_k - \boldsymbol{x}_*\|_2^2, \tag{2.23}$$

y

$$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_*\|_2 \;\leq\; \frac{c\sigma + \lambda}{2\lambda}\|\boldsymbol{x}_k - \boldsymbol{x}_*\|_2 \;<\; \|\boldsymbol{x}_k - \boldsymbol{x}_*\|_2. \tag{2.24}$$

Corolario 2.1. [5] *Si las hipótesis del* **Teorema 2.1** *se satisfacen y* $R(\boldsymbol{x}_*) = 0$ *, entonces existe* $\epsilon > 0$ *tal que para todo* $\boldsymbol{x}_0 \in N(\boldsymbol{x}_*, \epsilon)$*, la sucesión* $\boldsymbol{x}_k$ *generada por el método de Gauss-Newton está bien definida y converge* $q-$*cuadráticamente a* $\boldsymbol{x}_*$*.*

A partir de (2.22) y de la igualdad ([5] Ejercicio 2.5.3)

$$(J(\boldsymbol{x}) - J(\boldsymbol{x}_*))^T R(\boldsymbol{x}_*) = S(\boldsymbol{x}_*)(\boldsymbol{x} - \boldsymbol{x}_*) + O(\|\boldsymbol{x} - \boldsymbol{x}_*\|_2^2),$$

podemos ver que, la constante σ es una medida de la no linealidad y del tamaño del residuo del problema. Así, si R es lineal o $R(\boldsymbol{x}_*) = 0$, se deduce que $\sigma = 0$.

El cociente σ/λ es una medida relativa de la no linealidad y tamaño del residuo del problema. De esta manera, el **Teorema 2.1** indica que la rapidez de convergencia del método de *Gauss-Newton* decrece cuando el grado de no linealidad o el tamaño residual del problema crece, incluso si estas medidas son muy grandes, el método podría no converger.

A pesar de que el método de *Gauss-Newton* presenta varios problemas, es la base de algunos métodos prácticos, importantes y exitosos [5].

2.2.3. Método de Levenberg-Marquardt

El algoritmo de *Levenberg-Marquardt* es un método en el cual se presenta una ligera modificación sobre el método tradicional de *Newton*, consistente en aproximar la matriz hessiana mediante $\nabla^2 E(\boldsymbol{w}_k) \approx J(\boldsymbol{w}_k)^T J(\boldsymbol{w}_k) + \mu_k I$ y por lo tanto

$$\boldsymbol{x}_{k+1} \;=\; \boldsymbol{x}_k - [J(\boldsymbol{x}_k)^T J(\boldsymbol{x}_k) + \mu_k I]^{-1} J(\boldsymbol{x}_k)^T R(\boldsymbol{x}_k). \tag{2.25}$$

Este método surgió para evitar las dificultades del *método Gauss-Newton* cuando, a lo largo del proceso iterativo, en algún punto $\boldsymbol{x}_k$, la matriz Jacobiana no tiene *rango completo* o está *mal condicionada*. Cuando $\mu_k = 0$, la dirección de búsqueda es la dirección de *Gauss-Newton* y si μ_k toma un valor muy grande, la dirección de descenso es paralela con la dirección de *máximo descenso* ($\boldsymbol{d}_k = -\nabla f(\boldsymbol{x}_k)$), por lo cual podemos ver que este método combina los *métodos de descenso por gradiente y Gauss-Newton* [5] [4].

El método (2.25), del cual existen varias versiones que varían según la estrategia para escoger μ_k, fue sugerido por *Levenberg* en (1944) y luego por *Marquardt* en (1963).

El método de *Levenberg-Marquardt* también puede verse como una modificación del método de *Gauss-Newton* en la cual, usamos la estrategia de *región de confianza* para encontrar el punto $\boldsymbol{x}_{k+1}$, basándonos en el modelo (2.18) para resolver el subproblema de mínimos cuadrados lineales con restricciones

$$\begin{aligned} Minimizar \quad & \frac{1}{2} \parallel R(\boldsymbol{x}_k) + J(\boldsymbol{x}_k)(\boldsymbol{x} - \boldsymbol{x}_k) \parallel_2^2, \\ sujeto\ a \quad & \|\boldsymbol{x} - \boldsymbol{x}_k\| \leq \delta_k. \end{aligned} \tag{2.26}$$

La solución de (2.26) también es dada por (2.25), solo que, en este caso $\mu_k = 0$ si $\|J(\boldsymbol{x}_k)^T J(\boldsymbol{x}_k)]^{-1} J(\boldsymbol{x}_k)^T R(\boldsymbol{x}_k)\|_2 \leq \delta_k$, en cuyo caso, la dirección de *Gauss-Newton* es la solución y $\mu_k > 0$ en otro caso [5].

Las propiedades de convergencia del método de *Levenberg-Marquardt* son similares a las del método de *Gauss-Newton*, sin embargo, muchas implementaciones de su algoritmo superan las desventajas propias del método de *Gauss-Newton*; entre ellas, la implementación dada por *Moré*, contenida en el *MINPACK*[2], ha resultado muy exitosa en la práctica, por lo cual este método es recomendado para resolver *Problemas de Mínimos Cuadrados No Lineales*. Otra de las grandes ventajas de este método es que está bien definido aún cuando la matriz $J(\boldsymbol{x}_k)$ no sea de *rango completo*. Se ha demostrado que varias versiones del algoritmo de *Levenberg-Marquardt* tienen convergencia global, como las dadas por *Powell* (1975), *Osborne* (1976) y *Moré* (1977) [5].

[2]MINPACK es una biblioteca de subrutinas de FORTRAN altamente portátil, robusta y confiable, usada para resolver sistemas de ecuaciones no lineales o *Problemas de Mínimos Cuadrados Lineales* y *No Lineales* [19].

2.2.4. Métodos Secantes

En la iteración de *Newton* (2.13), es necesario calcular la matriz hessiana de f en $\boldsymbol{x}_k$ y para encontrar el paso de *Newton*, vía factorización de *Cholesky*, se requiere que la matriz $\nabla^2 f(\boldsymbol{x}_k)$ sea *definida positiva*; por lo cual, a no ser que ella tenga una estructura particular, dicha factorización es muy costosa computacionalmente. Si a lo anterior le agregamos el hecho de que, en general, el sólo cálculo del hessiano de f en $\boldsymbol{x}_k$ ya es muy costoso, se hace indispensable tener métodos que resuelvan el mismo problema de minimización, sin tener que realizar estos cálculos. De esta manera, surgen los denominados métodos *cuasi Newton*, los cuales usan una aproximación de la matriz hessiana en lugar de ella misma. Así, si B_k es una "buena" aproximación de la matriz hessiana $\nabla^2 f(\boldsymbol{x}_k)$, entonces la *dirección cuasi Newton* será la solución al sistema de ecuaciones lineales

$$B_k \boldsymbol{d}_k = -\nabla f(\boldsymbol{x}_k). \tag{2.27}$$

Cuando decimos una "buena" aproximación, nos referimos a que requerimos que B_k conserve las buenas propiedades del hessiano (simetría) y que permita encontrar direcciones de descenso (definida positiva). Así, al resolver un *problema de minimización*, usando un método *cuasi Newton*, ganamos estabilidad numérica, eficiencia y convergencia [21].

Si además, actualizamos la aproximación B_k de tal forma que satisfaga la llamada *ecuación secante*[3]

$$B_{k+1}(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k) = \nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k), \tag{2.28}$$

entonces surgen los denominados *métodos secantes*, cuya iteración básica es la siguiente,

$$\begin{aligned} B_k \boldsymbol{s}_k &= -\nabla f(\boldsymbol{x}_k) \\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \boldsymbol{s}_k \\ B_{k+1} \boldsymbol{s}_k &= \boldsymbol{y}_k, \end{aligned} \tag{2.29}$$

donde $\boldsymbol{s}_k = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k$, $\boldsymbol{y}_k = \nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k)$ y B_{k+1} es llamada *actualización secante.*

Existen varias *actualizaciones secantes* exitosas. Entre ellas están la *BFGS*, propuesta independientemente por *Broyden* en (1969) y *Fletcher*, *Goldfarb* y *Shano* en (1970), dada

[3]El nombre de *ecuación secante* se utiliza porque en el caso $n = 1$, B_{k+1} representa la pendiente de la recta secante a la gráfica de la función f' que une los puntos $(x_k, f'(x_k))$ y $(x_{k+1}, f'(x_{k+1}))$.

por

$$B_{k+1} = B_k + \frac{\boldsymbol{y}_k \boldsymbol{y}_k^T}{\boldsymbol{y}_k^T \boldsymbol{s}_k} - \frac{B_k \boldsymbol{s}_k \boldsymbol{s}_k^T B_k}{\boldsymbol{s}_k^T B_k \boldsymbol{s}_k} \tag{2.30}$$

y la actualización *DFP* propuesta por *Davidon* en (1959), *Fletcher* y *Powell* en (1963) definida por

$$B_{k+1} = B_k + \frac{(\boldsymbol{y}_k - B_k \boldsymbol{s}_k)\boldsymbol{y}_k^T + \boldsymbol{y}_k(\boldsymbol{y}_k - B_k \boldsymbol{s}_k)^T}{\boldsymbol{y}_k^T \boldsymbol{s}_k} - \frac{(\boldsymbol{y}_k - B_k \boldsymbol{s}_k)^T \boldsymbol{s}_k \boldsymbol{y}_k \boldsymbol{y}_k^T}{(\boldsymbol{y}_k^T \boldsymbol{s}_k)^2} \cdot \tag{2.31}$$

En la práctica, se recomienda la actualización *BFGS*, debido a que su desempeño numérico es mejor; sin embargo, la actualización *DFP* fue la primera actualización secante propuesta, por lo cual tiene un gran interés tanto histórico como analítico [21][5].

Infortunadamente, los *métodos secantes* tal y como han sido descritos hasta aquí, no aprovechan la estructura de la matriz hessiana dada en (2.4). Es decir, no aprovechan los cálculos del jacobiano ya realizados. Una alternativa para ello, la representan los llamados *métodos secantes estructurados*. Estos métodos son apropiados para problemas en los cuales la matriz hessiana, tal como sucede en el problema (2.1), se puede expresar en la forma

$$\nabla^2 f(\boldsymbol{x}) = C(\boldsymbol{x}) + S(\boldsymbol{x}),$$

donde $C(\boldsymbol{x})$ contiene información "fácil"de obtener y $S(\boldsymbol{x})$ contiene información que es "difícil"o imposible de calcular. Así, en un *método secante estructurado*, basta hacer una aproximación secante de la parte "difícil", $S(\boldsymbol{x})$, conservando el resto de la estructura.

En particular, para el problema de *mínimos cuadrados no lineales* (2.1), tenemos que $\nabla^2 f(\boldsymbol{x}) = C(\boldsymbol{x}) + S(\boldsymbol{x})$, donde

$$C(\boldsymbol{x}) = J(\boldsymbol{x})^T J(\boldsymbol{x}) \qquad y \qquad S(\boldsymbol{x}) = \sum_{i=1}^{m} r_i(\boldsymbol{x})^T \nabla^2 r_i(\boldsymbol{x}) \cdot \tag{2.32}$$

Con ello, en un método *secante estructurado*, para el problema de *mínimos cuadrados no lineales* (2.1), hacemos el proceso iterativo

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + s_k \\ A_{k+1} &= A_k + \Delta(s_k, \boldsymbol{y}_k^{\#}, A_k, v_k) \\ B_{k+1} &= A_{k+1} + C(\boldsymbol{x}_{k+1}), \end{aligned} \tag{2.33}$$

donde $\boldsymbol{s}_k$ es el paso *quasi-Newton* definido por (2.27), A_k es una aproximación a $S(\boldsymbol{x}_k)$ y $\Delta(\boldsymbol{s}_k, \boldsymbol{y}_k^{\#}, A_k, \boldsymbol{v}_k)$, llamada *corrección de actualización secante* que está definida por

$$\Delta(\boldsymbol{s}_k, \boldsymbol{y}_k^{\#}, A_k, \boldsymbol{v}_k) = \frac{(\boldsymbol{y}_k^{\#} - A_k\boldsymbol{s}_k)\boldsymbol{v}_k^T + \boldsymbol{v}_k(\boldsymbol{y}_k^{\#} - A_k\boldsymbol{s}_k)^T}{\boldsymbol{v}_k^T\boldsymbol{s}_k} - \frac{(\boldsymbol{y}_k^{\#} - A_k\boldsymbol{s}_k)^T\boldsymbol{s}_k\boldsymbol{v}_k\boldsymbol{v}_k^T}{(\boldsymbol{v}_k^T\boldsymbol{s}_k)^2}. \tag{2.34}$$

El vector $\boldsymbol{v}_k \in \mathbb{R}^n$ es denominado la *escala* y es a menudo una función de $\boldsymbol{s}$, $\boldsymbol{y}$ y B. Así, diferentes valores de la *escala* conducen a diferentes *actualizaciones*, por ejemplo:

$$\begin{aligned} PSB \qquad \boldsymbol{v} &= \boldsymbol{s} \\ DFP \qquad \boldsymbol{v} &= \boldsymbol{y} \end{aligned} \tag{2.35}$$

$$\begin{aligned} BFGS \qquad \boldsymbol{v} &= \boldsymbol{y} + \left[\frac{\boldsymbol{y}^T\boldsymbol{s}}{\boldsymbol{s}^TB\boldsymbol{s}}\right]^{1/2} B\boldsymbol{s} \\ SR1 \qquad \boldsymbol{v} &= \boldsymbol{y} - B\boldsymbol{s}. \end{aligned} \tag{2.36}$$

Los vectores $\boldsymbol{y}$ y $\boldsymbol{y}_k^{\#}$ son aproximaciones a $\nabla^2 f(\boldsymbol{x})\boldsymbol{s}$ y $\boldsymbol{S}(\boldsymbol{x}_k)\boldsymbol{s}$, respectivamente.

Dennis en (1976) y *Bartholomew-Biggs* en (1977) sugirieron independientemente la escogencia para $\boldsymbol{y}_k^{\#}$ dada por

$$\boldsymbol{y}_k^{\#} = (J(\boldsymbol{x}_{k+1})^T - J(\boldsymbol{x}_k)^T)R(\boldsymbol{x}_{k+1})$$

y *Al-Baali y Fletcher* en (1985) sugirieron usar

$$\boldsymbol{y}_k = \boldsymbol{y}_k^{\#} + J(\boldsymbol{x}_{k+1}^T)J(\boldsymbol{x}_{k+1})\boldsymbol{s}$$

en lugar de $\boldsymbol{y} = \nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k)$, mejorando así el comportamiento del código *NL2SOL* [4].

Entre las ventajas del método *secante estructurado* está, el hecho de que no es necesario el cálculo analítico de la matriz hessiana, sin embargo, se aprovecha su buena estructura, ya que se aproxima solo la parte del hessiano que contiene información de segundo orden, la cual es costosa de obtener computacionalmente o difícil de obtener analíticamente.

[4] *NL2SOL* es una biblioteca de FORTRAN90 que implementa un algoritmo para resolver problemas de mínimos cuadrados no lineales y fue creada por *John Dennis*, *David Gay* y *Roy Welsch*. Este es un método de implementación para el algoritmo secante estructurado que usa una estrategia de región de confianza para globalizarlo [16] [5].

La teoría de convergencia para los *métodos secante estructurados PSB*, *DFP* fue desarrollada en 1981 [6] mientras que la del método BFGS estructurado fue establecida en 1989 [17]. Una aplicación directa de esta teoría da la primera prueba de convergencia local y q-superlineal del método BFGS estructurado para el problema de mínimos cuadrados no Lineales, el cual es usado por Dennis, Gay, y Welsh en la versión actual del popular y exitoso código NL2SOL [7].

Capítulo 3

Redes Neuronales Artificiales (RNA)

Las redes neuronales artificiales son sistemas de procesamiento de información que copian esquemáticamente la estructura neuronal del cerebro (Figura 3.1) *para tratar de reproducir sus capacidades, por ello, son capaces de aprender de la experiencia a partir de las señales o datos provenientes del exterior*[14].

En este capítulo, describimos, en forma general, las *redes neuronales artificiales* desde su posición e importancia histórica y modelación matemática, hasta sus métodos (algoritmos) de *entrenamiento*. En particular, del *perceptrón multicapa*, una red neuronal artificial, cuyo algoritmo de entrenamiento denominado *retropropagación* de errores, hizo de ella una de las más usadas para resolver problemas de aplicación, que con algoritmos tradicionales eran muy difíciles o hasta imposibles de resolver. Sobre esta red, haremos una descripción detallada de su estructura y funcionamiento, centrándonos en su método de entrenamiento, el cual nos conduce a resolver un problema de *mínimos cuadrados no lineales* (ver **Capítulo 1**).

3.1. Breve reseña histórica

La historia de las *redes neuronales artificiales* está llena de creatividad individual en diferentes campos y ha sido documentada por varios autores. Un ejemplo de ello es el libro de *James Anderson* y *Edward Rosenfeld* titulado *Neurocomputing: Foundations of Research*, en el el cual se coleccionan y editan 43 documentos de interés histórico [15].

Figura 3.1: *Redes neuronales* [10].

Se puede decir que la construcción de *máquinas inteligentes* se inició durante la segunda guerra mundial, propiamente con el diseño de ordenadores analógicos para controlar cañones antiaéreos y de navegación. Esto inspiró a investigadores en diferentes áreas de la ciencia y la ingeniería para crear máquinas capaces de imitar algunas funciones propias de los animales, dando inicio a la *cibernética*, término usado para designar el estudio unificado del control y la comunicación entre los animales y las máquinas. Sin embargo, aunque estos computadores resolvían problemas muy complejos e incluso podían jugar ajedrez contra expertos y ganar, fue solo hasta la aparición de las *redes neuronales artificiales*, que hubo una máquina que desarrollara funciones, tan simples para el cerebro de una rana, como la de captar el vuelo de una mosca.

Desde antes de la aparición del primer computador hasta hoy, han ocurrido varios hechos que marcaron la historia de las *redes neuronales artificiales*; a continuación destacamos cronológicamente algunos de ellos [14] [1].

- *1888. Santiago Ramón y Cajal.* Descubrió que el sistema nervioso está compuesto por una red de células individuales (*neuronas*) ampliamente interconectadas.
- *1937. Alan Turing.* Emprendió el estudio formal de la computación e introdujo la *máquina de Turing.*

- *1943. Warren McCuloch (Neurofisiólogo)* y *Walter Pitts (Matemático)*. Concibieron los fundamentos de la computación neuronal y modelaron una red neuronal simple mediante circuitos eléctricos.

- *1944. John Von Newmann*. Concibió el computador basado en *lógica digital* que opera ejecutando una serie de instrucciones precisas; es decir, un *algoritmo*.

- *1949. Donald Hebb*. En su libro *The Organization of Behaviour*, establece una conexión entre psicología y fisiología. Fue el primero en explicar los procesos del aprendizaje desde un punto de vista psicológico, desarrollando una regla de aprendizaje que hoy en día es la base de las funciones de aprendizaje en las mayoría de las redes neuronales artificiales.

- *1957. Frank Rosenblatt*. Comenzó el desarrollo del *Perceptrón* (la red neuronal más antigua).

- *1960. Bernard Widrow* y *Marcial Hoff*. Desarrollaron el modelo *Adaline* (primera red neuronal aplicada a un problema real).

- *1969. Marvin Minsky* y *Seymour Papert*. Criticaron fuertemente el *Perceptrón*, lo que produjo un estancamiento de más de 10 años en el estudio de las *redes neuronales artificiales*, dando lugar a la introducción de *sistemas expertos*[1].

- *1974. Paul Werbos*. Desarrolló la idea básica del algoritmo de aprendizaje de propagación hacia atrás (*backpropagation*).

- *1985. John Hopfield*. Ayudó al renacimiento de las redes neuronales con su libro *Computación neuronal de decisiones en problemas de optimización*.

- *1986. David Rumelhart* y *G. Hinton*. Redescubrieron el algoritmo de aprendizaje de propagación hacia atrás (*backpropagation*), lo que mejoró notablemente el panorama oscuro dejado por las críticas de *Minsky* y *Papert*.

En la actualidad, son numerosos los trabajos que se realizan y publican cada año, las aplicaciones nuevas que surgen (sobretodo en el área de control) y las empresas que lanzan al mercado productos nuevos, tanto en hardware como en software (sobre todo para simulación).

Las *redes neuronales artificiales* están inspiradas en las redes neuronales biológicas del cerebro humano y por ello, presentan varias características que las hacen similares, no

[1]Los *sistemas expertos* son programas de computador complejos, en los cuales se reúne el conocimiento de una serie de expertos en un área particular en forma de reglas de decisión [14].

solo en su estructura, sino en su funcionamiento. Por ejemplo, ambas aprenden de la experiencia, generalizan de ejemplos previos a ejemplos nuevos y captan las características relevantes de una serie de datos. Así que, para entender como funciona una red neuronal artificial, primero es conveniente entender el funcionamiento de las redes neuronales biológicas (Figura 3.2).

3.2. Redes neuronales biológicas

Figura 3.2: *Red neuronal biológica* [11].

Una característica sobresaliente del cerebro frente al computador es la alta interconexión de sus elementos constituyentes más pequeños, *las neuronas*. Esta capacidad de operar en paralelo le permite realizar rápidamente tareas que necesitan una gran cantidad de cálculos y tiempo en potentes computadores.

Se estima que el sistema nervioso contiene alrededor de 100 mil millones de neuronas. Vistas en un microscopio, estas neuronas presentan diferentes formas. En general, la mayoría tienen un aspecto similar compuesto por un *cuerpo* o *soma*, del cual surge unas ramificaciones denominadas *dendritas* y una fibra tubular llamada *axón* que también se ramifica en su extremo final para conectarse con otras neuronas y formar así, las llamadas *redes neuronales biológicas* o simplemente *redes neuronales* (Figura 3.3).

Comparadas con un microprocesador elemental, de los que se usan en un computador sencillo, *las neuronas* son procesadores de información mucho más simples[2], sin embargo, funcionan de manera análoga, en el sentido de que tienen las *dendritas* como canal de entrada de la información procedente del exterior (en el caso de las *neuronas receptoras* o

[2]En realidad, la virtud de la *redes neuronales* radica en la alta interconexión entre las neuronas y no de su capacidad individual.

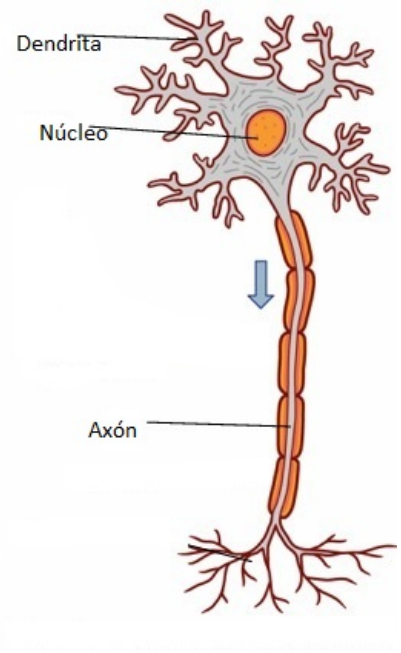

Figura 3.3: *Partes principales en una neurona biológica [3].*

sensoras) o de otras neuronas (en el caso de la *neuronas transmisoras* o *interneuronas*), *el soma* como la parte donde se procesa la información y *el axón* como canal de salida hacia otras neuronas (neuronas *transmisoras*) o hacia el exterior (*neuronas efectoras*)[14].

La manera como se transmite la información entre las neuronas se denomina *sinapsis*. Las neuronas que envían la información a otras neuronas se denominan *presinápticas* y aquellas que la reciben, *postsinápticas*. Dicha transmisión se hace de manera unidireccional y puede ocurrir mediante impulsos eléctricos o reacciones químicas que activan o desactivan la comunicación entre una neurona y otra. Todo esto se conjuga en el sistema nervioso de los seres vivos, haciendo que su funcionamiento se haga de manera *paralela* y de tal modo que compense la deficiencia individual de cada neurona, en cuanto a velocidad de procesamiento se refiere.

Los aspectos mencionados en los párrafos anteriores se centran en el funcionamiento básico de una *red neuronal artificial*. Esta red adecúa las características de la red biológica, mediante un programa algorítmico o mediante la construcción de un circuito electrónico.

3.3. Redes neuronales artificiales

Informalmente, un *sistema neuronal artificial* tiene una estructura análoga al *sistema neuronal biológico* cuyos elementos básicos, llamadas *neuronas artificiales*, se conectan entre sí y se organizan en *capas* para formar la *red neuronal*.

Formalmente, desde el punto de vista del grupo PDP(*Parallel Distributed Processing Research Group*[3], de la Universidad de California en San Diego), una *red neuronal artificial*, también conocida como *sistema neuronal artificial*, la componen los siguientes elementos: *un conjunto de procesadores elementales o neuronas artificiales, un patrón de conectividad o arquitectura, una dinámica de activaciones, una regla o dinámica de aprendizaje y el entorno donde opera* [14].

3.3.1. Modelo general de neurona artificial

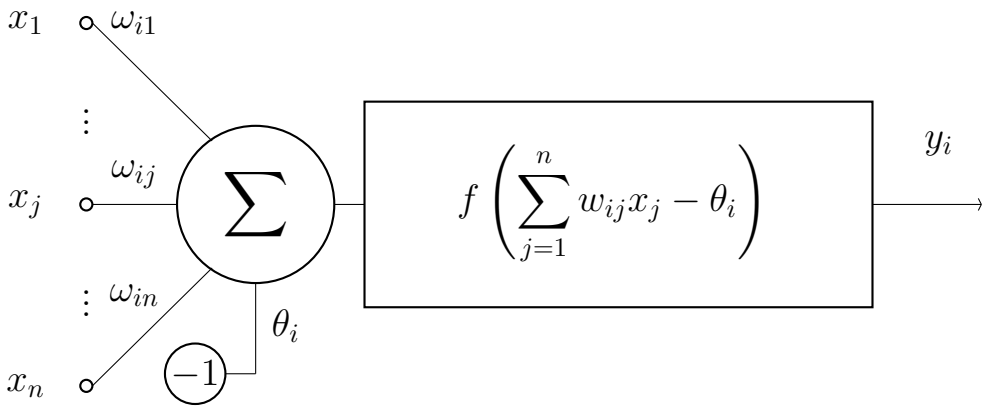

Figura 3.4: *Partes principales de una neurona artificial.*

Al igual que una *neurona biológica*, una *neurona artificial* posee unas entradas que pueden provenir del exterior o de otras neuronas conectadas a ella y proporciona una única salida. En una red *neuronal artificial*, los elementos que constituyen la "neurona i" son los siguientes (Figura 3.4) [14]

[3]Grupo de investigación en RNA, responsables, en gran medida, del renacimiento de las redes neuronales a mediados de los ochenta, cuyo trabajo se publicó en dos volúmenes considerados clásicos [*Rumelhart, MacClelland*] [14].

- **Entradas**: $x_j(t)$. Estas pueden ser *binarias* o *continuas* dependiendo del modelo de red y la aplicación que vaya a considerar.
- **Pesos sinápticos**: w_{ij}. Representan la intensidad de interacción entre cada neurona *presináptica* j y la neurona *postsináptica* i; dichos pesos se pueden representar mediante una matriz $W = (w_{ij}) \in \mathbb{R}^{m \times n}$.
- **Regla de propagación**: $\sigma(w_{ij}, x_j(t))$. Proporciona el valor del *potencial postsináptico* $h_i(t) = \sigma(w_{ij}, x_j(t))$ de "neurona i"en función de sus pesos y entradas. La más común es de tipo lineal y se calcula como la suma ponderada de las entradas con los pesos sinápticos, lo que vectorialmente sería el producto escalar de los vectores de entradas y de pesos

$$h_i(t) = \sum_{j=1}^{n} w_{ij} x_j = \boldsymbol{w}_i^T \cdot \boldsymbol{x}.$$

- **Función de activación** o **de transferencia**: $f_i(a_i(t-1), h_i(t))$. Proporciona el *estado de activación* actual $(a_i(t) = f_i(a_i(t-1), h_i(t))$ de la neurona i, en función de su estado anterior $(a_i(t-1))$ y de su *potencial postsináptico* actual, aunque en muchos modelos se considera que el estado anterior no interviene en el proceso. Las funciones más usadas se muestran en la **Tabla** 3.1.
- **Función de salida**: $F_i(a_i(t))$. Proporciona la *salida* actual $y_i(t) = F_i(a_i(t))$ de la neurona i en función de su *estado de activación*. Por lo regular se usa la *función identidad*, de tal manera que la función de activación de la red se considera como la misma salida.

En general, el modelo que habitualmente se usa es aquel cuya regla de propagación es la suma ponderada de las entradas y sus pesos respectivos, la función de activación proporciona su salida y tiene un parámetro adicional θ_i, conocido como *umbral* o *bias*, que puede tener diferentes usos dependiendo del modelo. Así, dicho modelo se puede expresar por la igualdad

$$y_i(t) = f_i\left(\sum_{j=1}^{n} w_{ij} x_j - \theta_i\right) = f_i\left(\sum_{j=0}^{n} w_{ij} x_j\right),$$

donde en la última parte hemos incluido el parámetro θ_i como si fuera el peso w_{i0}, con la convención de que $x_0 = -1$.

Funciones de activación	
Identidad	$y = x$
Escalón	$y = sign(x)$ $y = H(x)$
Lineal a tramos	$y = \begin{cases} -1 \text{ si } x < -1 \\ x \text{ si } -1 \leq x \leq 1 \\ 1 \text{ si } x > 1 \end{cases}$
Sigmoidea	$y = \frac{1}{1+e^{-x}}$ $y = \text{tgh}(x)$
Gaussiana	$y = Ae^{Bx^2}$
Sinusoidal	$y = A \,\text{sen}\,(\omega x + \varphi)$

Tabla 3.1: *Funciones de activación más usadas.*

3.3.2. Estructura de una red neuronal artificial

Como lo mencionamos anteriormente, la *neurona artificial* por sí sola posee una baja capacidad de procesamiento, su verdadero potencial radica en su alta interconexión con otra neuronas, lo que da lugar a la formación de las *redes neuronales artificiales*.

En una *red neuronal artificial*, las neuronas se conectan por medio de las sinapsis y esta estructura de conexiones determinan el comportamiento de la red [14]. Las conexiones sinápticas son unidireccionales; es decir, la información únicamente puede propagarse en un solo sentido de una neurona (*presináptica*) a otra (*postsináptica*). Las neuronas que reciben la misma información (misma entrada) se suelen agrupar en unidades estructurales denominadas *capas* o *niveles*. El conjunto de una o más capas constituye la red neuronal.

En una *red neuronal artificial* podemos distinguir tres niveles o capas. Una capa de **entrada** formada por las neuronas que reciben la información del medio exterior, una capa de **salida** formada por las neuronas que transfieren la información procesada al exterior y una capa **intermedia** u **oculta** (la cual puede o no existir), en la cual se procesa toda la información sin tener conexión con el entorno donde opera.

Dependiendo del enfoque, se pueden establecer diferentes arquitecturas de las redes neuronales artificiales. De acuerdo al número de capas, hablamos de redes neuronales *monocapa* compuestas por una única capa o redes neuronales *multicapa* compuestas por varias capas. En relación a la manera como fluye la información, tenemos las redes neuronales unidireccionales (*feedforward*), en las cuales la información fluye en un solo sentido y las redes recurrentes o retroalimentadas (*feedback*), en las que la información puede fluir en cualquier sentido, incluido el de entrada-salida.

3.3.3. Aprendizaje

La estructura de una RNA no está completa si no podemos garantizar que ésta funcione correctamente; para ello, al igual que nuestro cerebro funciona mejor en la medida que reciba un buen *aprendizaje* o *entrenamiento*, las redes neuronales artificiales necesitan pasar por un proceso de *aprendizaje*.

El *aprendizaje* de una *red neuronal artificial* es el proceso por el cual se ajustan sus pesos sinápticos, con el fin de adaptar su desempeño en el entorno donde opera. El tipo de aprendizaje es determinado de acuerdo a la manera en que dichos pesos son ajustados.

El ajuste se puede realizar en dos niveles. El más común consiste en modificar los pesos sinápticos siguiendo una cierta regla de aprendizaje, construida por lo regular a partir de la optimización de una *función de error*, que mide la veracidad en la respuesta de la red. El otro nivel de aprendizaje, incluido por algunos modelos, consiste en modificar la arquitectura de la red, añadiendo o destruyendo neuronas; en cualquiera de los dos procesos, la información relevante de los datos de entrada debe quedar incorporada en la estructura de la red.

Para el ajuste del primer nivel, existen básicamente dos tipos de aprendizaje, el *supervisado*, el cual se caracteriza por tener un control externo a través de un *supervisor* o *maestro*, el cual conoce las salidas deseadas correspondientes a las respectivas entradas y el *no supervisado* o *autoorganizado* que consiste en estimar los pesos de la red en función de la caracterización de los datos de entrada de acuerdo a un objetivo específico que

nos permite detectar sus patrones. Adicionalmente, existen el *aprendizaje híbrido* y el *aprendizaje reforzado* [14][1].

Lo que hacen algunos algoritmos de aprendizaje supervisado es minimizar una función error a través de *métodos iterativos*, por lo cual, pueden surgir problemas numéricos de convergencia o de costos computacionales.

También es importante distinguir entre el error cometido por la red en el proceso de entrenamiento y el error que la red ya entrenada comete ante entradas no utilizadas en el proceso de entrenamiento, esto mide la capacidad de *generalización* de la red [14].

3.4. Redes neuronales supervisadas

En esta sección, trataremos algunos de los modelos de redes neuronales más populares; las redes *unidireccionales* que usan aprendizaje *supervisado*, las cuales se conocen como *redes neuronales para representación o ajuste funcional* [14].

Dentro de este grupo de modelos de *redes neuronales artificiales*, están *el perceptrón simple, la adalina y el perceptrón multicapa*. El algoritmo de entrenamiento o aprendizaje del *perceptrón multicapa* se denomina de *retropropagación* de errores o *backpropagation* y hace que este modelo sea el más empleado hasta el momento en muchas aplicaciones prácticas. *El perceptrón simple y la adalina* son de gran interés histórico, pues su evolución representa la historia misma de las redes neuronales artificiales [14]. En general, la importancia de estos modelos se debe a su carácter de dispositivos entrenables, sus algoritmos de aprendizaje son llamados algoritmos de *corrección de errores* debido a que ajustan los pesos en proporción a la diferencia que existe entre la salida obtenida y la salida deseada, con el objetivo de minimizar el "tamaño" de dicha diferencia.

3.4.1. El perceptrón simple

El *perceptrón simple* o *perceptrón* fue introducido por *Rosenblatt* a finales de los años cincuenta y es un modelo unidireccional, compuesto por dos capas de neuronas, una sensorial o de entradas, y la otra efectora o de salida.

Esta *red neuronal artificial*, llamada también función de *ajuste de patrones* (*pattern-*

mapping), aprende a clasificar datos *linealmente separables*, mediante un aprendizaje supervisado y con salidas binarias [14].

La operación de un *perceptrón* con n neuronas de entrada y m de salida, se puede expresar mediante la igualdad

$$y_i = H\left(\sum_{j=1}^{n} w_{ij}x_j - \theta_i\right), \quad 1 \leq i \leq m,$$

donde el término θ_i representa un umbral de disparo o activación y H es la función *escalón* (**Tabla** 3.1).

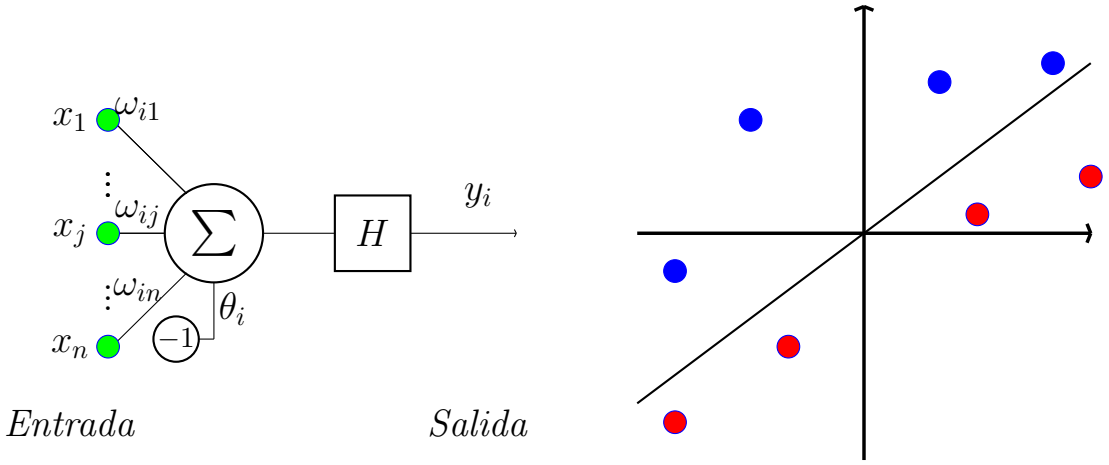

Figura 3.5: *Neurona típica del perceptrón simple y conjunto de datos linealmente separables.*

3.4.2. La adalina

La red *adalina* o *adaline* (***ada****ptive* ***li****near* ***e****lement*) fue introducida por *Widrow* en 1959 y utiliza una neurona similar a la del *perceptrón*, solo que su función de activación es *lineal*. Por otro lado, incorpora una unidad conocida como *bias*, igual al umbral del perceptrón, pero con el fin de no anclarla al origen del sistema de entrada [14].

De esta manera, la salida de la *adalina* es de la forma

$$y_i = \sum_{j=1}^{n}(w_{ij}x_j - \theta_i), \;\; 1 \leq i \leq m.$$

La característica principal del *perceptrón* y de la *adalina* es que permiten clasificar modelos *linealmente separables*; es decir, conjuntos de datos que se pueden separar mediante un hiperplano. Sin embargo, también son incapaces de resolver problemas de clasificación de conjuntos que no son linealmente separables.

3.4.3. El perceptrón multicapa

Debido a las fuertes limitaciones del *perceptrón* y la *adalina*, lo que fue evidenciado por *Minsky* y *Papert* en su libro *Perceptrons* [1][18], la investigación en redes neuronales se detuvo durante casi una década. Por otro lado, *Rosenblatt* ya intuía que si agregaba una capa de neuronas (*capa oculta*) entre las capas de entrada y de salida, podía resolver dichos problemas. No obstante, quedaba el interrogante de ¿cómo debía ser el entrenamiento de esta red, si no se conocía la salida obtenida en la capa oculta? Este interrogante fue resuelto en gran medida por *Paul Werbos* en (1974), cuando propone en su tesis doctoral el algoritmo denominado *retropropagación* de errores (*backpropagation*) [1], lo que dio origen al *perceptrón multicapa* o *MLP* (*Multi-Layer Perceptron*) [14] [1].

El algoritmo de *retropropagación* de errores se desarrolló en un contexto general, para cualquier tipo de red, siendo las redes neuronales una aplicación especial, razón por la cual el algoritmo no fue aceptado dentro de la comunidad de investigadores de redes neuronales. Fue solo hasta mediados de los años *80*, cuando este algoritmo fue redescubierto al mismo tiempo por varios investigadores, *David Rumelhart, Geoffrey Hinton, Ronal Williams, David Parker y Yann Le Cun* y se popularizó cuando fue incluido en el libro *Parallel Distributed Processing Group* (grupo *PDP*) por los psicólogos *David Rumelhart* y *James McClelland*. La publicación de este libro trajo consigo un auge en las investigaciones con redes neuronales, siendo el *perceptrón multicapa* una de las redes más ampliamente empleadas, aún en nuestros días [14][1].

El perceptrón multicapa es una red neuronal unidireccional constituida por, al menos, tres niveles o capas: una *capa de entrada*, otra *capa de salida* y el resto de capas intermedias denominadas *capas ocultas*. La estructura de un *perceptrón multicapa*, con una capa

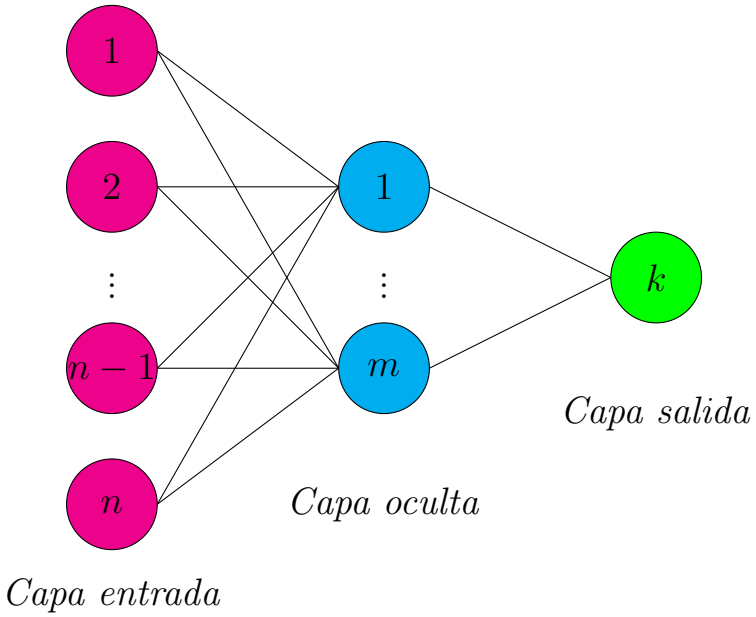

Figura 3.6: *Perceptrón multicapa (MLP).*

oculta, se representa en la Figura 3.6 [4].

Sean x_i, las entradas de la red; y_j, las salidas de la capa oculta; z_k, las salidas de la capa final; w_{ij}, los pesos de la capa oculta y θ_j, sus umbrales; w'_{kj}, los pesos de la capa de salida, y θ'_k, sus umbrales, para todo $i = 1, \ldots, n\,, j = 1, \ldots, q$ y para todo $k = 1, \ldots, m\,$. La operación de un *perceptrón multicapa* con estas características se expresa matemáticamente por la ecuación

$$z_k = \sum_{j=1}^{q} w'_{kj} y_j - \theta'_k = \sum_{j=1}^{q} w'_{kj} f\left(\sum_{i=1}^{n} w_{ji} x_i - \theta_j\right) - \theta'_k, \tag{3.1}$$

donde f es la *función de activación.*

Como ya mencionamos, el aprendizaje de un *perceptrón multicapa* se hace a través de la *minimización de una función error* que mide la diferencia entre la salida $\boldsymbol{z}$ obtenida por la red y la salida deseada $\boldsymbol{t}$.

[4]Existen diversas demostraciones de que este modelo de *perceptrón multicapa* es un *aproximador universal de funciones* [14].

Matemáticamente, la función error es

$$\begin{aligned} E: \ \mathbb{R}^n &\longrightarrow \mathbb{R} \\ \boldsymbol{w} &\longmapsto E(\boldsymbol{w}), \end{aligned}$$

donde para nuestro caso, el vector $\boldsymbol{w}$ tiene como componentes los *pesos sinápticos*. La gráfica de este campo escalar es una hipersuperficie. Así, asociado al *aprendizaje* de un perceptrón multicapa, tenemos el siguiente problema de minimización

$$\begin{aligned} &Minimizar \quad E(\boldsymbol{w}). \\ &\boldsymbol{w} \in \mathbb{R}^n \end{aligned} \tag{3.2}$$

En el caso de una muestra finita formada por los *patrones de entrada*, $\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots, \boldsymbol{x}^p$, vectores de $\mathbb{R}^n$, cada uno de los cuales tiene como componentes las *entradas de la red* y de los vectores en $\mathbb{R}^m$, $\boldsymbol{t}^1, \boldsymbol{t}^2, \ldots, \boldsymbol{t}^p$, que contienen las *salidas deseadas*, la *función error* E es la siguiente

$$E(\boldsymbol{w}) = \frac{1}{2} \sum_{\mu=1}^{p} \sum_{i=1}^{m} (\boldsymbol{t}_i^{\mu} - \boldsymbol{z}_i^{\mu}(w))^2 \equiv \frac{1}{2} \sum_{\mu=1}^{p} \|\boldsymbol{t}^{\mu} - \boldsymbol{z}^{\mu}(w)\|_2^2, \tag{3.3}$$

donde cada vector $\boldsymbol{z}^{\mu}(w) \in \mathbb{R}^m$, con $\mu = 1, \ldots, p$, contiene las respuestas de la red. Así, la función E permite obtener el *error cuadrático medio* de las salidas de la red respecto de las deseadas [14].

En el marco conceptual que estamos describiendo, el algoritmo de *retropropagación* de errores aparece como una con secuencia natural de extender el algoritmo (LMS) a las redes multicapa. Para ello, planteamos una *función de error* y derivamos, no solo respecto a los pesos de la capa de salida, sino también respecto a los pesos de las capas intermedias, haciendo uso de la regla de la cadena, para lo cual, es indispensable que las funciones de transferencia entre las capas sean diferenciables [14].

En el proceso iterativo del algoritmo de entrenamiento de una *red neuronal multicapa*, se lleva a cabo una fase de ejecución para los *patrones de entrenamiento*. Existen dos maneras de hacer esta ejecución, una denominada *aprendizaje por lotes*, que consiste en presentar a la red todos y cada uno de los *patrones de entrenamiento*, calcular para cada patrón, el *error en la salida* y por último, proceder a hacer la *actualización* de los *pesos sinápticos* y la otra llamada *aprendizaje en serie* que consiste en calcular el *error en la salida* y *actualizar* los *pesos sinápticos* tras la presentación de cada *patrón de aprendizaje*, teniendo presente que en cada iteración, el orden en la presentación de los patrones sea *aleatorio* [14].

Para el *perceptrón multicapa* definido anteriormente, si $\boldsymbol{x}^\mu$ para $\mu = 1, \ldots, p$ es un patrón de entrada, la operación de la red (3.1) se expresa como

$$z_k^\mu = g\left(\sum_{j=1}^{q} w'_{kj} y_j^\mu - \theta'_k\right) = g\left(\sum_{j=1}^{q} w'_{kj} f\left(\sum_{i=1}^{n} w_{ji} x_i^\mu - \theta_j\right) - \theta'_k\right), \tag{3.4}$$

donde g es la función de activación de las neuronas de salida y f de las ocultas (ambas funciones pueden ser *sigmoidales* aunque a menudo, la función g es la *identidad*). En este sentido, la función *error cuadrático medio* es

$$E\left(w_{ji}, \theta_j, w'_{kj}, \theta'_k\right) = \frac{1}{2} \sum_{\mu=1}^{p} \sum_{k=1}^{m} \left[t_k^\mu - g\left(\sum_{j=1}^{q} w'_{kj} y_j^\mu - \theta'_k\right)\right]^2. \tag{3.5}$$

El problema (3.2) con E definida por (3.3) es un *problema de mínimos cuadrados no lineales* [5]. Para este problema existen muchos métodos de solución, algunos de los cuales fueron tratados de manera general en el **Capítulo 1** [14].

Capítulo 4

Pruebas numéricas

En la actualidad, son numerosas las aplicaciones de las redes neuronales artificiales y la investigación aún continúa. Profesionales de diferentes áreas, tales como Ingeniería, Matemáticas, Filosofía, Fisiología y Psicología, entre otros, han unido sus esfuerzos debido al potencial que ofrecen estos modelos computacionales para resolver diferentes problemas en sus respectivas áreas de desempeño profesional [26].

En este capítulo, implementamos por primera vez, el método *secante estructurado* para el entrenamiento del perceptrón multicapa. Con el propósito de comparar su desempeño numérico, también implementamos los métodos de *Gauss-Newton* y *Levenverg-Marquardt*, ampliamente utilizados con el mismo propósito, en paquetes (o programas) como el *Toolbox* de redes neuronales de MATLAB®.

Usamos las cuatro fórmulas (2.35) para actualizar en cada iteración la matriz A_k dada por (2.33) con lo cual, tenemos cuatro versiones del algoritmo *secante estructurado*, las cuales llamaremos: método *PSBE*, *DFPE*, *BFGSE* y *SR1E*, respectivamente.

Para las pruebas numéricas, consideramos el entrenamiento de dos redes del tipo *perceptrón multicapa* para resolver sendos problemas: *evaluar la función seno* y *predecir el consumo de energía eléctrica* en una determinada región, en un día dado y una hora determinada en años futuros a los de la toma de la muestra.

Para escribir los códigos de los algoritmos y de las funciones objetivo de cada problema, usamos el software MATLAB®. Realizamos las pruebas numéricas en un computador Intel

(R) Core (TM) i5-CPU de 2.67 GHz. La presentación de los *parámetros de entrenamiento* la hicimos usando la técnica de *entrenamiento por lotes* descrita en el **Capítulo** 3.

4.1. Algoritmo general para el entrenamiento de un perceptrón multicapa

Como lo mencionamos en el **Capítulo** 2, matemáticamente, en el entrenamiento del *perceptrón multicapa* resolvemos un problema de minimización; concretamente

$$\begin{aligned} &Minimizar \quad E(\boldsymbol{w}) \\ &\boldsymbol{w} \in \mathbb{R}^n \end{aligned} \tag{4.1}$$

donde $E(\boldsymbol{w})$ está definida por (1.2).

Teniendo en cuenta que tanto los *pesos* como los *umbrales* son los parámetros a ajustar, la variable independiente de la función objetivo del problema de minimización a resolver, es un vector de la forma

$$\boldsymbol{w} = \begin{pmatrix} \boldsymbol{w}_1 \\ \boldsymbol{b}_1 \\ \vdots \\ \boldsymbol{w}_l \\ \boldsymbol{b}_l \end{pmatrix}$$

donde $\boldsymbol{w}_i$ y $\boldsymbol{b}_i$ son vectores que contienen, respectivamente, los *pesos* y los *umbrales*[1] de la i-ésima capa. El minimizador de (3.2) lo denotaremos $\boldsymbol{w}^*$.

En general, para la obtención de los *pesos iniciales*, se recomienda iniciar con vectores aleatorios [14]; sin embargo, dada la dificultad de que estos sean unos buenos pesos iniciales, los algoritmos están implementados, usando una estrategia de globalización denominada *búsqueda lineal*, que permita iniciar desde cualquier punto [20][5].

Para actualizar los pesos, usamos los métodos globalizados; es decir, en cada iteración, determinamos una *dirección de descenso* $\boldsymbol{d}_k$ y luego aplicamos una estrategia de *búsqueda lineal*, para encontrar un *tamaño de paso* λ con el cual definimos la aproximación siguiente mediante la iteración $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + \lambda \boldsymbol{d}_k$.

[1] En adelante, consideraremos a los *umbrales* como si fueran pesos con entrada constante igual a -1.

Usamos dos criterios de parada: uno relacionado con el tamaño del *gradiente de la función objetivo* ($\boldsymbol{g} = \nabla E(\boldsymbol{w})$) y el otro, relacionado con el *número de iteraciones* (n). Exactamente, declaramos *convergencia* si $\|\boldsymbol{g}\|_2 \leq tol$ y *divergencia* si $n > N$, donde tol es un número real muy pequeño que denota la *tolerancia* usada y N es el número máximo de iteraciones en el algoritmo.

Teniendo en cuenta lo anterior, presentamos a continuación la estructura general del algoritmo para el entrenamiento de un *perceptrón multicapa*.

Algoritmo 4.1. ***Entradas****: Patrones de entrenamiento:* $\boldsymbol{x}^\mu$ *y* $\boldsymbol{t}^\mu$, $\mu = 1, \ldots, p$.

Constantes*:* $\lambda_0 = 1,\ \alpha = 0.0001$ *y* $\alpha = 0.01$.

P.0. Inicialización

Generar los pesos iniciales $\boldsymbol{w}_0$. *Calcular la salida de la red para los* p *patrones de entrenamiento y el error en la salida.*

P.1. Criterios de parada

$\|\nabla E(\boldsymbol{w}_k)\|_2 > Tol$ *y* $k \leq N$

P.2. Búsqueda direccional

Calcule B_k *y encuentre* $\boldsymbol{s}_k$ *tal que* $B_k \boldsymbol{s}_k = -\nabla E(\boldsymbol{w}_k)$.

P.3. Búsqueda lineal

Calcular λ_k *tal que* $E(\boldsymbol{w}_k + \lambda_k \boldsymbol{s}_k) \leq E(\boldsymbol{w}_k) + \alpha \lambda_k \nabla E(\boldsymbol{w}_k)^T \boldsymbol{s}_k$.

P.4. Actualización

Definir $\boldsymbol{w}:\ \boldsymbol{w}_{k+1} = \boldsymbol{w}_k + \lambda_k \boldsymbol{s}_k$.

4.2. Pruebas numéricas

Realizamos dos tipos de pruebas numéricas:

1. Comparar el desempeño numérico de las cuatro versiones del método *secante estructurado*: Métodos *PSBE*, *DFPE*, *BFGSE* y *SR1E*.
2. Comparar el desempeño numérico de los métodos *BFGS*, *Levenverg-Marquard*, *Gauss-Newton* y "el mejor" de los anteriores.

Problema 1: **Evaluación de la función seno**.

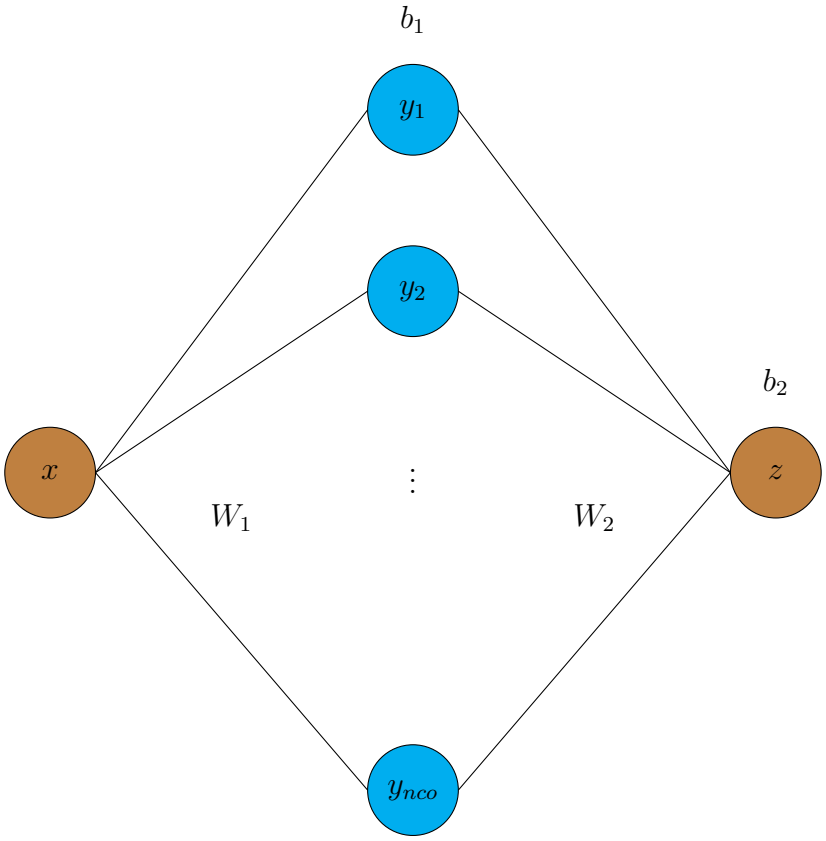

Figura 4.1: *Percetrón multicapa que aproxima la función seno.*

Este problema ilustra el uso del *percetrón multicapa* como *aproximador universal de funciones*. En efecto, para cualquier función de $\mathbb{R}^n$ en $\mathbb{R}^m$ siempre es posible diseñar y

entrenar un perceptrón multicapa, de tal manera que realice un ajuste de los datos de dicha función con un grado de precisión predefinido [1]. En particular, es relativamente sencillo evaluar una función de variable y valor real tal como la *función seno*, mediante una de estas redes.

Este problema lo resolvimos mediante una *percetrón multicapa* con una capa oculta (**Figura** 4.1), donde el número de neuronas en la capa oculta (*nco*) es definido por el usuario. Los patrones de entrenamiento fueron $\boldsymbol{x} = (x_1, x_2, \ldots, x_p)^T$, $p = 41$ y $\boldsymbol{t} = (t_1, t_2, \ldots, t_p)^T$ con $t_i = sen\, x_i$, donde las componentes del vector $\boldsymbol{x}$ (entradas de la red) son números reales distribuidos uniformemente en el intervalo $[0, 2\pi]$. Usamos como función de activación la función *sigmoidal (logsig)* y la *identidad (purelin)* en la *capa oculta* y de *salida*, respectivamente [1].

El vector de pesos iniciales para cada valor de *nco*, $\boldsymbol{w}_0^{nco}$, lo generamos aleatoriamente con la *función* de MATLAB® $randn(\cdot, \cdot)$, exactamente $\boldsymbol{w}_0^{nco} = randn(3nco+1, 1)$. Así,

$$\boldsymbol{w}_0^4 = \begin{pmatrix} 0.538 \\ 1.834 \\ -2.259 \\ 0.862 \\ 0.319 \\ -1.308 \\ -0.433 \\ 0.343 \\ 3.578 \\ 2.769 \\ -1.350 \\ 3.035 \\ 0.725 \end{pmatrix} \quad \boldsymbol{w}_0^5 = \begin{pmatrix} 0.538 \\ 1.834 \\ -2.259 \\ 0.862 \\ 0.319 \\ -1.308 \\ -0.433 \\ 0.343 \\ 3.578 \\ 2.769 \\ -1.350 \\ 3.035 \\ 0.725 \\ -0.063 \\ 0.715 \\ -0.205 \end{pmatrix} \quad \boldsymbol{w}_0^6 = \begin{pmatrix} -0.082 \\ -1.933 \\ -0.439 \\ -1.795 \\ 0.8405 \\ -0.889 \\ 0.100 \\ -0.544 \\ 0.303 \\ -0.600 \\ 0.490 \\ 0.739 \\ 1.719 \\ -0.194 \\ -2.138 \\ -0.839 \\ 1.354 \\ -1.072 \\ 0.961 \end{pmatrix}$$

Presentamos los resultados de estas pruebas para el **Problema 1** en dos tablas cuyas dos primeras columnas contienen la información sobre el *número de neuronas en la capa oculta* (*nco*) y la *tolerancia usada* (*Tol*). Las cuatro columnas siguientes contienen, para

cada método, el tiempo de ejecución (t), medido en *segundos*, y el *número de iteraciones* (n). El símbolo "-" indica que hubo *divergencia* del método considerado (se excedió el número máximo de iteraciones permitido ($N = 500$)).

	$N = 500$	***PSBE***		***DFPE***		***BFGSE***		***SR1E***	
nco	Tol	t	n	t	n	t	n	t	n
4	10^{-3}	1	152	–	–	0.5	69	–	–
4	10^{-5}	–	–	–	–	1	134	–	–
4	10^{-6}	–	–	–	–	2	319	–	–
5	10^{-3}	–	–	–	–	1	116	–	–
5	10^{-5}	–	–	–	–	1	153	–	–
5	10^{-6}	–	–	–	–	–	–	–	–
6	10^{-3}	–	–	–	–	1	78	–	–
6	10^{-5}	–	–	–	–	–	–	–	–

Tabla 4.1: *Resultados de los métodos secantes estructurados para la evaluación de la función seno.*

	$N = 500$	***GN***		***LM***		***BFGSE***	
nco	Tol	t	n	t	n	t	n
4	10^{-3}	-	-	1	160	0.5	69
4	10^{-5}	-	-	-	-	1	134
4	10^{-6}	-	-	-	-	2	319
5	10^{-3}	-	-	-	-	1	116
5	10^{-5}	-	-	-	-	1	153
5	10^{-6}	-	-	-	-	-	-
6	10^{-3}	-	-	-	-	1	78
6	10^{-5}	-	-	-	-	-	-

Tabla 4.2: *Resultados de los métodos de Gauss-Newton (GN), Levenberg-Marquardt (LM) y secante estructurado (BFGSE) para la evaluación de la función seno.*

En la **Tabla** 4.1, podemos observar que en general, el método *BFGSE* convergió en el menor tiempo y número de iteraciones, mientras que los métodos *DFPE* y *SR1E* no convergieron en ningún de los casos.

De la **Tabla** 4.2, observamos que para $Tol \leq 10^{-5}$ y $nco = 4$ y 5, el método *BFGSE* siempre converge y lo hace con un mejor desempeño numérico que los otros métodos

comparados aquí. Además, el método *Levenberg-Marquardt* presentó mejor desempeño numérico que *Gauss-Newton*.

Cabe mencionar que, la *divergencia* en los métodos *secantes estructurados PSBE*, *DFPE*, *Levenberg-Marquardt* y *Gauss-Newton*, ilustrada en las **Tablas** 4.1 y 4.2, se debe a que se alcanzó el número máximo de iteraciones permitido, N. Sin embargo, las **Tablas** 4.3 y 4.4 ilustran la sensibilidad de los métodos, en cuanto a convergencia se refiere, a la escogencia del número máximo de iteraciones.

	$N = 25000$	***PSBE***		***DFPE***		***BFGSE***	
nco	*Tol*	t	n	t	n	t	n
4	10^{-3}	1	152	61	1000	1	69
4	10^{-7}	25	4103	–	–	3	329
5	10^{-3}	36	4225	45	5374	1	116
5	10^{-5}	70	5298	273	+25000	1	153
6	10^{-3}	43	5017	43	5005	1	68
6	10^{-7}	87	1090	–	–	40	4709

Tabla 4.3: *Otros resultados de los métodos secantes estructurados para* $N = 25000$.

	$N = 25000$	***GN***		***LM***		***BFGSE***	
nco	Tol	t	n	t	n	t	n
4	10^{-7}	-	-	23	1395	3	329
4	10^{-3}	-	-	1	160	1	69
5	10^{-5}	-	-	-	-	1	116
5	10^{-3}	-	-	146	665	1	153
6	10^{-7}	52	+25000	-	-	1	68
6	10^{-3}	172	4230	-	-	40	4709

Tabla 4.4: *Otros resultados de los métodos de Gauss-Newton (GN), Levenberg-Marquardt (LM) y secante estructurado (BFGS) para* $N = 25000$.

Por otra parte, la divergencia del método *SR1E* se presentó por el mal condicionamiento de las matrices de actualización.

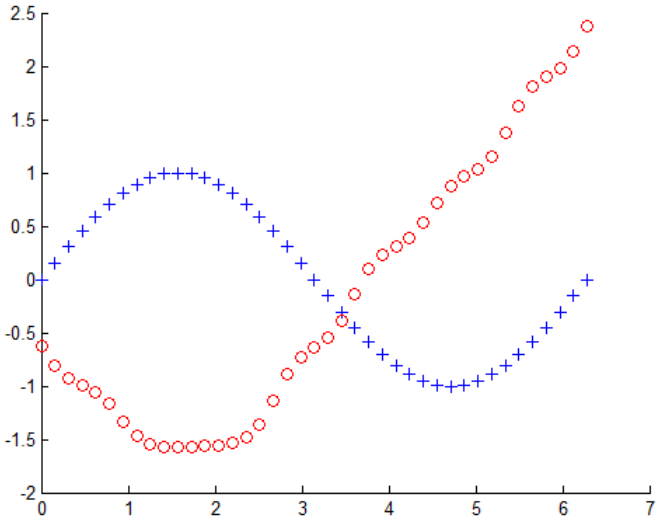

Figura 4.2: *Aproximación inicial de la función seno.*

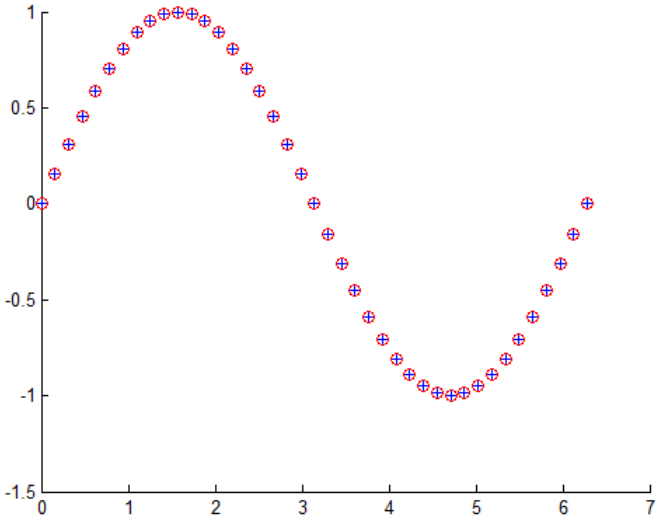

Figura 4.3: *Aproximación final de la función seno.*

Es importante mencionar que el mínimo obtenido en todos los casos fue absoluto, ya que, en este caso, el tamaño del residuo fue del orden de 10^{-3}, lo que podemos observar en las **Figuras** 4.2 y 4.3, en las cuales aparecen, la gráficas de la función seno (cruces

de color azul) y de su evaluación (círculos de color rojo) dada por el algoritmo, en la primera y la última iteración, respectivamente.

Problema 2: **Predicción de consumo eléctrico** [14][25].

Una empresa abastecedora de energía eléctrica, en una población, debe garantizar que el servicio siempre llegue con buena calidad y, de ser posible, a un precio justo. Para que esto ocurra, el servicio debe entregar energía a todos los puntos que lo requieran, mantener los limites de la frecuencia y la tensión con valores dentro de un rango tolerable y operar con costos mínimos, tanto económicos como ambientales. Por tal motivo, es indispensable una planeación exhaustiva del sistema que nos permita, no solo conocer su estado actual en cualquier momento, sino también estados futuros, con el fin de no producir en exceso, ya que habría desperdicio del servicio y daños en el medio ambiente, ni producir tan poco, que no sea suficiente para cubrir las necesidades del servicio.

Una de las partes indispensables en esta planeación es la predicción del consumo de carga eléctrica. El interés de esta predicción radica en la necesidad de que las empresas productoras o vendedoras de energía de la región conozcan con antelación las necesidades de su mercado para poder planear la distribución futura de la energía eléctrica, con el fin de optimizar tanto la producción como su abastecimiento. Por tal motivo, este problema consiste en predecir la demanda de consumo eléctrico en una región para una hora y un día cualquiera, en años futuros.

Hasta hace poco, los métodos usados por estas empresas eran estadísticos [25], sin embargo, este problema fue resuelto en una investigación en la **Universidad Tecnológica de Pereira**, en el año 2000 [25], mediante una *red neuronal* de 4 capas (Figura 4.4): la *capa de entrada*, con dos neuronas que corresponden al *día* y la *hora*; la *capa de salida*, con una neurona que corresponde al *consumo eléctrico* en kilovatios (kw); la *primera capa oculta*, con doce neuronas, y la *segunda capa oculta*, con ocho neuronas. Como funciones de activación usaron la *tangente sigmoidal (tansig)*, en ambas capas ocultas, y *la identidad (purelin)*, en la capa de salida, y como algoritmo de entrenamiento, el método de *Levenberg-Marquardt* que aparece en el **Toolbox** de MATLAB®.

Para generar vectores iniciales, procedimos de la siguiente forma. Inicialmente, generamos aleatoriamente dos vectores $\boldsymbol{\omega}_0 = randn(149, 1)$ y $\boldsymbol{\omega}_1 = randn(149, 1)$. Luego, generamos otros vectores iniciales de la forma $\alpha\boldsymbol{\omega}_0$ y $\mu\boldsymbol{\omega}_1+\boldsymbol{\omega}_0$, para $\alpha = 1,\ 1.2,\ 2,\ 2.5,\ 10,\ -10,\ 10^2$

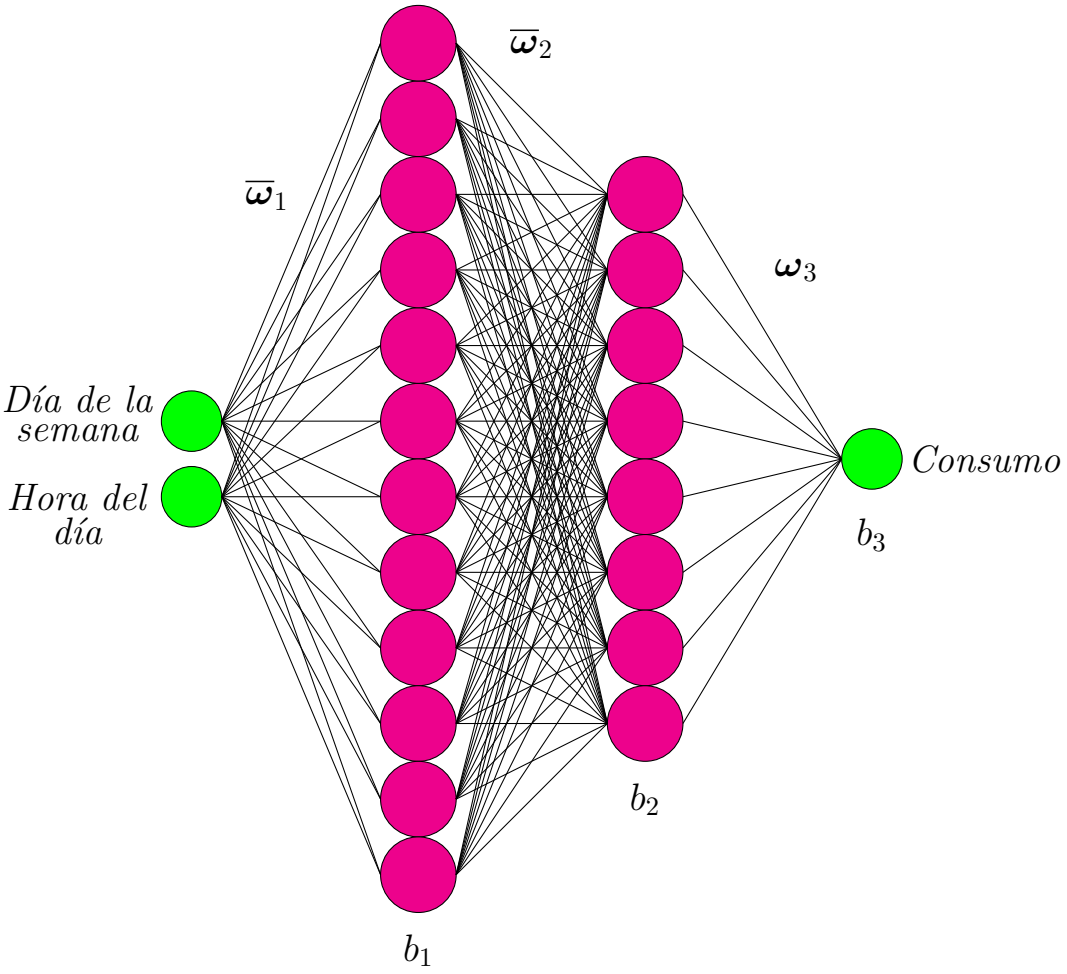

Figura 4.4: *Perceptrón multicapa, para el consumo eléctrico.*

y $\mu = 1,\ 10,\ 10^2$, respectivamente. El vector aleatorio $\boldsymbol{\omega}_0$ utilizado fue

$$\boldsymbol{\omega}_0 \quad = \quad \begin{pmatrix} \boldsymbol{\omega}_1 \\ \boldsymbol{b}_1 \\ \boldsymbol{\omega}_2 \\ \boldsymbol{b}_2 \\ \boldsymbol{\omega}_3 \\ \boldsymbol{b}_3 \end{pmatrix},$$

para $\boldsymbol{b}_3 = 2.436$ y

$$\overline{\boldsymbol{\omega}}_1 = \begin{pmatrix} 566666110.129 & 27979231.394 \\ 69469642.435 & -66918981.288 \\ -32640.109 & -4407825.231 \\ 6.936 & -19.096 \\ -0.004 & -0.018 \\ -0.252 & -0.275 \\ -3096794820.345 & 1751633.739 \\ 18808535.740 & -111934251.209 \\ -139321.291 & -145561.191 \\ 1.657 & -495.152 \\ 1.611 & -0.845 \\ -0.777 & 1.404 \end{pmatrix}$$

$$\overline{\boldsymbol{\omega}}_2 = \begin{pmatrix} -1.563 & 1.260 & -20.999 & 0.615 & -0.853 & -0.4612 & 0.925 & -1.349 \\ 0.111 & 0.608 & 0.426 & 0.832 & 0.786 & -0.269 & 24.727 & -1.310 \\ 1.397 & -0.822 & 0.552 & -0.599 & -0.903 & 0.345 & 1.702 & 0.220 \\ 2.096 & -0.142 & 19.961 & 0.654 & -0.771 & -0.808 & -2.357 & -0.319 \\ 2.560 & -0.562 & -0.361 & 1.637 & 1.214 & 0.836 & -23.666 & 0.257 \\ 1.267 & 0.784 & 1.410 & 0.985 & -1.231 & 0.566 & -0.163 & 1.040 \\ 0.948 & 0.111 & -0.836 & -1.585 & 21.288 & 0.212 & -23.633 & -0.340 \\ -1.387 & -0.969 & -0.473 & -1.895 & 0.105 & -0.985 & 0.371 & -1.319 \\ 26.197 & 0.659 & 23.105 & 0.387 & 2.659 & -0.251 & -0.513 & -0.163 \\ -0.523 & -0.580 & 0.174 & 1.834 & -21.086 & 1.453 & -3.314 & -1.040 \\ -2,269 & 0,503 & 1,685 & -1,378 & 0,318 & -1,066 & 1,083 & 0,052 \\ -20.404 & 0.013 & 10.334 & -0.719 & 0.005 & -1.077 & 0.206 & -0.890 \end{pmatrix}$$

$$\boldsymbol{w}_3 = \begin{pmatrix} 0.723 \\ -0.284 \\ 0.178 \\ 0.886 \\ 0.902 \\ -0.588 \\ -0.513 \\ -0.927 \end{pmatrix} \qquad \boldsymbol{b}_1 = \begin{pmatrix} -2123530102.903 \\ 809138.183 \\ 38981761.231 \\ -190610280.076 \\ -643175.476 \\ -1203339.139 \\ -1.266 \\ -407.789 \\ -4.144 \\ 5.733 \\ 3.931 \\ -3.363 \end{pmatrix} \qquad \boldsymbol{b}_2 = \begin{pmatrix} -1.938 \\ 2.050 \\ -22.34 \\ -0.664 \\ -0.1761 \\ -0.6515 \\ 1.418 \\ 2.421 \end{pmatrix},$$

donde $\boldsymbol{w}_1$ y $\boldsymbol{w}_2$ son vectores que se obtienen utilizando el comando de MATLAB® $\boldsymbol{w}_1 = reshape(\overline{\boldsymbol{\omega}}_1, 24, 1)$ y $\boldsymbol{w}_2 = reshape(\overline{\boldsymbol{\omega}}_2, 96, 1)$, dando como resultado dos vectores de 24 y 96 componentes, respectivamente.

En las pruebas numéricas para el **Problema 2**, usamos 112 datos de entrenamiento extraídos de los datos usados en [25], los cuales, para mayor claridad en la lectura de este documento, los presentamos en la **Tabla** 4.5, cuyas columnas contienen los promedios históricos de consumo eléctrico, en una *población de muestra* a la que se le hizo un seguimiento, hora a hora, durante una semana.

	0	1	2	3	4	5	6	7
Lun	1.2300	1.0889	1.0289	0.9879	0.9879	1.1050	1.3729	1.6649
Mar	1.6049	1.4389	1.3631	1.3559	1.3439	1.3890	1.5699	1.7750
Mie	1.6630	1.4689	1.3890	1.3751	1.3611	1.4140	1.6040	1.8009
Jue	1.7299	1.5159	1.4320	1.3931	1.3909	1.4310	1.6140	1.8170
Vie	1.7129	1.4569	1.3461	1.2880	1.2331	1.1911	1570	1.1700
Sab	1.7130	1.2821	1.1820	1.1220	1.0961	1.1059	1.1710	1.2751
Dom	1.4140	1.3250	1.2249	1.2239	1.1240	1.0191	0.9989	0.9989
	8	9	10	11	12	13	14	15
Lun	1.6649	2.1569	2.3230	2.3659	2.3731	2.2311	2.1560	2.2080
Mar	2.0180	2.1900	2.3359	2.3630	2.3359	2.1560	2.0799	0.1651
Mie	2.0739	2.2301	2.3649	2.3990	2.3580	2.2000	2.1231	2.1749
Jue	2.0989	2.2260	2.3810	2.3741	2.3021	2.1459	2.0581	2.0809
Vie	1.2139	1.3370	1.4799	1.5740	1.5951	1.5771	1.5629	1.5320

Sab	1.4121	1.5450	1.7110	1.7410	1.7129	1.6200	1.1570	1.5831
Dom	0.9790	1.0150	1.1271	1.271	1.2950	1.3130	1.2909	1.2600
	16	17	18	19	20	21	22	23
Lun	2.2949	2.3741	2.5000	2.4340	2.3560	2.0000	1.9890	1.8080
Mar	2.2551	2.3671	2.4770	2.4310	2.3540	2.2100	1.7085	1.7000
Mie	2.2049	2.3349	2.4640	2.3780	2.4140	2.0040	1.8582	1.7071
Jue	2.1651	2.2380	2.2820	2.1540	2.1020	1.9950	1.1.9040	1.8590
Vie	1.5440	1.6380	1.7310	1.7480	1.7921	1.8321	1.8620	1.7930
Sab	1.6251	1.6251	1.8950	1.9040	1.9310	1.9360	1.9580	1.7480
Dom	1.2669	1.3631	1.1530	1.6020	1.6440	1.6150	1.5030	1.4990

Tabla 4.5: *Datos del consumo eléctrico en una semana*

Los resultados de las pruebas numéricas para el **Problema 2**, los presentamos en las **Tablas** 4.6 y 4.7. La primera tabla contiene información de los algoritmos secantes estructurados obtenida a partir de puntos iniciales de la forma $\alpha \boldsymbol{w}_0$ y $\mu \boldsymbol{w}_1 + \boldsymbol{w}_0$. Exactamente, la primer columna contiene los valores de α y de μ utilizados. Las cuatro columnas siguientes contienen, para cada *método secante estructurado*, la *tolerancia usada* (Tol), el *tiempo de ejecución* (t), medido en segundos y el *número de iteraciones* (n). La segunda tabla, contiene los resultados de la comparación de los métodos de *Gauss-Newton*, *Levenberg-Marquardt* y *secante estructurado* (*BFGSE*), a partir de estos puntos iniciales. El símbolo "-" indica que hubo *divergencia* del método considerado (se excedió el número máximo de iteraciones permitido, ($N = 200$)).

	$N = 200$	***PSBE***		***DFPE***		***BFGSE***		***SR1E***	
α	Tol	t	n	t	n	t	n	t	n
1.0	10^{-5}	3.5	6	6.9	6	10.3	6	13.7	6
1.2	10^{-2}	2.6	4	5.1	4	7.6	4	10.1	4
1.2	10^{-6}	4.4	8	8.8	8	13.1	8	17.4	8
10	10^{-2}	3.5	6	6.9	6	10.3	6	13.8	6
10	10^{-6}	5.3	10	10.5	10	15.8	10	20.2	8
-10	10^{-2}	3.5	6	6.9	6	10.4	6	13.8	6
-10	10^{-6}	6	10	6	10	6	10	6	8
100	10^{-2}	4.5	8	8.9	8	13.2	8	17.5	8
100	10^{-6}	5.4	10	10.7	10	16.0	10	20.4	8
μ	Tol	t	n	t	n	t	n	t	n

1.0	10^{-2}	2.6	6	6.2	4	9.6	6	12.2	4
1.0	10^{-6}	5.3	10	9.7	8	14.0	8	18.4	8
10	10^{-2}	3.5	6	6.1	4	8.7	4	11.1	4
10	10^{-6}	4.4	8	9.8	10	15.0	10	19.4	8
10	10^{-14}	5	3	8	7	7	5	6	6
10^2	10^{-3}	165	223	6	3	8	7	8	7

Tabla 4.6: *Resultados de los métodos secantes estructurados para el problema del consumo eléctrico*

En la **Tabla** 4.6, podemos observar que, en general, los 4 métodos *secantes estructurados* tienen un buen desempeño numérico (similar en todos los casos), en cuanto a *tiempo de ejecución* y *número de iteraciones*, excepto, cuando $\mu = 10^2$, y $Tol = 10^{-3}$, caso en el cual, el método *PSBE* empleó más tiempo y convergió en un número mayor de iteraciones que los otros métodos.

		GN		***LM***		***BFGS***		***BFGSE***	
α	Tol	t	n	t	n	t	n	t	n
1.2	10^{-2}	-	-	64	112	15.1	4	7.6	4
1.2	10^{-6}	-	-	-	-	29.6	20	13.1	8
10	10^{-2}	-	-	-	-	27.0	24	10.3	6
10	10^{-6}	-	-	-	-	35.4	28	15.8	10
10^2	10^{-2}	-	-	-	-	31.7	26	13.2	8
10^2	10^{-6}	-	-	-	-	36.6	30	16.0	10
μ	Tol	t	n	t	n	t	n	t	n
1.0	10^{-2}	69	118	-	-	28.4	26	12.2	4
1.0	10^{-6}	92	164	-	-	30.1	24	18.4	8
10	10^{-2}	-	-	-	-	21.4	18	11.1	4
10	10^{-6}	-	-	-	-	35.7	30	19.4	8

Tabla 4.7: *Resultados de los métodos Gauss-Newton (GN), Levenberg-Marquardt (LM) y secante estructurado (BFGS) para el problema del consumo eléctrico*

Con relación a la **Tabla** 4.7, podemos observar, como esperábamos, el buen desempeño numérico del método *BFGSE* en comparación con los métodos *BFGS*, *Gauss-Newton* y *Levenberg-Marquardt*, en todos los aspectos comparados. Ahora, si solo comparamos los métodos *Gauss-Newton* y *Levenberg-Marquardt*, vemos que en casi todos los casos hay divergencia. Esta divergencia está condicionada al *número máximo de iteraciones* permitido en cada algoritmo (N), tal como sucedió para el **Problema 1**.

Por otra parte, podemos observar que los resultados obtenidos están acordes con la teoría sobre los métodos estudiados, la cual garantiza que para problemas de gran tamaño, la convergencia de los *métodos secantes estructurados*, en especial el método *BFGSE*, en general es mejor que la de los otros métodos con los cuales hicimos las comparaciones [5].

Capítulo 5

Comentarios finales

El estudio de *redes neuronales artificiales* constituye en la actualidad un amplio y activo campo en el que pueden interactuar investigadores de muchas y diferentes áreas, para resolver problemas prácticos y útiles tales como, *control de procesos industriales, reconocimiento de vehículos en los peajes de las autopistas, previsión de consumo eléctrico*, entre otros. En este contexto es quizá, el *perceptrón multicapa*, con su algoritmo de entrenamiento de *retropropagación de errores*, el modelo neuronal más utilizado.

Métodos numéricos tradicionalmente usados en el *entrenamiento supervisado* del *perceptrón multicapa* como por ejemplo, *descenso por gradiente*, *Newton*, *Gauss-Newton* y *Levenverg-Marquardt* requieren del cálculo de la *matriz hessiana* de la *función error*, es decir, requieren información de segundo orden, lo que representa, a pesar de las buenas propiedades de los métodos, una desventaja de ellos, ya que los hace inadecuados para problemas con un elevado número de *neuronas*. En este caso, el cálculo analítico del *hessiano* es muy difícil o muy *costoso*, computacionalmente, dado el gran número de operaciones involucradas en el proceso. Una alternativa la representa el *método secante estructurado*, el cual no requiere explícitamente el cálculo directo de la *matriz hessiana* de la función a minimizar y además, aprovecha la estructura del problema, sin contar con las buenas propiedades de convergencia que posee.

Motivados por las buenas características del *método secante estructurado*, en este trabajo, lo proponemos e implementamos, por primera vez, para el *entrenamiento* del *perceptrón multicapa*, y analizamos numéricamente su desempeño comparándolo con los métodos: *Gauss-Newton* y *Levenverg-Marquardt*.

Resultados de pruebas numéricas preliminares indican un buen comportamiento numérico del método propuesto, pero creemos que es necesario realizar más experimentación numérica con diversos problemas de aplicación e introducir *métodos de búsqueda global*, con lo cual se abre la puerta a nuevas investigaciones.

Bibliografía

[1] Caicedo, E. F. y López, J. A. *Una aproximación Práctica a las Redes Neuronales Artificiales.* Programa Editorial Universidad del Valle. Primera edición. 2009.

[2] Cruces, S. A. *El Método de Mínimos Cuadrados.* http://redesneuronales.wikispaces.com/space/showlogo/1275365407 /logo.jpg. 2010.

[3] Classe Qsi-www.enciclopediasalud.com-B.Barceló. Neurona biológica

[4] De la Fuente O. J. L. *Métodos Matemáticos de Especialidad Ingeniería Eléctrica.* Universidad Politécnica de Madrid. 2010.

[5] Dennis, J. E. and Schnabel, R. B. *Numerical methods for unconstrained optimization and nonlinear equations.* Prentice-Hall, New Jersey. 1983.

[6] Dennis, J. E. and Walker, H. F. *Covergence theorems for least change secant update methods.* SIAM Journal Numerical Analisys 18, 949-987, 19, 443.

[7] Dennis, J. E. Gay, D. M., and Welsch, R. E. *Algorithm 573 NL2SOL-An adaptative nonlinear least squares .* TOMS 7, 369-383.

[8] Fletcher, R. *Practical Methods of Optimization.* Wiley. Second Edition. 2000.

[9] Gavin, P. H. *The Levenberg-Marquardt method for nonlinear least squares curve-fitting problems.* Duke University. 2013.

[10] Imagen tomada de:http://redesneuronales.wikispaces.com/space/showlogo/1275365407/logo.jpg.

[11] Imagen tomada de:http://poderpersonalmexico.com/wp/content/uploads/2012/07/RedNeuronal.jpg.

[12] Kandel, E. R., Schwartz, T. H. and Jessel, T. M. *Principles of Neural Science.* McGraw-Hill. 4ª Edición. 1999.

[13] Kay, S. M. *Fundamentals of Statistical Signal Processing: Estimation Theory*, Prentice Hall, 1993. ISBN: 0-13-345711-7. Capítulos 4,6 y 8.

[14] Martín del Brío, B. y Sanz, A. *Redes Neuronales y Sistemas Borrosos.* Alfaomega. 2007.

[15] Martín, H., Demuth, H. B. and Beale, M. *Neural Network Design.* PWS Published Company. 1995.

[16] Martínez R., H.J., Engels J. *Local and superlinear convergence for partially known quasi-Newton methods.* Siam Journal On Optimization ISSN: 1052-6234 ed:v.1 fasc. p.42 - 56, 1991.

[17] Martínez R., H.J., Dennis J., Tapia R. *Convergence theory for the structured BFGS secant method with an application to nonlinear least squares.* Journal Of Optimization Theory And Applications. Vol.61 p.161 - 178, 1989.

[18] Minsky, M. and Papert, S. *Perceptrons.* MIT Press, 1969.

[19] Moré, J. J., Garbow, B. S., and Hillstrom, K. E. *User guide for MINPACK-1.* Argonne National Labs Report ANL-80-74. 1980.

[20] Nocedal, J. and Wright, S. J. *Numerical Optimization.* Springer. Second Edition. 2006.

[21] Pérez, R. y Díaz, T. *Minimización sin Restricciones.* Editorial Universidad del Cauca. 2010.

[22] Puig M., M. D. *Implementación del algoritmo de mínimos cuadrados aplicado al diseño de dispositivos de microondas.* Universidad Politécnica de Valencia. 2010.

[23] Ramos, A. *Optimización no lineal*. Universidad Pontificia Comillas. Madrid.

[24] Saavedra P. B. *Introducción a la optimización no lineal*. Universidad Autónoma Metropolitana de México. 2012.

[25] http://ohm.utp.edu.co/neuronales.

[26] Villanueva E., M. R. *Las Redes Neuronales Artificiales y su Importancia como Herramienta en la Toma de Decisiones*. Lima, 2002. Universidad Nacional Mayor de San Marcos. Red Peruana Tesis Digitales.