FAUSTO MIGUEL CASTRO CAICEDO

DISEÑO DE UN ALGORITMO DE APRENDIZAJE PARA REDES NEURONALES MLP BASADO EN LAS PROPIEDADES DEL ALGORITMO ACELERADOR REGRESIVO VERSIÓN γ

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Maestría en Electrónica y Telecomunicaciones

Popayán
2014

FAUSTO MIGUEL CASTRO CAICEDO

DISEÑO DE UN ALGORITMO DE APRENDIZAJE PARA REDES NEURONALES MLP BASADO EN LAS PROPIEDADES DEL ALGORITMO ACELERADOR REGRESIVO VERSIÓN γ

Tesis presentada a la Facultad de Ingeniería
Electrónica y Telecomunicaciones de la
Universidad del Cauca para la obtención del
Título de

Magister en
Electrónica y Telecomunicaciones

Director:
Ph.D. Pablo Emilio Jojoa Gómez

Popayán
2014

A mis Padres Rodrigo y Rosalina,
y a mi hermano Marcos Camilo
quienes incondicionalmente,
han inspirado y motivado
con sinapsis de amor
mi deseo de pertenecer a la
maravillosa red de los que hacen
investigación.

Agradecimientos

Al Ph.D. Pablo Emilio Jojoa Gómez, docente investigador de la Universidad del Cauca, quien con sus valiosos aportes y experiencia en el campo del filtrado adaptativo ha abierto espacios para hacer investigación en el apasionante mundo de la neurocomputación. Personalmente agradezco la confianza depositada, reflejada en su constante acompañamiento y paciencia como director de este trabajo de investigación.

Al Mg Jaime Orlando Ruiz Pasos, docente anexado al departamento de electrónica de la Universidad de Nariño, quien con sus orientaciones y asesorías me involucró tempranamente en este campo de investigación.

Al Esp. Rigo Nelson Benabides Cerón, gran amigo y colega con quien tuve la oportunidad de compartir varias horas de discusión en la materia.

Al Ing. Gustavo Adolfo Gómez, por su compañerismo y el notable interés por que este trabajo se logre con los mejores éxitos; agradezco sus recomendaciones.

Extiendo mis agradecimientos a la Facultad de Ingeniería Electrónica y Telecomunicaciones de la Universidad del Cauca; a mis compañeros y profesores del Departamento de Electrónica, Instrumentación y Control, por propiciar un ambiente favorable para el desarrollo de este trabajo.

Resumen

Se presenta un nuevo algoritmo para el entrenamiento de redes neuronales perceptrón multicapa denominado ARγ-GLE (Acelerador Regresivo versión γ con Gradiente Local de Error). Este algoritmo se basa en los principios que rigen la actualización de parámetros en el algoritmo ARγ, que fue desarrollado en el contexto del filtrado adaptativo a partir de la discretización de un algoritmo en tiempo continuo que ajusta la segunda derivada de parámetros. El algoritmo ARγ-GLE se valida mediante diferentes problemas relacionados con aproximación de funciones y reconocimiento de patrones. Los resultados muestran buen comportamiento en cuanto a convergencia y generalización superando en las pruebas realizadas (sin aumento considerable en la complejidad algorítmica) los inherentes problemas del algoritmo "backpropagation" relacionados con la influencia imprevisible de la tasa de aprendizaje.

Palabras Clave: Redes Neuronales, MLP, Algoritmo ARγ, Gradiente Local de Error, Reconocimiento de Patrones, Aproximación de Funciones.

Abstract

A new algorithm is presented for Multi-Layer Perceptron Neural Networks training, which is called ARγ-GLE (*Acelerador Regresivo versión γ con Gradiente Local de Error*). This algorithm is based in the same principles that let parameter actualization in ARγ algorithm. This last one is an algorithm created in adaptive filtering context and is obtained from discretization of a continuous time algorithm based on the second derivative adjustment of the parameter estimate. ARγ-GLE algorithm is validated through different problems related to pattern recognition and fitting function. Results show both good convergence as generalization, overcomming in the experiments the inherent disadvantages of backpropagation algorithm (without a significant increase in the algorithm complexity) related to the unforseeable influence of the learnig rate.

Key words: Artificial Neural Networks, Multi-Layer Perceptron, Algorithm ARγ, Local Gradient, Pattern Recognition, Fitting Function.

Contenido

Lista de Figuras

Pág.

Lista de Tablas

Lista de Símbolos

x	Letra minúscula en negrita denota vector
α	Parámetro de ajuste fijo de los algoritmos ARγ y ARγ-GLE
γ	Parámetro de ajuste fijo de los algoritmos ARγ y ARγ-GLE
m_1	Parámetro de ajuste fijo de los algoritmos ARγ y ARγ-GLE
$\hat{\alpha}$	Tasa de aprendizaje para los algoritmos LMS y BP
$\hat{\mu}$	Constante que ajusta el termino momento en el algoritmo BP
$x_j[n]$	Entrada asociada a la conexión j de una neurona estándar
$w_{j,i}[n]$	Peso sináptico asociado a la conexión j de una neurona i
$b_i[n]$	Bias o umbral de una neurona i
$a_i[n]$	Potencial postsináptico de una neurona i
$y_i[n]$	Salida de una neurona i
$t_i[n]$	Salida deseada de una neurona i
$e_i[n]$	Error de una neurona i
$Hs(.)$	Función escalón "heaviside"
$Tanh(.)$	Función tangente hiperbólica
$'$	Indica pertenencia a la capa de salida de un MLP ($b'_k[n]$, $a'_k[n]$)
$\Re$	Conjunto de los números reales
NaN	indeterminación
$\|\|.\|\|$	Magnitud de un vector
$.^T$	Transpuesta de un vector

Lista de Abreviaturas

AAC: *Algoritmo Acelerador Completo*

ANS: *Artificial Neural Systems*

APCM: *Algoritmo Acelerador Progresivo*

ARCM: *Algoritmo Acelerador Regresivo*

ARγ: *Algoritmo Acelerador Regresivo Versión* γ

ARγ-GLE: *Algoritmo Acelerador Regresivo Versión* γ *con Gradiente Local de Error*

BP: *Back-Propagation*

FIR: *Finite Impulse Response*

IIR: *Infinite Impulse Response*

LMS: *Least Mean Squared*

MLP: *Multi-Layer Percéptron*

MSE: *Mean Squared Error*

NLMS: *Normalized Least Mean Squared*

PDP: *Parallel Dristributed Processing Group*

RLS: *Request Least Square*

Capítulo 1
Introducción

Las redes neuronales artificiales o ANS (*Artificial Neural Systems*) surgen con el propósito de solucionar problemas en los cuales la información se presenta de forma masiva, imprecisa y distorsionada, estos problemas son comunes en la información que manejan aplicaciones relacionadas con procesamiento digital, reconocimiento de patrones y control automático. Las primeras aplicaciones de las redes neuronales a problemas reales se remontan a los años sesenta cuando fue introducido un modelo neuronal conocido como adalina o adaline, el cual se viene utilizando desde entonces en aplicaciones de filtrado adaptativo (por ejemplo, en los módems convencionales, canceladores de ecos, ecualizadores de canal, etc.), éste es un modelo muy conocido y ampliamente utilizado, y en ocasiones se hace más referencia a su carácter de dispositivo adaptativo lineal que a su naturaleza neuronal (Serrano, et al 2010).

En el contexto del filtrado adaptativo la neurona asociada a la adalina, guarda una estrecha relación con un filtro FIR (*Finite Impulse Response*), la diferencia es que ésta incorpora un parámetro adicional denominado bias o umbral, el cual le brinda un grado de libertad adicional. Así entonces, el entrenamiento de una red adalina es similar a modificar los coeficientes de un filtro FIR haciendo uso de un algoritmo adaptativo con el propósito de alcanzar un comportamiento deseado. Para cumplir esta tarea se han desarrollado diferentes algoritmos adaptativos, basados en uno de los dos más comunes criterios de optimización, el error cuadrático medio (algoritmos LMS, NLMS, ARγ, entre otros) o los mínimos cuadrados (algoritmos RLS, Kalman etc.) (Poularikas y Ramadan 2006). Siendo los algoritmos de la clase LMS (derivados de la teoría de Wiener) y RLS (derivados de la teoría de Kalman), los que más acogida han tenido en las aplicaciones prácticas (Brio y Sanz 2007).

El algoritmo Acelerador Regresivo versión γ (ARγ), incluido dentro de los algoritmos basados en el error cuadrático medio, resulta ser una posibilidad extraída del

filtrado adaptativo que puede aplicarse directamente a la adalina. Es preciso señalar que desde el punto de vista de la estimación o aproximación funcional, la utilidad de la adalina se ve limitada por ser un sistema lineal. Así, solamente podrá asociar correctamente patrones linealmente independientes, fallando en ocasiones ante patrones linealmente separables (Serrano, et al 2010).

Un esquema que se puede tener en cuenta para superar las limitaciones de la adalina, radica en la utilización de modelos no lineales de redes neuronales, entre los cuales se desea destacar el perceptrón multicapa MLP (*Multi-Layer Perceptron*), que permite abordar problemas complejos de clasificación y aproximación funcional de una manera eficaz y relativamente simple. El algoritmo mas ampliamente utilizado para entrenar esta arquitectura es el denominado retropropagación de errores o BP (*Back Propagation*), que al igual que el LMS (*Least Mean Square*) se basa en el método de gradiente descendiente, el cual busca minimizar una función de error o costo que depende de los parámetros de la red. Sin embargo, no se garantiza que usando esta técnica se tome el mejor camino hacia el mínimo y aún no existe un algoritmo que pueda considerarse superior en general, un buen método en un caso puede proporcionar un rendimiento pobre en otro. Por esto, es importante formular nuevos algoritmos de aprendizaje que busquen mejorar las prestaciones de los algoritmos actuales.

Dado que en el campo del filtrado adaptativo, el algoritmo ARγ presenta buen desempeño tanto en velocidad como en bajo desajuste, se propuso generar un algoritmo de aprendizaje que entrene redes neuronales MLP bajo los principios que rigen la actualización de parámetros en el algoritmo ARγ, con lo que surge el algoritmo ARγ-GLE cuyo diseño y validación por medio de experiencias basadas en simulación se presentan en este documento.

Contribución original: los estudios previos en referencia al algoritmo ARγ, se enmarcan por completo en el campo del filtrado adaptativo. Partiendo de lo anterior, el presente trabajo representa un salto que permite extender hacia las redes neuronales las buenas prestaciones demostradas por el algoritmo ARγ. Las redes neuronales son en la actualidad un activo campo de investigación en el que confluyen investigadores de diversas áreas (ingeniería, biología, estadística, entre muchas mas), por lo que los aportes en esta materia inciden en un amplio rango de aplicaciones.

Estructura de la tesis: en el capítulo 2 se presentan los aspectos fundamentales de la teoría de redes neuronales y filtrado adaptativo. En el Capítulo 3 se exponen los conceptos básicos sobre los cuales se desarrolla el algoritmo ARγ-GLE. En el Capítulo

4 se presenta un estudio experimental del algoritmo ARγ-GLE basado en simulación, el cual se realiza en torno a dos problemas relacionados con reconocimiento de patrones y aproximación funcional. En el Capítulo 5 se realiza un análisis comparativo de los resultados obtenidos con el algoritmo ARγ-GLE y el algoritmo BP ante tres reconocidos problemas de "benchmarking". Finalmente en el Capítulo 6 se presentan las conclusiones del trabajo realizado y se mencionan algunos proyectos futuros que pueden ser desarrollados a partir de las contribuciones del presente trabajo.

Capítulo 2
Fundamentos Teóricos

A lo largo de este capítulo se exponen los conceptos fundamentales de la teoría de redes neuronales. Se tratan inicialmente aquellos relacionados con la estructura de un sistema artificial, arquitecturas de red, algoritmos de entrenamiento y aplicaciones. Finalmente se presentan las ecuaciones características del algoritmo ARγ, abordando su descripción dentro de la teoría del filtrado adaptativo, que corresponde al contexto natural para el cual fue desarrollado.

2.1 Sistemas neuronales artificiales

Un sistema neuronal artificial o ANS (*Artificial Neural System*) es un esquema de procesamiento inspirado en la estructura de los sistemas nerviosos biológicos, que intenta reproducir sus capacidades para solucionar problemas, donde la información se presenta de forma masiva, imprecisa y distorsionada.

Tareas como el reconocimiento del habla, visión de objetos inmersos en el espacio natural, respuesta ante estímulos del entorno etc. son abordadas con gran éxito haciendo uso de sistemas neuronales. Este tipo de tareas son realizadas eficazmente por el cerebro biológico de especies que incluso pueden ser muy poco evolucionadas y, sin embargo, para los modernos computadores, pese a sus indiscutibles logros resultan un verdadero desafío. Los requerimientos computacionales que demandan este tipo de tareas son bastante elevados, por eso se hace necesario hacer uso de algún tipo de procesamiento paralelo para realizarlas, en eso radica la ventaja del cerebro en relación a las arquitecturas tipo Von Neumann (en las cuales se basan la mayoría de los computadores). El cerebro está compuesto por millones de procesadores elementales (*neuronas*) ampliamente interconectados que trabajan en paralelo, en tanto que las arquitecturas tipo Von Neumann utilizan un único procesador que actúa ejecutando

velozmente una secuencia de instrucciones. Cabe mencionar que ambos esquemas de procesamiento tienen ventajas en la realización de diferentes acciones, es así como las arquitecturas Von Neumann resultan mucho más adecuadas para resolver problemas de cálculo y razonamiento lógico (Yegnanarayana 2006).

2.1.1 Estructura de un sistema neuronal artificial

La unidad funcional de un sistema neuronal artificial es la neurona, la cual al agruparse con otras forma capas, y a su vez varias capas constituyen una red. Sin embargo, una red neuronal no es capaz de aprender por sí sola a realizar una tarea, para ello deben ser adicionados otros módulos que le permitan modificar los parámetros de su estructura a partir de señales provenientes del entorno. En general, un sistema neuronal está compuesto por los siguientes elementos (ver figura 2.1) (Brio y Sanz 2007).

- Un conjunto de neuronas artificiales.
- Un patrón de conexiones.
- Un algoritmo de entrenamiento.
- Interfaces de entrada y salida.

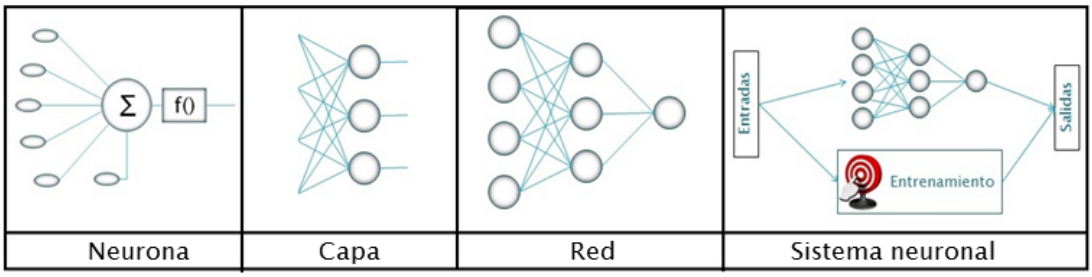


Figura 2.1. Estructura jerárquica de un sistema basado en ANS

Neurona artificial

Las ANS procesan la información en múltiples elementos llamados neuronas artificiales, las cuales pueden tener varias entradas pero únicamente una salida. El fun-

cionamiento general de una neurona i estándar obedece a las siguientes reglas (López 2010):

- Cada conexión de entrada trae asociado un parámetro denominado peso sináptico $w_{i,1}[n], w_{i,2}[n], w_{i,3}[n], ..., w_{i,R}[n]$ que se multiplica respectivamente por las señales de entrada $x_1[n], x_2[n], x_3[n], ..., x_R[n]$, los productos obtenidos se suman para calcular el potencial postsináptico $a_i[n]$. Ver figura 2.2.

- Una neurona i obtiene su salida $y_i[n]$ aplicando una función de activación o transferencia $f(.)$ al potencial postsináptico. Las principales funciones de activación son mostradas en la tabla 2.1.

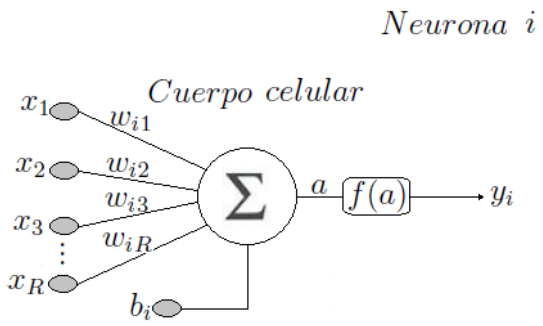


Figura 2.2. Modelo de neurona estándar

Con frecuencia se añade al conjunto de pesos de la neurona un parámetro adicional $b_i[n]$ denominado umbral o bias, el cual puede considerarse como un peso sináptico al que se le asocia permanentemente una entrada unitaria de valor negativo. La función de este parámetro es sencillamente brindarle a la neurona un grado de libertad adicional.

Patrón de conexiones

El patrón de conexiones determina la arquitectura de red y como tal su comportamiento, así entonces se pueden distinguir diferentes tipos de redes neuronales atendiendo a su estructura en capas y al flujo de datos interno de la red.

La estructura en capas permite diferenciar entre redes monocapa, compuestas por una única capa y redes multicapa, organizadas en varias capas. Por otro lado, el flujo

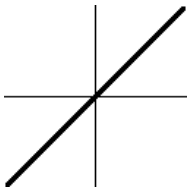
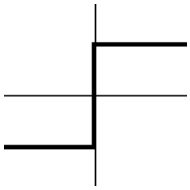
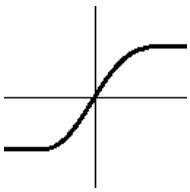
	Función	Rango	Gráfica
Identidad	$y_i[n] = a_i[n]$	$(-\infty, +\infty)$	
Escalón	$y_i[n] = Signo(a_i[n])$ $y_i[n] = Hs(a_i[n])$	$[-1, +1]$ $[0, +1]$	
Sigmoidea	$y_i[n] = Tanh(a_i[n])$ $y_i[n] = \frac{1}{1+e^{a_i[n]}}$	$(-1, +1)$ $(0, +1)$	

Tabla 2.1. Funciones de activación habituales

de datos internos de la red neuronal permite diferenciar entre redes unidireccionales y redes recurrentes o realimentadas. En las primeras, la información fluye en un solo sentido, mientras que en las ultimas, la información puede circular en diferentes direcciones.

Algoritmo de entrenamiento

Su función es gestionar las dinámicas de modificación y actualización de los pesos sinápticos de la red, con el propósito de aprender a asociar un conjunto entrada-salida. El algoritmo de aprendizaje por lo general se construye a partir de una función de error, la cual se optimiza mediante un proceso iterativo que se repite una y otra vez hasta cumplir con un criterio de parada.

Interfaces de entrada y salida

Son módulos de acondicionamiento que permiten al sistema neuronal interactuar con el entorno. Sus funciones son básicamente realizar una serie de preprocesamientos a las entradas y salidas de la red con el propósito de que sean utilizables por esta y por

los sistemas a los cuales esté conectada.

2.1.2 Modos de operación de un sistema neuronal

Un ANS puede operar en modo de entrenamiento y/o en modo de recuerdo (Brio y Sanz 2007).

Modo de entrenamiento

Una de las características principales de los ANS es su capacidad de aprender a partir del entorno, y mejorar su rendimiento a través del entrenamiento. El problema del entrenamiento se centra en determinar (a partir de un conjunto de pesos nulos o aleatorios) un conjunto de pesos sinápticos que permitan a la red realizar una función. Cuando un ANS se encuentra realizando este proceso se dice que está operando en modo de entrenamiento o aprendizaje. Los tipos de entrenamiento más aplicados en redes neuronales se presentan a continuación.

- **Entrenamiento supervisado.** Corresponde al conjunto de métodos más utilizados en el entrenamiento de ANS. Consiste en presentar un conjunto de entradas junto con sus respectivas salidas deseadas u objetivos, e iterativamente ajustar los pesos sinápticos hasta que la salida del sistema tienda a ser igual a la salida deseada. Durante este proceso se utiliza información detallada del error que se comete en cada paso. Al finalizar, los pesos se fijan en sus valores obtenidos quedando el sistema listo para ponerse en modo de ejecución.

- **Entrenamiento no supervisado o autoorganizado.** En el entrenamiento no supervisado no existe ningún maestro externo que indique si la red neuronal está operando correcta o incorrectamente, es decir que no se cuenta con un conjunto de salidas deseadas como en el caso del entrenamiento supervisado. Por tal motivo la red debe realizar un proceso de autoorganización que consiste en incorporar dentro de su estructura interna de pesos aquellos rasgos comunes, regularidades, correlaciones o categorías que vaya descubriendo por sí misma durante el entrenamiento.

- **Aprendizaje hibrido.** Es un tipo de aprendizaje en el que intervienen tanto el aprendizaje supervisado como el no supervisado, los cuales por lo general tienen lugar en diferentes capas de la red.

- **Aprendizaje reforzado.** Consiste en indicarle a la red si está actuando en forma correcta o incorrecta, sin brindar ningún detalle más, esto se realiza mediante un índice global de rendimiento de la red que se genera a partir del error. Se podría decir que es un tipo de aprendizaje intermedio entre el supervisado y el autoorganizado. En ocasiones se le denomina aprendizaje por premio o castigo.

Modo de recuerdo

Mencionado también en la literatura como modo de ejecución. Cuando un sistema neuronal se pone en modo de recuerdo (aunque no en todos los modelos) significa que ya cuenta con una estructura de pesos fija que asocia en buena manera los distintos patrones de entrada y salida utilizados en el entrenamiento, es entonces cuando el aprendizaje se desconecta y se espera que la red esté preparada para dar una solución correcta a entradas desconocidas.

2.1.3 Arquitecturas de red

Las redes neuronales se pueden clasificar atendiendo a dos de sus características más notables: el patrón de conexiones y el tipo de aprendizaje que utilizan, (ver figura 2.3).

Para efectos de este documento, se describen sólo los tres modelos más conocidos y habituales en las aplicaciones prácticas, como lo son: el perceptrón, la adalina y el perceptrón multicapa dada la importancia que tienen en esta investigación.

Perceptrón

Es un modelo unidireccional de una sola capa (ver figura 2.4). Fue el primero para el cual se planteó un mecanismo que permite determinar automáticamente los pesos sinápticos que clasifican correctamente a un conjunto de patrones, por esto tiene gran importancia histórica (Florez y Fernandez 2008).

El perceptrón puede utilizarse ya sea como un clasificador o para representar funciones booleanas, dado que utiliza funciones de activación escalón. Así entonces, para tareas de clasificación indicara con un 0 o un 1 la pertenencia o no a una clase (si se utiliza la función de activación $Hs(.)$).

Por otra parte, este modelo presenta importantes limitaciones, puesto que solamente es capaz de asociar patrones linealmente separables. Un patrón es linealmente separable cuando las regiones de decisión pueden ser divididas mediante una única

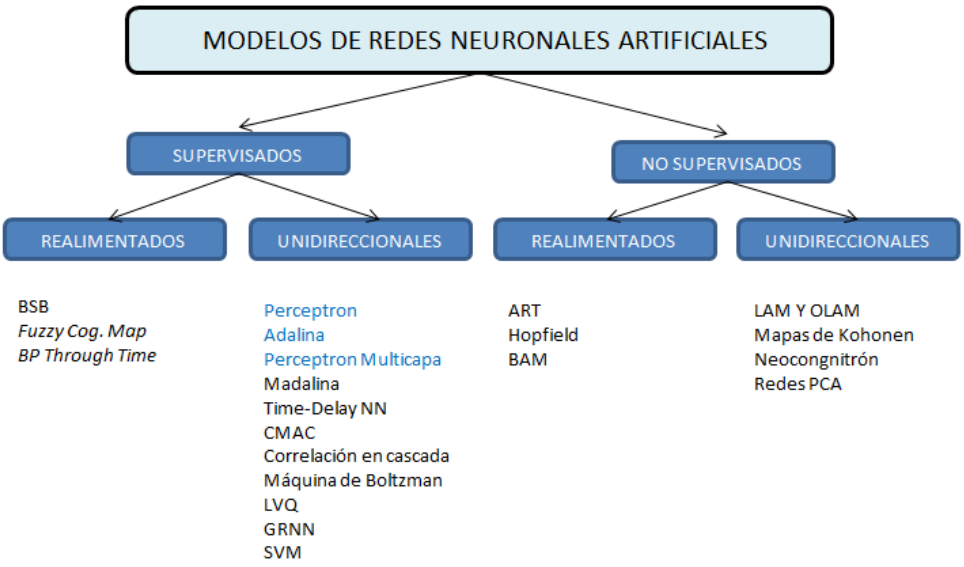


Figura 2.3. Clasificación de las redes neuronales artificiales

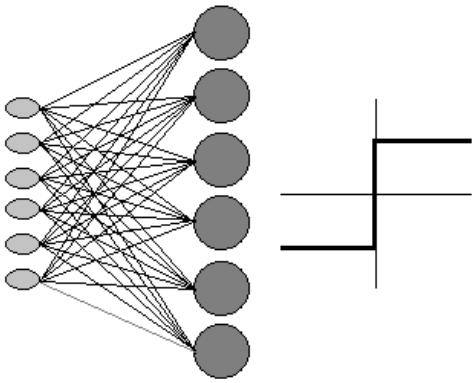


Figura 2.4. Arquitectura del perceptrón

condición lineal o hiperplano (ver figura 2.5). Una solución a las limitaciones del perceptrón consiste en incluir más capas en la arquitectura, con lo que se tendrá un perceptrón multicapa.

La operación de un perceptrón simple puede escribirse de la siguiente forma

$$y_i[n] = Hs(\sum_{j=1}^{R} w_{i,j}[n]x_j[n] - b_i[n]), \forall i, i \in D, \tag{2.1}$$

donde R representa el numero de entradas y D un conjunto que incluye a todas las neuronas de la red.

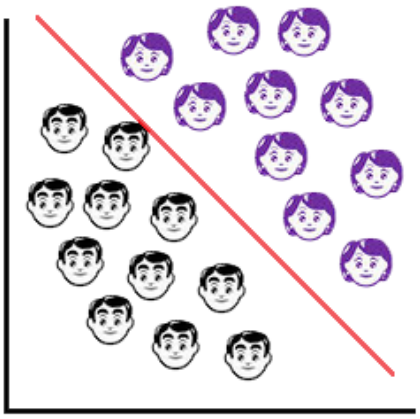


Figura 2.5. Patrones linealmente separables

Adalina

Es una de las arquitecturas más utilizadas en la práctica. Tiene una topología similar a la del perceptrón (ver figura 2.6) con la diferencia de que esta red usa una función de activación lineal (identidad) que permite a sus salidas tomar cualquier valor en un rango. La expresión matemática que define la operación de la adalina queda dada como sigue (Florez y Fernandez 2008)

$$y_i[n] = \sum_{j=1}^{R} w_{i,j}[n]x_j[n] - b_i[n], \forall i, i \in D, \tag{2.2}$$

nuevamente R representa el numero de entradas y D un conjunto que incluye a todas las neuronas de la red.

Por tratarse de un dispositivo lineal, la adalina se ve limitada en cuanto a que únicamente es capaz de discriminar patrones linealmente independientes, fallando en ocasiones ante patrones linealmente separables, que el perceptrón simple siempre discrimina.

Perceptrón multicapa (MLP)

Un perceptrón multicapa es un modelo neuronal no lineal compuesto de varias capas de neuronas entre la entrada y la salida, dichas capas se unen sin ningún tipo de realimentación, de tal manera que las entradas se conectan con la primera capa y las salidas de ésta con la siguiente capa y así sucesivamente hasta la última capa. Esta arquitectura puede establecer regiones de decisión más complejas (ver figura

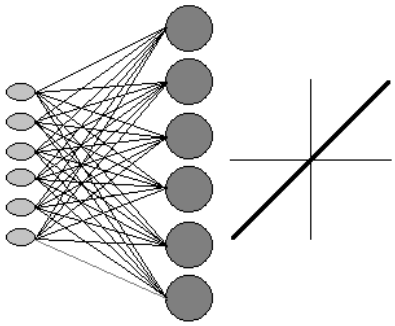


Figura 2.6. Arquitectura de la adalina

2.7) en comparación con el perceptrón simple, lo que le permite resolver problemas de clasificación que involucran patrones linealmente no separables, además puede ser usada también como un aproximador universal de funciones (Albuquerque, et al. 2009).

La figura 2.8 muestra la arquitectura abreviada de un MLP con una capa oculta. Por lo general, se utilizan funciones de activación sigmoideas para las capas ocultas y funciones de activación lineales para la capa de salida, esta es la arquitectura más común de MLP. Sin embargo, existen numerosas variantes (por lo general en problemas de clasificación) que utilizan funciones de activación sigmoideas en la capa de salida.

A menudo para hacer referencia a una arquitectura MLP se utiliza una secuencia de números, cuyos valores indican respectivamente el número de entradas, numero de neuronas en cada capa oculta y número de neuronas de salida. Así entonces, la arquitectura mostrada en la figura 2.8 es 6-6-2.

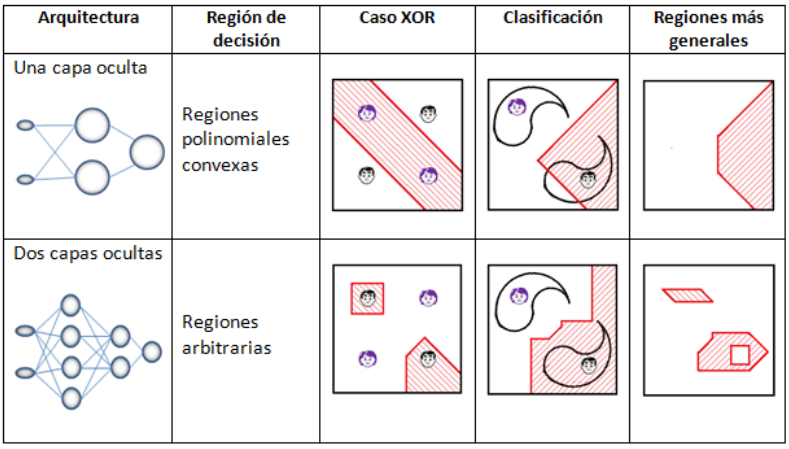


Figura 2.7. Regiones de decisión en un MLP

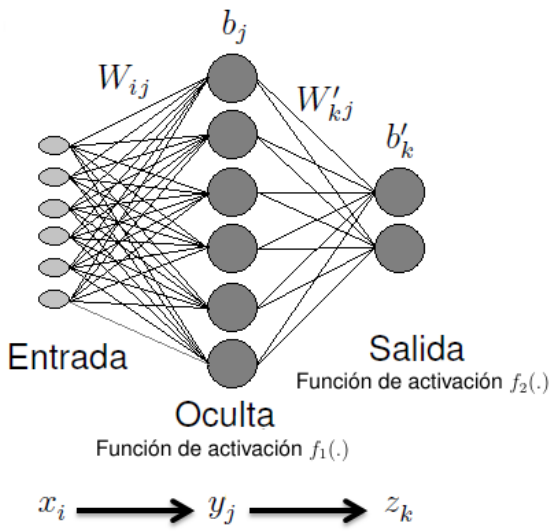


Figura 2.8. Arquitectura de un MLP

Si se denominan $x_i[n]$ a las entradas de la red, $y_j[n]$ a las salidas de la capa oculta y $z_k[n]$ a las salidas de la capa final o de salida. Y por otro lado, $w_{j,i}[n]$ a los pesos de la capa oculta y $b_j[n]$ sus bias, $w'_{k,j}[n]$ a los pesos de la capa de salida y $b'_k[n]$ sus bias. Entonces la operación de un MLP con una capa oculta se expresa matemáticamente como (Brio y Sanz 2007):

$$\begin{aligned} z_k[n] &= f_2\left(\sum_j w'_{k,j}[n]y_j[n] - b'_k[n]\right) \\ &= f_2\left(\sum_j w'_{k,j}[n]f_1\left(\sum_i w_{j,i}[n]x_i[n] - b_j[n]\right) - b'_k[n]\right), \end{aligned} \tag{2.3}$$

donde $f_1(.)$ y $f_2(.)$ son respectivamente las funciones de activación para las neuronas de la capa oculta y la capa de salida.

El tipo de funciones que pueden ser representadas por una red MLP depende del tamaño de la red. Aunque no está esclarecido totalmente la clase de funciones que pueden ser representadas por los diferentes tipos de redes, se han comprobado los siguientes enunciados (Mitchel 1997).

- **Funciones booleanas** Toda función booleana puede ser representada exactamente por una red con dos capas de neuronas, aunque el número de neuronas requeridas crece exponencialmente en el peor de los casos con el número de entradas. Para aclarar este concepto considérese un esquema general de red

(que permite representar cualquier función booleana) en el que para cada posible vector de entrada, se crea una neurona diferente en la capa oculta , fijando sus pesos sinápticos de tal modo que se active únicamente para un vector de entrada especifico. Hasta aquí se tiene una capa oculta que siempre tendrá una única neurona activa, lo que sigue es implementar la función lógica OR con una neurona de salida para que se active con los patrones deseados de entrada.

- **Funciones continuas.** Un MLP de una única capa oculta puede aproximar hasta un nivel deseado cualquier función continua en un intervalo, esto aplica para redes que usan neuronas con funciones de activación sigmoidales en la capa oculta y neuronas con funciones de activación lineales en la capa de salida.

- **Funciones arbitrarias.**

  Cualquier función puede ser aproximada hasta un nivel deseado por una red MLP de dos capas ocultas. Nuevamente las neuronas de salida deben tener función de activación lineal y las neuronas de las capas ocultas funciones de activación sigmoidal.

El modelo MLP y la aproximación funcional son respectivamente un modelo y un aspecto de gran importancia en la teoría de redes neuronales. Por esto los enunciados anteriores resultan de vital importancia puesto que proporcionan una sólida base teórica.

2.1.4 Algoritmos de entrenamiento.

Existen varios algoritmos para el entrenamiento de redes neuronales, los cuales se diferencian básicamente en el método que utilizan para minimizar la función de error y el modelo para el cual son aplicados. A continuación se describen los algoritmos utilizados en el entrenamiento de los modelos anteriormente descritos.

Algoritmo de aprendizaje del perceptrón.

Permite determinar en forma automática los pesos sinápticos que clasifican un conjunto de patrones (linealmente separables) partiendo de una arquitectura de perceptrón simple con pesos aleatorios (Yegnanarayana 2006). Ante un patrón de apren-

dizaje dado, este algoritmo calcula el incremento $\triangle w_{i,j}[n]$ del peso sináptico asociado a la entrada j de una neurona i perteneciente a un perceptrón simple con

$$\triangle w_{i,j}[n] = \begin{cases} 2\epsilon\ t_i[n]x_j[n], & si \quad y_i[n] \neq t_i[n] \\ 0, & si \quad y_i[n] = t_i[n] \end{cases} \tag{2.4}$$

siendo $y_i[n]$ y $t_i[n]$ la salida (definida en la ecuación 2.1) y salida deseada de la neurona i respectivamente.

La ecuación 2.4 puede ser reescrita como sigue

$$\triangle w_{i,j}[n] = \epsilon(t_i[n] - y_i[n])x_j[n]. \tag{2.5}$$

La regla del perceptrón se expresa haciendo uso de la ecuación 2.5 como

$$w_{i,j}[n+1] = w_{i,j}[n] + \epsilon(t_i[n] - y_i[n])x_j[n]. \tag{2.6}$$

Dado que las entradas y salidas del perceptrón únicamente toman valores de 1 y -1 (o 1 y 0 según se definan los valores lógicos) la actualización de pesos será siempre discreta. Así, las actualizaciones (cuando las entradas y salidas del perceptrón utilizan los valores lógicos de 1 y -1) toman únicamente valores de 0 o $\pm 2\epsilon$.

Algoritmo LMS

La regla de Windrow-Hoff o algoritmo LMS se utiliza para el entrenamiento de redes adalina y es uno de los algoritmos más habituales en la práctica por su simplicidad.

Una formulación que permite deducir el algoritmo LMS evitando consideraciones estocásticas, consiste en utilizar como función de costo la energía del error definida a continuación (Brio y Sanz 2007)

$$E(w_{i,j}[n]) = \frac{1}{2}\sum_{i \in D}(t_i[n] - y_i[n])^2, \tag{2.7}$$

donde para una neurona i perteneciente a una red adalina; $w_{i,j}[n]$ corresponde al peso sináptico asociado a la entrada j, D representa un conjunto que incluye todas las neuronas, $t_i[n]$ la salida deseada y $y_i[n]$ (definida en la ecuación 2.2) la salida de la neurona.

Para obtener el algoritmo LMS se aplica el método de gradiente descendiente a la ecuación 2.7. En términos generales, lo que se hace es determinar el sentido de

máxima variación (gradiente) en función de los pesos sinápticos $w_{i,j}[n]$ y modificar los pesos sinápticos siguiendo el sentido contrario como se muestra a continuación

$$\frac{\partial E(w_{i,j}[n])}{\partial w_{i,j}[n]} = \frac{1}{2}2(t_i[n] - y_i[n])\frac{\partial y_i[n]}{\partial w_{i,j}[n]} = -(t_i[n] - y_i[n])x_j[n]. \tag{2.8}$$

Usando la ecuación 2.8 se define el incremento $\triangle w_{i,j}[n]$ como

$$\triangle w_{i,j}[n] = -\hat{\alpha}\frac{\partial E(w_{i,j}[n])}{\partial w_{i,j}[n]} = \hat{\alpha}(t_i[n] - y_i[n])x_j[n] = \hat{\alpha}e_i[n]x_j[n], \tag{2.9}$$

donde $e_i[n]$ es el error cometido por la neurona i y $\hat{\alpha}$ es un parámetro de ajuste que determina el ritmo de aprendizaje.

Finalmente, la actualización de parámetros del algoritmo LMS se expresa como sigue

$$w_{i,j}[n+1] = w_{i,j}[n] + \hat{\alpha}e_i[n]x_j[n]. \tag{2.10}$$

La convergencia del algoritmo a la configuración $w_{i,j}*$ que minimiza el valor esperado del error, se garantiza si se cumplen las siguientes condiciones para los valores de $\hat{\alpha}$ (los cuales pueden depender de la iteración, $\hat{\alpha} = \hat{\alpha}[n]$):

$$\sum_{n=1}^{\infty}\hat{\alpha}(n) = \infty, \ \sum_{n=1}^{\infty}\hat{\alpha}^2(n) < \infty. \tag{2.11}$$

Por ejemplo $\hat{\alpha} = \hat{\alpha}[n] = n^{-1}$ satisface ambas condiciones, en muchas ocasiones es suficiente, con que $\hat{\alpha}$ tome valores pequeños ($0 < \hat{\alpha} < 1$).

Un aspecto que cabe destacar del algoritmo LMS es que conduce a actualizaciones de tipo continuo a diferencia de la regla de aprendizaje utilizada por el perceptrón.

Algoritmo "backpropagation" o BP

Es un algoritmo de aprendizaje supervisado que se deduce de aplicar el método de gradiente descendiente a las redes multicapa. Se utiliza en el entrenamiento de redes neuronales MLP y emplea dos fases para realizar la actualización de pesos. Con el propósito de describir las faces que lleva a cabo este algoritmo considérese la red neuronal con una neurona oculta mostrada en la figura 2.9.

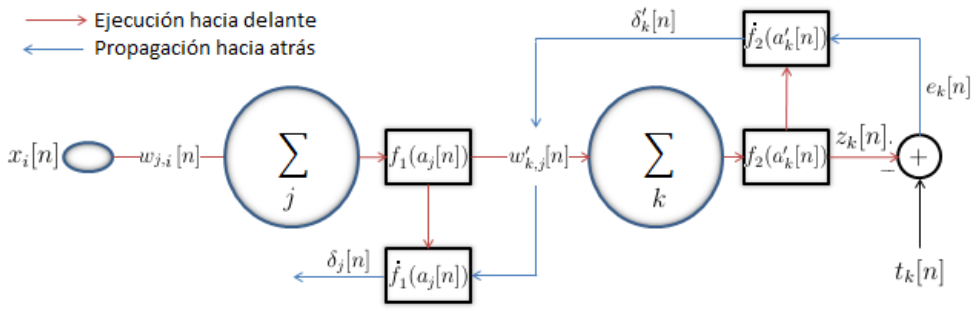


Figura 2.9. Red MLP con arquitectura 1-1-1

- **Fase de ejecución hacia delante.** Se procesa la señal de entrada $x_i[n]$ a través de la red con el propósito de obtener la señal $z_k[n]$, guardando en cada iteración las derivadas de las funciones de activación $\dot{f}_1(a_j[n])$ y $\dot{f}_2(a'_k[n])$ para cada neurona.

- **Fase de propagación hacia atrás.** Se calcula la señal de error $e_k[n]$ como la diferencia entre la salida deseada y la señal de salida ($t_k[n] - z_k[n]$). Esta se propaga hacia todas las capas de la red multiplicándose en su orden por $\dot{f}_2(a'_k[n])$, $w'_{k,j}[n]$ y $\dot{f}_1(a_j[n])$. La productoria resultante cada vez que interviene una derivada ($\dot{f}_2(a'_k[n]$ o $\dot{f}_1(a_j[n])$) nueva, se conoce como gradiente local de error y se utiliza en esta fase para calcular el incremento con el que se actualizan los pesos sinápticos.

El incremento se obtiene siguiendo el sentido contrario al gradiente de la energía del error. Puede calcularse a partir de los datos obtenidos en las fases de ejecución hacia delante y propagación hacia atrás como sigue (Palacios 2012):

$$\triangle w'_{k,j}[n] = -\hat{\alpha}\frac{\partial E[n]}{\partial w'_{k,j}[n]} = \hat{\alpha} y_j[n] e_k[n] \dot{f}_2(a'_k[n]) = \hat{\alpha} y_j[n] \delta'_k[n], \tag{2.12}$$

$$\triangle w_{i,j}[n] = -\hat{\alpha}\frac{\partial E[n]}{\partial w_{j,i}[n]} = \hat{\alpha} x_i[n] \delta'_k[n] w_{k,j}[n] \dot{f}_1(a_j[n]) = \hat{\alpha} x_i[n] \delta_j[n], \tag{2.13}$$

donde $\delta'_k[n]$ y $\delta_j[n]$ representan son los gradientes locales de error para las neuronas k y j respectivamente, y se obtienen en la fase de propagación hacia atrás.

Esta formulación puede extenderse a redes de varias capas siguiendo la misma heurística. Si la salida de una neurona está conectada a múltiples neuronas hacia adelante. Por ejemplo, suponiendo que hubieran s' neuronas en la capa de salida, con

gradientes locales $\delta'_1[n]$, $\delta'_2[n]$, ..., $\delta'_{s'}[n]$ respectivamente. Entonces el gradiente local de error para la neurona j de la capa oculta se calcularía como

$$\delta_j[n] = \delta'_1[n]w'_{1,j}[n]\dot{f}_1(a_j[n]) + \delta'_2[n]w'_{2,j}[n]\dot{f}_1(a_j[n]) + ... + \delta'_{s'}[n]w'_{s',j}[n]\dot{f}_1(a_j[n]), \quad (2.14)$$

siendo $w'_{k,j}[n]$ el peso sináptico asociado a la conexión entre la neurona j y la neurona k, para $k = 1, 2, ..., s'$.

La actualización de los pesos queda definida como sigue a continuación:

$$w'_{k,j}[n+1] = w'_{k,j}[n] + \hat{\alpha}y_j[n]\delta'_k[n], \quad (2.15)$$

$$w_{j,i}[n+1] = w_{j,i}[n] + \hat{\alpha}x_i[n]\delta_j[n], \quad (2.16)$$

donde $\hat{\alpha}$ representa la tasa de aprendizaje. Por lo general, en modo de entrenamiento, este parámetro se mantiene constante, sin embargo, si la tasa de aprendizaje es muy alta el algoritmo puede oscilar y entrar en inestabilidad, si por el contrario la tasa de aprendizaje es muy pequeña, el entrenamiento puede tornarse muy lento (Mukherjee y Routroy 2012).

Comúnmente se adiciona al algoritmo BP un término adicional (*momentum*) proporcional al incremento de la iteración anterior, con el propósito de introducir una cierta inercia en el entrenamiento. Esta variante se conoce como algoritmo BP con momento y las expresiones para la actualización se definen a continuación:

$$w'_{k,j}[n+1] = w'_{k,j}[n] - \hat{\alpha}\frac{\partial E[n]}{\partial w'_{k,j}[n]} + \hat{\mu}\triangle w'_{k,j}[n-1], \quad (2.17)$$

$$w_{j,i}[n+1] = w_{j,i}[n] - \hat{\alpha}\frac{\partial E[n]}{\partial w_{j,i}[n]} + \hat{\mu}\triangle w_{j,i}[n-1], \quad (2.18)$$

siendo $\hat{\mu}$ una constante que ajusta el termino momento, la cual puede tomar valores entre 0 y 1 (Brio y Sanz 2007).

2.1.5 Presentación de patrones de entrenamiento.

Un patrón de entrenamiento o aprendizaje está conformado por un vector de entradas $\mathbf{x}[n] = [x_1[n], x_2[n], x_3[n], ..., x_R[n]]^T$ y su respectivo vector de objetivos o salidas deseadas $\mathbf{t}[n] = [t_1[n], t_2[n], t_3[n], ..., t_{s'}[n]]^T$ (^{T} denota vector transpuesto). Durante el

entrenamiento, el aprendizaje se lleva a cabo mediante la presentación sucesiva de un conjunto de patrones de entrenamiento. Cuando todo este paquete de datos ingresa a la red, se dice que ha transcurrido una época o ciclo de entrenamiento. Así, el proceso de aprendizaje se repite época tras época hasta que los pesos sinápticos se estabilicen en sus valores adecuados o se cumpla algún otro criterio de parada.

Los patrones de entrenamiento son presentados a la red, generalmente, de dos formas (Beale, et al 2010):

Modo secuencial o "online"

Los pesos sinápticos se actualizan a medida que cada patrón de entrenamiento es presentado a la red neuronal.

Modo por lotes o "batch"

Los pesos sinápticos son actualizados una vez se haya presentado todo el conjunto de entrenamiento a la red, es decir cada que transcurre una época.

2.1.6 Complejidad computacional de un algoritmo

La complejidad de un algoritmo está relacionada con el número de operaciones elementales $OE(.)$ (como por ejemplo asignaciones, comparaciones, sumas, restas, multiplicaciones, divisiones, etc.) necesarias para resolver un problema dado. La complejidad del problema determina la cantidad de parámetros de red (neuronas, entradas, pesos sinápticos, etc.). En general si el problema es sencillo, bastarán pocos parámetros: deberá utilizarse una red pequeña. Si el problema es complejo, se necesitarán más parámetros: se necesitará una red de mayor tamaño. Así entonces, resulta conveniente expresar el número de operaciones $OE(.)$ realizadas por un algoritmo, en función de los parámetros de la red. En la práctica, la mejor forma para diferenciar la eficiencia de los algoritmos es estudiar su orden de complejidad, el cual se expresa en función del número de operaciones $OE(.)$.

Orden de complejidad

Se dice que la complejidad de un algoritmo es de orden $G(\Psi)$ (representado como $O(G(\Psi))$ si existe un Ψ_0 tal que, para todo valor de $\Psi \geq \Psi_0$ existe una constante C, tal

que $|OE(\Psi)| \leq C|G(\Psi)|$. Conforme a esto, si $OE(\Psi)$ es un polinomio de grado k, el algoritmo correspondiente tendrá una complejidad $O(\Psi^k)$. Así es como se formaliza la idea de que dos algoritmos tengan la misma eficiencia (Quetglás, et al 1994).

En el caso del algoritmo BP, el orden de complejidad se determina teniendo en cuenta las operaciones en la fase de ejecución hacia adelante y la fase de propagación hacia atrás. En la primera fase se determina la salida de la red y se calculan las derivadas de cada neurona. Calcular la salida de la red tiene una complejidad de orden $O(W)$, siendo W el número de pesos sinápticos de la red; El orden de complejidad para determinar las derivadas de cada neurona es $O(N)$, donde N representa el número de neuronas. Finalmente en la fase de propagación hacia atrás se actualizan los parámetros conforme a las expresiones propias del algoritmo BP. Por tanto el orden de complejidad total del algoritmo será $O(2(W + N))$, pero como $N \leq W$ este se expresa como un orden lineal con el número de pesos de la red $O(W)$ (Palacios 2012).

2.1.7 Problema de la generalización

La generalización de una red neuronal está relacionada con la capacidad que tiene para abstraer las características funcionales implícitas en los patrones de aprendizaje y no simplemente memorizar los ejemplos presentados. Es gracias a la generalización que los ANS son capaces de brindar respuestas adecuadas ante patrones jamás utilizados en el entrenamiento (Brio y Sanz 2007).

Las redes neuronales son herramientas de procesamiento poderosas por cuanto son capaces de aprender a modelar situaciones muy complejas. Sin embargo, esta característica puede llegar a convertirse en un serio problema, puesto que si se entrena demasiado una red neuronal se corre el riesgo de incurrir en sobreaprendizaje, apartándose del verdadero propósito de una red neuronal que consiste en generalizar.

El sobreaprendizaje es uno de los problemas que presentan las técnicas de regresión y clasificación en general, y se produce cuando los modelos utilizados se ajustan demasiado al conjunto de ejemplos (aprendiendo incluso el ruido presente en ellos), de forma que no consiguen generalizar lo suficientemente bien y fallan. Por esta razón la red debería entrenarse hasta un punto óptimo en el que el error de generalización es mínimo. Esto se realiza utilizando un procedimiento , denominado validación cruzada (Palacios 2012).

Validación cruzada

La validación cruzada es una de las técnicas más utilizadas en la evaluación de métodos de clasificación y regresión (entre los cuales se incluyen las ANS), como tal es frecuentemente utilizada para evaluar el entrenamiento en redes neuronales. El principal interés con el uso de la validación cruzada es la habilidad de generalización de la red, es decir el rendimiento ante patrones nuevos.

La validación cruzada consiste básicamente en entrenar y validar a la vez, para esto en el proceso de entrenamiento, debe considerarse un error de entrenamiento, un error de validación y un error de prueba. En la práctica, estos errores se suelen calcular como el error cuadrático medio o MSE (*Mean Square Error*) de los resultados proporcionados por la red ante patrones pertenecientes a tres subconjuntos de las muestras disponibles (Brio y Sanz 2007). La expresión matemática para calcular el MSE se define como sigue

$$MSE = \frac{1}{2P} \sum_{u=1}^{P} ||\mathbf{t}_u - \mathbf{F}(\mathbf{x}_u)||^2, \tag{2.19}$$

siendo $\mathbf{t}_u$ el vector de salidas deseadas para el vector de entrada $\mathbf{x}_u$, P el número de muestras en el conjunto de datos utilizado y **F(.)** la función de red propia del modelo utilizado, de tal forma que $\mathbf{F}(\mathbf{x}_u)$ es la salida de la red.

Los tres conjuntos de datos utilizados en el proceso de validación cruzada se describen a continuación:

- Conjunto de Entrenamiento: Es el subconjunto de muestras utilizado para entrenar la red y para determinar el error de aprendizaje.

- Conjunto de validación: Es el subconjunto de muestras con el cual se pretende determinar una configuración óptima de pesos. El error calculado con estos datos (error de validación) permite medir el rendimiento de la red ante patrones no utilizados previamente en el entrenamiento, como tal representa un estimativo importante para evaluar la generalización alcanzada por la red. Durante el entrenamiento, se pueden presentar varios mínimos para el conjunto de validación, por esto se deben ir guardando periódicamente las configuraciones de pesos intermedias para posteriormente seleccionar la que proporcionó el mínimo error de validación.

- Conjunto de Prueba: Es el subconjunto que permite determinar en forma totalmente objetiva el rendimiento final de la red y se utilizaría una única vez cuando se haya decidido la mejor configuración de parámetros. Sin embargo, resulta también útil determinar el error de prueba durante el entrenamiento, puesto que si este alcanza un mínimo en una iteración significativamente diferente al alcanzado con el conjunto de validación, esto sería el indicativo de una inadecuada partición de datos.

Cuando el tamaño de estos conjuntos no es lo suficientemente grande, se pueden emplear otras variantes del mismo procedimiento, como por ejemplo, utilizar m particiones de la parte de entrenamiento y validación. Uno de los subconjuntos se utiliza como datos de prueba y el resto $(m-1)$ como datos de entrenamiento. El proceso de validación cruzada es repetido durante k veces, con cada uno de los subconjuntos de prueba. Finalmente se promedia los resultados. Cuando los m subconjuntos son unitarios, se tiene el método de dejar uno fuera (*leave one out*) (Bergmeir y Benítez 2012).

2.2 Aplicaciones de las redes neuronales MLP

Las arquitecturas MLP representan un modelo central en la teoría de redes neuronales, y se aplican a diversos problemas relacionados principalmente con reconocimiento de patrones y aproximación de funciones. Su desarrollo representa gran parte de los avances de la neuromputación en general, por lo que tienen incidencia en diversos campos de aplicación, algunos de los cuales se presentan en la tabla 2.2 (Beale, et al 2010).

2.2.1 Clasificación de patrones

Las redes neuronales pueden utilizarse como clasificadores de patrones. La función de un clasificador es analizar los rasgos específicos de un patrón **x** de entrada, y asociarlo a una determinada clase. Un patrón se representa como un R-vector $\mathbf{x} = [x_1, x_2, x_3, ..., x_R]^T$, cuyas componentes representan rasgos o características importantes para clasificar un objeto. Entonces (asumiendo que existen C grupos o clases $\omega_1, \omega_2, \omega_3, ..., \omega_C$,), a cada patrón **x** se le debe asociar una variable categórica z de tal forma que si $z = l$, el patrón **x** pertenecerá a la clase ω_l donde $l \in \{1, 2, 3, ..., C\}$.

Como ejemplo de patrones se tienen las mediciones de ondas acústicas, usadas en reconocimiento del habla; mediciones de variables biológicas, para determinar alguna enfermedad; medidas de variables atmosféricas, para predicción del estado del tiempo o una imagen digitalizada para reconocimiento de caracteres o reconocimiento facial (Webb 2002).

2.2.2 Aproximación de funciones

Las redes neuronales son aproximadores universales de funciones, bastante útiles en tareas relacionadas con el modelado de sistemas, control automático, predicción de series temporales; son también ampliamente utilizadas en el campo financiero por ejemplo para la concesión de préstamos y análisis de mercados. En general el problema de la aproximación funcional puede describirse como sigue. Considérese el par (Martinez 2005)

$$(y_l, \mathbf{x}_l) = (y_l, x_{l,1}, x_{l,1}, x_{l,1}, ..., x_{l,R}),\ l = 1, 2, ..., n \tag{2.20}$$

donde los valores y_l han sido generados a partir del modelo

$$y_l = g(\mathbf{x}_l, \theta) + \epsilon_l,\ l = 1, 2, ..., n \tag{2.21}$$

siendo $\mathbf{x}_l$ las variables independientes del problema. La función g es una función desconocida definida en el espacio euclidiano de dimensión R con valores en $\Re$. Mientras que ϵ_l, para $l = 1, 2, ..., n$, son variables aleatorias de media cero, independientes entre sí e independientes de $\mathbf{x}_l$. El objetivo de la aproximación funcional es construir un estimador $\hat{g}$ en función de los datos $(y_l, \mathbf{x}_l)$, para aproximar la función desconocida g y usar esta estimación para predecir el valor de y dado un nuevo valor de $\mathbf{x}$, de tal modo que

$$y = \hat{g}(\mathbf{x}, \theta), \tag{2.22}$$

para esto, debe encontrarse una configuración optima de parámetros para algún modelo regresivo. Las redes neuronales son modelos regresivos poderosos. Está comprobado teóricamente que una red multicapa unidireccional de una capa oculta es capaz de aproximar hasta el nivel deseado cualquier función continua en un intervalo (Mitchel 1997).

Campo	Aplicación
Aeroespacial	Pilotos automáticos para avión, simuladores de vuelo, sistemas de control para aviones, ayuda al pilotaje, detección de fallas en máquinas de vuelo, simulación de componentes.
Automotriz	Sistemas automáticos de ensamblaje.
Bancario	Evaluación de créditos y lectura de cheques u otros documentos.
Militar	Guiado automático de misiles, combate aéreo, discriminación de objetos y reconocimiento facial.
Entretenimiento	Animación, efectos especiales y predicción de mercados.
Electrónica	Visión artificial, síntesis de voz, modelado no lineal y diseño de circuitos integrados.
Financiero	Calculo de avalúos, concesión de créditos, análisis de la linea de créditos, monitoreo de actividades con tarjeta de crédito, análisis corporativo y predicción de precios.
Industrial	Predicción de procesos industriales como salida de gases, detección de fallas y determinación del punto de operación de procesos.
Manufacturero	Control de procesos manufactureros, diseño y análisis de productos, diagnostico de procesos y maquinaria, identificación de partículas en tiempo real, sistemas de inspección de calidad, predicción de calidad en la elaboración de papel, análisis de calidad en soldadura y modelado de sistemas de procesamiento químico.
Medico	Análisis de células cancerosas, clasificación de señales electrocardiográficas y electroencefalográficas, predicción de la reacción de pacientes ante demasiados tratamientos y reducción de gasto en hospitales.
Petrolero	Exploración.
Rebotica	Control de trayectoria, sistemas de visión artificial y síntesis de voz.

Tabla 2.2. Aplicaciones reales de las redes neuronales

2.3 Filtrado adaptativo y algoritmo ARγ

El filtrado adaptativo hace referencia a un conjunto de métodos y estructuras utilizadas en procesamiento digital de señales, que se incluye dentro del extenso campo de la teoría de optimización. En general, las técnicas de filtrado adaptativo se utilizan para abordar tareas en las cuales se desconocen algunos parámetros del problema o bien, estos varían en forma lenta y desconocida. Como tal, resultan verdaderamente interesantes a la hora de diseñar aplicaciones en las cuales la variabilidad del entorno puede llegar a ser crítica, como es el caso de la transmisión de información en sistemas de comunicación, donde un cambio en las condiciones atmosféricas puede modificar las características del canal, haciendo difícil descifrar la información recibida. Para superar este inconveniente, se suele utilizar un filtro adaptativo en el receptor (ecualización adaptativa), capaz de ajustarse a los diferentes patrones de distorsión que introduce el medio (Poularikas y Ramadan 2006).

2.3.1 Filtros adaptativos

Un filtro adaptativo puede entenderse como un sistema capaz de ajustar algunos de sus parámetros con el propósito de modelar su relación entrada-salida, de tal manera que cumpla con un criterio definido por el propio sistema o por su entorno. La Figura 2.10 muestra el diagrama general de un filtro adaptativo, el cual está conformado por tres módulos fundamentales (Manolakis et al 2005).

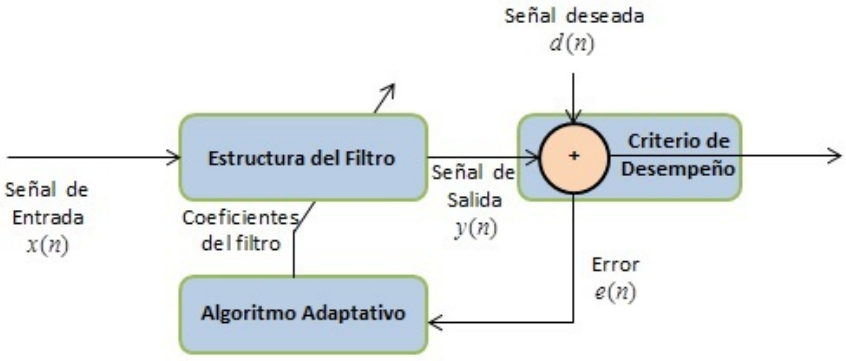


Figura 2.10. Elementos de un filtro adaptativo

- Estructura del filtro: Define la forma en como la señal de salida será calculada a partir de la señal de entrada, su funcionamiento está definido por un conjunto

de coeficientes, los cuales pueden ser modificados para cumplir con una determinada relación de entrada salida. La estructura del filtro puede ser de tipo FIR o de tipo IIR. Otra alternativa, se muestra en la figura 2.11, y consiste en una adalina conectada a un registro de estados (un registro de estados es un dispositivo que guarda $R-1$ estados pasados de la señal de entrada), la única diferencia de esta configuración y un filtro FIR es el bias b incorporado en la adalina.

- Criterio de desempeño: Evalúa el rendimiento del sistema respecto a los requerimientos de una aplicación en particular, para lo cual usa alguna función de error que le permite comparar la salida del filtro con la señal deseada.

- Algoritmo adaptativo: Es un procedimiento iterativo que permite ajustar los coeficientes del filtro en función del criterio de desempeño, de tal manera que se minimice la función de error.

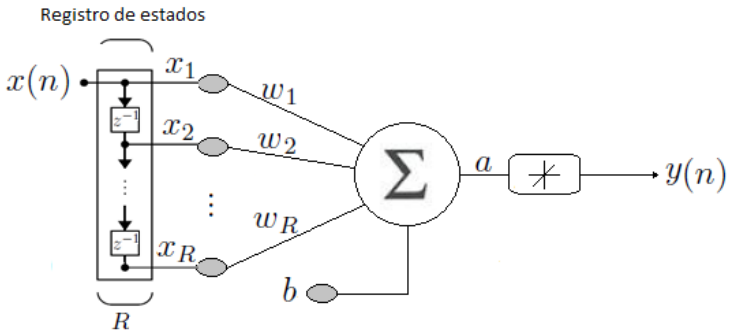


Figura 2.11. Neurona de la adalina configurada para aplicaciones de filtrado adaptativo

En la literatura se pueden encontrar diferentes tipos de algoritmos adaptativos, (LMS, NLMS, ARγ, etc.) entre los que se destaca el algoritmo ARγ por su buen desempeño tanto en velocidad como en bajo desajuste.

2.3.2 Algoritmo ARγ

El Algoritmo ARγ es una variante derivada del algoritmo acelerador en tiempo continuo propuesta en 1998 que ajusta la segunda derivada de parámetros, el cual al ser discretizado genera tres versiones conocidas como algoritmo acelerador completo (AAC), algoritmo acelerador progresivo (APCM) y algoritmo acelerador regresivo (ARCM). Es destacable la elevada complejidad computacional del algoritmo APCM en relación con

el buen desempeño del algoritmo ARCM y los excelentes resultados que arrojó el análisis de estabilidad para los algoritmos ARCM y AAC. Partiendo de esto, los esfuerzos se enfocaron en disminuir la complejidad de ARCM y mejorar su desempeño, con lo que surgió el algoritmo ARγ, que tiene como características el tener tres parámetros de ajuste, lograr una buena velocidad de convergencia y paralelamente una considerable reducción del error de medida final (Jojoa 2003). Las ecuaciones que describen el algoritmo ARγ son:

$$e\left[n\right] = \mathbf{x}^T\left[n\right]\mathbf{w}\left[n-1\right] - d\left[n\right], \tag{2.23}$$

$$g\left[n\right] = \frac{e\left[n\right] + \gamma\mathbf{x}^T\left[n\right]\mathbf{w}\left[n-1\right]}{1 + \alpha\gamma m_1\mathbf{x}^T\left[n\right]\mathbf{x}\left[n\right]}, \tag{2.24}$$

$$\mathbf{q}\left[n\right] = \frac{\gamma}{\alpha+\gamma}\left[\mathbf{q}\left[n-1\right] - \alpha m_1 g\left[n\right]\mathbf{x}\left[n\right]\right], \tag{2.25}$$

$$\mathbf{w}\left[n\right] = \mathbf{w}\left[n-1\right] + \alpha\mathbf{q}\left[n\right], \tag{2.26}$$

$d\left[n\right]$ corresponde a la señal deseada obtenida de la siguiente forma:

$$d\left[n\right] = \mathbf{x}^T\left[n\right]\mathbf{w}_0 + r\left[n\right], \tag{2.27}$$

donde:

$\mathbf{x}\left[n\right]$: Vector de la señal de entrada.
$\mathbf{w}\left[n\right]$: Vector de coeficientes del filtro adaptativo.
$d\left[n\right]$: Escalar que corresponde a la señal deseada en el instante n.
$e\left[n\right]$: Escalar que corresponde al error de medida en el instante n.
$g\left[n\right]$: Escalar auxiliar en el instante n.
$\mathbf{q}\left[n\right]$: Vector auxiliar.
$\mathbf{w}_0\left[n\right]$: Vector de coeficientes óptimo.
α: Parámetro de ajuste fijo.
γ: Parámetro de ajuste fijo.
m_1: Parámetro de ajuste fijo.

La ventaja que presenta este algoritmo es que los parámetros de ajuste son cantidades escalares (α, γ y m_1). El algoritmo converge para valores de α, γ y m_1 positivos ($\alpha > 0, \gamma > 0$ y $m_1 > 0$). Así mismo se determinó que éste presenta un mínimo error de desajuste cuando $\alpha\gamma m_1 \approx H$ (criterio de mínimo error) en que H es una constante real positiva con un valor aproximado a 2. En la Tabla 2.3 se indican algunos valores

de H para diversos valores de m_1, α y γ con los que se alcanza un mínimo desajuste (Jojoa 2003).

m_1	α	γ	H
1	1.0	2.04	2.040
1	1.5	1.35	2.025
3	1.0	0.68	2.040
3	1.5	0.45	2.025
11	1.0	0.18	1.980
11	1.5	0.12	1.980

Tabla 2.3. Valores de H para diferentes condiciones de m_1, α y γ

Capítulo 3

Un Algoritmo para Redes MLP Basado en el Algoritmo ARγ

Este capítulo articula los conceptos fundamentales bajo los cuales se desarrolla el algoritmo ARγ-GLE (Algoritmo Acelerador Regresivo Versión γ con Gradiente Local de Error) el cual permite el entrenamiento de modelos MLP haciendo uso de los principios que rigen la actualización de parámetros en el algoritmo ARγ. Se ha comentado anteriormente que el algoritmo ARγ puede aplicarse directamente al caso de la adalina. Sin embargo, como se muestra a continuación, para el caso de un perceptrón multicapa deben hacerse unas consideraciones adicionales relacionadas con la arquitectura propia del modelo.

3.1 Consideraciones iniciales

El algoritmo acelerador regresivo versión γ, al igual que el algoritmo LMS, se incluye dentro del grupo de algoritmos que utilizan como función de costo el error cuadrático medio. Esta función define una superficie de forma paraboloide que al igual que la parábola en el plano, usualmente posee un único mínimo, aunque en ocasiones puede presentar una forma degenerada con uno o más canales, pero todos de la misma profundidad. En cualquiera de los dos casos la función $E[w_{i,j}[n]]$ es mínima en ese punto o en cualquiera de los canales y los algoritmos LMS y ARγ permiten llegar directamente a él, sin importar la configuración inicial de pesos.

Por otro lado, no existe un único camino para llegar al mínimo, por tanto los caminos a tomar están relacionados directamente con los principios que rigen la actualización de parámetros en el algoritmo utilizado y estos a su vez en las técnicas de optimización utilizadas para la implementación del algoritmo. En el caso del algoritmo ARγ se emplean técnicas basadas en la segunda derivada para optimizar el error, mientras que

en el algoritmo LMS se usa el método de optimización conocido como gradiente descendiente.

El algoritmo ARγ no contempla una heurística que permita el entrenamiento de las capas ocultas del perceptrón multicapa, además fue concebido para ajustar modelos de tipo lineal, de allí que su aplicación directa a las redes neuronales, se restringe a modelos monocapa como la adalina. En la práctica, las capas ocultas de una red MLP no cuentan con un patrón de aprendizaje directo como el que tiene la capa de salida, este fue un problema que detuvo durante un largo tiempo el avance de las redes neuronales artificiales, cuya solución se logró en gran parte, gracias a los aportes del grupo PDP (*Parallel Distributed Processing Group*) de la Universidad de California, que presentó el algoritmo BP ante la comunidad internacional, como una técnica útil de resolución de problemas complejos, lo que despertó el interés no solo por el perceptrón multicapa, sino por la neurocomputación en general (Brio y Sanz 2007).

El algoritmo BP por su parte puede entenderse como una consecuencia natural de extender el algoritmo LMS a las redes multicapa, por tanto intenta optimizar un funcional de error similar haciendo uso del método de gradiente descendiente. Sin embargo, como la salida general de la red depende de los pesos sinápticos de todas las neuronas, las derivadas deben realizarse en función de los pesos sinápticos de la capa de salida, como de los pesos sinápticos de las capas ocultas. Como consecuencia, todas las funciones de activación deben ser derivables (generalmente no lineales) (Haykin 1998).

El algoritmo ARγ-GLE surge como resultado de incorporar en la estructura matemática del algoritmo ARγ, un funcional de error, conocido como gradiente local de error, el cual, más que un funcional, representa todo un sistema para la administración de errores en las capas de una red MLP, permitiendo entrenar las capas ocultas y además compensar las no linealidades introducidas por las funciones de activación utilizadas. Para sintetizar, el gradiente local de error en combinación con las propiedades del algoritmo ARγ, generan una nueva propuesta para recorrer el espacio de pesos en búsqueda de un mínimo apropiado de la función de error, con lo que surge el algoritmo ARγ-GLE .

3.2 Gradiente local de error

El gradiente local de error es un concepto que se deriva de aplicar el método de gradiente descendiente a las redes MLP y refleja una relación muy estrecha entre el algoritmo LMS y el algoritmo BP. Se define como el gradiente de la energía total de la red en función del potencial postsináptico de una neurona en particular, como tal pueden ocurrir dos casos, los cuales se desarrollan a continuación haciendo uso de las expresiones propias del algoritmo ARγ.

3.2.1 Caso 1: Neurona en capa de salida

Supóngase una neurona perteneciente a la capa de salida de una red MLP como la que se muestra en la figura 3.1, nótese que el bias $b'_k = w'_{k,0}$ se considera como un peso sináptico adicional, al cual se le asocia permanentemente una entrada de valor $y_0 = -1$. Entonces, el gradiente local de error $\delta'_k(n)$ para una neurona k de la capa de salida se define como

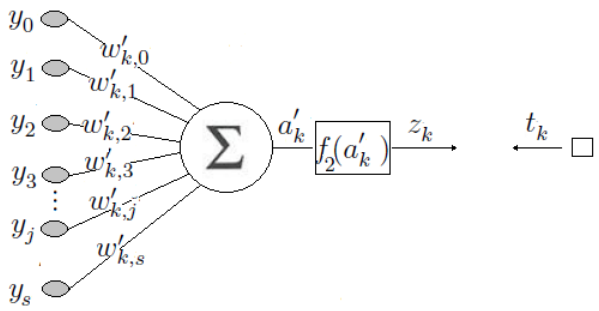


Figura 3.1. Conexiones de una neurona en capa de salida

$$\delta'_k[n] = \frac{\partial E[n]}{\partial a'_k[n]}, \tag{3.1}$$

donde $E[n]$ corresponde al valor instantáneo de la energía del error total y $a'_k[n]$ al potencial postsináptico de la neurona k (Haykin 1998).

El potencial postsináptico es análogo a la salida de un filtro adaptativo, como tal existe una expresión dentro de la estructura interna del algoritmo LMS y el algoritmo ARγ que lo estima. Sin embargo, no se hace de la misma forma en ambos casos, la diferencia radica en que el algoritmo LMS utiliza únicamente estados presentes de los pesos sinápticos. Por su parte, el algoritmo ARγ estima el potencial postsináptico,

haciendo uso de estados pasados de los pesos sinápticos. Así entonces, un algoritmo neuronal basado en el algoritmo ARγ calcularía el potencial postsináptico de una neurona de la capa de salida con la siguiente expresión

$$a'_k[n] = \sum_{j=0}^{U} y_j[n] w'_{k,j}[n-1]. \tag{3.2}$$

Por otro lado, el valor instantáneo de la energía del error total $E[n]$ se calcula sumando la energía del error instantáneo de todas las neuronas en la capa de salida. Por tanto, si se define la energía del error instantáneo para una neurona k como $1/2e_k^2[n]$, entonces $E[n]$ se expresa del siguiente modo (Haykin 1998)

$$E[n] = \frac{1}{2} \sum_{k \in D} e_k^2[n], \tag{3.3}$$

donde el conjunto D incluye todas las neuronas en la capa de salida de la red y $e_k[n]$ es el error cometido por la neurona de salida, que corresponde a la diferencia entre la señal de salida $z_k[n]$ y la señal deseada $t_k[n]$

$$e_k[n] = z_k[n] - t_k[n], \tag{3.4}$$

la señal de salida $z_k[n]$ se obtiene al aplicar una función de activación $f_2(.)$ al potencial postsináptico, por tanto se puede expresar como

$$z_k[n] = f_2(a'_k[n]). \tag{3.5}$$

Por otro lado, la regla de la cadena permite reescribir la ecuación 3.1 como

$$\delta'_k[n] = \frac{\partial E[n]}{\partial a'_k[n]} = \frac{\partial E[n]}{\partial e_k[n]} \frac{\partial e_k[n]}{\partial z_k[n]} \frac{\partial z_k[n]}{\partial a'_k[n]}. \tag{3.6}$$

Diferenciando a ambos lados de la ecuación 3.3 con respecto a $e_k[n]$ se obtiene

$$\frac{\partial E[n]}{\partial e_k[n]} = e_k[n], \tag{3.7}$$

del mismo modo, diferenciando a ambos lados de la ecuación 3.4 respecto de $z_k[n]$ se tiene

$$\frac{\partial e_k[n]}{\partial z_k[n]} = 1, \tag{3.8}$$

y finalmente, diferenciando a ambos lados de la ecuación 3.5 con respecto a $a'_k[n]$ se obtiene

$$\frac{\partial z_k[n]}{\partial a'_k[n]} = \frac{\partial f_2(a'_k[n])}{\partial a'_k[n]}. \tag{3.9}$$

Luego de remplazar las ecuaciones 3.7, 3.8 y 3.9 en la ecuación 3.6, se define el gradiente local de error para una neurona k en la capa de salida como

$$\delta'_k[n] = e_k[n]\frac{\partial f_2(a'_k[n])}{\partial a'_k[n]}. \tag{3.10}$$

3.2.2 Caso 2: Neurona en capa oculta

Considérese ahora, la situación presentada en la figura 3.2, la cual describe una neurona j perteneciente a la capa oculta de una red MLP, nuevamente el bias $b_j = w_{j,0}$ se concidera como un peso adicional cuya entrada asociada es $x_0 = -1$. Como no se especifica ninguna respuesta deseada para esta neurona, el gradiente local de error debe determinarse recursivamente, a partir de las señales de error de todas las neuronas de la capa siguiente a las cuales está conectada. Así, en conformidad con la definición de gradiente local de error, se puede redefinir la ecuación 3.1, para una neurona j perteneciente a la capa oculta como

$$\begin{aligned}
\delta_j[n] &= \frac{\partial E[n]}{\partial a_j[n]} \\
&= \frac{\partial E[n]}{\partial y_j[n]}\frac{\partial y_j[n]}{\partial a_j[n]} \\
&= \frac{\partial E[n]}{\partial y_j[n]}\frac{\partial f_1(a_j[n])}{\partial a_j[n]},
\end{aligned} \tag{3.11}$$

donde $f_1(.)$ en la tercera línea, corresponde a la función de activación de la neurona j (Haykin 1998).

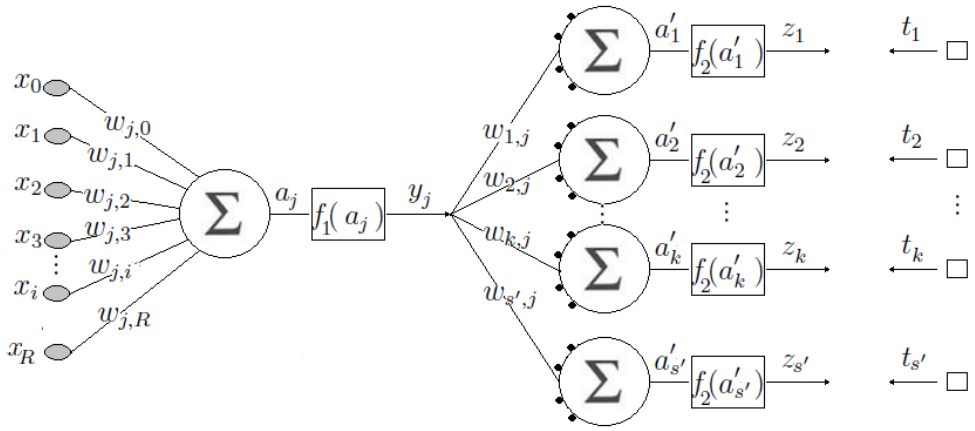


Figura 3.2. Conexiones de una neurona en capa oculta

Para calcular $\partial E[n]/\partial y_j[n]$ se debe diferenciar en ambos lados de la ecuación 3.3 con respecto a $y_j[n]$ como sigue a continuación

$$\frac{\partial E[n]}{\partial y_j[n]} = \sum_k e_k[n] \frac{\partial e_k[n]}{\partial y_j[n]}. \tag{3.12}$$

Nótese que el índice k hace referencia a las neuronas de la capa de salida, esto dado que $E[n]$ se calcula sumando la energía instantánea del error de todas las neuronas en esa capa.

Haciendo uso de la regla de la cadena se puede reescribir la ecuación 3.12 como

$$\frac{\partial E[n]}{\partial y_j[n]} = \sum_k e_k[n] \frac{\partial e_k[n]}{\partial a'_k[n]} \frac{\partial a'_k[n]}{\partial y_j[n]}, \tag{3.13}$$

$\partial e_k[n]/\partial a'_k[n]$ se obtiene remplazando la ecuación 3.5 en 3.4 y diferenciando con respecto a $a'_k[n]$ como sigue

$$\frac{\partial e_k[n]}{\partial a'_k[n]} = \frac{\partial f_2(a'_k[n])}{\partial a'_k[n]}, \tag{3.14}$$

$\partial a'_k[n]/\partial y_j[n]$ puede hallarse diferenciando a ambos lados de la ecuación 3.2 respecto de $y_j[n]$

$$\frac{\partial a'_k[n]}{\partial y_j[n]} = w'_{k,j}[n-1]. \tag{3.15}$$

Al remplazar las ecuaciones 3.15 y 3.14 en la ecuación 3.13 se encuentra que

$$\frac{\partial E[n]}{\partial y_j[n]} = \sum_k e_k[n] \frac{\partial f_2(a'_k[n])}{\partial a'_k[n]} w'_{k,j}[n-1] = \sum_k \delta'_k[n] w'_{k,j}[n-1]. \tag{3.16}$$

Finalmente la expresión matemática que define el gradiente local de error para una neurona en capa oculta se obtiene al remplazar la ecuación 3.16 en la ecuación 3.12 como sigue a continuación

$$\delta_j[n] = \frac{\partial f_1(a_j[n])}{\partial a_j[n]} \sum_k \delta'_k[n] w'_{k,j}[n-1]. \tag{3.17}$$

Nótese que el gradiente local de error de una capa oculta depende siempre del gradiente local de error de la capa siguiente. Este esquema se puede usar para calcular los gradientes locales de error para arquitecturas con más de dos capas.

3.3 Planteamiento del algoritmo ARγ-GLE

El algoritmo ARγ-GLE se obtiene al realizar una serie de modificaciones al algoritmo ARγ, las cuales se sintetizan a continuación:

- Remplazar el modelo de ajuste del algoritmo ARγ, por una expresión que calcule la respuesta general de una red MLP, almacenando para cada entrada el valor de las salidas y el valor de los potenciales postsinápticos de cada capa.

- Incluir una expresión que calcule los gradientes locales de error para cada capa.

- La actualización de parámetros se realiza neurona a neurona haciendo uso de las expresiones propias del algoritmo ARγ, pero utilizando como función de error el gradiente local de error asociado a cada una.

3.3.1 Descripción de la implementación realizada

El procedimiento seguido para realizar la implementación se resume a continuación:

$1.)$ Debe partirse de una configuración aleatoria de pesos sinápticos (si se parte de pesos y bias nulos el aprendizaje no progresa).

$2.)$ Para cada patrón de aprendizaje:

$2.1.)$ Obtener la respuesta general de la red de acuerdo con la ecuación 2.3.

$2.2.)$ Calcular los gradientes locales de error $\delta'_k[n]$ y $\delta_j[n]$ para cada neurona de la red.

$2.3.)$ Calcular el incremento parcial de los pesos y bias con

$$g'[n] = \frac{\delta'_k[n] + \gamma \sum y_j[n] w'_{k,j}[n-1]}{1 + \alpha\gamma m_1 \sum y_j^2[n]}, \tag{3.18}$$

$$q'_{k,j}[n] = \frac{\gamma}{\alpha + \gamma} \left[q'_{k,j}[n-1] - \alpha g'[n] m_1 y_j[n] \right], \tag{3.19}$$

$$g[n] = \frac{\delta_j[n] + \gamma \sum x_i[n] w_{j,i}[n-1]}{1 + \alpha\gamma m_1 \sum x_i^2[n]}, \tag{3.20}$$

$$q_{j,i}[n] = \frac{\gamma}{\alpha + \gamma} \left[q_{j,i}[n-1] - \alpha g[n] m_1 x_i[n] \right], \tag{3.21}$$

donde:

$g[n], g'[n]$: Escalares auxiliares en el instante n.
$q_{j,i}[n], q'_{k,j}[n]$: Incremento parcial de los pesos y bias en el instante n.
α: Parámetro fijo de entrenamiento.
γ: Parámetro fijo de entrenamiento.
m_1: Parámetro fijo de entrenamiento.

$3.)$ Actualizar los pesos y bias con

$$w'_{k,j}[n] = w'_{k,j}[n-1] + \alpha q'_{k,j}[n], \tag{3.22}$$

$$w_{j,i}[n] = w_{j,i}[n-1] + \alpha q_{j,i}[n]. \tag{3.23}$$

3.3.2 Complejidad computacional del algoritmo ARγ-GLE

La tabla 3.1 describe la complejidad computacional del algoritmo; siendo W el número de pesos en la red, WI el número de pesos conectados directamente a las entradas, Ns el número de neuronas de salida y No el número de neuronas ocultas.

Etapa	Multiplicaciones	Adiciones	Divisiones	Evaluación de funciones	Orden de complejidad
Salida de la red	W	$W-(No+Ns)$	0	$No+Ns$	$O(W)$
Gradiente local de error	$W-WI+Ns+No$	$W-WI-No+Ns$	0	$No+Ns$	$O(W)$
Actualizaciones	$5W+3(Ns+No)$	$4W$	$No+Ns$	0	$O(W)$

Tabla 3.1. Complejidad computacional del algoritmo ARγ-GLE

El orden de complejidad del algoritmo ARγ-GLE es $O(W)$, que corresponde al mismo orden de complejidad del algoritmo BP.

3.4 Planteamiento y estudio de la aplicación.

La funcionalidad del algoritmo ARγ-GLE se valida a continuación mediante un problema clásico en el campo de las redes neuronales, que consiste en representar la función lógica XOR, se trata de un caso práctico interesante que pese a su sencillez, pone en manifiesto características muy importantes de las redes MLP, dado que involucra patrones no separables linealmente. Así entonces, se mostrará como una red MLP puede aprender automáticamente a representar la función XOR haciendo uso del algoritmo ARγ-GLE implementado en Matlab®.

Considérese una función XOR con dos entradas lógicas x_1 y x_2 y una salida lógica y, que debe ser modelada por una red MLP. Partiendo de estas características, se deberá utilizar una arquitectura MLP con dos entradas y una neurona de salida. No existe una metodología definida para determinar cuántas neuronas son necesarias en la capa oculta, este es un problema específico que no está teóricamente esclarecido (Dai y Liu 2012), simplemente se debe colocar las suficientes. Para este caso, el número de neuronas ocultas se establece en dos. Así entonces, se plantea utilizar una arquitectura 2-2-1 como la que se muestra en la figura 3.3.

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

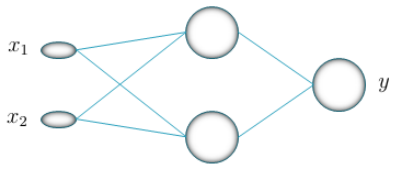


Figura 3.3. Arquitectura propuesta para el MLP y tabla de verdad de la compuerta XOR

En lo que respecta a los patrones de entrenamiento se cuenta con cuatro de ellos dados directamente por la tabla de verdad de la función XOR. Como tal, para valores lógicos iguales en las entradas corresponde un cero lógico (0) en la salida y, para valores lógicos diferentes en las entradas, corresponde un uno lógico (1) en la salida (ver figura 3.3).

Es importante destacar que una red MLP no trabaja con valores lógicos por tanto se debe asignar un valor real a cada valor lógico. Para este caso se establece que el

valor real -1 corresponde al valor lógico cero y el valor real 1 corresponde al valor lógico uno. Lo que sigue es definir una función de activación que recoja las características elegidas. La función de activación para las neuronas de la capa oculta y la neurona de la capa de salida es la siguiente sigmoidea:

$$f(a) = \frac{2}{1 + e^{-2a}} - 1, \tag{3.24}$$

que trabaja en el intervalo [-1, +1] (donde a representa el potencial postsináptico de una neurona en capa oculta o en capa de salida).

El valor de los pesos iniciales se toman aleatoriamente en el intervalo [-1.2, +1.2], los paramentos de entrenamiento del algoritmo α, γ y m_1 en principio toman los valores de 1.5, 0.12, y 11 respectivamente y el numero de épocas o ciclos de entrenamiento se fija en 50. En este caso como lo que se trata es de memorizar la tabla de la función XOR, no puede haber sobreaprendizaje.

3.4.1 Resultados

El objetivo del entrenamiento es determinar los pesos $w'_{k,j}$, $w_{j,i}$ y además los biases b'_k, b_j. Los resultados se muestran en la figura 3.4. Los datos obtenidos se probaron en la ecuación 2.3 (modo de ejecución) generando los resultados esperados, es decir que dicha red implementa la función XOR. Este resultado indica que la implementación realizada es consistente desde el punto de vista experimental. Aún no se ha determinado el papel que juegan los parámetros α, γ y m_1 en el entrenamiento, este aspecto se aclarará en capítulos siguientes.

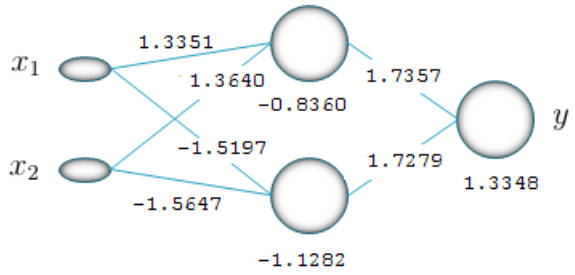


Figura 3.4. Red neuronal entrenada con el algoritmo ARγ-GLE para modelar la función XOR

Capítulo 4

Estudio Experimental del Algoritmo ARγ-GLE Basado en Simulación

Habiendo implementado y probado las modificaciones enunciadas, lo que sigue es un proceso experimental basado en simulación, cuyo propósito fundamental es investigar el comportamiento del algoritmo ARγ-GLE; esclarecer la influencia que los parámetros de entrenamiento α, γ y m_1 tienen en modo de aprendizaje y analizar los resultados obtenidos con diferentes arquitecturas de red. Este estudio gira en torno a dos problemas específicos relacionados respectivamente con reconocimiento de patrones y aproximación funcional.

4.1 Problema 1: clasificación de patrones

El problema consiste en discriminar patrones **x** pertenecientes a dos clases bidimensionales traslapadas y con distribución gaussiana (ver figuras 4.1 y 4.2). Las funciones de distribución $P_x(\mathbf{x}|\omega_1)$ y $P_x(\mathbf{x}|\omega_2)$ para las clases 1 y 2 respectivamente, están dadas por

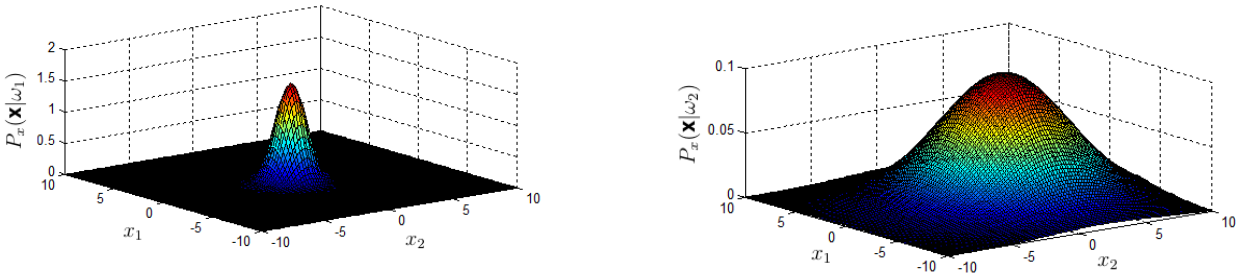
$$P_x(\mathbf{x}|\omega_1) = \frac{1}{2\pi\sigma_1} e^{\left(\frac{1}{2\sigma_1^2}||\mathbf{x}-\boldsymbol{\mu}_1||^2\right)}, \text{ con media } \boldsymbol{\mu}_1 = [0,\ 0]^T \text{ y varianza } \sigma_1 = 1 \qquad (4.1)$$

y

$$P_x(\mathbf{x}|\omega_2) = \frac{1}{2\pi\sigma_2} e^{\left(\frac{1}{2\sigma_2^2}||\mathbf{x}-\boldsymbol{\mu}_2||^2\right)}, \text{ con media } \boldsymbol{\mu}_2 = [2,\ 0]^T \text{ y varianza } \sigma_2 = 4. \qquad (4.2)$$

Ambas clases se asumen equiprobables, por cuanto las probabilidades apriori $P(\omega_1)$ y $P(\omega_2)$ para las clases 1 y 2 respectivamente son

$$P(\omega_1) = P(\omega_2) = \frac{1}{2}. \tag{4.3}$$



a) b)

Figura 4.1. Funciones de probabilidad: a) $P_x(\mathbf{x}|\omega_1)$ b) $P_x(\mathbf{x}|\omega_2)$

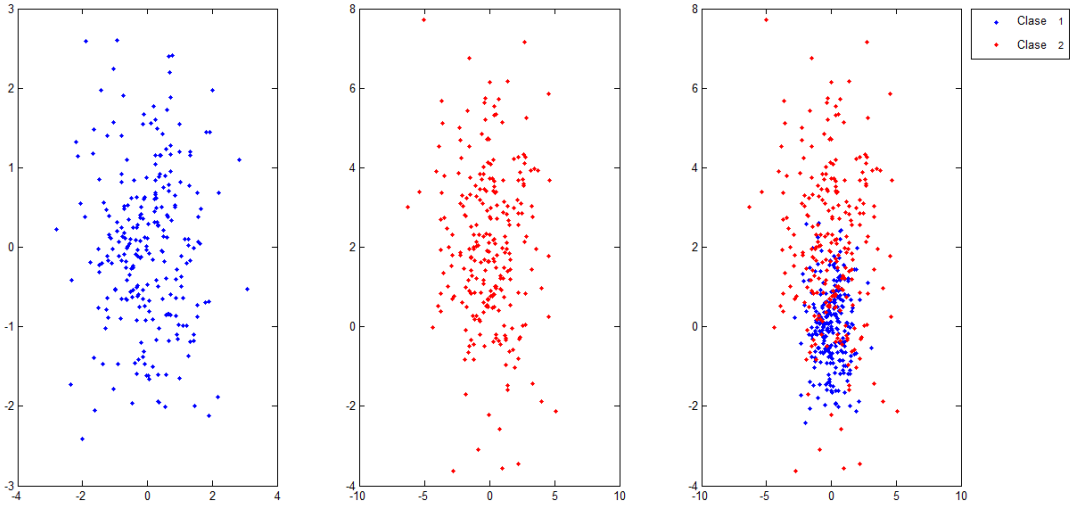


Figura 4.2. Diagrama de puntos para 250 muestras de la clase ω_1 y 250 muestras para la clase ω_2

Este problema y la metodología seguida en esta sección para el análisis del algoritmo ARγ-GLE se toman de Haykin (1998) en donde se presenta un estudio similar para el algoritmo BP con momento.

4.1.1 Determinación teórica de la probabilidad de correcta clasificación

Si se asume que los costos para correcta clasificación son cero y que los costos para errores de clasificación son iguales. Entonces, de acuerdo al criterio de Bayes para mínimo error (en el anexo A se presentan los fundamentos teóricos del criterio de Bayes), se tiene que **x** pertenece a la clase 1 si

$$l_r = \frac{P(\mathbf{x}|\omega_1)}{P(\mathbf{x}|\omega_2)} > \frac{P(\omega_1)}{P(\omega_2)}, \text{ cualquier otro caso implica que } \mathbf{x} \in \omega_2, \tag{4.4}$$

l_r es el radio de vecindad. El cual, considerando las condiciones establecidas, queda determinado como

$$l_r = \frac{\sigma_2^2}{\sigma_1^2} e^{\left(-\frac{1}{2\sigma_1^2}||\mathbf{x}-\boldsymbol{\mu}_1||^2+\frac{1}{2\sigma_2^2}||\mathbf{x}-\boldsymbol{\mu}_2||^2\right)}, \tag{4.5}$$

por tanto, la frontera de decisión optima se expresa como sigue

$$l_r = \frac{\sigma_2^2}{\sigma_1^2} e^{\left(-\frac{1}{2\sigma_1^2}||\mathbf{x}-\boldsymbol{\mu}_1||^2+\frac{1}{2\sigma_2^2}||\mathbf{x}-\boldsymbol{\mu}_2||^2\right)} = 1 \tag{4.6}$$

o equivalentemente

$$\left(-\frac{1}{2\sigma_1^2}||\mathbf{x}-\boldsymbol{\mu}_1||^2+\frac{1}{2\sigma_2^2}||\mathbf{x}-\boldsymbol{\mu}_2||^2\right) = 4 \ln\left(\frac{\sigma_1}{\sigma_2}\right). \tag{4.7}$$

La ecuación 4.7 tiene la forma característica de un circulo con centro $\mathbf{x}_c$ y radio r de tal forma que

$$||\mathbf{x}-\mathbf{x}_c||^2 = r^2, \tag{4.8}$$

donde

$$\mathbf{x}_c = \frac{\sigma_2^2\boldsymbol{\mu}_1 - \sigma_1^2\boldsymbol{\mu}_2}{\sigma_2^2-\sigma_1^2} \tag{4.9}$$

y

$$r^2 = \frac{\sigma_2^2\sigma_1^2}{\sigma_2^2-\sigma_1^2}\left[\frac{||\boldsymbol{\mu}_1-\boldsymbol{\mu}_2||^2}{\sigma_2^2-\sigma_1^2} + 4\ln\left(\frac{\sigma_1}{\sigma_2}\right)\right]. \tag{4.10}$$

Concretamente para el experimento, se tiene una frontera de decisión circular con centro en

$$\mathbf{x}_c = \begin{bmatrix} -2/3 \\ 0 \end{bmatrix}$$

y radio

$$r \simeq 2.34.$$

Con todo esto, se puede calcular la probabilidad de error $P(error)$ como

$$P(error) = P(\omega_1)P(error|\omega_1) + P(\omega_1)P(error|\omega_2), \tag{4.11}$$

donde

$$P(error|\omega_1) = \int_{C[\Omega_1]} P(\mathbf{x}|\omega_1)d\mathbf{x} \tag{4.12}$$

y

$$P(error|\omega_2) = \int_{\Omega_1} P(\mathbf{x}|\omega_2)d\mathbf{x}, \tag{4.13}$$

siendo Ω_1 la región interna del circulo de decisión y $C[\Omega_1]$ su complemento.

Dadas las condiciones del problema que se está tratando, las integrales 4.12 y 4.13 se pueden evaluar numéricamente, encontrándose que

$$P(error|\omega_1) \simeq 0.1056$$

y

$$P(error|\omega_2) \simeq 0.2642.$$

Finalmente, una vez conocidos los valores de $P(error|\omega_1)$ y $P(error|\omega_2)$, y teniendo en cuenta que las condiciones del problema establecen que $P(\omega_1) = P(\omega_2) = 1/2$, se puede calcular la probabilidad de error de clasificación $P(error)$ haciendo uso de la ecuación 4.11 con lo que se obtiene

$$P(error) \simeq 0.1849$$

y equivalentemente una probabilidad de correcta clasificación P_c

$$P_c = 1 - P(error) \simeq 0.8151.$$

4.1.2 Determinación de la arquitectura de red

Partiendo de los resultados encontrados en la sección 4.1.1, el objetivo en estas simulaciones iniciales, es determinar una arquitectura adecuada para solucionar el problema. A continuación se presenta el criterio para hacer la selección.

La arquitectura deberá tener un rendimiento mínimo del 80.51 $\%$ en la clasificación, esto es, un 1 $\%$ menos en relación al rendimiento de un clasificador de Bayes. Cumplido este requisito se seleccionara aquella con el menor número de neuronas.

Para realizar las pruebas, se generaron dos conjuntos de datos, con funciones de distribución $P_x(\mathbf{x}|\omega_1)$ y $P_x(\mathbf{x}|\omega_2)$, a partir de los cuales se conformaron conjuntos de entrenamiento de 500, 2000 y 8000 patrones. En cada conjunto de entrenamiento el 50 $\%$ de los datos son de la clase 1 y el 50 $\%$ restante son de la clase 2.

En las pruebas se hace uso de redes MLP con una capa oculta. Así entonces, como cada patrón de entrenamiento se compone de dos entradas y una salida deseada (las salidas deseadas son valores 1 o -1 para las clases 1 o 2 respectivamente), las arquitecturas utilizadas son de la forma 2-No-1, siendo No el número de neuronas ocultas. En lo que respecta a las funciones de activación, todas son de tipo sigmoidal.

La tabla 4.1 muestra los resultados obtenidos con dos neuronas ocultas, la tabla 4.2 con tres neuronas ocultas y la tabla 4.3 con cuatro neuronas ocultas. Para evitar problemas asociados con variaciones en los conjuntos de entrenamiento, el número de épocas (o ciclos de entrenamiento) se elige de tal forma que el conjunto de ejemplos presentados en cada sesión de entrenamiento sea constante. En estas primeras pruebas los parámetros de entrenamiento (α, γ y m_1) se escogen arbitrariamente.

Simulaciones	**Número de patrones**	**Número de épocas**	**Probabilidad de correcta clasificación**
1	500	320	80.8688%
2	2000	80	80.9031%
3	8000	20	80.9688%

Tabla 4.1. Resultados de simulación para dos neuronas ocultas y con parámetros de entrenamiento $m_1 = 0.01$, $\alpha = 0.1$ y $\gamma = 8$

Simulaciones	Número de patrones	Número de épocas	Probabilidad de correcta clasificación
1	500	320	80.7531%
2	2000	80	80.8812%
3	8000	20	81.0187%

Tabla 4.2. Resultados de simulación para tres neuronas ocultas y con parámetros de entrenamiento $m_1 = 0.01$, $\alpha = 0.1$ y $\gamma = 8$

Simulaciones	Número de patrones	Número de épocas	Probabilidad de correcta clasificación
1	500	320	80.7938%
2	2000	80	80.8375%
3	8000	20	81.0625%

Tabla 4.3. Resultados de simulación para cuatro neuronas ocultas y con parámetros de entrenamiento $m_1 = 0.01$, $\alpha = 0.1$ y $\gamma = 8$

La probabilidad de correcta clasificación, se determina en modo de ejecución, aplicando como entradas una cantidad N de muestras que no hayan sido usadas en el entrenamiento, posteriormente se determina el porcentaje de aciertos, el cual puede considerarse como un estimativo de la probabilidad de correcta clasificación. En Haykin (1998) se determina experimentalmente que cuando $N > 32000$, el estimativo hallado tiene una certidumbre superior al 99%. Como tal se emplearon 32000 muestras para estimar la probabilidad de correcta clasificación.

El rendimiento de clasificación para una red MLP con dos neuronas ocultas presentado en la tabla 4.1 cumple con el criterio estipulado inicialmente. Nótese también en las tablas 4.2 y 4.3 que para el caso de tres y cuatro neuronas en la capa oculta, la probabilidad de correcta clasificación no varía significativamente, con lo cual se determina que una arquitectura 2-2-1 es adecuada para resolver el problema de clasificación propuesto.

4.1.3 Efecto general de los parámetros α, γ y m_1

Hasta el momento nada se ha hablado sobre el valor que deben tomar los parámetros de entrenamiento α, γ y m_1 para alcanzar un rendimiento adecuado. Por tratarse de un algoritmo que se basa en los principios que rigen la actualización de parámetros en el algoritmo ARγ, una condición necesaria pero no suficiente, es que dichos valores deberán ser reales positivos mayores que cero ($\alpha > 0$, $\gamma > 0$ y $m_1 > 0$).

Partiendo de lo anterior, en lo que sigue se presentan una serie de experiencias que buscan esclarecer el efecto que tienen los parámetros α, γ y m_1 en la curva de entrenamiento. Los resultados obtenidos representan la base experimental utilizada para seleccionar una configuración adecuada de estos parámetros en experiencias posteriores.

La curva de entrenamiento brinda información detallada del error cometido por la red en el proceso de entrenamiento, el cual se mide en términos del error cuadrático medio (ecuación 2.19) para cada época transcurrida. El error cuadrático medio, es el criterio matemático que se minimiza durante el entrenamiento, cabe destacar que un mínimo error de entrenamiento no implica necesariamente una mejor generalización.

Las simulaciones que se presentan a continuación, se basan en una característica fundamental del algoritmo ARγ, relacionada con un parámetro implícito del mismo llamado H ($H = \alpha\gamma m_1$), el cual está directamente relacionado con el error de desajuste. En base a lo anterior, resulta natural estudiar el efecto que H tiene en la curva de entrenamiento del algoritmo ARγ-GLE.

Teniendo en cuenta que el parámetro H está en función de los parámetros α, γ y m_1, existen múltiples configuraciones de los parámetros de entrenamiento que conllevan a un mismo valor de H, por esto debe estudiarse el efecto de H para distintas configuraciones de α, γ y m_1.

En este estudio se toman como referencia los valores óptimos para mínimo desajuste de la versión estándar del algoritmo ARγ estipulados en la tabla 2.3. En esta tabla se proponen seis configuraciones diferentes para α, γ y m_1. Para obtener los distintos valores de H el procedimiento a seguir consiste en tomar una de las seis configuraciones y variar uno de los tres parámetros de entrenamiento, manteniendo los otros dos fijos. Este procedimiento se repite para los tres parámetros en cada configuración respectiva. El valor de los parámetros así como los resultados de las simulaciones se presentan en las figuras 4.3, 4.4 y 4.5

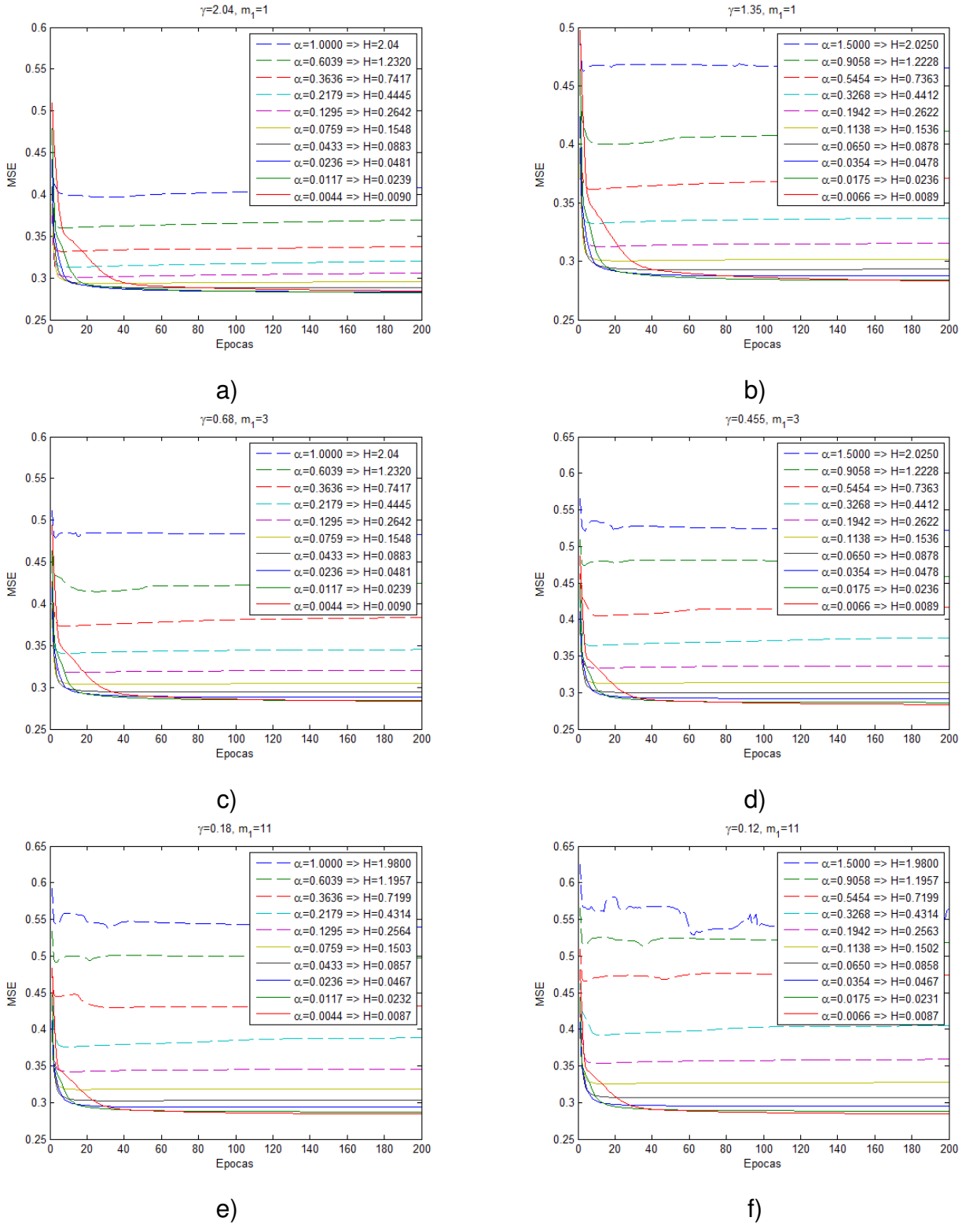


Figura 4.3. Curvas de entrenamiento para diferentes valores de H con variación de α en distintas configuraciones de γ y m_1 fijas

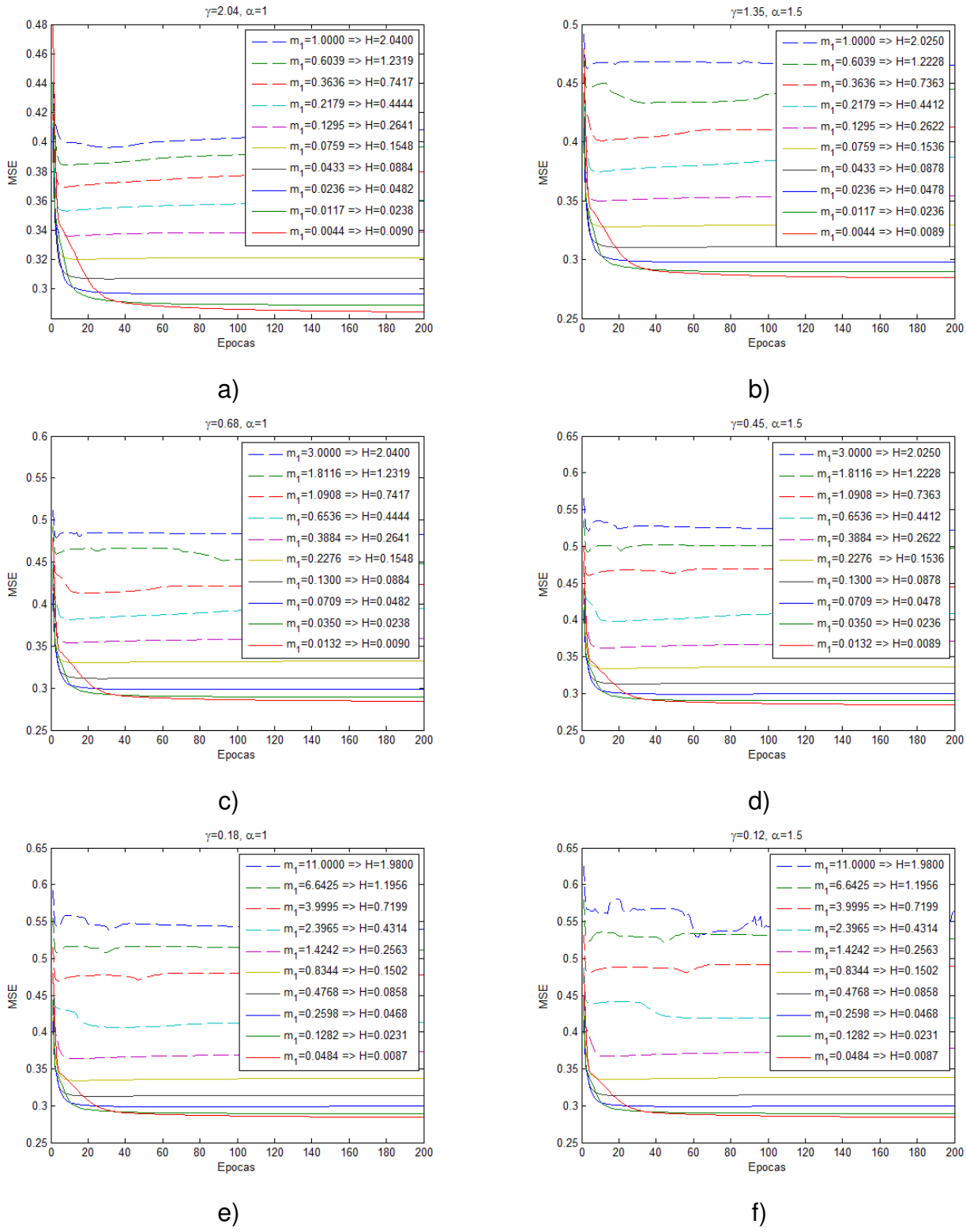


Figura 4.4. Curvas de entrenamiento para diferentes valores de H con variación de m_1 en distintas configuraciones de γ y α fijas

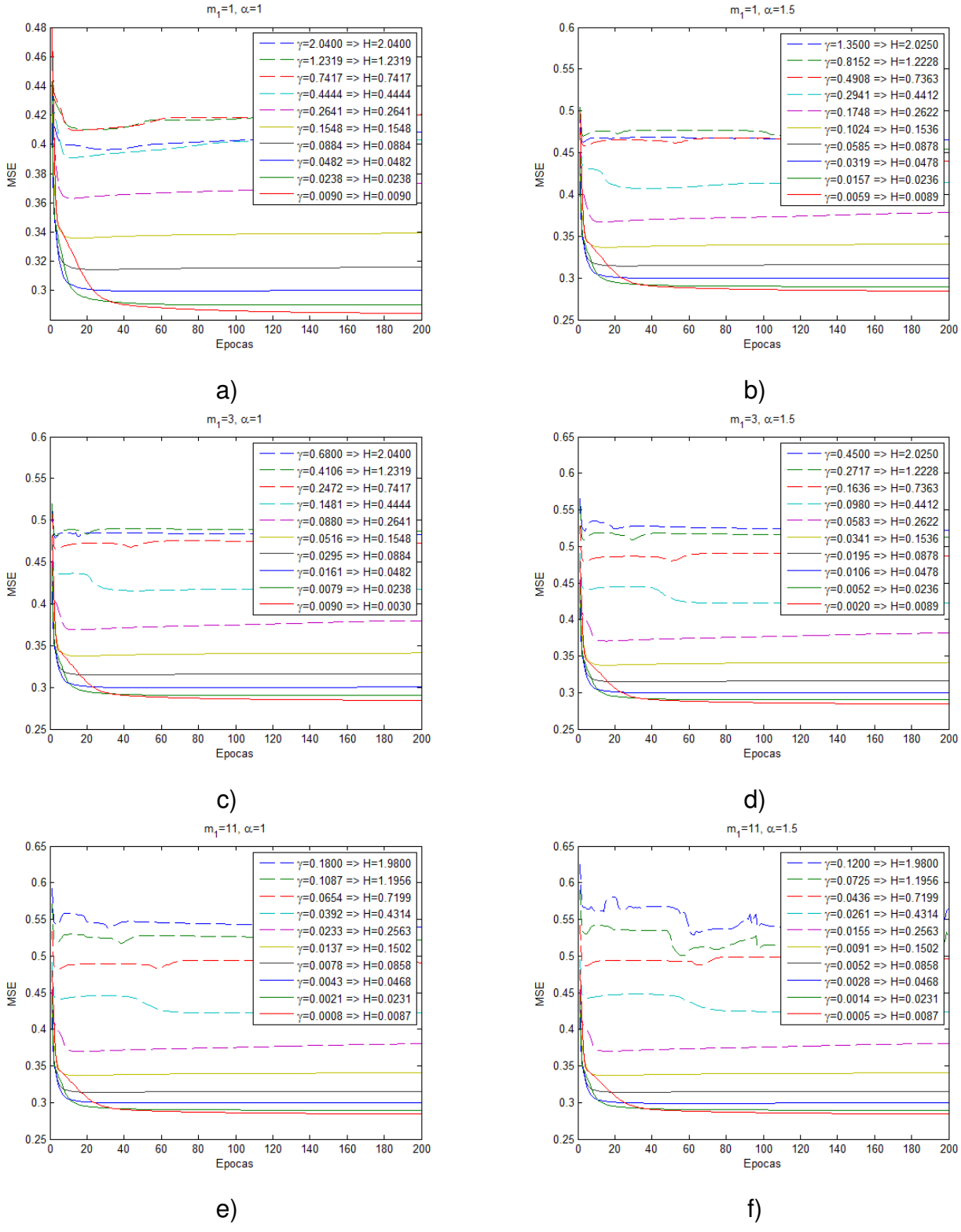


Figura 4.5. Curvas de entrenamiento para diferentes valores de H con variación de γ en distintas configuraciones de α y m_1 fijas

Todas las experiencias partieron desde los mismos pesos sinápticos y utilizando el mismo conjunto de entrenamiento con 500 patrones, de tal forma que los resultados pueden ser comparados directamente. Se realizaron 200 épocas de entrenamiento en todas las simulaciones.

Los resultados ilustran sobre las características de la curva de entrenamiento en función del parámetro H, y destacan la influencia del parámetro α sobre los otros dos (γ y m_1) en el mismo proceso. Nótese en las figuras 4.3, 4.4 y 4.5 que aunque un mismo valor de H puede llevar a resultados diferentes, las curvas de entrenamiento sugieren las siguientes tendencias.

- Los valores de H próximos a 0.2 logran un compromiso aceptable en términos de velocidad de convergencia y cota de error.

- Para un mínimo local dado, los valores de H menores a 0.2, permiten alcanzar cotas de error más pequeñas entre más próximos a cero sean dichos valores, pero sacrificando a su vez velocidad de convergencia.

- Valores de H mayores a 0.2 no ofrecen aumentos significativos en términos de velocidad de convergencia, para estos valores los efectos de H son directamente proporcionales a la cota de error alcanzada.

Un aspecto importante a resaltar, es que en promedio la cota de error más baja para H próximo a 0.2 se logró variando el parámetro α (ver figura 4.3), seguido por m_1 y γ respectivamente. Lo anterior sugiere que el parámetro α tiene efectos más significativos en relación a los otros dos parámetros, en cuanto a velocidad de convergencia y cota de error alcanzada para un mínimo local dado, con lo cual se establece experimentalmente el orden lógico en el que se deberían seleccionar los parámetros.

4.1.4 Variación de γ y m_1 con α fijo

Estas simulaciones tienen como objetivo determinar para un α dado los valores que deben asumir γ y m_1 para minimizar el error cuadrático medio. Así entonces, utilizando un perceptrón multicapa de dos neuronas ocultas se realizan simulaciones usando combinaciones de los parámetros $\gamma \in$ [0.3333, 0.76, 1.4, 1.6133, 2.04] y $m_1 \in$ [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11], dejando fijo el valor de α en 0.1. Los resultados de las simulaciones se muestran en la figura 4.6.

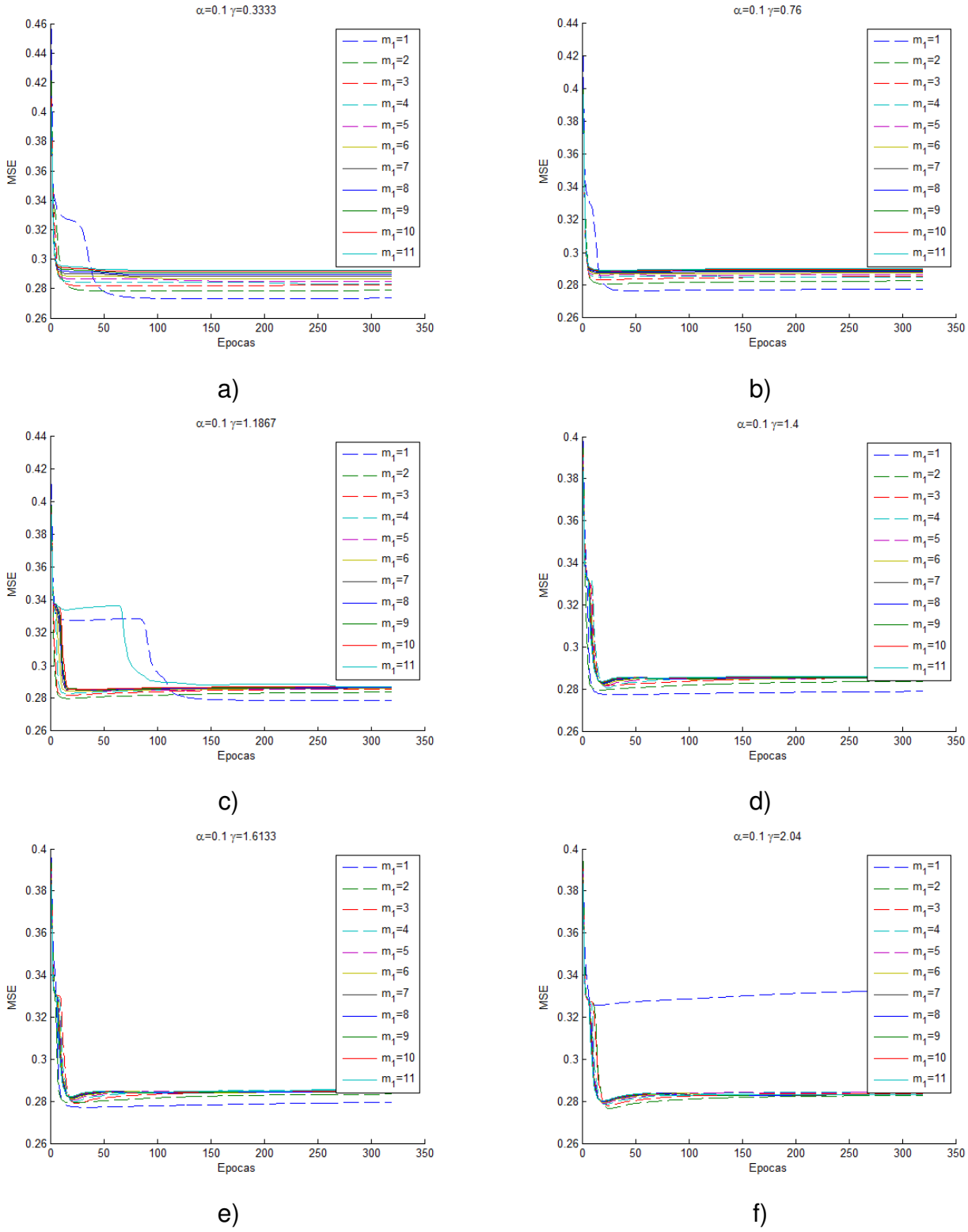


Figura 4.6. Curvas de entrenamiento para α=0.1, variando m_1 para distintos valores de γ

Los resultados indican que valores pequeños de m_1 permiten alcanzar errores de entrenamiento mas pequeños. Sin embargo, debe tenerse mucho cuidado en la selección de este parámetro, puesto que para un valor γ dado, los valores demasiado pequeños de m_1 pueden conllevar a que la búsqueda pierda capacidad de fuga, es decir que se aumente la posibilidad de que el entrenamiento se estanque en los primeros mínimos locales renunciando a inspeccionar otros mínimos mas profundos (ver figura 4.5f) o en su defecto retardar considerablemente la convergencia hacia un mínimo local apropiado como en el caso de la figura 4.5c.

En la tabla 4.4 se presentan las configuraciones que ofrecen mejores resultados.

α	γ	m_1	**MSE**
0.1	1.4	1	0.2773
0.1	1.6133	1	0.2771
0.1	0.5467	2	0.2805
0.1	0.76	2	0.2805

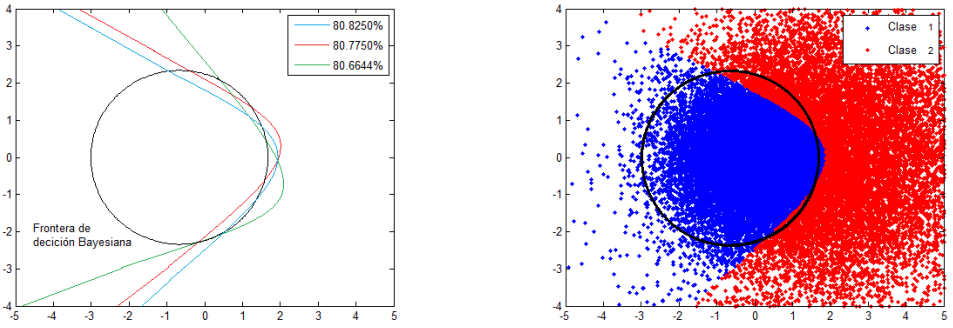
Tabla 4.4. Configuraciones de entrenamiento que obtuvieron los mejores resultados

4.1.5 Evaluación de resultados

Para evaluar los resultados obtenidos se utilizara la configuración de parámetros α=0.1, γ=1.6133 y m_1=1 (tomada de la tabla 4.4). La red obtenida se pone en modo de ejecución con el propósito de determinar su frontera de decisión y la probabilidad de correcta clasificación. Como la función de aprendizaje obtenida es puramente estocástica dado que se utilizan conjuntos de entrenamiento finitos, las mediciones de rendimiento se obtendrán promediando los resultados de 20 redes independientemente entrenadas. Cada conjunto de entrenamiento consta de 1000 patrones aleatoriamente ordenados con igual probabilidad de ocurrencia para las clases 1 y 2. Se utilizan 320 épocas o ciclos de entrenamiento y, para calcular la probabilidad de correcta clasificación, se utilizan 32000 patrones al igual que en la sección 4.1.2.

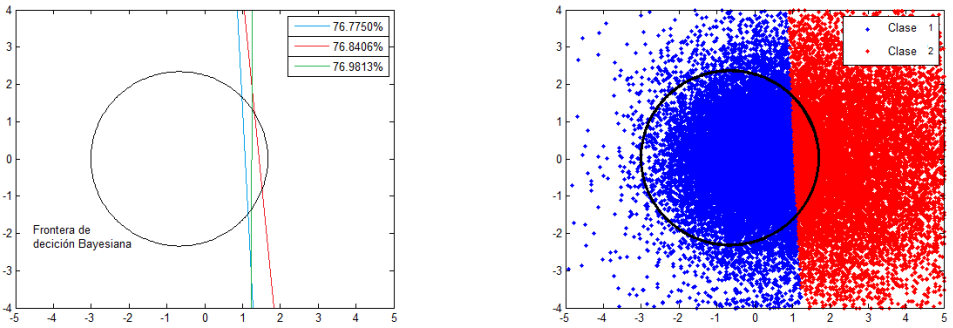
Una vez realizadas las 20 sesiones de entrenamiento se calculó la probabilidad de correcta clasificación. y en base a esta medida de rendimiento se determinaron las tres mejores fronteras de decisión. En la figura 4.7a se presentan las tres mejores fronteras de decisión resultantes y en la figura 4.7b se muestra el diagrama de clasificación para

la red que obtuvo la mejor probabilidad de correcta clasificación. Así mismo también se obtuvieron las tres peores fronteras de decisión y el diagrama de clasificación para la red con menor probabilidad de correcta clasificación (ver figura 4.8). En dichas gráficas se incluye como referencia la frontera circular de decisión bayesina. Nótese que las fronteras de decisión con mejores rendimientos tienen forma convexa, lo cual deja ver la capacidad que tiene la combinación *Arquitectura MLP + Algoritmo ARγ-GLE* para construir fronteras de decisión no lineales.



a) b)

Figura 4.7. Mejores fronteras de decisión obtenidas: a) Fronteras de decisión con probabilidades de clasificación de 80.82%, 80.77% y 80.66% b) Diagrama de clasificación para la mejor frontera de decisión obtenida



a) b)

Figura 4.8. Peores fronteras de decisión obtenidas: a) Fronteras de decisión con probabilidades de clasificación de 76.77%, 76.84% y 76.98% b) Diagrama de clasificación para la peor frontera de decisión obtenida

La tabla 4.5 muestra estadísticas relacionadas con las 20 sesiones de entrenami-

ento. Las medidas de rendimiento son la probabilidad de correcta clasificación y el error cuadrático medio para las cuales se indica el promedio y la desviación estándar obtenidas en las 20 sesiones de entrenamiento. Cabe recordar que el limite teórico para la probabilidad de correcta clasificación definido en base al criterio de Bayes es 81.51 $\%$.

Rendimiento	Media	Desviación estándar
Probabilidad de correcta clasificación	79.8530%	1.1374%
Error cuadrático medio (MSE)	0.2850	0.0129

Tabla 4.5. Estadísticas de rendimiento para 20 sesiones de entrenamiento

4.2 Problema 2: aproximación de funciones

El objetivo es determinar mediante validación cruzada el grado de generalización alcanzado con múltiples arquitecturas de red, cuando se entrenan para determinar una función, de tal forma que sea lo más cercana posible a otra. La función a modelar es la de Saito-Nakano, la cual es una version modificada de la propuesta por Sutton y Matheus en (Sutton and Matheus, 1991). Inicialmente la función era

$$y = 2 + 3x_1x_2 + 4x_3x_4x_5. \tag{4.14}$$

La modificación realizada por Saito y Nakano es

$$y = 2 + 3x_1^{-1}x_2^3 + 4x_3x_4^{1/2}x_5^{-1/3} \tag{4.15}$$

4.2.1 Condiciones experimentales

- Para estas simulaciones, se generaron 500 patrones de entrenamiento, de los cuales 300 se usan para entrenamiento, 100 para generalización y 100 para prueba. Los valores de las variables independientes se han elegido aleatoriamente en el intervalo [0, +1].

- las salidas deseadas se obtienen aplicando directamente la ecuación de Saito y Nakano.

- Los parámetros de entrenamiento son $\alpha = 0.1$, $\gamma = 0.1$ y $m_1 = 0.1$ para todas las experiencias.

- Se utilizan funciones de activación sigmoidal en las capas ocultas y funciones de activación lineal (identidad) en las capas de salida.

4.2.2 Determinación de arquitecturas de red

En este experimento se prueban varias arquitecturas MLP de una y dos capas ocultas aplicando la técnica de validación cruzada. El error de validación (ecuación 2.19) se determina cada que transcurre una época de entrenamiento, guardando en cada vez la configuración de pesos más adecuada, es decir aquella que vaya obteniendo el menor error de validación. El número de épocas utilizadas para el entrenamiento en todas las experiencias es 1200.

Las pruebas inician utilizando arquitecturas MLP con una capa oculta. Los resultados obtenidos se muestran en la tabla 4.6.

Arquitectura	Error de validación
5-2-1	15.2319
5-3-1	16.6260
5-5-1	18.3466
5-8-1	26.2607
5-12-1	17.9720
5-18-1	24.7685
5-32-1	90.5154
5-50-1	69.1988
5-100-1	142.8098

Tabla 4.6. Resultados de validación para arquitecturas de una capa oculta

Aunque teóricamente está demostrado que para aproximar cualquier función continua en un intervalo basta con utilizar modelos MLP de una capa oculta (Mitchel 1997), resulta interesante estudiar el error de validación con arquitecturas de más capas ocultas, por lo que en las siguientes pruebas se utilizan arquitecturas con dos

capas ocultas. Las arquitecturas y los resultados obtenidos se presentan en la tabla 4.7.

Arquitectura	Error de validación
5-3-2-1	41.7988
5-5-3-1	66.7135
5-8-4-1	42.0799
5-12-6-1	57.2675
5-18-14-1	93.9097
5-32-18-1	114.2566
5-65-35-1	140.9689
5-100-50-1	155.0890
5-150-75-1	163.3494

Tabla 4.7. Resultados de validación para arquitecturas de dos capas ocultas

En base a las tablas 4.6 y 4.7, los mejores resultados en términos de generalización se obtienen con la arquitectura 5-2-1, y en este mismo sentido, los resultados más pobres se obtienen con la arquitectura 5-150-75-1. En adelante, las pruebas realizadas se enfocan en estas dos arquitecturas.

4.2.3 Resultados y discusión

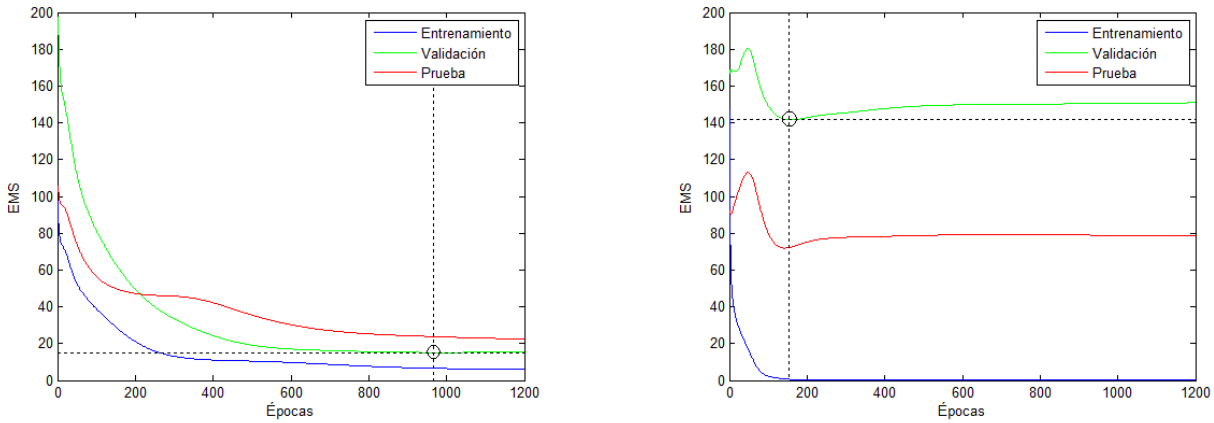
Dada la condición estocástica de las funciones de aprendizaje obtenidas, que se asocia al carácter finito del conjunto de entrenamiento, se realizaron mediciones promediando los resultados de 20 sesiones de entrenamiento para ambas arquitecturas, nuevamente las condiciones y los parámetros de entrenamiento fueron las mismas estipuladas en la sección 4.2.1. Los resultados se presentan en la tabla 4.8.

Arquitectura	MSE entrenamiento				MSE validación			
	Media	Desviación	Mejor	Peor	Media	Desviación	Mejor	Peor
5-2-1	6.1718	0.2785	5.9497	7.2174	15.2556	0.0751	15.1708	15.5529
5-150-75-1	5.43 E-4	3.36 E-4	4.40 E-5	0.0012	162.6697	12.1800	141.569	191.1421

Tabla 4.8. Estadísticas de rendimiento para 20 sesiones de entrenamiento con las arquitecturas seleccionadas

En la tabla 4.8, se presenta un hecho experimental relacionado con sobreaprendizaje. Ambas arquitecturas se entrenaron con el mismo número de épocas, pero la arquitectura 5-150-75-1 logró alcanzar en el entrenamiento una cota de error mucho menor en relación a la arquitectura 5-2-1. Sin embargo, la eficacia general del sistema o generalización para la arquitectura 5-150-75-1 es mucho menor dado que su error de validación es más alto.

La explicación a este fenómeno es la siguiente. Nótese en la figura 4.9 que en un inicio las arquitecturas se adaptan progresivamente al conjunto de entrenamiento, acomodándose al problema y mejorando la generalización. Pero en un momento dado (época 174), la arquitectura 5-150-75-1 se empieza a ajustar demasiado a las particularidades de los patrones empleados en el entrenamiento, por lo que crece el error que comete ante patrones nuevos. Para la arquitectura 5-2-1 el sobreajuste comienza en la época 967, sin embargo, la arquitectura 5-150-75-1 por ser más grande, tiene más memoria asociativa, con lo que, puede incluso, simplemente memorizar dichas particularidades, y alcanzar errores de entrenamiento más bajos sin captar la verdadera tendencia del problema.

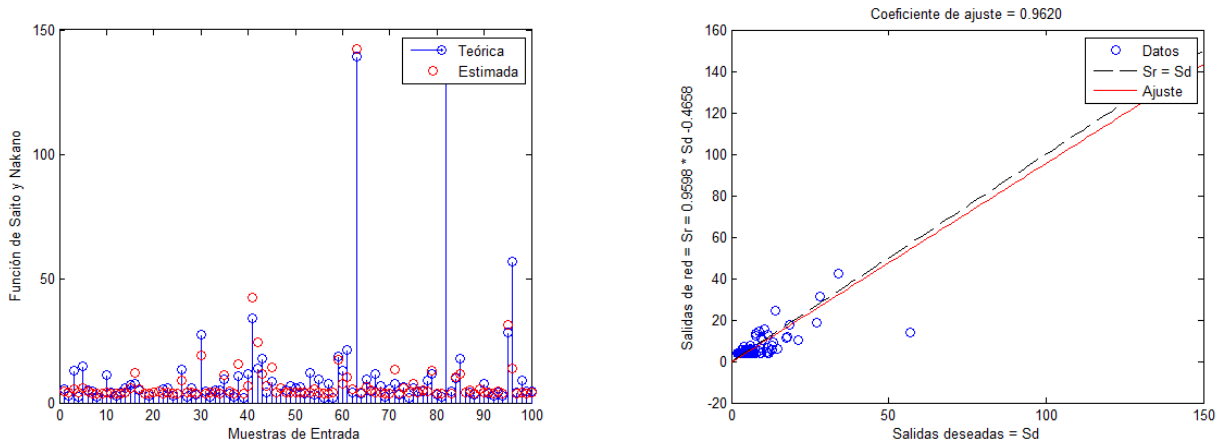


a) b)

Figura 4.9. Curvas de entrenamiento, validación y prueba con arquitectura 5-2-1: a) Mejor b) Peor

Los datos que miden objetivamente la eficacia de la red, son los que pertenecen al conjunto de prueba, cuyas respuestas se muestran en las figuras 4.10 y 4.11 para las arquitecturas 5-2-1 y 5-150-75 1 respectivamente. Los diagramas de muestras (figuras 4.10a y 4.11a), indican la respuesta teórica y estimada ante los datos de prueba;

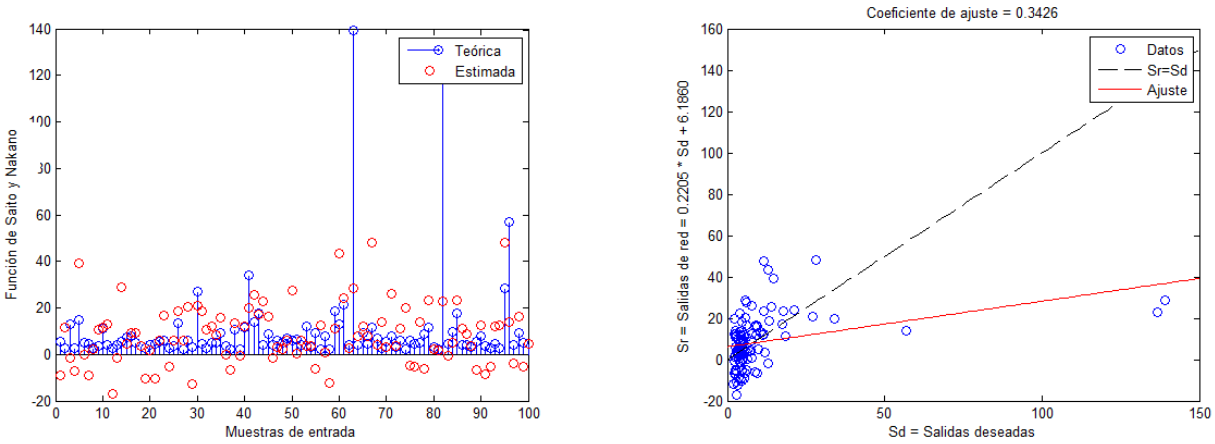
es evidente que los datos estimados con la arquitectura 5-2-1, se acercan más a los proporcionados por el modelo teórico. Esta apreciación se corrobora con los diagramas de regresión (figuras 4.10b y 4.11b), indicando mediante el coeficiente de ajuste (o coeficiente de correlación), una medida de la confiabilidad de la estimación proporcionada por cada modelo, de tal manera que entre más cercano a 1 sea el coeficiente de ajuste, más confiable será la estimación realizada por la red; para un ajuste ideal los datos deberían quedar sobre la línea punteada, en tal caso coincidirían también con la recta de ajuste (línea roja).



a) b)

Figura 4.10. Respuestas del mejor modelo estimado (arquitectura 5-2-1) usando datos de prueba: a) Diagrama de muestras b) Diagrama de regresión

Los resultados indican que el algoritmo ARγ-GLE tiene capacidades bastante poderosas en lo que respecta a aproximación de funciones mediante el entrenamiento de redes MLP. Sin embargo, también evidencian sus posibilidades de incurrir en problemas de sobreaprendizaje al igual que otras técnicas si no se controlan algunos aspectos.



a) b)

Figura 4.11. Respuestas del peor modelo estimado (arquitectura 5-150-75-1) usando datos de prueba: a) Diagrama de muestras b) Diagrama de regresión

Capítulo 5

Análisis Comparativo del Algoritmo Propuesto con el Algoritmo BP

5.1 Descripción de los datos utilizados

El análisis comparativo que se presenta a continuación se basa en tres tipos de datos usados comúnmente en problemas de "benchmarking". Los dos primeros son casos de aproximación funcional y el último es un caso de clasificación. A excepción del primer conjunto de datos (*función coseno*), para el cual las muestras se pueden obtener directamente, los otros dos conjuntos están disponibles en *UCI Machine Learning Repository* [1]. A continuación se describen los datos utilizados.

5.1.1 Función coseno

Inicialmente se compara el rendimiento de ambos algoritmos mediante un problema de regresión que consiste en aproximar la función coseno $y = cos(x)$. Con el propósito de mantener la señal de salida en el rango [0, +1], la función es cambiada por $y = (cos(2x) + 1)/2$. Los patrones de entrenamiento se obtienen evaluando la función $y = (cos(2x) + 1)/2$ en 315 puntos escogidos aleatoriamente en el intervalo [0, +π]. Los datos obtenidos se dividen en tres conjuntos de 315 (60$\%$), 63 (20$\%$) y 63 (20$\%$) para entrenamiento validación y prueba respectivamente.

5.1.2 Precio de vivienda (*House pricing dataset*)

Se cuenta con 506 patrones de entrenamiento, cuyas entradas son 13 características asociadas a un vecindario, en función a las cuales se puede estimar el valor promedio

[1] http://mlearn.ics.uci.edu/MLRepository.html (Acceso en Enero de 2014)

de una vivienda ocupada por sus propietarios. Los atributos de vecindario se relacionan respectivamente con.

1. Tasa de crimen per cápita
2. Proporción de área residencial.
3. Proporción de negocios no minoristas.
4. Presencia de ríos en la zona.
5. Concentración de óxidos nitrosos.
6. Promedio de habitaciones por vivienda.
7. Proporción de casas ocupadas por sus propietarios.
8. Distancia a centros de empleo.
9. Índice de accesibilidad.
10. Valor de los impuestos.
11. Tasa maestros/alumnos en la zona.
12. Proporción de grupos raciales.
13. Porcentaje de la población de estrato bajo.

Se utilizan 304 (60%) patrones para entrenamiento, 101 (20%) para validación y 101 (20%) para prueba.

5.1.3 Cáncer de seno (*Brest cancer dataset*)

Estos datos se relacionan con muestras de biopsias tomadas a diferentes pacientes. Se cuenta con 699 ejemplos, de los cuales 419 (60%), 140 (20%) y 140 (20%) se utilizan para entrenamiento, validación y prueba respectivamente. Los elementos de cada vector de entrada se corresponden con nueve características, a partir de las cuales se busca determinar la condición benigna o maligna de un tumor mamario. Estas características son las siguientes.

1. Espesor de la masa.
2. Uniformidad del tamaño de la célula.
3. Uniformidad del estado de la célula.
4. Adherencia marginal.
5. Epitelio simple del tamaño de la célula.
6. Núcleo descubierto.
7. Cromatina blanda.
8. Nucléolo normal.
9. Mitosis.

Cada vector de salida deseada tiene dos elementos, un uno en el primer elemento, indica que el tumor es benigno y un uno en el segundo elemento indica que el tumor es maligno. No pueden haber dos elementos iguales en el vector de salida deseada (ver tabla 5.1).

Elemento 1	Elemento 2	Tipo de tumor
1	0	Benigno
0	1	Maligno
0	0	Error de clasificación
1	1	Error de clasificación

Tabla 5.1. Clasificación según las salidas deseadas

5.2 Preprocesamiento y características del entrenamiento

La actualización de parámetros en el algoritmo BP se lleva a cabo en modo secuencial. El procesamiento se realiza bajo las siguientes condiciones.

- **Escalado de datos**: primeramente se realiza un estandarizado, que consiste en transformar las variables de entrada y salida de tal modo que presenten media cero y varianza unitaria. Para esto Matlab® incluye la rutina especializada "*mapstd*". Posteriormente se hace uso de la rutina "*mapminmax*" para normalizar los datos de tal modo que tomen valores en el rango [-1, +1].

- **Parámetros de red**: para este análisis se utilizan redes de una y dos capas ocultas, y las pruebas involucran diferente número de neuronas en estas capas. Para los problemas de aproximación funcional (*función coseno* y *precio de vivienda*) se hace uso de funciones de activación sigmoidales $Tanh(.)$ en las capas ocultas y funciones de activación lineal $y(.)$ en las capas de salida; para el problema de clasificación (*cáncer de seno*) se utilizan funciones de activación sigmoidales ($Tanh(.)$) en todas las neuronas. Las expresiones matemáticas de las funciones de activación se presentan a continuación

$$Tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}, \tag{5.1}$$

$$y(a) = a, \tag{5.2}$$

donde $Tanh(a)$ es la tangente hiperbólica y $y(a)$ es la función identidad, siendo a el potencial postsináptico de una neurona cualquiera.

- **Paramentos de entrenamiento**: Se utilizan los mismos parámetros de entrenamiento en todas las neuronas de la red. Los valores que toman estos parámetros se presentan en la tabla 5.2.

Algoritmo	Parámetro	Función coseno	Precio de vivienda	Cáncer de seno
Algoritmo BP	$\hat{\alpha}$	$0.1, 0.5, 0.9$	$0.01, 0.1, 0.5$	$0.1, 0.5, 0.9$
	$\hat{\mu}$	$0, 0.5, 0.9$	$0, 0.5, 0.9$	$0, 0.5, 0.9$
Algoritmo ARγ-GLE	α	$0.1, 0.5, 0.9$	$0.01, 0.1, 0.5$	$0.1, 0.5, 0.9$
	(γ,m_1)	$(2.04, 1), (1.61, 1), (0.12, 11)$	$(2.04, 1), (1.61, 1), (0.12, 11)$	$(2.04, 1), (1.61, 1), (0.12, 11)$

Tabla 5.2. Parámetros de entrenamiento seleccionados

- **Rendimiento de la red**: los resultados de entrenamiento, validación y prueba se miden en términos del MSE (ecuación 2.19). Teniendo en cuenta que el entrenamiento se realiza con salidas deseadas a las cuales se les aplica un preprocesamiento previo, antes de calcular el MSE se debe aplicar a todas las salidas un procesamiento que revierta el preprocesamiento inicial, en Matlab® ésto se puede hacer con las mismas rutinas utilizadas inicialmente (*mapminmax* y *mapstd*).

- **Inicialización de pesos**: una heurística (de la cual se hace uso en las siguientes simulaciones) que suele dar buenos resultados consiste en elegir los pesos sinápticos iniciales aleatoriamente en el intervalo [-2.4/R, +2.4/R] siendo R el número de entradas a la neurona respectiva (Principe et al 2000).

5.3 Resultados de simulación

Los resultados que se presentan a continuación, obedecen a promedios calculados a partir de 20 sesiones de entrenamiento para cada caso. Se realizó inicialmente una serie de pruebas variando los parámetros de entrenamiento en cada algoritmo, durante las simulaciones se midió para diferentes épocas, el error de entrenamiento, validación y prueba. La elección de los parámetros de entrenamiento tiene en cuenta el hecho de que α en el algoritmo ARγ-GLE y $\hat{\alpha}$ en el algoritmo BP son parámetros análogos relacionados con la rata de aprendizaje. Para cada α y $\hat{\alpha}$ dados se probaron tres configuraciones distintas de γ y m_1 para el algoritmo ARγ-GLE y tres momentos distintos para el algoritmo BP.

Posteriormente se escogieron los parámetros con los que se obtuvieron los mejores resultados y se entrenaron diferentes arquitecturas midiendo en cada prueba el error de entrenamiento y validación. Los resultados obtenidos se organizan de la siguiente forma.

En las tablas 5.3 y 5.4 se presentan los resultados obtenidos con los datos *función coseno*, en las tablas 5.6 y 5.7 los resultados obtenidos con los datos *precio de vivienda* y en las tablas 5.9 y 5.10 los resultados obtenidos con los datos *cáncer de seno*. En estas primeras pruebas se utilizaron diferentes parámetros de entrenamiento. Nótese que los resultados para cada problema propuesto se representan con un par de tablas (para el algoritmo ARγ-GLE y el algoritmo BP respectivamente) cuyas columnas se corresponden una a una, puesto que los resultados se obtuvieron simultáneamente; partiendo de los mismos pesos y con patrones de entrenamiento presentados en el mismo orden.

En las tablas 5.5, 5.8 y 5.11 se presentan los resultados obtenidos con los datos *función coseno*, *precio de vivienda* y *cáncer de seno* respectivamente. En estas pruebas se utilizaron distintas arquitecturas, las cuales se entrenaron con la configuración de parámetros de entrenamiento que previamente obtuvo los menores errores de validación.

Épocas	Datos	$\gamma = 2.04\ m_1 = 1$			$\gamma = 1.61\ m_1 = 1$			$\gamma = 0.12\ m_1 = 11$		
		$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.9$	$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.9$	$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.9$
300	Entrenamiento	2,21E-05	2,39E-03	3,54E-05	2,05E-05	2,41E-05	2,93E-03	1,87E-05	3,30E-03	5,14E-03
	Validación	1,65E-05	2,47E-03	3,37E-05	1,55E-05	2,04E-05	2,37E-03	1,47E-05	2,84E-03	4,87E-03
	Prueba	1,85E-05	2,70E-03	3,48E-05	1,74E-05	2,20E-05	2,54E-03	1,68E-05	3,02E-03	5,06E-03
600	Entrenamiento	1,78E-05	2,37E-03	3,54E-05	1,67E-05	2,43E-05	3,78E-05	1,87E-05	3,26E-03	5,11E-03
	Validación	1,31E-05	2,44E-03	3,36E-05	1,24E-05	2,06E-05	3,60E-05	1,46E-05	2,98E-03	4,65E-03
	Prueba	1,51E-05	2,67E-03	3,49E-05	1,43E-05	2,23E-05	3,68E-05	1,65E-05	3,10E-03	4,86E-03
900	Entrenamiento	1,64E-05	2,37E-03	3,55E-05	1,64E-05	2,42E-05	3,76E-05	1,87E-05	1,87E-03	5,12E-03
	Validación	1,20E-05	2,47E-03	3,34E-05	1,23E-05	2,11E-05	3,53E-05	1,47E-05	1,55E-03	4,77E-03
	Prueba	1,39E-05	2,69E-03	3,47E-05	1,41E-05	2,26E-05	3,65E-05	1,66E-05	1,62E-03	4,94E-03
1200	Entrenamiento	1,63E-05	2,39E-03	3,57E-05	1,65E-05	2,42E-05	3,75E-05	1,86E-05	4,99E-05	5,13E-03
	Validación	1,20E-05	2,42E-03	3,37E-05	1,22E-05	2,10E-05	3,56E-05	1,48E-05	4,84E-05	4,52E-03
	Prueba	1,39E-05	2,65E-03	3,49E-05	1,40E-05	2,26E-05	3,65E-05	1,67E-05	4,94E-05	4,78E-03

Tabla 5.3. Resultados función coseno: entrenamiento con el algoritmo ARγ-GLE en arquitectura 1-2-1

Épocas	Datos	$\hat{\mu} = 0$			$\hat{\mu} = 0.5$			$\hat{\mu} = 0.9$		
		$\hat{\alpha} = 0.1$	$\hat{\alpha} = 0.5$	$\hat{\alpha} = 0.9$	$\hat{\alpha} = 0.1$	$\hat{\alpha} = 0.5$	$\hat{\alpha} = 0.9$	$\hat{\alpha} = 0.1$	$\hat{\alpha} = 0.5$	$\hat{\alpha} = 0.9$
300	Entrenamiento	3,84E-03	1,13E-01	NaN	2,77E-05	NaN	NaN	1,70E-01	NaN	NaN
	Validación	3,46E-03	1,14E-01	NaN	2,49E-05	NaN	NaN	1,64E-01	NaN	NaN
	Prueba	3,70E-03	1,15E-01	NaN	2,62E-05	NaN	NaN	1,69E-01	NaN	NaN
600	Entrenamiento	3,79E-03	1,12E-01	NaN	2,77E-05	NaN	NaN	1,69E-01	NaN	NaN
	Validación	3,54E-03	1,13E-01	NaN	2,47E-05	NaN	NaN	1,65E-01	NaN	NaN
	Prueba	3,78E-03	1,14E-01	NaN	2,62E-05	NaN	NaN	1,69E-01	NaN	NaN
900	Entrenamiento	3,82E-03	1,05E-01	NaN	2,77E-05	NaN	NaN	1,69E-01	NaN	NaN
	Validación	3,47E-03	1,04E-01	NaN	2,45E-05	NaN	NaN	1,61E-01	NaN	NaN
	Prueba	3,70E-03	1,05E-01	NaN	2,59E-05	NaN	NaN	1,66E-01	NaN	NaN
1200	Entrenamiento	3,82E-03	1,05E-01	NaN	2,76E-05	NaN	NaN	1,70E-01	NaN	NaN
	Validación	3,50E-03	1,04E-01	NaN	2,49E-05	NaN	NaN	1,61E-01	NaN	NaN
	Prueba	3,74E-03	1,05E-01	NaN	2,64E-05	NaN	NaN	1,66E-01	NaN	NaN

Tabla 5.4. Resultados función coseno: entrenamiento con el algoritmo BP en arquitectura 1-2-1

Arquitectura	**ARγ-GLE** (α=0.1, γ=2.04, m_1=1)				**BP** ($\hat{\alpha}$=0.1, $\hat{\mu}$=0.5)			
	Entrenamiento		**Validación**		**Entrenamiento**		**Validación**	
	Media	Desviación	Media	Desviación	Media	Desviación	Media	Desviación
1-2-1	2,22E-05	9,48E-06	1,84E-05	9,10E-06	2,99E-05	3,82E-14	2,51E-05	2,52E-07
1-5-1	1,41E-05	4,09E-06	1,12E-05	3,10E-06	2,52E-05	1,26E-05	1,35E-05	8,01E-06
1-12-1	9,20E-06	6,40E-06	6,61E-06	3,56E-06	3,37E-05	2,62E-05	9,36E-06	4,05E-06
1-18-1	7,56E-06	6,13E-06	5,99E-06	4,40E-06	6,39E-01	3,86E-01	1,09E-01	1,67E-01
1-28-1	5,42E-06	5,13E-06	3,56E-06	3,54E-06	NaN	NaN	–	–
1-42-1	4,25E-06	3,69E-06	4,05E-06	4,38E-06	NaN	NaN	–	–
1-3-2-1	4,90E-03	1,48E-02	3,53E-03	1,07E-02	3,66E-03	1,52E-02	2,57E-03	1,06E-02
1-8-4-1	5,13E-06	3,55E-06	4,00E-06	3,61E-06	2,55E-04	8,21E-04	1,06E-04	3,02E-04
1-12-6-1	5,19E-06	2,59E-06	4,68E-06	3,48E-06	2,61E-03	5,66E-03	6,86E-04	1,51E-03
1-18-10-1	1,34E-05	3,92E-05	4,41E-06	3,03E-06	7,92E-02	9,02E-02	1,57E-02	2,95E-02

Tabla 5.5. Resultados función coseno: entrenamiento y validación con el algoritmo ARγ-GLE y el algoritmo BP usando diferentes arquitecturas

Épocas	**Datos**	$\gamma = 2.04\ m_1 = 1$			$\gamma = 1.61\ m_1 = 1$			$\gamma = 0.12\ m_1 = 11$		
		$\alpha = 0.01$	$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.01$	$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.01$	$\alpha = 0.1$	$\alpha = 0.5$
50	Entrenamiento	3,77	3,99	2,90	4,03	3,74	2,80	3,26	2,39	2,47
	Validación	23,01	51,04	37,40	19,83	56,84	41,36	36,39	37,71	41,80
	Prueba	39,80	46,44	48,95	41,66	49,21	47,75	39,00	46,91	61,42
100	Entrenamiento	3,24	3,51	2,51	3,13	2,93	2,53	2,77	2,08	2,13
	Validación	17,05	70,47	43,13	23,49	92,85	37,15	30,48	39,05	52,20
	Prueba	36,40	57,87	44,25	39,47	66,42	44,88	38,45	45,91	68,23
200	Entrenamiento	2,89	2,60	2,18	2,86	2,50	2,23	2,32	1,67	1,81
	Validación	15,10	144,75	58,94	27,56	202,25	45,03	30,31	32,98	53,03
	Prueba	37,05	80,71	48,87	42,96	94,46	43,51	39,42	36,49	61,72
400	Entrenamiento	2,23	2,16	2,09	2,24	2,21	1,91	1,79	1,38	1,53
	Validación	22,87	261,10	103,06	60,33	322,31	47,73	29,86	37,81	73,05
	Prueba	43,01	113,41	72,80	64,89	114,96	47,91	37,81	37,26	72,49

Tabla 5.6. Resultados precio de vivienda: entrenamiento con el algoritmo ARγ-GLE en arquitectura 13-15-1

Épocas	**Datos**	$\hat{\mu} = 0$			$\hat{\mu} = 0.5$			$\hat{\mu} = 0.9$		
		$\hat{\alpha} = 0.01$	$\hat{\alpha} = 0.1$	$\hat{\alpha} = 0.5$	$\hat{\alpha} = 0.01$	$\hat{\alpha} = 0.1$	$\hat{\alpha} = 0.5$	$\hat{\alpha} = 0.01$	$\hat{\alpha} = 0.1$	$\hat{\alpha} = 0.5$
50	Entrenamiento	2,66	4,27	NaN	2,43	175,97	NaN	4,10	NaN	NaN
	Validación	26,39	51,99	NaN	23,35	227,91	NaN	29,24	NaN	NaN
	Prueba	36,69	45,78	NaN	37,48	261,94	NaN	36,80	NaN	NaN
100	Entrenamiento	2,29	3,95	NaN	2,15	310,79	NaN	3,54	NaN	NaN
	Validación	26,03	64,78	NaN	25,58	1554,09	NaN	33,49	NaN	NaN
	Preba	38,19	52,31	NaN	40,09	825,58	NaN	38,49	NaN	NaN
200	Entrenamiento	1,90	3,97	NaN	1,77	306,66	NaN	3,26	NaN	NaN
	Validación	28,27	77,14	NaN	30,05	3174,19	NaN	34,39	NaN	NaN
	Prueba	39,70	57,24	NaN	41,93	1566,35	NaN	43,99	NaN	NaN
400	Entrenamiento	1,48	3,69	NaN	1,38	477,47	NaN	3,30	NaN	NaN
	Validación	29,93	88,64	NaN	36,71	2577,80	NaN	32,08	NaN	NaN
	Prueba	39,85	68,10	NaN	44,16	1785,25	NaN	42,88	NaN	NaN

Tabla 5.7. Resultados precio de vivienda: entrenamiento con el algoritmo BP en arquitectura 13-15-1

Arquitectura	**ARγ-GLE** (α=0.01, γ=2.04, m_1=1)				**BP**($\hat{\alpha}$=0.01, $\hat{\mu}$=0.5)			
	Entrenamiento		**Validación**		**Entrenamiento**		**Validación**	
	Media	Desviación	Media	Desviación	Media	Desviación	Media	Desviación
9-2-2	13,583	0,092	12,503	3,656	3,082	0,049	22,185	3,205
9-5-2	13,039	0,173	11,010	3,369	1,924	0,141	20,052	3,835
9-12-2	12,816	0,135	10,057	1,384	1,288	0,082	19,316	4,114
9-18-2	12,682	0,153	9,601	1,430	1,214	0,092	15,647	2,604
9-28-2	12,655	0,155	10,005	1,344	1,145	0,090	16,644	1,922
9-42-2	12,645	0,131	9,766	1,019	1,136	0,114	16,288	1,816
9-3-2-2	13,142	0,233	12,405	2,365	2,313	0,091	13,405	2,014
9-8-4-2	12,641	0,162	12,405	2,602	1,107	0,075	14,101	3,268
9-12-6-2	12,472	0,159	11,285	2,932	0,874	0,080	11,781	3,756
9-18-10-2	12,431	0,129	10,857	1,213	0,782	0,065	12,667	2,786

Tabla 5.8. Resultados precio de vivienda: entrenamiento y validación con el algoritmo ARγ-GLE y el algoritmo BP usando diferentes arquitecturas

Épocas	Datos	$\gamma = 2.04\ m_1 = 1$			$\gamma = 1.61\ m_1 = 1$			$\gamma = 0.12\ m_1 = 11$		
		$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.9$	$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.9$	$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 0.9$
300	Entrenamiento	2,23E-02	2,26E-02	2,31E-02	2,19E-02	2,33E-02	2,25E-02	2,37E-02	2,22E-02	2,28E-02
	Validación	2,80E-02	2,88E-02	2,74E-02	3,05E-02	2,72E-02	2,80E-02	2,75E-02	2,71E-02	2,82E-02
	Prueba	3,60E-02	3,51E-02	3,31E-02	3,59E-02	3,38E-02	3,39E-02	3,36E-02	3,12E-02	3,12E-02
600	Entrenamiento	2,20E-02	2,22E-02	2,27E-02	2,13E-02	2,33E-02	2,21E-02	2,34E-02	2,19E-02	2,21E-02
	Validación	2,86E-02	2,91E-02	2,79E-02	3,18E-02	2,75E-02	2,82E-02	2,78E-02	2,79E-02	2,79E-02
	Prueba	3,64E-02	3,53E-02	3,29E-02	3,70E-02	3,34E-02	3,37E-02	3,40E-02	3,13E-02	3,06E-02
900	Entrenamiento	2,20E-02	2,18E-02	2,24E-02	2,13E-02	2,31E-02	2,16E-02	2,30E-02	2,17E-02	2,21E-02
	Validación	2,90E-02	2,91E-02	2,76E-02	3,27E-02	2,83E-02	2,82E-02	2,79E-02	2,80E-02	2,83E-02
	Prueba	3,69E-02	3,54E-02	3,23E-02	3,69E-02	3,29E-02	3,34E-02	3,38E-02	3,13E-02	3,04E-02
1200	Entrenamiento	2,19E-02	2,18E-02	2,24E-02	2,12E-02	2,28E-02	2,19E-02	2,31E-02	2,15E-02	2,21E-02
	Validación	2,92E-02	2,93E-02	2,80E-02	3,30E-02	2,83E-02	2,82E-02	2,80E-02	2,83E-02	2,83E-02
	Prueba	3,72E-02	3,55E-02	3,24E-02	3,67E-02	3,28E-02	3,34E-02	3,37E-02	3,12E-02	3,05E-02

Tabla 5.9. Resultados cáncer de seno: entrenamiento con el algoritmo ARγ-GLE en arquitectura 13-10-2

Épocas	Datos	$\hat{\mu} = 0$			$\hat{\mu} = 0.5$			$\hat{\mu} = 0.9$		
		$\hat{\alpha} = 0.1$	$\hat{\alpha} = 0.5$	$\hat{\alpha} = 0.9$	$\hat{\alpha} = 0.1$	$\hat{\alpha} = 0.5$	$\hat{\alpha} = 0.9$	$\hat{\alpha} = 0.1$	$\hat{\alpha} = 0.5$	$\hat{\alpha} = 0.9$
300	Entrenamiento	2,32E-02	4,37E-02	4,76E-02	2,51E-02	4,88E-02	5,15E-02	4,82E-02	6,02E-02	6,00E-02
	Validación	2,93E-02	3,77E-02	3,77E-02	2,77E-02	3,77E-02	3,97E-02	4,01E-02	4,65E-02	4,72E-02
	Prueba	2,94E-02	3,26E-02	3,24E-02	2,95E-02	3,29E-02	3,55E-02	3,69E-02	4,18E-02	4,35E-02
600	Entrenamiento	2,31E-02	4,24E-02	4,75E-02	2,51E-02	4,76E-02	5,13E-02	4,80E-02	5,92E-02	5,99E-02
	Validación	3,04E-02	3,68E-02	3,86E-02	2,80E-02	3,77E-02	3,97E-02	4,08E-02	4,64E-02	4,70E-02
	Prueba	2,89E-02	3,19E-02	3,32E-02	2,89E-02	3,25E-02	3,57E-02	3,55E-02	4,04E-02	4,35E-02
900	Entrenamiento	2,29E-02	4,24E-02	4,73E-02	2,48E-02	4,74E-02	5,06E-02	4,75E-02	5,88E-02	5,98E-02
	Validación	3,10E-02	3,70E-02	3,83E-02	2,81E-02	3,68E-02	4,00E-02	4,05E-02	4,89E-02	4,70E-02
	Prueba	2,94E-02	3,21E-02	3,34E-02	2,90E-02	3,18E-02	3,61E-02	3,51E-02	4,23E-02	4,35E-02
1200	Entrenamiento	2,29E-02	4,15E-02	4,77E-02	2,50E-02	4,75E-02	5,10E-02	4,78E-02	5,86E-02	5,99E-02
	Validación	3,09E-02	3,60E-02	3,82E-02	2,82E-02	3,63E-02	4,06E-02	4,02E-02	4,88E-02	4,69E-02
	Prueba	2,94E-02	3,21E-02	3,35E-02	2,91E-02	3,21E-02	3,65E-02	3,51E-02	4,23E-02	4,35E-02

Tabla 5.10. Resultados cáncer de seno: entrenamiento con el algoritmo BP en arquitectura 13-10-2

Arquitectura	**ARγ-GLE** (α=0.5, γ=0.12, m_1=11)				**BP** ($\hat{\alpha}$=0.1, $\hat{\mu}$=0.5)			
	Entrenamiento		**Validación**		**Entrenamiento**		**Validación**	
	Media	Desviación	Media	Desviación	Media	Desviación	Media	Desviación
9-2-2	2,84E-02	1,91E-03	1,98E-02	1,39E-03	3,07E-02	2,21E-03	1,94E-02	1,37E-03
9-5-2	2,48E-02	2,73E-03	2,08E-02	9,25E-04	2,61E-02	3,23E-03	2,01E-02	5,03E-04
9-12-2	2,10E-02	2,07E-03	2,19E-02	7,51E-04	2,31E-02	2,61E-03	2,05E-02	6,81E-04
9-18-2	2,09E-02	1,85E-03	2,21E-02	5,87E-04	2,44E-02	1,76E-03	2,23E-02	1,44E-03
9-28-2	2,11E-02	2,28E-03	2,28E-02	8,34E-04	2,83E-02	5,94E-03	2,56E-02	3,19E-03
9-42-2	2,01E-02	1,91E-03	2,28E-02	4,81E-04	3,99E-02	5,31E-03	2,98E-02	2,28E-03
9-3-2-2	3,35E-02	2,31E-03	1,96E-02	1,40E-03	3,34E-02	1,89E-03	1,88E-02	1,37E-03
9-8-4-2	2,64E-02	3,39E-03	1,97E-02	1,25E-03	3,10E-02	3,48E-03	1,91E-02	7,08E-04
9-12-6-2	2,40E-02	3,05E-03	1,94E-02	1,35E-03	2,98E-02	4,15E-03	1,87E-02	1,20E-03
9-18-10-2	2,44E-02	2,62E-03	2,01E-02	1,39E-03	2,98E-02	4,53E-03	1,90E-02	9,72E-04

Tabla 5.11. Resultados cáncer de seno: entrenamiento y validación con el algoritmo ARγ-GLE y el algoritmo BP usando diferentes arquitecturas

En aplicaciones de clasificación relacionadas con diagnósticos biomédicos, a menudo resulta útil calificar el rendimiento del entrenamiento en términos de exactitud, sensibilidad, especificidad falsos positivos, falsos negativos, valor predictivo positivo y valor predictivo negativo (Marcano, et al. 2011). Estas medidas indican la validez de un test, por lo tanto deben ser preferiblemente calculadas con datos totalmente objetivos, es decir que no se hayan utilizado ni para entrenar la red ni para validarla. Por lo tanto para obtenerlas se utilizan los datos de prueba (en el anexo B se presenta la definición y expresiones relacionadas con estas medidas de rendimiento para el problema *cáncer de seno*).

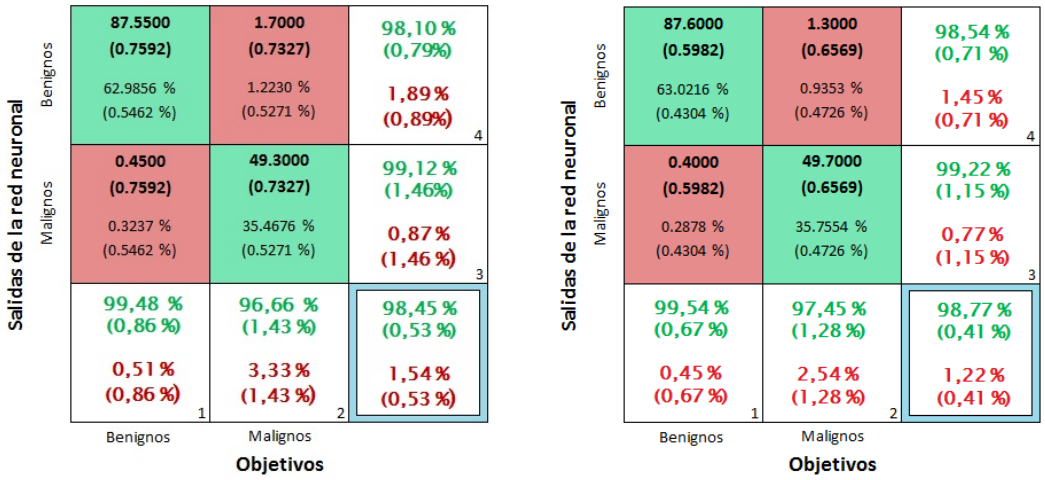


Figura 5.1. Matriz de confusión para datos de prueba: a) Algoritmo ARγ-GLE b) Algoritmo BP

La figura 5.1 muestra las matrices de confusión para los datos de prueba, donde se especifican los valores promedio y desviación estándar (los valores en paréntesis son las desviaciones estándar) de los parámetros mencionados anteriormente para 20 sesiones de entrenamiento. la información en cada matriz de confusión se presenta como sigue.

- Los recuadros verdes muestran el número y el porcentaje de los tumores bien clasificados.

- Los recuadros rojos indican el número y porcentaje de los tumores mal clasificados.

- En el recuadro blanco 1, con verde se indica el porcentaje de especificidad y con rojo el porcentaje de falsos positivos.

- En el recuadro blanco 2, con verde se indica el porcentaje de sensibilidad y con rojo los falsos negativos.

- En el recuadro blanco 3, con verde se indica el valor predictivo negativo y con rojo su complemento porcentual.

- En el recuadro blanco 4, con verde se indica el valor predictivo positivo y con rojo su complemento porcentual.

- En el recuadro enmarcado de azul, con verde se indica el porcentaje de exactitud y con rojo el porcentaje de error.

5.3.1 Análisis de resultados

Como se ha dicho, α y $\hat{\alpha}$ son parámetros análogos relacionados con el ritmo de aprendizaje, que afectan proporcionalmente la actualización calculada por los algoritmos BP y ARγ-GLE respectivamente. En general, entre más pequeños se escojan estos parámetros, menores serán los cambios en los pesos sinápticos para cada iteración. Los resultados obtenidos indican que el comportamiento del algoritmo ARγ-GLE (en relación con el algoritmo BP) para diferentes ritmos de aprendizaje es más adecuado, puesto que cuando $\hat{\alpha}$ toma valores altos, el error de entrenamiento con el algoritmo BP presenta fluctuaciones indeseables e incluso inestabilidad. Esto se puede notar en las tablas 5.4 y 5.7, en las cuales para tasas de aprendizaje altas el error es indeterminado, hecho que no se presenta en ninguna de las pruebas realizadas con el algoritmo ARγ-GLE.

Para los datos *función coseno* y *precio de vivienda*, el algoritmo ARγ-GLE obtuvo en promedio los mejores resultados en términos de generalización para las diferentes arquitecturas utilizadas (ver tablas 5.5 y 5.8). En el caso de la *función coseno*, las ventajas del algoritmo ARγ-GLE son mucho más marcadas para redes con más neuronas ocultas. En el caso de los datos *precio de vivienda*, el algoritmo ARγ-GLE obtuvo mejores resultados que el algoritmo BP, siendo más significativas las ventajas en aquellas arquitecturas con menos neuronas ocultas.

Un aspecto positivo del algoritmo ARγ-GLE, es que los resultados de entrenamiento obtenidos en las diferentes arquitecturas, son más constantes que los del algoritmo BP. Comúnmente en el algoritmo BP la escogencia de la tasa de aprendizaje resulta imprevisible, por lo que tasas de aprendizaje que dan buenos resultados en una arquitectura (en términos de error de entrenamiento) pueden ocasionar comportamientos indeseables en otras (ver tabla 5.5), por lo general (aunque no necesariamente) para arquitecturas muy grandes suelen ser necesarias tasas de aprendizaje muy bajas, lo cual se traduce en una diminución de la velocidad de convergencia. En las pruebas realizadas son notorias las ventajas que tiene el algoritmo ARγ-GLE para sobrellevar este problema.

El rendimiento del algoritmo BP fue mejor para el conjunto de datos *cáncer de seno* en las distintas arquitecturas probadas (en términos de error de generalización). Esto se corrobora en la matriz de confusión con mejor exactitud, sensibilidad y valor predictivo positivo para los datos de prueba (ver figura 5.1).

Ambos algoritmos se ven afectados por el problema del sobreaprendizaje, así entonces, un error mínimo en entrenamiento, no necesariamente implica un alto rendimiento o un mínimo error de validación, esto puede notarse claramente en las tablas 5.5, 5.8 y 5.11 para las cuales se buscó obtener los mejores resultados en términos de generalización. Cabe destacar, que en promedio los menores errores de entrenamiento en las primeras épocas se obtuvieron con el algoritmo ARγ-GLE.

Capítulo 6
Conclusiones y Trabajos Futuros

6.1 Conclusiones

El algoritmo ARγ-GLE permite el entrenamiento de redes neuronales MLP haciendo uso de los principios que rigen la actualización de parámetros en el algoritmo ARγ. El primero se obtiene al incorporar en la estructura interna del segundo, un funcional conocido como gradiente local de error. El gradiente local de error se deriva al aplicar el método de gradiente descendiente a las redes MLP y es el concepto que permite al algoritmo ARγ-GLE entrenar las capas ocultas y además compensar las posibles no linealidades introducidas por las funciones de activación.

Mediante simulación se validó la convergencia y generalización de la combinación *Arquitectura MLP + Algoritmo ARγ-GLE* para diferentes problemas relacionados con clasificación de patrones y aproximación funcional, con los que se determinó que dicha combinación es capaz de: resolver problemas que involucran patrones de entrada que no son linealmente separables (como el caso de la función XOR); establecer regiones de decisión distintas a una línea recta, alcanzando probabilidades de clasificación muy cercanas a las teóricas en el sentido de Bayes; y estimar funciones (como la de Saito y Nakano) con un alto coeficiente de ajuste.

Se estudió experimentalmente el efecto que los parámetros de entrenamiento (α, γ, y m_1) del algoritmo ARγ-GLE tienen en la curva de entrenamiento. A partir de lo cual se estableció un orden lógico para la escogencia de dichos parámetros siendo α (seguido respectivamente por m_1 y γ), el que más efectos tiene en términos de convergencia y desajuste para un mínimo local dado.

Se realizo un análisis comparativo basado en simulción del algoritmo ARγ-GLE y el algoritmo BP en torno a tres reconocidos problemas de "benchmarking". En base a los resultados obtenidos, el ARγ-GLE presenta un mejor comportamiento en lo rela-

cionado al ritmo de aprendizaje, evitando la inestabilidad en todas las pruebas realizadas, lo cual es un resultado bastante prometedor que invita a investigar teóricamente el algoritmo.

6.2 Trabajos Futuros

- Estudiar los efectos de aplicar distintas técnicas de parada temprana al algoritmo ARγ-GLE buscando evitar el sobreajuste y largas sesiones de entrenamiento.

- Esta investigación se centra en un algoritmo diseñado para redes neuronales MLP. Sin embargo, queda abierta la posibilidad de utilizarlo en otros modelos como lo son las redes dinámicas, para las cuales el entrenamiento es similar (Beale, et al 2010).

- Realizar estudios teóricos y prácticos para mejorar el desempeño del algoritmo ARγ-GLE, pensando en la posibilidad de entrenar redes con parámetros de entrenamiento y de red variables en el tiempo.

Referencias

Albuquerque, V. H.; Alexandria, A. R.; Cortez, P. C.; Tavares, J. M. (2009b). Evaluation of Multilayer Perceptron and Self-Organizing Map Neural Network Topologies Applied on Microstructure Segmentation from Metallographic Images, *NDT & E International. Vol. 42, No. 7, pp. 644-651.*

Beale, M. H.; Hagan, M. T.; Demuth, H. (2010). Neural Network Toolbox 7 User's Guide, *Natick, MA, The MathWorks.*

Bergmeir, C.; Benítez, J. M. (2012a). On the Use of Cross-Validation for Time Series Predictor Evaluation, *Information Sciences. Vol. 191, pp. 192-213.*

Brio, B. M.; Sanz, A. (2007). Redes Neuronales Artificiales y Sistemas Borrosos, *3° Edición, Mexico, Alfaomega Ra-Ma.*

Dai, Q.; Liu N. (2012a). Alleviating the Problem of Local Minima in Backpropagation Through Competitive Learning, *Neurocomputing. Vol. 94, pp. 152-158.*

Florez, R.; Fernandez, J. M. (2008). Las Redes Neuronales Artificiales. Fundamentos Teóricos y Aplicaciones Practicas, *España, Netbiblo.*

Haykin, S. (1998). Neural Networks: A Comprehensive Foundation, *2° Edición, Ontario, McMaster University.* .

Jojoa, P.E. (2003). Um Algoritmo Acelerador de Parámetros, *(Tesis de Doctorado). Brasil: Escola Polit'ecnica da Universidad de S'ao Paulo.*

LeCun, Y.; Bottou, L.; Orr, G. B.; Müller, K. R. (1998). Eficient Backpro., *Neural Networks: Tricks of the Trade. Springer-Verlag.*

López, W. G. (2010). Pronóstico de una Serie Temporal Usando Redes Neuronales, *(Trabajo de grado). Oaxaca: Universidad Tecnológica de la Mixteca.*

Manolakis, D.; Ingle, V.; Kogon, S. (2005). Statical and Adaptive Signal Processing: Spectral Estimation, Signal Modeling, Adaptive Filtering and Array Processing, *Boston, Artech House.*

Marcano A.; Quintanilla J.; Andina D. (2011). Breast Cancer Classification Applying Artificial Metaplasticity Algorithm, *Neurocomputing. Vol. 74, pp. 1243-1250*

Martinez, A. C. (2005). Modelos de Regresión Basados en Redes Neuronales de Unidades Producto Diseñadas y Entrenadas Mediante Algoritmos de Optimización Híbrida. Aplicaciones, *(Tesis de Doctorado). España: Universidad de Granada, Departamento de Ciencias de la Computación e Ingeniería.*

Mitchel, T. M. (1997). Machine Learning, *USA, McGraw-Hill.*

Mukherjee, I.; Routroy S. (2012). Comparing the Performance of Neural Networks Developed by Using Levenberg Marquardt and Quasi-Newton with the Gradient Descent Algorithm for Modelling a Multiple Response Grinding Process, *Expert Systems with Applications. Vol. 39, pp. 2397-2407.*

Palacios, A. (2012). Implementación de un Módulo para el Entrenamiento y Evaluación de Redes Neuronales Mediante GPUs, *(Trabajo de grado). España: Universitat Politècnica de València, Escola Tècnica Superior d'Enginyeria Informàtica.*

Poularikas, A. D.; Ramadan, Z. M. (2006). Adaptive Filtering Primer with Matlab, *Boca Raton, Taylor & Francis Group.*

Principe, J. C.; Euliano N. R.; Lefebvre W. C. (2000). Neural and Andaptive Systems. Fundamentals Through Simulation, *New York, John Wiley & Sons*

Quetglás, G. M.; Toledo, F.; Cerverón, V. (1995). Fundamentos de Informática y Programación, *España, Universidad de Valencia.*

Serrano, A. J.; Soria, E.; Martin, J. D. (2010). Redes Neuronales Artificiales, *Escola Técnica Superior d' Enginyeria, Deparmente d' Enginyeria Electronica.*

Webb, A. (2002). Statistical Pattern Recognition, *2° Edición, UK, John Wiley & Sons.*

Yegnanarayana, B. (2006). Artificial Neural Networks, *India, Pretice-Hall.*

Anexo A

Reconocimiento Estadístico de Patrones

A.1 Criterio de Bayes para mínimo error

Considérese C clases $\omega_1, \omega_2, \omega_3, ..., \omega_C$, con probabilidades a priori $P(\omega_1)$, $P(\omega_2)$, $P(\omega_3)$, ..., $P(\omega_C)$ conocidas. Si lo que se desea es minimizar la probabilidad de cometer un error en la clasificación de un objeto y la única información que se tiene son las probabilidades a priori para cada clase, entonces se podría asignar el objeto a la clase ω_j sí

$$P(\omega_j) > P(\omega_k) \;\; k = 1, 2, 3, ..., C; \; k \neq j. \tag{A.1}$$

Con esto, todos los objetos quedarían incluidos en una única clase. Para clases equiprobables, es decir con igual probabilidad, los patrones se asignarían indiscriminadamente en cualquiera de ellas (Webb 2002).

Sin embargo, como lo que se tiene es un vector **x**, el cual se debe asignar a cualquiera de las C clases, resulta mas indicado hacer uso de un criterio basado en probabilidades que consiste en asignar el patrón **x** a la clase ω_j, si la probabilidad de ω_j dada la muestra **x** (probabilidad a posteriori $P(\omega_j|\mathbf{x})$) es mayor que para todas las clases $\omega_1, \omega_2, \omega_3, ..., \omega_C$. Así entonces, se asigna **x** a la clase ω_j sí

$$P(\omega_j|\mathbf{x}) > P(\omega_k|\mathbf{x}) \;\; k = 1, 2, 3, ..., C; \; k \neq j. \tag{A.2}$$

Las probabilidades a posteriori $P(\omega_j|\mathbf{x})$ se pueden expresar en términos de las probabilidades a priori y de las funciones de probabilidad de las clases $P(\mathbf{x}|\omega_j)$ usando el teorema de Bayes como

$$P(\omega_i|\mathbf{x}) = \frac{P(\mathbf{x}|\omega_i)P(\omega_i)}{P(\mathbf{x})}. \tag{A.3}$$

donde $P(\mathbf{x})$ es la probabilidad marginal de el patrón **x**. Con lo que el criterio de decisión anterior (ecuación A.2) puede reescribirse como

$$\frac{P(\mathbf{x}|\omega_j)P(\omega_j)}{P(\mathbf{x})} > \frac{P(\mathbf{x}|\omega_k)P(\omega_k)}{P(\mathbf{x})} \quad k = 1, 2, 3, ..., C; \; k \neq j. \tag{A.4}$$

Por tanto, se asigna **x** a la clase ω_j sí.

$$P(\mathbf{x}|\omega_j)P(\omega_j) > P(\mathbf{x}|\omega_k)P(\omega_k) \quad k = 1, 2, 3, ..., C; \; k \neq j, \tag{A.5}$$

esto se conoce como el criterio de Bayes para mínimo error.

La probabilidad de cometer un error $P(error)$ puede ser expresada como

$$P(error) = \sum_{i=1}^{C} P(error|\omega_i)P(\omega_i) \tag{A.6}$$

donde $P(error|\omega_i)$ es la probabilidad de clasificar incorrectamente patrones de la clase ω_i.

Si se tiene en cuenta que el criterio de Bayes particiona el espacio de medida en C regiones Ω_1, Ω_2, Ω_3,...., Ω_C tales que si **x** $\in \Omega_j$ entonces **x** pertenece a la clase ω_j, se puede definir $P(error|\omega_i)$ como.

$$P(error|\omega_i) = \int_{C[\Omega_i]} P(\mathbf{x}|\omega_i)d\mathbf{x} \tag{A.7}$$

donde la región $C[\Omega_i]$ corresponde al complemento de la región Ω_i expresada como $\sum_{i=1,j\neq i}^{C} \Omega_j$.

Utilizando la ecuación A.7 se puede reescribir la ecuación A.6 como

$$P(error) = \sum_{i=1}^{C} \int_{C[\Omega_i]} P(\mathbf{x}|\omega_i)P(\omega_i)d\mathbf{x}$$

$$= \sum_{i=1}^{C} P(\omega_i)(1 - \int_{\Omega_i} P(\mathbf{x}|\omega_i)d\mathbf{x}) \tag{A.8}$$

$$= 1 - \sum_{i=1}^{C} P(\omega_i) \int_{\Omega_i} P(\mathbf{x}|\omega_i)d\mathbf{x}.$$

La ecuación A.8 significa que minimizar la probabilidad de cometer un error de clasificación es equivalente a maximizar

$$\sum_{i=1}^{C} P(\omega_i) \int_{\Omega_i} P(\mathbf{x}|\omega_i) d\mathbf{x} \tag{A.9}$$

que viene siendo la probabilidad de clasificar correctamente un patrón. Por consiguiente, lo que se desea es escoger regiones Ω_i para las cuales la integral de la ecuación A.9 es un máximo. Esto se puede lograr seleccionando una región Ω_i para la cual $P(\omega_i)P(\mathbf{x}|\omega_i)$ sea la mayor (en relación a las otras clases), es decir aplicando el criterio de Bayes para mínimo error. Así la probabilidad de correcta clasificación P_c esta dada por la siguiente expresión

$$P_c = \int max_i P(\omega_i) P(\mathbf{x}|\omega_i) d\mathbf{x}, \tag{A.10}$$

para la cual la región de integración es todo el espacio de medida. Así, el error de Bayes se puede expresar como

$$e_B = 1 - (\int max_i P(\omega_i) P(\mathbf{x}|\omega_i) d\mathbf{x}) \tag{A.11}$$

Anexo B

Medidas de Clasificación

El rendimiento en clasificación con el algoritmo propuesto y el algoritmo BP para los datos *cáncer de seno* son comparados en términos de exactitud, sensibilidad, especificidad, falsos positivos, falsos negativos, valor predictivo positivo y valor predictivo negativo.

La exactitud determina el rendimiento total de la red, en esencia indica la capacidad adquirida por la red para identificar el tipo de tumor y se define como:

$$\%Exactitud = \frac{\#\ de\ tumores\ clasificados\ correctamente}{\#\ total\ de\ datos\ en\ el\ set\ utilizado} x100. \tag{B.1}$$

La sensibilidad estima la probabilidad de que un tumor maligno sea clasificado como maligno. Se calcula como el porcentaje de tumores malignos bien clasificados en relación con el número total de tumores malignos en el set utilizado, como se describe en la siguiente ecuación:

$$\%Sensibilidad = \frac{\#\ de\ tumores\ malignos\ bien\ clasificados}{\#\ total\ de\ tumores\ malignos\ en\ el\ set\ utilizado} x100. \tag{B.2}$$

La especificidad estima la probabilidad de que un tumor benigno sea clasificado como benigno. Se calcula como el porcentaje de tumores benignos bien clasificados en relación con el número total de tumores benignos, de acuerdo con

$$\%Especificidad = \frac{\#\ de\ tumores\ benignos\ bien\ clasificados}{\#\ total\ de\ tumores\ benignos\ en\ el\ set\ utilizado} x100. \tag{B.3}$$

El porcentaje de falsos negativos $\%FN$ describe el porcentaje de tumores malignos que fueron clasificados incorrectamente en relación al número total de tumores malignos en el set utilizado y el porcentaje de falsos positivos $\%FP$ indica el porcentaje de

tumores benignos mal clasificados en relación con el número total de tumores benignos en el set de datos utilizado, de acuerdo con

$$\%FN = \frac{\#\ de\ tumores\ malignos\ mal\ clasificados}{\#\ total\ de\ tumores\ malignos\ en\ el\ set\ utilizado} x100, \tag{B.4}$$

$$\%FP = \frac{\#\ de\ tumores\ benignos\ mal\ clasificados}{\#\ total\ de\ tumores\ benignos\ en\ el\ set\ utilizado} x100. \tag{B.5}$$

Los valores predictivos están asociados a la sensibilidad y especificidad de una prueba, en esencia miden la probabilidad de un resultado. Así entonces, el valor predictivo positivo, estima la probabilidad de que un tumor sea maligno si la red neuronal lo clasifico como maligno y el valor predictivo negativo estima la probabilidad de que el tumor sea benigno si la red neuronal lo clasifico como benigno. El porcentaje del valor predictivo positivo $\%VPP$ y porcentaje del valor predictivo negativo $\%VPN$ se calculan como

$$\%VPP = \frac{\#\ de\ tumores\ benignos\ bien\ clasificados}{\#\ total\ de\ tumores\ clasificados\ como\ benignos} x100, \tag{B.6}$$

$$\%VPN = \frac{\#\ de\ tumores\ malignos\ bien\ clasificados}{\#\ total\ de\ tumores\ clasificados\ como\ malignos} x100. \tag{B.7}$$