FAUSTO MIGUEL CASTRO CAICEDO

ENTRENAMIENTO DE REDES NEURONALES ARTIFICIALES SLFN EN APLICACIONES DE CLASIFICACIÓN QUE INVOLUCREN CANTIDADES MASIVAS DE INFORMACIÓN

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Doctorado en Ciencias de la Electrónica

Popayán
2019

FAUSTO MIGUEL CASTRO CAICEDO

ENTRENAMIENTO DE REDES NEURONALES ARTIFICIALES SLFN EN APLICACIONES DE CLASIFICACIÓN QUE INVOLUCREN CANTIDADES MASIVAS DE INFORMACIÓN

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

Doctor en
Ciencias de la Electrónica

Director:
Ph.D. Pablo Emilio Jojoa Gomez

Popayán
2019

A Dios. No se necesita verlo
para saber que mueve las cosas.

Agradecimientos

A mi familia; Rosalina (mamá), Rodrigo (papá) y Marcos (hermano); por su incondicional apoyo a lo largo de mi vida. Ustedes me enseñaron a valorar las cosas y a luchar por conseguirlas. Siempre agradezco y agradeceré todo el cariño y el amor que me han brindado. Los quiero mucho.

A mi asesor de tesis Ph.D. Pablo Emilio Jojoa Gomez con quien hemos compartido innumerables charlas relacionadas con filtrado adaptativo y redes neuronales. Ha sido un placer y verdaderamente un honor el haber recibido sus orientaciones, así como el que haya compartido su experiencia investigativa conmigo. Un agradecimiento especial por su constante acompañamiento, confianza, respaldo y paciencia en aquellos momentos de la investigación cuando los buenos resultados parecían esquivos y por su oportuna gestión en lo relacionado con todo tipo de trámites y documentos necesarios a lo largo de mis estudios doctorales.

Al Ph.D Giovanny Lopez Perafán por su colaboración en relación a la pasantía de investigación. Todas sus recomendaciones fueron muy acertadas y esta fue una experiencia muy enriquecedora a nivel investigativo y personal. Gracias por su gestión.

Al Ph.D Guefry Agredo uno de mis profesores de maestría y compañero en el ciclo de doctorado con quien se dieron muy buenas terapias de grupo y quien siempre fue solidario a la hora de compartir documentación útil.

A mi compañero y gran amigo Mg. Gustavo Adolfo Gómez quien con su creatividad y estilo aportó dándole un toque adicional de elegancia y claridad a muchas de las presentaciones y actividades que desarrollé en el marco de mis estudios de doctorado.

Al Ph.D Ruben Mateo Lorenzo Toledo (España) por su valiosa asesoría durante mi estancia de investigación en la Universidad de Valladolid y por su gestión en lo relacionado con los elementos logísticos, técnicos y en general todos aquellos trámites requeridos para la realización y aprovechamiento de dicha estancia. Seis meses de mucho aprendizaje que hoy por hoy recuerdo con especial cariño, tiempo que, aunque corto, fue suficiente para que forjásemos una bonita amistad que se fortaleció con

kilómetros y kilómetros de asfalto a lo largo y ancho de la hermosa Pucela más una que otra caña. Me sentí como en casa tío.

Al Ph.D Ramon Duran (España) por su disposición a colaborar con mi proceso investigativo durante la pasantía. Agradezco sus recomendaciones en lo relacionado con el lenguaje de programación R, sus sugerencias en relación a la clasificación de enlaces ópticos y por todas aquellas charlas realizadas bajo su dirección en las que fui acogido como un investigador más del Grupo de Comunicaciones Ópticas GCO de la universidad de Valladolid.

Al Ph.D Wilfredo Alfonso Morales por sus importantes aportes en materia de redacción y por ayudar a enriquecer la calidad de este documento con sus valiosas recomendaciones.

Extiendo mis agradecimientos a la Facultad de Ingeniería Electrónica y Telecomunicaciones de la Universidad del Cauca, la Escuela Técnica Superior de Ingeniería de Telecomunicación ETSIT de la Universidad de Valladolid, a mis compañeros, profesores y todas aquellas personas que contribuyeron propiciando un ambiente favorable para el desarrollo de este trabajo, en especial a Anyela Ximena Vidal, Eiver Ordoñez Ibarra, Neil Jimenes, Noemí Merayo (España) e Ignacio de Miguel (España).

Resumen

En este documento se presenta el diseño y prueba de un modelo de entrenamiento para redes neuronales de propagación hacia adelante de una sola capa oculta. El nuevo modelo se denomina máquina de entrenamiento comprimido basada en ELM o MEC-ELM y, actúa comprimiendo la información proveniente de la capa oculta por medio de un subconjunto de nodos de la misma capa, esto permite disminuir considerablemente la complejidad computacional en comparación con la solución de ELM para grandes conjuntos de entrenamiento. Resultados experimentales para diferentes problemas de clasificación indican que el modelo propuesto, permite una reducción alrededor del 45.3 $\%$ en el tiempo de entrenamiento en comparación con ELM, alcanzando a la vez rendimientos similares en términos de generalización.

Palabras Clave: ELM, redes neuronales, entrenamiento automático, técnicas de clasificación, aprendizaje supervisado

Abstract

The design and testing of a new training model for Single hidden Layer Feedforward Networks is presented, which is called Compressed Training Machine based on ELM o MEC-ELM. This model acts by compressing the information coming from the hidden layer through a subset of nodes from the same layer, which allows to reduce the computational complexity compare to ELM solution for large datasets. Experimental results for different classification problems indicate that the proposed model allows a reduction of around 45.3 $\%$ in training time compared to ELM, reaching at the same time similar performances in terms of generalization.

Key words: ELM; neural networks; machine learning; classification technics; supervised learning

Contenido

Pág.

Lista de Figuras

Pág.

Lista de Tablas

Pág.

Lista de Símbolos

E	Letra mayúscula en negrita denota matriz
e	Letra minúscula en negrita denota vector
E o e	Letra cursiva sin negrita denota escalar
$<$ **e**.**v** $>$	Producto escalar entre el vector **e** y el vector **v**
$\mathbf{E}^{\dagger}$	Pseudoinversa o inversa generalizada de More Penrose de la matriz **E**
$\|\|.\|\|_2$	Norma euclidiana o norma ℓ_2
$.^T$	Transpuesta
$\mathbf{I}_{r\times r}$	Matriz identidad de dimensión $r \times r$
$\mathbf{x}_j$	j-ésimo patrón de entrada
$\mathbf{t}_j$	j-ésimo patrón de salida (salida deseada u objetivo)
$\mathbf{w}_i$	vector de pesos de entrada del i-ésimo nodo oculto
b_i	Bias del i-ésimo nodo oculto
B	Matriz de pesos de salida en ELM (solución ELM)
$\mathbf{B}_{\epsilon}$	Matriz de pesos de salida en MEC-ELM (solución MEC-ELM)
H	Matriz de salida de los nodos ocultos
Q	Matriz de salida de los nodos de compresión
$\mathbf{H}_{oc}$	Matriz de salida de los nodos ocultos que son nodos compresión
$\mathbf{H}_{o}$	Matriz de salida de los nodos ocultos que no son nodos de compresión
T	Matriz de salidas deseadas
D	Matriz de pesos de salida de los nodos de compresión
W	Matriz auxiliar de MEC-ELM
Z	Matriz auxiliar de MEC-ELM
$\mathbf{V}_{oc}$	Matriz auxiliar de MEC-ELM
$\mathbf{W}_{oc}$	Matriz auxiliar de MEC-ELM
N	Número de datos de entrenamiento
n	Dimensión del espacio de entrada

L	Número de nodos ocultos
q	Número de nodos de compresión
m	Dimensión del espacio de salida o número de clases
C	Constante de regularización
K	Número de particiones del conjunto de entrenamiento en validación cruzada
R	Número de parámetros a probar en validación cruzada
$\Psi(.)$	Función auxiliar de MEC-ELM
$\Re$	Conjunto de los números reales
ACC	Exactitud
SEN	Sensibilidad
ESP	Especificidad
ACC	Precisión
AUC	Área bajo la curva ROC

Lista de Abreviaturas

BD: *Big Data*

BP: *Back-Propagation*

ELM: *Extreme Learning Machine*

I-ELM: *Incremental ELM*

KKT: Karush-Kunh-Tucker

MEC-ELM: Máquina de entrenamiento comprimido basada en ELM

OOK: *On Off Keying*

OS-ELM: *Online Secuential Extreme Learning Machine*

QRI-ELM: *QR Factorization Based Incremental-ELM*

SLFN: Single hidden Layer Feedforward Network

SVD: *Singular Value Decomposition*

Capítulo 1

Introducción

Con el desarrollo de las nuevas tecnologías de la información y las comunicaciones se ha disparado la cantidad de datos que se generan como resultado de las diversas actividades modernas. Este fenómeno tiene gran impacto en casi todos los sectores de la sociedad y se incluye dentro de lo que hoy en día se conoce como "*Big Data*" (Torrecilla y Romo, 2018).

En su gran mayoría las actividades modernas generan datos que contienen información potencialmente valiosa para la toma de decisiones, dicha información representa un valor de uso que debe ser capturado y aprovechado para mejorar un determinado proceso (Wang *et al.*, 2016). Por otro lado, el volumen, la velocidad y la variabilidad con la que se generan los datos actualmente, hacen que capturar dicho valor de uso sea un importante desafió tecnológico que debe ser abordado mediante la investigación y diseño de potentes herramientas para el tratamiento de datos (Sivarajah *et al.*, 2017). En este contexto han venido cobrando importancia aquellos modelos que aprenden a partir de muestras o técnicas de aprendizaje automático entre las que se destacan las redes neuronales.

Las redes neuronales artificiales son esquemas con inherente capacidad de procesamiento paralelo que se aplican con gran éxito en la resolución de complejos problemas de aproximación y clasificación en los que la información se presenta de forma masiva, imprecisa y distorsionada (Brio y Sanz, 2006). Estos esquemas permiten el procesamiento de grandes cantidades de datos siempre y cuando se explote adecuadamente su capacidad de procesamiento paralelo. No obstante, antes deben ser configurados mediante un proceso de entrenamiento que busca ajustar adecuadamente sus parámetros, para esto es necesario utilizar un algoritmo de aprendizaje y un conjunto de entrenamiento. En la actualidad los conjuntos de entrenamiento pueden ser muy grandes y como consecuencia el proceso de entrenamiento puede

tornarse muy lento o incluso inabordable con algoritmos de aprendizaje iterativos, este problema ha sido motivo de numerosos esfuerzos investigativos en procura de crear métodos más eficientes. Una de las propuestas más importantes en este sentido es el algoritmo "*Extreme Learning Machine*" (ELM) propuesto por Huang *et al.* (2006).

En los últimos años ELM ha despertado mucho interés en la comunidad científica debido a que, para un amplio rango de aplicaciones sus prestaciones en términos de generalización y velocidad de aprendizaje son mucho mejores en comparación con las de los tradicionales algoritmos basados en gradiente y, en general mejores que las de las máquinas de vectores soporte y las máquinas de vectores soporte por mínimos cuadrados (Huang *et al.*, 2010, 2012, 2015a).

Por lo anterior, ELM resulta interesante para abordar aplicaciones que involucran grandes conjuntos de entrenamiento. Sin embargo, en este tipo de escenarios se pueden presentar problemas de procesamiento cuando el número de nodos ocultos es muy elevado (Chen *et al.*, 2017a). Entre las iniciativas que buscan resolver dichos problemas se incluyen algunas variantes de tipo paralelo (Heeswijk *et al.*, 2011; Krawczyk, 2016) y distribuido (Xin *et al.*, 2014; Sun *et al.*, 2011) implementadas sobre GPU ("*Graphics Processing Unit*") o usando *Map-Reduce* (Xin *et al.*, 2016). No obstante, en estas propuestas persisten los problemas de procesamiento tipo cuello de botella relacionados con la multiplicación e inversión de grandes matrices.

Los estudios previos en relación a ELM se centran en hacer que este algoritmo de aprendizaje sea más eficiente y adecuado para aplicaciones específicas, entre estas mejoras se incluye una versión de ELM indicada para problemas con grandes conjuntos de entrenamiento (Huang *et al.*, 2015a), la cual ha sido ampliamente utilizada en aplicaciones de *Big Data* (Cao y Lin, 2015). En la literatura pueden encontrarse numerosas iniciativas que buscan mejorar esta versión. Sin embargo, en su gran mayoría dichas iniciativas son aportes a nivel de implementación que no consideran la dimensión de las matrices involucradas en el entrenamiento. Con base en lo anterior, en este documento se presenta el diseño y prueba de un nuevo algoritmo de aprendizaje para redes SLFN basado en las propiedades de ELM, el cual actúa comprimiendo la información proveniente de la capa oculta por medio de un subconjunto de nodos ocultos denominados nodos de compresión, esto permite disminuir el tamaño de las matrices involucradas en el entrenamiento y la complejidad computacional en relación a la solución de ELM para grandes conjuntos de entrenamiento.

1.1 Contribución Original

Este trabajo de investigación permite avanzar en el conocimiento de ELM y su aporte es básicamente el diseño y prueba de un nuevo algoritmo de entrenamiento para redes SLFN basado en sus propiedades. Este nuevo algoritmo denominado MEC-ELM permite abordar aplicaciones de clasificación que involucran grandes cantidades de datos de una manera más eficiente, reduciendo considerablemente la complejidad computacional y el tiempo de entrenamiento, algo que resulta crucial dados los modernos desafíos que impone el fenómeno de *Big Data*. Adicionalmente se plantean tres implementaciones distintas; las dos primeras permiten manejar el costo computacional de MEC-ELM en procesos de validación cruzada mediante la reutilización de matrices y la tercera se plantea para calcular la solución de MEC-ELM en forma distribuida y usando grandes "*clúster*" de computadoras, se trata de una implementación basada en la metodología *Map-Reduce* propuesta por Google en el 2006 para el tratamiento de masivas cantidades de datos (Lämmel, 2008).

1.1.1 Publicaciones relacionadas con la investigación

Castro, F. M. y Jojoa, P. E.,*Entrenamiento Comprimido Basado en Extreme Learning Machine*, aceptado para publicación en la revista internacional "Información Tecnológica", ISSN: 0718-0764, a publicarse en el volumen 30 número 4 (última semana de agosto) del año 2019.

Castro, F. M. y Jojoa, P. E., *Entrenamiento de máquinas de aprendizaje extremo con el algoritmo acelerador*, "I+T+C Investigación, Tecnología Y Ciencia", ISSN: 1909-5775, numero 9, 2015.

Castro, F. M. y Jojoa, P. E., *Algoritmo Acelerador Regresivo versión Gamma con Gradiente Local de Error para Entrenamiento de Redes Neuronales Perceptron Multicapa*, "Gerencia Tecnológica Informática" ISSN: 1657-8236, volumen 14, número 39, 65-74, 2015.

1.2 Estructura de la tesis

El capítulo 2 presenta los fundamentos teóricos de las redes neuronales haciendo especial énfasis en las redes de propagación hacía adelante de una sola capa oculta y el algoritmo "*Extreme Learning Machine*".

En el capítulo 3 se detalla el planteamiento matemático y análisis computacional de MEC-ELM. En este mismo capítulo se presentan los resultados de una serie de pruebas preliminares relacionadas con clasificación de clases gaussianas traslapadas.

En el capítulo 4 se presenta una implementación de MEC-ELM que utiliza validación cruzada de K iteraciones. Con base en esta implementación se desarrolla una serie de experimentos que buscan esclarecer la influencia de los nodos de compresión. Seguidamente, se presenta el desarrollo de varias pruebas de eficiencia basadas en simulación que comparan el rendimiento de MEC-ELM con el de ELM para diferentes problemas de clasificación binaria y multiclase.

En el capítulo 5 se presenta un análisis comparativo entre MEC-ELM y ELM para dos problemas relacionados con clasificación de enlaces en una red de comunicaciones ópticas. Este análisis se realiza utilizando una implementación de MEC-ELM basada en el método de retención.

El capítulo 6 detalla el diseño y puesta a prueba de una implementación de MEC-ELM basada en *Map-Reduce*.

Finalmente, en el capítulo 7 se exponen las conclusiones y se plantean algunos trabajos futuros.

Capítulo 2

Fundamentos Teóricos

2.1 Redes Neuronales Artificiales

Las redes neuronales artificiales son esquemas de procesamiento inspirados en la estructura de los sistemas nerviosos biológicos. Estos esquemas han sido ampliamente utilizados en diversos campos científicos e ingenieriles tales como la minería de datos, diagnosis y clasificación, procesamiento de señales, diseño de sistemas de control, reconocimiento del habla, visión de objetos inmersos en el espacio natural, resolución de ecuaciones y en general todos aquellos problemas donde la información se presenta de forma masiva imprecisa y distorsionada. Estos esquemas procesan la información en múltiples elementos llamados neuronas artificiales, los cuales se agrupan siguiendo un patrón de conexiones para conformar una red. Cada conexión tiene asociado un parámetro denominado peso sináptico cuyo valor debe ser ajustado dependiendo del problema que se pretende solucionar (Jaddi *et al.*, 2015)(Ata, 2015).

Por otro lado, las redes neuronales no aprenden por sí solas a aproximar funciones o a clasificar muestras, para esto es necesario un proceso de aprendizaje que se centra en determinar (utilizando un algoritmo de entrenamiento) una configuración de pesos sinápticos que asocien correctamente un conjunto de patrones. Este proceso puede ser de dos tipos: supervisado o no supervisado. El primero consiste en presentar a la red un conjunto de entradas junto a sus respectivas salidas deseadas, para posteriormente ajustar los pesos sinápticos de modo que se optimice una función de error. Por su parte, en el entrenamiento no supervisado se carece de un maestro externo que indique si la red está operando correcta o incorrectamente, por lo que ésta debe realizar un proceso de auto organización que le permita incorporar dentro de su estructura de pesos, aquellos rasgos comunes, regularidades, correlaciones o categorías que vaya descubriendo por sí misma durante el entrenamiento (Brio y Sanz, 2006).

Las redes neuronales se clasifican atendiendo a dos de sus características más notables: el patrón de conexiones y el tipo de aprendizaje que utilizan, en la figura 2.1 se presentan algunas de las arquitecturas más populares. Mención aparte requieren las redes multicapa de propagación hacia adelante dentro de las cuales se incluyen las redes SLFN (*Single hidden Layer Feedforward Network*) en torno a las cuales gira la temática central de este documento.

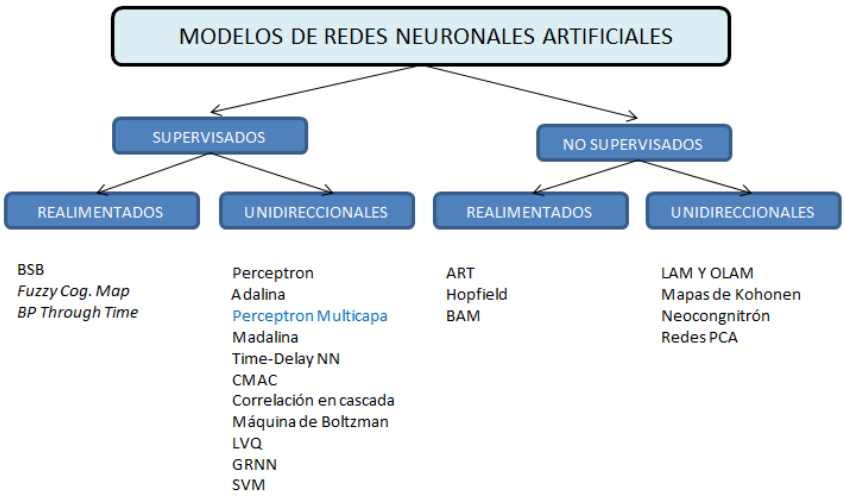


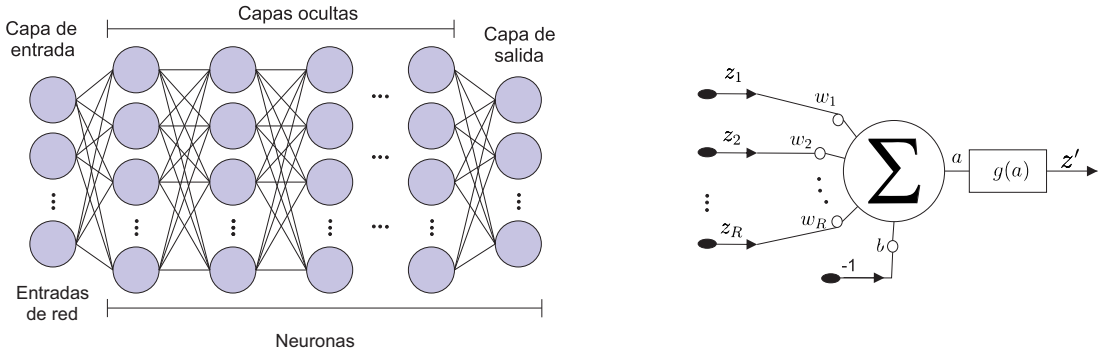
Figura 2.1. Clasificación de las Redes Neuronales Artificiales.

Las redes multicapa de propagación hacia adelante son modelos de aprendizaje supervisado conformados por varios nodos que se agrupan formando capas, dichas capas a su vez se conectan a otras capas sin ningún tipo de realimentación tal y como se muestra en la figura 2.2a. La primera capa es la capa de entrada, las capas intermedias son llamadas capas ocultas y la última capa es la capa de salida. Los nodos de la capa de entrada corresponden a las entradas de la red y los otros nodos (nodos ocultos y nodos de salida) son neuronas artificiales (Brio y Sanz, 2006). El modelo estándar de una neurona artificial se presenta en la figura 2.2b y su salida $z' \in \mathbb{R}$ se define matemáticamente como:

$$z' = g\left(a\right) = g\left(\sum_{u=1}^{R} w_u z_u + b\right), \tag{2.1}$$

donde $z_u \in \mathbb{R}$ es la u-ésima entrada, $w_u \in \mathbb{R}$ el peso sináptico asociado a la u-ésima conexión de entrada, $b \in \mathbb{R}$ un parámetro denominado bias y $g(.)$ una función de activación. El bias o umbral, es 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.



a) b)

Figura 2.2. Esquema general y neurona de una red de propagación hacia adelante: a) Arquitectura b) Neurona artificial

2.1.1 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 las redes neuronales son capaces de brindar respuestas adecuadas ante patrones jamás utilizados en el entrenamiento.

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, dado que pueden incurrir en sobre aprendizaje, apartándose del verdadero propósito que consiste en generalizar. El sobre aprendizaje 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 muestras de entrenamiento (aprendiendo incluso el ruido presente en ellos), de forma que no consiguen generalizar lo suficientemente bien y fallan. Por esta razón debe seleccionarse un conjunto de pesos adecuado para que el error de generalización sea mínimo, a costa incluso de aumentar el error con

los datos de entrenamiento. Para esto se utiliza usualmente un procedimiento, denominado validación cruzada (Aras y Kocakoç, 2016).

2.2 Consideraciones en relación a las redes SLFN

Las redes SLFN (*Single hidden Layer Feedforward Network*) son redes multicapa de propagación hacia adelante de una sola capa oculta, estos modelos permiten abordar problemas complejos de clasificación y aproximación funcional de una manera eficaz y relativamente simple. El modelo matemático de una red SLFN de L nodos ocultos como la mostrada en la figura 2.3 esta dado por

$$z_v = f(\sum_{i=1}^{L} \beta_{i,v} h_i + \theta_v) = f(\sum_{i=1}^{L} \beta_{v,i} g(\sum_{u=1}^{n} w_{u,i} x_u - b_i) + \theta_v) \text{ para } v = 1, 2, ..., m, \quad (2.2)$$

donde:

$z_v \in \mathbb{R}$: salida del v-ésimo nodo de salida.
$h_i \in \mathbb{R}$: salida del i-ésimo nodo oculto.
$\beta_{i,v} \in \mathbb{R}$: peso sináptico entre el i-ésimo nodo oculto y el v-ésimo nodo de salida.
$w_{u,i} \in \mathbb{R}$: peso sináptico entre el u-ésimo nodo de entrada y el i-ésimo nodo oculto.
$\theta_v \in \mathbb{R}$: bias del v-ésimo nodo de salida.
$b_i \in \mathbb{R}$: bias del i-ésimo nodo oculto.
$f(.)$: función de activación de los nodos de la capa de salida.
$g(.)$: función de activación de los nodos de la capa oculta.

Dos de los resultados teóricos más importantes en el campo de las redes neuronales, están relacionados con la capacidad de aproximación y clasificación de las redes SLFN. En relación a dichas capacidades se han comprobado los siguientes enunciados:

- **Capacidad de aproximación**: cualquier función continua $f_p(.)$ puede ser aproximada hasta un nivel deseado por una red SLFN con funciones de activación no polinomiales en los nodos ocultos y funciones de activación lineal en los nodos de salida. Siempre y cuando cuente con un número de nodos ocultos apropiado (Leshno *et al.*, 1993).

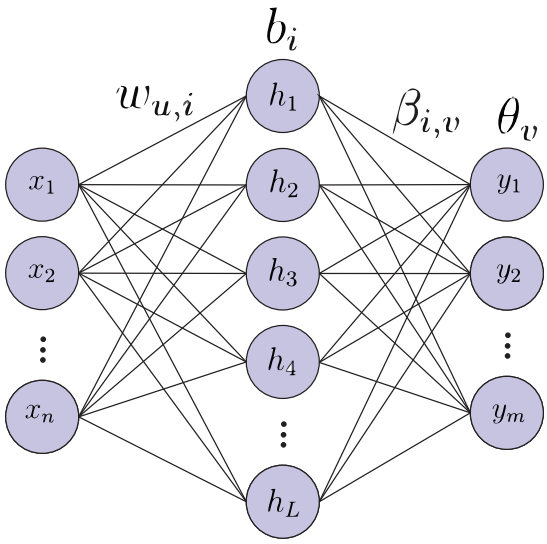


Figura 2.3. Modelo de una red SLFN estándar

- **Capacidad de clasificación**: si una SLFN puede aproximar cualquier función continua $f_p(\mathbf{.})$ entonces también puede diferenciar entre cualquier grupo de regiones disyuntas (Huang *et al.*, 2000).

No existe un método para calcular el número apropiado de nodos ocultos, esto es un aspecto que no está esclarecido en la literatura, no obstante, Huang y Babri (1998) prueban que una red SLFN con al menos L nodos ocultos no lineales y nodos de salida lineales, puede aprender sin error al menos L observaciones distintas a partir de un conjunto finito de aprendizaje. Cabe destacar que estos resultados fueron obtenidos bajo la concepción de que todos los pesos sinápticos de la red debían ser ajustados. Sin embargo, Huang *et al.* (2006); Huang y Chen (2007, 2008) prueban rigurosamente que una red SLFN puede mantener sus capacidades de aproximación y clasificación incluso si los pesos y bias se seleccionan aleatoriamente.

2.2.1 Algoritmos de aprendizaje para redes SLFN

Tradicionalmente, las redes SLFN se han entrenado con el algoritmo *Backpropagation* o BP cuyo descubrimiento representa uno de los logros más importantes en el desarrollo de las redes neuronales, pese a ser popularmente conocido como un algoritmo ineficiente debido principalmente a su lenta convergencia y posibilidad de estancamiento en mínimos locales (Dai y Liu, 2012; Ata, 2015; Wang *et al.*, 2015). No

obstante, se trata de un algoritmo de muy baja complejidad computacional y alta estabilidad, por esta razón, mejorar la velocidad del algoritmo BP y evitar el estancamiento prematuro ha sido un campo muy activo de investigación desde hace varios años (Rady, 2011; Lima *et al.*, 2015; Wilamowski y Yu, 2010).

El algoritmo BP permite actualizar los pesos sinápticos de la red siguiendo un esquema basado en el método de gradiente descendiente cuya convergencia puede requerir de muchos pasos. Es importante destacar que este proceso se hace capa a capa (permite distribuir la carga computacional), muy diferente a cualquier sistema generalizado, en el que se actualicen todos de pesos sinápticos de la red siguiendo el mismo método de gradiente descendiente (Zhang *et al.*, 2015). Ahora bien, para mejorar el proceso de optimización, se puede utilizar el método de Newton, pero esto solo funciona cuando es posible hacer aproximaciones cuadráticas de la superficie de error, de otro modo el entrenamiento se torna altamente inestable. El método de Newton involucra el uso de la matriz hessiana, con lo que se logra una reducción significativa de pasos en la convergencia del proceso, pero todo atado a la naturaleza misma del problema, es decir, que la función de optimización sea continua y diferenciable dos veces, lo que a su vez implica un mayor coste computacional (Yu y Wilamowski, 2011). Para disminuir la complejidad computacional del método de Newton, se introdujo el método de Gauss Newton, el cual también disminuye el número de pasos en el proceso de optimización (en comparación con el algoritmo BP), pero en lugar de la matriz hessiana, usa la matriz jacobbiana. No obstante, hereda el problema de estabilidad que tiene método de Newton. Para solucionar dicho problema, se propuso el algoritmo de Levenberg-Marquartd, que utiliza la matriz jacobbiana y ademas incluye un factor de ajuste para que el sistema siempre sea positivo definido, asegurando que tenga tasas de convergencia. El problema de esto es que implica una pérdida de generalidad en la solución y por lo tanto una mayor varianza (Nawi *et al.*, 2013). Entre las iniciativas que pretenden resolver este inconveniente se incluyen las redes con regularización bayesiana (Ticknor, 2013). No obstante, estas técnicas enfrentan grandes desafíos en cuanto a convergencia y escalabilidad, particularmente en el tratamiento de grandes conjuntos de datos (Chandra *et al.*, 2019).

Por otro lado, los algoritmos mencionados hasta el momento implican diferentes formas de estimar el gradiente de la superficie de error para posteriormente recorrer el espacio de pesos en la dirección opuesta. Sin embargo, al igual que en el algoritmo BP, dicha superficie puede tener muchos mínimos locales y el entrenamiento puede quedar

estancado en uno de ellos prematuramente (Mukherjee y Routroy, 2012). Algunas de las propuestas más sobresalientes para superar dicho problema implican el uso de métodos híbridos, los cuales combinan algoritmos basados en gradiente con técnicas meta heurísticas como los algoritmos genéticos, la optimización por nube de partículas o lógica difusa (Huang *et al.*, 2015b; Ata, 2015). En general, estas últimas propuestas permiten encontrar buenas soluciones, sin embargo, el costo a pagar es un tiempo de convergencia muy alto.

Huang *et al.* (2006) propuso un nuevo método para entrenamiento de redes SLFN conocido como ELM (*Extreme Learning Machine*), se trata de un eficiente algoritmo que supera las principales limitantes de los algoritmos basados en gradiente, como son la lenta convergencia y el estancamiento en mínimos locales alcanzando altos niveles de generalización en menores tiempos de procesamiento. El principio de funcionamiento de ELM consiste en seleccionar aleatoriamente los pesos de la capa oculta y calcular una solución analíticamente (en el sentido de los mínimos cuadrados) solo para los pesos de la capa de salida (Mohapatra *et al.*, 2015; Anam y Al-Jumaily, 2017).

Varios estudios teóricos y de aplicación se han desarrollado en relación a ELM desde la pasada década, como resultado en la literatura actual se pueden encontrar varias variantes de la versión original desarrolladas en procura de hacer que ELM sea más eficiente y adecuado para aplicaciones específicas (Huang *et al.*, 2015a). Entre estas variantes se incluyen implementaciones secuenciales (Liang *et al.*, 2006; Ye *et al.*, 2013; Zhao *et al.*, 2012), adaptaciones para trabajo con datos desbalanceados (Zong *et al.*, 2013), versiones de ELM para trabajo en entornos ruidosos (Man *et al.*, 2011, 2012), entre otras que buscan superar alguna de sus problemáticas relacionadas principalmente con la gran cantidad de operaciones matemáticas necesarias para la inversión de matrices y al hecho de que ELM requiere muchos nodos en la capa oculta (Huang *et al.*, 2008; Luo *et al.*, 2014; Ye y Qin, 2015; Yu y Deng, 2012; Cao *et al.*, 2012). Superar tales problemas es un activo campo de investigación en la actualidad, puesto que implican serias limitaciones al uso de ELM en aplicaciones que involucran grandes cantidades de datos (Grigorievskiy *et al.*, 2016).

2.3 ELM (*Extreme Learning Machine*)

ELM propuesto por Huang *et al.* (2006) es un algoritmo para redes SLFN y actualmente un importante paradigma conceptual y computacional dentro de las redes neuronales, puesto que su velocidad de procesamiento y su rendimiento en términos de generalización son muchísimo mejores en comparación con los tradicionales algoritmos basados en gradiente.

ELM hace un mapeo aleatorio del espacio de entrada hacia otro espacio usualmente de mayor dimensión llamado espacio de características ELM. Este mapeo es realizado por los nodos de la capa oculta, los cuales reciben el nombre de neuronas aleatorias dado que sus pesos y bias son seleccionados aleatoriamente. En el espacio de características ELM los datos se ajustan mediante un proceso de optimización basado en mínimos cuadrados (ver figura 2.4).

Los principios que rigen la determinación de los parámetros de entrenamiento en ELM son los siguientes:

Principio 1. Los pesos y bias de las neuronas ocultas de una red SLFN con funciones de activación no lineales y continuas a trozos, pueden ser generados aleatoriamente a partir de cualquier función de probabilidad continua, tales neuronas son independientes de los datos de entrenamiento y como tal también del ambiente de aprendizaje (Huang, 2014).

Principio 2. Para propósitos de estabilidad y rendimiento en términos de generalización, se deben minimizar el error de entrenamiento y la norma de los pesos de salida (Huang, 2014), acorde con la teoría de generalización para redes neuronales propuesta por Bartlett (1998).

Principio 3. Los nodos de salida no deben tener bias o los bias deben ser cero (Huang, 2014).

2.3.1 Modelado matemático de ELM

Un modelo ELM es una red SLFN entrenada con el algoritmo *Extreme Learning Machine* y, como se ha mencionado anteriormente, los pesos y bias de dichos modelos

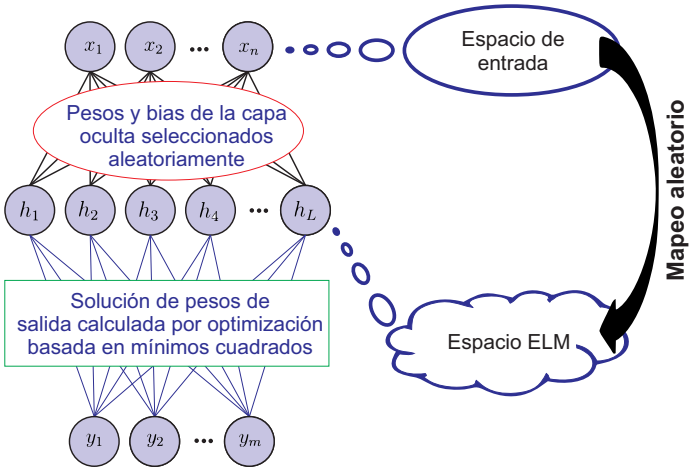


Figura 2.4. Mapeo de características ELM

son seleccionados aleatoriamente. Con base en lo anterior, dada una entrada arbitraria $\mathbf{x} \in \mathbb{R}^n$, la salida de una modelo ELM con L nodos ocultos, se puede expresar como

$$f_L(\mathbf{x}) = \sum_{i=1}^{L} \boldsymbol{\beta}_i g(\mathbf{w}_i \cdot \mathbf{x} + b_i) \tag{2.3}$$

donde, $\mathbf{w}_i = \begin{bmatrix} w_{i,1} & w_{i,2} & \dots & w_{i,m} \end{bmatrix}^T$ es el vector de pesos del i-ésimo nodo oculto, $\boldsymbol{\beta}_i = \begin{bmatrix} \beta_{i,1} & \beta_{i,2} & \dots & \beta_{i,m} \end{bmatrix}^T$ es el vector de pesos entre el i-ésimo nodo oculto y la capa de salida, b_i el bias del i-ésimo nodo oculto y, $g(.)$ es la función de activación de los nodos ocultos.

Ahora bien, sea $\eta = \{\{(\mathbf{x}_j, \mathbf{t}_j)\}_{j=1}^{N} | \mathbf{x}_j \in \mathbb{R}^n, \mathbf{t}_j \in \mathbb{R}^m\}$ un conjunto de N entradas con sus respectivas salidas deseadas. Entonces, con base en la ecuación 2.3, la salida para todas las muestras del conjunto η puede ser expresada como una matriz $\mathbf{F} \in \mathbb{R}^{N \times m}$ tal que

$$\mathbf{F} = \mathbf{HB}, \tag{2.4}$$

donde, $\mathbf{B} = \begin{bmatrix} \boldsymbol{\beta}_1 & \boldsymbol{\beta}_2 & \dots & \boldsymbol{\beta}_L \end{bmatrix}^T$ es la matriz de pesos de salida y $\mathbf{H}$ es la matriz de salida de la capa oculta dada por

$$\mathbf{H} = \begin{bmatrix} g(\mathbf{w}_1 \cdot \mathbf{x}_1 + b_1) & \cdots & g(\mathbf{w}_L \cdot \mathbf{x}_1 + b_L) \\ \vdots & \cdots & \vdots \\ g(\mathbf{w}_1 \cdot \mathbf{x}_N + b_1) & \cdots & g(\mathbf{w}_L \cdot \mathbf{x}_N + b_L) \end{bmatrix}. \tag{2.5}$$

De modo que, entrenar la red SLFN es equivalente a encontrar la solución por mínimos cuadrados del sistema lineal

$$\mathbf{HB} = \mathbf{T}, \tag{2.6}$$

donde

$$\mathbf{T} = \begin{bmatrix} \mathbf{t}_1 & \mathbf{t}_2 & ... & \mathbf{t}_N \end{bmatrix}^T_{N \times m}, \tag{2.7}$$

es la matriz de salidas deseadas u objetivos.

Solución original

A menudo el sistema lineal planteado en la ecuación 2.7 es sobre determinado ($N > L$) y no se puede resolver, por lo que con frecuencia se usa una solución en el sentido de los mínimos cuadrados que se obtiene al minimizar la norma ℓ_2 del error de entrenamiento. De este modo, si $\hat{\mathbf{B}} \in \mathbb{R}^{L \times m}$ es cualquier configuración posible de pesos de salida, entonces:

$$||\mathbf{HB} - \mathbf{T}||_2^2 = \min_{\hat{\mathbf{B}}} ||\mathbf{H}\hat{\mathbf{B}} - \mathbf{T}||_2^2, \tag{2.8}$$

de donde surge la solución original de ELM dada por

$$\mathbf{B} = \mathbf{H}^\dagger \mathbf{T} \tag{2.9}$$

siendo $\mathbf{H}^\dagger$, la inversa generalizada de *Moore-Penrose* de la matriz **H**, que cumple con los axiomas presentados en la tabla 2.1.

Num.	Axioma
1)	$\mathbf{H}\mathbf{H}^{\dagger}\mathbf{H} = \mathbf{H}$
2)	$\mathbf{H}^{\dagger}\mathbf{H}\mathbf{H}^{\dagger} = \mathbf{H}^{\dagger}$
3)	$(\mathbf{H}\mathbf{H}^{\dagger})^T = \mathbf{H}\mathbf{H}^{\dagger}$
4)	$(\mathbf{H}^{\dagger}\mathbf{H})^T = \mathbf{H}^{\dagger}\mathbf{H}$

Tabla 2.1. Axiomas de la inversa generalizada de *Moore-Penrose*.

La solución presentada en la ecuación 2.9 es única, y además, tiene la norma más pequeña de pesos entre todas las soluciones por mínimos cuadrados. También es importante destacar, que (Huang y Chen, 2007, 2008) prueban rigurosamente que ELM conserva las capacidades de aproximación universal y clasificación universal pese a seleccionar aleatoriamente los pesos y bias de la capa oculta.

Por otro lado, para calcular $\mathbf{H}^{\dagger}$ pueden usarse varios métodos, entre los que se destacan el de proyección ortogonal y descomposición en valores singulares SVD (*Singular Value Decomposition*). El método SVD permite calcular $\mathbf{H}^{\dagger}$ en todos los casos. Por su parte, el método de proyección ortogonal calcula $\mathbf{H}^{\dagger}$ como $\mathbf{H}^{\dagger} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T$, pero este método tiene problemas cuando la matriz $\mathbf{H}^T\mathbf{H}$ es singular o tiende a ser singular.

Solución regularizada

En aprendizaje automático es muy habitual usar términos de regularización para mejorar la generalización de un modelo, dichos términos se adicionan a la función de costo para minimizar o penalizar el valor de los parámetros. Existen varios tipos de regularización, entre estos ℓ_1, ℓ_2 y *Elastic Net* (Carrasco *et al.*, 2016), pero en el contexto de ELM la más común es la regularización ℓ_2 o *Ridge*. La regularización ℓ_2 permite minimizar el error de entrenamiento y a la vez la norma de los pesos de acuerdo con el siguiente problema de optimización:

$$\begin{aligned} &\text{Minimizar: } L_{P_{ELM}} = \frac{1}{2}\|\mathbf{B}\|_2^2 + C\frac{1}{2}\sum_{j=1}^{N}\|\boldsymbol{\xi}_j\|_2^2 \\ &\text{Sujeto a: } \mathbf{h}(\mathbf{x}_j)\mathbf{B} = \mathbf{t}_j - \boldsymbol{\xi}_j \quad j = 1, 2, ..., N. \end{aligned} \tag{2.10}$$

donde $\mathbf{h}(\mathbf{x}_j) = [g(\mathbf{w}_1 \cdot \mathbf{x}_j + b_1)...g(\mathbf{w}_L \cdot \mathbf{x}_j + b_L)]^T$ y $\boldsymbol{\xi}_j$ el vector de error de salida para la entrada $\mathbf{x}_j$ definido como $\boldsymbol{\xi}_j = [\xi_{j,1}, \xi_{j,2}, ..., \xi_{j,m}]^T$.

La solución del problema de optimización de la ecuación 2.10 conduce a dos formas distintas del algoritmo ELM, las cuales están indicadas tanto para problemas de clasificación como de regresión. Ahora bien, con base en el teorema de Karush-Kunh-Tucker (KKT) (Suárez, 2016), resolver el problema dado por 2.10 equivale a encontrar la solución del siguiente problema de optimización

$$L_{D_{ELM}} = \frac{1}{2}\|\mathbf{B}\|_2^2 + C\frac{1}{2}\sum_{j=1}^{N}\|\boldsymbol{\xi}_j\|_2 - \sum_{j=1}^{N}\sum_{i=1}^{m}\alpha_{j,i}(\mathbf{h}(\mathbf{x}_j)\boldsymbol{\beta}_i - t_{j,i} + \xi_{j,i}) \tag{2.11}$$

donde $\alpha_{j,i}$ para $j = 1, 2, ..., N$ e $i = 1, 2, ..., m$, son los denominados multiplicadores de Lagrange y $\boldsymbol{\beta}_i = [\beta_{i,1}, \beta_{i,2}, ..., \beta_{i,m}]^T$ es el vector de pesos entre la capa oculta y el i-ésimo nodo de salida, que a su vez es la i-ésima fila de la matriz $\mathbf{B}$ presentada en la ecuación 2.4 (Deng *et al.*, 2009; Huang *et al.*, 2012).

A continuación, se aplican las condiciones de KKT:

$$\frac{\partial L_{D_{ELM}}}{\partial \boldsymbol{\beta}_i} = 0 \rightarrow \boldsymbol{\beta}_i = \sum_{j=1}^{N}\alpha_{j,i}\mathbf{h}(\mathbf{x}_j)^T \rightarrow \mathbf{B} = \mathbf{H}^T\mathbf{A} \tag{2.12}$$

$$\frac{\partial L_{D_{ELM}}}{\partial \boldsymbol{\xi}_j} = 0 \rightarrow \boldsymbol{\alpha}_j = C\boldsymbol{\xi}_j \quad j = 1, 2, ..., N \tag{2.13}$$

$$\frac{\partial L_{D_{ELM}}}{\partial \boldsymbol{\alpha}_j} = 0 \rightarrow \mathbf{h}(\mathbf{x}_j)\mathbf{B} - \mathbf{t}_j + \boldsymbol{\xi}_j = \mathbf{0} \quad j = 1, 2, ..., N. \tag{2.14}$$

donde $\boldsymbol{\alpha}_j = [\alpha_{j,1}, \alpha_{j,2}, ..., \alpha_{j,m}]^T$ y $\mathbf{A} = [\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, ..., \boldsymbol{\alpha}_j, ..., \boldsymbol{\alpha}_N]^T$.

Las ecuaciones 2.12 y 2.13 son resultado de aplicar la primera condición KKT y la ecuación 2.14 el resultado de aplicar la denominada condición complementaria (segunda condición KKT). Con base en estos resultados, se pueden obtener dos soluciones diferentes atendiendo a criterios de eficiencia y al tamaño del conjunto de entrenamiento:

- **Caso 1**: Para pequeños conjuntos de entrenamiento

  Si se sustituye 2.12 y 2.13 en 2.14 entonces

$$\left(\frac{\mathbf{I}}{C} + \mathbf{H}\mathbf{H}^T\right)\mathbf{A} = \mathbf{T}. \tag{2.15}$$

Ahora, a partir de 2.12 y 2.15 se obtiene

$$\mathbf{B} = \mathbf{H}^T \left(\frac{\mathbf{I}}{C} + \mathbf{H}\mathbf{H}^T \right)^{-1} \mathbf{T} \tag{2.16}$$

- **Caso 2**: Para grandes conjuntos de entrenamiento

  Si el número de muestras de entrenamiento es mucho más grande que la dimensionalidad del mapeo del espacio de características, esto es, $N > L$ se tiene la siguiente solución alternativa.

  De las expresiones 2.12 y 2.13 se puede encontrar que

$$\mathbf{B} = C\mathbf{H}^T\mathbf{E} \quad \text{con} \quad \mathbf{E} = [\boldsymbol{\xi}_1, \boldsymbol{\xi}_2, ..., \boldsymbol{\xi}_N]^T \tag{2.17}$$

$$\mathbf{E} = \frac{1}{C}(\mathbf{H}^T)^\dagger \mathbf{B} \tag{2.18}$$

  Finalmente, sustituyendo 2.18 en 2.14 se tiene

$$\mathbf{H}\mathbf{B} - \mathbf{T} + \frac{1}{C}(\mathbf{H}^T)^\dagger \mathbf{B} = \mathbf{0} \tag{2.19}$$

$$\mathbf{H}^T \left(\mathbf{H} + \frac{1}{C}(\mathbf{H}^T)^\dagger \right) \mathbf{B} = \mathbf{H}^T\mathbf{T} \tag{2.20}$$

$$\mathbf{B} = \left(\frac{\mathbf{I}}{C} + \mathbf{H}^T\mathbf{H} \right)^{-1} \mathbf{H}^T\mathbf{T} \tag{2.21}$$

En teoría, no existe ningún requerimiento especifico en relación al conjunto de entrenamiento, puesto que las expresiones 2.16 y 2.21 pueden usarse sin importar el número de muestras. Sin embargo, el costo computacional de estas dos soluciones es muy diferente dependiendo de N. (Huang *et al.*, 2012). En este sentido, seria impráctico abordar aplicaciones en las que $N \gg L$ utilizando la solución 2.16 ya que involucraría la inversión de una matriz de tamaño $N \times N$. Para este tipo de aplicaciones es mucho más indicado el uso de la solución 2.21 que involucra la inversión de una matriz de tamaño $L \times L$.

La solución regularizada de ELM para grandes conjuntos de entrenamiento supone un importante adelanto en términos de generalización respecto a la solución original de ELM y otras variantes (incluso más recientes) que se derivan de esta última (Deng *et al.*, 2009), como por ejemplo: *Incremental ELM* (I-ELM) (Huang *et al.*, 2008), *On Line Secuential* (OS-ELM) (Liang *et al.*, 2006) y *QR Factorization Based Incremental-ELM* (QRI-ELM) (Ye y Qin, 2015). Por estas razones, en las fases experimentales de esta investigación se utiliza la solución planteada en la ecuación 2.21 para hacer las respectivas comparaciones.

2.3.2 Entrenamiento de una red ELM

El algoritmo ELM para entrenamiento de redes SLFN se presenta en la tabla 2.2.

	Entradas:	$\eta = \{(\mathbf{x}_j, \mathbf{t}_j), \mathbf{x}_j \in \mathbb{R}^n, \mathbf{t}_j \in \mathbb{R}^m\}_{j=1}^N \rightarrow$ Conjunto de datos $L \rightarrow$ Número de nodos ocultos $C \rightarrow$ Constante de regularización $g(.) \rightarrow$ Función de activación de los nodos ocultos
	Salidas:	$\{\mathbf{w}_i, b_i\}_{i=1}^L \rightarrow$ Pesos y bias ocultos **B** $\rightarrow$ Solución de ELM (Matriz de pesos de salida)
1		Asignar aleatoriamente los pesos $\mathbf{w}_i$ y los bias b_i para $i = 1, 2, ..., L$.
2		Calcular la matriz de salidas de la capa oculta **H**
3		Calcular los pesos de salida **B** con las expresiones 2.16 o 2.21 dependiendo del respectivo caso.

Tabla 2.2. Entrenamiento de un modelo ELM

Para garantizar la capacidad de aproximación universal de la red ELM, las funciones de activación $g(.)$ de los nodos ocultos deben ser no lineales y continúas definidas a trozos (ver teorema 2.3.2).

En lo que respecta a la constante de regularización C, la importancia de este parámetro no solo está relacionada con el rendimiento en términos de generalización, sino que también hace que el entrenamiento sea más estable numéricamente dado que un apropiado valor asegura que las matrices a invertir sean definidas positivas (Huang *et al.*, 2015a). En la práctica el valor de C se ajusta usando técnicas de prueba y ensayo generalmente basadas en validación cruzada (Kokkinos y Margaritis, 2018). Es importante destacar que estas técnicas requieren mediciones de rendimiento sobre conjuntos de prueba y como tal están estrechamente relacionadas con la aplicación, más adelante en la sección 2.4 cuando se considere a ELM en el contexto de la clasifi-

cación se expondrán algunas de las medidas de rendimiento más importantes en este campo y además algunas de las técnicas de validación cruzada más utilizadas en la práctica.

2.3.3 OS-ELM (*Online Sequential-ELM*)

ELM es un algoritmo que trabaja en lote, esto es, que asume tener todas las muestras de entrenamiento disponibles previamente, por lo que su utilización no está indicada para aplicaciones en las que los datos se presenten de forma secuencial. Buscando superar este inconveniente Liang *et al.* (2006) propuso el algoritmo OS-ELM, el cual es una implementación secuencial de la versión original de ELM que puede trabajar por muestras o por bloques de muestras.

En la formulación de OS-ELM se considera que $\mathbf{rank}(\mathbf{H}) = L$. Bajo esta condición $\mathbf{H}^T\mathbf{H}$ es no singular y $\mathbf{H}^\dagger = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T$, con lo que se puede reescribir la ecuación 2.9 como

$$\hat{\mathbf{B}} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{T}. \tag{2.22}$$

El algoritmo OS-ELM termina siendo el resultado de implementar secuencialmente la ecuación 2.22; se trata de un procedimiento que consta de dos fases: fase de inicialización y fase de aprendizaje secuencial. Para los casos en que la matriz $\mathbf{H}^T\mathbf{H}$ tienda a ser singular, esto puede evitarse seleccionando un número de nodos ocultos L más pequeño o aumentando el número de datos utilizados en la fase de inicialización.

En la fase de inicialización se selecciona un conjunto de N_0 muestras, con las que se calcula "la matriz de salida inicial de la capa oculta" $\mathbf{H}_0$. Para asegurar que $\mathbf{rank}(\mathbf{H}_0) = L$, el número de muestras distintas requeridas para conformar $\mathbf{H}_0$ debe ser al menos igual al número de neuronas ocultas L, en los casos en que las muestras no sean distintas, deben seleccionarse más muestras. La fase de inicialización termina con la obtención de una matriz de pesos de salida inicial $\mathbf{B}^0$ y en seguida comienza la fase de aprendizaje, encargada de actualizar los pesos de salida tomando las muestras de entrada una a una o por bloques (el tamaño de cada bloque puede variar entre uno y otro). El procedimiento general que sigue OS-ELM para entrenar una red SLFN estándar se presenta en el anexo A.

2.3.4 Propiedades de las Redes ELM

A continuación, se revisan algunos resultados teóricos referentes a las redes ELM, entre los que se incluyen aquellos relacionados con la teoría de interpolación, la capacidad de aproximación universal y la capacidad de clasificación universal.

Teoría de interpolación

Huang *et al.* (2006) prueba rigurosamente que dado cualquier conjunto de entrenamiento, siempre existirá una red ELM con funciones de activación infinitamente diferenciables, que asocie los patrones con un error de salida (en el sentido del error cuadrático medio) lo suficientemente pequeño. Y Además, que el número de nodos ocultos de dicha red, no es mayor al número de muestras de entrenamiento distintas y, en el caso de que el número de nodos ocultos sea igual al número de muestras distintas, con probabilidad uno, el error de entrenamiento será cero. Estos resultados se unifican y se enuncian formalmente en el siguiente teorema:

Teorema 2.3.1 *Dado un valor positivo $\epsilon > 0$ arbitrariamente pequeño, una función de activación $g : \mathbb{R} \to \mathbb{R}$ infinitamente diferenciable en cualquier intervalo y N muestras distintas $(\mathbf{x}_j, \mathbf{t}_j) \in \mathbb{R}^n \times \mathbb{R}^m$ seleccionadas de forma arbitraria, existe $L < N$ tal que para cada par $\{\mathbf{w}_i, b_i\}_{i=1}^{L}$ generados aleatoriamente a partir de cualquier intervalo $\mathbb{R}^n \times \mathbb{R}$ de acuerdo con cualquier función de probabilidad continua, con probabilidad uno, $||\mathbf{HB} - \mathbf{T}|| < \epsilon$. Mas aún, si $L = N$, con probabilidad uno $||\mathbf{HB} - \mathbf{T}|| = 0$* (Huang *et al.*, 2015a).

Por consiguiente, desde la perspectiva de la teoría de interpolación, ELM puede ajustar cualquier conjunto de entrenamiento, siempre y cuando el número de nodos ocultos de la red sea lo suficientemente grande.

Capacidad de aproximación universal

Un modelo ELM con arquitectura de red fija, en la que los pesos de salida son determinados a través de mínimos cuadrados, mantiene su capacidad de aproximación universal pese a seleccionar aleatoriamente sus pesos y bias ocultos. Este resultado se formaliza a continuación:

Teorema 2.3.2 *Dada una función $g : \mathbb{R} \to \mathbb{R}$ continua a trozos y no constante, si el espacio generado $\{G(\mathbf{x}, \mathbf{w}, b) : (\mathbf{w}, b) \in \mathbb{R}^n \times \mathbb{R}\}$ es denso en L^2, para cualquier*

función objetivo f continua y cualquier secuencia $\{g_L(\mathbf{x}) = G(\mathbf{x}, \mathbf{w}_L, b_L)\}$ generada aleatoriamente en base a una distribución continua, $\lim_{L\to\infty}\|f - f_L\| = 0$ se mantiene con probabilidad uno si los pesos de salida $\boldsymbol{\beta}_i$ se determinan por mínimos cuadrados para minimizar $\|f(\mathbf{x}) - \sum_{i=1}^{L} \boldsymbol{\beta}_i g_i(\mathbf{x})\|$ (Huang y Chen, 2007, 2008).

Con base en el teorema 2.3.2, no es necesario que las funciones de activación sean continuas y diferenciables. Por consiguiente, la mayoría de las funciones de activación usadas en la práctica cumplen con la condición de aproximación universal, incluyendo a la función escalón (Huang *et al.*, 2015a).

Capacidad de clasificación universal

Teniendo en cuenta que una red ELM puede aproximar cualquier función continua, puede demostrarse también que es capaz de aprender a definir regiones arbitrarias por medio de un hiperplano de decisión, como lo afirma el teorema 2.3.3.

Teorema 2.3.3 *Dado cualquier mapeo $\mathbf{h}(\mathbf{x})$ del espacio de características, si $\mathbf{h}(\mathbf{x})$ es denso en $C(\mathbb{R}^n)$* o en *$C(\Omega)$ donde Ω es un conjunto compacto de $\mathbb{R}^n$, entonces una red SLFN estándar con mapeo $\mathbf{h}(\mathbf{x})$ aleatorio en la capa oculta, puede separar arbitrariamente regiones disyuntas de cualquier forma en $\mathbb{R}^n$ o Ω* (Huang *et al.*, 2012).

Lo que el teorema 2.3.3 establece básicamente es que una red ELM puede aproximar cualquier frontera de decisión compleja en clasificación, si se garantiza que el número de nodos ocultos sea lo suficientemente grande (Huang *et al.*, 2015a).

2.4 ELM para aplicaciones de clasificación

Por lo general, en aplicaciones de clasificación se utilizan conjuntos de entrenamiento $\eta = \{\{(\mathbf{x}_j, \zeta_j)\}_{j=1}^{N} \mid \mathbf{x}_j \in \mathbb{R}^n, \zeta_j \in \{1, 2, ..., m\}\}$ donde ζ_j es una variable de tipo categórico que representa la clase actual de la entrada $\mathbf{x}_j$. Teniendo en cuenta que ELM no puede tratar este tipo de variables, es necesario asegurar (independientemente del preprocesamiento de los datos) que cada ζ_j sea codificada adecuadamente para obtener las salidas deseadas $\mathbf{t}_j \in \mathbb{R}^m$. Comúnmente este proceso se lleva a cabo siguiendo la siguiente codificación

$$t_{j,v} = \begin{cases} 1 \quad si \quad v = \zeta_j \\ \\ 0 \quad si \quad v \neq \zeta_j \end{cases} \qquad v = 1, 2, ..., m \quad j = 1, 2, ..., N, \tag{2.23}$$

las salidas deseadas creadas con esta codificación son vectores unitarios para cada clase, los cuales son ortogonales a los vectores de otras clases, de tal modo que la independencia se mantiene.

Una vez el conjunto de entrenamiento ha sido codificado completamente se procede a entrenar un modelo ELM, el cual habiendo sido entrenado y estando en modo de ejecución puede usarse para predecir la categoría a la cual pertenece una muestra dada. Esto se realiza utilizando una función de decisión. Usualmente la función de decisión es tal que si $(\mathbf{x}, \zeta^{(a)})$ es un patrón de prueba, entonces la clase predicha $\zeta^{(p)}$ esta dada por

$$\zeta^{(p)} = \underset{v \in \{1, 2, ..., m\}}{\arg\max}\ y_v \tag{2.24}$$

donde $\arg\max$ es una función que determina el índice del máximo elemento y y_v es el v-ésimo componente de $f_L(\mathbf{x}) = [y_1, y_2, ..., y_m]$, siendo $f_L(\mathbf{x})$ el vector de salidas de la red ELM que se obtiene usando la ecuación 2.3.

El procedimiento para entrenar un modelo ELM para clasificación o clasificador ELM se presenta en la tabla 2.3 y su configuración en modo de ejecución se ilustra en la figura 2.5.

	Entradas:	$\eta_c = \{\{(\mathbf{x}_j, \mathbf{t}_j)\}_{j=1}^{N} \mid \mathbf{x}_j \in \mathbb{R}^n, \zeta_j \in \{1, 2, ..., m\} \rightarrow$ Conjunto de datos $L \rightarrow$ Número de nodos ocultos $C \rightarrow$ Constante de regularización $g(.) \rightarrow$ Función de activación de los nodos ocultos
	Salidas:	$\{\mathbf{w}_i, b_i\}_{i=1}^{L} \rightarrow$ Pesos y bias ocultos $\mathbf{B} \rightarrow$ Solución de ELM (Matriz de pesos de salida)
1		Obtener las salidas deseadas $\mathbf{t}_j = [t_{j,1} \quad t_{j,2} \quad ... \quad t_{j,m}]^T$ codificando ζ_j de acuerdo con $t_{j,v} = \begin{cases} 1 \quad si \quad v = \zeta_j \\ \\ 0 \quad si \quad v \neq \zeta_j \end{cases} \quad v = 1, 2, ..., m \quad j = 1, 2, ..., N,$
2		Dado $\eta = \{\{(\mathbf{x}_j, \mathbf{t}_j)\}_{j=1}^{N} \mid \mathbf{x}_j \in \mathbb{R}^n, \mathbf{t}_j \in \mathbb{R}^m\}$, entrenar un modelo ELM (proceso especificado en la tabla 2.2).

Tabla 2.3. Entrenamiento de un clasificador ELM

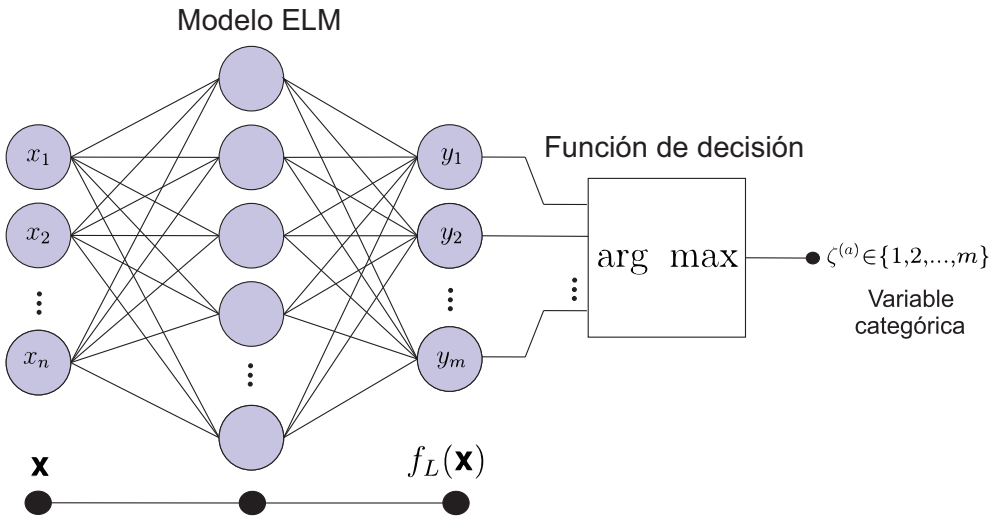


Figura 2.5. Esquema general de un clasificador ELM

Por otro lado, durante el entrenamiento de un clasificador ELM resulta conveniente sacrificar el rendimiento sobre los datos del conjunto de entrenamiento en procura de mejorar la generalización del modelo, esto se logra seleccionando adecuadamente el valor de la constante de regularización C. En la práctica dicha selección se realiza utilizando técnicas de validación cruzada cuya idea fundamental consiste en dividir el conjunto de datos en dos subconjuntos; uno para entrenar (conjunto de entrenamiento) y otro para evaluar el rendimiento de múltiples modelos (conjunto de validación) obtenidos como resultado de probar distintos valores de C; una vez todos los modelos hayan sido evaluados se procede a seleccionar el de mejor rendimiento.

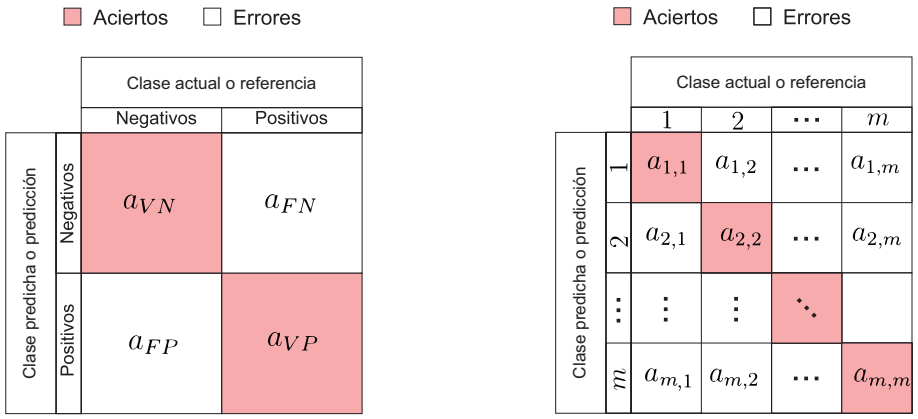
Es importante tener en cuenta que los conjuntos de entrenamiento y validación intervienen en el ajuste y selección del modelo final, por lo que no resultan indicados para evaluar su generalización. No se debe olvidar que el fin último en el entrenamiento de una red neuronal es que ésta generalice, es decir que aprenda a dar respuesta a patrones nunca antes vistos en el entrenamiento, por lo que posterior a un proceso de validación cruzada se debe considerar un tercer conjunto de datos (conjunto de prueba), que permita medir el rendimiento del modelo ante patrones desconocidos, en otras palabras, que permita evaluar su generalización.

Antes de presentar formalmente las técnicas de validación cruzada más habituales en la práctica, vale la pena revisar algunas de las medidas de rendimiento más importantes en el contexto de la clasificación de patrones.

2.5 Medidas de rendimiento en clasificación

Las medidas de rendimiento de un modelo de clasificación se basan en los conceptos de error y acierto. Se habla de un acierto en clasificación cuando dado un patrón de prueba $(\mathbf{x}, \zeta^{(a)})$ la clase predicha $\zeta^{(p)}$ coincide con la clase actual $\zeta^{(a)}$, en caso contrario se habla de un error de clasificación. La información para calcular las medidas de rendimiento se obtiene utilizando un conjunto de prueba, en la práctica resulta útil organizar dicha información en un arreglo denominado matriz de confusión, el cual muestra el número de errores y aciertos en función de la clase actual y la clase predicha. En la figura 2.6 se presentan dos matrices de confusión una relacionada con un problema de clasificación binario y otra relacionada con un problema de clasificación multiclase.

Ahora bien, en aquellos casos en los que cualquier error tiene la misma importancia, el número total de errores observados puede ser un indicador de que tan bien trabaja un clasificador. Este enfoque se basa en la exactitud (ACC) como indicador de rendimiento y su definición esta dada por la razón entre el número de aciertos y el número total de muestras



a) b)

Figura 2.6. Matrices de confusión: a) Clasificación binaria b) Clasificación multiclase

$$ACC = \frac{Numero\ de\ aciertos}{Numero\ total\ de\ muestras} \tag{2.25}$$

o equivalentemente, en aplicaciones de clasificación binaria

$$ACC = \frac{a_{VN} + a_{VP}}{a_{VN} + a_{VP} + a_{FP} + a_{FN}} \tag{2.26}$$

donde:

a_{VN}: número de aciertos de clase negativa (Verdaderos Negativos).
a_{VP}: número de aciertos de clase positiva (Verdaderos Positivos).
a_{FP}: número de errores clasificados como positivos (Falsos Positivos).
a_{FN}: número de errores clasificados como negativos (Falsos Negativos).

La exactitud tiene desventajas en cuanto a que esta medida depende de la distribución de las clases en el conjunto de prueba y además por que no considera las diferencias entre los diferentes tipos de errores y aciertos.

Por otra parte, la mayoría de las medidas de rendimiento están definidas para evaluar modelos de clasificación binarios, esto no es un inconveniente cuando se trata de problemas multiclase puesto que este tipo de aplicaciones puede presentarse como una serie de problemas binarios, de modo que podría elegirse una (pueden ser varias) clase objetivo como clase positiva y las clases restantes serian la clase negativa. Por lo anterior, las medidas de rendimiento presentadas a continuación se definen únicamente para problemas de dos clases. Algunas de las medidas más sobresalientes en clasificación son:

Sensibilidad (SEN): llamada también razón de éxitos

$$SEN = \frac{a_{VP}}{a_{VP} + a_{FN}}. \tag{2.27}$$

Especificidad (ESP): o razón de verdaderos negativos

$$ESP = \frac{a_{VN}}{a_{VN} + a_{FP}}. \tag{2.28}$$

Precisión (PRE):

$$PRE = \frac{a_{VP}}{a_{VP} + a_{FP}}. \tag{2.29}$$

Área bajo la curva ROC (AUC): La curva ROC es un gráfico que relaciona los verdaderos positivos (sensibilidad) con los falsos positivos (1-especificidad) para diferentes umbrales de discriminación. En la Figura 2.7 se muestra una curva ROC típica, donde la línea diagonal punteada (línea de no discriminación) representa la curva de un clasificador cuyo rendimiento no se diferencia al del azar. Nótese además que el

modelo ideal (perfecta clasificación) está representado por un punto en la esquina superior Izquierda.

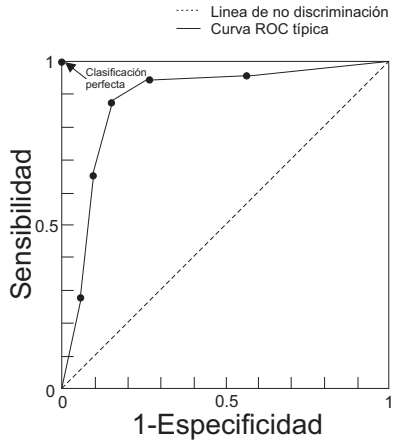


Figura 2.7. Curvas en el espacio ROC

Para obtener la curva ROC de un clasificador ELM es necesario incorporar una función de salida continua que permita calcular (para cada muestra) la probabilidad de pertenencia a la clase positiva. De modo que la pertenencia a una clase u otra se determina estableciendo un umbral el cual puede variarse para obtener los diferentes puntos de la curva ROC. Ahora bien, el área bajo la curva obtenida es un indicador poderoso del rendimiento de un clasificador, de tal forma que entre más cercano a uno sea el valor de AUC mejor será el rendimiento.

2.6 Técnicas de validación cruzada

Dependiendo del valor de los parámetros (L, C) se pueden obtener múltiples modelos entre los que se debe seleccionar el que ofrezca el mejor rendimiento, esto generalmente se realiza utilizando técnicas de validación cruzada. La idea fundamental detrás de estas técnicas gira en torno al método de retención ("*hold out*") que consiste en dividir el conjunto de entrenamiento en dos subconjuntos; el primer subconjunto es para ajustar los pesos y bias del modelo, por lo que vuelve a tomar el nombre de conjunto de entrenamiento y el segundo subconjunto es para llevar a cabo un proceso de validación que consiste en seleccionar el modelo de mejor rendimiento entre múltiples modelos obtenidos con distintas configuraciones (C, L), denominado conjunto de validación. En

el anexo B se presenta el proceso de entrenamiento de un clasificador ELM utilizando el método de retención con R valores de C y un único valor de L.

El método de retención es la técnica más simple de validación cruzada, a partir de la cual se derivan otras variantes como la validación cruzada de k iteraciones ("k-*fold cross validation*") y la técnica de dejar uno fuera ("*leave one out*"). El uso de estas técnicas en el entrenamiento de redes ELM requiere un manejo adecuado del costo computacional, esto implica la reutilización de las matrices que intervienen en el cálculo de las posibles soluciones, un tratado acerca de esta temática puede ser encontrado en el trabajo de Kokkinos y Margaritis (2018).

2.6.1 Validación cruzada con k iteraciones

La validación cruzada con k iteraciones es un procedimiento que consiste en dividir el conjunto de entrenamiento en k subconjuntos o particiones. Posteriormente se utilizan $(k-1)$ particiones para entrenar y la partición restante se utiliza para validar el rendimiento de los modelos obtenidos al probar las distintas configuraciones de parámetros. De modo que por cada configuración (L, C) se obtienen k modelos diferentes, cada uno entrenado con una combinación diferente de $(k-1)$ particiones. Luego por cada configuración se promedia el rendimiento de los k modelos obtenidos y se selecciona la que en promedio tenga el mejor rendimiento. El modelo final se obtiene entrenando con todo el conjunto de entrenamiento y utilizando la configuración de parámetros seleccionada (Hastie *et al.*, 2009). En el anexo C se presenta el proceso de entrenamiento de un clasificador ELM con validación cruzada de k iteraciones utilizando diferentes valores de C y un valor de L.

2.6.2 Técnica de dejar uno fuera

Consiste en entrenar utilizando todas las muestras de entrenamiento excepto una $(N-1)$, la muestra faltante es usada para validar. De modo que por cada configuración de parámetros se obtienen N modelos validados con una muestra diferente (suponiendo que las muestras de entrenamiento son distintas entre sí). El modelo final se crea utilizando la configuración con el mejor rendimiento promedio (Hastie *et al.*, 2009). Se trata de una técnica de validación cruzada computacionalmente costosa (la más extrema de todas) e inviable en aplicaciones con grandes conjuntos de entrenamiento, por lo que su utilización no se considera en este trabajo de investigación.

2.7 Consideraciones en relación a "*Big Data*"

Estamos en la época del "*Big Data*". Diariamente inmensas cantidades de datos son generados y compartidos por las empresas, las administraciones públicas y numerosos sectores industriales, investigativos y sin ánimo de lucro. Estos datos incluyen desde contenido textual (estructurado, semiestructurado y no estructurado), hasta contenido multimedia (videos, imágenes, audio) en una multiplicidad de plataformas, entre éstas: comunicaciones máquina a máquina, redes sociales, redes de sensores, sistemas físicos e Internet de las cosas (Sivarajah *et al.*, 2017). Este fenómeno tiene unas proporciones gigantescas, a nivel ilustrativo se puede señalar que desde el inicio de la historia hasta 2003, la humanidad produjo aproximadamente 5 exabytes (es decir, 5 mil millones de gigabytes) de información y en 2011 ya se creaba la misma cantidad de información cada dos días (Gil, 2016). Esta tendencia, junto a todos los aspectos relacionados con la captura, almacenamiento, representación, análisis y manejo de los datos se enmarcan dentro de lo que se conoce actualmente como "*Big Data*" o BD (Wang *et al.*, 2016).

El impacto de BD en todos los aspectos de la sociedad moderna es enorme (influye por ejemplo en decisiones estatales, comerciales y medicas), razón por la cual recibe mucha atención por parte de investigadores pertenecientes a diversas disciplinas entre las que se incluyen las ciencias naturales, la ingeniería e incluso el arte y las humanidades. Lo anterior evidencia el enorme valor de uso que trae consigo BD, valor que se extrae a través de los procesos de descubrimiento, integración y explotación de los datos y que juega un papel preponderante a la hora de tomar decisiones en diversos entornos. Para muestra, el instituto Mckinsey reportó que más del 50% entre 560 importantes empresas consideran que BD contribuye enormemente al incremento de la eficiencia operacional, a la selección de las directrices estratégicas y a brindar un mejor servicio a los clientes (Wang *et al.*, 2016).

Es importante destacar que BD como una disciplina de investigación está todavía en evolución y una definición comprensible del fenómeno aún no se ha establecido plenamente, empezando por que el concepto "*big*" (en referencia a masivas cantidades) es problemático, dado que un conjunto de entrenamiento que puede parecer actualmente masivo, con seguridad, parecerá de dimensiones pequeñas en el futuro cercano, además en la práctica, no necesariamente un conjunto de datos extenso es complejo, así como tampoco un pequeño conjunto de datos obligatoriamente es simple. De modo que las características de los datos a tratar resultan un factor determinante

a la hora de definir lo que son o no son "*masivas cantidades*". Pese a todo esto, hoy en día BD es una realidad con resultados tangibles e importantes proyecciones hacia el futuro (Sivarajah *et al.*, 2017).

2.7.1 Concepto y características de BD

Más que definir BD en términos de talla (gigabytes, terabytes, pentabytes) el concepto de BD usualmente se aborda desde diferentes perspectivas (orientadas a producto, proceso, cognición y movimiento social), por su carácter técnico, aquí se destacan la perspectiva orientada a producto y la perspectiva orientada a procesos. La primera enfatiza en las características de los datos y la segunda destaca la novedad de los procesos requeridos e involucrados en el almacenamiento, gestión, agrupación, búsqueda y análisis de los datos.

En la literatura se pueden encontrar varias propuestas que buscan definir las características esenciales de BD. El enfoque más común se distingue como 3Vs en alusión al *Volumen*, la *Velocidad* y la *Variedad* de los datos. Sin embargo, algunos gigantes de la industria tecnológica han propuesto otras: IBM propone como cuarta característica a la *Veracidad*, SAS por su parte agrego dos más; la*Variabilidad* y la *Complejidad* y Oracle introdujo una última referida como *Valor* (Sivarajah *et al.*, 2017). Así entonces, desde la perspectiva orientada a producto, se define BD como un conjunto de datos que posee las siguientes características:

- **Volumen:** BD implica recoger, almacenar y tratar grandes cantidades de datos y metadatos en un volumen tan grande que no pueden ser analizados mediante herramientas y procesos tradicionales. Tales cantidades implican un reto puesto que se hace necesario desarrollar nuevas tecnologías para almacenamiento, acceso, y custodia de los datos.

- **Velocidad:** la velocidad a la que se crean y procesan los datos está en continuo aumento, y con frecuencia para las organizaciones es importante poder analizarlos de forma muy rápida, incluso en tiempo real, algo que en ocasiones es imposible con los sistemas tradicionales. BD requiere transferir datos de forma económica y eficiente, y así se pueden analizar tanto los datos dinámicos que se van creando, como los datos estáticos o históricos que ya han sido almacenados de forma previa. Por ejemplo, el análisis de datos en tiempo real puede ayudar a

predecir la intensidad y la trayectoria de huracanes, para predecir dónde puede presentarse un desastre con horas o incluso días de antelación.

- **Variedad:** el enorme volumen de datos no es consistente, estos no siguen un modelo o formato especifico, si no que se capturan en formas diferentes y desde diversas fuentes. Así entones, la heterogeneidad en los datos, entendida como una propiedad natural de BD, exige desarrollar novedosos sistemas que permitan tratar conjuntos de datos a pesar de que no se encuentren almacenados en ficheros con la misma estructura.

- **Veracidad:** sean las fuentes estructuradas o no, siempre existen factores que pueden hacer que los datos recolectados no sean fiables, por ejemplo, las opiniones de los clientes en las redes sociales o en la web, pueden en su mayoría no ser claras y precisas dada la interacción humana. Sin embargo, como fuente de información, la web resulta ser una herramienta de innegable importancia. Por tanto, tratar con datos imprecisos es otra faceta de BD y el cómo abordarla, necesariamente implica realizar esfuerzos investigativos a fin de mejorar las actuales herramientas para manejo y minería de datos.

- **Valor:** la finalidad última de los procesos de BD es crear valor, ya sea entendido como oportunidades económicas o como innovación. Sin él, los esfuerzos dejan de tener sentido.

Ahora bien, evidentemente cada una de las características de BD impone desafíos en materia de investigación, infraestructura y procesos. Por tanto, también puede definirse BD en función de una necesidad tecnológica que surge cuando la aplicación normal de la tecnología actual, no permite a los usuarios obtener respuestas oportunas, rentables y de calidad a preguntas basadas en datos. En este sentido y desde la perspectiva orientada a procesos, se hace referencia a BD, como a un conjunto de datos cuyo tamaño extremadamente grande, obliga a mirar más allá de los métodos probados y verdaderos que prevalecen en la actualidad, a fin de lograr su almacenamiento, gestión, agrupación, búsqueda y análisis.

2.7.2 ELM en aplicaciones de Big Data

Desarrollar métodos de entrenamiento para redes SLFN tomando como base los principios que rigen el funcionamiento de ELM, resulta ser una opción muy atractiva en lo

que respecta al procesamiento de BD y, en general para resolver problemas que involucran grandes conjuntos de entrenamiento. No obstante, ELM enfrenta importantes desafíos a la hora de realizar tales tareas.

El desafío más importante que enfrenta ELM de cara a resolver eficazmente problemas de BD, está relacionado con la necesidad de invertir grandes matrices y, se debe al elevado número de neuronas requeridas en la capa oculta, lo cual además de causar problemas de almacenamiento y limitaciones de naturaleza computacional, ocasiona también problemas de estabilidad. Para superar estas dificultades se han planteado una serie de mejoras a nivel funcional y de implementación. A continuación, se describen algunas de las más destacadas.

- **Mejoras en relación a la estabilidad:** calcular los pesos de salida de una red ELM equivale a resolver un problema de mínimos cuadrados o regresión contraída ("*Rigde Regretion*") donde se invierten matrices de $N \times N$ o $L \times L$. La estabilidad en este proceso puede manejarse seleccionando apropiadamente el valor de C para asegurar que las matrices a invertir $\left(\frac{\mathbf{I}}{C} + \mathbf{H}\mathbf{H}^T\right)$ o $\left(\frac{\mathbf{I}}{C} + \mathbf{H}^T\mathbf{H}\right)$ sean definidas positivas. Sin embargo, seleccionar un valor C muy grande puede generar mayores errores de predicción (Huang *et al.*, 2015a). Una forma de mejorar la estabilidad sin sacrificar considerablemente la precisión consiste en hacer un mapeo más adecuado en la capa oculta. Wang *et al.* (2011) prueban que para ciertas funciones de activación (funciones de base radial) existe un conjunto de pesos de entrada, tales que **H** es de rango columna completo o rango fila completo, con lo que proponen un algoritmo que remplaza el mapeo aleatorio de ELM y mejora la estabilidad en el cálculo de los pesos de salida. Posteriormente este algoritmo se extendió a redes con funciones de activación sigmoidal (Chen *et al.*, 2013).

- **Entrenamiento dinámico:** en esta categoría se incluyen todos aquellos algoritmos que buscan mejorar la compactividad de la red mediante la creación, eliminación o reemplazo de neuronas ocultas durante el entrenamiento. Estos algoritmos intentan evitar neuronas irrelevantes en la capa oculta que puedan causar problemas de sobre ajuste debido a ruido y datos correlacionados. Se destacan variantes importantes de ELM como por ejemplo OP-ELM (Miche *et al.*, 2010), *Incremental ELM* (I-ELM) (Huang *et al.*, 2008), *Sparse Bayesian-ELM* (Luo *et al.*, 2014) y *QR Factorization Based Incremental-ELM* (QRI-ELM) (Ye y Qin, 2015).

- **Reducción usando algoritmos de gradiente:** Yu y Deng (2012) proponen disminuir el tamaño de la capa oculta haciendo uso del algoritmo de gradiente descendiente. Los autores comprueban experimentalmente que la red requiere un tamaño dieciséis veces menor y como tal un tiempo de prueba dieciséis veces más corto en relación con el algoritmo ELM estándar. Aqlan *et al.* (2008) propone otra iniciativa similar que combina el algoritmo de Levenberg Marquart con ELM.

- **Métodos ELM basados en ensamblaje:** dentro de esta categoría se incluyen propuestas que consisten en dividir el conjunto de entrenamiento en múltiples subconjuntos y entrenar con cada uno de estos una red ELM para posteriormente (usando todos los modelos obtenidos) estimar la salida general, esto buscando superar problemas de sobre entrenamiento y a la vez reducir los tiempos de prueba. Propuestas de este estilo pueden ser encontradas en los trabajos de Liu y Wang (2010) y Cao *et al.* (2012).

- **Implementaciones paralelas de ELM:** para mejorar los tiempos de entrenamiento y utilizar ELM en aplicaciones con grandes conjuntos de entrenamiento, comúnmente se hace uso de técnicas de procesamiento paralelo. Varios esfuerzos investigativos se han realizado en este sentido, entre los cuales se destacan aquellas propuestas que plantean usar técnicas basadas en *MapReduce* e implementaciones GPU (*Graphics Processing Unit*).

  MapReduce es un popular modelo de programación paralela diseñado para manejar grandes conjuntos de datos (Kamal *et al.*, 2016). Huang *et al.* (2016) por ejemplo, presenta el diseño de un algoritmo paralelo de tipo secuencial basado en este modelo para análisis y procesamiento de datos a gran escala llamado *Parallel Ensemble of Online Sequential- ELM*.

  Por su parte, las implementaciones GPU hacen referencia al uso de unidades de procesamiento gráfico (GPU) en lugar de usar el procesador CPU (*Central Processing Unit*) aprovechando el hecho de que el rendimiento de las tarjetas de vídeo se ha incrementado mucho más rápido que el de los convencionales computadores de escritorio (Heeswijk *et al.*, 2011). Krawczyk (2016) propone una metodología para implementar un algoritmo de este tipo usando la librería cuBLAS (*NVIDIA CUDA Basic Linear Algebra Subroutines*), esto permite un sig-

nificante incremento de velocidad en la realización de las operaciones que demandan un mayor costo computacional, como por ejemplo el cálculo de la pseudoinversa. Otra iniciativa similar puede ser encontrada en el trabajo de Heeswijk *et al.* (2011).

- **Técnicas basadas en *Cloud Computing*:** dado el crecimiento en el volumen y la complejidad estructural de los datos involucrados en las aplicaciones modernas y, teniendo en cuenta que no todos los usuarios disponen de procesamiento especializado, se han propuesto metodologías que buscan explotar los recursos que brinda internet para potencializar el procesamiento con ELM. Una propuesta en este sentido es la de Lin *et al.* (2013), en la que se plantea un mecanismo basado en ELM para manejar procesamiento de datos a gran escala, se trata de una variante llamada *Partitioned-ELM* que mejora significativamente el desempeño de ELM mediante la exportación (hacia plataformas de computo especializadas) de las operaciones matemáticas que demandan el mayor costo computacional. Los resultados analíticos y experimentales muestran que *Partitioned-ELM* disminuye considerablemente el tiempo de entrenamiento.

Capítulo 3

Máquina de entrenamiento comprimido basada en ELM

Este capítulo articula los conceptos fundamentales bajo los cuales se desarrolla la Máquina de entrenamiento comprimido basada en ELM o MEC-ELM. Este nuevo algoritmo de entrenamiento para redes SLFN actúa comprimiendo la información proveniente de la capa oculta por medio de un subconjunto de nodos de la misma capa, permitiendo disminuir considerablemente la complejidad computacional en comparación con la solución de ELM para grandes conjuntos de entrenamiento. Finalmente, se presenta los resultados de una serie de pruebas preliminares relacionadas con clasificación de clases bidimensionales de distribución gaussiana.

3.1 Diseño matemático de MEC-ELM

Considérese inicialmente la solución óptima de ELM para el caso en que $N > L$, esto es

$$\mathbf{B} = \left(\frac{\mathbf{I}_{L \times L}}{C} + \mathbf{H}^T \mathbf{H} \right)^{-1} \mathbf{H}^T \mathbf{T} \tag{3.1}$$

y un conjunto de patrones $\eta_Q = \{\{(\mathbf{x}_j, \mathbf{h}_j)\}_{j=1}^{N} | \mathbf{x}_j \in \mathbb{R}^n, \mathbf{h}_j \in \mathbb{R}^L\}$ tal que

$$\mathbf{H} = \begin{bmatrix} \mathbf{h}_1 & \mathbf{h}_2 & \dots & \mathbf{h}_j & \dots & \mathbf{h}_N \end{bmatrix}^T. \tag{3.2}$$

Entonces, con base en el teorema 2.3.1, las muestras del conjunto de entrenamiento η_Q pueden ser aproximadas tanto como se desee usando un segundo modelo ELM de q nodos ocultos. Por simplicidad, en adelante los nodos ocultos de este segundo modelo serán referidos como nodos de compresión. Por lo tanto, siempre existirá un modelo ELM auxiliar tal que:

$$||\mathbf{QD} - \mathbf{H}|| \leq \epsilon_0 \tag{3.3}$$

donde $\mathbf{Q} \in \mathbb{R}^{N \times q}$ es la matriz de salida de los nodos de compresión, $\mathbf{D} \in \mathbb{R}^{q \times L}$ la matriz de pesos de salida de los nodos de compresión y ϵ_0 un valor real positivo tan pequeño como se quiera. La configuración propuesta se presenta en la figura 3.1.

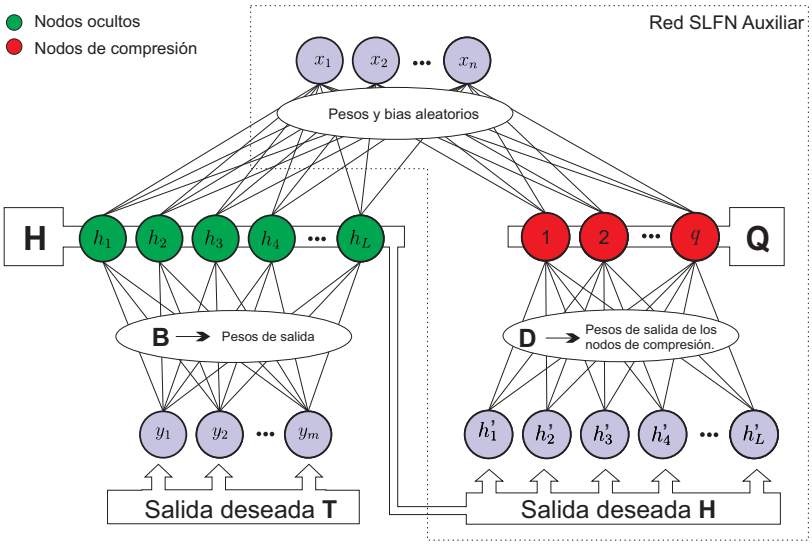


Figura 3.1. Configuración propuesta para el diseño de MEC-ELM

Ahora bien, con base en la ecuación 3.3 se tiene que, si ϵ_0 tiende a cero, entonces $(\mathbf{QD})^T\mathbf{H}$ tiende a $\mathbf{H}^T\mathbf{H}$, de modo que la ecuación 3.1 se puede reescribir como

$$\mathbf{B} = \left(\frac{\mathbf{I}_{L\times L}}{C} + \left|(\mathbf{QD})^T\mathbf{H}\right|_{\epsilon_0 \to 0}\right)^{-1} \mathbf{H}^T\mathbf{T}. \tag{3.4}$$

Adicionalmente, la ecuación 3.4 sugiere que se pueden seleccionar un conjunto de nodos de compresión que permitan comprimir la información proveniente de la capa oculta de un modelo ELM en función de la matriz de pesos **D**, para calcular una aproximación de la solución **B** planteada en 3.1, como sigue

$$\mathbf{B} \approx \left(\frac{\mathbf{I}_{L\times L}}{C} + \left|(\mathbf{QD})^T\mathbf{H}\right|_{\epsilon_0 \to \epsilon}\right)^{-1} \mathbf{H}^T\mathbf{T} \tag{3.5}$$

donde ϵ es un número arbitrariamente pequeño. Más aún, si se tiene en cuenta que el teorema 2.3.1 garantiza que siempre existirá un conjunto de $q \leq N$ nodos de compresión seleccionados aleatoriamente que satisfagan la ecuación 3.3, se puede plantear una solución $\mathbf{B}_\epsilon$ dado cualquier valor ϵ, tal que

$$\mathbf{B}_\epsilon = \left(\frac{\mathbf{I}_{L\times L}}{C} + (\mathbf{QD})^T\mathbf{H}\right)^{-1}\mathbf{H}^T\mathbf{T} \tag{3.6}$$

o equivalentemente

$$\mathbf{B}_\epsilon = \mathbf{V}^{-1}\mathbf{H}^T\mathbf{T}, \tag{3.7}$$

donde

$$\mathbf{V} = \frac{\mathbf{I}_{L\times L}}{C} + \mathbf{D}^T\mathbf{I}_{q\times q}\mathbf{Q}^T\mathbf{H}. \tag{3.8}$$

Posteriormente, aplicando la fórmula de inversión matricial de Woodbundry[1] a la ecuación 3.8, de modo que $U = \mathbf{D}^T$ y $W = \mathbf{Q}^T\mathbf{H}$ y, teniendo en cuenta que en dicha fórmula A y C son matrices identidad de diferentes tamaños, se tiene que:

$$\begin{aligned}\mathbf{V}^{-1} &= C\mathbf{I}_{L\times L} - C^2\mathbf{D}^T\left(\mathbf{I}_{q\times q} + C\mathbf{Q}^T\mathbf{H}\mathbf{D}^T\right)^{-1}\mathbf{Q}^T\mathbf{H} \\ &= C\mathbf{I}_{L\times L} - C\mathbf{D}^T\left(\frac{\mathbf{I}_{q\times q}}{C} + \mathbf{Q}^T\mathbf{H}\mathbf{D}^T\right)^{-1}\mathbf{Q}^T\mathbf{H}.\end{aligned} \tag{3.9}$$

Y finalmente, al remplazar la ecuación 3.9 en la ecuación 3.7 se obtiene:

$$\mathbf{B}_\epsilon = C\mathbf{H}^T\mathbf{T} - C\mathbf{D}^T\left(\frac{\mathbf{I}_{q\times q}}{C} + \mathbf{Q}^T\mathbf{H}\mathbf{D}^T\right)^{-1}\mathbf{Q}^T\mathbf{H}\mathbf{H}^T\mathbf{T}. \tag{3.10}$$

Ahora bien, para calcular la matriz de pesos de salida **D** de los nodos de compresión, se plantea usar la solución original de ELM sobre el conjunto de entrenamiento η_Q de modo que se minimice $||\mathbf{QD} - \mathbf{H}||$. Por tanto, si $\mathbf{Q}^\dagger$ es la inversa generalizada de *More Penrose* de la matriz **Q**, entonces

$$\mathbf{D} = \mathbf{Q}^\dagger\mathbf{H}. \tag{3.11}$$

Por otro lado, en forma similar a Tamura y Tateishi (1997) y Huang (2003), se puede probar que para N muestras distintas, el rango de **Q** es q y por tanto que $\mathbf{Q}^T\mathbf{Q}$ es no singular. Esto permite calcular **D** como

$$\mathbf{D} = \left(\mathbf{Q}^T\mathbf{Q}\right)^{-1}\mathbf{Q}^T\mathbf{H}. \tag{3.12}$$

La solución $\mathbf{B}_\epsilon$ con q nodos de compresión es una aproximación de la solución óptima **B** de un modelo ELM con L nodos ocultos. En relación al número de nodos

[1] $(A + UCW)^{-1} = A^{-1} - A^{-1}U(C^{-1} + WA^{-1}U)^{-1}WA^{-1}$

de compresión, el teorema 2.3.1 garantiza que la matriz **H** puede ser aproximada sin error usando la solución original de ELM con $L = N$ nodos ocultos. Por lo tanto, el teorema 2.3.1 también asegura la existencia de una solución $\mathbf{B}_\epsilon$ con $q = N$ nodos de compresión seleccionados aleatoriamente, tal que

$$\mathbf{B}_\epsilon = \mathbf{B}. \tag{3.13}$$

No obstante, para que se satisfaga la ecuación 3.13, solo son necesarios $q = L$ nodos de compresión. Esto se puede comprobar fácilmente seleccionando como nodos de compresión los mismos nodos de la capa oculta. De esta forma $q = L$, $\mathbf{Q} = \mathbf{H}$, $\mathbf{D} = \mathbf{I}_{LxL}$ y la ecuación 3.10 se reduce a la ecuación 3.1.

Lo anterior sugiere que los nodos de compresión deben seleccionarse entre los nodos ocultos. Siguiendo esta idea la matriz de salidas de la capa oculta **H** puede ser expresada como una matriz partida, esto es

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}_{oc} & \mathbf{H}_o \end{bmatrix} \tag{3.14}$$

donde, $\mathbf{H}_{oc} \in \mathbb{R}^{N \times q}$ es la matriz de salida de los nodos ocultos que a su vez son nodos de compresión y $\mathbf{H}_o \in \mathbb{R}^{N \times (L-q)}$ es la matriz de salida de los nodos ocultos que no son nodos de compresión. Por tanto

$$\mathbf{Q} = \mathbf{H}_{oc} \tag{3.15}$$

y la ecuación 3.12 puede reescribirse como

$$\mathbf{D} = \left(\mathbf{H}_{oc}^T \mathbf{H}_{oc}\right)^{-1} \mathbf{H}_{oc}^T \begin{bmatrix} \mathbf{H}_{oc} & \mathbf{H}_o \end{bmatrix} \tag{3.16}$$

o equivalentemente

$$\mathbf{D} = \begin{bmatrix} \mathbf{I}_{q \times q} & \mathbf{V}_{oc}^{-1} \mathbf{W}_{oc} \end{bmatrix}, \tag{3.17}$$

donde

$$\mathbf{V}_{oc} = \mathbf{H}_{oc}^T \mathbf{H}_{oc} \tag{3.18}$$

y

$$\mathbf{W}_{oc} = \mathbf{H}_{oc}^T \mathbf{H}_o. \tag{3.19}$$

Ahora, tomando $\mathbf{Q}^T\mathbf{H}$ de la ecuación 3.12, pero de la forma presentada en 3.16, para posteriormente hacer el producto entre $\mathbf{H}_{oc}^T$ y la matriz partida $\mathbf{H}$ (de la ecuación 3.14), se puede definir una nueva matriz $\mathbf{Z}$ en función de las matrices $\mathbf{V}_{oc}$ y $\mathbf{W}_{oc}$ de las ecuaciones 3.18 y 3.19 respectivamente, tal que:

$$\mathbf{Z} = \mathbf{Q}^T\mathbf{H} = \begin{bmatrix} \mathbf{V}_{oc} & \mathbf{W}_{oc} \end{bmatrix}. \tag{3.20}$$

En estos nuevos términos, la solución $\mathbf{B}_\epsilon$ planteada en la ecuación 3.10 se puede reescribir como sigue

$$\mathbf{B}_\epsilon = C\mathbf{W} - C\mathbf{D}^T \left(\frac{\mathbf{I}_{q\times q}}{C} + \mathbf{Z}\mathbf{D}^T \right)^{-1} \mathbf{Z}\mathbf{W}, \tag{3.21}$$

donde

$$\mathbf{W} = \mathbf{H}^T\mathbf{T}. \tag{3.22}$$

3.2 MEC-ELM: modelo y entrenamiento

MEC-ELM es un algoritmo de entrenamiento para redes SLFN, en el que los pesos y bias de la capa oculta se seleccionan aleatoriamente y los pesos de salida se calculan en función de un subconjunto de nodos ocultos (llamados nodos de compresión) que permiten comprimir la información proveniente de la capa oculta. Análogamente al concepto de modelo ELM, se introduce el concepto de modelo MEC-ELM entendido como una red SLFN entrenada con la máquina de aprendizaje comprimido basada en ELM. De modo que, para una entrada arbitraria $\mathbf{x} \in \mathbb{R}^n$, la salida de un modelo MEC-ELM con L nodos ocultos esta dada por

$$f_L(\mathbf{x}) = \sum_{i=1}^{L} \boldsymbol{\beta}'_i g(\mathbf{w}_i \cdot \mathbf{x} + b_i) \tag{3.23}$$

donde, $\mathbf{w}_i = \begin{bmatrix} w_{i,1} & w_{i,2} & ... & w_{i,m} \end{bmatrix}^T$ es el vector de pesos entre la capa de entrada y el i-ésimo nodo oculto, $\boldsymbol{\beta}'_i = \begin{bmatrix} \beta'_{i,1} & \beta'_{i,2} & ... & \beta'_{i,m} \end{bmatrix}^T$ es el vector de pesos entre el i-ésimo nodo oculto y la capa de salida, b_i el bias del i-ésimo nodo oculto y, $g(.)$ es la función de activación de los nodos ocultos.

Nótese que la salida de un modelo MEC-ELM es igual a la de un modelo ELM, la diferencia entre ambos tiene lugar cuando están operando en modo de entrenamiento.

De modo que, dado un conjunto de entrenamiento $\eta = \{(\mathbf{x}_j, \mathbf{t}_j), \mathbf{x}_j \in \mathbb{R}^n, \mathbf{t}_j \in \mathbb{R}^m\}$, la matriz de salida de la capa oculta $\mathbf{H} \in \mathbb{R}^{N \times L}$ de un modelo MEC-ELM es exactamente igual a la de un modelo ELM y se define según la ecuación 2.5.

El proceso de entrenamiento con MEC-ELM se especifica en la tabla 3.1, donde $\mathbf{B}_\epsilon = \begin{bmatrix} \boldsymbol{\beta'}_1^T & \boldsymbol{\beta'}_2^T & ... & \boldsymbol{\beta'}_L^T \end{bmatrix}^T_{L \times m}$ es la matriz de pesos de salida del modelo MEC-ELM o solución MEC-ELM. Nótese en el paso 3, que las matrices **D**, **W** y **Z** se representan en función de **H**, **T** y q, esto es

	Entradas:	$\eta = \{(\mathbf{x}_j, \mathbf{t}_j), \mathbf{x}_j \in \mathbb{R}^n, \mathbf{t}_j \in \mathbb{R}^m\}_{j=1}^N \rightarrow$ Conjunto de datos $L \rightarrow$ Número de nodos ocultos $q \rightarrow$ Número de nodos de compresión ($q \in \aleph$, tal que $q \leq L$) $C \rightarrow$ Constante de regularización $g(.) \rightarrow$ Función de activación de los nodos ocultos
	Salidas:	$\{\mathbf{w}_i, b_i\}_{i=1}^L \rightarrow$ Pesos y bias ocultos $\mathbf{B}_\epsilon \rightarrow$ Solución de MEC-ELM (Matriz de pesos de salida)
1		Asignar aleatoriamente los pesos $\mathbf{w}_i$ y los bias b_i para $i = 1, 2, ..., L$.
2		Calcular la matriz de salidas de la capa oculta **H**
3		Calcular las matrices **D**, **W** y **Z** tales que $[\mathbf{D}, \mathbf{W}, \mathbf{Z}] = \Psi(\mathbf{H}, \mathbf{T}, q)$ (procedimiento especificado en la tabla 3.2).
4		Calcular los pesos de salida $\mathbf{B}_\epsilon$ como: $\mathbf{B}_\epsilon = C\mathbf{W} - C\mathbf{D}^T \left(\frac{\mathbf{I}}{C} + \mathbf{Z}\mathbf{D}^T\right)^{-1} \mathbf{Z}\mathbf{W}$

Tabla 3.1. Entrenamiento de un modelo MEC-ELM

$$[\mathbf{D}, \mathbf{W}, \mathbf{Z}] = \Psi(\mathbf{H}, \mathbf{T}, q), \tag{3.24}$$

donde $\mathbf{T} = \begin{bmatrix} \mathbf{t}_1 & \mathbf{t}_2 & ... & \mathbf{t}_N \end{bmatrix}^T$ es la matriz de salidas deseadas y Ψ una función denominada función auxiliar de MEC-ELM. La implementación de Ψ se presenta en la tabla 3.2.

En relación a la función auxiliar de MEC-ELM, cabe recordar que los nodos de compresión se seleccionan como un subconjunto de los nodos ocultos, de modo que debe distinguirse entre dos tipos de nodos ocultos: unos que a su vez también funcionan como nodos de compresión ($\mathbf{H}_{oc}$) y otros que únicamente son nodos ocultos ($\mathbf{H}_o$). Ahora bien, como se observa en la ecuación 3.14, las matrices $\mathbf{H}_{oc}$ y $\mathbf{H}_o$ son submatrices de **H**, y por tanto al calcularse **H**, implícitamente también se calculan dichas matrices, de modo que $\mathbf{H}_{oc}$ y $\mathbf{H}_o$ simplemente deben identificarse (como indica el paso 2 del procedimiento mostrado en la tabla 3.2), para esto únicamente se debe definir el

número de nodos de compresión q y seleccionar las matrices apropiadas. Por tanto, si **H** esta dada por

$$\mathbf{H} = \begin{bmatrix} h_{1,1} & h_{1,2} & \cdots & h_{1,q} & h_{1,(q+1)} & h_{1,(q+1)} & \cdots & h_{1,L} \\ h_{2,1} & h_{2,2} & \cdots & h_{2,q} & h_{2,(q+1)} & h_{2,(q+2)} & \cdots & h_{1,L} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ h_{N,1} & h_{N,2} & \cdots & h_{N,q} & h_{N,(q+1)} & h_{N,(q+2)} & \cdots & h_{N,L} \end{bmatrix}. \tag{3.25}$$

Entonces

$$\mathbf{H}_{oc} = \begin{bmatrix} h_{1,1} & h_{1,2} & \cdots & h_{1,q} \\ h_{2,1} & h_{2,2} & \cdots & h_{2,q} \\ \vdots & \vdots & \ddots & \vdots \\ h_{N,1} & h_{N,2} & \cdots & h_{N,q} \end{bmatrix} \tag{3.26}$$

y

$$\mathbf{H}_{o} = \begin{bmatrix} h_{1,(q+1)} & h_{1,(q+1)} & \cdots & h_{1,L} \\ h_{2,(q+1)} & h_{2,(q+2)} & \cdots & h_{1,L} \\ \vdots & \vdots & \ddots & \vdots \\ h_{N,(q+1)} & h_{N,(q+2)} & \cdots & h_{N,L} \end{bmatrix}. \tag{3.27}$$

	Entradas:	**H** $\rightarrow$ Matriz de salida de la capa oculta **T** $\rightarrow$ Matriz de salidas deseadas $q \rightarrow$ Número de nodos de compresión ($q \in \aleph$, tal que $q \leq L$)
	Salidas:	**D**, **W** y **Z** $\rightarrow$ Matrices auxiliares de MEC-ELM
1		Identificar $\mathbf{H}_{oc} \in \mathbb{R}^{N \times q}$ y $\mathbf{H}_{o} \in \mathbb{R}^{N \times (L-q)}$ tales que $\mathbf{H} = [\mathbf{H}_{oc} \quad \mathbf{H}_{o}]$ y calcular $\mathbf{V}_{oc}$ y $\mathbf{W}_{oc}$ como: $\mathbf{V}_{oc} = \mathbf{H}_{oc}^{T}\mathbf{H}_{oc}$ $\mathbf{W}_{oc} = \mathbf{H}_{oc}^{T}\mathbf{H}_{o}$
2		Calcular $\mathbf{V}_{oc}^{-1}\mathbf{W}_{oc}$
3		Conformar las matrices $\mathbf{D} = \begin{bmatrix} \mathbf{I} & \mathbf{V}_{oc}^{-1}\mathbf{W}_{oc} \end{bmatrix}$ y $\mathbf{Z} = [\mathbf{V}_{oc} \quad \mathbf{W}_{oc}]$
4		Calcular **W** como: $\mathbf{W} = \mathbf{H}^{T}\mathbf{T}$

Tabla 3.2. Función auxiliar de MEC-ELM $[\mathbf{D}, \mathbf{W}, \mathbf{Z}] = \Psi(\mathbf{H}, \mathbf{T}, q)$.

3.2.1 Consideraciones en relación a la complejidad computacional.

El modelo de entrenamiento comprimido basado en ELM es un esquema de aprendizaje propuesto para aplicaciones que involucran grandes conjuntos de datos. En este tipo de escenarios las máquinas de entrenamiento extremo han ganado gran interés puesto que su solución se puede calcular en forma relativamente rápida (Akusok et al., 2015).

Por lo anterior, resulta interesante analizar los requerimientos computacionales y de almacenamiento de MEC- ELM en contraste con los que demanda ELM. Para esto considérese la tabla 3.3, en la que se presentan todas las operaciones que intervienen en el cálculo de la solución para ambos modelos junto a sus respectivos ordenes de complejidad computacional y de almacenamiento.

Operación	**Complejidad Computacional**		**Requerimientos de Memoria ($\tilde{N} \leq N$)**	
	MEC-ELM	**ELM**	**MEC-ELM**	**ELM**
Datos $\{\mathbf{x}_j \mid \mathbf{x}_j \in \mathbb{R}^n$ para $j = 1, 2, ..., N\}$	$O(Nn)$		$O(\tilde{N}n)$	
H ecuación 3	$O(NLn)$		$O(\tilde{N}L)$	
$\mathbf{V}_{oc} = \mathbf{H}_{oc}^T \mathbf{H}_{oc}$	$O(Nq^2)$	–	$O(q^2)$	–
$\mathbf{W}_{oc} = \mathbf{H}_{oc}^T \mathbf{H}_o$	$O(NLq)$	–	$O(Lq)$	
$\mathbf{D} = \begin{bmatrix} \mathbf{I}_{q\times q} & \mathbf{V}_{oc}^{-1}\mathbf{W}_{oc} \end{bmatrix}$	$O(Lq^2)$	–	$O(Lq)$	
$\mathbf{Z} = \begin{bmatrix} \mathbf{V}_{oc} & \mathbf{W}_{oc} \end{bmatrix}$	–	–	–	–
$\mathbf{W} = \mathbf{H}^T\mathbf{T}$	$O(NLm)$		$O(Lm)$	
$\mathbf{A} = \frac{1}{C}\mathbf{I}_{q\times q} + \mathbf{Z}\mathbf{D}^T$	$O(Lq^2)$	–	$O(q^2)$	–
$\mathbf{B}_\epsilon = C\mathbf{W} - C\mathbf{D}^T\mathbf{A}^{-1}\mathbf{Z}\mathbf{W}$	$O(L^2q + L^2m)$	–	$O(L^2)$	–
$\mathbf{S} = \frac{1}{C}\mathbf{I}_{L\times L} + \mathbf{H}^T\mathbf{H}$	–	$O(NL^2)$	–	$O(L^2)$
$\mathbf{B} = \mathbf{S}^{-1}\mathbf{W}$	–	$O(L^3 + L^2m)$	–	$O(L^2 + Lm)$
Total	$O(NL\text{max}(q, n, m))$	$O(NL\text{max}(L, n, m))$	$O(L^2 + Lm)$	$O(L^2 + Lm)$

Tabla 3.3. Requerimientos computacionales y de almacenamiento para MEC-ELM y ELM ($N \geq L$)

En la tabla 3.3 se asume que el número de muestras de entrenamiento es mayor al número de nodos ocultos, es decir $N > L$, teniendo en cuenta que bajo esta condición se propuso el modelo MEC-ELM. En lo que respecta a MEC-ELM, las operaciones con mayor orden computacional son según el caso: calcular la matriz **H** si $n > q$ y $n > m$; calcular la matriz $\mathbf{W}_{oc}$ si $q > n$ y $q > m$ o bien calcular **W** si $m > q$ y $m > n$. Con lo que la complejidad computacional de MEC-ELM está dada por $O(NL\text{max}(q, n, m))$.

En la misma tabla 3.3, se puede observar que la complejidad computacional de ELM cuando $N > L$ está dada por $O(NL\text{max}(L, m, n))$. Esto implica que los reque-

rimientos computacionales de MEC-ELM se pueden reducir en relación a los de ELM dependiendo del número de nodos de compresión siempre y cuando $m < q$ y $n < q$.

Lo anterior resulta conveniente teniendo en cuenta que en MEC-ELM el número de nodos de compresión q es menor o igual al número de nodos ocultos L. En el contexto de las redes SLFN determinar el número de nodos ocultos L es un problema que aún no está lo suficientemente esclarecido. Como consecuencia en la literatura solo suele especificarse que se debe usar un número suficiente de nodos en la capa oculta, por lo que en la práctica generalmente se selecciona de inicio un número grande de nodos ocultos. Cabe anotar que, si bien seleccionar un número elevado de nodos ocultos en algunas aplicaciones puede disminuir el rendimiento en términos de generalización, esto no representa un gran problema cuando se cuenta con grandes conjuntos de entrenamiento, puesto que la gran cantidad de datos evita el sobreajuste.

En lo que respecta a los requerimientos de memoria, nótese en la tabla 3.3 que estos son constantes para MEC-ELM y ELM independientemente del número de muestras de entrenamiento N, ya que las matrices $\mathbf{H}^T\mathbf{H}$, $\mathbf{H}_{oc}^T\mathbf{H}_{oc}$, $\mathbf{H}_{oc}^T\mathbf{H}_o$ y $\mathbf{H}^T\mathbf{T}$ pueden ser calculadas por lotes de $\tilde{N} \leq N$ muestras, de modo que al final el resultado se puede obtener sumando los resultados parciales de cada lote. Esto no incrementa el costo computacional por cuanto, a nivel de hardware y software, la multiplicación y suma de matrices se implementan en una única operación (Golub y Loan, 2013) .

3.3 MEC-ELM en aplicaciones de clasificación.

Al igual que en ELM, el primer paso a la hora de entrenar un modelo MEC-ELM para clasificación, o clasificador MEC-ELM, es asegurar que las etiquetas de clase se codifiquen de acuerdo a la ecuación 2.23 sin importar el preprocesamiento del conjunto de datos, posteriormente se procede a entrenar el modelo tal como se especifica en la tabla 3.1. Suponiendo un conjunto de datos para clasificación $\eta_c = \{\{(\mathbf{x}_j, \zeta_j)\}_{j=1}^N \mid \mathbf{x}_j \in \mathbb{R}^n, \zeta_j \in \{1, 2, ..., m\}\}$, el proceso para entrenar un clasificador MEC-ELM se presenta en la tabla 3.4.

Una vez entrenado, para que un clasificador MEC-ELM pueda operar en modo de ejecución es necesario incluir un módulo decisor para predecir la etiqueta de clase en función de las salidas de red. El esquema de un clasificador MEC-ELM en modo de ejecución se presenta en la figura 3.2 y la función de decisión está dada por

$$\zeta^{(p)} = \underset{v \in \{1, 2, ..., m\}}{\arg\max} \; y'_v \tag{3.28}$$

	Entradas:	$\eta_c = \{\{(\mathbf{x}_j, \mathbf{t}_j)\}_{j=1}^{N} \mid \mathbf{x}_j \in \mathbb{R}^n, \zeta_j \in \{1, 2, ..., m\} \rightarrow$ Conjunto de datos $L \rightarrow$ Número de nodos ocultos $q \rightarrow$ Número de nodos de compresión ($q \in \aleph$, tal que $q \leq L$) $C \rightarrow$ Constante de regularización $g(.) \rightarrow$ Función de activación de los nodos ocultos
	Salidas:	$\{\mathbf{w}_i, b_i\}_{i=1}^{L} \rightarrow$ Pesos y bias ocultos $\mathbf{B}_\epsilon \rightarrow$ Solución de MEC-ELM (Matriz de pesos de salida)
1		Obtener las salidas deseadas $\mathbf{t}_j = [t_{j,1} \quad t_{j,2} \quad ... \quad t_{j,m}]^T$ codificando ζ_j de acuerdo con $$t_{j,v} = \begin{cases} 1 & si \quad v = \zeta_j \\ 0 & si \quad v \neq \zeta_j \end{cases} \qquad v = 1, 2, ..., m \quad j = 1, 2, ..., N,$$
2		Dado $\eta = \{\{(\mathbf{x}_j, \mathbf{t}_j)\}_{j=1}^{N} \mid \mathbf{x}_j \in \mathbb{R}^n, \mathbf{t}_j \in \mathbb{R}^m\}$, entrenar un modelo MEC-ELM (proceso especificado en la tabla 4).

Tabla 3.4. Entrenamiento de un clasificador MEC-ELM

donde $\arg\max$ es una función que determina el índice del máximo elemento y y'_v es el v-ésimo componente de $f_{Lq}(\mathbf{x}) = [y'_1, y'_2, ..., y'_m]$, siendo $f_{Lq}(\mathbf{x})$ el vector de salidas de la red ELM que se obtiene usando la ecuación 3.23.

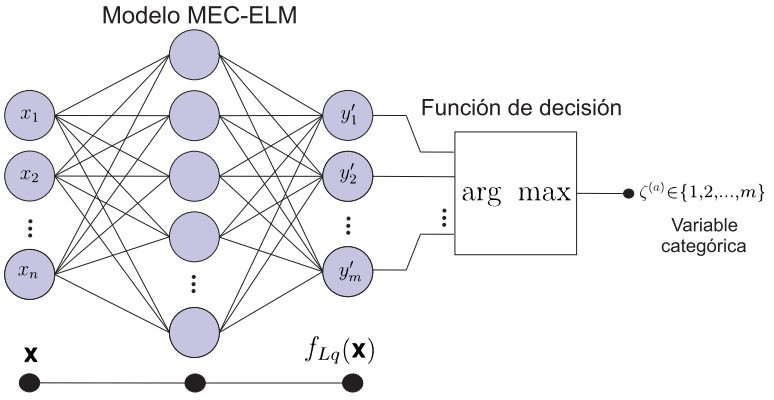


Figura 3.2. Esquema general de un clasificador MEC-ELM en modo de ejecución

3.4 Simulaciones preliminares

En la práctica puede suceder que las clases en el conjunto de entrenamiento estén traslapadas, esta situación puede presentarse por varias razones, entre las que se

incluyen el ruido o simplemente errores en la adquisición de los datos. Como consecuencia se puede generar la presencia de muestras corruptas en el conjunto de entrenamiento y, un algoritmo robusto[2] debe ser capaz de manejar esta situación adecuadamente. Con base en lo anterior, la robustez de MEC-ELM se puso a prueba mediante un problema clásico en el campo de la clasificación de patrones relacionado con la discriminación de clases traslapadas con distribución gaussiana. Este experimento resulta muy interesante puesto que permite analizar el rendimiento de MEC-ELM en presencia de ruido gaussiano.

Las simulaciones se realizaron en el entorno de programación R, el cual cuenta con paquetes especializados para el tratamiento de datos y "*Machine Learning*" entre los que se incluye el paquete CARET ("*Classification And REgression Training*") que dispone de funciones para el diseño y evaluación de modelos predictivos, preprocesamiento, selección de características entre otras más.

La implementación de MEC-ELM utilizada se diseñó siguiendo las indicaciones del paquete CARET ("*create your own model*"), esto permite aprovechar la funcionalidad de este paquete facilitando por ejemplo el uso de distintas variantes de validación cruzada.

3.4.1 Clasificación de clases gaussianas traslapadas

El experimento consiste en discriminar patrones **x** pertenecientes a dos clases bidimensionales traslapadas y con distribución gaussiana (ver figuras 3.3 y 3.4). 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

$$P_x(\mathbf{x}|\omega_1) = \frac{1}{2\pi\sigma_1} e^{\left(\frac{1}{2\sigma_1^2}||\mathbf{x}-\boldsymbol{\mu}_1||^2\right)}, \text{ con media } \boldsymbol{\mu}_1 = [0,\ 0]^T \text{ y varianza } \sigma_1 = 1 \qquad (3.29)$$

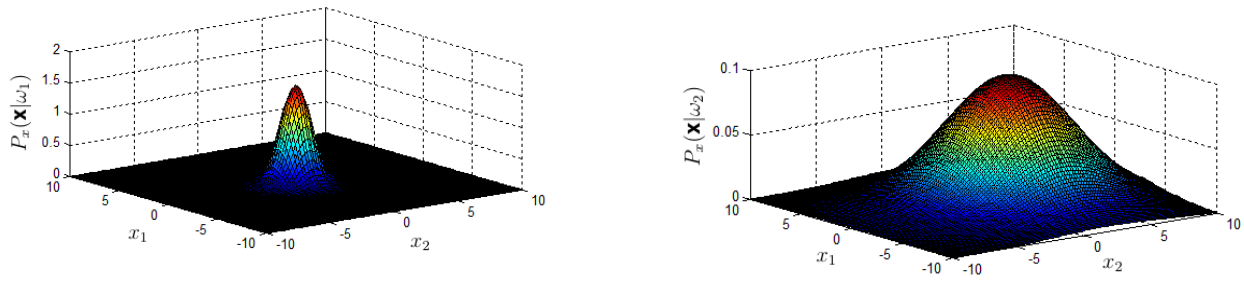
y

$$P_x(\mathbf{x}|\omega_2) = \frac{1}{2\pi\sigma_2} e^{\left(\frac{1}{2\sigma_2^2}||\mathbf{x}-\boldsymbol{\mu}_2||^2\right)}, \text{ con media } \boldsymbol{\mu}_2 = [2,\ 0]^T \text{ y varianza } \sigma_2 = 4. \qquad (3.30)$$

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

[2] 1990. IEEE Standard Glossary of Software Engineering Terminology define robustez (*robustness*) como el grado al que un sistema o componente puede funcionar correctamente en presencia de entradas invalidas o condiciones de ambiente adversas.

$$P(\omega_1) = P(\omega_2) = \frac{1}{2}. \tag{3.31}$$



a) b)

Figura 3.3. Funciones de probabilidad: a) $P_x(\mathbf{x}|\omega_1)$ b) $P_x(\mathbf{x}|\omega_2)$

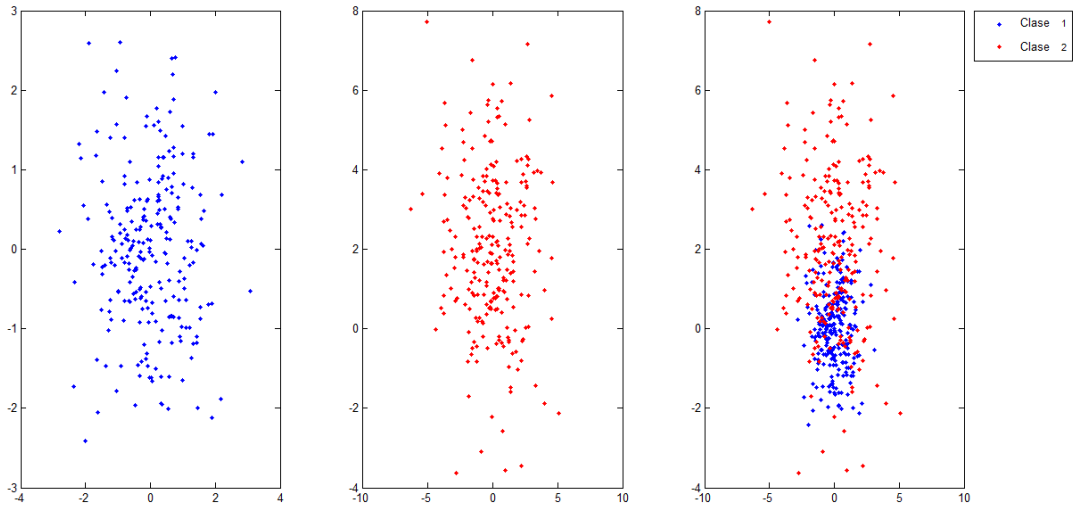


Figura 3.4. Diagrama de puntos para 250 muestras de la clase ω_1 y 250 muestras para la clase ω_2

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

3.4.2 Determinación teórica de la probabilidad de correcta clasificación

Si se asume que los costos para la 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:

$$l_r = \frac{P(\mathbf{x}|\omega_1)}{P(\mathbf{x}|\omega_2)} > \frac{P(\omega_1)}{P(\omega_2)}, \text{ cualquier otro caso implica que } \mathbf{x} \in \omega_2, \tag{3.32}$$

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

$$l_r = \frac{\sigma_2^2}{\sigma_1^2} e^{\left(-\frac{1}{2\sigma_1^2}||\mathbf{x}-\boldsymbol{\mu}_1||^2+\frac{1}{2\sigma_2^2}||\mathbf{x}-\boldsymbol{\mu}_2||^2\right)}, \tag{3.33}$$

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

$$l_r = \frac{\sigma_2^2}{\sigma_1^2} e^{\left(-\frac{1}{2\sigma_1^2}||\mathbf{x}-\boldsymbol{\mu}_1||^2+\frac{1}{2\sigma_2^2}||\mathbf{x}-\boldsymbol{\mu}_2||^2\right)} = 1 \tag{3.34}$$

o equivalentemente

$$\left(-\frac{1}{2\sigma_1^2}||\mathbf{x}-\boldsymbol{\mu}_1||^2+\frac{1}{2\sigma_2^2}||\mathbf{x}-\boldsymbol{\mu}_2||^2\right) = 2\ln\left(\frac{\sigma_1}{\sigma_2}\right). \tag{3.35}$$

La ecuación 3.35 tiene la forma característica de un círculo con centro $\mathbf{x}_c$ y radio r de tal forma que

$$||\mathbf{x}-\mathbf{x}_c||^2 = r^2, \tag{3.36}$$

donde

$$\mathbf{x}_c = \frac{\sigma_2^2\boldsymbol{\mu}_1 - \sigma_1^2\boldsymbol{\mu}_2}{\sigma_2^2-\sigma_1^2} \tag{3.37}$$

y

$$r^2 = \frac{\sigma_2^2\sigma_1^2}{\sigma_2^2-\sigma_1^2}\left[\frac{||\boldsymbol{\mu}_1-\boldsymbol{\mu}_2||^2}{\sigma_2^2-\sigma_1^2} + 4\ln\left(\frac{\sigma_1}{\sigma_2}\right)\right]. \tag{3.38}$$

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

$$P(error) = P(\omega_1)P(error|\omega_1) + P(\omega_1)P(error|\omega_2), \tag{3.39}$$

donde

$$P(error|\omega_1) = \int_{C[\Omega_1]} P(\mathbf{x}|\omega_1)d\mathbf{x} \tag{3.40}$$

y

$$P(error|\omega_2) = \int_{\Omega_1} P(\mathbf{x}|\omega_2)d\mathbf{x}, \tag{3.41}$$

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

Dadas las condiciones del problema que se está tratando, las integrales 3.40 y 3.41 se pueden evaluar numéricamente, encontrándose que:

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

y

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

Finalmente, recordando que, $P(\omega_1) = P(\omega_2) = 1/2$ y remplazando valores en la ecuación 3.39, se obtiene una probabilidad de error de clasificación

$$P(error) \simeq 0.1849$$

y equivalentemente una probabilidad de correcta clasificación

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

3.4.3 Desarrollo de las pruebas

Para realizar las simulaciones, se generaron aleatoriamente dos conjuntos de datos; uno con función de distribución $P_x(\mathbf{x}|\omega_1)$ y otro con función de distribución $P_x(\mathbf{x}|\omega_2)$, a partir de los cuales se conformaron tres conjuntos de entrenamiento con 500, 2000, 8000 patrones. La proporción de clases en todos los conjuntos de entrenamiento fue 50 $\%$ para la clase 1 y 50 $\%$ para la clase 2.

Por cada conjunto de entrenamiento se entrenaron múltiples modelos MEC-ELM variando el número de nodos ocultos L y utilizando en cada caso validación cruzada de 10 iteraciones para seleccionar la constante de regularización C. Los valores que tomó C en las pruebas fueron tales que $C \in \psi$ donde $\psi = \{2^k\}_{k=-4}^{4}$ para $k = -4, -3, ..., 4$.

La exactitud ACC medida sobre un conjunto de prueba con N_p muestras, es un estimativo de la probabilidad de correcta clasificación. En Haykin (1998) se determina experimentalmente (para el problema planteado) que cuando $N_p > 32000$, dicho estimativo tiene una certidumbre superior al 99$\%$. Por lo que se utilizó un conjunto de prueba con 32000 patrones. El proceso de validación cruzada y su respectiva prueba se repitió cincuenta veces por cada valor de L.

Los valores que tomo L en las simulaciones fueron $\{10, 20, 30, 40\}$. En todos los casos la función de activación de los nodos ocultos fue tangente hiperbólica

$$\text{Tanh}(a) = \frac{1 - e^{-a}}{1 + e^{-a}}, \tag{3.42}$$

los pesos ocultos se seleccionaron en el intervalo $[-1, 1]$ y el número de nodos de compresión q se eligió de modo que fuera la mitad del número de nodos ocultos L, esto es $q = L/2$.

3.4.4 Resultados y discusión

El promedio de los resultados obtenidos junto con las respectivas desviaciones estándar se presentan en las tablas 3.5, 3.6 y 3.7.

Nodos Ocultos (L)	**Probabilidad de correcta clasificación (P_ϵ)**	**Desviación Estándar ($Std\ P_\epsilon$)**
10	0.7849	0.0186
20	0.8028	0.0094
30	0.8046	0.0093
40	0.8047	0.0084

Tabla 3.5. Resultados de simulación con $N = 500$ patrones de entrenamiento y usando validacion cruzada de 10 iteraciones con $C \in \psi$ tal que $\psi = \{2^k\}_{k=-4}^{4}$ para $k = -4, -3, ..., 4$.

Nodos Ocultos (L)	**Probabilidad de correcta clasificación (P_ϵ)**	**Desviación Estándar ($Std\ P_\epsilon$)**
10	0.7911	0.0140
20	0.8055	0.0157
30	0.8145	0.0011
40	0.8151	0.0006

Tabla 3.6. Resultados de simulación con $N = 2000$ patrones de entrenamiento y usando validacion cruzada de 10 iteraciones con $C \in \psi$ tal que $\psi = \{2^k\}_{k=-4}^{4}$ para $k = -4, -3, ..., 4$.

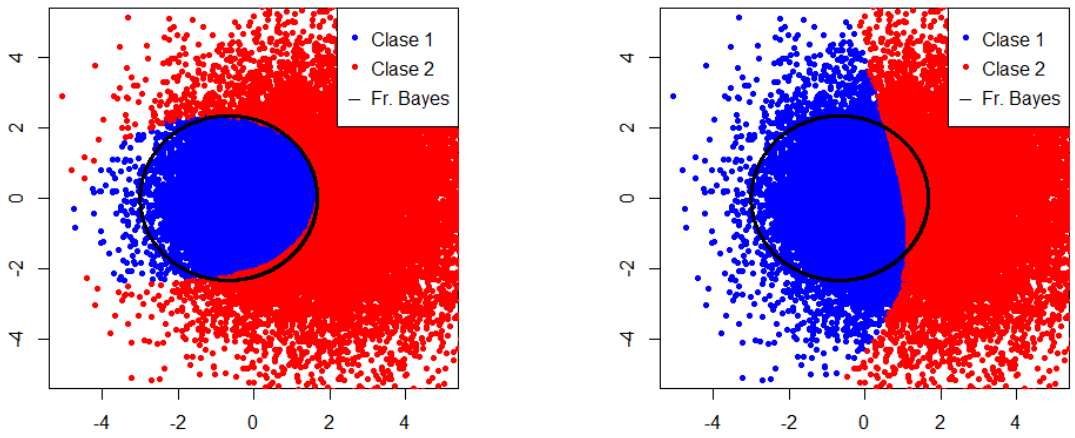
Nodos Ocultos (L)	**Probabilidad de correcta clasificación (P_ϵ)**	**Desviación Estándar ($Std\ P_\epsilon$)**
10	0.7902	0.0130
20	0.8084	0.0070
30	0.8142	0.0035
40	0.8155	0.0005

Tabla 3.7. Resultados de simulación con $N = 8000$ patrones de entrenamiento y usando validacion cruzada de 10 iteraciones con $C \in \psi$ tal que $\psi = \{2^k\}_{k=-4}^{4}$ para $k = -4, -3, ..., 4$.

Los resultados presentados en la tabla 3.6 muestran que en promedio un modelo MEC-ELM de 40 nodos ocultos y 20 nodos de compresión, entrenado con 2000 patrones de entrenamiento, es capaz de discriminar las clases 1 y 2 con un rendimiento

igual al de un clasificador bayesiano ($P_c = 0.8151$) o incluso con un rendimiento levemente superior si se incrementa el número de datos (ver tabla 3.7), lo que da cuenta de la gran capacidad de MEC-ELM para establecer fronteras de decisión adecuadas cuando las muestras de entrenamiento están traslapadas o contaminadas con mucho ruido.

Por otro lado, con base en la desviación estándar de los rendimientos presentados en en la tabla 3.5, resulta claro que hubo modelos MEC-ELM de 20 nodos ocultos y 10 nodos de compresión entrenados con 500 patrones que alcanzaron rendimientos muy cercanos al de un clasificador bayesiano, esto puede verse más claramente en la figura 3.5 donde se presentan la mejor y peor frontera de decisión obtenidas para este tipo de modelos.



a) b)

Figura 3.5. Fronteras de decisión obtenidas con MEC-ELM ($L = 20$, $q = L/2$ y $N = 500$ patrones): a) Mejor frontera ($P_\epsilon = 0.8139$) b) Peor frontera ($P_\epsilon = 0.7774$)

3.5 Resumen de capítulo

Se presentó el diseño matemático de un algoritmo de entrenamiento para redes neuronales SLFN llamado MEC-ELM, en el cual los pesos y bias de la capa oculta son seleccionados aleatoriamente y los pesos de la capa de salida son calculados en función de un subconjunto de nodos ocultos denominados nodos de compresión, dichos nodos

permiten comprimir la información proveniente de la capa oculta usando las propiedades de "*Extreme Learning Machine*", y esto a su vez, permite reducir la complejidad computacional de la solución en relación a ELM.

A partir de un análisis de requerimientos computacionales y de almacenamiento se determinó que el orden de complejidad computacional de MEC-ELM está dado por $O(NL\text{max}(q,n,m))$ y el orden de los requerimientos de almacenamiento por $O(L^2 + Lm)$, donde N es el número de muestras de entrenamiento, L el número de nodos ocultos, q el número de nodos de compresión, n el número de nodos de entrada y m el número de nodos de salida.

La complejidad computacional de MEC-ELM se puede reducir en relación a la de ELM, dependiendo del número de nodos de compresión q utilizados en el entrenamiento, siempre y cuando $n < q < N$ y $m < q$. Por su parte, el orden de complejidad de los requerimientos de almacenamiento de MEC-ELM y ELM son iguales, dichos requerimientos además son constantes sin importar el número de muestras N utilizadas en el entrenamiento.

La robustez de MEC-ELM se puso a prueba mediante un problema de clasificación relacionado con la discriminación de clases bidimensionales traslapadas y con distribución gaussiana, a partir del cual se pudo comprobar que MEC-ELM es capaz de establecer fronteras de decisión complejas y alcanzar rendimientos óptimos en el sentido de Bayes.

Capítulo 4

Estudio experimental de MEC-ELM basado en simulación

Este capítulo presenta una implementación de MEC-ELM basada en validación cruzada de K iteraciones y una serie de resultados experimentales que buscan esclarecer la influencia de los nodos de compresión q en el entrenamiento de un modelo MEC-ELM y mostrar la consistencia de este algoritmo ante distintos problemas de clasificación reales de tipo binario y multiclase.

4.1 Entrenamiento de MEC-ELM utilizando validación cruzada de K iteraciones

Los resultados de simulación presentados en el capítulo 3 se realizaron aprovechando las facilidades que ofrece el paquete CARET para llevar a cabo procesos validación cruzada. Sin embargo, dichas facilidades son herramientas de propósito general que no contemplan las particularidades de cada algoritmo. Como consecuencia, el manejo del costo computacional es limitado cuando se usan en combinación con MEC-ELM.

A continuación, se presenta una implementación para el entrenamiento de modelos MEC-ELM usando validación cruzada de K iteraciones. La implementación propuesta permite manejar el costo computacional mediante la reutilización de matrices. Adicionalmente, en el anexo B.2 se presenta la versión análoga para ELM.

La implementación de MEC-ELM usando validación cruzada de K iteraciones se presenta en la tabla 4.1. Esta implementación utiliza K particiones del conjunto de datos para el entrenamiento y validación de los diferentes modelos. En relación a la r-ésima partición, tal que $r = 1, 2, ..., K$, se considera la siguiente notación:

H$_r$: matriz de salidas de la capa oculta para la r-ésima partición.

$\mathbf{H}_{-r}$: matriz de salidas de la capa oculta para todos los datos excepto la r-ésima partición.

$\mathbf{T}_r$: matriz de salidas deseadas de la r-ésima partición (codificadas según la ecuación 2.23).

$\mathbf{T}_{-r}$: matriz de salidas deseadas de todos los datos excepto la r-ésima partición (codificadas según la ecuación 2.23).

$\mathbf{D}_r$, $\mathbf{W}_r$ y $\mathbf{Z}_r$: matrices auxiliares tales que $[\mathbf{D}_r, \mathbf{W}_r, \mathbf{Z}_r] = \Psi(\mathbf{H}_r, \mathbf{T}_r, q)$.

$\mathbf{D}_{-r}$, $\mathbf{W}_{-r}$ y $\mathbf{Z}_{-r}$: matrices auxiliares tales que $[\mathbf{D}_{-r}, \mathbf{W}_{-r}, \mathbf{Z}_{-r}] = \Psi(\mathbf{H}_{-r}, \mathbf{T}_{-r}, q)$.

Por cada valor de C_a tal que $a = 1, 2, ..., R$, donde C_a es uno de los R valores de la constante de regularización C se obtienen K modelos, a cada uno de los cuales se les mide el rendimiento. Posteriormente, se suma el rendimiento de los K modelos obtenidos por cada valor de C y se selecciona el valor de C con el que se obtuvo el mayor rendimiento acumulado. Por último el valor de C seleccionado se utiliza para entrenar el modelo final en función de las matrices **D**, **W** y **Z** tales que $[\mathbf{D}, \mathbf{W}, \mathbf{Z}] = \Psi(\mathbf{H}_r, \mathbf{T}_r, q)$.

Ahora bien, con base en la notación planteada se puede establecer las siguientes relaciones

$$\mathbf{Z} = \sum_{r=1}^{K} \mathbf{Z}_r \tag{4.1}$$

$$\mathbf{W} = \sum_{r=1}^{K} \mathbf{W}_r \tag{4.2}$$

Nótese que, aunque $\mathbf{D}_r$ y $\mathbf{D}_{-r}$ surgen de forma natural dada la notación que se está utilizando, estas matrices no se usan para entrenar ningún modelo, en su lugar se plantea utilizar **D** en todos los casos, toda vez que **D** es independiente de la constante de regularización C y es en últimas la matriz que se utiliza para calcular el modelo final.

Entradas: $\eta_c = \{\{(\mathbf{x}_j, \mathbf{t}_j)\}_{j=1}^{N} \mid \mathbf{x}_j \in \mathbb{R}^n, \zeta_j \in \{1, 2, ..., m\} \rightarrow$ Conjunto de datos
$L \rightarrow$ Número de nodos ocultos
$q \rightarrow$ Número de nodos de compresión ($q \in \aleph$, tal que $q \leq L$)
$C \rightarrow$ Constante de regularización
$g(.) \rightarrow$ Función de activación de los nodos ocultos

Salidas: $\{\mathbf{w}_i, b_i\}_{i=1}^{L} \rightarrow$ Pesos y bias ocultos
$\mathbf{B}_\epsilon \rightarrow$ Solución de MEC-ELM (Matriz de pesos de salida)

1 Determinar la matriz de salidas deseadas $\mathbf{T} = [\mathbf{t}_1 \quad \mathbf{t}_2 \quad ... \quad \mathbf{t}_N]^T$ donde $\mathbf{t}_j = [t_{j,1} \quad t_{j,2} \quad ... \quad t_{j,m}]^T$ para $j = 1, 2, ..., N$ se obtiene codificando ζ_j de acuerdo con

$$t_{j,v} = \begin{cases} 1 & si \quad v = \zeta_j \\ 0 & si \quad v \neq \zeta_j \end{cases} \qquad v = 1, 2, ..., m$$

2 Asignar aleatoriamente los pesos $\mathbf{w}_i$ y los bias b_i para $i = 1, 2, ..., L$.

3 Calcular la matriz **H**. Seguidamente crear k particiones de H y sus respectivas particiones de **T**, tales que

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}_1^T & \mathbf{H}_2^T & ... & \mathbf{H}_K^T \end{bmatrix}^T$$
$$\mathbf{T} = \begin{bmatrix} \mathbf{T}_1^T & \mathbf{T}_2^T & ... & \mathbf{T}_K^T \end{bmatrix}^T$$

4 Calcular las matrices auxiliares **D**, **W** y **Z**.

5 Para cada C_a tal que $a = 1, 2, .., R$, siendo C_a uno de los R valores de C:

5.1 Para cada r tal que $r = 1, 2, .., K$:

5.1.1 Calcular las matrices auxiliares $\mathbf{W}_{-r}$ y $\mathbf{Z}_{-r}$.

5.1.2 Calcular la solución $\mathbf{B}_{-r}^{(a)}$ como

$$\mathbf{B}_{-r}^{(a)} = C\mathbf{W}_{-r} - C\mathbf{D}^T \left(\frac{\mathbf{I}}{C} + \mathbf{Z}_{-r}\mathbf{D}^T\right)^{-1} \mathbf{Z}_{-r}\mathbf{W}_{-r}.$$

5.1.3 Determinar la exactitud ACC_a (u otra medida de rendimiento) de la solución $\mathbf{B}_E^{(a)}$ configurando el modelo como clasificador MEC-ELM y usando $\mathbf{H}_V$ para validar.

5.1.4 Actualizar el acumulador de rendimiento REN_a

$$REN_a = REN_a + ACC_{a,r}.$$

6 Calcular la solución final $\mathbf{B}_\epsilon$ tal que

$$\mathbf{B}_\epsilon = C\mathbf{W} - C\mathbf{D}^T \left(\frac{\mathbf{I}}{C} + \mathbf{Z}\mathbf{D}^T\right)^{-1} \mathbf{Z}\mathbf{W}$$

donde C_μ es la constante de regularización con la que se obtuvo el mayor rendimiento acumulado. Esto es $C_\mu \in \{C_1, C_2, ..., C_R\}$, tal que el índice μ esta dado por

$$\mu = \underset{a \in \{1, 2, ..., R\}}{\arg\max} \; REN_a.$$

Tabla 4.1. Entrenamiento de un clasificador MEC-ELM usando validación cruzada de K iteraciones

Como se mencionó anteriormente, la solución del modelo final se calcula en función de **D**, **W** y **Z**. En lo que respecta a las demás soluciones, estas se obtienen en función de **D**, $\mathbf{W}_{-r}$ y $\mathbf{Z}_{-r}$ dependiendo de r. Estas matrices se pueden calcular fácilmente como

$$\mathbf{Z}_{-r} = \mathbf{Z} - \mathbf{Z}_r \tag{4.3}$$

y

$$\mathbf{W}_{-r} = \mathbf{W} - \mathbf{W}_r. \tag{4.4}$$

Lo anterior implica que dado un valor de r, las matrices $\mathbf{W}_{-r}$ y $\mathbf{Z}_{-r}$ se utilizan R veces (una vez por cada valor de C) y la matriz **D** se utiliza $RK + 1$ veces. Por tanto, aunque a lo largo de todo el procedimiento se ajustan $R \times K + 1$ modelos (para seleccionar C) solo se deben invertir $R \times K + 2$ matrices de tamaño $q \times q$ y no $2(R \times K + 1)$ como sería necesario si se aplicara el método de validación cruzada de la forma convencional.

La versión análoga del procedimiento planteado en la tabla 4.1 con ELM (para grandes conjuntos de entrenamiento) se presenta en el anexo B.2. Ahora bien, la implementación de MEC-ELM requiere invertir $RK + 2$ matrices de tamaño $q \times q$ y la implementación de ELM requiere invertir $RK + 1$ matrices de tamaño $L \times L$, por lo que en términos computacionales, resulta mucho más indicado usar la técnica de validación cruzada de K iteraciones con MEC-ELM, que con ELM, siempre y cuando el número de nodos de compresión q sea menor al número de nodos ocultos L.

4.2 Desarrollo de los experimentos

A continuación, se presenta una serie de experimentos basados en simulación relacionados con problemas de clasificación de tipo binario y multiclase. Las pruebas se desarrollaron en un procesador AMD de 2.70 GHz y haciendo uso del lenguaje de programación R. Los datos utilizados están disponibles en las bases "*UCI Machine Learning Repository*" (Dua y Taniskidou, 2017) y el portal LIBSVM (Chang y Lin, 2011).

Previo a cada prueba, se llevó a cabo una etapa de preprocesamiento en la que el conjunto de entrenamiento se normalizó de modo que las entradas tengan valores en el rango [-1,1].

4.2.1 Efecto general de los nodos de compresión

Los nodos de compresión en un modelo MEC-ELM, se encargan de comprimir la información proveniente de la capa oculta, y como se demostró en la sección 3.2.1, dependiendo de su número puede reducirse la complejidad computacional en el entrenamiento. Como consecuencia de esto, surge un compromiso entre rendimiento y complejidad computacional, que debe ser analizado en busca de determinar las ventajas prácticas de MEC-ELM frente a ELM.

El análisis que se presenta a continuación se basa en dos tipos de datos usados comúnmente en problemas de "*benchmarking*". El primer conjunto de datos denominado "*Skin Segmentation*," está relacionado con el reconocimiento de piel humana a partir de los valores R G B obtenidos al muestrear aleatoriamente pixeles de imágenes faciales (de varios grupos de personas) y de objetos comunes. El segundo conjunto de datos denominado "*Landsat Satellite*," está relacionado con la clasificación de distintos tipos de suelo a partir de los valores espectrales de una vecindad de 3×3 pixeles pertenecientes a una imagen satelital.

Descripción de las simulaciones

El proceso inició con la selección de un conjunto para entrenamiento y un conjunto para prueba a partir del total de muestras disponibles para cada caso. Las características de los conjuntos utilizados se presentan en la tabla 4.2

Conjunto de datos	# Total de Muestras	# Muestras de entrenamiento	# Muestras de prueba	Atributos	Clases
skin segmentation	245057	30000	2000	4	2
Landsat Satellite	6435	4435	2000	36	6

Tabla 4.2. Información de los datos utilizados en las pruebas

Posteriormente se entrenaron múltiples modelos MEC-ELM utilizando validación cruzada de 10 iteraciones en ambos casos (*Skin Segmentation* y *Landsat Satellite*). Las simulaciones se realizaron variando el número de nodos ocultos L y el número de nodos de compresión q. Los valores que tomaron L y q fueron tales que $L \in \psi_L$ y $q \in \psi_{Lq}$ donde:

$$\psi_L = \{\ 10,\ \ 20,\ \ 40,\ \ 80,\ \ 160,\ \ 320,\ \ 640,\ \ 1280\ \}$$

$$\psi_{Lq} = \{\ \tfrac{L}{10},\ \tfrac{2L}{10},\ \tfrac{3L}{10},\ \tfrac{4L}{10},\ \tfrac{5L}{10},\ \tfrac{6L}{10},\ \tfrac{7L}{10},\ \tfrac{8L}{10},\ \tfrac{9L}{10}\ \}$$

En lo que respecta a los valores que tomó la constante de regularización C en cada proceso de validación cruzada, estos fueron tales que $C \in \psi_C$ donde $\psi_C = \{2^k\}_{k=-14}^{14}$ para $k = -14, -13, ..., 14$.

En todos los casos la función de activación de los nodos ocultos fue tangente hiperbólica $g(a) = (e^a - e^{-a})/(e^a + e^{-a})$ y los pesos de los nodos ocultos se seleccionaron aleatoriamente en el intervalo [-1,1] con función de distribución uniforme.

El proceso de entrenamiento y prueba se repitió 50 veces por cada combinación de parámetros (L, q), midiendo en cada caso la exactitud ACC para estimar el rendimiento.

Resultados y discusión

El interés detrás de estas pruebas es analizar la generalización de múltiples arquitecturas MEC-ELM a medida que aumenta el número de nodos de compresión. De modo que los resultados por cada combinación de parámetros (L, q), corresponden al promedio de las exactitudes medidas sobre el conjunto de prueba, es decir, sobre datos no utilizados en el entrenamiento.

En la tabla 4.3 se presentan los resultados obtenidos con los datos *Skin Segmentation* y en la tabla 4.4 se presentan los resultados obtenidos con los datos *Landsat Satellite*. Todos los resultados presentados en las tablas 4.3 y 4.4 están acompañados por sus respectivos intervalos de confianza calculados al 95%, además en las mismas tablas se presentan los resultados obtenidos con ELM para las distintas arquitecturas de red.

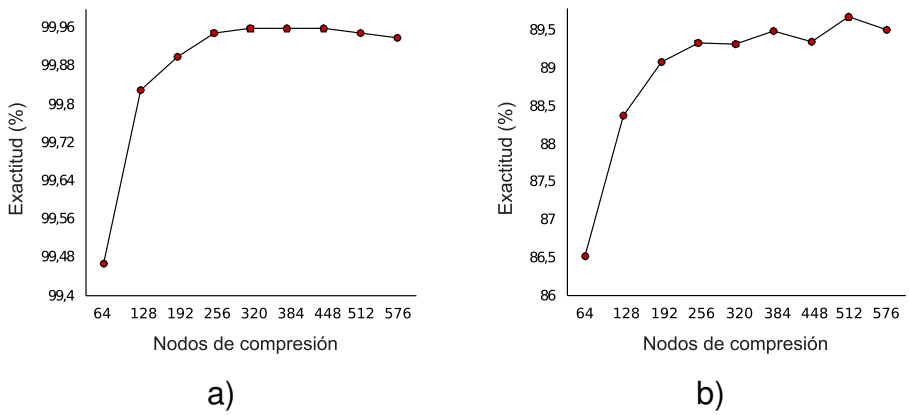
Como era de esperar, dada una arquitectura de red, entre mayor es el número de nodos de compresión mejor es el rendimiento. Sin embargo, en las arquitecturas con mayor número de nodos ocultos ($L = 640$ y $L = 1280$) esta tendencia no resulta tan clara si el número de nodos de compresión q supera el 50% de los nodos ocultos L. En otras palabras, no se aprecia un efecto significativo al incrementar los nodos de compresión más arriba del 50% de los nodos ocultos en las arquitecturas con $L = 640$ y $L = 1280$. Esta situación es común para los dos tipos de datos propuestos en esta sección y puede notarse claramente en la figura 4.1 y en la figura 4.2 respectivamente, donde se gráfica el rendimiento de las dos arquitecturas con mayor número de nodos ocultos a medida que se incrementan los nodos de compresión.

	$q = L/10$	$q = 2L/10$	$q = 3L/10$	$q = 4L/10$	$q = 5L/10$	$q = 6L/10$	$q = 7L/10$	$q = 8L/10$	$q = 9L/10$	ELM
$L = 10$	78.88±2.19	79.26±2.38	80.1±2.05	81.13±2.03	80.54±1.89	80.22±1.71	79.63±2.36	81.39±1.9	81.04±3.19	81.73±2.43
$L = 20$	86.63±1.56	85.54±1.35	86.46±1.26	87.09±1.43	87.63±1.51	89.26±1.02	87.33±1.55	88.3±1.26	89.08±0.89	90.91±1.09
$L = 40$	89.91±1.2	90.9±0.86	91.21±1	92.53±0.75	92.96±0.79	93.17±0.89	93.82±0.94	94.62±0.68	94.83±0.7	96.15±0.44
$L = 80$	93.44±0.94	93.82±0.78	96.04±0.37	96.72±0.45	97.03±0.35	97.66±0.33	97.61±0.46	97.92±0.38	98.45±0.28	98.85±0.24
$L = 160$	95.73±0.8	97.68±0.34	98.53±0.27	99.1±0.18	99.2±0.14	99.48±0.11	99.49±0.15	99.54±0.08	99.72±0.06	99.78±0.04
$L = 320$	98.46±0.3	99.32±0.13	99.58±0.08	99.78±0.07	99.8±0.06	99.86±0.04	99.85±0.07	99.86±0.05	99.88±0.05	99.95±0.02
$L = 640$	99.47±0.11	99.83±0.06	99.9±0.03	99.95±0.02	99.96±0.02	99.96±0.02	99.96±0.02	99.95±0.03	99.94±0.03	99.98±0.01
$L = 1280$	99.87±0.05	99.91±0.06	99.96±0.02	99.97±0.03	99.95±0.06	99.98±0.01	99.97±0.02	99.96±0.04	99.98±0.02	99.99±0.01

Tabla 4.3. Exactitud(%) de MEC-ELM y ELM para el conjunto de datos *Skin Segmentation*

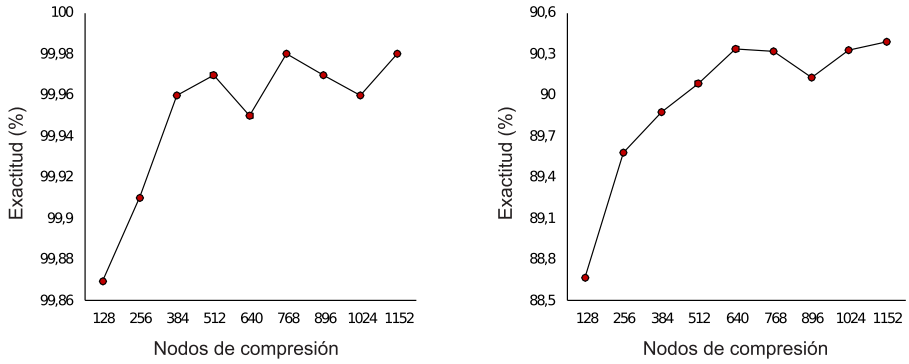
	$q = L/10$	$q = 2L/10$	$q = 3L/10$	$q = 4L/10$	$q = 5L/10$	$q = 6L/10$	$q = 7L/10$	$q = 8L/10$	$q = 9L/10$	ELM
$L = 10$	56.48±3.09	61.9±2.56	64.54±2.03	67.94±2.28	68.54±1.92	71.4±2.34	71.88±1.39	73.15±1.79	74.57±0.99	76.23±1.13
$L = 20$	60.65±3.54	64.74±2.63	70.38±2.96	74.3±1.85	76.17±1.35	78.13±1.22	78.93±0.52	79.22±0.65	79.22±0.74	81.14±0.31
$L = 40$	58.58±3.01	71.64±1.90	78.84±0.83	80.94±0.58	81.45±0.36	82.17±0.70	82.57±0.42	82.64±0.39	82.59±0.33	83.59±0.33
$L = 80$	68.47±2.59	79.63±0.92	83.25±0.34	83.6±0.42	84.38±0.41	84.75±0.28	84.47±0.38	85.04±0.38	84.76±0.37	85.96±0.23
$L = 160$	74.72±1.73	84.52±0.39	85.51±0.32	86.16±0.27	86.72±0.29	86.62±0.29	87.08±0.28	87.11±0.26	87.17±0.23	87.76±0.18
$L = 320$	82.02±0.58	86.7±0.31	87.53±0.25	87.95±0.25	88.16±0.19	88.41±0.24	88.61±0.19	88.67±0.14	88.67±0.23	88.95±0.09
$L = 640$	86.54±0.30	88.39±0.24	89.09±0.19	89.34±0.14	89.32±0.22	89.5±0.19	89.36±0.22	89.69±0.15	89.52±0.16	89.86±0.13
$L = 1280$	88.67±0.24	89.58±0.15	89.88±0.18	90.09±0.17	90.34±0.18	90.32±0.14	90.13±0.13	90.33±0.21	90.39±0.11	90.42±0.16

Tabla 4.4. Exactitud(%) de MEC-ELM y ELM para el conjunto de datos *Landsat Satellite*



a) b)

Figura 4.1. Rendimiento de MEC-ELM variando el número de nodos de compresión q y $L = 640$: a) *Skin Segmentation* b) *Landsat Satellite*



a) b)

Figura 4.2. Rendimiento de MEC-ELM variando el número de nodos de compresión q y $L = 1280$: a) *Skin Segmentation* b) *Landsat Satellite*

Esta tendencia resulta conveniente, puesto que, si se aumenta el número de nodos de compresión hasta igualar el número de nodos ocultos, la solución de MEC-ELM se convierte en la solución de ELM. Por tanto, dicha tendencia implica poder alcanzar rendimientos similares a los de ELM, usando un modelo MEC-ELM con el mismo número de nodos ocultos L y entrenado con un número de nodos de compresión q tal que $q < L$, siempre y cuando L sea lo suficientemente grande. Esto se puede verificar en la figura 4.3 donde se compara el rendimiento de MEC-ELM con el rendimiento de ELM. La figura 4.3a muestra el rendimiento de dos modelos MEC-ELM (entrenados utilizando el 50% y el 60% de los nodos ocultos como nodos de compresión) en contraste con los correspondientes modelos ELM y utilizando el conjunto de datos *Skin*

Segmentation. En la figura 4.3b se compara el rendimiento de los mismos modelos, pero utilizando el conjunto de datos *Landsat Satellite*. Claramente las gráficas presentadas muestran un comportamiento similar en términos de exactitud para los modelos MEC-ELM y ELM, nótese además como dicha similitud aumenta mucho más a medida que se incrementan los nodos ocultos.

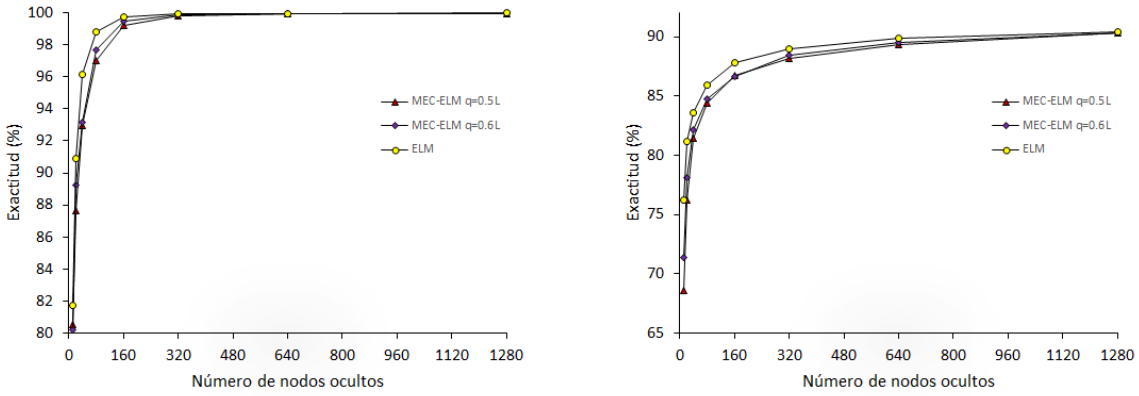


Figura 4.3. Rendimiento de MEC-ELM ($q = 0.5L$ y $q = 0.6L$) Vs ELM usando validación cruzada de 10 iteraciones con $C \in \psi_C$ donde $\psi_C = \{2^k\}_{k=-14}^{14}$ para $k = -14, -13, ..., 14$: a) *Skin Segmentation* b) *Landsat Satellite*

4.2.2 Pruebas de consistencia

En esta sección se comparan los resultados de MEC-ELM y los obtenidos con ELM para diferentes problemas de clasificación. El rendimiento de MEC-ELM se puso a prueba utilizando una colección de conjuntos de entrenamiento relacionados con aplicaciones de clasificación binaria y multiclase. La información referente al número de datos, atributos y clases se presentan en la Tabla 3.

Todos los modelos utilizados en las simulaciones eran de 1000 nodos ocultos con funciones activación tangente hiperbólica $g(a) = (e^a - e^{-a})/(e^a + e^{-a})$ y con pesos ocultos seleccionados aleatoriamente en el intervalo [-1,1] con función de distribución uniforme. Para los modelos MEC-ELM simulados, el número de nodos de compresión se fijó en la mitad del número de nodos ocultos, es decir 500 en todos los casos.

El parámetro de regularización C de MEC-ELM y ELM, se seleccionó mediante pruebas preliminares utilizando validación cruzada de 10 iteraciones. En este proceso los valores que tomó la constante C fueron tales que $C \in \psi_C$ donde $\psi_C = \{2^{(2k+1)}\}_{-7}^{1}$

Conjunto de datos	# de muestras de entrenamiento	# de muestras de prueba	Atributos	Clases
Mushroom	4062	4062	22	2
SVDguide1	3089	4000	4	2
Magic	9510	9510	10	2
COD RNA	29768	29767	8	2
Spambase	2301	2300	57	2
Adult	6414	26147	14	2
Im. Segment	1155	1155	18	7

Tabla 4.5. Características de los conjuntos de entrenamiento usados en las pruebas de consistencia

para $k = -7, -6, ..., 1$. Posteriormente, se realizaron cincuenta simulaciones con MEC-ELM y cincuenta simulaciones con ELM. Para medir el rendimiento se obtuvo la exactitud con las muestras de entrenamiento y con las muestras de prueba.

Resultados y discusión

El promedio y la respectiva desviación estándar de los resultados para cada conjunto de datos se presentan en la tabla 4.6. Los resultados de rendimiento en entrenamiento y prueba también se pueden contrastar con los obtenidos por Inaba *et al.* (2018).

El rendimiento en la fase de prueba permite estimar qué tan buena es la generalización de un modelo, es decir, qué tan bien responde ante patrones que no han sido utilizados en el entrenamiento. Ahora bien, con base en el promedio y la desviación estándar de los resultados de exactitud en la fase de prueba, se puede decir que no hay diferencias significativas en el rendimiento de MEC-ELM y ELM y como tal, que ambos modelos generalizan de forma similar. Adicionalmente, los resultados indican altos rendimientos con poca varianza, lo que da cuenta de la buena estabilidad de MEC-ELM en las pruebas.

En lo que respecta al rendimiento en la fase de entrenamiento, los resultados tampoco muestran diferencias significativas entre ELM y MEC-ELM. El rendimiento en la fase de entrenamiento indica qué tan acertado es el modelo al clasificar las muestras que se utilizaron para entrenarlo. Cabe resaltar, que en la práctica lo que se busca es que un modelo aprenda a dar respuestas correctas ante patrones jamás vistos en el entrenamiento, es decir, que generalice.

Datos	Algoritmo	Tiempo (s)		Rendimiento (%)				C
		Entrenamiento	Prueba	Entrenamiento		Prueba		
				Tasa	Dev. Estd	Tasa	Dev. Estd	
Mushroom	MEC-ELM	8.91	1.60	100	0.00	99.99	9.59×10^{-5}	2^{-7}
	ELM	15.55	1.67	100	0.00	99.99	5.69×10^{-5}	2^{-1}
SVDguide1	MEC-ELM	6.90	1.053	97.36	0.04	96.61	0.05	2^{-1}
	ELM	11.99	1.13	97.37	0.04	96.61	0.06	2^{-1}
Magic	MEC-ELM	15.81	2.32	87.20	0.09	86.47	0.11	2^{-5}
	ELM	31.48	2.31	88.22	0.08	86.54	0.14	2^{-1}
COD RNA	MEC-ELM	42.63	6.49	95.18	0.04	95.09	0.06	2^{-3}
	ELM	94.85	6.54	95.22	0.04	95.15	0.05	2^{-3}
Spambase	MEC-ELM	5.67	0.58	95.65	0.22	93.01	0.27	2^{-7}
	ELM	8.95	0.57	95.35	0.18	93.09	0.26	2^{-7}
Adult	MEC-ELM	12.01	9.18	84.93	0.15	83.63	0.09	2^{-13}
	ELM	22.08	9.11	84.92	0.14	83.61	0.09	2^{-13}
Im. Segment	MEC-ELM	4.09	0.28	98.85	0.05	95.96	0.28	1
	ELM	5.29	0.28	98.57	0.05	96.08	0.29	2

Tabla 4.6. Resultados de MEC-ELM (con $q = 500$ y $L = 1000$) y ELM (con $L = 1000$) para todos los conjuntos de datos

En la tabla 4.6 también se muestra los tiempos que en promedio tomaron la fase de prueba y la fase de entrenamiento para todos los casos de clasificación tratados. Nótese que los tiempos de entrenamiento de MEC-ELM son menores a los de ELM y como era de esperarse, los tiempos en la fase de pruebas en ambos modelos es relativamente igual.

Adicionalmente, para cada uno de los modelos entrenados con el conjunto de datos "*Image Segmentation*" se obtuvieron las matrices de confusión en la fase de prueba, los resultados obtenidos se promediaron y se calcularon las respectivas desviaciones estándar. Las matrices de confusión resultantes para MEC-ELM y ELM se presentan en la figura 4.4. Como puede notarse, las clases que se pretende identificar son: ladrillo, cielo, follaje, cemento, ventana, camino y hierba, dado que el conjunto de datos *Image Segmentation* está relacionado con la clasificación de exteriores a partir de pixeles obtenidos de imágenes captadas al aire libre.

En la figura 4.4 las casillas rosadas indican el promedio (y desviación estándar) de los datos que fueron correctamente clasificados y las casillas blancas indican el promedio de los errores de clasificación en función de la clase predicha y la de referencia.

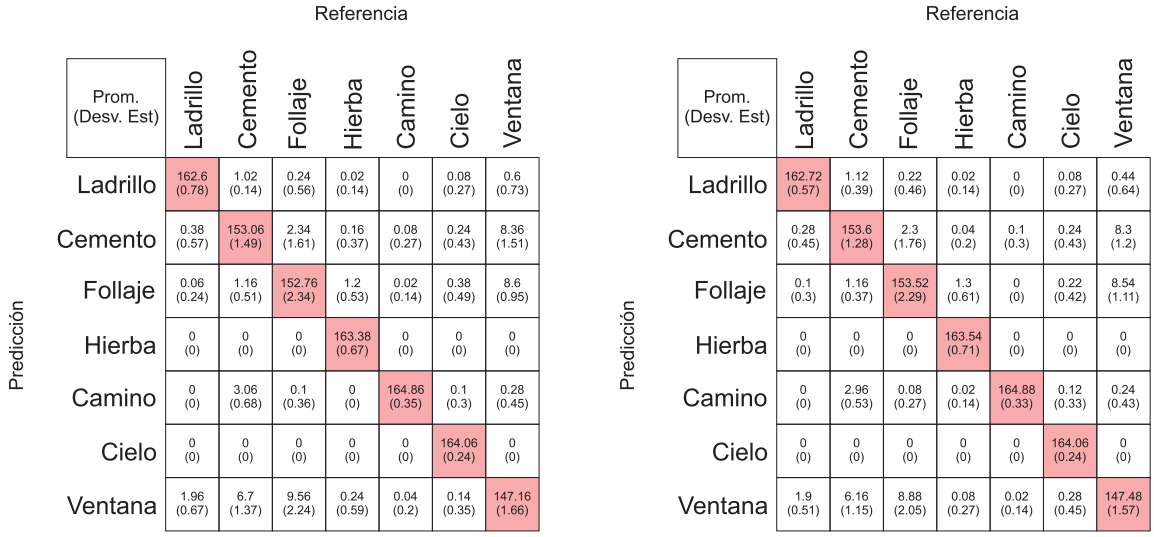


Figura 4.4. Matrices de confusión con el conjunto de entrenamiento "Image Segmentation": a) MEC-ELM b) ELM

Luego de comparar casilla a casilla las matrices de confusión de MEC-ELM y ELM, es claro que existe una alta semejanza estadística entre ambos modelos. Este hecho

se puede observar con más claridad aún en las figuras 4.5, 4.6 y 4.7, donde se presentan respectivamente los resultados de precisión, sensibilidad y especificidad para ambos modelos.

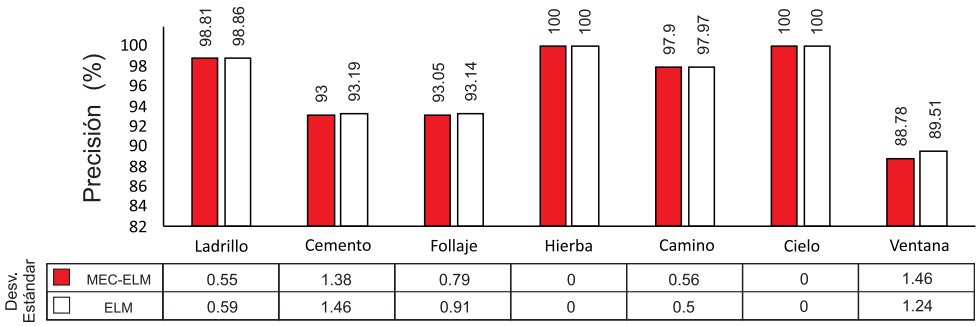


Figura 4.5. Resultados de precisión para MEC-ELM y ELM con el conjunto de datos "*Image Segmentation*"

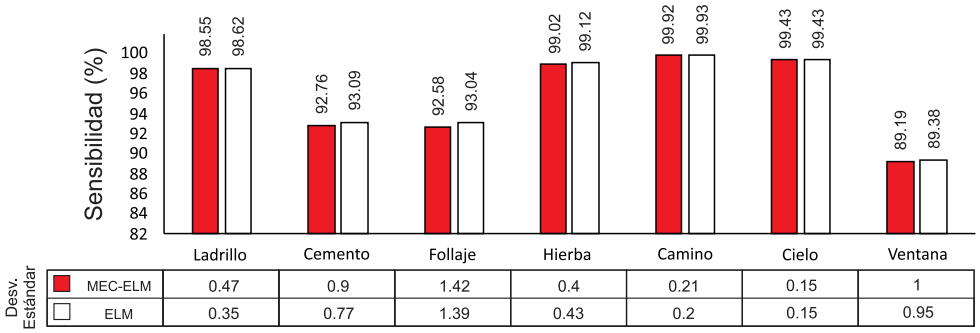


Figura 4.6. Resultados de sensibilidad para MEC-ELM y ELM con el conjunto de datos "*Image Segmentation*"

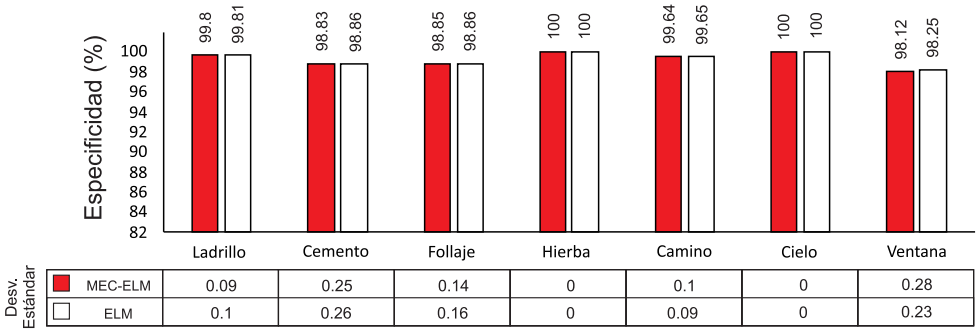


Figura 4.7. Resultados de especificidad para MEC-ELM y ELM con el conjunto de datos "*Image Segmentation*"

Las mediciones de precisión, sensibilidad y especificidad se realizan en función de una clase específica, por lo que es necesario elegir una clase positiva y una clase negativa. La clase positiva es la que se pretende evaluar y la clase negativa son todas las demás. Ahora bien, los resultados mostrados en la figura 4.5, indican que ambos modelos alcanzaron precisiones altas en lo que respecta a las clases ladrillo, hierba, camino y cielo, nótese que para las clases hierba y cielo los resultados son incluso del 100%. Vale destacar que la precisión aumenta cuando disminuyen los falsos positivos en la predicción de positivos, por tanto, una precisión del 100% implica que todas las muestras que se predijeron como pertenecientes a la clase positiva son correctas, esto se puede verificar en las filas hierba y cielo de las matrices de confusión presentadas en la figura 4.4. En lo que respecta a las clases cemento, follaje y ventana, la precisión fue un poco más baja, pero no se evidencian diferencias estadísticas significativas en el rendimiento de ambos modelos.

Al igual que en las mediciones de precisión, se obtuvieron sensibilidades altas para las ladrillo, hierba, camino y cielo y, sensibilidades un poco más bajas para las clases cemento, follaje y ventana. Sin embargo, nótese en la figura 4.6, que las diferencias de sensibilidad entre ambos modelos son aún más estrechas. Una sensibilidad alta indica una alta confiabilidad cuando el modelo predice que una muestra pertenece a la clase positiva, y por tanto, entre más estrecha es la diferencia de sensibilidad entre MEC-ELM y ELM, mas similar es su capacidad para evitar falsos negativos al clasificar muestras de clase positiva.

Como se puede observar la figura 4.7, la diferencia en términos de especificidad entre ambos modelos es despreciable. Adicionalmente, se obtuvieron especificidades muy altas (iguales o superiores al 98.12%) para todas las clases, lo cual indica una alta confiabilidad al predecir que una muestra no pertenece a la clase positiva, y como tal, alta capacidad para evitar falsos positivos al clasificar muestras de clase negativa.

Con base en los resultados obtenidos se puede decir que no existen diferencias estadísticas en el rendimiento ambos modelos. Lo cual evidencia las ventajas practicas del algoritmo MEC-ELM respecto a un referente tan importante en la literatura de las redes SLFN como lo es ELM, toda vez que los tiempos de entrenamiento, en general son menores para MEC-ELM, esto se puede verificar en la tabla 4.6.

4.3 Resumen de capítulo

En este capítulo se presentó una implementación de MEC-ELM que permite manejar (mediante la reutilización de matrices) el costo computacional de la técnica de validación cruzada de K iteraciones. Dicho procedimiento aumenta las ventajas prácticas de MEC-ELM respecto a ELM cuando $q < L$, dado que involucra la inversión de $RK + 2$ matrices de tamaño $q \times q$ en comparación con la implementación análoga de ELM que involucra la inversión de $RK + 1$ matrices de tamaño $L \times L$, donde R es el número de parámetros a probar y K el número de particiones del conjunto de entrenamiento.

Se estudió el efecto general de los nodos de compresión de MEC-ELM con base en dos problemas de clasificación reales, uno de tipo binario relacionado con el reconocimiento de piel humana y otro de tipo multiclase relacionado con identificación de suelos. Los resultados indican en ambos casos que el rendimiento (en términos de exactitud) de un modelo MEC-ELM de L nodos ocultos, aumenta a medida que se incrementan los nodos de compresión, alcanzando valores máximos similares a los de ELM con un número de nodos de compresión q menor a L. Más aún, en las arquitecturas con más nodos ocultos, MEC-ELM alcanzó rendimientos similares a los de ELM usando solo el 50% de los nodos ocultos como nodos de compresión.

Con base en múltiples problemas de clasificación reales de tipo binario y multiclase se pudo verificar que ELM logra alcanzar rendimientos similares a los de ELM en menores tiempos de entrenamiento. Dichas pruebas muestran la consistencia y las ventajas prácticas de MEC-ELM a la hora de abordar problemas de clasificación que involucran grandes conjuntos de entrenamiento.

Capítulo 5

Clasificación de enlaces ópticos: un análisis comparativo entre MEC-ELM y ELM

Los enlaces ópticos (*"lightpaths"*) permiten transportar información desde el origen al destino en una red de comunicaciones ópticas. En redes transparentes, los enlaces ópticos se establecen sin cambiar el dominio óptico de la señal al dominio eléctrico y, por tanto, no existe regeneración de la misma en los nodos intermedios. Una vez los enlaces ópticos son establecidos, es necesario seleccionar una ruta para cada uno y asignarle una longitud de onda, ésto es referido como el problema de asignación de ruta y longitud de onda o problema RWA ("*Route and Wavelength Assingment*"). Ahora bien, el hecho de que no haya regeneración de la señal, implica necesariamente que se deban tomar medidas adicionales para evitar que las deficiencias del medio de propagación (en este caso fibra óptica) afecten considerablemente el rendimiento de la red (Azodolmolky *et al.*, 2009b).

Por lo anterior, es necesario tomar en consideración las deficiencias de la capa física al diseñar y operar una red de comunicaciones ópticas. En este sentido, desde hace algunos años, se vienen realizando una serie de esfuerzos investigativos, que buscan desarrollar avanzados algoritmos RWA en procura de asegurar altas calidades de transmisión o QoT (*Quality of Transmission*") (Dizdarević *et al.*, 2016).

Particularmente, Azodolmolky *et al.* (2009a) presentan el desarrollo del proyecto DISCONET ("*Dynamic Impairment Constraint Optical Networking*") cuyo objetivo es el diseño y puesta en marcha de una herramienta para la planeación y operación de redes ópticas transparentes. Dicha herramienta incorpora un estimador de QoT basado en modelos lineales y no lineales de la capa física llamado Q-Tool. Con la herramienta Q-Tool se pueden hacer estimaciones relativamente buenas del factor Q, esto permite evaluar el rendimiento de los enlaces ópticos, de modo que puedan ser seleccionados o descartados antes de ser establecidos. Sin embargo, su funcionamiento tiene algu-

nas limitaciones relacionadas con su complejidad computacional, por lo que su uso no está indicado en aquellas aplicaciones que requieran control en tiempo real, además su utilización está restringida únicamente para redes OOK (*On-Off Keying*) de 10 Gb/s (Simeonidou *et al.*, 2010).

Un enfoque alternativo para predecir la calidad de transmisión de un enlace óptico, es utilizar modelos que aprendan a partir de muestras, con lo que se puede explotar la información almacenada en las bases de datos. La ventaja consiste en que, una vez obtenido el modelo, es posible estimar la calidad de transmisión del enlace sin recurrir a cálculos excesivamente complejos y con un grado relativamente alto de precisión. Algunas iniciativas en este sentido, pueden ser consultadas en los trabajos de Jiménez *et al.* (2013) y Gómez (2017).

Jimenez, et al. (2013) proponen un estimador de QoT cognitivo, el cual permite clasificar enlaces ópticos utilizando la técnica de razonamiento basado en casos o CBR (*"Case Base Reasoning"*), alcanzando rendimientos por encima del 98$\%$ y mostrando además, que las técnicas de aprendizaje y olvido permiten optimizar el modelo. Por su parte, Gomez (2017) presenta un análisis comparativo de diversas técnicas de aprendizaje automático para clasificar la calidad de transmisión de enlaces ópticos, entre las técnicas estudiadas se incluyen: máquinas de vectores soporte, regresión logística, árboles de decisión y bosques aleatorios, destacándose fundamentalmente las máquinas de vectores soporte por su rendimiento y tiempo de cálculo.

En este capítulo se presenta un análisis comparativo entre los resultados de MEC-ELM y ELM para dos escenarios de simulación relacionados con clasificación de enlaces ópticos. La implementación de MEC-ELM utilizada en el desarrollo de las pruebas se describe a continuación.

5.1 Implementación de MEC-ELM: método de retención

La técnica de validación cruzada de K iteraciones presentada en el capítulo 4 requiere calcular $K \times R + 1$ modelos, siendo R el número de valores que toma la constante de regularización C durante el entrenamiento. Por su parte, el método de retención requiere únicamente R modelos y adicionalmente un conjunto de datos los suficientemente extenso para determinar el valor más adecuado de C, esto supone un ahorro computacional considerable en aquellas aplicaciones en las que el número de datos no es una limitación. Por lo anterior, a continuación se presenta una implementación de

MEC-ELM que utiliza el método de retención y que permite manejar el costo computacional mediante la reutilización de matrices. La implementación propuesta se presenta en la tabla 5.1. En el proceso especificado se considera la siguiente notación:

Entradas:	$\eta_c = \{\{(\mathbf{x}_j, \mathbf{t}_j)\}_{j=1}^{N} \mid \mathbf{x}_j \in \mathbb{R}^n, \zeta_j \in \{1, 2, ..., m\} \rightarrow$ Conjunto de datos $L \rightarrow$ Número de nodos ocultos $q \rightarrow$ Número de nodos de compresión ($q \in \aleph$, tal que $q \leq L$) $C \rightarrow$ Constante de regularización $g(.) \rightarrow$ Función de activación de los nodos ocultos
Salidas:	$\{\mathbf{w}_i, b_i\}_{i=1}^{L} \rightarrow$ Pesos y bias ocultos $\mathbf{B}_\epsilon \rightarrow$ Solución de MEC-ELM (Matriz de pesos de salida)
1	Determinar la matriz de salidas deseadas $\mathbf{T} = \begin{bmatrix}\mathbf{t}_1 & \mathbf{t}_2 & ... & \mathbf{t}_N\end{bmatrix}^T$ donde $\mathbf{t}_j = \begin{bmatrix}t_{j,1} & t_{j,2} & ... & t_{j,m}\end{bmatrix}^T$ para $j = 1, 2, ..., N$ se obtiene codificando ζ_j de acuerdo con $$t_{j,v} = \begin{cases} 1 & si \quad v = \zeta_j \\ & \qquad v = 1, 2, ..., m \\ 0 & si \quad v \neq \zeta_j \end{cases}$$
2	Asignar aleatoriamente los pesos $\mathbf{w}_i$ y los bias b_i para $i = 1, 2, ..., L$.
3	Calcular la matriz de salidas de la capa oculta **H** e identificar las matrices $\mathbf{H}_E$ y $\mathbf{H}_V$, así como sus respectivas matrices de salida deseadas $\mathbf{T}_E$ y $\mathbf{T}_V$ tales que $$\mathbf{H} = \begin{bmatrix}\mathbf{H}_E^T & \mathbf{H}_V^T\end{bmatrix}^T$$ $$\mathbf{T} = \begin{bmatrix}\mathbf{T}_E^T & \mathbf{T}_V^T\end{bmatrix}^T$$
4	Calcular las matrices auxiliares $\mathbf{D}_E$, $\mathbf{W}_E$ y $\mathbf{Z}_E$
5	Para cada C_a tal que $a = 1, 2, .., R$, siendo C_a uno de los R valores de C:
5.1	Calcular la solución $\mathbf{B}_E^{(a)}$ como $$\mathbf{B}_E^{(a)} = C\mathbf{W}_E - C\mathbf{D}_E^T \left(\frac{\mathbf{I}}{C} + \mathbf{Z}_E\mathbf{D}_E^T\right)^{-1} \mathbf{Z}_E\mathbf{W}_E$$
5.1	Determinar la exactitud ACC_a (u otra medida de rendimiento) de la solución $\mathbf{B}_E^{(a)}$ configurando el modelo como clasificador MEC-ELM y usando $\mathbf{H}_V$ para validar.
6	Seleccionar la solución final $\mathbf{B}_\epsilon$ tal que $$\mathbf{B}_\epsilon = \mathbf{B}_E^{(\mu)}$$ donde μ es el índice del modelo que obtuvo el mayor rendimiento. Esto es $$\mu = \underset{a \in \{1, 2, ..., R\}}{\arg\max} \; ACC_a.$$

Tabla 5.1. Entrenamiento de un clasificador MEC-ELM usando el método de retención

$\mathbf{H}_E$: matriz de salidas de la capa oculta para el conjunto de entrenamiento.

$\mathbf{H}_V$: matriz de salidas de la capa oculta para el conjunto de validación.

$\mathbf{T}_E$: matriz de salidas deseadas del conjunto de entrenamiento (codificadas según la ecuación 2.23).

$\mathbf{T}_V$: matriz de salidas deseadas del conjunto de validación (codificadas según la ecuación 2.23).

$\mathbf{D}_E$, $\mathbf{W}_E$ y $\mathbf{Z}_E$: matrices auxiliares tales que $[\mathbf{D}_E, \mathbf{W}_E, \mathbf{Z}_E] = \Psi(\mathbf{H}_E, \mathbf{T}_E, q)$.

La implementación de MEC-ELM planteada permite evaluar R valores de la constante de regularización C, esto implica ajustar R modelos distintos. Los modelos obtenidos se evalúan usando el conjunto de validación y finalmente se selecciona el de mejor rendimiento. Es importante tener en cuenta que los pesos de la capa oculta son independientes del conjunto de entrenamiento y del conjunto de validación, por lo que todos los modelos pueden tener la misma capa oculta y por consiguiente las mismas matrices $\mathbf{H}_E$, $\mathbf{D}_E$, $\mathbf{W}_E$ y $\mathbf{Z}_E$ en el entrenamiento. Lo anterior representa un importante ahorro computacional, dado que después de ajustar el primer modelo, por cada modelo adicional, la operación más compleja es la inversión de una única matriz de tamaño $q \times q$. Es similar en la versión para grandes conjuntos de entrenamiento de ELM (ver implementación de ELM usando el método de retención en el anexo B.1). Sin embargo, en el caso de ELM, por cada modelo adicional la operación computacionalmente más compleja es la inversión de una matriz de tamaño $L \times L$. En este sentido resulta más indicado usar el método de retención en el entrenamiento con MEC-ELM que en el entrenamiento con ELM teniendo en cuenta que $q \leq L$.

5.2 Descripción de los conjuntos de entrenamiento

Se contó con dos conjuntos de entrenamiento (φ_{DT32}, φ_{DT64}) obtenidos al simular la red Deutsche Telekom DT de 14 nodos para 32 y 64 longitudes de onda por link (Azodolmolky *et al.*, 2011). Las características de entrada corresponden a identificadores de fuente, destino, link, longitud, entre otros. La salida en todos los casos es el factor Q (dB) de los enlaces ópticos establecidos y fue obtenida usando la herramienta Q-Tool.

Sea φ_{DTx} cualquiera de los conjuntos de entrenamiento disponibles. Entonces

$$\varphi_{DTx} = \{\{(\mathbf{x}_j, Q_j)\}_{j=1}^{N_T}, \mathbf{x}_j \in \mathbb{R}^n, Q_j \in \mathbb{R}\}, \tag{5.1}$$

donde, $\mathbf{x}_j$ es el vector de características para el j-ésimo *lightpath*, Q_j su respectivo factor Q (dB) y N_T el número total de patrones disponibles. Ahora bien, como el objetivo era entrenar clasificadores MEC-ELM y ELM para que discriminen entre *lightpaths* de alta QoT y *lightpaths* de baja QoT, fue necesario procesar las salidas deseadas de φ_{DTx} para obtener otro conjunto ψ_{DTx}, tal que

$$\psi_{DTx} = \{\{(\mathbf{x}_j, \zeta_j)\}_{j=1}^{N_T}, \mathbf{x}_j \in \mathbb{R}^n, \zeta_j \in \{A, B\}\}, \tag{5.2}$$

donde ζ_j es una identificador categórico que representa la calidad del j-ésimo *lightpath*, de tal modo que si $\zeta_j = A$, entonces la calidad de transmisión del j-ésimo *lightpath* es alta y por el contrario si $\zeta_j = B$, entonces la calidad de transmisión del j-ésimo *lightpath* es baja. Los datos son etiquetados siguiendo el mismo criterio utilizado por Jimenez, et al. (2013), esto es

$$t_{j,v} = \begin{cases} A & si \quad Q_j > 16.9 \quad dB \\ B & si \quad Q_j \leq 16.9 \quad dB \end{cases} \tag{5.3}$$

con esto se asegura que la BER o tasa de errores de bit para los *lightpaths* de alta calidad este por debajo de 10^{-12}.

Los conjuntos utilizados en las pruebas se obtienen al procesar las salidas deseadas de los conjuntos de datos φ_{DT32} y φ_{DT64} según la ecuación 5.3. Con lo que se obtienen los conjuntos ψ_{DT32} y ψ_{DT64} para la red DT con 32 y 64 lambdas respectivamente.

Posteriormente, cada conjunto de datos (ψ_{DT32} y ψ_{DT64}) se dividió en dos subconjuntos reservados para tareas de entrenamiento y prueba, la partición se realizó en forma aleatoria, pero cuidando que se conserve la proporción de clases de los conjuntos originales, los detalles se presentan en la tabla 5.2

Conjunto de Datos	# características entrada	# clases	# muestras para tareas de entrenamiento	# muestras para tareas de prueba	Porcentaje clase A	Porcentaje clase B
ψ_{DT32}	79	2	200000	120000	99.13 %	0.87 %
ψ_{DT64}	79	2	320000	120000	98.98 %	1.02 %

Tabla 5.2. Características de los datos usados en las simulaciones

5.3 Desarrollo de las simulaciones

Las simulaciones se desarrollaron en el entorno de programación R y utilizando un procesador AMD de 2.70 GHz. Previo a todas las pruebas se llevó a cabo una etapa de preprocesamiento en la que el conjunto de entrenamiento se normalizó de modo que las entradas tengan valores en el rango [-1,1].

En todos los casos, las funciones de activación $h(a)$ de los nodos ocultos fueron sigmoides tales que

$$h(a) = \frac{1}{1 + e^{-a}} \tag{5.4}$$

y los pesos ocultos se seleccionaron aleatoriamente en el intervalo [-1,1] con función de distribución uniforme.

Para seleccionar la arquitectura más óptima, se realizó una serie de pruebas iniciales con ELM utilizando validación cruzada de 10 iteraciones, las muestras de entrenamiento y de prueba se seleccionaron aleatoriamente a partir de los respectivos conjuntos reservados para tales fines (ver tabla 5.2). Los resultados obtenidos en estas pruebas se presentan en la tabla 5.3.

Conjunto de datos	# de muestras Entrenamiento	# de muestras Prueba	C	Exactitud en prueba				Arquitectura seleccionada
				$L = 500$	$L = 1000$	$L = 1500$	$L = 2000$	
ψ_{DT32}	3000	6000	$\{2^{-u}\}_{u=0}^{9}$	0.9920	0.9944	0.9952	0.9951	$L = 1500$
ψ_{DT64}	3000	6000	$\{2^{-u}\}_{u=0}^{9}$	0.9911	0.9927	0.9942	0.9942	$L = 2000$

Tabla 5.3. Pruebas iniciales con ELM usando validación cruzada de 10 iteraciones

Habiendo seleccionado las arquitecturas, se procedió a entrenar diferentes clasificadores ELM y MEC-ELM utilizando el método de retención. Para entrenar cada modelo se seleccionaron aleatoriamente N patrones de entrenamiento y 6000 patrones de validación a partir de las muestras reservadas para tareas de entrenamiento en ambos casos (ψ_{DT32} y ψ_{DT64}).

Durante las simulaciones el valor de N se incrementó de 1000 en 1000 (los patrones de validación fueron siempre 6000) y cada modelo obtenido se probó con 6000 patrones seleccionados aleatoriamente a partir de las respectivas muestras reservadas para tareas de prueba. Este proceso de entrenamiento y prueba se repitió 50 veces.

5.4 Resultados y discusión

Las medidas de rendimiento se obtuvieron en modo de ejecución utilizando los patrones de prueba y considerando a la clase A (*lightpaths* de alta calidad) como la clase positiva. Las mediciones obtenidas corresponden a la exactitud ACC, la sensibilidad SEN, la especificidad ESP, la precisión PRE y el área bajo la curva ROC AUC. Adicionalmente, se midió también el tiempo de entrenamiento y el tiempo por muestra procesada en modo de ejecución.

El área bajo la curva ROC AUC se calcula en función de las probabilidades de pertenencia a la clase positiva. Si bien, un clasificador ELM o MEC-ELM no determina la probabilidad con la que una muestra pertenece a una clase u otra, dichas probabilidades se pueden estimar en función de las salidas de red, cabe mencionar que en este proceso no se tiene en cuenta la función de decisión.

Sea $f_L(\mathbf{x}) = \begin{bmatrix} y_1 & y_2 \end{bmatrix}$ (solo para el caso binario) la salida de red, dada una muestra **x** y, $P_T(A|\mathbf{x})$ la probabilidad de que dicha muestra pertenezca a la clase A. Entonces

$$P_T(A|\mathbf{x}) = \frac{1}{2}P_1 + \frac{1}{2}P_2 \tag{5.5}$$

donde P_1 y P_2 son estimadores parciales de la probabilidad de clase que se calculan en función de las salidas de red y_1 y y_2 respectivamente. Se definen como:

$$P_1 = \frac{1}{1 + e^{-y_1}} \tag{5.6}$$

$$P_2 = 1 - \frac{1}{1 + e^{-y_2}}. \tag{5.7}$$

Ahora bien, para calcular el AUC de un clasificador en específico dado un conjunto de prueba, se utilizó la función especial "*roc*" del paquete caret, pasando como argumentos las probabilidades de pertenencia a la clase positiva (estimadas con la ecuación 5.5) y las respectivas salidas deseadas.

Una vez finalizadas las 50 sesiones de entrenamiento y prueba, se promediaron todas las mediciones de rendimiento obtenidas y se calcularon los respectivos intervalos de confianza al 95 %. Los resultados de MEC-ELM y ELM para el conjunto de datos ψ_{DT32} se presentan en las tablas 5.4 y 5.5 respectivamente. Así mismo, los resultados de MEC-ELM y ELM para el conjunto de datos ψ_{DT64} se presentan en las tablas 5.6 y 5.7.

# de Muestras Entrenamiento	ACC (%)	AUC (%)	SEN (%)	ESP (%)	PRE (%)	T. Entrenamiento (s)	T. Prueba (ms)
1000	99.2683±0.0394	94.61±1.3809	99.888±0.0245	30.8516±3.901	99.3771±0.0441	29.1338±0.0677	0.2969±0.002
2000	99.3893±0.039	94.8624±1.3923	99.8856±0.0228	44.6566±3.677	99.5002±0.0386	31.386±0.1263	0.316±0.0034
3000	99.4433±0.0318	94.6577±1.4409	99.8766±0.0215	50.657±3.3923	99.5633±0.0334	33.558±0.0888	0.3108±0.0038
4000	99.4997±0.0222	96.5851±0.7468	99.854±0.0233	60.167±2.8901	99.6419±0.0254	35.7542±0.1594	0.3091±0.0048
5000	99.5227±0.0364	97.3201±0.9049	99.8775±0.0198	61.2709±3.0518	99.6416±0.0328	38.4704±0.1479	0.3075±0.0047
6000	99.589±0.0279	97.9641±0.4798	99.8857±0.0161	65.8577±2.6032	99.7003±0.0266	40.3346±0.3418	0.3089±0.0026

Tabla 5.4. Resultados de MEC-ELM usando ψ_{DT32} (red DT con 32 lambdas)

# de Muestras Entrenamiento	ACC (%)	AUC (%)	SEN (%)	ESP (%)	PRE (%)	T. Entrenamiento (s)	T. Prueba (ms)
1000	99.2733±0.0448	93.4294±1.461	99.8923±0.021	33.0222±4.1121	99.3777±0.0484	54.2016±0.255	0.3106±0.0017
2000	99.4343±0.0298	95.3177±1.3085	99.887±0.018	48.7765±3.145	99.5439±0.0304	57.7676±0.2203	0.3049±0.002
3000	99.47±0.0276	96.7604±0.9339	99.8873±0.0171	53.591±2.2202	99.5792±0.0256	61.4896±0.0902	0.3095±0.0011
4000	99.5067±0.0282	97.4207±1.0652	99.8829±0.0169	57.8771±2.9632	99.6203±0.0289	65.2038±0.1785	0.3092±0.0057
5000	99.5697±0.0274	97.5826±0.8198	99.8991±0.0149	63.7457±2.3	99.6674±0.0266	69.5872±0.1395	0.3104±0.0069
6000	99.551±0.0273	97.7327±0.6408	99.8685±0.0186	65.0151±2.8256	99.6791±0.0272	72.7818±0.2154	0.3096±0.0062

Tabla 5.5. Resultados de ELM usando ψ_{DT32} (red DT con 32 lambdas)

# de Muestras Entrenamiento	ACC (%)	AUC (%)	SEN (%)	ESP (%)	PRE (%)	T. Entrenamiento (s)	T. Prueba (ms)
1000	99.152±0.045	92.3699±1.9258	99.8916±0.0226	28.1026±3.7126	99.2566±0.0503	29.2482±0.0803	0.3179±0.0069
2000	99.3193±0.0377	93.9207±1.3613	99.8381±0.0236	46.6531±3.4627	99.4766±0.0404	31.482±0.1114	0.3117±0.0021
3000	99.3847±0.0298	96.3596±0.906	99.8306±0.021	54.6561±2.565	99.5492±0.0288	33.6512±0.079	0.31±0.0017
4000	99.401±0.0305	95.9033±1.0388	99.8555±0.0234	54.625±2.9265	99.5409±0.0334	35.369±0.1189	0.3063±0.0021
5000	99.4603±0.0296	96.5727±0.9246	99.8562±0.0258	59.7583±2.516	99.5998±0.0239	38.4716±0.1496	0.3073±0.0026
6000	99.475±0.0323	97.3362±0.8125	99.8683±0.0202	62.0582±2.7056	99.6024±0.0319	40.0748±0.1457	0.3117±0.0056

Tabla 5.6. Resultados de MEC-ELM usando ψ_{DT64} (red DT con 64 lambdas)

# de Muestras Entrenamiento	ACC (%)	AUC (%)	SEN (%)	ESP (%)	PRE (%)	T. Entrenamiento (s)	T. Prueba (ms)
1000	99.2367±0.0374	93.4255±1.5593	99.8853±0.0239	32.2824±3.8434	99.3478±0.0466	53.8826±0.1896	0.3322±0.0086
2000	99.3407±0.0298	95.9957±0.9586	99.8502±0.0216	47.8386±3.5626	99.486±0.0365	57.446±0.155	0.3147±0.0014
3000	99.368±0.0367	96.6285±0.8439	99.8667±0.0177	50.9466±3.1327	99.4969±0.039	61.6268±0.1682	0.312±0.0049
4000	99.4237±0.0332	97.7332±0.6289	99.8539±0.023	56.2201±2.8091	99.5653±0.0287	65.2646±0.2077	0.3122±0.0026
5000	99.47±0.0306	97.5694±0.7254	99.8522±0.0216	62.2686±2.5954	99.6134±0.029	69.4198±0.1557	0.3197±0.0013
6000	99.5003±0.0299	97.3473±0.6554	99.8559±0.0213	63.4873±2.3515	99.6402±0.0248	73.1528±0.2756	0.3165±0.0044

Tabla 5.7. Resultados de ELM usando ψ_{DT64} (red DT con 64 lambdas)

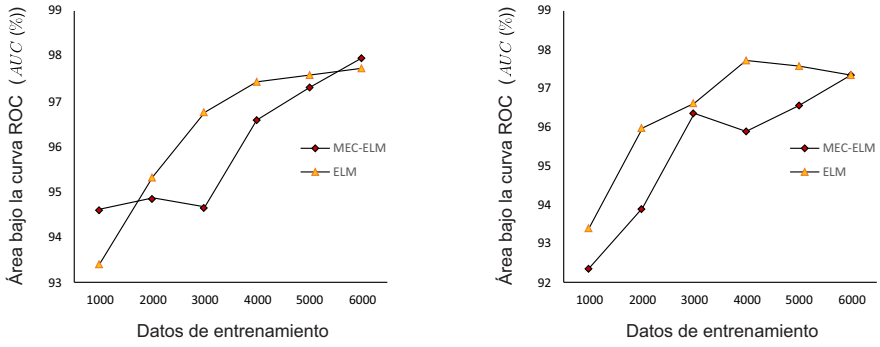
Los resultados indican que MEC-ELM logró clasificar con mejor exactitud los *ligth-paths* de la red DT con 32 lambdas y ELM algo mejor los de la red DT con 64 lambdas. Sea cual sea el caso, ambos modelos alcanzaron exactitudes relativamente altas, no obstante, es importante considerar que los conjuntos de entrenamiento utilizados estaban desbalanceados (más patrones de una clase que de otra) hecho que se puede notar en la tabla 5.2, esta característica puede hacer que la exactitud como medida de rendimiento no sea la más indicada, toda vez que un modelo únicamente podría estar aprendiendo a distinguir la clase más probable sin establecer en realidad fronteras de decisión apropiadas. La exactitud está estrechamente relacionada con la distribución de las muestras, por ejemplo, podría ocurrir que los modelos entrenados estén clasificando todas las muestras como pertenecientes a la clase A y con esto ya se estarían alcanzando exactitudes aproximadamente del 99.13 % para la red DT con 32 lambdas y del 98.89 % para la red DT con 64 lambdas (o un error del 0.87 % y 1.11 % respectivamente), sin embargo, este no es el caso, considerando que las exactitudes logradas con MEC-ELM y ELM para los problemas planteados, están por encima de dichos valores en todos los casos.

Una medida más objetiva del rendimiento de un clasificador es el área bajo la curva ROC. Se trata de una medida que evalúa las capacidades de un clasificador para distinguir ambas clases y no simplemente el porcentaje de aciertos, de modo que si un modelo estuviera clasificando todas las muestras como pertenecientes a la clase A, esto sería indicado con un rendimiento del 50 % en términos del área bajo la curva ROC. Los resultados para el caso de la red DT de 32 lambdas a medida que se incrementaron las muestras de entrenamiento indican que MEC-ELM pasó de tener una AUC del 94.61% al 97.96% en comparación con ELM que pasó de tener una AUC del 93.42 % al 97.73 %, tal y como se muestra en la figura 5.1a.

En el caso de la red DT de 64 lambdas, los resultados indican que a medida que se incrementaron los datos de entrenamiento, el AUC pasó de un 92.36% a un 97.33% con MEC-ELM en comparación con un AUC que pasó de 93.42 % a 97.34 % en el caso de ELM, esto se muestra en la figura 5.1b.

Como puede notarse, no se obtuvieron diferencias significativas de rendimiento en términos de área bajo la curva ROC entre MEC-ELM y ELM. Sin embargo, es importante destacar que el rendimiento de MEC-ELM fue menor en las pruebas con menos número de datos y fue aumentando a medida que se incrementó dicho número, hasta superar el rendimiento de ELM en el caso de la red DT con 32 lambdas o hasta casi

igualar el rendimiento de ELM en el caso de la red DT con 64 lambdas.



a) b)

Figura 5.1. Rendimiento de MEC-ELM vs el de ELM en términos de área bajo la curva ROC: a) Red DT con 32 lambdas b) Red DT con 64 lambdas

La sensibilidad indica la capacidad de un modelo para clasificar como positivas las muestras que realmente son positivas. Considerando los resultados en términos de sensibilidad, puede establecerse que ambos modelos obtuvieron iguales rendimientos a la hora de clasificar exclusivamente *lightpaths* de alta calidad. Esto ocurre en la red DT con 32 y 64 lambdas e independientemente del número de muestras de entrenamiento. Lo anterior implica, que las diferencias en el rendimiento general de ambos modelos están relacionadas con la capacidad de éstos para clasificar *lightpaths* de baja calidad, esto se puede notar en los resultados de especificidad.

La especificidad indica la capacidad de un modelo para clasificar como negativas las muestras que realmente son negativas, particularmente para los dos casos relacionados con la red DT, las muestras negativas corresponden a los *lightpaths* de baja calidad cuya proporción en los respectivos conjuntos de entrenamiento era muy reducida respecto a los *lightpaths* de alta calidad, haciendo que su identificación en la fase de prueba sea bastante complicada. En este sentido, es claro que los dos casos propuestos ponen a prueba con un alto grado de exigencia las capacidades de clasificación de ambos algoritmos.

En figura 5.2a se presenta el rendimiento en términos de especificidad de MEC-ELM y ELM a medida que se incrementaron los datos de entrenamiento para el caso de la red DT con 32 lambdas. Como puede observarse, el rendimiento de ambos modelos aumentó con el número de muestras hasta llegar a 65.86 % en el caso de MEC-ELM y 65.01 % en el caso de ELM. Si bien el rendimiento de MEC-ELM fue superior al

final, cabe mencionar que las diferencias entre ambos modelos fueron muy estrechas y, en general no se observa una clara diferencia, dado que la especificidad de ELM fue mejor con 3000 y 5000 muestras, mientras que la especificidad de MEC-ELM fue mejor con 4000 y 6000 muestras. Algo similar ocurrió con la red DT de 64 lambdas, esto se puede observar en la figura 5.2b, aunque en este caso la especificidad de ELM al final (63.49 %) fue un poco mejor a la de MEC-ELM (62.06 %).

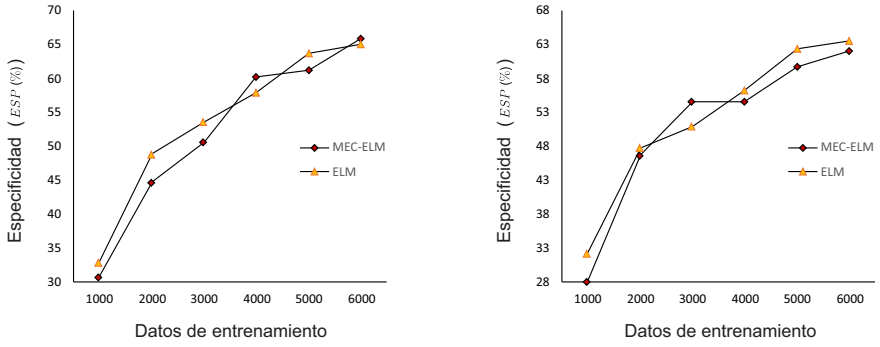


Figura 5.2. Rendimiento de MEC-ELM vs el de ELM en términos de especificidad: a) Red DT con 32 lambdas b) Red DT con 64 lambdas

Por su parte, la precisión mide el grado de exactitud en la clasificación de muestras exclusivamente de clase positiva y está dada por la relación entre el número de muestras clasificadas como positivas sobre la totalidad de muestras positivas en el conjunto de prueba. Los resultados indican altos niveles de precisión con ambos modelos y para ambos conjuntos de datos (ψ_{DT32} y ψ_{DT64}). La precisión de los distintos modelos fue mayor o igual a 99.25 % y no se encontró ninguna diferencia significativa entre MEC-ELM y ELM a medida que se incrementaron las muestras de entrenamiento.

Ahora bien, todas las medidas de rendimiento fueron tomadas utilizando conjuntos de prueba, es decir datos no utilizados en el entrenamiento, de modo que los resultados son indicadores que permiten comparar la capacidad de generalización de ambos modelos. En términos generales y considerando que no hubo resultados de rendimiento significativamente diferentes, se puede decir que ambos modelos lograron generalizar las muestras de entrenamiento en forma similar. Sin embargo, es importante destacar que MEC-ELM logró una reducción en tiempo de entrenamiento de alrededor del 45.3% en relación a ELM (los detalles al respecto pueden apreciarse claramente en la tabla 5.8). Esta ventaja representa un gran ahorro de tiempo a medida que se

Número de Muestras de Entrenamiento	Tiempo de Entrenamiento					
	DT32			DT64		
	MEC-ELM T_m (s)	ELM T_M (s)	Reducción $(1-\frac{T_m}{T_M})100\%$	MEC-ELM T_m (s)	ELM T_M (s)	Reducción $(1-\frac{T_m}{T_M})100\%$
1000	29.13	54.20	46.25 %	29.25	53.88	45.71 %
2000	31.39	57.78	45.67 %	31.48	57.45	45.20 %
3000	33.56	61.49	45.42 %	33.65	61.63	45.39 %
4000	35.75	65.20	45.16 %	35.37	65.26	45.80 %
5000	38.47	69.59	44.71 %	38.47	69.42	44.58 %
6000	40.33	72.78	44.58 %	40.07	73.15	45.22 %

Tabla 5.8. Tiempos de entrenamiento para MEC-ELM y ELM

incrementan los datos, algo que resulta crucial a la hora de abordar aplicaciones que involucran grandes conjuntos de entrenamiento.

En lo que respecta al tiempo de prueba por muestra procesada, los resultados no muestran ninguna diferencia entre ambos modelos y en la práctica no tendría por qué existir dicha diferencia, considerando que se trata de la misma arquitectura de red SLFN. Sin embargo, es preciso mencionar que la velocidad para procesar muestras en la fase de prueba es uno de los grandes fuertes que tienen las redes neuronales, esto como resultado de su inherente capacidad de procesamiento paralelo, una capacidad que se puede explotar fácilmente mediante su implementación en hardware y que hace que estos esquemas de procesamiento sean ideales para abordar aplicaciones en las que la información se presenta de forma masiva, imprecisa y distorsionada (Brio y Sanz, 2006).

5.5 Resumen de capítulo

En este capítulo ser realizó un análisis comparativo entre MEC-ELM y ELM con base en dos problemas relacionados con la identificación de enlaces ópticos en un sistema de comunicaciones. El objetivo básicamente es evaluar la alta o baja calidad de los en enlaces ópticos o "*lightpahts*", de modo que puedan ser seleccionados o descartados antes de ser establecidos. Para esto, se contó con dos conjuntos de entrenamiento (φ_{DT32}, φ_{DT64}) obtenidos al simular la red Deutsche Telekom (DT) de 14 nodos para 32 y 64 longitudes de onda. Las salidas de dichos conjuntos se codificaron de modo que la BER o tasa de errores de bit para los *lightpaths* de alta calidad sea menor que 10^{-12}, con lo que se obtuvo los conjuntos utilizados en las pruebas (ψ_{DT32} y ψ_{DT64}), los cuales tienen como característica el ser altamente desbalanceados.

Las pruebas se realizaron utilizando una implementación de MEC-ELM basada en

el método de retención, dicha implementación permite manejar el costo computacional mediante la reutilización de matrices y, es más indicada que la implementación basada en validación cruzada que se presentó en el capítulo 4, en aquellos problemas en lo que el número de datos no representa un limitante importante. El método propuesto involucra la inversión de $R+1$ matrices de tamaño $q \times q$, en comparación con la implementación análoga de ELM que involucra la inversión de R matrices de tamaño $L \times L$, donde R es el número de valores a probar de la constante de regularización C. Con base en lo anterior, en términos computacionales la utilización del método de retención resulta más adecuada en combinación con MEC-ELM, que en combinación con ELM, teniendo en cuenta que $q < L$.

Las pruebas realizadas muestran el rendimiento, el tiempo de entrenamiento y el tiempo de prueba de MEC-ELM y ELM, a medida que se incrementan los datos de entrenamiento. Para medir el rendimiento se calculó: la exactitud, el área bajo la curva ROC, la sensibilidad, la especificidad y la precisión de los modelos obtenidos, dichas medidas se tomaron utilizando conjuntos de prueba. En general, MEC-ELM fue mejor al clasificar los *lightpahts* de la red DT con 32 lambdas y ELM un poco mejor al clasificar los *lightpahts* de la red DT con 64 lambdas. Sin embargo, no se evidencian diferencias de rendimiento significativas en entre ambos algoritmos, pero claramente se puede apreciar que MEC-ELM logra reducciones en el tiempo de entrenamiento de alrededor del 45.3 % en relación a ELM, algo que resulta crucial a la hora de abordar aplicaciones que involucran masivas cantidades de datos.

Capítulo 6

Implementación de MEC-ELM usando la metodología *Map-Reduce*

"*Map-Reduce*" es un modelo de programación creado para procesar en forma paralela grandes conjuntos de entrenamiento, este modelo dado a conocer por Google en el año 2004, es una de las respuestas más reconocidas en relación a los desafíos que impone "*Big Data*" y se caracteriza por tener una gran escalabilidad, muy buena tolerancia a fallas, fácil programación y alta flexibilidad (Chen *et al.*, 2017b). *Map-Reduce* proporciona un medio para distribuir la computación sin los costos propios de la programación paralela, de modo que el desarrollador se centre únicamente en aplicar la técnica de divide y vencerás al problema planteado, sin preocuparse, por ejemplo, de asuntos relacionados con el envío y recepción de mensajes (Vidal-Silva *et al.*, 2018).

Para implementar un algoritmo usando la metodología *Map-Reduce* únicamente es necesario especificar dos bloques funcionales: mapeador y reducidor. Todo lo demás es transparente al desarrollador. El mapeador toma un conjunto de pares (*clave,valor*) y los convierten en otro conjunto intermedio de pares (*clave,valor*). Por su parte, el reducidor recibe los datos provenientes de múltiples mapeadores organizados por clave y los procesa para producir un resultado final.

Una implementación *Map-Reduce* puede correr sobre múltiples hilos en un procesador, múltiples procesadores en una máquina multiprocesador, o múltiples máquinas en un clúster o red de máquinas, siendo esta última, la manera ideal de procesar grandes cantidades de datos. Ahora bien, para desarrollar una implementación *Map-Reduce* pueden usarse varios lenguajes de programación, pero más allá de esto, existen "*frameworks*" de código abierto como Hadoop que facilitan el flujo de trabajo. El funcionamiento de Hadoop se ilustra en la figura 6.1 y se resume en cuatro pasos:

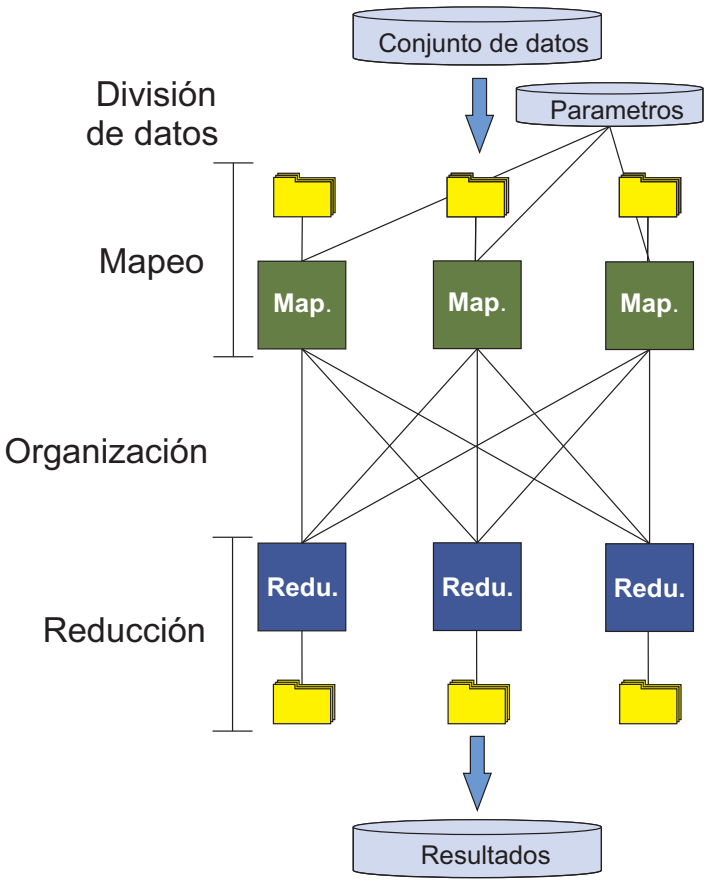


Figura 6.1. Esquema general del funcionamiento de Hadoop

- División de datos: El conjunto de datos se carga en el sistema de archivos distribuidos de Hadoop o directorio HDFS ("*Hadoop Distributed File System*"), éste divide los datos en fragmentos denominados *Splits*, los cuales son entregados a varios mapeadores en forma de pares (*clave,valor*).

- Mapeo: los *Splits* son procesados por múltiples mapeadores (corriendo en paralelo el mismo código), este proceso arroja como resultado un conjunto intermedio de pares (*clave,valor*).

- Organización: el "*framework*" toma las salidas de los mapeadores y automáticamente las organiza y agrupa por clave, generando un nuevo conjunto de pares con formato (*clave*,*lista*(*valor*)) .

- Reducción: varios reducidores corriendo en paralelo el mismo código, procesan los datos identificados con una misma clave. Al igual que los mapeadores, los

reducidores deben generar un nuevo conjunto de datos con formato (*clave,valor*). Finalmente, La salida de los reducidores se archiva en el directorio HDFS.

Considerando las ventajas que MEC-ELM tiene sobre ELM en lo relacionado con el tiempo de entrenamiento, en este capítulo se propone una implementación de MEC-ELM utilizando la metodología de *Map-Reduce*, tal y como se ha hecho con otros algoritmos de "*Machine Learning*" incluidas varias versiones de ELM (Chu *et al.*, 2007; Alham *et al.*, 2013; Xin *et al.*, 2016; Chen *et al.*, 2017b; He *et al.*, 2013), buscando abordar aplicaciones con masivas cantidades de datos de una manera distribuida y más eficiente.

6.1 Desarrollo de la implementación

Para paralelizar un algoritmo usando la metodología *Map-Reduce* es necesario identificar el formato de tres tipos de pares (*clave,valor*):

- Pares de entrada: son los pares (*clave,valor*) que entran a los mapeadores, éstos deben estar cargados inicialmente en el sistema de directorios HDFS.

- Pares intermedios: hacen referencia a los pares (*clave,valor*) que salen de los mapeadores y a los pares (*clave,valor*) que entran a los reducidores, cuyo formato es igual para un mismo trabajo *Map-Reduce*.

- Pares de salida: son los pares (*clave,valor*) que salen de los reducidores.

Con base en lo anterior, para diseñar un mapeador es necesario identificar el formato de la clave de entrada, el valor de entrada, la clave intermedia y valor intermedio. Así mismo, para diseñar un reducidor se debe identificar el formato de la clave intermedia, el valor intermedio, la clave de salida y el valor de salida.

Una vez identificados los pares (*clave,valor*) se procede a diseñar el código de los mapeadores y reducidores. A continuación, para describir el código de dichos módulos, se adopta una notación orientada a programación. Esta nueva notación resulta natural en el contexto *Map-Reduce* e incluye términos como: "*String*" (para referirse a cadenas de caracteres), arreglos e instrucciones estándar como "*for*" y "*while*". Además, para referirse a un elemento de una matriz o arreglo se utiliza el nombre del arreglo o matriz y se indica el elemento entre corchetes, por ejemplo:

- $A[j]$ indica el j-ésimo elemento del arreglo A.
- $M[i][j]$ hace referencia al elemento de la i-ésima fila y la j-ésima columna de la matriz M.

El proceso *Map-Reduce* que se propone a continuación permite calcular las matrices **W** y **Z** de MEC-ELM. Una vez descritos los módulos de este proceso, se retoma la notación matricial utilizada en los anteriores capítulos y se plantea el proceso para calcular la solución de MEC-ELM.

6.1.1 Implementación del mapeador

Inicialmente se debe cargar el conjunto de entrenamiento en el sistema de directorios HDFS como una secuencia de pares $(clave1, valor1)$, donde $clave1$ es la posición del patrón en el conjunto de entrenamiento, y $valor1$ es el contenido del patrón (incluye vector de entrada y salida deseada). Posteriormente, el "*framework*" asigna aleatoriamente los pares $(clave1, valor1)$ a los mapeadores y éstos deben procesarlos para obtener los resultados parciales de las matrices **W** y **Z**.

Con base en lo anterior, $(clave1, valor1)$ son los pares de entrada y éstos deben ser procesados por los mapeadores para obtener los resultados parciales de las matrices **W** y **Z**. Una vez calculados, los resultados parciales son enviados a los reducidores en forma de un par intermedio. En la implementación propuesta, la clave intermedia $clave2$ corresponde a un número que indica la posición de un elemento en las matrices de resultados parciales de **W** o **Z** y el valor intermedio $valor2$ es el valor del respectivo elemento.

La operación del mapeador se describe en la tabla 6.1, donde las líneas 1 y 2 permiten seleccionar los pesos de los nodos ocultos e inicializar la matriz en la que se almacenarán los resultados intermedios. En las líneas 3 a 9, se calculan los resultados parciales de **W** y **Z**. Finalmente, en las líneas 10 a 21 se generan los pares intermedios que posteriormente se envían a la salida del mapeador.

Nótese que las líneas 4 y 5 se especifican mediante dos bloques funcionales denominados $MapeoAleatorio$ y $MatAux$ respectivamente. El primero permite calcular las salidas de la capa oculta ($SalCapOc$) y el vector de salidas deseadas ($Objetivos$) en función de la clave de entrada $valor1$ y los pesos ocultos $PesosOc$. Su implementación se describe en la tabla 6.2. Por su parte, el segundo bloque funcional ($MatAux$) permite calcular la matriz $MatAuxUnPat$ que contiene los resultados parciales de **W** y **Z**

relativos a un único patrón de entrada. Esta matriz se calcula en función de las salidas de la capa oculta y las salidas deseadas. Los detalles de su implementación se presentan en la tabla 6.3.

El bloque funcional $MatAux$ se ejecuta una vez por cada par de entrada y, los resultados obtenidos se van acumulando en una matriz llamada $MatAuxParcial$ tal y como se especifica en la línea 7 de la tabla 6.1. Los elementos de esta matriz son los valores intermedios $valor2$ que se ponen como salida del mapeador y su posición (especificando la pertenencia a **W** y **Z**) son las claves intermedias $clave2$.

	Entradas:	Par $(clave1, valor1)$ # $clave1$: *String* que contiene la posición del patrón. # $valor1$: *String* que guarda el contenido de un patrón.
	Salidas:	Par $(clave2, valor2)$ # $clave2$: *String* que indica la posición de un elemento de **W** o **Z** . # $valor2$: *String* que contiene el valor de un elemento de **W** o **Z**.
1		Generar aleatoriamente los pesos y bias ocultos utilizando una semilla inicial sem y almacenarlos en una matriz llamada $PesosOc$.
2		Inicializar un arreglo numérico nulo llamado $MatAuxParcial$.
3		$C1 = "V"$ si existe un par (*clave1,valor1*) en la entrada, en caso contrario $C1 = "F"$. Entonces: *while* $(C1 == "V")$
4		$[SalCapOc, Objetivos] = MapeoAleatorio(valor1, PesosOc)$.
5		$MatAuxUnPat = MatAux(SalCapOc, Objetivos)$.
6		Sea L el número de nodos ocultos, q el número de nodos de compresión y m el número de clases. Entonces: *for* $j = 1$ *to* $Lq + Lm$
7		$MatAuxParcial[j] = MatAuxParcial[j] + MatAuxUnPat[j]$
8		*endfor*
9		*endwhile*
10		Sea L el número de nodos ocultos. Entonces: *for* $i = 1$ *to* L
11		Sea q el número de nodos de compresión. Entonces: *for* $j = 1$ *to* q
12		Concatenar "Z:", i, "," y j en un *String* llamado $valor2$.
13		$clave2$ recibe $MatAuxParcial[(i-1)q + j]$ como *String*.
14		Retornar como salida el par $(clave2, valor2)$.
15		*endfor*
16		Sea m el número de nodos de clases. Entonces: *for* $j = Lq + 1$ *to* $L(q + m)$
17		Concatenar "W:", i, "," y j en un *String* llamado $valor2$.
18		$clave2$ recibe $MatAuxParcial[(i-1)q + j]$ como *String*.
19		Retornar como salida el par $(clave2, valor2)$.
20		*endfor*
21		*endfor*

Tabla 6.1. Seudocódigo del mapeador

	Entradas:	$valor1 \rightarrow$ *String* que guarda el contenido de un patrón. $PesosOc \rightarrow$ Matriz que contiene pesos y bias ocultos.
	Salidas:	$SalCapOc \rightarrow$ Arreglo numérico que contiene el vector de salida de la capa oculta. $Objetivos \rightarrow$ Arreglo numérico que contiene el vector de salida deseada.
1		Escribir los elementos del vector de entrada en un arreglo numérico llamado $Entrada$ (el vector de entrada está contenido en $valor1$).
2		Codificar la salida deseada según la ecuación 2.23 y guardar el resultado en un arreglo numérico llamado $Objetivos$ (la salida deseada está contenida en $valor1$).
3		Normalizar los elementos del arreglo $Entrada$
4		Inicializar un arreglo numérico nulo llamado $SalCapOc$.
5		Sea L es el número de nodos ocultos y $b[i]$ el bias del i-ésimo nodo oculto: *for* $i = 1$ *to* L
6		$a = b[i]$;
7		Sea n la longitud del arreglo $Entrada$: *for* $j = 1$ *to* n
8		Sea $W[i][j]$ el peso entre la j-ésima entrada y el i-ésimo nodo oculto, calcular:
9		$a = a + W[i][j] \times Entrada[j]$
10		*endfor*
11		Sea $g(.)$ la función de activación de los nodos ocultos entonces: $SalCapOc[i] = g(a)$.
12		*endfor*
13		Retornar como salida ($SalCapOc$,$Objetivos$)

Tabla 6.2. Función $[SalCapOc, Objetivos] = MapeoAleatorio(clave1, PesosOc)$.

	Entradas:	$SalCapOc \rightarrow$ Arreglo numérico que contiene el vector de salida de la capa oculta. $Objetivos \rightarrow$ Arreglo numérico que contiene el vector de salida deseada.
	Salidas:	$MatAuxUnPat \rightarrow$ Arreglo numérico que contiene resultados intermedios de **Z** y **W** (para un único patrón de entrada).
1		Inicializar un arreglo numérico nulo llamado $ElementosZ$.
2		Inicializar un arreglo numérico nulo llamado $ElementosW$.
3		Sea L es el número de nodos ocultos: *for* $i = 1$ *to* L
4		Sea q el número de nodos de compresión: *for* $j = 1$ *to* q
5		Adjuntar ($SalCapOc[i] \times SalCapOc[j]$) a $ElementosZ$.
6		*endfor*
7		Sea m el número de clases: *for* $j = 1$ *to* m
8		Adjuntar ($Objetivos[i] \times Objetivos[j]$) a $ElementosW$.
9		*endfor*
10		*endfor*
11		Concatenar $ElementosZ$ y $ElementosW$ en un arreglo numérico llamado $MatAuxUnPat$.
12		Retornar como salida $MatAuxUnPat$.

Tabla 6.3. Función $MatAuxUnPat = MatAux(SalCapOc, Objetivos)$.

6.1.2 Implementacion del reducidor

El reducidor se encarga de organizar y sumar los resultados parciales provenientes de los mapeadores, para obtener las matrices **W** y **Z** de MEC-ELM. De modo que cada valor de salida $valor3$ es un elemento de las matrices **W** o **Z**, y la clave de salida $clave3$ es una cadena de caracteres que especifica la pertenencia de dicho elemento a la respectiva matriz, así como su posición en la misma.

El reducidor recibe los pares intermedios ordenados por clave y, a medida que éstos ingresan se van sumando los valores respectivos, acumulando el resultado en una variable entera denominada val. En el momento en que ingresa un par intermedio con una clave distinta a la actual, se escriben los pares de salida en la salida estándar y se reinicia la variable val desde cero. Posteriormente, el proceso se repite otra vez para la nueva clave. La operación de los reducidores se detalla en la tabla 6.4.

	Entradas:	Par $(clave2, valor2)$ # $clave2$: *String* aleatorio # $valor2$: *String* que contiene resultados intermedios
	Salidas:	Par $(clave3, valor3)$ # $clave3$: *String* que contiene los elementos de **W** y **Z** # $valor3$: *String* aleatorio
1		Inicializar un *String* llamado $claveActual$ con la cadena de caracteres "Ninguna".
2		Inicializar una variable entera nula llamada val.
3		$C1 = "V"$ si hay un par intermedio $(clave2, valor2)$ en la entrada, en caso contrario $C1 = "F"$. Entonces:
		while $(C1 == "V")$
4		Almacenar el contenido de $valor2$ en una variable entera llamada $valor$
5		*if* $(clave2 == claveActual)$
6		$val = val + valor$
7		*else*
8		*if* $(clave2!$ ="Ninguna")
9		$clave3 = claveActual$
10		$valor3$ recibe val como *String*
11		Retornar como salida el par $(clave3, valor3)$
12		*endif*
13		$val = valor$
14		$claveActual = clave2$
15		*endif*
16		*endwhile*
17		*if*$(clave2 == claveActual)$
18		$clave3 = claveActual$
19		$valor3$ recibe val como *String*
20		Retornar como salida el par $(clave3, valor3)$
21		*endif*

Tabla 6.4. Seudocódigo de los Reducidores.

6.1.3 Solución MEC-ELM usando *Map-Reduce*

Para calcular la solución de MEC-ELM primero se deben generar aleatoriamente los pesos y bias de las neuronas ocultas. Luego se calculan las matrices **W** y **Z** usando *Map-Reduce*. Una vez el proceso *Map-Reduce* termina, se procede a identificar las matrices $\mathbf{V}_{oc} \in \mathbb{R}^{q \times q}$ y $\mathbf{W}_{oc} \in \mathbb{R}^{q \times (L-q)}$. Finalmente, se obtiene la matriz **D** y se calcula la solución MEC-ELM. El proceso de entrenamiento se especifica en la tabla 6.5.

	Entradas:	$\eta = \{(\mathbf{x}_j, \mathbf{t}_j), \mathbf{x}_j \in \mathbb{R}^n, \mathbf{t}_j \in \mathbb{R}^m\}_{j=1}^N \rightarrow$ Conjunto de datos $L \rightarrow$ Número de nodos ocultos $q \rightarrow$ Número de nodos de compresión ($q \in \aleph$, tal que $q \leq L$) $C \rightarrow$ Constante de regularización $g(.) \rightarrow$ Función de activación de los nodos ocultos
	Salidas:	$\{\mathbf{w}_i, b_i\}_{i=1}^L \rightarrow$ Pesos y bias ocultos $\mathbf{B}_\epsilon \rightarrow$ Solución de MEC-ELM (Matriz de pesos de salida)
1		Generar aleatoriamente los pesos $\mathbf{w}_i$ y los bias b_i para $i = 1, 2, ..., L$ utilizando una semilla inicial sem.
2		Calcular las matrices auxiliares **Z** y **W** usando *Map-Reduce*.
3		Identificar las matrices $\mathbf{V}_{oc} \in \mathbb{R}^{q \times q}$ y $\mathbf{W}_{oc} \in \mathbb{R}^{q \times (L-q)}$ tales que: $$\mathbf{Z} = [\mathbf{V}_{oc} \quad \mathbf{W}_{oc}]$$
4		Calcular la matriz **D** como: $$\mathbf{D} = \left[\mathbf{I} \quad \mathbf{V}_{oc}^{-1}\mathbf{W}_{oc}\right]$$
5		Calcular los pesos de salida $\mathbf{B}_\epsilon$ como: $$\mathbf{B}_\epsilon = C\mathbf{W} - C\mathbf{D}^T \left(\frac{\mathbf{I}}{C} + \mathbf{Z}\mathbf{D}^T\right)^{-1} \mathbf{Z}\mathbf{W}$$

Tabla 6.5. Entrenamiento de un modelo MEC-ELM usando *Map-Reduce*

6.1.4 Pruebas de consistencia

La implementación de MEC-ELM presentada en la tabla 6.5, se puso a prueba con ayuda de una utilidad conocida como Hadoop Streaming, esta utilidad que viene con la distribución de Hadoop, permite crear y ejecutar trabajos *Map-Reduce* utilizando múltiples lenguajes de programación. Particularmente, la versión de Hadoop utilizada en las pruebas fue la 2.7.7 y los módulos involucrados en el proceso *Map-Reduce* (Mapeador y Reducidor) se implementaron utilizando el lenguaje de programación R.

Por otro lado, el clúster utilizado estaba conformado por 9 computadores, uno de los cuales se configuró como maestro y los 8 restantes como esclavos. Las características de los equipos utilizados se presentan a continuación:

- Maestro: procesador Intel i7 de 3.4 GHz, 16 Gb de memoria y sistema operativo Linux.

- Esclavos: procesadores AMD A10 de 2.6 GHz, 8 Gb de memoria y sistema operativo Linux.

El objetivo en estas pruebas es analizar el tiempo de entrenamiento a medida que se varía el número de computadores esclavos, el número de datos y el número de nodos ocultos. En este proceso se utilizaron dos conjuntos de datos. El primero relacionado con el problema planteado en la sección 3.4.1, que consiste en discriminar patrones pertenecientes a dos clases bidimensionales traslapadas y con distribución gaussiana (las funciones de distribución para las clases 1 y 2 están definidas en las ecuaciones 3.29 y 3.30 respectivamente). El segundo conjunto de datos se encuentra disponible en "*UCI Machine Learning Repository*" (Dua y Taniskidou, 2017) y se denomina "*image segmentation*".

Las pruebas se realizaron utilizando el 50$\%$ y el 70$\%$ de los nodos ocultos como nodos de compresión (MEC-ELM q=0.5L y MEC-ELM q=0.7L respectivamente). Adicionalmente, los resultados se comparan con los obtenidos haciendo uso de una implementación de ELM basada en *Map-Reduce* denominada A-ELM (Xin *et al.*, 2016). En todos los casos, las funciones de activación utilizadas fueron del tipo tangente hiperbólica $Tanh(a) = (1 - e^{-a})/(1 + e^{-a})$.

Variación de computadores esclavos

Inicialmente se considera la influencia de los computadores esclavos en el proceso de entrenamiento. Estas pruebas se llevaron a cabo con 40 nodos ocultos y 6×10^7 patrones de entrenamiento generados con las funciones de distribución planteadas en las ecuaciones 3.29 y 3.30. Los patrones eran tales, que el 50 $\%$ pertenecían a la clase 1 y el 50 $\%$ pertenecían a la clase 2. Los resultados obtenidos pueden observarse en la figura 6.2.

La figura 6.2 muestra como disminuye el tiempo de entrenamiento a medida que se incrementa el paralelismo. Sin embargo, esta situación tiene lugar hasta que el número

de computadores esclavos es 4, en adelante el tiempo de entrenamiento se mantiene constante. Este comportamiento resulta claro, si se tiene en cuenta que incrementar el número de computadores esclavos, hace que las tareas relacionadas con el manejo de la red y las comunicaciones se vuelvan más complejas computacionalmente. Como puede observarse, esta situación es similar para los tres modelos probados.

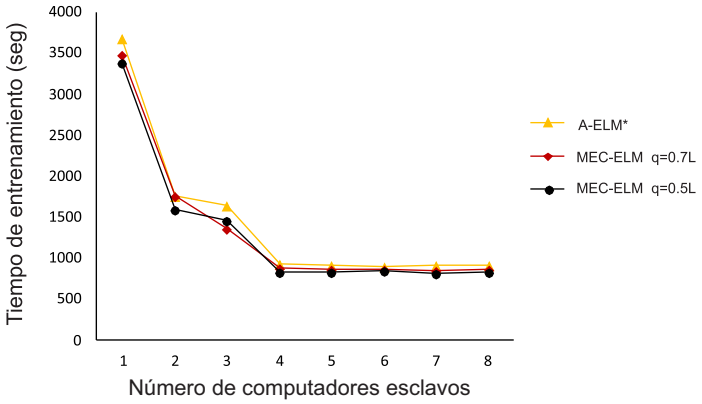


Figura 6.2. Tiempo de entrenamiento vs computadores esclavos

Variación del número de datos

Utilizando 40 nodos ocultos y la totalidad de computadores esclavos, se realizaron distintas pruebas variando el número de datos. Nuevamente, los patrones de entrenamiento utilizados se generaron con base en las funciones de distribución planteadas en las ecuaciones 3.29 y 3.30, con el 50 $\%$ de patrones perteneciendo a la clase 1 y el otro 50 $\%$ de patrones perteneciendo a la clase 2. Los resultados obtenidos se presentan en la figura 6.3.

Como puede observarse, no existe una diferencia considerable en los tiempos de entrenamiento de ELM y los dos modelos MEC-ELM para las pruebas con menor número de datos. Sin embargo, a medida que los datos se incrementan por encima de 8×10^6 puede apreciarse una reducción en el tiempo de entrenamiento de los modelos MEC-ELM respecto a ELM, siendo más pequeño el tiempo de entrenamiento de MEC-ELM cuando se utilizan el 50$\%$ de los nodos ocultos como nodos de compresión.

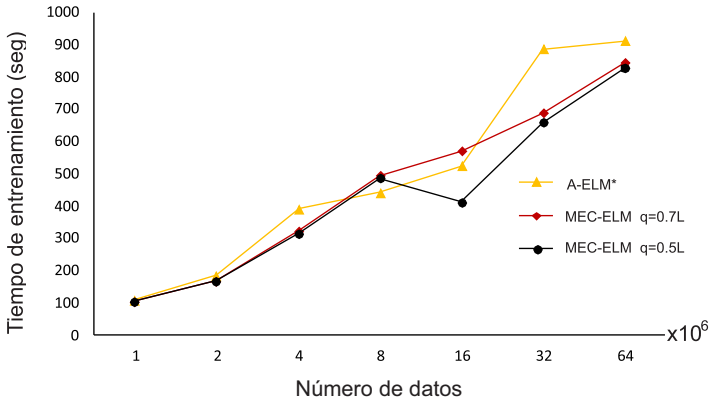


Figura 6.3. Tiempo de entrenamiento vs datos de entrenamiento

Es importante destacar que, en la práctica, el uso de *Map-Reduce* tiene sentido sólo si se analizan enormes cantidades de datos, de lo contrario el costo computacional asociado a tareas de manejo de la red y comunicaciones puede llegar a ser más complejo que el problema que se pretende resolver.

Variación del número de nodos ocultos

Por último, se realizó una serie de pruebas variando el número de nodos ocultos. Las pruebas se realizaron utilizando el conjunto de entrenamiento "*image segmentation*". Este conjunto de datos tiene las siguientes características:

- Muestras de entrenamiento: 2310
- Características de entrada: 18
- Número de clases: 7

El proceso de entrenamiento se llevó a cabo utilizando la totalidad de los computadores esclavos y replicando 1000 veces el conjunto de datos, para un total de 2.310.000 patrones de entrenamiento.

En general, MEC-ELM permite reducir el tiempo de entrenamiento en relación a ELM. Esta diferencia es más considerable a medida que se incrementan los nodos ocultos y respecto al modelo con menor número de nodos de compresión. Esto puede corroborarse en la figura 6.4.

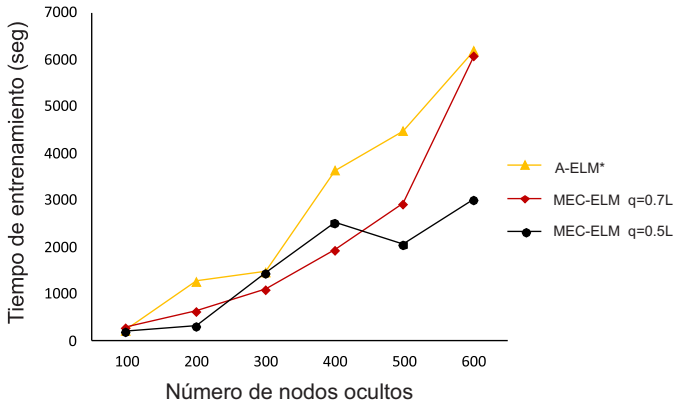


Figura 6.4. Tiempo de entrenamiento vs nodos ocultos

Los resultados obtenidos indican que la implementación de MEC-ELM basada en *Map-Reduce* no solo permite procesar grandes cantidades de datos, sino que también utiliza de forma más eficiente la infraestructura computacional disponible.

6.2 Resumen de capítulo

MEC-ELM alcanza rendimientos similares a ELM en términos de generalización reduciendo a la vez los tiempos de entrenamiento. Es importante destacar que se trata de una mejora a nivel funcional (y no a nivel de implementación), con lo que se abre la posibilidad de desarrollar nuevas y mejores variantes utilizando técnicas y metodologías que ya se han aplicado a ELM. Para muestra, en este capítulo se presentó una implementación de MEC-ELM usando la metodología de programación "*Map-Reduce*", la cual permite potenciar las capacidades de procesamiento de MEC-ELM gracias a la utilización de grandes "*clúster* de computadoras trabajando en paralelo.

Para verificar la consistencia de la implementación presentada, se utilizó un "*cluster*" conformado por 9 computadores y se realizó una serie de pruebas variando el número de computadores esclavos, el número de datos y el número de nodos ocultos. En general, MEC-ELM permite un uso más eficiente de la infraestructura computacional y logra procesar cantidades considerables de datos en tiempos de entrenamiento más cortos en comparación con A-ELM (A-ELM es una implementación de ELM usando *MapReduce*), estas ventajas se hacen más evidentes a medida que se incrementan los nodos ocultos y para las implementaciones de MEC-ELM con menos nodos de compresión.

Capítulo 7
Conclusiones y trabajos futuros

7.1 Conclusiones

MEC-ELM mejora el proceso de aprendizaje de redes neuronales SLFN en aplicaciones de clasificación que involucran grandes conjuntos de entrenamiento. Sus ventajas prácticas se ven reflejadas en altos rendimientos en términos de generalización, cortos tiempos de entrenamiento y aprovechamiento eficientemente de la infraestructura computacional disponible; tres aspectos verdaderamente cruciales de cara a resolver los modernos desafíos que impone "*Big data*".

El algoritmo MEC-ELM es robusto ante problemas de clasificación, lo cual se evidencia en su capacidad para discriminar clases bidimensionales traslapadas con distribución gaussiana, estableciendo fronteras de decisión óptimas en el sentido de Bayes y por su capacidad de aprendizaje a partir de conjuntos de entrenamiento altamente desbalanceados, logrando rendimientos similares a los de ELM en menos tiempo de entrenamiento. Adicionalmente, sus bajos requerimientos computacionales lo hacen ideal para abordar problemas en los que la información se presenta de forma masiva.

La consistencia de MEC-ELM asegura su utilización exitosa en una vasta gama de aplicaciones reales, tal y como lo confirman múltiples experimentos basados en simulación para diferentes problemas de clasificación binaria y multiclase, en los que se puede apreciar la estabilidad y gran capacidad de generalización de MEC-ELM a lo largo de todas las pruebas.

MEC-ELM permite reducir el tiempo de entrenamiento en relación a ELM, alcanzando a la vez rendimientos similares en términos de generalización. Más aún, experimentos relacionados con clasificación de enlaces ópticos indican que MEC-ELM logra reducciones alrededor del 45.3 % en relación a ELM, lo que resulta muy significativo a medida que se incrementan los datos de entrenamiento.

MEC-ELM representa una mejora funcional en relación al algoritmo ELM, lo que permite usar eficientemente la infraestructura computacional disponible, mejorar los procesos de validación cruzada y optimizar la ejecución de procesos *Map-Reduce*.

7.2 Trabajos futuros

A largo de este documento se presentó una gran cantidad de resultados a partir de los cuales se pudo establecer que MEC-ELM permite abordar con éxito aplicaciones de clasificación que involucran grandes conjuntos de entrenamiento. Por otro lado, no existe una razón evidente que impida su utilización exitosa en aplicaciones de aproximación. Sin embargo, esto no ha sido verificado experimentalmente.

Los nodos de compresión en un modelo MEC-ELM, se encargan de comprimir la información proveniente de la capa oculta, y dependiendo de su número puede reducirse la complejidad computacional en el entrenamiento. Como consecuencia surge un compromiso entre rendimiento y complejidad computacional. Dicho compromiso se podría manejar más eficientemente si el número de nodos de compresión se determina dinámicamente. Esto se puede hacer utilizando alguna técnica de tipo incremental (como QRI-ELM, I-ELM, entre otras). Sin embargo, no está claro que técnica usar y como debería incorporarse en la estructura matemática de MEC-ELM.

Recientemente se han propuesto una serie de algoritmos denominados metacognitivos, los cuales se basan en el modelo de metacognición humana propuesto por Nelson y Narens (1980). Estos algoritmos buscan mejorar el proceso de aprendizaje incorporando un componente metacognitivo que les permite determinar: qué aprender, cómo aprender y cuándo aprender. En la literatura pueden encontrarse varias iniciativas en este sentido, las cuales reportan mejoras en relación a la capacidad de generalización (Savitha *et al.*, 2014; Revanasiddappa *et al.*, 2018). Con base en lo anterior, los principios metacognitivos podrían potenciar el rendimiento de modelos MEC-ELM entrenados con un reducido número de nodos de compresión. Sin embargo, es preciso investigar como combinar eficientemente los principios metacognitivos con MEC-ELM.

Teniendo en cuenta que se han desarrollado varios trabajos en procura de hacer que ELM sea más eficiente y adecuado para aplicaciones específicas, se espera que muchas de las mejoras realizadas a este algoritmo, puedan ser extendidas a MEC-ELM. Lo anterior abre una amplia gama de posibilidades entre las que se incluyen el desarrollo de nuevos algoritmos basados en MEC-ELM y la realización de nuevos

comparativos. Para muestra, en el capítulo 6 de este documento, se propone una implementación de MEC-ELM utilizando la metodología *Map-Reduce*.

Referencias

Alham, N. K., Li, M., Liu, Y., y Qi, M. (2013). A mapreduce-based distributed svm ensemble for scalable image classification and annotation. *Computers & Mathematics with Applications*, 66(10):1920–1934.

Anam, K. y Al-Jumaily, A. (2017). Evaluation of extreme learning machine for classification of individual and combined finger movements using electromyography on amputees and non-amputees. *Neural Networks*, 85:51–68.

Aqlan, A. M., El-Wahed, W. F. A., y El-Wahed, M. A. A. (2008). Hybrid extreme learning machine with Levenberg-Marquardt algorithm using AHP method. *INFOS*, pp. 110–117.

Aras, S. y Kocakoç, İ. D. (2016). A new model selection strategy in time series forecasting with artificial neural networks: IHTS. *Neurocomputing*, 174:974–987.

Ata, R. (2015). Artificial neural networks applications in wind energy systems: a review. *Renewable and Sustainable Energy Reviews*, 49:534–562.

Azodolmolky, S., Klinkowski, M., Marin, E., Careglio, D., Pareta, J. S., y Tomkos, I. (2009a). A survey on physical layer impairments aware routing and wavelength assignment algorithms in optical networks. *Computer Networks*, 53(7):926–944.

Azodolmolky, S., Klonidis, D., Tomkos, I., Ye, Y., Saradhi, C. V., Salvadori, E., Gunkel, M., Telekom, D., Manousakis, K., y Vlachos, K. (2009b). A dynamic impairment-aware networking solution for transparent mesh optical networks. *IEEE Communications Magazine*, 47(5).

Azodolmolky, S., Perelló, J., Angelou, M., Agraz, F., Velasco, L., Spadaro, S., Pointurier, Y., Francescon, A., Saradhi, C. V., Kokkinos, P., y Otros, . (2011). Experimental demonstration of an impairment aware network planning and operation tool

for transparent/translucent optical networks. *Journal of Lightwave Technology*, 29(4):439–448.

Bartlett, P. L. (1998). The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. *IEEE transactions on Information Theory*, 44(2):525–536.

Brio, B. M. y Sanz, A. (2006). *Redes neuronales artificiales y sistemas borrosos*. Tercera Edición, México, Alfaomega Ra-Ma.

Cao, J. y Lin, Z. (2015). Extreme learning machines on high dimensional and large data applications: a survey. *Mathematical Problems in Engineering*, 2015.

Cao, J., Lin, Z., Huang, G.-B., y Liu, N. (2012). Voting based extreme learning machine. *Information Sciences*, 185(1):66–77.

Carrasco, J. J., Millán-Giraldo, M., Caravaca, J., Escandell-Montero, P., Martinez-Martinez, J. M., y Soria-Olivas, E. (2016). Elm regularized method for classification problems. *International Journal on Artificial Intelligence Tools*, 25(01):1550026.

Chandra, R., Jain, K., Deo, R. V., y Cripps, S. (2019). Langevin-gradient parallel tempering for bayesian neural learning. *Neurocomputing*.

Chang, C. C. y Lin, C. J. (2011). Libsvm: A library for support vector machines, (en linea http://www.csie.ntu.edu.tw/ cjlin/libsvm, acceso: 20 de agosto de 2018). *ACM Transactions on Intelligent Systems and Technology*.

Chen, C., Li, K., Duan, M., y Li, K. (2017a). Extreme learning machine and its applications in big data processing. En *Big Data Analytics for Sensor-Network Collected Intelligence*, pp. 117–150. Elsevier.

Chen, C., Li, K., Ouyang, A., y Li, K. (2017b). A parallel approximate ss-elm algorithm based on mapreduce for large-scale datasets. *Journal of Parallel and Distributed Computing*, 108:85–94.

Chen, Z. X., Zhu, H. Y., y Wang, Y. G. (2013). A modified extreme learning machine with sigmoidal activation functions. *Neural Computing and Applications*, 22(3-4):541–550.

Chu, C.-T., Kim, S. K., Lin, Y.-A., Yu, Y., Bradski, G., Olukotun, K., y Ng, A. Y. (2007). Map-reduce for machine learning on multicore. En *Advances in neural information processing systems*, pp. 281–288.

Dai, Q. y Liu, N. (2012). Alleviating the problem of local minima in backpropagation through competitive learning. *Neurocomputing*, 94:152–158.

Deng, W., Zheng, Q., y Chen, L. (2009). Regularized extreme learning machine. En *2009 IEEE symposium on computational intelligence and data mining*, pp. 389–395. IEEE.

Dizdarević, H., Dizdarević, S., Skrbić, M., y Hadžiahmetović, N. (2016). A survey on physical layer impairments aware routing and wavelength assignment algorithms in transparent wavelength routed optical networks. En *Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2016 39th International Convention on*, pp. 530–536. IEEE.

Dua, D. y Taniskidou, K. (2017). Uci machine learning repository, (en linea: http://archive.ics.uci.edu/ml, acceso: 20 de agosto de 2018). *Irvine, CA: University of California, School of Information and Computer Science*.

Gil, E. (2016). Big data, privacidad y protección de datos. Technical report, Agencia Española de Protección de Datos.

Golub, G. H. y Loan, C. (2013). Matrix computations, forth edition.

Golub, G. H. y Loan, C. F. (1996). *Matrix computations*, volumen 3. Baltimore, MD: The Johns Hopkins Univ. Press.

Gómez, J. (2017). Application of machine learning techniques to optical communication systems and networks. *Universidad de Valladolid*.

Grigorievskiy, A., Miche, Y., Käpylä, M., y Lendasse, A. (2016). Singular value decomposition update and its application to (Inc)-OP-ELM. *Neurocomputing*, 174:99–108.

Hastie, T., Tibshirani, R., y Friedman, J. (2009). The elements of statistical learning: prediction, inference and data mining. *Springer-Verlag, New York*.

He, Q., Shang, T., Zhuang, F., y Shi, Z. (2013). Parallel extreme learning machine for regression based on mapreduce. *Neurocomputing*, 102:52–58.

Heeswijk, M., Miche, Y., Oja, E., y Lendasse, A. (2011). GPU-accelerated and parallelized ELM ensembles for large-scale regression. *Neurocomputing*, 74(16):2430–2437.

Huang, G., Huang, G.-B., Song, S., y You, K. (2015a). Trends in extreme learning machines: A review. *Neural Networks*, 61:32–48.

Huang, G.-B. (2003). Learning capability and storage capacity of two-hidden-layer feedforward networks. *IEEE Transactions on Neural Networks*, 14(2):274–281.

Huang, G.-B. (2014). An insight into extreme learning machines: random neurons, random features and kernels. *Cognitive Computation*, 6(3):376–390.

Huang, G.-B. y Babri, H. A. (1998). Upper bounds on the number of hidden neurons in feedforward networks with arbitrary bounded nonlinear activation functions. *IEEE Transactions on Neural Networks*, 9(1):224–229.

Huang, G.-B. y Chen, L. (2007). Convex incremental extreme learning machine. *Neurocomputing*, 70(16):3056–3062.

Huang, G.-B. y Chen, L. (2008). Enhanced random search based incremental extreme learning machine. *Neurocomputing*, 71(16):3460–3468.

Huang, G.-B., Chen, Y.-Q., y Babri, H. A. (2000). Classification ability of single hidden layer feedforward neural networks. *IEEE Transactions on Neural Networks*, 11(3):799–801.

Huang, G.-B., Ding, X., y Zhou, H. (2010). Optimization method based extreme learning machine for classification. *Neurocomputing*, 74(1):155–163.

Huang, G.-B., Lia, M.-B., Chenb, L., y Siewa, C.-K. (2008). Incremental extreme learning machine with fully complex hidden nodes. *Neurocomputing*, 71:576–583.

Huang, G.-B., Zhou, H., Ding, X., y Zhang, R. (2012). Extreme learning machine for regression and multiclass classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(2):513–529.

Huang, G.-B., Zhu, Q.-Y., y Siew, C.-K. (2006). Extreme learning machine: theory and applications. *Neurocomputing*, 70(1):489–501.

Huang, H.-X., Li, J.-C., y Xiao, C.-L. (2015b). A proposed iteration optimization approach integrating backpropagation neural network with genetic algorithm. *Expert Systems with Applications*, 42(1):146–155.

Huang, S., Wang, B., Qiu, J., Yao, J., Wang, G., y Yu, G. (2016). Parallel ensemble of online sequential extreme learning machine based on MapReduce. *Neurocomputing*, 174:352–367.

Inaba, F. K., Salles, E. O. T., Perron, S., y Caporossi, G. (2018). Dgr-elm–distributed generalized regularized elm for classification. *Neurocomputing*, 275:1522–1530.

Jaddi, N. S., Abdullah, S., y Hamdan, A. R. (2015). Optimization of neural network model using modified bat-inspired algorithm. *Applied Soft Computing*, 37:71–86.

Jiménez, T., Aguado, J. C., de Miguel, I., Durán, R. J., Angelou, M., Merayo, N., Fernández, P., Lorenzo, R. M., Tomkos, I., y Abril, E. J. (2013). A cognitive quality of transmission estimator for core optical networks. *Journal of Lightwave Technology*, 31(6):942–951.

Kamal, S., Ripon, S. H., Dey, N., Ashour, A. S., y Santhi, V. (2016). A MapReduce approach to diminish imbalance parameters for big deoxyribonucleic acid dataset. *Computer methods and programs in biomedicine*, 131:191–206.

Kokkinos, Y. y Margaritis, K. G. (2018). Managing the computational cost of model selection and cross-validation in extreme learning machines via cholesky, svd, qr and eigen decompositions. *Neurocomputing*, 295:29–45.

Krawczyk, B. (2016). GPU-accelerated extreme learning machines for imbalanced data streams with concept drift. *Procedia Computer Science*, 80:1692–1701.

Lämmel, R. (2008). Google's mapreduce programming model-revisited. *Science of computer programming*, 70(1):1–30.

Leshno, M., Lin, V. Y., Pinkus, A., y Schocken, S. (1993). Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867.

Liang, N.-Y., Huang, G.-B., Saratchandran, P., y Sundararajan, N. (2006). A fast and accurate online sequential learning algorithm for feedforward networks. *IEEE Transactions on Neural networks*, 17(6):1411–1423.

Lima, A. R., Cannon, A. J., y Hsieh, W. W. (2015). Nonlinear regression in environmental sciences using extreme learning machines: a comparative evaluation. *Environmental Modelling & Software*, 73:175–188.

Lin, J., Yin, J., Cai, Z., Liu, Q., Li, K., y Leung, V. (2013). A secure and practical mechanism of outsourcing extreme learning machine in cloud computing. *IEEE Intelligent Systems*, 28(6):35–38.

Liu, N. y Wang, H. (2010). Ensemble based extreme learning machine. *IEEE Signal Processing Letters*, 17(8):754–757.

Luo, J., Vong, C.-M., y Wong, P.-K. (2014). Sparse bayesian extreme learning machine for multi-classification. *IEEE Transactions on Neural Networks and Learning Systems*, 25(4):836–843.

Man, Z., Lee, K., Wang, D., Cao, Z., y Khoo, S. (2012). Robust single-hidden layer feedforward network-based pattern classifier. *IEEE transactions on neural networks and learning systems*, 23(12):1974–1986.

Man, Z., Lee, K., Wang, D., Cao, Z., y Miao, C. (2011). A new robust training algorithm for a class of single-hidden layer feedforward neural networks. *Neurocomputing*, 74(16):2491–2501.

Miche, Y., Sorjamaa, A., Bas, P., Simula, O., Jutten, C., y Lendasse, A. (2010). Op-elm: optimally pruned extreme learning machine. *IEEE Transactions on Neural Networks*, 21(1):158–162.

Mohapatra, P., Chakravarty, S., y Dash, P. (2015). An improved cuckoo search based extreme learning machine for medical data classification. *Swarm and Evolutionary Computation*, 24:25–49.

Mukherjee, I. y 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*, 39(3):2397–2407.

Nawi, N. M., Khan, A., y Rehman, M. (2013). A new Levenberg Marquardt based back propagation algorithm trained with cuckoo search. *Procedia Technology*, 11:18–23.

Nelson, T. y Narens, L. (1980). Metacognition: Core readings, to nelson (ed.) ed.

Rady, H. A. K. (2011). Shannon entropy and mean square errors for speeding the convergence of multilayer neural networks: a comparative approach. *Egyptian Informatics Journal*, 12(3):197–209.

Revanasiddappa, M., Harish, B., y Kumar, S. A. (2018). Meta-cognitive neural network based sequential learning framework for text categorization. *Procedia computer science*, 132:1503–1511.

Savitha, R., Suresh, S., y Kim, H. J. (2014). A meta-cognitive learning algorithm for an extreme learning machine classifier. *Cognitive Computation*, 6(2):253–263.

Simeonidou, D., Nejabati, R., Qin, Y., Cheng, K., Triay Marquès, J., Escalona, E., Zervas, G. S., Zarris, G., Amaya González, N., y Cervelló Pastor, C. (2010). Demonstration of c/s based hardware accelerated qot estimation tool in dynamic impairment-aware optical network. En *36th European Conference and Exhibition on Optical Communication*, pp. 1–3. IEEE Press. Institute of Electrical and Electronics Engineers.

Sivarajah, U., Kamal, M. M., Irani, Z., y Weerakkody, V. (2017). Critical analysis of big data challenges and analytical methods. *Journal of Business Research*, 70:263–286.

Suárez, E. J. C. (2016). Tutorial sobre máquinas de vectores soporte (svm). Technical report, Dpto. de Inteligencia Artificial, ETS de Ingeniería Informática, Universidad Nacional de Educación a Distancia (UNED), Madrid.

Sun, Y., Yuan, Y., y Wang, G. (2011). An os-elm based distributed ensemble classification framework in p2p networks. *Neurocomputing*, 74(16):2438–2443.

Tamura, S. y Tateishi, M. (1997). Capabilities of a four-layered feedforward neural network: four layers versus three. *IEEE Transactions on Neural Networks*, 8(2):251–255.

Ticknor, J. L. (2013). A bayesian regularized artificial neural network for stock market forecasting. *Expert Systems with Applications*, 40(14):5501–5506.

Torrecilla, J. L. y Romo, J. (2018). Data learning from big data. *Statistics & Probability Letters*.

Vidal-Silva, C. L., Bustamante, M. A., Lapo, M. d. C., y Núñez, M. d. l. Á. (2018). En la búsqueda de soluciones mapreduce modulares para el trabajo con bigdata: Hadoop orientado a aspectos. *Información tecnológica*, 29(2):133–140.

Wang, H., Xu, Z., Fujita, H., y Liu, S. (2016). Towards felicitous decision making: an overview on challenges and trends of big data. *Information Sciences*, 367:747–765.

Wang, S., Jiang, Y., Chung, F.-L., y Qian, P. (2015). Feedforward kernel neural networks, generalized least learning machine, and its deep learning with application to image classification. *Applied Soft Computing*, 37:125–141.

Wang, Y., Cao, F., y Yuan, Y. (2011). A study on effectiveness of extreme learning machine. *Neurocomputing*, 74(16):2483–2490.

Webb, A. R. (2003). *Statistical pattern recognition*. John Wiley & Sons.

Wilamowski, B. M. y Yu, H. (2010). Neural network learning without backpropagation. *IEEE Transactions on Neural Networks*, 21(11):1793–1803.

Xin, J., Wang, Z., Chen, C., Ding, L., Wang, G., y Zhao, Y. (2014). Elm$*$: distributed extreme learning machine with mapreduce. *World Wide Web*, 17(5):1189–1204.

Xin, J., Wang, Z., Qu, L., Yu, G., y Kang, Y. (2016). A-elm$*$: Adaptive distributed extreme learning machine with mapreduce. *Neurocomputing*, 174:368–374.

Ye, Y. y Qin, Y. (2015). QR factorization based incremental extreme learning machine with growth of hidden nodes. *Pattern Recognition Letters*, 65:177–183.

Ye, Y., Squartini, S., y Piazza, F. (2013). Online sequential extreme learning machine in nonstationary environments. *Neurocomputing*, 116:94–101.

Yu, D. y Deng, L. (2012). Efficient and effective algorithms for training single-hidden-layer neural networks. *Pattern Recognition Letters*, 33(5):554–558.

Yu, H. y Wilamowski, B. M. (2011). *Industrial electronics handbook*, volumen 5, capítulo Levenberg–marquardt training, p. 1. CRC Press.

Zhang, J., Ji, N., Liu, J., Pan, J., y Meng, D. (2015). Enhancing performance of the backpropagation algorithm via sparse response regularization. *Neurocomputing*, 153:20–40.

Zhao, J., Wang, Z., y Park, D. S. (2012). Online sequential extreme learning machine with forgetting mechanism. *Neurocomputing*, 87:79–89.

Zong, W., Huang, G.-B., y Chen, Y. (2013). Weighted extreme learning machine for imbalance learning. *Neurocomputing*, 101:229–242.

Anexo A
Proceso de Entrenamiento en OS-ELM

Dado un conjunto de entrenamiento $\eta = \{(\mathbf{x}_j, \mathbf{t}_j), \mathbf{x}_j \in \mathbb{R}^n, \mathbf{t}_j \in \mathbb{R}^m\}$ y una red SLFN estándar de L nodos ocultos con funciones de activación $g(.)$

Paso 1. **Fase de inicialización:** seleccionar un bloque inicial de muestras de entrenamiento $\eta_0 = \{(\mathbf{x}_j, \mathbf{t}_j)\}_{j=1}^{N_0}$ a partir de un conjunto de entrenamiento $\eta = \{(\mathbf{x}_j, \mathbf{t}_j) | \mathbf{x}_j \in \mathbb{R}^n, \mathbf{t}_j \in \mathbb{R}^m, j = 1, 2, ..., N\}$, con $L \leq N_0 \leq N$.

a. Asignar aleatoriamente los pesos $\mathbf{w}_i$ y los bias b_i para $i = 1, 2, ..., L$.

b. Calcular la matriz de salida de la capa oculta inicial $\mathbf{H}_0$

$$\mathbf{H}_0 = \begin{bmatrix} g(\mathbf{w}_1 \cdot \mathbf{x}_1 + b_1) & \cdots & g(\mathbf{w}_L \cdot \mathbf{x}_1 + b_L) \\ \vdots & \cdots & \vdots \\ g(\mathbf{w}_1 \cdot \mathbf{x}_{N_0} + b_1) & \cdots & g(\mathbf{w}_L \cdot \mathbf{x}_{N_0} + b_L) \end{bmatrix}_{N_0 \times L} . \tag{A.1}$$

c. Determinar los pesos de salida iniciales $\mathbf{B}^{(0)}$ como $\mathbf{B}^{(0)} = \mathbf{P}_0 \mathbf{H}_0^T \mathbf{T}_0$ donde $\mathbf{P}_0 = (\mathbf{H}_0^T \mathbf{H}_0)^{-1}$ y $\mathbf{T}_0 = [\mathbf{t}_1, \mathbf{t}_2, ..., \mathbf{t}_{N_0}]^T$.

d. Iniciar contador k en ceros ($k = 0$).

Paso 2. **Fase de aprendizaje:** presentar el $(k+1)$-ésimo bloque de muestras

$$\eta_{k+1} = \{(\mathbf{x}_j, \mathbf{t}_j)\}_{j=(\sum_{l=0}^{k} N_l)+1}^{\sum_{l=0}^{k+1} N_l} \tag{A.2}$$

donde N_{k+1} es el número de muestras que tiene el bloque $(k+1)$.

a. Calcular la matriz de salida de la capa oculta parcial $\mathbf{H}_{k+1}$ para el $(k+1)$-ésimo bloque de muestras η_{k+1} como

$$\mathbf{H}_{k+1} = \begin{bmatrix} g(\mathbf{w}_1 \cdot \mathbf{x}_{(\sum_{l=0}^{k} N_l)+1} + b_1) & \cdots & g(\mathbf{w}_L \cdot \mathbf{x}_{(\sum_{l=0}^{k} N_l)+1} + b_L) \\ \vdots & \cdots & \vdots \\ g(\mathbf{w}_1 \cdot \mathbf{x}_{\sum_{l=0}^{k+1} N_l} + b_1) & \cdots & g(\mathbf{w}_L \cdot \mathbf{x}_{\sum_{l=0}^{k+1} N_l} + b_L) \end{bmatrix}_{N_{(k+1)} \times L}. \tag{A.3}$$

b. Calcular los pesos de salida $\mathbf{B}^{(k+1)}$

$$\begin{aligned} \mathbf{P}_{k+1} &= \mathbf{P}_k - \mathbf{P}_k \mathbf{H}_{k+1}^T (\mathbf{I} + \mathbf{H}_{k+1} \mathbf{P}_k \mathbf{H}_{k+1}^T)^{-1} \mathbf{H}_{k+1} \mathbf{P}_k \\ \mathbf{B}^{(k+1)} &= \mathbf{B}^{(k)} + \mathbf{P}_{k+1} \mathbf{H}_{k+1}^T (\mathbf{T}_{k+1} - \mathbf{H}_{k+1} \mathbf{B}^{(k)}) \end{aligned} \tag{A.4}$$

donde $\mathbf{T}_{k+1} = [\mathbf{t}_{(\sum_{l=0}^{k} N_l)+1}, ..., \mathbf{t}_{\sum_{l=0}^{k+1} N_l}]^T$ y $\mathbf{P}_k$ una matriz auxiliar ($\mathbf{P}_0$ es calculado en el Paso 1c.).

c. Incrementar el contador k en uno ($k = k + 1$). Volver al Paso 2.

El número N_{k+1} de muestras de entrenamiento del $(k+1)$-ésimo bloque no debe ser necesariamente igual a N_k. En otras palabras, los bloques no necesariamente deben tener el mismo tamaño. En el caso especial de que las muestras sean recibidas una a una, esto es $N_{k+1} \equiv 1$, la ecuación A.4 se reduce a

$$\begin{aligned} \mathbf{P}_{k+1} &= \mathbf{P}_k - \frac{\mathbf{P}_k \mathbf{h}_{k+1} \mathbf{h}_{k+1}^T \mathbf{P}_k}{\mathbf{I} + \mathbf{h}_{k+1}^T \mathbf{P}_k \mathbf{h}_{k+1}} \\ \mathbf{B}^{(k+1)} &= \mathbf{B}^{(k)} + \mathbf{P}_{k+1} \mathbf{h}_{k+1} (\mathbf{T}_{k+1} - \mathbf{h}_{k+1}^T \mathbf{B}^{(k)}), \end{aligned} \tag{A.5}$$

donde $\mathbf{h}_{k+1} = [g(\mathbf{w}_1 \cdot \mathbf{x}_{k+1} + b_1)...g(\mathbf{w}_L \cdot \mathbf{x}_{k+1} + b_L)]^T$. Cabe destacar que la ecuación A.5 tiene la misma forma de la fórmula de *Sherman-Morrison* (Golub y Loan, 1996).

Para que OS-ELM sea más robusto puede utilizarse el método SVD para calcular $\mathbf{B}^{(k)}$ y $\mathbf{P}_k$ para $(k = 1, 2, ...)$. Esto es por si dado el caso $\mathbf{H}_0$ o $(\mathbf{I} + \mathbf{H}_{k+1} \mathbf{P}_k \mathbf{H}_{k+1}^T)$ sean singulares o tiendan a ser singulares.

Finalmente, nótese que si $N_0 = N$ el algoritmo OS-ELM se convierte en ELM. Así entonces, ELM puede ser considerado como un caso especial de OS-ELM (Liang *et al.*, 2006).

Anexo B

Entrenamiento de clasificadores ELM usando validación cruzada

B.1 Método de retención

Dado un conjunto de entrenamiento $\eta = \{(\mathbf{x}_j, \mathbf{t}_j), \mathbf{x}_j \in \mathbb{R}^n, \mathbf{t}_j \in \mathbb{R}^m\}$ y una red SLFN estándar de L nodos ocultos con funciones de activación $g(.)$:

1. Asignar aleatoriamente los pesos $\mathbf{w}_i$ y los bias b_i para $i = 1, 2, ..., L$.

2. Calcular la matriz **H** e identificar la matriz **T** (con las ecuaciones 2.5 y 2.7 respectivamente). Seguidamente identificar las matrices $\mathbf{H}_E$ y $\mathbf{H}_V$, tales que

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}_E^T & \mathbf{H}_V^T \end{bmatrix}^T \tag{B.1}$$

   y las matrices $\mathbf{T}_E$ y $\mathbf{T}_V$, tales que

$$\mathbf{T} = \begin{bmatrix} \mathbf{T}_E^T & \mathbf{T}_V^T \end{bmatrix}^T \tag{B.2}$$

3. Calcular las matrices $\mathbf{H}_E^T\mathbf{H}_E$ y $\mathbf{H}_E^T\mathbf{T}_E$.

4. Para cada C_a tal que $a = 1, 2, .., R$, siendo C_a uno de los R valores de C:

   4.1 Calcular la solución $\mathbf{B}^{(a)}$ como

$$\mathbf{B}^{(a)} = (\frac{\mathbf{I}}{C_a} + \mathbf{H}_E^T\mathbf{H}_E)^{-1}\mathbf{H}_E^T\mathbf{T}_E. \tag{B.3}$$

   4.2 Determinar la exactitud ACC_a (u otra medida de rendimiento) de la solución $\mathbf{B}^{(a)}$ configurando el modelo como clasificador ELM y usando $\mathbf{H}_V$ para validar.

5. Seleccionar la solución final **B** tal que

$$\mathbf{B} = \mathbf{B}^{(\mu)} \tag{B.4}$$

donde μ es el índice del modelo que obtuvo el mayor rendimiento. Esto es

$$\mu = \underset{a \in \{1,2,...,R\}}{\arg\max}\ ACC_a. \tag{B.5}$$

B.2 Validación cruzada con K iteraciones

Dado un conjunto de entrenamiento $\eta = \{(\mathbf{x}_j, \mathbf{t}_j), \mathbf{x}_j \in \mathbb{R}^n, \mathbf{t}_j \in \mathbb{R}^m\}$ y una red SLFN estándar de L nodos ocultos con funciones de activación $g(.)$:

1. Asignar aleatoriamente los pesos $\mathbf{w}_i$ y los bias b_i para $i = 1, 2, ..., L$.

2. Calcular la matriz **H** e identificar la matriz **T** (con las ecuaciones 2.5 y 2.7 respectivamente). Seguidamente crear k particiones de H, tales que

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}_1^T & \mathbf{H}_2^T & ... & \mathbf{H}_K^T \end{bmatrix}^T \tag{B.6}$$

y las correspondientes particiones de **T**, tales que

$$\mathbf{T} = \begin{bmatrix} \mathbf{T}_1^T & \mathbf{T}_2^T & ... & \mathbf{T}_K^T \end{bmatrix}^T \tag{B.7}$$

3. Calcular $\mathbf{\Phi} = \sum_{r=1}^{K} \mathbf{\Phi}_r$ y $\mathbf{\Upsilon} = \sum_{r=1}^{K} \mathbf{\Upsilon}_r$ siendo $\mathbf{\Phi}_r = \mathbf{H}_r^T \mathbf{H}_r$ y $\mathbf{\Upsilon}_r = \mathbf{H}_r^T \mathbf{T}_r$ para $r = 1, 2, ..., K$.

4. Para cada C_a tal que $a = 1, 2, .., R$, siendo C_a uno de los R valores de C:

   4.1 Para cada r tal que $r = 1, 2, .., K$:

      4.1.1 Calcular la solución $\mathbf{B}^{(a,r)}$ como

$$\mathbf{B}^{(a,r)} = (\frac{\mathbf{I}}{C_a} + \mathbf{\Phi} - \mathbf{\Phi}_r)^{-1}(\mathbf{\Upsilon} - \mathbf{\Upsilon}_r). \tag{B.8}$$

      4.1.2 Determinar la exactitud $ACC_{a,r}$ (u otra medida de rendimiento) de la solución $\mathbf{B}^{(a,r)}$ configurando el modelo como clasificador ELM y usando la partición $\mathbf{H}_r$ para validar.

4.1.3 Actualizar el acumulador de rendimiento REN_a

$$REN_a = REN_a + ACC_{a,r} \tag{B.9}$$

5. Calcular la solución final **B** como

$$\mathbf{B} = (\frac{\mathbf{I}}{C_\mu} + \mathbf{\Phi})^{-1}\mathbf{\Upsilon} \tag{B.10}$$

donde C_μ es la constante de regularización con la que se obtuvo el mayor rendimiento acumulado. Esto es $C_\mu \in \{C_1, C_2, ..., C_R\}$, tal que el índice μ esta dado por

$$\mu = \underset{a \in \{1,2,...,R\}}{\arg\max}\ REN_a. \tag{B.11}$$

Anexo C

Reconocimiento Estadístico de Patrones

C.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

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

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, 2003).

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

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

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

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

Por tanto el criterio de decisión anterior (ecuación C.2) puede reescribirse como sigue.

Se asigna **x** a la clase ω_j sí.

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

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

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

$$P(error) = \sum_{i=1}^{C} P(error|\omega_i)P(\omega_i) \tag{C.5}$$

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.

$$P(error|\omega_i) = \int_{C[\Omega_i]} P(\mathbf{x}|\omega_i)d\mathbf{x} \tag{C.6}$$

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 C.6 se puede reescribir la ecuación C.5 como

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

$$= \sum_{i=1}^{C} P(\omega_i)(1 - \int_{\Omega_i} P(\mathbf{x}|\omega_i)d\mathbf{x}) \tag{C.7}$$

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

La ecuación C.7 significa que minimizar la probabilidad de cometer un error de clasificación es equivalente a maximizar

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

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 C.8 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

$$P_c = \int max_i P(\omega_i)P(\mathbf{x}|\omega_i)d\mathbf{x}, \tag{C.9}$$

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

$$e_B = 1 - (\int max_i P(\omega_i)P(\mathbf{x}|\omega_i)d\mathbf{x}) \tag{C.10}$$