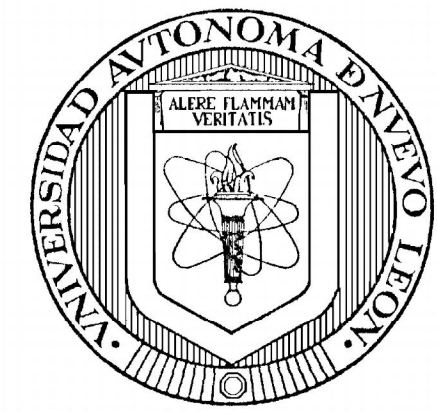
Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

Subdirección de Postgrado



Redes neuronales embebidas con aprendizaje en línea

por

Ing. Luis Lauro González Estrada

en opción al grado de

Maestría en Ciencias

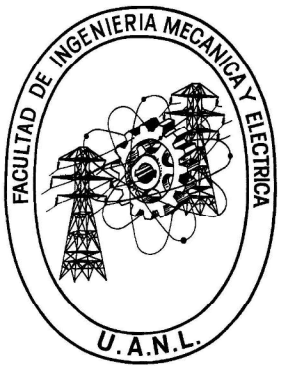
de la Ingeniería Eléctrica

San Nicolás de los Garza, Nuevo León Julio 2013

Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

Subdirección de Postgrado



Redes neuronales embebidas con aprendizaje en línea

por

Ing. Luis Lauro González Estrada

en opción al grado de

Maestría en Ciencias

de la Ingeniería Eléctrica

San Nicolás de los Garza, Nuevo León Julio 2013

Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

Subdirección de Postgrado

Los miembros del Comité de Tesis recomendamos que la Tesis «Redes neuronales embebidas con aprendizaje en línea», realizada por el alumno Ing. Luis Lauro González Estrada, con número de matrícula 1337784, sea aceptada para su defensa como opción al grado de Maestría en Ciencias en Ingeniería Eléctrica.

El Comité de Tesis

Dr. Luis Martín Torres Treviño
Asesor

Dra. Griselda Quiroz Compeán
Revisor

Dr. Ismael López Juárez
Revisor

Vo. Bo.

Dr. Moisés Hinojosa Rivera
Subdirección de Postgrado

San Nicolás de los Garza, Nuevo León, Julio 2013

Agradecimientos

A mi director de tesis, el Dr. Luis Martín Torres Treviño, por su apoyo y recomendaciones durante este proyecto. Especialmente en los momentos clave y más importantes de la tesis.

A la Dra. Griselda Quiróz Compéan y el Dr. Ismael López Juárez por sus revisiones y comentarios.

Al Dr. Miguel Mata Pérez por su apoyo y guía en el uso de LaTeX para escribir este documento.

Al Instituto Potosino de Investigación Científica y Tecnológica (IPICYT) y la Dra. Griselda Quiróz Compéan por los datos del sensor de glucosa utilizados en esta tesis, mediante el sistema CGM (*Guardian® REAL-time Continuous Glucose Monitoring Systems* por MiniMed Inc.).

A la FIME, la UANL y el CONACYT por la beca otorgada y proporcionar los medios para desarrollar los estudios de Maestría.

A mis Padres, Ing. Luis Lauro González Guerra y Sra. Laura Estrada de González, por sus enseñanzas de vida y apoyo incondicional.

A los Padres de mi esposa, Ing. José Guadalupe Cantú Ruiz y Sra. Diana Elizabeth González Cantú, quienes han sido segundos padres y segundo hogar.

A mi esposa, Mónica Lizeth Alejandra Cantú González, por su amor, tiempo, comprensión y soportar los sacrificios implicados en este trabajo.

Con mi amor para ti, Mónica.

Índice general

Índice de figuras

Índice de tablas

Nomenclaturas

API	Application Programming Interface
ASIC	Application-Apecific Integrated Circuit
B	Byte
$\mathbf{C_{RN\text{-}MET}}$	Complejidad computacional del método «MET» de la red neuronal «RN»
CAD	Convertidor Análogo-Digital
DSP	Digital Signal Processor
E/S	Entrada/Salida
ECM	Error Cuadrático Medio
EDI	Entorno de Desarrollo Integrado
ELM	Extreme Learning Machine
fa	Función de Activación
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
I	Método I (inicialización) de la red neuronal
IA	Inteligencia Artificial
Icmd	Comando para ejecutar el método I
KB	Kilobyte
KHz	Kilohertz
L	Método L (aprendizaje) de la red neuronal
Lcmd	Comando para ejecutar el método L
$\boldsymbol{M}$	Número de entradas de la red neuronal
MCU	Microcontrolador
$\mathbf{MEM_{RN}}$	Consumo de memoria de la red neuronal «RN»

MHz	Megahertz
MPU	Microprocesador
N	Número de neuronas en la capa oculta
O2LAB	Online Learning Laboratory (laboratorio desarrollado en esta tesis)
OSELM	Online Sequential Extreme Learning Machine
P	Número de salidas de la red neuronal
PISVD	Pseudo-Inversa de una matriz mediante el cálculo SVD
PM	Perceptrón Multicapa
PWM	Pulse Width Modulation
R	Método R (funcionamiento) de la red neuronal
RAM	Random Access Memory
RAN	Resource Allocating Network
RBF	Radial Basis Function
Rcmd	Comando para ejecutar el método R
RISC	Reduced Instruction Set Computer
RLS	Recursive Least Squares
RN	Red Neuronal artificial
RNAUC	Red Neuronal hacia Adelante de Una Capa oculta
RNAUC-AL	Red Neuronal hacia Adelante de Una Capa oculta de Aprendizaje en Línea
RNMS	Red Neuronal de Máxima Sensibilidad
ROM	Read Only Memory
S	Estado de la red neuronal
SBC	Single Board Computer
SE	Sistema Embebido
SI	Sistema Inteligente
SRAM	Static Random Access Memory
SVD	Singular Value Decomposition
T1DM	Diabetes mellitus tipo 1
UART	Universal Asynchronous Receiver-Transmitter
UP	Unidad de Procesamiento

Resumen

Ing. Luis Lauro González Estrada.
Candidato para el grado de Maestro en Ciencias en Ingeniería Eléctrica.

Universidad Autónoma de Nuevo León.
Facultad de Ingeniería Mecánica y Eléctrica.

Título del estudio:

Redes neuronales embebidas con aprendizaje en línea

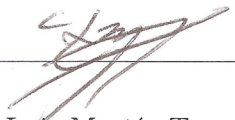
Número de páginas: **??**LastPage.

Resumen

Existen aplicaciones industriales, robóticas, biomédicas y domóticas que pueden ser diseñadas mediante ejemplos de cómo deben funcionar. Las redes neuronales, que están dentro del área de sistemas inteligentes, tienen capacidad de aprendizaje y pueden aproximar el modelo de sistemas complejos con determinada precisión, cambiando el paradigma de programación por el de aprendizaje.

Una dificultad importante en estos sistemas es la complejidad de su algoritmo de aprendizaje, que puede ser tal que requiera mucho tiempo de procesamiento siendo imposible de implementar en sistemas embebidos por restricciones de hardware.

En esta tesis se estudian arquitecturas de redes neuronales que puedan implementarse sobre sistemas embebidos de gama ligera y con aprendizaje en línea para permitir una interacción activa de la red neuronal con su entorno. Se presentan simulaciones, experimentos y comparaciones de las arquitecturas revisadas utilizando un laboratorio desarrollado en este proyecto de investigación.

Firma del asesor: 

Dr. Luis Martín Torres Treviño

Capítulo 1

Introducción

1.1 Motivación

Existen importantes aplicaciones industriales, robóticas, biomédicas y domóticas que pueden ser diseñadas mediante imitiación o ejemplos de su correcta operación o funcionamiento. Aplicaciones industriales y robóticas incluyen sistemas de reconocimiento de patrones y control de procesos. Las aplicaciones biomédicas contienen sistemas de prótesis, exoesqueletos y dispositivos biomédicos invasivos y no invasivos. En domótica[1] encontramos controles de temperatura, luz, humedad y sistemas de toma de decisiones, entre otros.

En estas aplicaciones se tiene en común el esfuerzo matemático y técnico por encontrar una solución y muchas veces una solución exacta no es posible. Así, el desarrollo de sistemas con cierto grado de complejidad puede ser abordado por los Sistemas Inteligentes (SI) los cuales están basados en técnicas de la Inteligencia Artificial como los Sistemas Difusos, Cómputo Evolutivo y Redes Neuronales Artificiales (por simplicidad, en este trabajo se hará referencia a ellas como RN).

Dentro de estas ramas, las RN son un buen enfoque de solución por su naturaleza conexionista de múltiples nodos no-lineales conocidos como neuronas artificiales. Los beneficios de las RN son: naturaleza no-lineal, capacidad de hacer un mapeo entrada-salida, adaptabilidad, generalización y tolerancia a ruido [1].

[1] La domótica es un conjunto de sistemas para automatizar viviendas y edificios.

Además, las computadoras y modernos sistemas digitales son actualmente utilizados como fuente principal de procesamiento en tales aplicaciones. Esto permite a los sistemas la capacidad de reprogramación, velocidad, comunicación con otros dispositivos e incluso abaratamiento y rápida implementación cuando el hardware es relativamente simple. Gran parte de estos sistemas son los llamados Sistemas Embebidos (SE) ya que son programados para ejecutar un número reducido de tareas [2].

En Ingeniería son deseadas las RN capaces de ser implementadas en sistemas embebidos con aprendizaje en línea para permitir una interacción activa entre la RN y su entorno. En RN se conoce al algoritmo de aprendizaje en línea como uno que permite aprender nuevo conocimiento a partir de ejemplos en cualquier momento. Es decir, dicho algoritmo no requiere que se presente un conjunto de ejemplos una sola y única vez, sino que en cualquier instante un ejemplo puede ser ingresado al algoritmo; de esta manera se permite dicha interacción activa. Además, el término «en línea» indica la capacidad de ejecutar los algoritmos de aprendizaje y funcionamiento de una RN dentro de una restricción de tiempo, la cual suele ser relativamente rápida, muchas veces en el orden de los milisegundos.

1.2 Antecedentes

Los trabajos recientes y aplicaciones comerciales que involucran RN emplean software primordialmente en lugar de algún tipo de SE. Por ejemplo, en [3], utilizan simulaciones en computadora convencional para probar el desempeño de un sistema domótico. En este sistema hacen uso de sistemas difusos y RN para el control de humedad y temperatura en una habitación.

También existen implementaciones en SE, pero de mayor escala, como son los FPGA[2]. En estos sistemas se implementa una arquitectura de RN sin su algoritmo de aprendizaje como en [4, 5, 6]. Es decir, se realiza el aprendizaje de la RN en una

[2]FPGA: Field Programmable Gate Array

computadora convencional. Luego se programa al chip[3] FPGA con la arquitectura de RN y su conocimiento embebido. Esto impide que la aplicación posea la capacidad de adaptación debido a que el algoritmo de aprendizaje de la RN no está embebido en el chip.

Este tipo de implementación, sin aprendizaje en el SE, también se realiza en arquitecturas más simples, como los microcontroladores de 8 bits, los cuales son de interés principal de esta tesis. Por ejemplo, en [7] se describe la implementación de una RN en un microcontrolador de 8 bits de la marca Microchip® utilizando lenguaje ensamblador. En [8] se utiliza la implementación antes comentada en una aplicación de linealización de las características no-lineales en sensores.

Sin embargo, también se realizan implementaciones con aprendizaje en el mismo sistema pero requieren de hardware más poderoso, fuera del interés de este trabajo. Por ejemplo, en [9] se implementa reconocimiento y síntesis emocional del habla utilizando sistemas difusos y RN ambos programados en una plataforma de desarrollo basada en un microprocesador de 32 bits ARM Cortex-M3. Así también, en [10] se implementa una RN con su algoritmo de aprendizaje utilizando un microprocesador de 32 bits ARM9. Ambos ejemplos descritos utilizan arquitecturas mucho más sofisticadas que un sencillo microcontrolador de 8 bits, incluso, suelen ser arquitecturas más poderosas que algunas permiten instalar Sistemas Operativos, ejecutar programas escritos en Java o ejecutar aplicaciones tipo DSP[4].

1.3 Planteamiento del problema

Dado que para muchas aplicaciones es deseado mantener simples/mínimas las características de los SE, en este trabajo se utilizan SE de gama ligera, como microcontroladores RISC[5] de 8 bits, en los que se ejecutan programas de manera secuencial, instrucción por instrucción. El utilizar este tipo de SE trae ventajas como

[3]Chip: otro nombre con que es conocido a los circuitos integrados.
[4]DSP: Digital Signal Processor
[5]RISC: Reduced Instruction Set Computer

bajo consumo de potencia, tamaño reducido, bajo costo y rápida implementación. Sin embargo, las características de precisión y velocidad de los algoritmos se ven comprometidas. Por lo tanto, el obstáculo más importante de este enfoque de solución es la complejidad del algoritmo de aprendizaje de la RN, que puede ser tal que requiera mucho tiempo de procesamiento y recursos computacionales que impida su implementación en SE por las restricciones de hardware.

Resumiendo: la motivación es el uso de algoritmos de aprendizaje en línea por sus aplicaciones. Se aplica la restricción de SE ligeros por sus ventajas. Así se opta por un enfoque de RN que son aptas para el aprendizaje en línea. Sin embargo, como son afectadas por la restricción de SE ligeros, el problema que se aborda es el estudio y medición de desempeño de RN con aprendizaje en línea en SE de gama ligera.

1.4 Hipótesis

Se puede establecer una red neuronal con velocidad de ejecución suficientemente rápida para permitir aprendizaje en línea, complejidad suficientemente baja para ser programadas en sistemas embebidos y precisión suficiente para ser utilizada en aplicaciones de complejidad media.

1.5 Objetivos

1.5.1 Objetivo general

Implementación de una red neuronal de aprendizaje en línea en un sistema embebido de gama ligera.

1.5.2 Objetivos particulares

Los objetivos particulares de la tesis se enlistan a continuación:

1. Desarrollo de una plataforma en base a sistemas embebidos para el desarrollo y experimentación con redes neuronales de aprendizaje en línea.

2. Establecer arquitecturas en base a redes neuronales para el aprendizaje en línea.

3. Implementación y validación de las arquitecturas para el desarrollo de máquinas por imitación.

1.6 Metodología

La metodología que se sigue para el estudio es:

1. Establecimiento del estado del arte mediante revisión bibliográfica.

2. Análisis de complejidad computacional. Para cada algoritmo de cada red neuronal, se obtiene una función que dé una aproximación de la cantidad de operaciones que debe ejecutar el sistema embebido. Analizando principalmente operaciones matemáticas con ciclos inherentes en el peor caso de ejecución.

3. Análisis de consumo de memoria. Se hace un conteo de las celdas de memoria que requiere cada variable involucrada en una red neuronal.

4. Simulaciones y experimentos. Utilizando la plataforma de sistemas embebidos Arduino® se llevan a cabo pruebas justas para valorar cada red neuronal. Las pruebas se hacen para casos de estudio en identificación de sistemas y predicción de series de tiempo. Además de un caso de aplicación en suplemento ante falla de sensores.

1.7 Contribuciones

Las contribuciones de esta tesis son:

1. Establecimiento de un estado del arte, conceptos y definiciones relacionados con el aprendizaje en línea mediante arquitecturas de redes neuronales y sus aplicaciones.

2. Los resultados de análisis de los algoritmos de redes neuronales elegidas y su implementación.

3. Desarrollo del laboratorio O2LAB para ejecutar simulaciones y experimentos.

4. Conceptualización de la aplicación ante falla de sensores y la experimentación para un sensor de glucosa en particular.

5. Publicación de artículos en revista.

1.8 Organización del trabajo

Este trabajo se organiza como sigue:

En el capítulo 2 se dan los preliminares introduciendo a la vez algunas ideas y conceptos relacionados con este trabajo. Se describe qué son los Sistemas Inteligentes, las Redes Neuronales, sus elementos y conceptos relacionados. También se describen los Sistemas Embebidos, sus componentes y programación. Finalmente se redacta sobre algoritmos para poder abordar el análisis de RN.

En el capítulo 3 se exponen tres arquitecturas de RN elegidas para ser estudiadas. En cada una se presenta una descripción, sus algoritmos y ecuaciones. A la vez se presentan los resultados del análisis de complejidad computacional y consumo de memoria. El capítulo finaliza resumiendo lo analizado en forma de tabla comparitiva teórica y unos comentarios.

En el capítulo 4 se describen los tres casos de estudio utilizados en las pruebas de simulación y experimentación, así como el caso de aplicación utilizado en experimentación. En este capítulo también se incluye la metodología que se sigue en las pruebas de los cuatro casos.

El capítulo 5 contiene los resultados de las pruebas ejecutadas.

Finalmente, el capítulo 6 presenta las conclusiones de este trabajo y recomendaciones del trabajo abierto relacionado con este proyecto de investigación.

Capítulo 2

Preliminares

Este capítulo inicia definiendo qué es un Sistema Inteligente en la sección 2.1 y una Red Neuronal Artificial en 2.2. Dentro de esta última sección, se inicia presentando un modelo clásico de neurona artificial y una arquitectura clásica de RN en la subsección 2.2.1 para dar nociones de cómo están formadas las arquitecturas que son estudiadas en este trabajo. De dicha subsección se destaca que la declaración de las ecuaciones de las funciones de activación son las que se emplean en el resto del documento. Una vez dadas las nociones, se presenta la totalidad de las bases preliminares en la subsección 2.2.2 declaradas de forma tal que ayuda a explicar las arquitecturas de RN analizadas en el capítulo 3. La subsección 2.2.3 termina la sección relacionada con RN comentando lo que es el aprendizaje en línea. Para acabar el capítulo, las secciones 2.3 y 2.4 presentan preliminares relacionados con Sistemas Embebidos y análisis de algoritmos, necesarios para llevar a cabo los análisis del capítulo 3.

2.1 Sistemas inteligentes (SI)

La Inteligencia Artificial (IA) tiende a centrarse en habilidades del tipo humanas, como la toma de decisiones y solución de problemas. Uno de sus objetivos principales es el desarrollo de máquinas que realicen tareas que hasta el momento son desempeñadas mejor por los seres humanos.

Se ha tenido mucho éxito en el modelado de la inteligencia natural biológica resultando en los llamados **Sistemas Inteligentes (SI)**. Estos deben poseer alguna

forma de aprendizaje para entender el entorno y la manera de interactuar con el mismo. El aprendizaje implica, precisamente, cambios que se adaptan en el sentido de que le permiten al dispositivo llevar a cabo la misma tarea o tareas a partir de las mismas condiciones de un modo más eficiente cada vez [11, 12].

La clave del comportamiento inteligente es poseer mucho conocimiento e incorporarlo a una computadora puede ser una tarea asombrosa. Para evitarlo, se podría dejar que las máquinas adquieran el conocimiento de manera independiente, al igual que el hombre. Las redes neuronales artificiales han tenido éxito en esta tarea.

2.2 Redes neuronales artificiales (RN)

Una **Red Neuronal Artificial (RN)** es una interconexión de pequeños procesadores no-lineales, cada uno construido por una **neurona artificial**. Estas neuronas artificiales tienen propensión natural a guardar conocimiento experimental y hacerlo disponible para su uso. Entre los beneficios de las RN están: naturaleza no-lineal, capacidad de hacer un mapeo entrada-salida, adaptabilidad, generalización (o interpolación) y tolerancia a ruido. Su desventaja principal es que son como «cajas negras», ya que forman en su interior su propia representación del modelo que aproximan sin indicar los detalles físicos o matemáticos del mismo [1].

2.2.1 Modelo clásico

La neurona artificial, o solo neurona en este trabajo, modela de forma reducida o limitada a una neurona natural biológica. La Figura 2.1 muestra la adopción clásica de una neurona. Aquí cada elemento W_m, llamado peso sináptico, hace una ponderación de su respectiva señal de entrada x_m. Dicha entrada puede provenir del entorno o de las salidas de otras neuronas previas. Luego, estas señales ponderadas son integradas mediante una sumatoria $\sum$ dando un valor escalar que pasa por la función de activación fa$(\cdot)$ que regula su intensidad. Finalmente, la señal regulada puede ser conectada a otras neuronas mediante pesos sinápticos o ser salida de la RN.

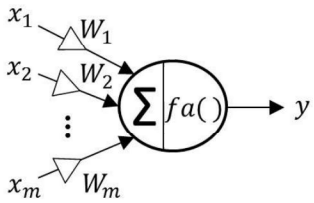


Figura 2.1: Modelo clásico de una neurona artificial.

La ecuación de salida de una neurona con este modelo en particular es (2.1).

$$y = \text{fa}\left(\sum_{m=1}^{M} x_m * W_m\right) \tag{2.1}$$

La función de activación fa$(\cdot)$ es quien proporciona la naturaleza no-lineal a la RN. Las funciones de activación más comunes son la sigmoide (2.2), tangencial hiperbólica (2.3) y gaussiana (2.4) cuyas formas se ven en la Figura 2.2. Como puede verse en la ecuación (2.4) la función de activación gaussiana requiere dos argumentos. El resultado del procesamiento (salida) de una neurona la llamamos «cómputo de neurona», «cómputo neuronal» o «cómputo neural».

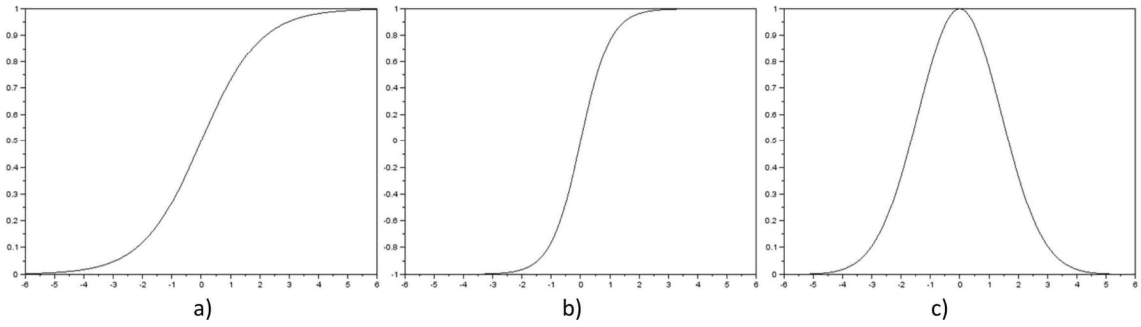


Figura 2.2: Forma de las funciones de activación: a) Sigmoide, b) Tangencial hiperbólica, c) Gaussiana.

$$\text{fa}_\text{s}(a) = \frac{1}{1 + exp(-a)} \tag{2.2}$$

$$\text{fa}_\text{t}(a) = \tanh(a) \tag{2.3}$$

$$\mathrm{fa}_{\mathrm{g}}(a, l) = exp(-(a/l)^2) \tag{2.4}$$

Una construcción clásica de RN utilizando neuronas de este tipo es el **Perceptrón Multicapa (PM)**. Esta construcción o arquitectura es del tipo hacia adelante (*feedforward*, en inglés) porque las señales solo viajan en un sentido (entrada hacia salida), es decir, no existen retroalimentaciones presentes. La Figura 2.3 muestra su representación. Cada **capa** corresponde a una «columna» de neuronas. La primer capa, o «capa de entrada», usualmente está formada por sencillas neuronas de distribución. Dicho tipo de neuronas se encarga de distribuir la señal de entrada que recibe, a las neuronas que tiene conectadas en su salida sin ejecutar procesamiento alguno. La última capa, o «capa de salida», proporciona las salidas de la RN. El resto de las capas, las intermedias, reciben el nombre de «capas ocultas» y cada una puede tener una cantidad arbitraria de neuronas.

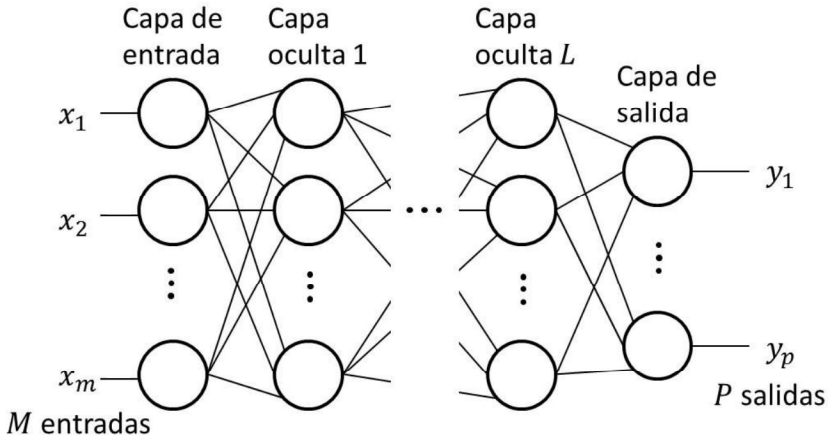


Figura 2.3: Perceptrón multicapa.

En el PM suele añadirse una entrada adicional, llamada bias, a cada neurona. El bias hace que la salida de la neurona sea diferente de cero al tener cero en todas las entradas cuando se utiliza la función de activación sigmoide o tangencial hiperbólica.

En este caso, los valores de pesos sinápticos y bias tienen «codificado» el conocimiento de la RN y son quienes deben ser ajustados durante el aprendizaje para que la RN ejecute un mapeo adecuado según la aplicación. El algoritmo de aprendizaje

más ampliamente utilizado para el PM es el de Retro-Propagación de Error (RPE o *EBP* en inglés). Puede verse que a mayor cantidad de capas existirán mayor cantidad de elementos que ajustar durante el aprendizaje. Aquí solo cabe mencionar que el RPE, a pesar de su gran uso, es un algoritmo muy lento y no es el de mejor desempeño. Es inviable para aplicaciones en línea y de sistemas embebidos.

2.2.2 Red neuronal hacia adelante de una capa oculta (RNAUC)

Una de las arquitecturas de RN más ampliamente utilizada es la **Red Neuronal hacia Adelante de Una Capa oculta (RNAUC)** representada por la Figura 2.4. Está formada por la capa de entrada, una capa oculta y la capa de salida. Dado que el resto del documento trabaja con arquitecturas tipo RNAUC, se les hace referencia de cualquiera de las dos maneras, RNAUC o simplemente RN.

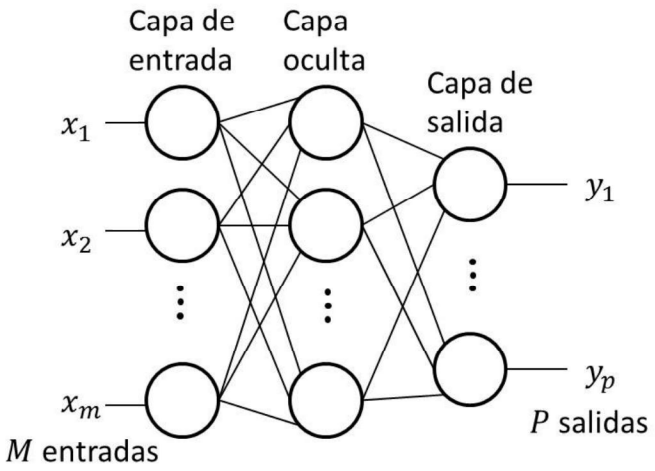


Figura 2.4: Red Neuronal hacia Adelante de Una Capa oculta (RNAUC).

Descripción de una RNAUC y algunas notas.

Los elementos presentes en una RNAUC son:

- $x \in \mathbb{R}^M$ es el vector de entradas, también conocido como **patrón**, donde M es la cantidad de entradas de la RN.
- $y \in \mathbb{R}^P$ es el vector de salidas, donde P es la cantidad de salidas de la RN.
- $N \in \mathbb{Z}^+$ es la cantidad de neuronas en la capa oculta (un número entero positivo).
- $W \in \mathbb{R}^{N \times M}$ es la matriz de pesos sinápticos de entrada. Cada elemento $W_{n,m}$ conecta la m-ésima entrada a la n-ésima neurona.
- $Q \in \mathbb{R}^{N \times P}$ es la matriz de pesos de salida. Cada elemento $Q_{n,p}$ conecta el n-ésimo cómputo neuronal con la p-ésima salida.
- $h \in \mathbb{R}^N$ es el vector de salidas de la capa oculta. Cada elemento h_n contiene el resultado del n-ésimo cómputo neuronal.
- $b \in \mathbb{R}^N$ es el vector de bias, utilizado en neuronas tipo aditiva-sigmoide. Cada elemento b_n es utilizado en la n-ésima neurona.
- $\lambda \in \mathbb{R}^N$ es el vector de anchos de campana, utilizado en neuronas tipo RBF. Cada elemento λ_n es utilizado en la n-ésima neurona.

Para tener acceso a un elemento de un vector o matriz, con objetivo de leer o escribir su contenido, utilizamos el subíndice n, m y p para elementos relacionados con neuronas, entradas y salidas respectivamente. Ejemplos: 1) El elemento $W_{n,m}$ corresponde al peso sináptico que conecta a la m-ésima entrada con la n-ésima neurona. 2) El elemento y_p corresponde a la p-ésima salida de la RN. En este documento, cualquier otro subíndice (excepto i) en cualquier variable (escalar, vector o matriz) será propio del nombre de la misma. Por ejemplo, más adelante es descrito el vector de salidas deseadas y_d en donde la «d» es parte del nombre de la variable y no

representa un acceso a algún elemento. Por el contrario, y_{d_p} significa que estamos accesando al elemento p-ésimo del vector de salidas deseadas y_d. Además, denotamos un barrido sobre todas las neuronas, entradas y salidas mediante $\forall n$, $\forall m$ y $\forall p$ respectivamente. Finalmente, para tener acceso a todos los elementos de una fila o una columna de una matriz se emplea el símbolo «:» (dos puntos). Ejemplos: 1) La expresión $A_{i,:} = v^T$ guarda el vector v en la i-ésima fila de A. 2) La expresión $B_{:,i} = v$ guarda el vector v en la i-ésima columna de B.

Ecuaciones genéricas en las capas de una RNAUC.

Capa de entrada.

La capa de entrada corresponde a neuronas de distribución.

Capa oculta.

En la capa oculta normalmente se utilizan neuronas tipo **aditiva-sigmoide** o **RBF** (*radial basis function*, en inglés), cuyos modelos son (2.5) y (2.6) respectivamente.

$$h_n(x, W, b) = \mathrm{fa}_s\left(b_n + \sum_{m=1}^{M} W_{n,m} * x_m\right), \quad \forall n \tag{2.5}$$

$$h_n(x, W, \lambda) = \mathrm{fa}_g\left(\sqrt{\sum_{m=1}^{M} (W_{n,m} - x_m)^2}, \lambda_n\right), \quad \forall n \tag{2.6}$$

En estas ecuaciones podemos ver las llamadas a las funciones de activación sigmoide $\mathrm{fa}_s()$ y gaussiana $\mathrm{fa}_g()$ expresadas en la subsección 2.2.1 como (2.2) y (2.4) respectivamente.

Capa de salida.

La capa de salida suele ponderar e integrar los cómputos neuronales (con o sin bias y sin una función de activación) como se muestra en la ecuación (2.7). Aquí, *param* representa a b o λ según el tipo de neurona con que se esté trabajando.

$$y_p = \sum_{n=1}^{N} Q_{n,p} * h_n\left(x, W, param\right), \quad \forall p \tag{2.7}$$

Particularmente, esta ecuación que genera las salidas de la RN, puede ser diferente en otras arquitecturas propuestas en la literatura. Por esto, en el capítulo 3 se presentan las ecuaciones específicas de cada arquitectura revisada.

Estado y métodos de una RNAUC.

Para una mejor visualización e implementación, se propone establecer a S como el **estado** de una RN compuesto por sus variables internas. El «esqueleto» de un estado S de una RN es declarado en (2.8).

$$S_{\text{RN}} = \{W, Q, ...\} \tag{2.8}$$

La composición de S depende de cada arquitectura, aunque generalmente se tienen los pesos sinápticos W y los pesos de salida Q. Luego, los tres puntos (...) denotan más variables que deben escribirse dentro de los corchetes {} para una arquitectura en particular.

Por otro lado, dentro de este enfoque de sistemas embebidos, se propone agrupar todos los algoritmos involucrados en una RN dentro de tres procedimientos, **métodos** o funciones:

- **Método Run (R)**. Método de funcionamiento, en español. Contiene las ecuaciones mencionadas anteriormente, es decir, calcula las salidas de la RN para un patrón dado. Se hace referencia a este método como R. Así, la ecuación (2.9) exhibe a la RN como un mapeo entrada-salida.

- **Método Learn (L)**. Método de aprendizaje, en español. Contiene un algoritmo específico, o serie de reglas, que sintoniza o ajusta una o más variables en S con fin de que la RN trabaje como es deseado a través de R. Se hace referencia a este método como L.

- **Método Initialization (I)**. Método de inicialización, en español. Una RN no puede ser utilizada si nada ha sido dado para aprender, así que este método

contiene cualquier cosa que sea necesario hacer antes de utilizar (por primera vez) al método L y, obviamente, antes de utilizar el método R. Se hace referencia a este método como I.

$$y = R(x, S) \tag{2.9}$$

2.2.3 Aprendizaje en línea

Como se comentó en el capítulo 1, el **aprendizaje en línea** permite una interacción activa entre la RN y su entorno gracias a que es posible aprender nuevo conocimiento a partir de ejemplos en cualquier momento. Es decir, no se requiere que se presente un conjunto de ejemplos una sola y única vez, sino que en cualquier instante un ejemplo puede ser ingresado al algoritmo. Esto es una especie de forma de apredizaje «asíncrona» o «atemporal» si se quiere ver así, ya que el instante en que el método L es ejecutado no depende explícitamente de la RN. Se comentó también que el término «en línea» indica la capacidad de ejecutar los algoritmos de aprendizaje y funcionamiento de una RN, es decir, los métodos L y R, dentro de una restricción de tiempo, la cual puede estar en el orden de los milisegundos.

La presentación de ejemplos en el aprendizaje de una RN es llamado «aprendizaje supervisado» dentro del ámbito de RN. Sin embargo, dado nuestro enfoque de estudio, éste término es absorbido y se hace referencia al conjunto {aprendizaje supervisado + ejecución rápida + ejecución asíncrona} como «aprendizaje en línea». Esta arquitectura con este modo de aprendizaje es llamado **Red Neuronal hacia Adelante con Una Capa oculta de Aprendizaje en Línea (RNAUC-AL)**.

Un ejemplo dado para aprender consiste en un par $\{x, y_d\}$ obtenido del sistema desconocido que se desea aprender. En este par, $y_d \in \mathbb{R}^P$ es el vector de salidas deseadas que serán aproximadas por la RN, el cual es el vector de salidas con que el sistema desconocido responde cuando se le es presentado un patrón x. La Figura 2.5 ilustra a un sistema desconocido conectado a una RNAUC-AL donde las conexiones o terminales libres en la parte inferior, etiquetadas como *Icmd*, *Rcmd* y *Lcmd* ordenan

la ejecución del método I, R y L respectivamente. Patrón tras patrón, el aprendizaje es llevado a cabo según las necesidades de una aplicación particular, eventualmente la RN será capaz de emular al sistema desconocido con cierta precisión. Así, el método L antes mencionado, es utilizado como se muestra en la ecuación (2.10) con fin de ajustar o sintonizar la RN.

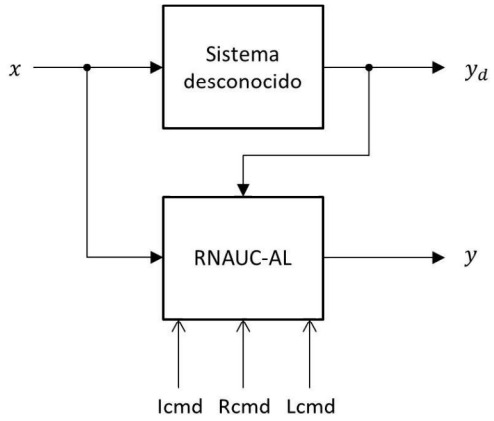


Figura 2.5: Esquema de aprendizaje en línea para una RNAUC (RNAUC-AL).

$$S \leftarrow L(x, y_d, S) \tag{2.10}$$

Aunque S está presente como salida y argumento de entrada en la expresión (2.10), no necesariamente todas las variables en S son modificadas, esto es según cada arquitectura en especial. Aquí cabe hacer ver que el símbolo $\leftarrow$ o $\rightarrow$ es utilizado en lugar del símbolo $=$ cuando una o más variables aparecen en ambos lados de una ecuación o expresión.

Apartir de aquí se hace referencia a una «RNAUC-AL» como «RNAUC» o simplemente «RN», de manera indistinta y por simplicidad. Los términos «RNAUC-AL» o «RNAUC» tratan de reservarse para esquemas, diagramas, figuras y/o tablas, mientras que el término «RN» es más utilizado en el texto por practicidad.

Por otra parte, cuando una RN «vive» en un entorno (está embebida), es necesario saber cuando ejecutar los métodos I, R y L. Mediante las terminales *Icmd*,

Rcmd y *Lcmd* se representa genéricamente el comando de los métodos de la RN. Cuando una terminal recibe una señal, el método respectivo es ejecutado. En una aplicación de hardware esto puede verse materializado como pulsadores mecánicos para comandar los métodos de la RN de forma manual. También se puede ver materializado como señales de reloj externas que comandan los métodos de la RN de forma automática e incluso a una determinada frecuencia. El modo de empleo de estas terminales depende de la aplicación en particular.

Este trabajo presenta los métodos I, R y L de cada arquitectura. Sin embargo, los análisis del capítulo 3 están enfocados al estudio de los métodos R y L. El método I no es de mucha importancia ya que es ejecutado solo al inicio o arranque del sistema embebido. Por lo tanto, se permite cierta complejidad, en teoría, para el método I. El tiempo de cómputo consumido en la ejecución del método I puede ser tomado como tiempo de *start-up* (arranque) del sistema embebido, aunque se desea un tiempo rápido de *start-up*. Entonces, el estudio del método I no es objetivo de este trabajo por ser relativamente inofensivo en este sentido. De hecho, será notado que la importancia del método I más bien está en dotar a la RN de un mecanismo de olvido o borrado del conocimiento adquirido.

2.3 Sistemas embebidos (SE)

2.3.1 Definición

Los **Sistemas Embebidos (SE)** son sistemas digitales programados para ejecutar (procesar) una cantidad reducida de tareas [2, 13]. La composición de un SE depende de los requerimientos de la aplicación pero es común encontrar: una Unidad de Procesamiento (UP, o *PU* en inglés) con algún tipo de memoria interna o externa, periféricos de E/S (entrada/salida) y posiblemente algunos otros chips. Pueden ser muy pequeños en tamaño y arquitectura siendo baratos para producción de gran escala. Normalmente su «tamaño global» impacta en factores como:

- Consumo de potencia y costo.- Ambos factores son directamente proporcionales al número de componentes y están relacionados con el tipo de componentes. Ejemplos: 1) un chip de comunicación inalámbrica puede agotar la batería en aplicaciones muy pequeñas, 2) un procesador muy sofisticado puede resultar muy costoso aún para grandes volumenes de compra.

- Complejidad de programación.- Las habilidades y herramientas de programación, tiempo de desarrollo y acceso a recursos requeridos varían de acuerdo a la arquitectura de UP: microcontroladores, sistemas multi-núcleo, DSPs, FPGAs, SBCs (todos ellos descritos en la siguiente sección). A mayor sofisticación en la UP, mayor es la complejidad de programación.

2.3.2 Arquitecturas para sistemas embebidos

Comparado con una computadora, un SE presenta mucho menos poder de cómputo y memoria. De cualquier modo, los SE resuelven tareas particulares con ventajas como bajo costo y diseño más simple. El diseño más simple se ve reflejado tanto en el hardware como en el software. A continuación se describe brevemente las arquitecturas de SE.

Microcontrolador (MCU).

Son manufacturados con memoria interna para almacenar el programa (típicamente FLASH) y una memoria de datos (típicamente SRAM[1]) para realizar operaciones. Además se les incorporan periféricos, todo en el mismo chip. Normalmente se les clasifica de acuerdo al ancho en bits de su memoria de datos. Las arquitecturas más comunes son de 8 y 16 bits. Desde los 8 bits ya se tiene un buen desempeño para muchas aplicaciones y 16 bits suelen requerirse en aplicaciones más demandantes de poder computacional, aunque con ello se lleva mayor costo. El tamaño de la memoria FLASH puede estar entre las pocas decenas de kilobytes (16, 32, 64 KB), hasta algunos cientos de kilobytes (128, 256 KB). El tamaño de la SRAM puede

[1]SRAM: Static Random Access Memory

estar entre pocos bytes (64, 128, 256, 512 B), hasta algunos pocos kilobytes (1, 2, 8, 32 KB). Dentro de los periféricos encontramos elementos como: reloj interno (desde unos kilohertz hasta unos 8 MHz o incluso 20 MHz), puertos E/S (entrada y salida digital), interrupciones digitales (que provocan la ejecución de porciones de código específicas ante la llegada de señales exteriores), timers (con 8 o 16 bits de resolución) para ejecutar temporizaciones precisas y generar señales PWM (*pulse width modulation*) en conjunto con un comparador, CAD (convertidores análogo-digital) comúnmente de 8 o 10 bits de resolución, módulos UART (*universal asynchronous receiver-transmitter*) para permitir comunicación con una computadora convencional u otros dispositivos, etc.

Microprocesador (MPU).

Estos fueron utilizados en los primeros SE hace décadas. Típicamente tienen mayor poder computacional que los microcontroladores, aunque también hay diferentes variantes en 8, 16, 32 o 64 bits. Normalmente requiren de algún tipo de memoria de datos (RAM[2]) externa y de algún tipo de memoria de programa (ROM[3]) externa. Los diseños más avanzados incluyen varias unidades de procesamiento (UPs, *PUs*, o *cores* en inglés) en el mismo chip e incluso memoria caché[4] para optimizar la ejecución de los programas.

Field programmable gate array (FPGA).

El FPGA es un chip diseñado para ser configurado luego de su fabricación. Contiene una matriz de bloques lógicos rodeada por un perímetro de bloques de E/S. Todos los bloques pueden ser ensamblados arbitrariamente empleando un lenguaje HDL (*hardware description language*). Muchos FPGA incluyen elementos de memoria como flip-flops[5] o bloques más completos como memoria RAM. Los FPGA son utilizados principalmente en una de dos maneras: 1) codificando cada operación

[2]RAM: Random Access Memory

[3]ROM: Read Only Memory

[4]Caché: tipo de memoria de pequeño tamaño y acceso rápido, integrada con la UP. Reduce el tiempo de acceso a los datos de uso frecuente ubicados en la memoria principal.

[5]Flip-flop: bloque lógico que almacena el estado de un solo bit.

matemática o lógica de un algoritmo como un bloque que realiza dicha operación, 2) o mediante la emulación de una arquitectura de MCU o MPU dentro del FPGA para, posteriormente, ejecutar un programa existente compatible con la arquitectura implementada.

Digital signal processor (DSP).

Los DSP son procesadores que realizan operaciones paralelas en una sola instrucción. Están diseñados con bloques multiplicadores y que ejecutan operaciones de procesamiento digital de señales. Usualmente son requeridos en procesamiento de imagen y audio donde los microcontroladores tienen menor rendimiento por la limitación de poder computacional.

Application-specific integrated circuit (ASIC).

Similar a un FPGA, un ASIC se diseña mediante un lenguaje HDL para luego ser producido en masa. En estos chips pueden implementarse desde sencillas operaciones hasta toda una computadora integrada (UP, memoria y otros bloques). Estos son factibles solo cuando se desean manufacturar una gran cantidad de ellos debido a los altos costos iniciales de ingeniería.

Single board computer (SBC).

Este tipo de arquitectura, a diferencia de las demás previamente comentadas, no constituye un solo circuito integrado, sino que se trata de una pequeña placa o tarjeta de circuito electrónico. En dicha placa se colocan los componentes típicos encontrados en computadoras convencionales como las de escritorio y laptops. Es decir, en una placa se encuentra una UP, memoria RAM, receptor de micro-tarjetas a modo de «disco duro» para almacenar un sistema operativo y archivos, conectividad de gran potencia como USB, Ethernet y Wifi, así como salidas de audio y video (análogo o digital) y entrada de micrófono. Se trata de toda una computadora embebida en una placa y requiere de un sistema operativo que contenga los controladores de los periféricos y protocolos de comunicación. Por lo tanto, normalmente es utilizado un teclado, ratón y pantalla para tener acceso al hardware de la placa y correr programas.

2.3.3 Elección de una arquitectura para sistema embebido

En este trabajo se elige la arquitectura de MCU de 8 bits, la cual se encuentra dentro de los SE de gama ligera. Se dan varias razones:

- Los MCU de 8 bits poseen una arquitectura sencilla que puede ser abordada si llega a ser necesario estudiar su nivel más bajo de construcción. Además poseen periféricos que ayudan a resolver parte de (o toda) la aplicación. Su consumo de corriente puede ser tan bajo, en el orden de los μA (microampere) en modo de operación o en el orden de los nA (nanoampere) en modo de espera; aún así los fabricantes están constantemente buscando menores consumos de energía. Una enorme cantidad de diseños se encuentran disponibles a través de varias familias por una variedad de marcas, dando lugar a muchas posibilidades de eligir la que más se adecúe a una aplicación. También poseen una basta documentación en internet, libros, revistas y otros recursos.

- Empiezan a ser populares ciertos modelos de chip «híbridos» que conjuntan la funcionalidad de un MCU con un MPU. Es decir, se trata de un MPU con un número de periféricos integrados. Sin embargo, estas arquitecturas caen dentro de la clasificación de «complejas» que se tiene en este trabajo. Esto significa que su diseño y uso sigue siendo superior en complejidad a un MCU de 8 bits.

- A parte de las ventajas de reconfiguración en los FPGA, esto también representa su desventaja y mayor complejidad contra un MCU de 8 bits. En especial si el FPGA se pretende utilizar como emulador de otra arquitectura de MCU o MPU existente como lo indican las tendencias.

- Los recursos para conseguir y programar sistemas basados en FPGA y DSP quedan fuera del alcance a muchos estudiantes e interesados independientes en el desarrollo de aplicaciones de SE. Así también, lidiar con la fabricación de ASICs es un gran problema.

- Una SBC es de mayor complejidad, se pueden involucrar licencias en el uso de sistemas operativos, controladores y puesta a punto de un software corriendo en el sistema operativo. Esto para permitir el desarrollo de aplicaciones que requieren de cálculos potentes y control del hardware.

En resumen, con un MCU de 8 bits se pueden resolver las pocas tareas que un SE normalmente debe ejecutar en muchas aplicaciones sin recurrir a complicados métodos de programación y puesta a punto de una plataforma más compleja.

Este trabajo utiliza la plataforma de sistemas embebidos Arduino®. Esta plataforma ofrece una familia de tarjetas basadas en MCU de 8 bits de la marca Atmel® y un software EDI (entorno de desarrollo integrado) para escribir, compilar y cargar programas a la memoria de programa FLASH del MCU [14]. A partir de aquí, cuando se haga referencia a un sistema embebido o SE en el documento, significa un sistema en base a MCU como Arduino®.

2.3.4 Programación de sistemas embebidos

Las plataformas de SE poseen herramientas para computadoras convencionales (de ahora en adelante «computadora») que permiten el desarrollo del programa que reside en el SE. Dichas herramientas consisten en un software que permite la creación, depuración y compilación del programa, y además, un quemador. El quemador es un dispositivo externo a la computadora y se encarga de recibir el programa compilado desde la computadora (a través de una conexión serial, paralela, USB, ...) y ejecutar una secuencia de señales sobre ciertos pines específicos del MCU que permiten, finalmente, la transferencia del programa a la memoria del MCU. La Figura 2.6 ilustra la programación común de un SE.

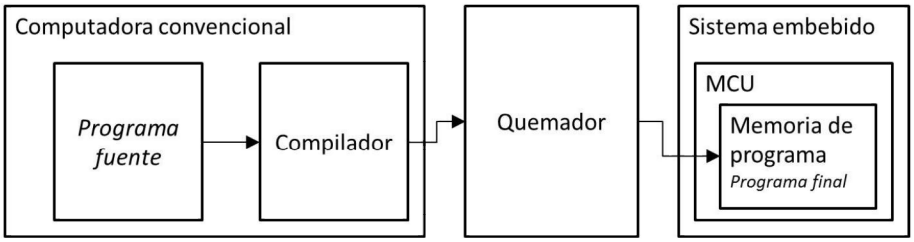


Figura 2.6: Diagrama de programación de un sistema embebido.

Sin embargo, las placas Arduino® poseen un pequeño programa, conocido como *Bootloader*, precargado en la memoria de programa FLASH del MCU. Este programa evita el uso de un quemador para realizar la programación. La Figura 2.7 ilustra este enfoque de programación. Aquí, el EDI de Arduino® posee, además del compilador, un software que se comunica con el *Bootloader* a través de comunicación serial (emulada como USB a nivel de hardware para la conexión). El *Bootloader* es quien administra la carga del programa final en otra zona de la memoria.

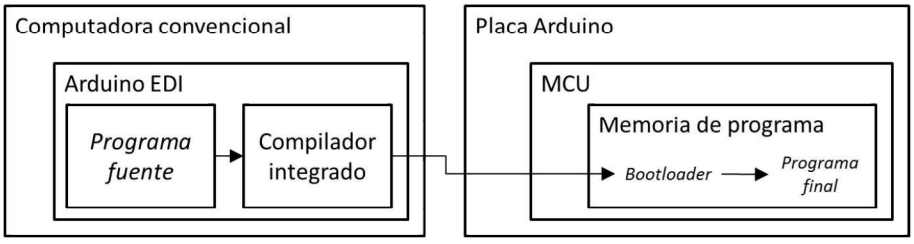


Figura 2.7: Diagrama de programación de la plataforma Arduino®.

Este enfoque de programación es muy práctico ya que evita la adquisición y uso de un quemador, el cual puede ser costoso y/o difícil obtención. Esto es logrado a cambio de sacrificar una pequeña cantidad de memoria de programa del MCU donde reside el *Bootloader* (alrededor de 2 KB en Arduino®).

Un término que se ha utilizado pero no se ha comentado es el de «compilador». El **compilador** es un software que se encarga de «traducir» lenguajes de programación. Un compilador no hace falta si se escribiera el programa utilizando directamente el lenguaje del MCU (conocido como lenguaje máquina). Sin embargo, dado la difi-

cultad de escribir en lenguaje máquina, los programas suelen escribirse en lenguaje ensamblador o lenguajes de **alto nivel** como el C, C++, Basic, etc.

El ensamblador es la traducción más sencilla del lenguaje máquina y junto con éste último se consideran lenguajes de **bajo nivel**. El lenguaje ensamblador utiliza mnemónicos en lugar de los códigos binarios que representan las instrucciones máquina. Entonces, el compilador simplemente cambia los mnemónicos por los códigos reales. Normalmente se reserva el uso del ensamblador para programas o secciones de código que deben ejecutarse con alta optimización y eficiencia en uso de memoria y tiempo de cómputo.

El lenguaje C es de los más extendidos y su utilización va desde sistemas pequeños como los SE, hasta supercomputadoras. C es considerado de alto nivel aunque tiene características que le permiten accesar el bajo nivel, particularmente acceso a bits y registros de la UP. Aquí, y en otros lenguajes como C++ (C orientado a objetos), Basic, Java, etc., el compilador ya es un software de mayor sofisticación que ejecuta la traducción al lenguaje máquina. Los lenguajes de alto nivel son de mayor uso en computadoras aunque también se utilizan en SE. Estos lenguajes ahorran mucha escritura de código y permiten reutilizar funciones o código escritos para otras arquitecturas de computación. Esto es posible ya que los diseñadores de compiladores son los que se encargan de hacer la traducción apropiadamente desde el programa fuente escrito en un cierto lenguaje, hacia el programa final que será ejecutado por esa determinada arquitectura de computación.

Las ventajas de utilizar el lenguaje C es que brinda el soporte para implementar aritmética en punto flotante, manejo de arreglos y operaciones matemáticas que pertenecen a su repertorio estándar como `pow` (para elevar a una potencia), `sqrt` (para calcular raíz cuadrada), `exp` (para calcular una potencia en base e), entre otras. Los arreglos permiten la implementación de vectores y matrices, aunque realizar operaciones matriciales (de algebra lineal) requiere de operaciones secuenciales (no paralelas) para los SE que se abordan en este trabajo.

La plataforma Arduino® tiene su propio lenguaje y está basado (y es muy parecido) al C/C++ gracias a que utiliza el compilador avr-gcc[6] y otras herramientas de software. La aritmética en punto flotante para las placas Arduino® basadas en MCU de 8 bits implementa el estándar IEEE 754 de 32 bits [15].

2.4 Algoritmos

Un **algoritmo** es un método para resolver algún problema, descrito paso a paso a través de un conjunto finito de instrucciones [16]. En este trabajo se aborda el algoritmo que está dentro del método I, el del método R y el del método L de una RN. Cada uno de los algoritmos descritos en el capítulo 3 lleva una secuencia de pasos y a veces la secuencia se bifurca por la evaluación de una condición. De cualquier modo, cada algoritmo es explicado textualmente en el orden que debe seguir la secuencia, haciendo referencia a las ecuaciones o expresiones que deben ejecutarse en cada paso.

2.4.1 Análisis de complejidad computacional y memoria

El análisis de complejidad computacional y memoria de un algoritmo se refiere principalmente, a la cantidad de tiempo y espacio necesarios para ejecutar el algoritmo respectivamente. El tiempo de ejecución, sea cual sea la unidad de tiempo, depende de la cantidad de operaciones que reliza la UP. El espacio requerido por el algoritmo se refiere a la cantidad de memoria de datos que necesita para soportar el almacenamiento de las variables involucradas. Ambos criterios, tiempo y espacio, son los más importantes en el análisis del desempeño de un algoritmo.

En este trabajo, para comparar cada RN en el sentido de complejidad computacional y uso de memoria, se analizan las ecuaciones presentes en los algoritmos y las variables en el estado de la RN.

[6]avr-gcc: Compilador para microcontroladores AVR® y que es parte de GCC (colección de compiladores del GNU®)

En el **análisis de complejidad computacional**, o solo complejidad en este trabajo, se obtiene una función que dé una aproximación de la cantidad de operaciones que debe ejecutar el SE. Para esto, se analizan principalmente operaciones matemáticas con ciclos inherentes en el peor caso de ejecución.

Operaciones matemáticas como el operador norma ($\|v\|$), sumatoria ($\sum$), multiplcación de matrices (AB), función de búsqueda de mínimo o máximo ($\min(v)$, $\max(v)$), entre otras, son llamadas «con ciclos inherentes» porque contienen operaciones iterativas en su construcción. Aquí cabe declarar que las funciones $\min(v)$ y $\max(v)$ regresan dos valores $[val, pos]$ con el valor val y posición pos del valor mínimo o máximo en un vector v respectivamente.

Por otro lado, el peor caso de ejecución del algoritmo, será cuando en una llamada al mismo, se ejecuten sus secciones de código más pesadas (de mayor cantidad de operaciones). Esto es, si en el algorimo existen dos bloques de código cuyas ejecuciones son decididas mediante una evaluación de condición, se analiza el bloque más pesado y es éste quien forma parte de la expresión de complejidad. Por lo tanto, en cada RN se expresa la complejidad en términos del número de neuronas N, entradas M y salidas P mediante una declaración tipo (2.11) utilizando $C_{\text{RN-Método}}$ como complejidad de un método de una RN.

$$\mathrm{C}_{\mathrm{RN-Metodo}} = f(N, M, P) \tag{2.11}$$

El **análisis de consumo de memoria** se hace mediante un conteo de las celdas de memoria que requiere cada variable involucrada en el estado S de una RN. El término «celda de memoria» hace referencia a cada variable escalar y cada elemento en un vector o matriz. Este análisis, al igual que el de complejidad, queda expresado en una función de N, M y P mediante una declaración tipo (2.12) utilizando $\mathrm{MEM}_{\mathrm{RN}}$ como consumo de memoria de la RN.

$$\mathrm{MEM}_{\mathrm{RN}} = f(N, M, P) \tag{2.12}$$

Capítulo 3

Análisis de las redes neuronales elegidas

Dentro de la literatura se destacan tres arquitecturas para realizar aprendizaje en línea. Las tres son del tipo RNAUC y son las siguientes:

- Red neuronal de máxima sensibilidad (RNMS).
- Online sequential extreme learning machine (OSELM).
- Resource allocating network (RAN).

3.1 Red neuronal de máxima sensibilidad (RNMS)

La **Red neuronal de máxima sensibilidad (RNMS)** ha sido presentada como una RN basada en la teoría de sistemas funcionales con rápido aprendizaje. Al inicio tiene una sola neurona, luego incrementa su tamaño conforme la aplicación requiere más neuronas. En caso de agotarlas es capaz de reutilizarlas apropiadamente [17, 18].

Cuando un patrón es presentado al método R las neuronas se excitan, y si la neurona más excitada sobrepasa un margen de sensibilidad (ms), entonces solo esa

neurona es utilizada para calcular las salidas de la RN. En este caso se dice que se está en un estado de máxima sensibilidad. Si al presentar el patrón al método R no se detecta dicho estado, entonces todas las neuronas son utilizadas para calcular las salidas de la RN. Este mecanismo es similar al de *winner-take-all* (el ganador lo toma todo). Además, este estado de máxima sensibilidad es utilizado para establecer un mecanismo de aprendizaje en el método L. A continuación la lista de algunas características importantes de la RNMS:

- Cualquier función de activación con forma de campana puede ser utilizada, como la gaussiana, campana, o triangular.
- Utiliza ecuaciones simples que la hacen fácil de ser implementada, aún en SE diminutos.
- Requiere dos parámetros (ms y λ) que deben ser seleccionados antes de iniciar el aprendizaje.
- Trabaja para valores normalizados. Cada elemento de los vectores x y y_d deben ser normalizados al rango $[0, 1] \in \mathbb{R}$ antes de entrar al método R y/o L. Luego pueden ser utilizados para el procesamiento de la RN y, finalmente, el vector de salidas y que produce la RNMS debe ser desnormalizado desde el mismo rango.

Dado que la RNMS puede variar su tamaño, N es utilizado como la cantidad máxima de neuronas que puede utilizar y N_C es utilizado como la cantidad de neuronas actualmente en uso en un instante determinado.

3.1.1 RNMS, método I

El método I de la RNMS requiere los siguientes pasos:

- Decidir la función de activación a usar (gaussiana, campana o triangular) y la cantidad máxima de neuronas N.

- Establecer $N_C = 1$, es decir, empezar con una sola neurona.
- Asignar -1 a $W_{n,m}$, $\forall n$ y $\forall m$.
- Asignar 0 (cero) a $Q_{n,p}$, $\forall n$ y $\forall p$.
- Crear $h \in \mathbb{R}^N$, el vector de cómputos neuronales (salidas de la capa oculta).
- Crear $U \in \mathbb{R}^N$, el vector de uso de neuronas, con todos los elementos a 1 (este valor indica el uso máximo).
- Establecer el valor $fu = 0.001$ (factor de olvido) u otro pequeño valor en el rango $[0, 1] \in \mathbb{R}$.
- Establecer el valor del margen de sensibilidad ms y el ancho de campana λ (todas las neuronas emplean el mismo valor de λ, por lo que λ aquí es escalar). Ambos valores, ms y λ, en el rango $[0, 1] \in \mathbb{R}$.

El estado de la RNMS es (3.1).

$$S_{\text{RNMS}} = \{W, Q, \lambda, ms, fu, U, h, N_C, [maxval, minval], [maxn, updaten]\} \quad (3.1)$$

3.1.2 RNMS, método R

Descripción

Esta RN no utiliza la ecuación (2.7) presentada en la subsección 2.2.2 para calcular la salidas de la RN para un patrón dado debido a que su diseño proviene de la teoría de sistemas funcionales. La función de activación de la RNMS determina una magnitud de distancia entre los pesos sinápticos de las neuronas ocultas y las entradas de la RN. Es necesario que una distancia pequeña provea un alto nivel de excitación o activación, así que una función de activación con un pico a distancia cero puede ser utilizada.

Cuando es utilizada una función de activación gaussiana (2.4) la RNMS se parece a una RN tipo RBF. Pero, como ha sido comentado, se pueden utilizar funciones campana (3.2) o incluso triangular (3.3).

$$\mathrm{fa_c}(a,l) = \begin{cases} \dfrac{1}{1+(a/l)^2} & \text{si } l \neq 0 \\ 0 & \text{si } l = 0 \end{cases} \tag{3.2}$$

$$\mathrm{fa_{tr}}(a,l) = \begin{cases} 1 - l|a| & \text{si } l|a| < 1 \\ 0 & \text{otra manera} \end{cases} \tag{3.3}$$

Cualquiera que sea la elegida, esta RN utiliza el mismo valor en todos sus elementos del vector de anchos de campana λ el cual es pasado como argumento l, mientras que el argumento a siempre es $a = \sqrt{\sum_{m=1}^{M} (W_{n,m} - x_m)^2}$ para una neurona n en particular.

En el método R de la RNMS, primero, es leído el patrón x. Luego, el vector de salidas ocultas h es calculado utilizando la ecuación (3.4), en la cual $\mathrm{fa_{elegida}}(\cdot)$ es la función de activación elegida durante el método I.

$$h_n(x, W, \lambda) = \mathrm{fa_{elegida}}\,(a, \lambda)\,, \quad \forall n = 1, 2, ..., N_C \tag{3.4}$$

Para determinar la neurona más estimulada o excitada se utiliza la ecuación (3.5)[1], y si $maxval > ms$ entonces la RNMS está en un estado de máxima sensibilidad. En este estado, las salidas de la RN son calculadas utilizando solamente la neurona estimulada de forma máxima utilizando la ecuación (3.6) y es indicado un alto nivel de uso para esta neurona $maxn$ mediante la asignación $U_{maxn} = 1$. El registro de actividad o uso de cada neurona mediante el vector U es con fin de que el método L pueda determinar la neurona menos utilizada, es decir, menos frecuentemente estimulada a máxima sensibilidad.

$$[maxval, maxn] = \max(h_1, h_2, ..., h_{N_C}) \tag{3.5}$$

[1]La función $\max(v)$ regresa la posición y valor del elemento máximo del vector v realizando la búsqueda del máximo a partir del primer elemento del vector. Si existe más de un elemento máximo se regresa el primero encontrado.

$$y_p = Q_{maxn,p} * h_{maxn}, \quad \forall p \tag{3.6}$$

Por otro lado, cuando la RNMS no está en un estado de máxima sensibilidad, las salidas de la RN son calculadas utilizando la ecuación (3.7).

$$y_p = \frac{\sum_{n=1}^{N_C} Q_{n,p} * h_n}{\sum_{n=1}^{N_C} h_n}, \quad \forall p \tag{3.7}$$

Diagrama de flujo

La Figura 3.1 presenta al algoritmo en un diagrama de flujo.

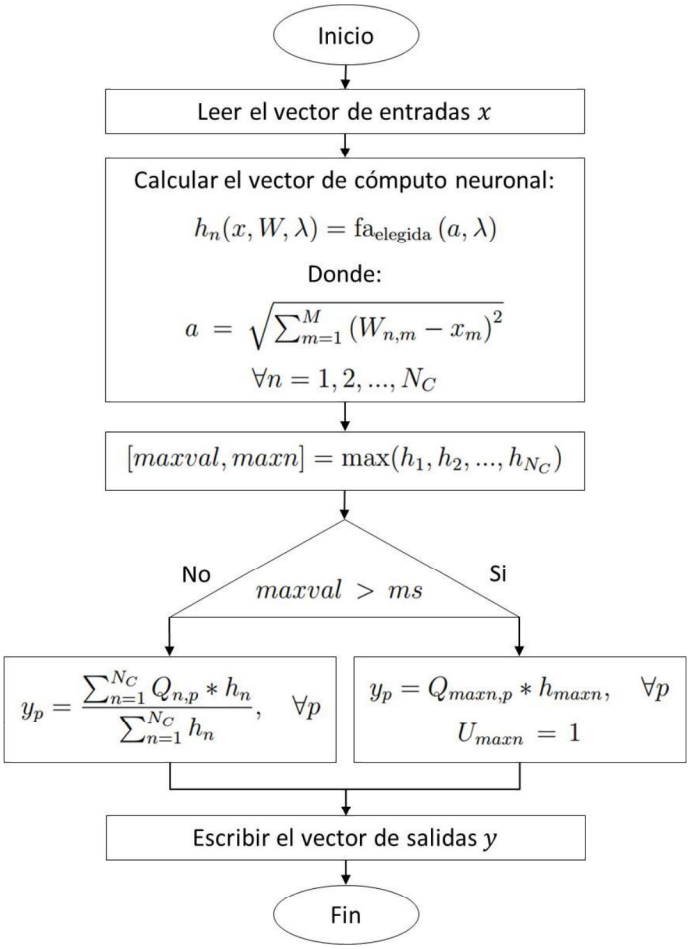


Figura 3.1: Diagrama de flujo del método R para RNMS.

Análisis de complejidad

La complejidad para este método es $N_C(M+1)$ cuando hay un estado de máxima sensibilidad, y es $N_C(M+P+2)$ cuando no lo hay. Esto significa que la primer cantidad (de operaciones) $N_C M$ es consumida calculando el vector h mediante la ecuación (3.4), luego una cantidad N_C es consumida en la función $\max(\cdot)$. Adicionalmente, si no hay un estado de máximo sensibilidad se consume $N_C P$ operaciones en el numerador y N_C en el denominador de la división en la ecuación (3.7).

Así, considerando el peor caso donde $N_C = N$ y no hay un estado de máxima sensibilidad, la complejidad es lineal y es expresada como (3.8).

$$\mathrm{C_{RNMS-R}} = N(M+P+2) \tag{3.8}$$

3.1.3 RNMS, método L

Descripción

Como ha sido comentado, el estado de máxima sensibilidad es utilizado para llevar a cabo el aprendizaje en la RNMS. Para esto, primero es calculado el vector h y es revisada la máxima sensibilidad de acuerdo al margen ms. Cuando una neurona está en máxima sensibilidad entonces es promediada con la información presente, sino, la información es almacenada directamente en una nueva neurona o en la menos usada si es que no hay más neuronas vacías disponibles.

El algoritmo del método L inicia leyendo el par $\{x, y_d\}$, luego calcula el correspondiente vector de salidas ocultas h justo como es hecho en el método R. Entonces, se actualiza el vector de uso U mediante la ecuación (3.9). Después, se determina si existe máxima sensibilidad de acuerdo a la ecuación (3.5) y la condición $maxval > ms$.

$$U_n \leftarrow U_n - fu, \quad \forall n = 1, 2, ..., N_C \tag{3.9}$$

Si hay un estado de máxima sensibilidad entonces solo la neurona $maxn$ es ajustada mediante las ecuaciones (3.10), (3.11) y (3.12).

$$W_{maxn,m} \leftarrow (W_{maxn,m} + x_m)/2, \quad \forall m \tag{3.10}$$

$$Q_{maxn,p} \leftarrow (Q_{maxn,p} + y_{d_p})/2, \quad \forall p \tag{3.11}$$

$$U_{maxn} = 1 \tag{3.12}$$

Si no hay máxima sensibilidad, una nueva neurona es empleada (exactamente la siguiente, $updaten = N_C + 1$) con la información directa mediante las ecuaciones (3.13), (3.14) y (3.15), pero solo si hay neuronas disponibles, es decir, si $N_C + 1 \leq N$.

$$W_{updaten,m} = x_m, \quad \forall m \tag{3.13}$$

$$Q_{updaten,p} = y_{d_p}, \quad \forall p \tag{3.14}$$

$$U_{updaten} = 1 \tag{3.15}$$

Si no hay más neuronas, entonces las mismas ecuaciones son aplicadas pero para la neurona menos usada que se determina mediante la ecuación (3.16).

$$[minval, updaten] = \min(U) \tag{3.16}$$

Diagrama de flujo

La Figura 3.2 presenta al algoritmo en un diagrama de flujo.

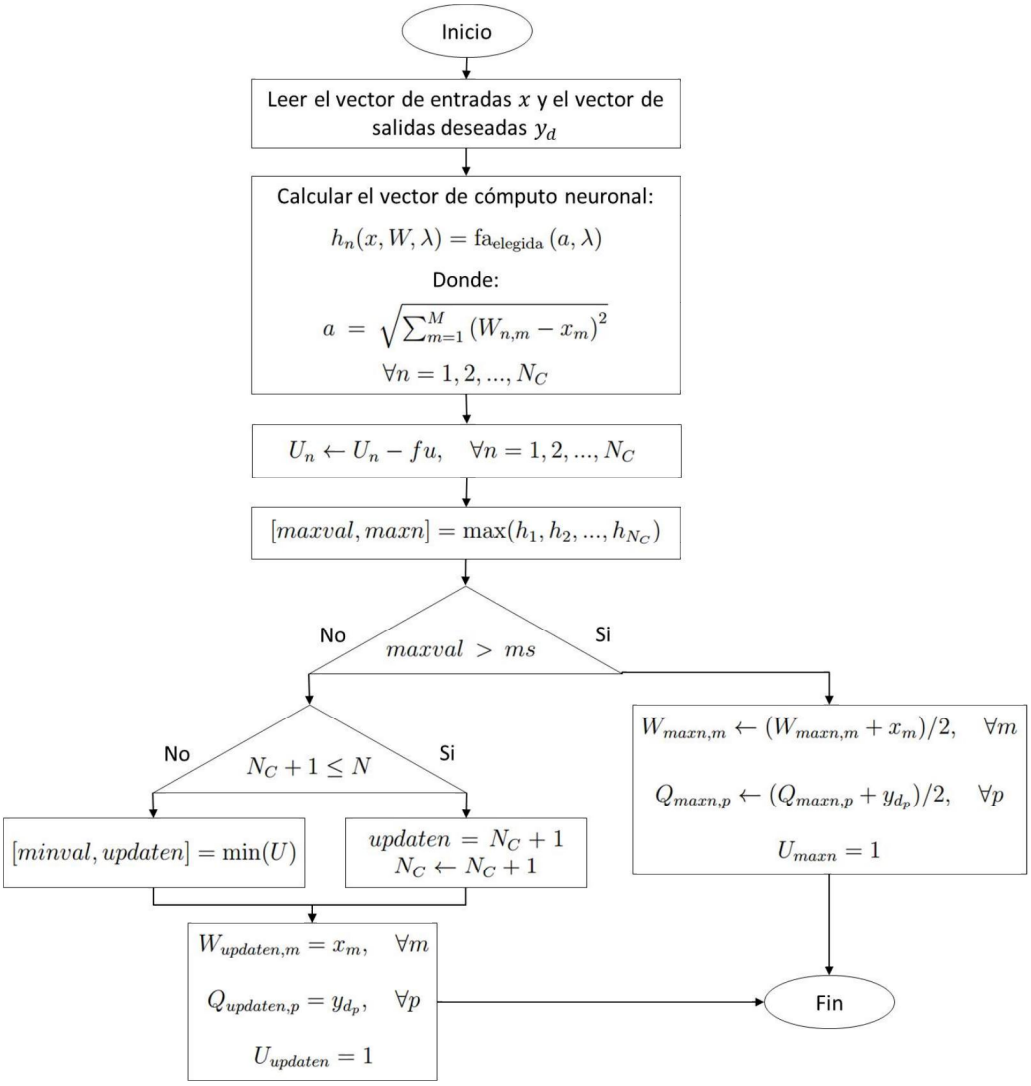


Figura 3.2: Diagrama de flujo del método L para RNMS.

Análisis de complejidad

La complejidad para este método es $N_C(M+2)+M+P$ cuando hay un estado de máxima sensibilidad y es $N_C(M+2)+N+M+P$ cuando no hay un estado de máxima sensibilidad y no hay más neuronas disponibles. En ambos casos una cantidad (de operaciones) $N_C(M+1)$ es consumida como ocurre en el método R, es decir, en el cálculo del vector h y en el uso de la función $\max(\cdot)$. Luego, una cantidad N_C es consumida en la actualización del vector U dando como resultado el término $N_C(M+2)$. Finalmente, se consume una cantidad $M+P$ en el aprendizaje

propiamente, es decir, en la actualización de las matrices W y Q teniendo una cantidad adicional de N operaciones en el uso de la función $\min(\cdot)$ cuando no hay máxima sensibilidad.

Así, el peor caso es el último mencionado con $N_C = N$, por lo tanto, la complejidad para este método es lineal y es expresada como (3.17).

$$\mathrm{C}_{\mathrm{RNMS-L}} = N(M+3) + M + P \tag{3.17}$$

3.1.4 RNMS, consumo de memoria

Considerando que cada elemento de cada variable del estado S_{RNMS} ocupa una celda de la memoria de datos, entonces el consumo de memoria está dado por la ecuación (3.18). Es decir, las variables escalares aportan una celda cada una, mientras que las variables tipo vector o matriz aportan una o más celdas de memoria según su dimensión. Ejemplos: el vector h consume N celdas y la matriz W consume NM celdas.

$$\mathrm{MEM}_{\mathrm{RNMS}} = N(M+P+2) + 6 \tag{3.18}$$

Aquí se ha considerando a λ como escalar debido a que todas las neuronas utilizan el mismo ancho de campana. Además es considerado que *minval* y *maxval* son alias de la misma celda de memoria, y lo mismo para *maxn* y *updaten* los cuales son alias de otra celda de memoria. «Alias» significa que son diferentes nombres para la misma celda.

3.2 Online sequential extreme learning machine (OSELM)

La **Online sequential extreme learning machine (OSELM)** ha sido propuesta como una RN de aprendizaje rápido y que no requiere parámetros que deban seleccionarse al iniciar el aprendizaje, excepto por la cantidad de neuronas N [19].

OSELM se construyó a partir de ELM (*extreme learning machine*) [20, 21] a la cual se le reempleza la operación de pseudo-inversa por una solución RLS (*recursive least squares*) con fin de convertir a ELM en una forma en línea. A continuación la lista de algunas características importantes de OSELM:

- Trabaja para neuronas aditivas con cualquier función de activación acotada, continua a trozos y para neuronas RBF con cualquier función de activación integrable continua a trozos.
- Los pesos sinápticos de entrada W y los valores del bias b (para neuronas aditiva-sigmoide) o λ (para neuronas RBF), son elegidos aleatoriamente al inicio y nunca son ajustados durante el aprendizaje.
- Los pesos de salida Q son determinados analíticamente para cada nuevo patrón entrante.
- Los patrones entrantes pueden ser de a uno-por-uno, como es usual en el aprendizaje en línea, pero también es soportado de a bloque-por-bloque de tamaño arbitrario.

En este trabajo se ha analizado a OSELM para neuronas tipo aditiva-sigmoide y RBF, sin embargo, también es posible trabajar con otro tipo de neuronas que se soporte en OSELM.

3.2.1 OSELM , método I

El método I de OSELM requiere los siguientes pasos:

- Decidir el tipo de neurona a utilizar: aditiva-sigmoide o RBF, y la cantidad de neuronas de la capa oculta N.
- Especificar el tamaño N_0 del bloque inicial que será aprendido. Este tamaño debe ser mayor o igual a N ($N_0 \geq N$).

- Asignar valores aleatorios a $W_{n,m}$, $\forall n$ y $\forall m$.
- Asignar valores aleatorios a b_n, $\forall n$ si se usan neuronas aditiva-sigmoide o asignarlos a λ_n, $\forall n$ si se usan neuronas RBF.
- Establecer a cero cada elemento de la matriz Q.
- Crear $h \in \mathbb{R}^N$, el vector de salidas de la capa oculta.
- Crear $H_0 \in \mathbb{R}^{N_0 \times N}$, la matriz de salidas de la capa oculta utilizada en la inicialización de RLS.
- Crear $H \in \mathbb{R}^{N \times N}$, la matriz que se arrastra en cada iteración de RLS.
- Crear $D \in \mathbb{R}^{N_0 \times P}$, la matriz de las N_0 salidas deseadas correspondientes al bloque inicial.

El estado de OSELM es el siguiente:

$$S_{\text{OSELM}} = \{W, Q, [b, \lambda], N_0, h, H_0, H, D\} \tag{3.19}$$

3.2.2 OSELM, método R

Descripción

El método R en OSELM emplea las ecuaciones vistas en la subsección 2.2.2. Para un patrón x que se presenta al método R, primero se calcula el vector de salidas de la capa oculta h utilizando la ecuación (3.20) si se trata de neuronas aditiva-sigmoide o la ecuación (3.21) si se trata de neuronas RBF. Finalmente se aplica la ecuación (3.22) para calcular las salidas de la RN.

$$h_n(x, W, b) = \text{fa}_s \left(b_n + \sum_{m=1}^{M} W_{n,m} * x_m \right), \quad \forall n \tag{3.20}$$

$$h_n(x, W, \lambda) = \text{fa}_g \left(\sqrt{\sum_{m=1}^{M} (W_{n,m} - x_m)^2}, \lambda_n \right), \quad \forall n \tag{3.21}$$

$$y_p = \sum_{n=1}^{N} Q_{n,p} * h_n\left(x, W, param\right), \quad \forall p \tag{3.22}$$

Diagrama de flujo

La Figura 3.3 presenta al algoritmo en un diagrama de flujo. En esta figura el bloque de cómputo neuronal corresponde a neuronas aditiva-sigmoide (ecuación 3.20) pero puede utilizarse la ecuación para neuronas RBF (ecuación 3.21).

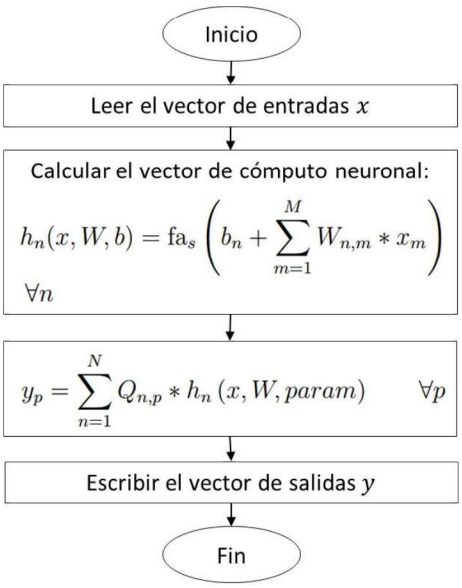


Figura 3.3: Diagrama de flujo del método R para OSELM.

Análisis de complejidad

La complejidad para este método de OSELM es lineal y está expresada como (3.23). Aquí una cantidad (de operaciones) NM es consumida en el cálculo del vector h y una cantidad NP es consumida en el cálculo de las salidas y.

$$\mathrm{C}_{\mathrm{OSELM-R}} = N(M + P) \tag{3.23}$$

3.2.3 OSELM, método L

Descripción

El aprendizaje en OSELM está formado por dos fases. La primera es la llamada *fase boosting* y solo es ejecutada para el primer par $\{x, y_d\}$, el segundo par, el tercer par, ..., hasta el N_0-ésimo par. La segunda fase es la llamada *fase secuencial* y es ejecutada para el resto de los patrones en cada llamada al método L, es decir, para el par $(N_0 + 1)$-ésimo, el par $(N_0 + 2)$-ésimo, ..., hasta que el aprendizaje finaliza. En OSELM, los patrones pueden llegar uno-por-uno o de a bloque-por-bloque; sin embargo, con fin de mantener las cosas más simples como es deseado para los SE, se utilizan las ecuaciones en su versión simplificada que permiten solo el aprendizaje de patrones uno-por-uno. Además, esto es con la intención de igualmente comparar el desempeño contra las otras arquitecturas analizadas en este trabajo que, como es visto en su texto correspondiente, aprenden solo patrones de a uno-por-uno.

El método L de OSELM es como sigue. Sea i el contador de patrones entrantes, el algoritmo inicia leyendo el i-ésimo par $\{x, y_d\}$. Cuando $i \leq N_0$ la *fase boosting* es ejecutada, y cuando $i > N_0$ la fase *fase secuencial* es ejecutada.

Fase boosting.

En esta fase, primero se calcula el vector h mediante las ecuaciones (3.20) y (3.21) según el tipo de neurona elegido. Luego se almacena el vector h en la fila i-ésima de H_0 y se almacena y_d en la fila i-ésima de D. Cuando i sea igual a N_0 $(i = N_0)$ se utilizan los pares acumulados para ejecutar el modo *batch* utilizando las ecuaciones (3.24) y (3.25).

$$H = (H_0^T H_0)^{-1} \tag{3.24}$$

$$Q = H H_0^T D \tag{3.25}$$

Fase secuencial.

En esta fase, primero se calcula el vector h mediante las ecuaciones (3.20) y (3.21) según el tipo de neurona elegido. Luego se actualiza H y después Q utilizando las ecuaciones (3.26) y (3.27) respectivamente.

$$H \leftarrow H - (Hhh^T H)/(1 + h^T Hh) \tag{3.26}$$

$$Q \leftarrow Q + Hh(y_d^T - h^T Q) \tag{3.27}$$

Diagrama de flujo

La Figura 3.4 presenta al algoritmo en un diagrama de flujo. En esta figura el bloque de cómputo neuronal corresponde a neuronas aditiva-sigmoide (ecuación 3.20) pero puede utilizarse la ecuación para neuronas RBF (ecuación 3.21).

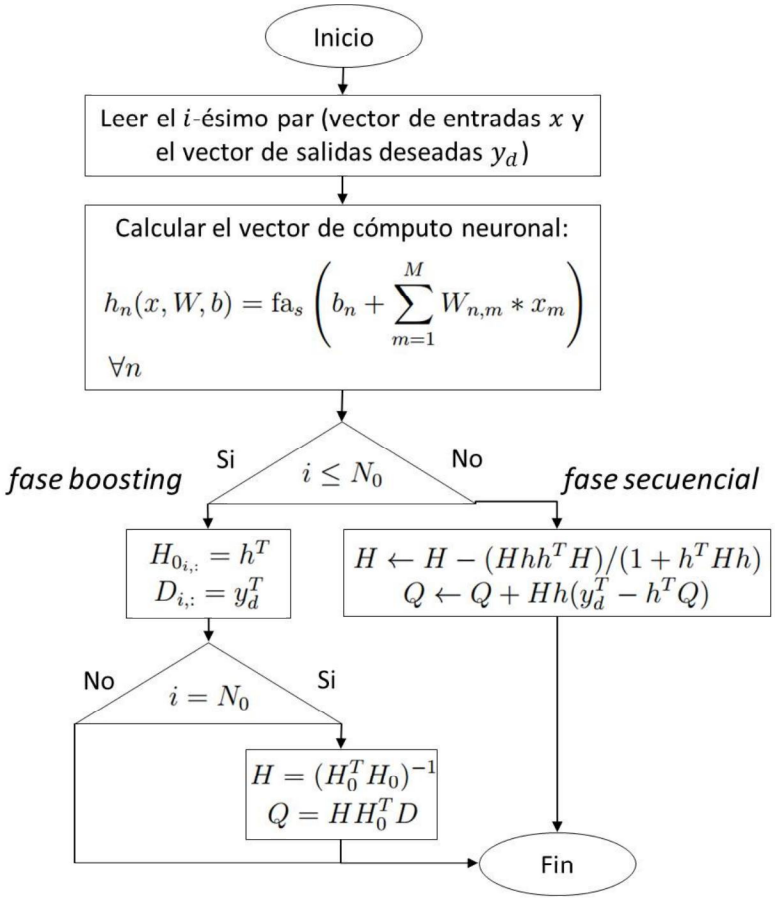


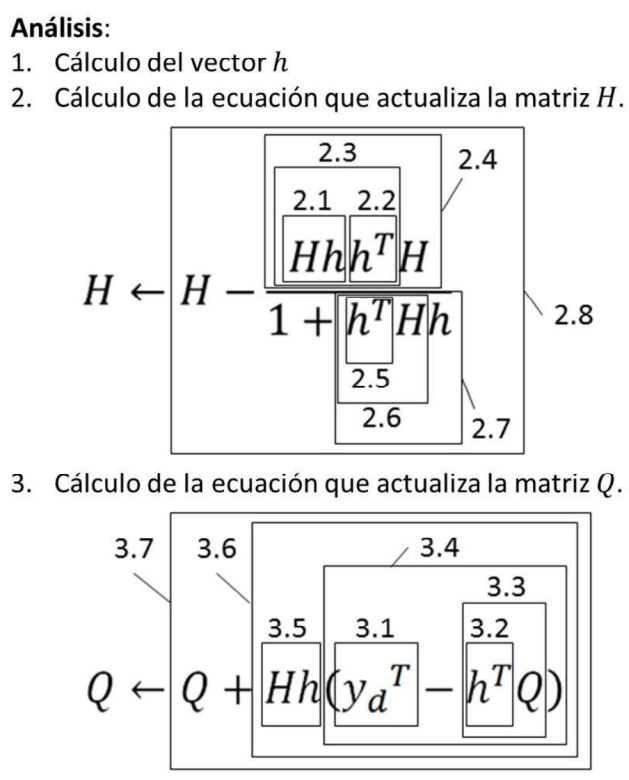
Figura 3.4: Diagrama de flujo del método L para OSELM.

Análisis de complejidad

Se analiza la complejidad del método L de OSELM utilizando algoritmos sencillos para ejecutar las operaciones matriciales. El método L está compuesto de tres casos: 1) cuando $i < N_0$, 2) cuando $i = N_0$, 3) cuando $i > N_0$. Se evita el anásis de complejidad del caso 2 ya que es ejecutado una sola vez, y se evita el análisis del caso 1 porque es utilizado muy pocas veces comparado con las muchas veces que el caso 3 se ejecuta en aplicaciones reales. Así, la complejidad para el caso 3 ($i > N_0$) es cúbica y es expresada en (3.28).

$$\mathrm{C}_{\mathrm{OSELM-L}} = N^3 + 5N^2 + (M + 3P + 4)N + 2P \tag{3.28}$$

Para comprender como es obtenida esta expresión de complejidad la Figura 3.5 desglosa las operaciones matemáticas del caso 3 ($i > N_0$). Una vez analizada cada operación como indica la mencionada figura, se suman todas las cantidades de operaciones consumidas que finalmente resultan en la presentada expresión de complejidad (3.28).



Operaciones consumidas en cada uno de los puntos analizados:

1.	NM
2.1.	N^2
2.2.	N
2.3.	N^2
2.4.	N^3
2.5.	N
2.6.	N^2
2.7.	N
2.8.	N^2
3.1.	P
3.2.	N
3.3.	NP
3.4.	P
3.5.	N^2
3.6.	NP
3.7.	NP

Figura 3.5: Análisis de operaciones para obtener la complejidad del método L de OSELM.

Cabe hacer ver que en el caso 2 hay una operación de matriz inversa, sin embargo, como es comentado en la literatura, el uso de la operación de pseudo-inversa basada en el cálculo SVD[2] (PISVD) hace a este algoritmo más robusto y ayuda a lidiar con matrices singulares o cerca de la singularidad. PISVD es un cálculo pesado que puede ser considerado ejecutar fuera de línea o en línea según si el SE lo soporta. Como se comenta en el capítulo 5, no se implementó a PISVD en el SE para los experimentos, así que la complejidad del método L declarada en (3.28) no incluye la *fase boosting*.

[2]SVD: Singular Value Decomposition

3.2.4 OSELM, consumo de memoria

Considerando que cada elemento en cada variable del estado S_{OSELM} ocupa una celda de la memoria de datos, entonces el consumo de memoria es $N(N+M+P+N_0+2)+N_0P+1$, no obstante, dado que este trabajo es con enfoque en SE, entonces N_0 se elige lo más pequeña posible, es decir, se elige $N_0 = N$. Así, bajo esta consideración, el consumo de memoria es (3.29).

$$\text{MEM}_{\text{OSELM}} = 2N^2 + (M + 2P + 2)N + 1 \tag{3.29}$$

3.3 Resource allocating network (RAN)

La **Resource allocating network (RAN)** ha sido propuesta como una RN tipo RBF, con apredizaje rápido y algunos parámetros que deben ajustarse antes de iniciar el aprendizaje. Esta RN empieza utilizando una sola neurona, luego conforme se dan patrones para aprender, más neuronas son utilizadas de acuerdo con un criterio de novedad. Este criterio decide cuando ajustar las neuronas en uso actual a través de un algoritmo gradiente y cuando asignar una nueva neurona [22]. A continuación la lista de algunas características importantes de RAN:

- El criterio de novedad ayuda a mantener una estrucutra comptacta, es decir, usar el mínimo de neuronas.
- Como una RN con neuronas RBF, se utilizan funciones de activación gaussiana que responden a una pequeña región del espacio, así una nueva neurona asignada no interfiere con las neuronas asignadas previamente.
- El método de aprendizaje utiliza el error de aproximación para asignar los centros de las neuronas incrementando la exactitud.
- Las ecuaciones son relativamente simples lo que la hace relativamente fácil de ser implementada.

Tal como en la RNMS, N es utilizado como la cantidad máxima de neuronas que puede utilizar y N_C es utilizado como la cantidad de neuronas actualmente en uso en un instante determinado.

3.3.1 RAN, método I

El método I de RAN requiere los siguientes pasos:

- Decidir la cantidad máxima de neuronas N.
- Establecer $N_C = 1$, es decir, empezar con una sola neurona.
- Asignar 0 (cero) a $W_{n,m}$, $\forall n$ y $\forall m$.
- Asignar 0 (cero) a $Q_{n,p}$, $\forall n$ y $\forall p$.
- Crear $h \in \mathbb{R}^N$, el vector de salidas de neuronas en la capa oculta.
- Crear $b \in \mathbb{R}^P$, el vector de bias. A diferencia de una RN basada en neuronas aditiva-sigmoide presentada en la subsección 2.2.2, aquí es utilizado el bias b en la capa de salida.
- Crear $\lambda \in \mathbb{R}^N$, el vector de ancho de campana.
- Crear $E \in \mathbb{R}^P$, el vector de error.
- Establecer $me = 0$, la magnitud del vector de error a cero.
- Establecer el valor de cada parámetro: factor de traslape k, intervalos de resolución $\delta_{\min}$, $\delta_{\max}$, constante de decaimiento τ, factor de aprendizaje α y radio de error ϵ.
- Establecer la escala de resolución δ como el máximo, es decir, $\delta = \delta_{\max}$.
- Establecer a cero la distancia al centro más cercano $d_{\min} = 0$.

El estado de RAN es (3.30).

$$S_{\text{RAN}} = \{W, Q, b, \lambda, h, E, me, N_C, k, \delta, \delta_{\min}, \delta_{\max}, \tau, \alpha, \epsilon, d_{\min}\} \tag{3.30}$$

3.3.2 RAN, método R

Descripción

A diferencia de las ecuaciones presentadas en la subsección 2.2.2, RAN utiliza el vector de bias en la capa de salida y no en la capa oculta. Para un patrón x que se presenta al método R, primero se calcula el vector de salidas de la capa oculta h utilizando la ecuación (3.31). Finalmente se aplica la ecuación (3.32) para calcular las salidas de la RN.

$$h_n(x, W, \lambda) = \mathrm{fa}_g \left(\sqrt{\sum_{m=1}^{M} (W_{n,m} - x_m)^2}, \lambda_n \right), \quad \forall n = 1, 2, ..., N_C \tag{3.31}$$

$$y_p = b_p + \sum_{n=1}^{N_C} Q_{n,p} * h_n(x, W, \lambda), \quad \forall p \tag{3.32}$$

Diagrama de flujo

La Figura 3.6 presenta al algoritmo en un diagrama de flujo.

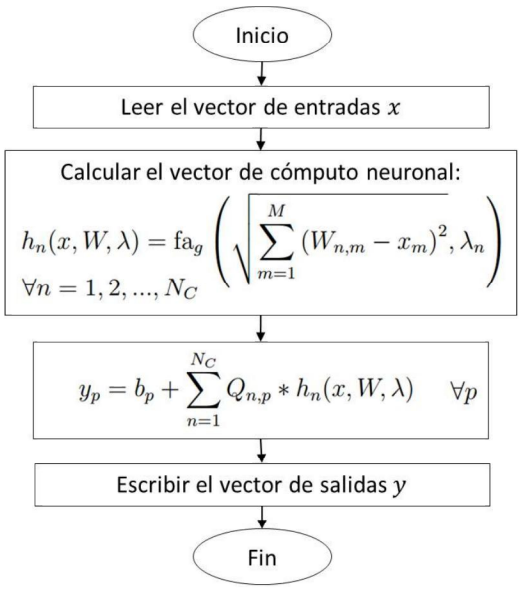


Figura 3.6: Diagrama de flujo del método R para RAN.

Análisis de complejidad

La complejidad para este método de RAN es $N_C(M+P)$, pero considerando el peor caso donde $N_C = N$, la complejidad es lineal y es expresada en (3.33). Aquí una cantidad (de operaciones) NM es consumida en el cálculo del vector h y una cantidad NP es consumida en el cálculo de las salidas y.

$$\mathrm{C_{RAN-R}} = N(M+P) \tag{3.33}$$

3.3.3 RAN, método L

Descripción

La idea de aprendizaje en RAN es calcular la respuesta al patrón, luego, de acuerdo con el error de aproximación y la distancia a la neurona más cercana, el método L determina la novedad del patrón, es decir, que tanto se conoce sobre él. Esta condición de novedad puede dictar el ajuste de todas las neuronas para asimilar el nuevo patrón, o asignar una nueva neurona para reconocer este patrón en futuros momentos.

El método L inicia leyendo el par $\{x, y_d\}$ y si es el primer patrón dado para aprender entonces se ejecuta además la asignación $b = y_d$. Luego, el método R es ejecutado para obtener la respuesta y para este patrón x. Después, se calcula el vector de error de aproximación $E = y_d - y$ y su magnitud $me = \|E\| = \sqrt{\sum_{p=1}^{P} E_p^2}$.

La ecuación (3.34) es utilizada para determinar la distancia a la neurona más cercana y el criterio de novedad (3.35) es evaluado.

$$[d_{\min}, minn] = \min_n(\|W_{n,m} - x_m\|), \quad \forall m \tag{3.34}$$

$$(me > \epsilon) \quad \& \quad (d_{\min} > \delta) \quad \& \quad (N_C + 1 \leq N) \tag{3.35}$$

Si el criterio (3.35) cumple, entonces una nueva neurona es asignada ($N_C \leftarrow N_C + 1$) utilizando las ecuaciones (3.36) y (3.37), además, si es la primer neurona en ser asignada, entonces $\lambda_{N_C} = k\delta$, de otro modo $\lambda_{N_C} = kd_{\min}$.

$$W_{N_C,m} = x_m, \quad \forall m \tag{3.36}$$

$$Q_{N_C,p} = y_{d_p}, \quad \forall p \tag{3.37}$$

Pero si el criterio de novedad (3.35) no cumple, entonces todas las N_C neuronas son ajustadas mediante gradiente descendiente utilizando las ecuaciones (3.38), (3.39) y (3.40). Finalmente, si $\delta > d_{\min}$ entonces se actualiza $\delta \leftarrow \delta * \exp(-1/\tau)$.

$$Q_{n,p} \leftarrow Q_{n,p} + (\alpha * E_p * h_n), \quad \forall p, \forall n = 1, 2, ..., N_C \tag{3.38}$$

$$\begin{aligned} W_{n,m} \leftarrow W_{n,m} + (2 * \alpha/\lambda_n) * (x_m - W_{n,m}) * \\ h_n * \left[\sum (E_p * Q_{n,p})\right], \quad \forall m, \forall p, \forall n = 1, 2, ..., N_C \end{aligned} \tag{3.39}$$

$$b_p \leftarrow b_p + (\alpha * E_p), \quad \forall p \tag{3.40}$$

Diagrama de flujo

La Figura 3.7 presenta al algoritmo en un diagrama de flujo.

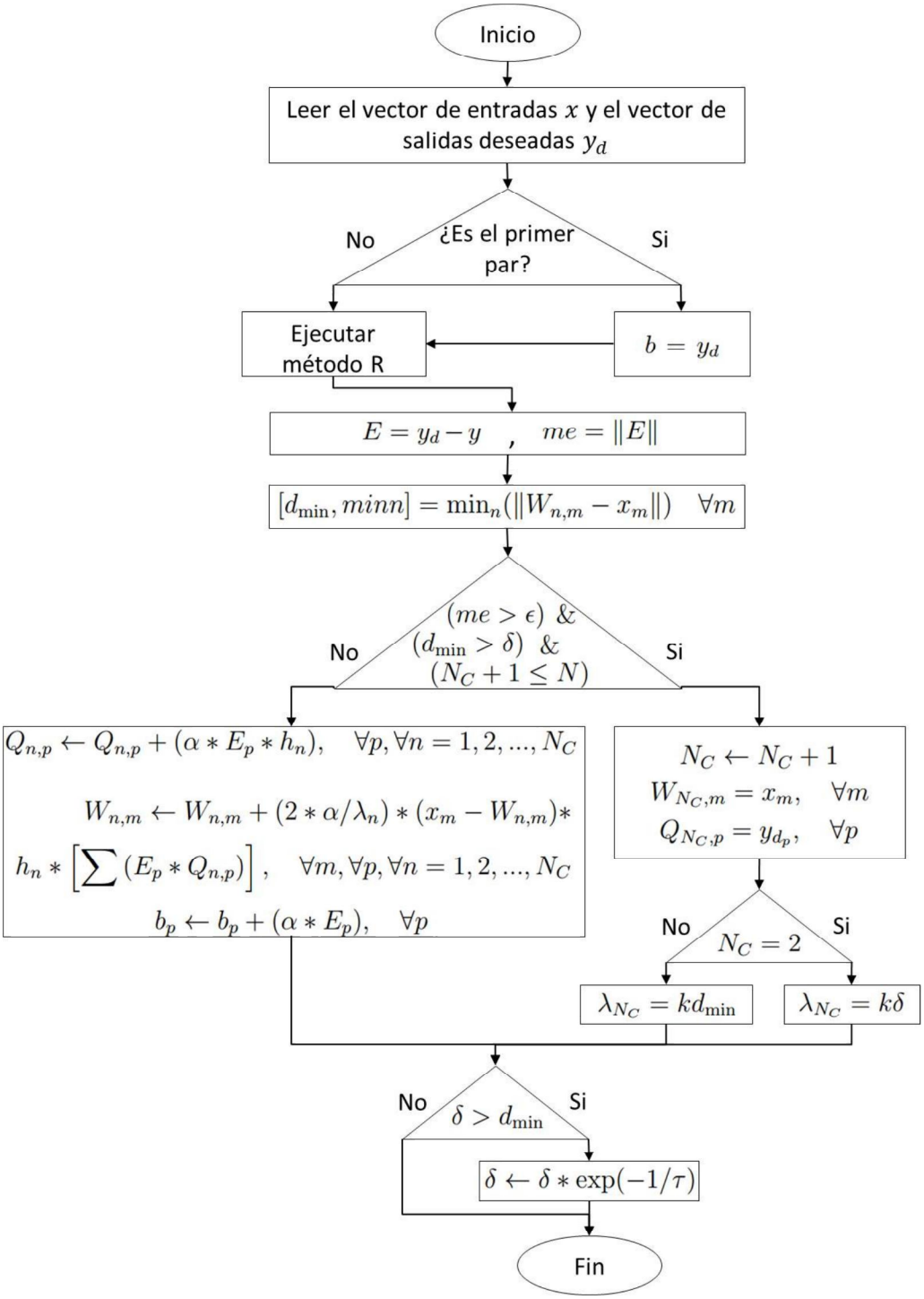


Figura 3.7: Diagrama de flujo del método L para RAN.

Análisis de complejidad

La complejidad de este método es $N_C(2M + P) + M + 2P$ cuando una nueva neurona es asignada y es $N_C(3M+3P)+2P$ cuando todas las neuronas son ajustadas. Así, el peor caso es la última expresión con $N_C = N$, por lo tanto, la complejidad para este método es lineal y es expresada en (3.41).

$$\mathrm{C_{RAN-L}} = N(3M + 3P) + 2P \tag{3.41}$$

Es decir: $N(M+P)$ operaciones son consumidas en el método R que se ejecuta dentro del método L, P operaciones en el cálculo del vector E y su magnitud me, NM operaciones en la ecuación (3.34), NP en la ecuación (3.38), $N(M + P)$ en la ecuación (3.39) y finalmente, P operaciones en la ecuación (3.40).

3.3.4 RAN, consumo de memoria

Considerando que cada elemento en cada variable del estado S_{RAN} ocupa una celda de la memoria de datos, entonces el consumo de memoria para RAN es (3.42).

$$\mathrm{MEM_{RAN}} = N(M + P + 2) + 2P + 10 \tag{3.42}$$

3.4 Comparativa teórica

La Tabla 3.1 resume la complejidad de los métodos R y L y el uso de memoria en cada arquitectura analizada.

	Complejidad del método R
RNMS	$N(M+P+2)$
OSELM	$N(M+P)$
RAN	$N(M+P)$
	Complejidad del método L
RNMS	$N(M+3)+M+P$
OSELM	$N^3+5N^2+(M+3P+4)N+2P$
RAN	$N(3M+3P)+2P$
	Consumo de memoria
RNMS	$N(M+P+2)+6$
OSELM	$2N^2+(M+2P+2)N+1$
RAN	$N(M+P+2)+2P+10$

Tabla 3.1: Comparación teórica de las redes neuronales analizadas (N: número de neuronas, M: número de entradas, P: número de salidas).

Vale la pena decir que estos análisis de memoria y complejidad representan una buena aproximación de como impactarán los algoritmos en la implementación del SE. Aunque estas expresiones no son el comportamiento absoluto, están bastante cerca. Además de los algoritmos, las habilidades de programación, los compiladores y la arquitectura de computación que ejecutará el programa, determinan el comportamiento exacto de dicho programa. Ejemplos: 1) un programador puede escribir un programa menor (que ocupe menos memoria) que uno escrito por otro programador, 2) escribir en lenguaje ensamblador produce código que corre mucho más rápido que el código producido por un lenguaje de alto nivel. De todos modos, estos algoritmos de RN son suficientemente diferentes para ser igualmente comparados si el mismo programador con el mismo compilador escribe los programas para correr en el mismo hardware. Más allá, una competencia puede ser creada con objetivo de lograr el mejor desempeño en una implementación de RNAUC-AL utilizando los mínimos recursos de hardware, aquí es donde la importancia de los algoritmos resalta.

Capítulo 4

Descripción experimental

En este trabajo, se llama «prueba» a una simulación o experimento. Se utiliza «simulación» para referirse a un sistema desconocido conectado a una RN donde ambos (sistema desconocido y RN) están ejecutándose o corriendo en una computadora. Se utiliza «experimento» al caso en que un sistema desconocido corriendo en una computadora es conectado a una RN corriendo en un SE.

En este capítulo primero es presentado, en la sección 4.1, el laboratorio creado para realizar pruebas. Luego, en las secciones 4.2 y 4.3, son presentados los modelos y diagramas relacionados con los casos de estudio y aplicación elegidos para probar el desempeño de las RN analizadas. Aquí cabe hacer ver que los casos de estudio y aplicación sí son conocidos con fin de controlar las pruebas y medir el desempeño, sin embargo, son desconocidos para las RN ya que los modelos de los casos elegidos no tienen ninguna influencia explícita o implícita en el diseño o ejecución de las RN.

Finalmente, en la sección 4.4, se presentan los pasos que se siguen en la ejecución de las pruebas.

4.1 Desarrollo de O2LAB, el laboratorio para ejecutar las pruebas

Para la ejecución de pruebas, tanto simulación como experimentación, es necesario un laboratorio con flexibilidad de: ejecutar una RN en la computadora o en

el SE, probar distintos casos de estudio, probar distintas arquitecturas de RN, capacidad de graficar y guardar resultados, entre otras características. Este laboratorio ha sido llamado **O2LAB (*Online Learning LABoratory*)**.

O2LAB está basado en Scilab® y Arduino®. Scilab® es un software de cómputo numérico [23] distribuido bajo la licencia CeCILL [24] a la fecha de publicación de este trabajo. La plataforma de SE Arduino ya ha sido comentada en el capítulo 2. Se emplea el modelo de tarjeta Arduino-MEGA-2560-R3® con la versión 1.0.5 del EDI Arduino® y la versión 5.4.1 de Scilab®. Todo el software es ejecutado en una computadora de 64 bits, procesador Intel-i7®, 6 GB de RAM y Sistema Operativo Windows 8®. Dadas las características multi-plataforma de Scilab® y Arduino®, O2LAB puede ser implementado en diferentes Sistemas Operativos (Windows®, Linux®, MacOS®) y utilizando diferentes SE de la familia Arduino®.

Ahora bien, a partir del esquema de aprendizaje en línea presentado en la figura 2.5 del capítulo 2 y la necesidad de pruebas simuladas (en Scilab®) y experimentales (en Arduino®), se tienen dos esquemas presentados en la siguiente figura.

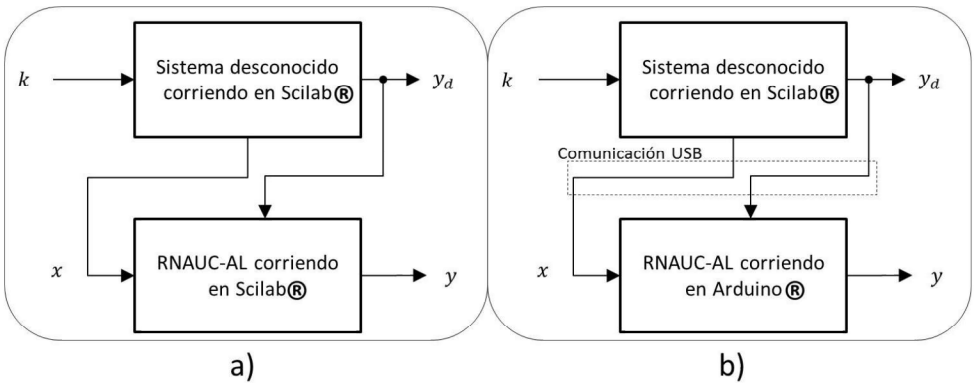


Figura 4.1: Esquemas de a) simulación y b) experimentación.

En la Figura 4.1, a cada paso de simulación k, Scilab® simula a un sistema desconocido para obtener el patrón x y salida deseada y_d necesarios para hacer prue-

bas sobre una RN. En el lado izquierdo de la figura se muestra a la RN simulada también en Scilab®. Del lado derecho de la figura se muestra a la RN corriendo dentro del SE Arduino® para lo cual requiere comunicarse vía USB con la computadora, particularmente con Scilab®, para recibir los vectores x, y_d, los comandos de métodos *Icmd*, *Rcmd*, *Lcmd*, y cualquier otro elemento necesario para las pruebas.

La Figura 4.2 muestra al diagrama general interno de O2LAB. Aquí, cada bloque está compuesto por uno o más archivos de código de Scilab® o de código de Arduino® según corresponda. Los bloques tipo «API» son archivos de interfaz entre diferentes SE, casos de estudio o RN según la API[1] en cuestión.

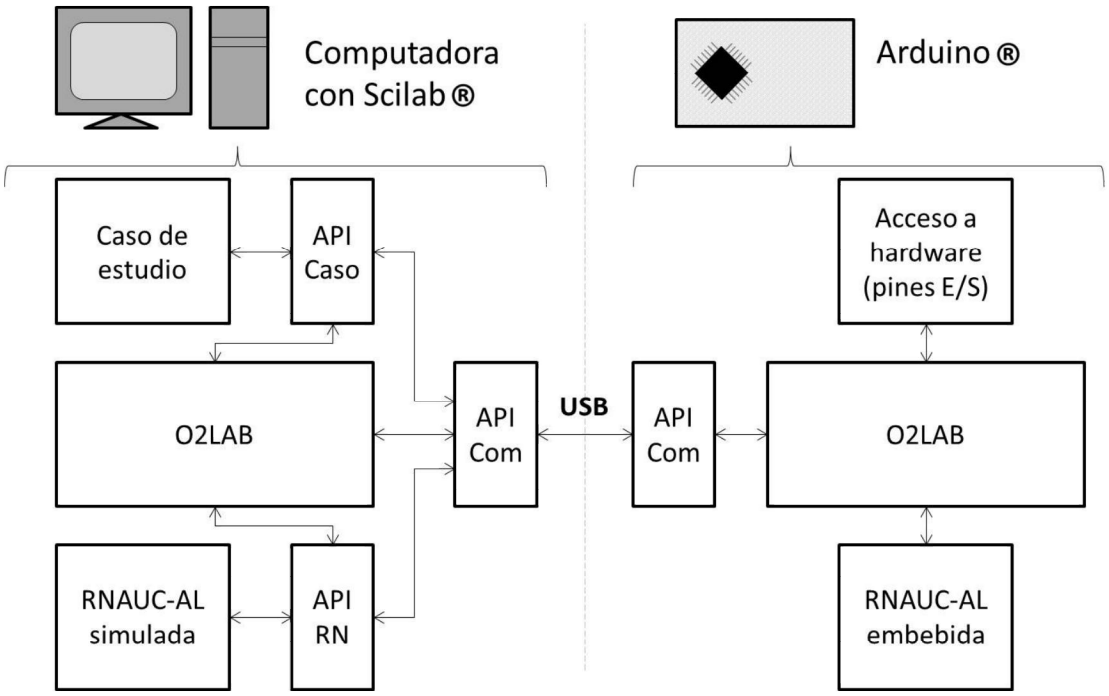


Figura 4.2: Diagrama general interno de O2LAB.

La Tabla 4.1 presenta una comparación de la tarjeta Arduino-MEGA-2560-R3®, utilizada en este trabajo, contra la Arduino-UNO-edición-SMD®. Puede verse que a menor cantidad de memoria RAM en el SE se tiene menos capacidad de manejo de neuronas. Cabe hacer ver que la capacidad de neuronas que se muestra en dicha tabla es para 10 entradas y 1 salida. Además, el tamaño de programa tiene añadido

[1]API: Application Programming Interface

el espacio que ocupan las funciones privadas relacionadas con O2LAB. Esto significa que la capacidad de neuronas será otra para diferente cantidad de entradas y salidas, y que el tamaño de programa puede ser menor si se eliminan las funciones de O2LAB dejando únicamente el código de la RN.

		MEGA	UNO
Características	SRAM (KB)	8	2
	FLASH (KB)	256	32
	MCU	ATmega 2560	ATmega 328
		Reloj: 16 MHz	Reloj: 16 MHz
		MCU de 8 bits	MCU de 8 bits
Máximo de neuronas	OSELM	22	8
	RNMS	138	28
	RAN	137	28
Tamaño de programa (KB)	OSELM	13.33	11.53
	RNMS	12.71	11.00
	RAN	13.05	11.35

Tabla 4.1: Caracteríticas de la implementación en Arduino®.

4.2 Casos de estudio: identificación de sistemas y predicción de series de tiempo

Se ejecutaron dos tipos de pruebas: identificación de sistemas y predicción de series de tiempo. La identificación de sistemas consiste en construir modelos de sistemas dinámicos a partir de datos medidos. Se puede realizar fuera de línea o en línea.

Hacer la identificación fuera de línea requiere tener todo el conjunto de datos para ser presentados al algoritmo que ejecuta la identificación. Por otro lado, hacer

la identificación en línea requiere que el algoritmo acepte pares de entrada-salida con los que ir refinando el modelo en cada presentación de par de entrada-salida.

Una serie de tiempo consiste en una secuencia de datos típicamente medidos a iguales intervalos de tiempo y existen como procesos comunes en muchas ramas de las ciencias. La predicción consiste en utilizar un modelo para determinar el valor de la serie en futuros instantes de tiempo a partir de valores en instantes pasados de tiempo.

La capacidad de mapeo entrada-salida de una RN permite, en teoría, que la misma actúe como identificador de sistemas o predictora de series de tiempo. La clave aquí es la forma en que la RN es utilizada en el aprendizaje. La RN puede aprender pares $\{x, y_d\}$ que le permitan, mediante su método R, identificar un sistema a partir del vector de entradas x el cual podría contener un determinado número de entradas del sistema que se quiere identificar. O bien, los pares $\{x, y_d\}$ utilizados en el aprendizaje le pueden permitir predecir una serie de tiempo a partir del vector de entradas x que podría contener valores pasados de la serie de tiempo que se desea predecir.

En la literatura se puede encontrar la identificación de dos procesos discretos. El Proceso 1 [25, 27] está modelado por la ecuación (4.1), mientras que el Proceso 2 [25] está modelado por la ecuación (4.2). Se toma como caso de estudio 1 al Proceso 1 y caso de estudio 2 al Proceso 2.

$$y(k+1) = \frac{y(k)y(k-1)[y(k)-0.5]}{1+y^2(k)+y^2(k-1)} + x(k) \tag{4.1}$$

$$y(k+1) = 0.3y(k) + 0.6y(k-1) + f(x(k)) \tag{4.2}$$

Estos dos procesos discretos tienen una señal de entrada específica, en (4.1), $x(k) = \sin(2\pi k/25)$ mientras que en (4.2) se tiene $f(x(k)) = 0.6\sin(\pi x(k)) + 0.3\sin(3\pi x(k)) + 0.1\sin(5\pi x(k))$ con su entrada $x(k) = \sin(2\pi k/200)$. Además,

ambos son considerados como sistemas causales, es decir, $y(k) = 0, \forall k < 0$. La condición inicial es $y(0) = 0$. La Figura 4.3 muestra las conexiones del Proceso 1 o 2 con la RN. Las señales que se ilustran más cerca del bloque de la RN son para denotar la abstracción de las señales del proceso conectado hacia las entradas, salidas y salida deseada de una RN.

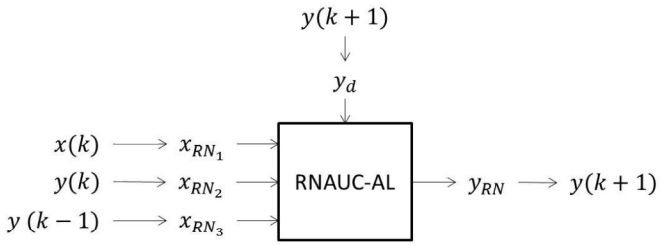


Figura 4.3: Diagrama de conexiones del Proceso 1 y 2 con una RNAUC-AL.

El tercer caso de estudio corresponde a la predicción de la serie de tiempo caótica Mackey-Glass [26, 27] modelada por la ecuación (4.3).

$$\frac{dy(t)}{dt} = \frac{ay(t-\tau)}{1+y^{10}(t-\tau)} - by(t) \tag{4.3}$$

En Mackey-Glass se establece $a = 0.2$, $b = 0.1$ y $\tau = 17$. Se emplea el método de Runge-Kutta de 4^{to} orden con paso de 0.1 para obtener la respuesta a la ecuación diferencial (4.3) a instantes enteros de tiempo. Igualmente a los Procesos 1 y 2, se utiliza la consideración de sistema causal, $y(k) = 0, \forall k < 0$. Y la condición inicial $y(0) = 1.2$ es utilizada para lograr caos. La Figura 4.4 muestra las conexiones de Mackey-Glass con la RN.

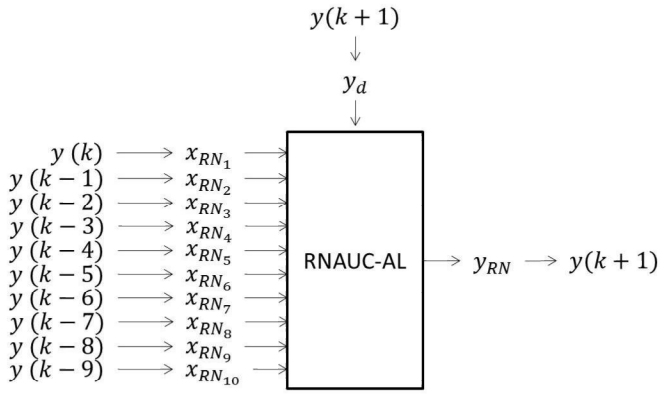


Figura 4.4: Diagrama de conexiones de Mackey-Glass con una RNAUC-AL.

La Tabla 4.2 resume las configuraciones de E/S aplicada a una RN en cada caso de estudio. Los Procesos 1 y 2 tienen tres entradas y una salida, mientras que Mackey-Glass tiene diez entradas y una salida. Los tres casos evolucionan a cada paso de tiempo k el cual es considerado como un intervalo constante de tiempo.

	RN x	RN y_d
Proceso 1	$x(k)$, $y(k)$, $y(k-1)$	$y(k+1)$
Proceso 2	$x(k)$, $y(k)$, $y(k-1)$	$y(k+1)$
Mackey-Glass	$y(k)$, $y(k-1)$, ..., $y(k-9)$	$y(k+1)$

Tabla 4.2: Especificaciones de E/S de los casos de estudio.

Los detalles de la metodología para ejecutar las pruebas son explicados en la sección 4.4, sin embargo, aquí cabe hacer ver solamente los tiempos de aprendizaje y prueba. En el caso del Proceso 1 y 2 se desea que la RN aprenda a identificar el sistema. En el caso de Mackey-Glass se desea que la RN aprenda a predecir la serie de tiempo. A continuación, para definir los tiempos de aprendizaje y prueba, se aborda primero el caso del Proceso 1 y 2, luego el caso de Mackey-Glass.

En el caso del Proceso 1 y 2, durante cierto tiempo K (k-mayúscula), se solicita a la RN que ante las entradas $x(k)$, $y(k)$ y $y(k-1)$ aprenda a identificar la señal de

salida deseada $y(k+1)$. Luego de dicho tiempo de aprendizaje, se solicita a la RN que desempeñe por ella misma la identificación del proceso en cuestión durante un intervalo de k siguiente. Es decir, durante el intervalo de aprendizaje la RN aprende a identificar un proceso, luego, durante un intervalo siguiente, la RN identifica al proceso utilizando su conocimiento recién adquirido. Este último intervalo es lo que usualmente se llama «prueba de generalización» en el ámbito de RN. La Tabla 4.3 muestra los intervalos elegidos para realizar las pruebas en ambos procesos.

	$K_{\text{aprendizaje}}$	$K_{\text{identificacion}}$
Proceso 1	$k = [1, 80]$	$k = [81, 100]$
Proceso 2	$k = [1, 320]$	$k = [321, 400]$

Tabla 4.3: Intervalos de aprendizaje e identificación del Proceso 1 y 2.

El caso de Mackey-Glass es parecido al del Proceso 1 y 2. La diferencia es que al intervalo de tiempo que viene después del aprendizaje es llamado intervalo de predicción. La Tabla 4.4 muestra los intervalos elegidos para realizar las pruebas en Mackey-Glass.

	$K_{\text{aprendizaje}}$	$K_{\text{prediccion}}$
Mackey-Glass	$k = [201, 3200]$	$k = [5001, 5500]$

Tabla 4.4: Intervalo de aprendizaje y predicción de Mackey-Glass.

Los intervalos de aprendizaje y prueba aquí presentados podrían ser otros, sin embargo, los dados para el Proceso 1 y 2 logran mostrar suficiente rango de la señal y los intervalos de Mackey-Glass, al no iniciar desde el primer instante de tiempo y estar separados temporalmente, permiten probar la predicción en otra dinámica de la serie caótica. En la sección 4.4 se presenta la metodología de pruebas paso a paso y qué es lo que ocurre en los intervalos mencionados.

4.3 Caso de aplicación: suplemento en falla de sensores

Como ha sido comentado, la capacidad de mapeo entrada-salida de una RN le permite un elevado nivel de abstracción y por lo tanto tener una gran variedad de aplicaciones. Este trabajo trata con una aplicación en particular. Primero, en esta sección, se describe como utilizar una RN para suplementar la señal de un sensor cuando éste falla. Luego, se describe una aplicación biomédica en la que una RN sustituye la señal de un sensor de glucosa cuando éste falla.

4.3.1 Suplemento en falla de sensores

Identificar la falla de un sensor es de vital importancia en muchas aplicaciones. Así mismo, es altamente deseado tener un sensor adicional o, en su defecto, algún mecanismo para poder proporcionar una señal suplementaria cuando el sensor falle. Por lo tanto, se propone una forma de utilizar una RN para suplir a un sensor en una situación como esta. Primero se describen los elementos necesarios para implementar esta solución, luego se presenta un diagrama de conexiones y, finalmente, se describe el diagrama.

Elementos necesarios:

1. Acceso al sensor.

2. Un banco de retrasos (memorias) para almacenar la señal del sensor en instantes pasados de tiempo.

3. Alguna forma de identificar la falla del sensor.

4. Los instantes de tiempo k son de intervalo constante.

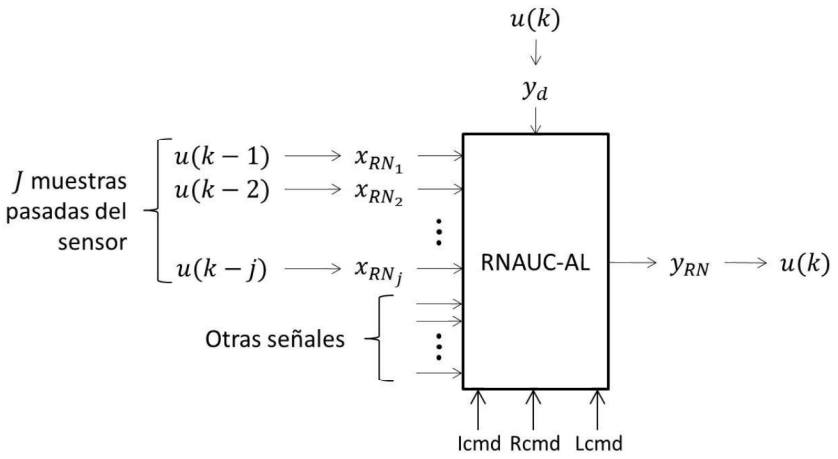


Figura 4.5: Diagrama de una RNAUC-AL como suplemento en falla de un sensor.

La Figura 4.5 muestra el diagrama de conexiones. La forma de resolver esta aplicación es conectar la señal $u(k)$ del sensor a un banco de retrasos para disponer de J muestras pasadas. Mientras el sensor funcione correctamente, la RN aprende la operación del sensor utilizando el comando *Lcmd* , las J muestras pasadas del sensor y, posiblemente, algunas otras señales del entorno que sean convenientes. Cuando el sensor falla, la RN debe proporcionar una señal suplementaria utilizando el comando *Rcmd* y las señales de entrada utilizadas en el aprendizaje.

4.3.2 Suplemento en falla de un sensor de glucosa

Un estudio reciente para la identificación de un modelo entrada-salida de un paciente con Diabetes Mellitus Tipo 1 (T1DM) en régimen de terapia intensiva, es decir, monitoreo continuo de glucosa e infusión continua subcutánea de insulina mediante un control robusto ha sido presentado [28]. Como caso de aplicación se presenta a una RN utilizada para suplementar la señal de un sensor de glucosa en caso de falla.

La Figura 4.6 presenta el diagrama de conexiones basado en el concepto presentado en la subsección anterior. Si la muestra actual de glucosa $glu(k)$ proveniente del sensor no da valor, entonces la RN proporciona un valor utilizando el valor actual

de insulina $ins(k)$, tres valores pasados de glucosa ($glu(k-1)$, $glu(k-2)$, $glu(k-3)$) y el método R. Si la muestra actual de glucosa proveniente del sensor sí da valor, entonces la RN ejecuta el método L para seguir asimilando conocimiento de como emular al sensor cuando éste falle. La muestra actual de insulina y tres muestras pasadas de glucosa muestran buen desempeño en los resultados del capítulo 5.

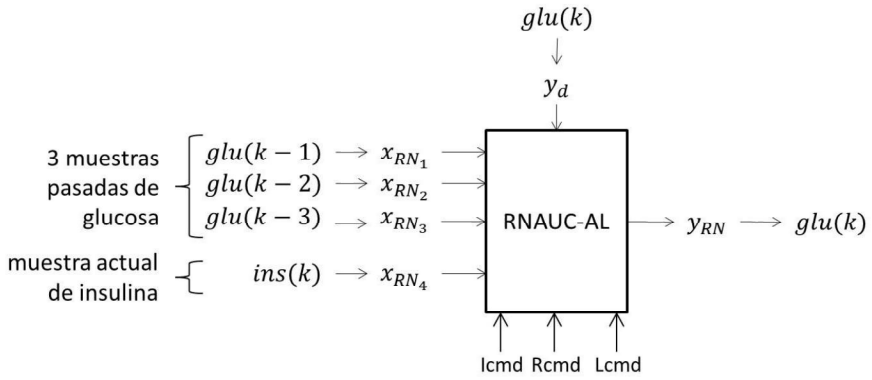


Figura 4.6: Diagrama de una RNAUC-AL como suplemento en falla de un sensor de glucosa.

Para la implementación de las pruebas en este caso de aplicación se utilizan 4489 registros arrojados durante 9 días por el CGM (*Guardian® REAL-time Continuous Glucose Monitoring Systems* por MiniMed Inc.) [28]. Los registros fueron divididos en dos tablas que han sido proporcionadas para realizar estos experimentos. Cada tabla tiene un vector de valores del sensor de glucosa tomados cada 5 minutos. También se tiene un vector de valores de insulina tomados en los mismos instantes que los valores de glucosa. Los elementos de cada vector corresponden en el tiempo, es decir, el elemento 1 de glucosa es tomado al mismo instante que el elemento 1 de insulina, y así para el elemento 2, 3, ..., hasta la longitud del vector (2030 en la tabla 1 y 2459 en la tabla 2). Los resultados para estas pruebas, en el capítulo 5, son para cada una de las dos tablas.

4.4 Metodología para la ejecucicón de pruebas

Los tres casos de estudio y el caso de aplicación evolucionan con el tiempo k. La metodología que se sigue para los tres casos de estudio es:

1. Lanzar el comando *Icmd* para inicializar la RN.

2. En el caso del Proceso 1 y 2 se presenta su propia señal de entrada $x(k)$ para obtener un par $\{x, y_d\}$. En el caso de Mackey-Glass es independiente de una entrada, por lo que se obtiene directamente un par $\{x, y_d\}$. Para recordar la composición de un par $\{x, y_d\}$ en cada caso ir a la tabla 4.2.

3. Durante un intervalo de tiempo $K_{\text{aprendizaje}}$ especificado en las tablas 4.3 y 4.4 se ejecuta aprendizaje para la RN utilizando el comando *Lcmd*.

4. Una vez terminados de dar los ejemplos de aprendizaje, se dan nuevamente esos patrones y se ejecuta el comando *Rcmd* en cada uno para medir el desempeño de aprendizaje. Esta medición es mediante el **Error Cuadrático Medio (ECM)** (4.4).

5. Luego, para medir el desempeño de generalización, se dan algunos patrones que vienen después del aprendizaje (en la secuencia de k) y mediante el comando *Rcmd* y el ECM (4.4) se ejecuta la medición. Este intervalo de generalización es $K_{\text{identificacion}}$ para el Proceso 1 y 2, y es $K_{\text{prediccion}}$ para Mackey-Glass, ambos pueden verse en las tablas 4.3 y 4.4.

$$\text{ECM} = \sqrt{\frac{1}{K}\sum_{k=1}^{K}(y_{d_k} - y_k)^2} \tag{4.4}$$

La metodología para el caso de aplicación consite en:

1. Lanzar el comando *Icmd* para inicializar la RN. Iniciar en $k = 1$.

2. Leer el registro k-ésimo de una tabla.

3. Si el valor de glucosa en este registro es diferente de cero (el sensor funciona correctamente), ejecutar aprendizaje de este ejemplo mediante el comando *Lcmd*. Además, ejecutar el método R mediante el comando *Rcmd* solo para medir el desempeño de aprendizaje mediante ECM (4.4). Para recordar cuáles son las señales de entrada, salida y salida deseada de esta aplicación ver la figura 4.6.

4. Si el valor de glucosa en este registro es igual a cero (el sensor falla), ejecutar el

método R mediante el comando *Rcmd* para que la RN proporcione la señal suplementaria. En este caso no es posible medir el desempeño de generalización debido a que no sabemos precisamente cual es el valor correcto de glucosa (ya que el sensor falló).

5. Establecer $k \leftarrow k + 1$ e ir al paso 2. Este lazo se repite hasta que se agotan los registros de la tabla.

Además, en las pruebas de los cuatro casos se mide el tiempo promedio de ejecución de los métodos R y L y se dan como t_R y t_L respectivamente. La cantidad de neuronas empleada en cada caso y cada RN es revisada y dada como $N_{C_{\max}}$.

Comúnmente cuando se trabaja con RN, los elementos en x y y_d son normalizados desde su rango natural hacia otro rango. En el caso de la RNMS es declarado que trabaja con el rango $[0, 1] \subset \mathbb{R}$, así que se utiliza este mismo rango para OSELM y RAN para todos los elementos en x y y_d. Esto significa que la salida de la RN y, debe ser desnormalizada desde el rango $[0, 1] \subset \mathbb{R}$ hacia el rango natural original para ver los valores reales.

Finalmente, en el caso de los experimentos con el SE, cada entrada y salida en x y y_d son digitalizadas con 8 bits, y así, la salida de la RN es pasada de digital a continua utilizando 8 bits también.

Capítulo 5

Resultados

En este capítulo se presentan los resultados de las pruebas. En la sección 5.1 se presentan los resultados de las pruebas para los tres casos de estudio, primero mediante simulaciones, luego con experimentos. En la sección 5.2 se presentan los resultados para el caso de aplicación experimentando con la RNMS. Se dan las tablas que muestran los parámetros específicos de cada RN que han sido empleados. Los resultados comprenden gráficas de respuesta y las tablas con los resultados numéricos: ECM de aprendizaje y generalización, tiempo promedio del método R y L, y cantidad de neuronas empleadas. Las gráficas de respuesta tienen en el eje vertical al comportamiento de la RN contra el deseado ambos sin dimensiones, y en el eje horizontal se tiene cada paso de simulación k.

5.1 Resultados de los casos de estudio

5.1.1 Simulaciones con los casos de estudio

Las Tablas 5.1, 5.2 y 5.3 presentan los parámetros utilizados en cada RN para ejecutar las simulaciones. Se encontró mejor desempeño para caso de neurona aditiva-sigmoide de OSELM que con neurona RBF; razón por la cual se ha decidido utilizar solamente la aditiva-sigmoide y efectuar una comparación más clara contra las demás RN. El resto de las RN (RNMS y RAN) utilizan los parámetros que dan mejor desempeño.

	N	Tipo de neurona
Proceso 1	40	aditiva-sigmoide
Proceso 2	50	aditiva-sigmoide
Mackey-Glass	60	aditiva-sigmoide

Tabla 5.1: Parámetros de OSELM utilizados en las simulaciones de los casos de estudio.

	N	λ	ms
Proceso 1	40	0.1	0.95
Proceso 2	100	0.2	0.95
Mackey-Glass	120	0.1	0.95

Tabla 5.2: Parámetros de RNMS utilizados en las simulaciones de los casos de estudio.

	N	k	$\delta_{\min}$	$\delta_{\max}$	τ	α	ϵ
Proceso 1	40	0.87	0.07	0.7	5	0.02	0.01
Proceso 2	80	0.87	0.07	0.7	100	0.02	0.02
Mackey-Glass	80	0.87	0.07	0.7	500	0.02	0.01

Tabla 5.3: Parámetros de RAN utilizados en las simulaciones de los casos de estudio.

Las Figuras 5.1, 5.2 y 5.3 muestran la gráfica de simulación de aprendizaje de cada RN para el Proceso 1, 2 y Mackey-Glass respectivamente. Luego, las Figuras 5.4, 5.5 y 5.6 muestran la gráfica de simulación de generalización de cada RN, es decir, identificación del sistema para el Proceso 1 y 2, y predicción del sistema para Mackey-Glass, respectivamente. En las seis figuras puede verse que el eje horizonal (del tiempo) no corresponde exactamente con los intervalos presentados en las tablas 4.3 y 4.4 del capítulo 4. Esto es por motivos de la implementación de simulaciones, sin embargo, dichos intervalos dados en las tablas sí se siguen en la ejecución de las pruebas.

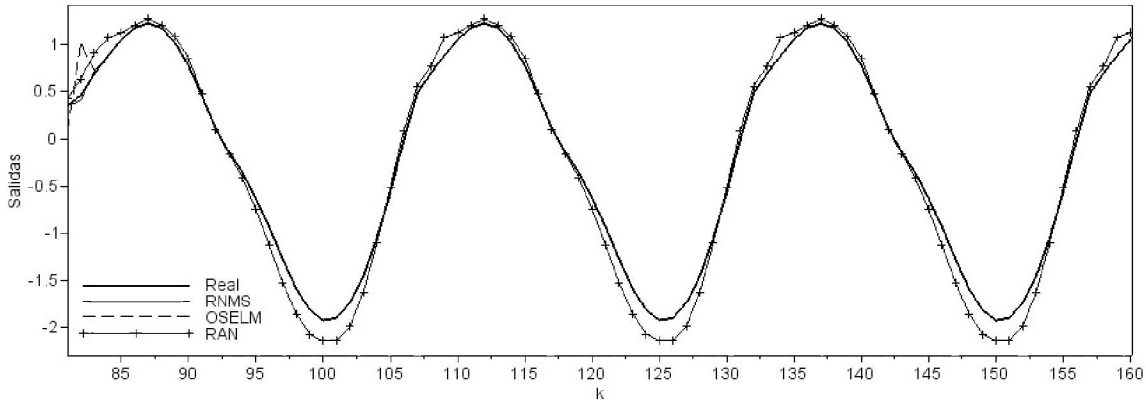


Figura 5.1: Simulación de aprendizaje para el Proceso 1.

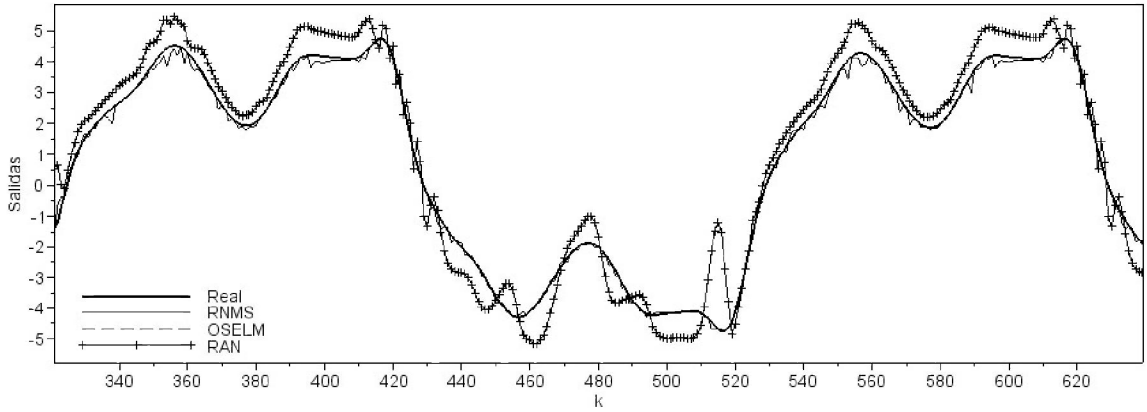


Figura 5.2: Simulación de aprendizaje para el Proceso 2.

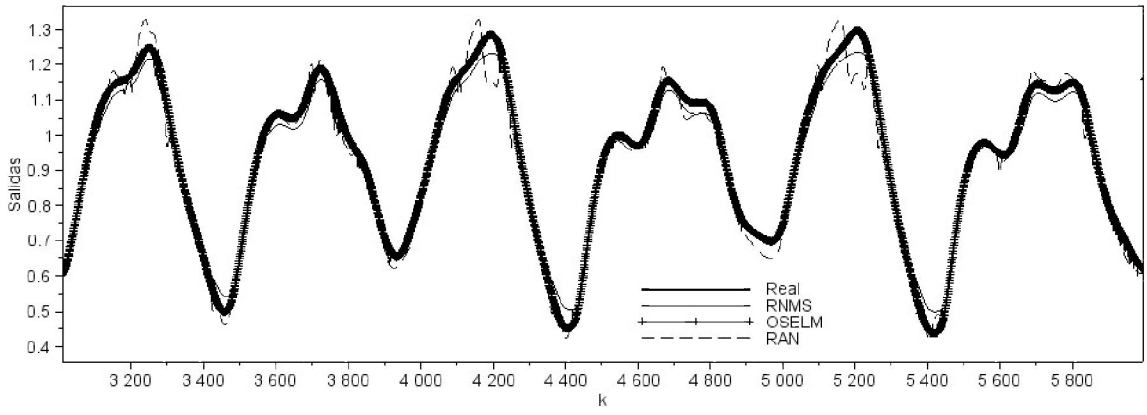


Figura 5.3: Simulación de aprendizaje para Mackey-Glass.

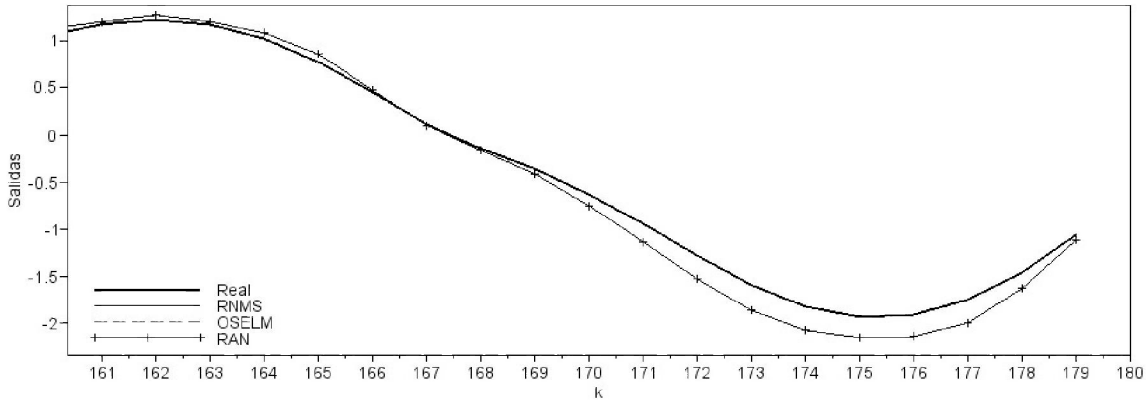


Figura 5.4: Simulación de identificación para el Proceso 1.

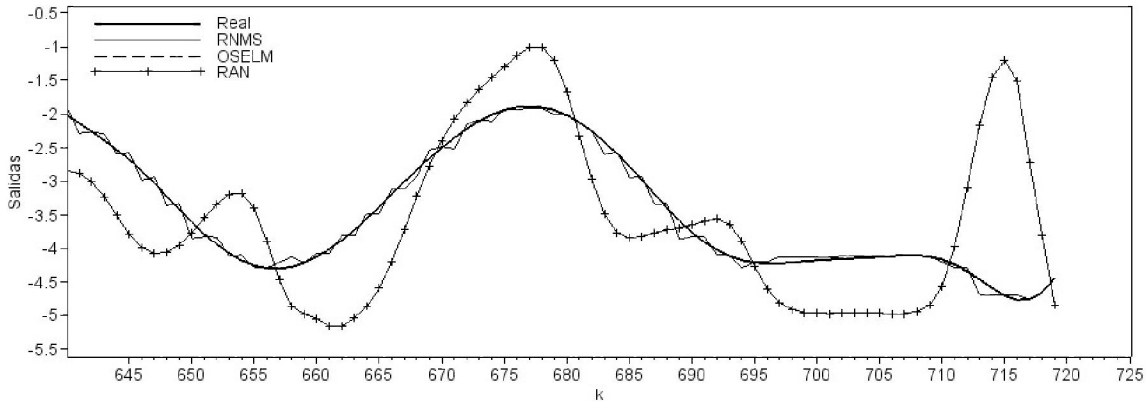


Figura 5.5: Simulación de identificación para el Proceso 2.

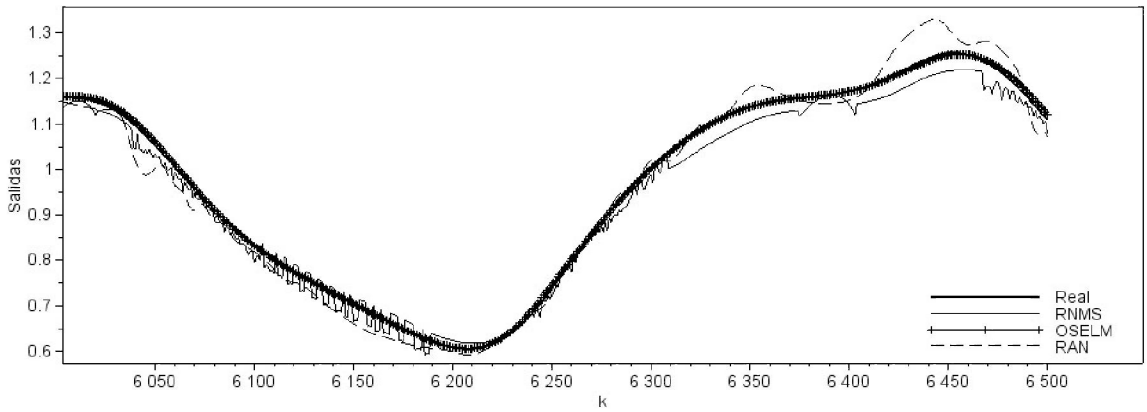


Figura 5.6: Simulación de predicción para Mackey-Glass.

Las Tablas 5.4, 5.5 y 5.6 muestran los resultados de simulación para el Proceso 1, 2 y Mackey-Glass. En estas tablas, $N_{C_{\max}}$ indica la cantidad de neuronas empleadas, ECM_A y ECM_G son el error cuadrático medio de aprendizaje y generalización (identificación o predicción según corresponda) respectivamente, y t_L y t_R indican el tiempo promedio de ejecución del método L y R respectivamente, ambos en milisegundos.

	$N_{C_{\max}}$	ECM_A	ECM_G	t_L	t_R
RNMS	30	0.007	0.000	9.98	12.01
OSELM	40	0.070	0.004	5.34	1.94
RAN	26	0.148	0.153	16.19	10.61

Tabla 5.4: Resultados de simulación para el Proceso 1.

	$N_{C_{\max}}$	ECM_A	ECM_G	t_L	t_R
RNMS	93	0.147	0.091	23.23	35.01
OSELM	50	0.007	0.006	2.27	2.20
RAN	53	0.767	1.035	15.74	22.06

Tabla 5.5: Resultados de simulación para el Proceso 2.

	$N_{C_{\max}}$	ECM_A	ECM_G	t_L	t_R
RNMS	120	0.028	0.027	40.19	45.94
OSELM	60	0.004	0.005	2.66	2.46
RAN	53	0.038	0.032	25.31	20.93

Tabla 5.6: Resultados de simulación para Mackey-Glass.

5.1.2 Experimentos con los casos de estudio

No se tuvieron problemas en la implementación de RNMS y RAN en Arduino®, sin embargo, OSELM tiene problemas en implementar su *fase boosting* debido al

cálculo PISVD (cálculo de pseudo-inversa basado en SVD), la cual es una operación mucho más compleja que una multiplicación matricial. Así que únicamente OSELM ha sido implementado parcialmente, esto es, la *fase boosting* de OSELM es calculada en Scilab®, y luego Arduino® trabaja autónomamente con la *fase secuencial* de OSELM. RNMS y RAN sí se encuentran implementados completamente para que Arduino® trabaje 100 % autónomo como es deseado.

Las Tablas 5.7, 5.8 y 5.9 presentan los parámetros utilizados en cada RN para ejecutar las simulaciones. Por la misma razón que la comentada en simulaciones de la subsección 5.1.1, se ha utilizado únicamente la neurona aditiva-sigmoide para OSELM, mientras que las demás RN utilizan los parámetros que dan mejor desempeño en ellas.

	N	Tipo de neurona
Proceso 1	22	aditiva-sigmoide
Proceso 2	22	aditiva-sigmoide
Mackey-Glass	22	aditiva-sigmoide

Tabla 5.7: Parámetros de OSELM utilizados en los experimentos de los casos de estudio.

	N	λ	ms
Proceso 1	40	0.1	0.95
Proceso 2	100	0.2	0.95
Mackey-Glass	120	0.1	0.95

Tabla 5.8: Parámetros de RNMS utilizados en los experimentos de los casos de estudio.

	N	k	$\delta_{\min}$	$\delta_{\max}$	τ	α	ϵ
Proceso 1	40	0.77	0.07	0.7	0.1	0.02	0.05
Proceso 2	80	0.77	0.01	0.7	50	0.02	0.05
Mackey-Glass	80	0.87	0.07	0.7	500	0.02	0.02

Tabla 5.9: Parámetros de RAN utilizados en los experimentos de los casos de estudio.

Las Figuras 5.7, 5.8 y 5.9 muestran la gráfica de experimento de aprendizaje de cada RN para el Proceso 1, 2 y Mackey-Glass respectivamente. Luego, las Figuras 5.10, 5.11 y 5.12 muestran la gráfica de experimento de generalización de cada RN, es decir, identificación del sistema para el Proceso 1 y 2, y predicción del sistema para Mackey-Glass, respectivamente. Igualmente, como en el caso de las simulaciones, el eje horizonal (del tiempo) no corresponde exactamente con los intervalos presentados en las tablas 4.3 y 4.4 del capítulo 4 por la misma razon dada en la subsección anterior.

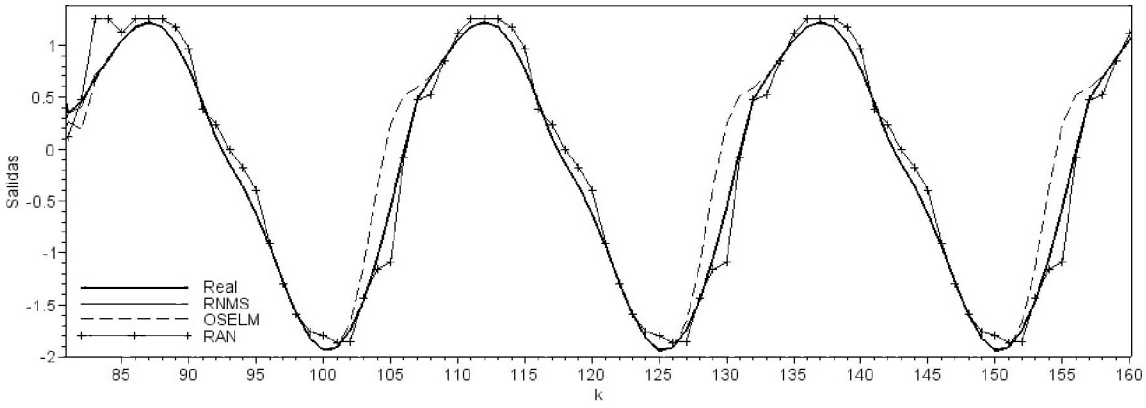


Figura 5.7: Experimento de aprendizaje para el Proceso 1.

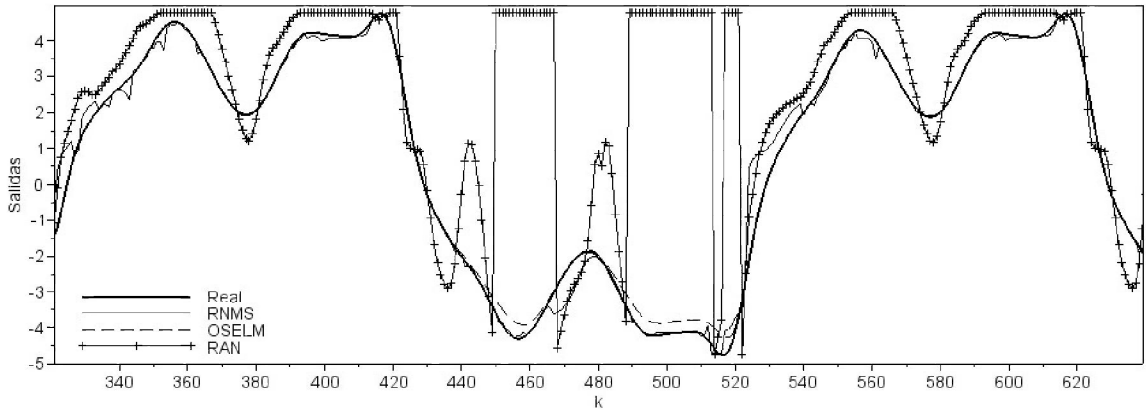


Figura 5.8: Experimento de aprendizaje para el Proceso 2.

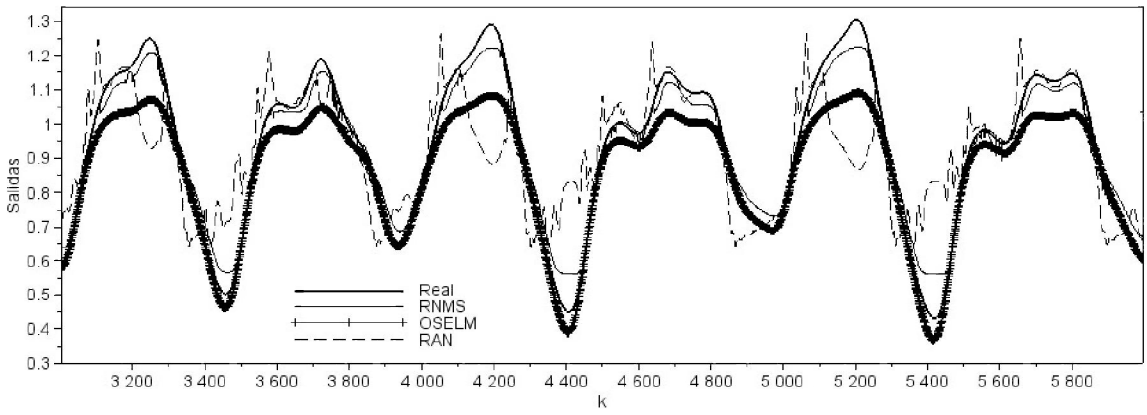


Figura 5.9: Experimento de aprendizaje para Mackey-Glass.

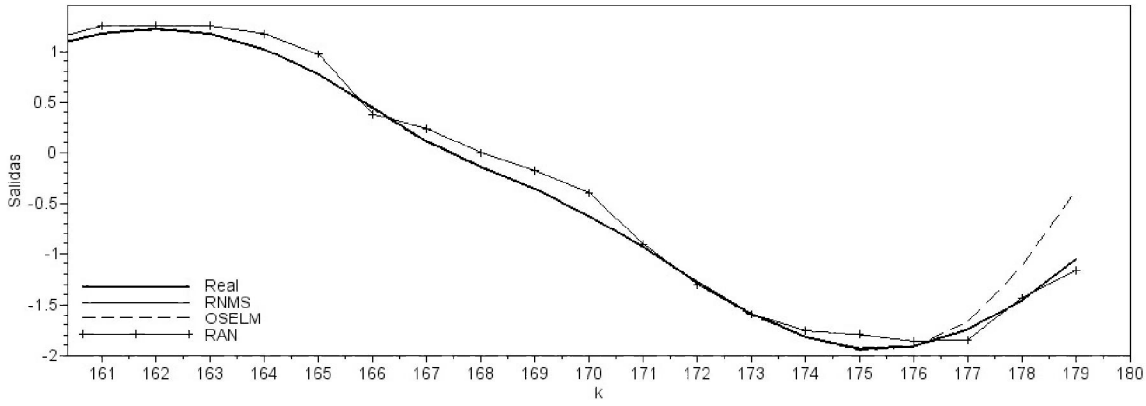


Figura 5.10: Experimento de identificación para el Proceso 1.

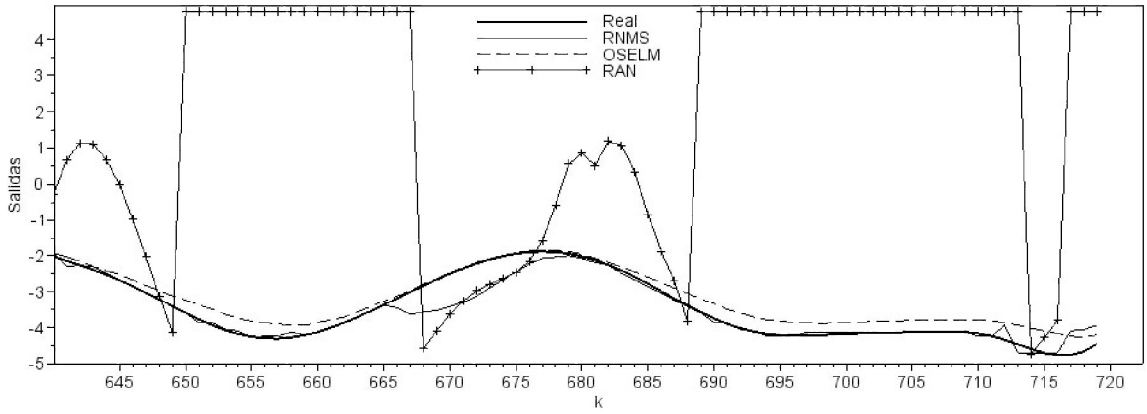


Figura 5.11: Experimento de identificación para el Proceso 2.

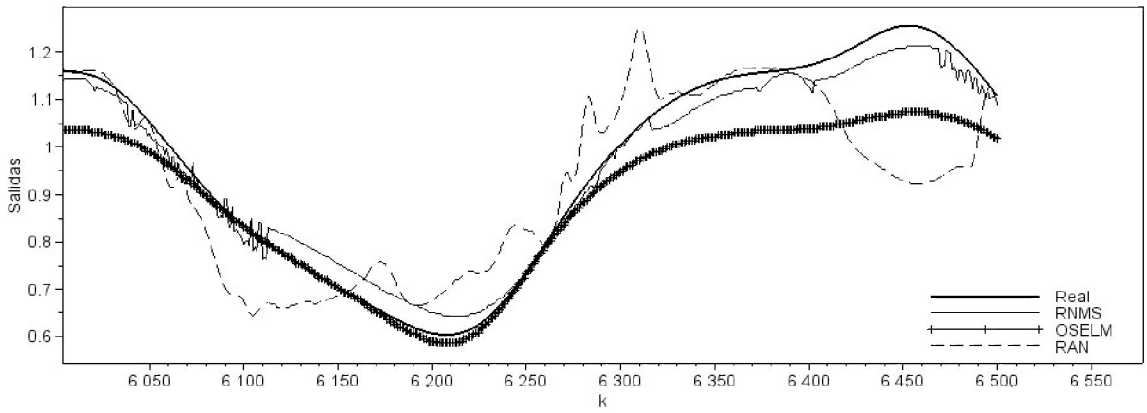


Figura 5.12: Experimento de predicción para Mackey-Glass.

Las Tablas 5.10, 5.11 y 5.12 muestran los resultados de experimentación para el Proceso 1, 2 y Mackey-Glass. Igual que en las simulaciones, $N_{C_{\max}}$ indica la cantidad de neuronas empleadas, ECM_A y ECM_G son el error cuadrático medio de aprendizaje y generalización (identificación o predicción según corresponda) respectivamente, y t_L y t_R indican el tiempo promedio de ejecución del método L y R respectivamente, ambos en milisegundos.

	$N_{C_{\max}}$	ECM_A	ECM_G	t_L	t_R
RNMS	40	0.008	0.003	7.69	9.08
OSELM	22	0.244	0.171	180.21	5.94
RAN	35	0.168	0.113	11.97	7.47

Tabla 5.10: Resultados de experimentación para el Proceso 1.

	$N_{C_{\max}}$	ECM_A	ECM_G	t_L	t_R
RNMS	100	0.367	0.302	26.46	33.53
OSELM	22	0.153	0.301	180.21	5.90
RAN	25	3.553	6.815	9.79	7.62

Tabla 5.11: Resultados de experimentación para el Proceso 2.

	$N_{C_{\max}}$	ECM_A	ECM_G	t_L	t_R
RNMS	120	0.047	0.051	54.87	58.99
OSELM	22	0.083	0.092	183.03	8.73
RAN	80	0.138	0.118	63.80	32.91

Tabla 5.12: Resultados de experimentación para Mackey-Glass.

5.2 Resultados del caso de aplicación

En los experimentos presentados en la subsección 5.1.2 se demuestra que la RNMS es quien tiene mejor desempeño. Por lo tanto, se elige para utilizarse en los experimentos de suplemento para falla en sensor de glucosa.

La Tabla 5.13 presenta los parámetros utilizados en la RNMS para ejecutar los experimentos para ambas tablas que se tienen de registros del sensor de glucosa.

N	λ	ms
130	0.2	0.95

Tabla 5.13: Parámetros de RNMS utilizados en los experimentos del caso de aplicación.

La gráfica del desempeño ante la tabla de registros 1 es mostrada en las figuras 5.13, 5.14 y 5.15. La Figura 5.13 muestra el experimento completo, mientras que las Figuras 5.14 y 5.15 muestran un acercamiento al mismo en diferentes ventanas de tiempo con fin de apreciar mejor la operación de este caso de aplicación.

Así mismo, la gráfica del desempeño ante la tabla de registros 2 es mostrada en las figuras 5.16, 5.17 y 5.18. La Figura 5.16 muestra el experimento completo, mientras que las Figuras 5.17 y 5.18 muestran un acercamiento al mismo en diferentes ventanas de tiempo.

En todas estas figuras, las marcas (+) son los valores proporcionados por el sensor y las marcas (□) son los valores proporcionados por la RNMS cuando el sensor falla. Así, la línea continua representa la señal generada por el sensor y la RNMS. Esta «señal continua» es la que utilizaría, por ejemplo, un sistema de control de glucosa.

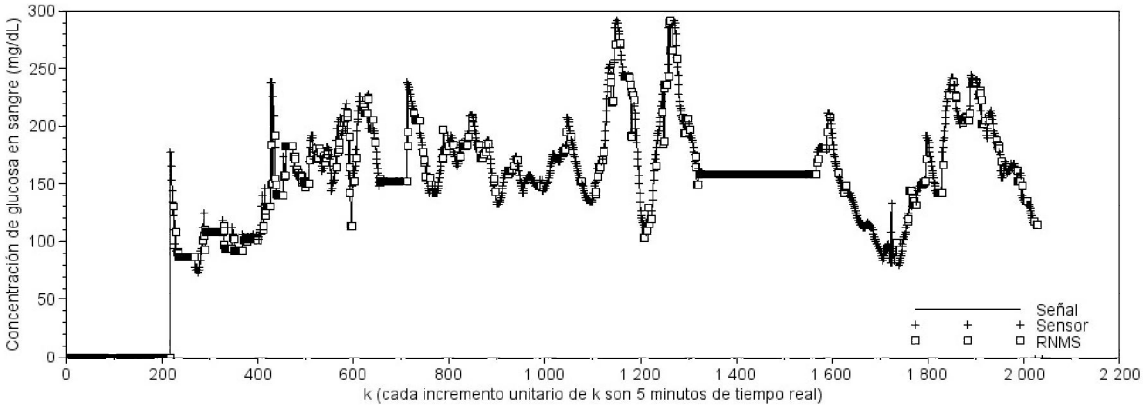


Figura 5.13: Experimento completo del caso de aplicación utilizando la RNMS ante la tabla de registros 1.

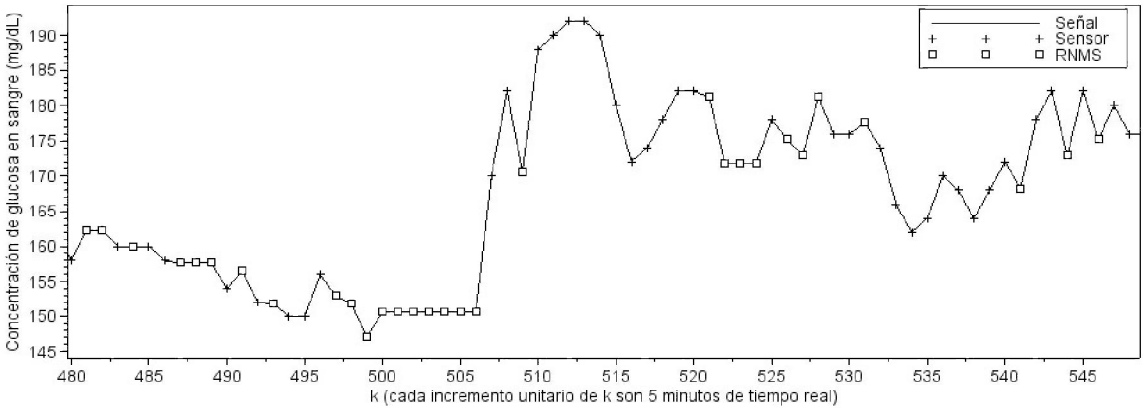


Figura 5.14: Acercamiento a la ventana [480-545] del experimento del caso de aplicación utilizando la RNMS ante la tabla de registros 1.

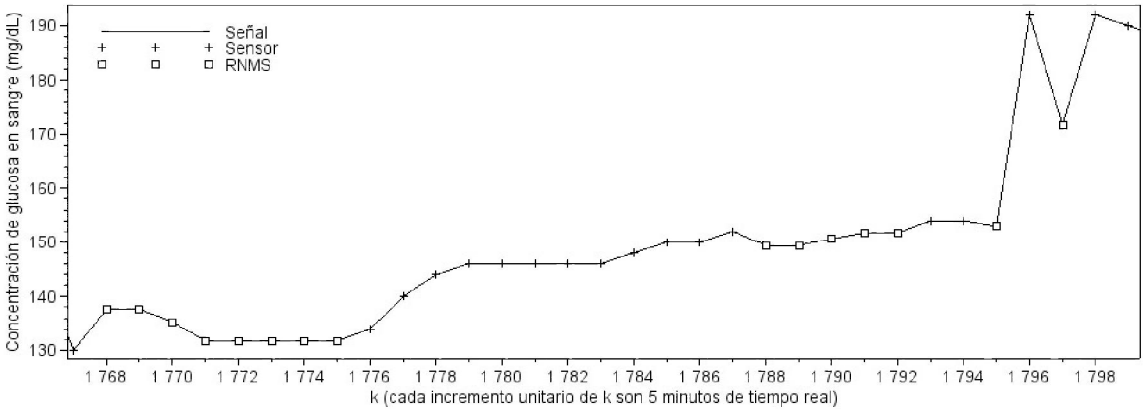


Figura 5.15: Acercamiento a la ventana [1768-1798] del experimento del caso de aplicación utilizando la RNMS ante la tabla de registros 1.

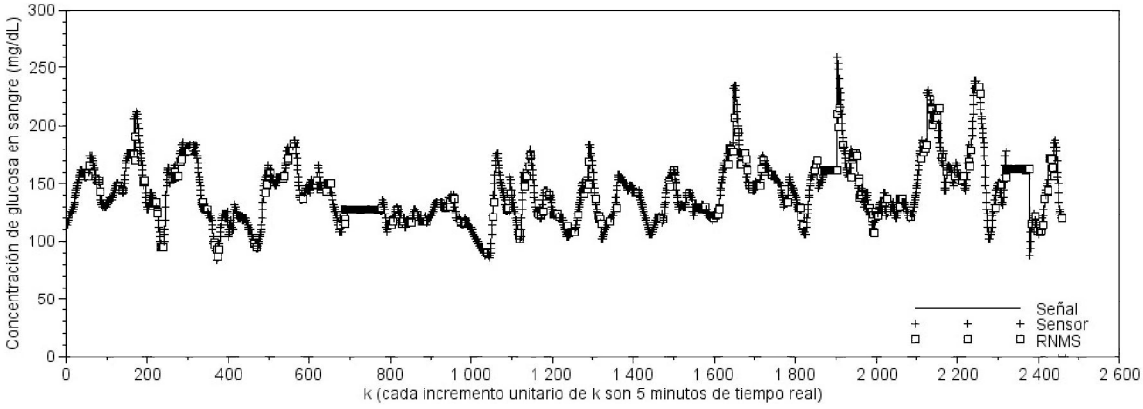


Figura 5.16: Experimento completo del caso de aplicación utilizando la RNMS ante la tabla de registros 2.

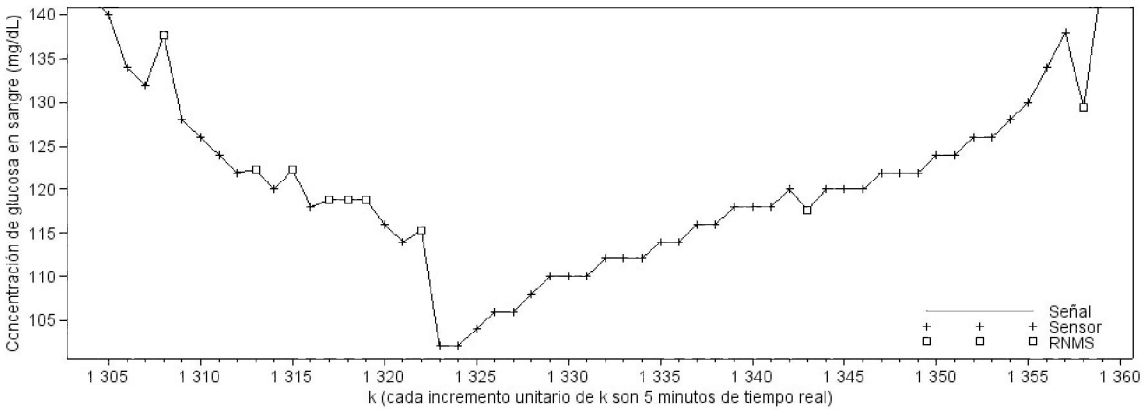


Figura 5.17: Acercamiento a la ventana [1305-1360] del experimento del caso de aplicación utilizando la RNMS ante la tabla de registros 2.

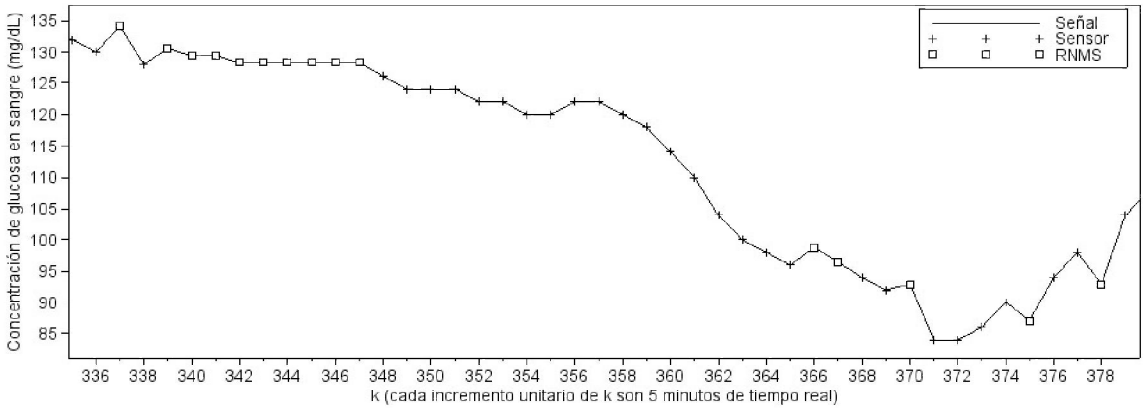


Figura 5.18: Acercamiento a la ventana [336-378] del experimento del caso de aplicación utilizando la RNMS ante la tabla de registros 2.

La Tabla 5.14 muestra los resultados de estos experimentos. Igual que en las pruebas anteriores, $N_{C_{\max}}$ indica la cantidad de neuronas empleadas, t_L y t_R indican el tiempo promedio de ejecución del método L y R respectivamente, ambos en milisegundos. $\mathrm{ECM_A}$ es el error de aprendizaje donde se mide que tan bien la RNMS aprende del sensor cuando éste funciona correctamente. Finalmente, dado que no es posible conocer la señal correcta del sensor cuando éste falla, entonces resulta imposible calcular el error cuadrático medio de generalización ($\mathrm{ECM_G}$), es decir, que tan bien sustituye la RNMS al sensor; por ello nos limitamos a establecer únicamente el error de aprendizaje ($\mathrm{ECM_A}$) y hacer una inspección visual a las gráficas para ver que «tan dentro» está de lo tolerable la señal sustituta por parte de la RNMS.

	$N_{C_{\max}}$	$\mathrm{ECM_A}$	t_L	t_R
Tabla de registros 1	48	0.03971	11.57	16.05
Tabla de registros 2	37	0.03769	8.46	12.22

Tabla 5.14: Resultados de experimentación de RNMS para el caso de aplicación.

Capítulo 6

Conclusiones y trabajo abierto

En este proyecto de investigación se ha propuesto un enfoque de sistemas embebidos para el estudio de redes neuronales. Este enfoque comprende principalmente tres partes: 1) Analizar la complejidad computacional de la red neuronal, considerando las operaciones matemáticas con ciclos inherentes como sumatoria, producto matricial, operador norma, etc., 2) Analizar el consumo de memoria de la red neuronal, contando las celdas de memoria empleadas en cada variable del estado de una red neuronal, 3) Llevar a cabo simulaciones y experimentos para probar y medir el desempeño de las redes neuronales ante varios casos de estudio.

Para las pruebas, es desarrollado el laboratorio O2LAB compuesto por el software Scilab® y la plataforma de sistemas embebidos Arduino®. Esta plataforma consiste de tarjetas electrónicas basadas en microcontrolador de 8 bits y un lenguaje basado en C/C++ para su programación.

Los resultados muestran que la RNMS cumple adecuadamente la hipótesis:

- Tiene buen desempeño.
- Es de rápida ejecución, destacando a su método L aún para una cantidad relativamente grande de neuronas.
- Se puede embeber completamente.

La única desventaja que se reporta para la RNMS es que requiere del ajuste de dos parámetros (λ y ms) y consume más neuronas que las otras arquitecturas.

Por otro lado, OSELM tiene buen desempeño y no requiere parámetros, excepto por N, sin embargo no se ha podido embeber completamente en un sistema embebido como el utilizado debido a que la operación PISVD no pudo ser programada en la memoria del MCU y tuvo que ser ejecutada en una computadora convencional. Si PISVD se programara en la memoria del MCU, OSELM estaría 100 % embebido. Además, su método L es muy lento aún para pocas neuronas y el consumo de memoria es cuadrático respecto al número de neuronas.

En cuanto a la RAN, ésta no tiene un desempeño tan bueno como las demás y presenta algunas diferencias entre las simulaciones y experimentos. Además, requiere del ajuste de seis parámetros (τ, $\delta_{\min}$, $\delta_{\max}$, ϵ, α y k) y su método L es relativamente lento cuando se tiene relativamente muchas neuronas.

Finalmente, cabe decir que este trabajo ha presentado un estado del arte, definiciones y procedimientos que permiten continuar con una línea de investigación y desarrollo de sistemas inteligentes embebidos. Dentro de los temas que pueden ampliarse y mejorarse están:

- Análisis y estudio de otras arquitecturas de redes neuronales.
- Diseñar nuevas arquitecturas y algoritmos de redes neuronales.
- Considerar otras áreas de los sistemas inteligentes como sistemas difusos y cómputo evolutivo.
- El laboratorio O2LAB puede ser mejorado y ampliado en capacidades.
- Evaluar nuevos casos de estudio en simulación.
- Proponer y probar nuevas aplicaciones reales.

Bibliografía

[1] Haykin Simon. *Neural Networks, A comprehensive foundation.* Pearson Prentice Hall, 2 edition, 1999.

[2] Galeano Gustavo. *Programacion de Sistemas Embebidos en C, teoria y practicas aplicadas a cualquier microcontrolador.* Alfaomega, 2009.

[3] Xiaoru Song; Yu Shang. Research and implementation of embedded intelligent home system. In *Third International Symposium on Intelligent Information Technology Application*, pages 82–85, 2009.

[4] Esmaeilzadeh H.; Saeedi P.; Araabi B.N.; Lucas C.; Sied Mehdi Fakhraie. Neural network stream processing core (nnsp) for embedded systems. In *IEEE International Symposium on Circuits and Systems*, pages 2773–2776, 2006.

[5] Pealoza U.C.; Esquer J.; Rios B. A methodology for implementation of the execution phase of artificial neural networks in digital hardware devices. In *Congreso de Electronica, Robotica y Mecanica Automotriz*, pages 422–427, 2008.

[6] Skoda P.; Lipic T.; Srp A.; Rogina B.M.; Skala K.; Vajda F. Implementation framework for artificial neural networks on fpga. In *Proceedings of the 34th International Convention on Information and Communication Technology, Electronics and Microelectronics*, pages 274–278, 2011.

[7] Cotton N.J.; Wilamowski B.M.; Dundar G. A neural network implementation on an inexpensive eight bit microcontroller. In *International Conference on Intelligent Engineering Systems*, pages 109–114, 2008.

[8] Cotton N.J.; Wilamowski B. Compensation of nonlinearities using neural networks implemented on inexpensive microcontrollers. *IEEE Transactions on Industrial Electronics*, 58:733–740, 2011.

[9] Malcangi M. Smart recognition and synthesis of emotional speech for embedded systems with natural user interfaces. In *The 2011 International Joint Conference on Neural Networks (IJCNN)*, pages 867–871, 2011.

[10] Bao Jian; Chen Yu; Yu JinShou. Neural networks with limited precision weights and its application in embedded systems. In *Second International Workshop on Education Technology and Computer Science (ETCS)*, pages 86–91, 2010.

[11] Rich Elaine; Knight Kevin. *Artificial Intelligence*. McGraw Hill, 2 edition, 1991.

[12] P. Engelbrecht Andries. *Computational Intelligence, An Introduction*. John Wiley and Sons, Ltd, 2002.

[13] Malinowski Aleksander; Yu Hao. Comparison of embedded system design for industrial applications. *IEEE Transactions on Industrial Informatics*, 7:244–254, 2011.

[14] Banzi Massimo; Cuartielles David; et. al. Arduino official web site, Jun 2013. http://www.arduino.cc.

[15] IEEE Computer Society. *IEEE standard for floating-point arithmetic*, august 2008.

[16] Johnsonbaugh Richard. *Discrete Mathematics*. Prentice Hall, 4 edition, 1999.

[17] Torres T. Luis Martin. Controladores dinamicos con la red neuronal de maxima sensibilidad. Master's thesis, Universidad Autonoma de San Luis Potosi, San Luis Potosi, Mexico, 1998.

[18] Perez Pedro; Zambrano Patricia Escamilla Indira; Torres T. Luis Martin. A comparison between back propagation and the maximum sensibility neural net-

work to surface roughness prediction in machining of titanium (ti 6al 4v) alloy. In *MICAI 2008: Advances in Artificial Intelligence*, pages 1009–1019, 2008.

[19] Liang Nan-Ying; Huang Guang-Bin; Saratchandran P.; et al. A fast and accurate online sequential learning algorithm for feedforward networks. *IEEE Transactions on Neural Networks*, 17:1411–1423, 2006.

[20] Huang Guang-Bin; Zhu Qin-Yu; Siew Chee-Kheong. Extreme learning machine: Theory and applications. *Neurocomputing*, 70:489–501, 2006.

[21] Huang Guang-Bin; Wang Dian Hui; Lan Yuan. Extreme learning machines: a survey. *International Journal of Machine Learning and Cybernetics*, 2:107–122, 2011.

[22] Platt John. A resource-allocating network for function interpolation. *Neural Computation*, 3(2):213–225, 1991.

[23] Scilab Enterprises. Scilab official web site, Jun 2013. http://www.scilab.org.

[24] Scilab Enterprises. Scilab's licence web site, Jun 2013. http://www.scilab.org/scilab/license.

[25] de Jesus Rubio J. Sofmls: Online self-organizing fuzzy modified least-squares network. *IEEE Transactions on Fuzzy Systems*, 17:1296–1309, 2009.

[26] Kasabov NK; Song Q. Denfis: Dynamic evolving neural-fuzzy inference system and its application for time-series prediction. *IEEE Transactions on Fuzzy Systems*, 10:144–154, 2002.

[27] Rong HJ; Sundararajan N; Huang GB; et al. Sequential adaptive fuzzy inference system (safis) for nonlinear system identification and prediction. *Fuzzy Sets and Systems*, 157:1260–1275, 2006.

[28] Quiroz G.; Flores-Gutierrez C.P.; Femat R. Robust stabilizing control oriented on identified patient-specific t1dm model. In 8^{th} *IFAC Symposium on Biological and Medical Systems*, volume 8, pages 67–72, 2012.