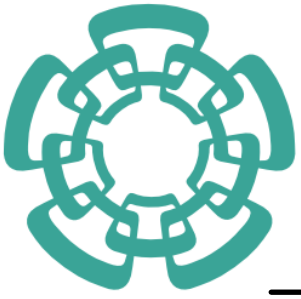


CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL INSTITUTO POLITÉCNICO NACIONAL

UNIDAD ZACATENCO
DEPARTAMENTO DE CONTROL AUTOMÁTICO

IDENTIFICACIÓN Y CONTROL DE SISTEMAS NO LINEALES UTILIZANDO EL ALGORITMO DE APRENDIZAJE DE LEVENBERG–MARQUARDT PARA REDES NEURONALES COMPLEJAS RECURRENTES

TESIS QUE PRESENTA
Edmundo Pérez Reynaud

PARA OBTENER EL GRADO DE
Maestro en Ciencias

EN LA ESPECIALIDAD DE
Control Automático

DIRECTOR DE TESIS
Dr. Ieroham Solomon Baruch

MÉXICO, D.F. OCTUBRE DEL 2015

Agradecimientos

A Dios, por iluminarme y construir mi camino, pues no estaría aquí sin Él. “Señor, tu nos das los dones, pero nos pides a cambio la fatiga.”

A CONACyT, por el apoyo económico otorgado durante los dos años de maestría (número de apoyo: 364197, número de becario: 295620), sin el cual no se podría haber realizado este trabajo y mi formación como Maestro en Ciencias.

Al Departamento de Control Automático del CINVESTAV-IPN, por todos los conocimientos, experiencia y formación que me han dado para ser un mejor profesionista, y a los Doctores y colegas con los cuales tuve la oportunidad de convivir a lo largo de mis estudios de maestría.

Al Doctor Ieroham Solomon Baruch, por la paciencia, la orientación, el tiempo y los conocimientos, con los cuales este trabajo no sería posible.

Al Doctor Moisés Bonilla Estrada y al Doctor Carlos Román Mariaca Gaspar, por el tiempo invertido, sus correcciones y comentarios que han hecho de este un mejor trabajo.

A mi familia, mis padres y hermanos, por apoyarme en cada una de las decisiones en mi carrera, por alentarme a ser mejor como persona y como profesionista, el cariño incondicional y la educación que me han inculcado.

A mis amigos, viejos y nuevos, por los buenos momentos, las pláticas, los consejos, las experiencias compartidas y las enseñanzas sobre la vida que me han dado. A todos y cada uno de ellos.

A Emilio Carrillo, por acompañarme a lo largo de todos estos años, su apoyo, consejos y darle sentido a mi vida.

A Jorge Castañeda, por tener en él un amigo en el cual siempre puedo contar.

Resumen

En este trabajo se desarrolla el algoritmo de optimización de Levenberg–Marquardt recursivo para el dominio de los números complejos, a partir de una modificación del algoritmo de Error de Predicción Recursivo. Se introduce el concepto de optimización de la función de error con el algoritmo de primer orden de gradiente descendiente y la obtención de los términos de gradiente por medio del método diagramático, que puede ser utilizado para cualquier topología de red neuronal para construir su red adjunta. Posteriormente, se presenta una topología de Red Neuronal Recurrente con pesos sinápticos complejos, y una función de activación construida para eliminar puntos de singularidad. Se presenta la versión del algoritmo de Levenberg–Marquardt recursivo en el dominio complejo para esta topología, la cual será utilizada a lo largo de este trabajo.

Se introducen los problemas de identificación y control adaptable de sistemas dinámicos, junto con la descripción matemática de dos plantas mecánicas, no lineales y oscilatorias, una de una entrada y una salida, y la otra de múltiples entradas y múltiples salidas. El problema de identificación es resuelto con el uso de una Red Neuronal Compleja Recurrente entrenada con el algoritmo de aprendizaje desarrollado, dentro de un esquema de aproximación del comportamiento dinámico por corrección del error.

Para el problema de control adaptable, dos esquemas de control neuronal son propuestos, donde se utilizan tres Redes Neuronales Complejas Recurrentes entrenadas con el algoritmo de aprendizaje desarrollado: la primera red identifica el comportamiento de la planta no lineal, la segunda funciona como un controlador por realimentación de estado, y la tercera funciona como controlador de tipo *feedforward*. El primer esquema de control se conforma por estas tres redes neuronales, mientras que el segundo esquema de control agrega un término de acción integral del error de control para eliminar posibles errores de seguimiento.

El desempeño del identificador, así como de los esquemas de control, aplicados a ambas plantas propuestas, son medidos por medio del Error Medio Cuadrático total. Luego, los resultados obtenidos utilizando el algoritmo de entrenamiento de Levenberg-Marquardt recursivo son comparados con los obtenidos al utilizar el algoritmo de *Backpropagation* complejo, desarrollado en trabajos anteriores. Estos resultados presentan un buen desempeño de los esquemas de identificación y control adaptable al ser utilizado el algoritmo de entrenamiento desarrollado a lo largo del trabajo.

Abstract

In this work, we develop the recursive Levenberg–Marquardt optimization algorithm in the domain of complex numbers, from a modification of the Recursive Prediction of the Error algorithm. We introduce the concept of optimization of an error function through the first order gradient descent algorithm, and the derivation of the gradient terms by means of a diagrammatic method, which can be applied to any neural network topology to construct its adjoint network. Subsequently, we present a Recurrent Neural Network topology with complex–valued synaptic weights and a constructed activation function that eliminates any singularities that could be present in its domain. Then the version in the complex domain of the recursive Levenberg–Marquardt algorithm for this topology is presented, which will be used throughout this work.

We introduce the problems of identification and adaptive control of dynamical systems, along with the mathematical description of two nonlinear, oscillatory mechanical plants, one is a single–input single–output, and the other is a multiple–input multiple–output plant. The identification problem is solved with the use of one Complex–Valued Recurrent Neural Network trained with the developed training algorithm, in an approximation of dynamic behavior scheme by means of error correction.

For the problem of adaptive control, we propose two neural control schemes, where we use three Complex–Valued Recurrent Neural Networks trained with the developed training algorithm: the first network identifies the nonlinear plant behavior, the second one works as a state feedback controller, and the third one works as a feedforward controller. The first control scheme is made of these three neural networks, while the second control scheme adds an integral action term that depends on the control error, which eliminates possible tracking errors.

The identifier and control schemes performance, applied to both proposed plants, are measured with the total Mean Squared Error. Then, the results obtained using the recursive Levenberg–Marquardt training algorithm are compared with the ones obtained using the complex Backpropagation algorithm, developed in past works. These results present a good performance of the identification and adaptive control schemes when the training algorithm developed in this work is applied.

Índice General

Índice de Figuras

Índice de Tablas

1. Introducción

El uso de redes neuronales artificiales en el contexto de los sistemas dinámicos ha aumentado en los últimos años, ya que estas presentan rápida adaptación, capacidad de aproximación y capacidad de integración masiva y en paralelo [1], [2].

En 1990, Narendra y Parthasarathy publicaron el artículo [3], el cual se considera como el primer trabajo de redes neuronales aplicadas a sistemas dinámicos, donde se muestra la utilidad de las redes neuronales para la identificación y control de sistemas, tanto lineales como no lineales, con redes de alimentación hacia adelante y redes recurrentes. Este trabajo abre las puertas al control utilizando redes neuronales artificiales.

En [4], los autores utilizan redes neuronales para la identificación, predicción y control de sistemas, proponiendo su uso en diferentes plantas mecánicas y presentando buenos resultados.

En [5], los autores han implementado topologías que utilizan lógica difusa, creando multi–modelos neuronales aplicados al control de sistemas dinámicos con resultados satisfactorios.

En la actualidad, el uso de redes neuronales artificiales es fundamental en el campo del control adaptable debido a su capacidad de aproximar funciones lineales y no lineales, y a la capacidad de procesar cualquier tipo de información, no importando la naturaleza de la planta a controlar.

Por otro lado, las aplicaciones de redes neuronales en el dominio complejo aún son muy escazas, y su desarrollo está reducido únicamente al análisis de sus arquitecturas, funciones de activación y algunos cuantos algoritmos de entrenamiento.

En 1971, Aizenberg, Ivaskiv y Pospelov [6] presentaron una extensión de las redes neuronales artificiales ya existentes, utilizando funciones de activación con valores en el plano complejo.

En 1975, Widrow [7] presentó el algoritmo de mínimos cuadrados en su versión compleja, en una aplicación de acondicionamiento de señales para telecomunicaciones.

Posteriormente en 1991, Leung y Haykin [8] derivaron el algoritmo de *Backpropagation* complejo, el cual es utilizado para el entrenamiento de una red neuronal compleja aplicada en una tarea de clasificación.

Aun así, el uso de redes neuronales artificiales en el dominio complejo para tareas de control es muy escaso, y la mayoría de aplicaciones existentes están limitadas a sistemas oscilatorios o que por su naturaleza quedan mejor representados por números complejos. Tal es el caso de fenómenos electromagnéticos, ópticos, sistemas electrónicos y de información o sistemas de potencia [9].

En [10], se propone una red neuronal compleja para la identificación de un evaporador industrial. En [11], se utiliza un tipo especial de red neuronal compleja para modelar el comportamiento de un transformador de potencia. En [12], una red neuronal compleja de gran dimensión es utilizada para controlar el efector final de un manipulador robótico de tres grados de libertad. En [13], los autores utilizan redes neuronales complejas para la identificación de circuitos eléctricos analógicos. En [14], se utilizan como un predictor de comportamiento para un generador eólico. En [15], se propone el uso de redes neuronales complejas para el diseño de filtros digitales de tipo FIR. Y en [16], se utilizan para el modelado de un sistema dinámico de Lorenz presentando un buen desempeño.

Todos estos ejemplos muestran la versatilidad del uso de redes neuronales artificiales en el dominio complejo, y la diversidad de tipo de sistemas a los cuales se pueden aplicar.

1.1 Motivación del Trabajo de Tesis

¿Por qué las redes neuronales artificiales en el dominio complejo? El campo de los números complejos $\mathbb{C}$ presenta mayores propiedades algebraicas que el plano real $\mathbb{R}^2$, razón por la cual no es lo mismo hacer una red neuronal en el dominio complejo, a una red neuronal en el dominio real con el doble de parámetros. La información contenida en fenómenos oscilatorios queda mejor representada por medio de números complejos.

En el contexto de sistemas dinámicos, un sistema con parámetros reales presentará un comportamiento en forma de sumas de términos únicamente exponenciales. Mientras que para sistemas con parámetros complejos, su comportamiento también tendrá términos oscilatorios, lo cual presenta una mayor riqueza de información codificada en su dinámica.

Es por esto que surge el interés de introducir las redes neuronales utilizadas en la identificación y control de sistemas dinámicos al dominio de los números complejos, puesto que nos dan una mayor riqueza de dinámicas con las cuales poder aproximar el comportamiento de estos.

¿Por qué el algoritmo de Levenberg–Marquardt recursivo? El algoritmo de optimización de Levenberg–Marquardt surge como una extensión de los algoritmos de segundo orden basados en el método de Newton y el cálculo de la matriz Hessiana. Presenta una convergencia rápida de los parámetros que ajusta respecto a algoritmos de primer orden, ya que utiliza una aproximación de la información de segundo orden con la que no cuentan métodos como el de gradiente descendiente.

Por otro lado, el algoritmo de Levenberg–Marquardt presenta un menor costo computacional respecto a los algoritmos de optimización de segundo orden; puesto que estos, por lo general, necesitan el cálculo e inversión de matrices que se vuelven más grandes conforme aumenta el número de parámetros ajustables.

La versión recursiva del algoritmo asegura que, para cada iteración, el ajuste de los parámetros seguirá un comportamiento dinámico, disminuyendo el tiempo de convergencia hasta llegar a la solución óptima.

En general, el algoritmo recursivo de Levenberg–Marquardt se encuentra en un punto intermedio entre los algoritmos de optimización de primer y segundo orden: presenta un costo computacional semejante a los primeros, pero con la rapidez de convergencia cercana a los últimos. Estas propiedades hacen de este algoritmo una herramienta poderosa en la implementación en tiempo real de redes neuronales artificiales para la identificación y control de sistemas dinámicos.

1.2 Planteamiento del Problema y Objetivos

Trabajar con redes neuronales en el dominio complejo presenta mayor dificultad que el caso del dominio real, debido a que hay que tener mayores consideraciones para elegir su arquitectura, funciones de activación, y el desarrollo de los algoritmos de entrenamiento. También se está en desventaja respecto a la bibliografía disponible, puesto que esta es escaza para redes neuronales complejas.

Por otro lado, el uso de algoritmos de entrenamiento de mayor orden presenta una mejora en la etapa de aprendizaje y en el desempeño de las redes neuronales, lo cual las hace una mejor solución en el estudio de sistemas dinámicos para su implementación en tiempo real. Esto claro, tomando en cuenta el aumento en costo computacional al introducir algoritmos de entrenamiento de mayor complejidad.

Este trabajo busca desarrollar el algoritmo de entrenamiento de Levenberg–Marquardt recursivo en el dominio complejo. Se sigue la metodología presentada en [17] para el desarrollo del algoritmo de entrenamiento, y en [18] para la obtención de los términos de gradiente, basada en el método diagramático presentado en [19].

También se presenta una topología de Red Neuronal Compleja Recurrente, con una función de activación compleja construida, a la cual se le aplica el algoritmo de entrenamiento obtenido. Esta rede se aplica al problema de identificación de sistemas dinámicos y también como una solución en esquemas de control neuronal adaptable.

Posteriormente, los resultados obtenidos en los esquemas de identificación y control, con las redes neuronales complejas entrenadas con el algoritmo de Levenberg–Marquardt recursivo complejo, son comparados de forma cuantitativa con los obtenidos en [18], donde las redes neuronales son entrenadas con el algoritmo de *Backpropagation* complejo.

1.3 Organización de la Tesis

Este trabajo de tesis se divide en ocho capítulos. El capítulo uno presenta el trasfondo histórico del uso de las redes neuronales en los sistemas de control adaptable y las redes neuronales en el dominio complejo, la motivación y los objetivos de este trabajo.

El capítulo dos introduce los conceptos matemáticos necesarios a lo largo de la tesis, los cuales constan de una visión general del cálculo de variable compleja, una discusión sobre funciones de error complejas, el cálculo de Wirtinger y algunas definiciones de cálculo vectorial.

El capítulo tres explica los conceptos básicos sobre las redes neuronales, sus aplicaciones y propiedades, modelo matemático y diagramático, arquitectura y funciones de activación, para finalizar con una discusión de las redes neuronales en el dominio complejo y los problemas que presentan las funciones de activación complejas.

El capítulo cuatro explica el proceso de aprendizaje de una red neuronal, el algoritmo de gradiente descendiente y el método diagramático para la derivación de los términos de gradiente de forma gráfica. Posteriormente habla de los algoritmos de entrenamiento cuasi–Newton, Levenberg–Marquardt fuera de línea y Levenberg–Marquardt recursivo, el cual es tema central de este trabajo.

El capítulo cinco condensa la topología de red neuronal compleja recurrente utilizada en este trabajo, con una función de activación compleja construida, junto con sus términos de gradiente y el algoritmo de entrenamiento de Levenberg–Marquardt recursivo y complejo.

El capítulo seis explica el problema de identificación en el contexto de los sistemas dinámicos. Posteriormente presenta dos modelos de plantas mecánicas, no lineales y oscilatorias, una de una entrada y una salida, y la otra de múltiples entradas y múltiples salidas. Estas son sujetas a un esquema de identificación. Por último presenta los resultados obtenidos en simulación que validan la red neuronal y el algoritmo de entrenamiento propuestos.

El capítulo siete explica el problema de control neuronal adaptable y presenta dos esquemas de control directo. Los dos modelos presentados en el capítulo anterior son sujetos a los esquemas de control propuestos. Posteriormente presenta los resultados obtenidos en simulación que validan dichos esquemas.

El capítulo ocho presenta la discusión de los resultados, las conclusiones generales de esta tesis y el trabajo a futuro.

2. Preliminares Matemáticos

En este capítulo se introducen los conceptos matemáticos necesarios a lo largo de este trabajo, referentes al análisis sobre el campo de los números complejos [20], [21], una discusión sobre funciones de error, el cálculo de Wirtinger y algunos conceptos de cálculo vectorial [22].

2.1 El Campo de los Números Complejos

Los números complejos se pueden definir como pares ordenados de números reales (x, y), los cuales pueden ser interpretados como puntos en el plano complejo, con coordenadas rectangulares x y y. Dado que un número real x puede ser representado como el punto $(x, 0)$, queda claro que el conjunto de números reales es un subconjunto de los números complejos. Un número complejo (x, y) se puede denotar de la forma:

$$z = (x, y) = x + iy \tag{2.1}$$

Donde $i^2 = -1$ es la unidad imaginaria. Siendo $x = \mathrm{Re}(z)$ y $y = \mathrm{Im}(z)$ las partes real e imaginaria respectivamente del número complejo z [20].

Proposición. Dos números complejos $z_1 = x_1 + iy_1, z_2 = x_2 + iy_2$ son iguales entre sí (i.e. $z_1 = z_2$) si y solo si $x_1 = x_2$ y $y_1 = y_2$.

Definición. El conjunto de números complejos $\mathbb{C}$ se define como:

$$\mathbb{C} \coloneqq \{z = x + iy \mid x, y \in \mathbb{R}, i^2 = -1\} \tag{2.2}$$

Definición. Sean $z_1 = x_1 + iy_1, z_2 = x_2 + iy_2$ dos números complejos, la *suma* entre ellos se define como:

$$z_1 + z_2 \coloneqq (x_1 + x_2) + i(y_1 + y_2) \tag{2.3}$$

La operación de suma cumple con las siguientes propiedades:

- $z_1 + z_2 = z_2 + z_1$ (propiedad conmutativa)
- $z_1 + (z_2 + z_3) = (z_1 + z_2) + z_3$ (propiedad asociativa)
- $\exists 0 = 0 + i0$, tal que $0 + z = z + 0 = z$ (elemento neutro aditivo)
- $\forall z = x + iy, \exists(-z) = -x - iy$, tal que $(-z) + z = z + (-z) = 0$ (elemento inverso aditivo)

Definición. Sean $z_1 = x_1 + iy_1, z_2 = x_2 + iy_2$ dos números complejos, el *producto* entre ellos se define como:

$$z_1 z_2 \coloneqq (x_1 x_2 - y_1 y_2) + i(x_1 y_2 + x_2 y_1) \tag{2.4}$$

La operación del producto cumple con las siguientes propiedades:

- $z_1 z_2 = z_2 z_1$ (propiedad conmutativa)
- $z_1(z_2 z_3) = (z_1 z_2) z_3$ (propiedad asociativa)
- $\exists 1 = 1 + i0$, tal que $1 \cdot z = z \cdot 1 = z$ (elemento neutro multiplicativo)
- $\forall z \neq 0, \exists z^{-1}$, tal que $z z^{-1} = z^{-1} z = 1$ (elemento inverso multiplicativo)
- $z_1(z_2 + z_3) = z_1 z_2 + z_1 z_3$ (distributividad respecto a la suma)

Cabe mencionar que $0 \cdot z = z \cdot 0 = 0$ para cualquier número complejo z.

Definición. Sea $z = x + iy$ un número complejo, su *complejo conjugado* se define como:

$$\bar{z} \coloneqq x - iy \tag{2.5}$$

Definición. Sea $z = x + iy$ un número complejo, su *norma Euclidiana* se define como:

$$|z| \coloneqq \sqrt{x^2 + y^2} \tag{2.6}$$

El conjugado y la norma de un número complejo cumplen con las siguientes propiedades:

- $\overline{(z_1 + z_2)} = \bar{z_1} + \bar{z_2}$
- $\overline{(z_1 z_2)} = \bar{z_1}\bar{z_2}$
- $z\bar{z} = |z|^2$
- $|z_1 z_2| = |z_1||z_2|$
- $|z| \geq 0$, luego $|z| = 0 \iff z = 0$
- $|z_1 + z_2| \leq |z_1| + |z_2|$ (desigualdad del triángulo)

El conjunto de los números complejos, junto con las operaciones de suma y producto, forman un campo que geométricamente es similar a $\mathbb{R}^2$, pero que difiere en cuanto a

la falta del inverso multiplicativo en $\mathbb{R}^2$, y otras propiedades analíticas. Por ejemplo, en $\mathbb{R}^2$ existen funciones derivables en todo un conjunto abierto, las cuales son infinitamente derivables en el conjunto y sin embargo su serie de Taylor no converge a la función original; para el caso del campo $\mathbb{C}$ la serie de Taylor de una función infinitamente diferenciable siempre converge a la función original [21].

2.2 Funciones Complejas, Funciones Holomorfas y Funciones Armónicas

Definición. Se define como *función compleja* a un mapeo $f: D \subseteq \mathbb{C} \to \mathbb{C}$, tal que, para todo $z = x + iy$, el resultado de aplicar f al dominio D da como resultado un número complejo $w \in \mathbb{C}$, i.e. $f(z) = w$. Dicho mapeo puede expresarse como:

$$f(z) = f(x, y) = u(x, y) + iv(x, y) = w \tag{2.7}$$

Donde u y v son funciones definidas en $\mathbb{R}^2$.

Definición. Sea $f: D \subseteq \mathbb{C} \to \mathbb{C}$. La función compleja f se dice *continua* en $z \in D$ si, para todo $\varepsilon > 0, \varepsilon \in \mathbb{R}$, existe $\delta > 0, \delta \in \mathbb{R}$ tal que:

$$|z - z'| < \delta \implies |f(z) - f(z')| < \varepsilon \tag{2.8}$$

Definición. Sea $D \subseteq \mathbb{C}$ un conjunto abierto y $z_0 \in D$. Sea $f: D \to \mathbb{C}$, entonces f se dice que es *complejo diferenciable* en z_0 si el límite:

$$f'(z_0) = \lim_{z \to z_0} \frac{f(z) - f(z_0)}{z - z_0} \tag{2.9}$$

Existe, y en ese caso, $f'(z_0)$ es la *derivada* de f en el punto z_0. También denotado por $\frac{d}{dz} f(z_0)$.

Teorema. Si la derivada de una función compleja f es cero en todo un dominio D, entonces f es constante en cualquier punto del dominio D.

Cabe mencionar que el concepto de derivada es diferente en el dominio real y el dominio complejo. Por ejemplo, si consideramos la función $f: \mathbb{C} \to \mathbb{C}, f(z) = \bar{z}$ en el dominio real, se obtiene la función $f: \mathbb{R}^2 \to \mathbb{R}^2, f(x, y) = (x, -y)$, la cual es real diferenciable, mientras que f no es complejo diferenciable en ningún punto de $\mathbb{C}$.

Proposición. La existencia de la derivada de una función en un punto del plano complejo implica la continuidad de la función en ese punto.

Definición. Si f es diferenciable en todo el conjunto abierto $D \subseteq \mathbb{C}$ (i.e. el límite presentado en la ecuación (2.9) existe para todo $z_0 \in D$), entonces se dice que f es una función *holomorfa* en D.

Las reglas de derivación para el caso complejo son las mismas que en el dominio real. Cabe mencionar dos propiedades de las funciones complejas que no se cumplen para el caso de funciones reales: toda función compleja que cumple con la propiedad de ser holomorfa es también una función analítica[1]; y se tiene que las funciones holomorfas son infinitamente complejo diferenciables.

Es importante saber si una función compleja es diferenciable antes de aplicar cualquier regla de derivación. Las condiciones de Cauchy–Riemann dan condiciones necesarias de diferenciabilidad de una función compleja.

Condiciones de Cauchy–Riemann. Sea $f(z) = u(x,y) + iv(x,y)$ una función compleja. Si $f'(z)$ existe para algún punto $z \in \mathbb{C}$, entonces las derivadas parciales de primer orden, respecto a x y y, de cada uno de los componentes u y v existen en ese punto z, y además satisfacen las condiciones:

$$\frac{\partial}{\partial x}u(x,y) = \frac{\partial}{\partial y}v(x,y), \qquad \frac{\partial}{\partial y}u(x,y) = -\frac{\partial}{\partial x}v(x,y) \tag{2.10}$$

Luego, $f'(z)$ puede ser expresado en función de las derivadas parciales:

$$f'(z) = \frac{\partial}{\partial x}u(x,y) + i\frac{\partial}{\partial x}v(x,y) = \frac{\partial}{\partial y}v(x,y) - i\frac{\partial}{\partial y}u(x,y) \tag{2.11}$$

A partir de ciertas condiciones de continuidad, el siguiente teorema provee condiciones suficientes de diferenciabilidad de una función compleja.

Teorema. Sea $f(z) = u(x,y) + iv(x,y)$ una función compleja definida para cierta vecindad de radio ε alrededor del punto $z_0 = x_0 + iy_0$, y suponiendo que las derivadas parciales de primer orden, respecto a x y y, de u y v existen en cualquier punto de esta vecindad. Si estas derivadas parciales son continuas en el punto (x_0, y_0), y satisfacen (2.10) en este punto, entonces $f'(z_0)$ existe.

En el contexto de las redes neuronales, la función de activación juega un papel importante en el desarrollo del algoritmo de entrenamiento, así como la estabilidad y desempeño de toda la red. En el dominio real, se busca que una función de activación

[1] Resultado de un teorema muy importante en análisis complejo [21]. Aunque en el contexto de funciones complejas los términos *holomorfa* y *analítica* son equivalentes, en este trabajo se optó por el uso del término *holomorfa*.

sea diferenciable y acotada, mientras que en el dominio complejo estas dos condiciones no son suficientes.

Definición. Una función compleja f se dice que es *holomorfa* en un punto z_0 si su derivada $f'(z)$ existe en z_0 y en cada punto de una vecindad alrededor de z_0. O bien, f es holomorfa en un conjunto abierto $D \subseteq \mathbb{C}$ si su derivada existe para cada punto de este conjunto.

Definición. Una función compleja f se dice que es *entera* si es holomorfa en todo $\mathbb{C}$.

Ejemplos de funciones holomorfas son: z^n, e^z, $\sin(z)$, $\cos(z)$. Ejemplos de funciones no holomorfas son: $\bar{z}$, $|z|^2$.

A continuación se presenta el concepto de función armónica, el cual se utiliza como herramienta para mostrar cuando una función es holomorfa.

Definición. Sea $f(x,y)$ una función real de dos variables, se dice que f es *armónica* sobre un dominio del plano real $D \subseteq \mathbb{R}^2$ si esta tiene derivadas parciales continuas de primer y segundo orden definidas para cualquier punto en D, y además cumplen con la ecuación:

$$\frac{\partial^2}{\partial x^2} f(x,y) + \frac{\partial^2}{\partial y^2} f(x,y) = 0 \tag{2.12}$$

A esta ecuación en derivadas parciales se le conoce como ecuación de Laplace, y aparece en gran número de aplicaciones en matemáticas, física e ingeniería. De las ecuaciones de condición de Cauchy–Riemann (2.10) se observa que las derivadas parciales de segundo orden de u y v son:

$$\frac{\partial^2}{\partial x^2} u(x,y) = \frac{\partial^2}{\partial x \partial y} v(x,y), \qquad \frac{\partial^2}{\partial y^2} u(x,y) = -\frac{\partial^2}{\partial y \partial x} v(x,y) \tag{2.13}$$

Dado que el lado derecho en (2.13) es igual, pero con signo contrario, se sigue que:

$$\frac{\partial^2}{\partial x^2} u(x,y) + \frac{\partial^2}{\partial y^2} u(x,y) = 0 \tag{2.14}$$

De igual forma se puede obtener el mismo resultado para la función v. El resultado anterior se resume en el siguiente teorema:

Teorema. Sea $f(z) = u(x,y) + iv(x,y)$ una función compleja holomorfa, entonces las funciones reales u y v son funciones armónicas.

Si las dos funciones u y v son armónicas en el dominio D, y sus derivadas parciales de primer orden satisfacen (2.14), entonces se dice que v es el conjugado armónico de u. Luego se tiene el siguiente teorema:

Teorema. Una función $f(z) = u(x, y) + iv(x, y)$ es holomorfa en el dominio D si y solo si v es el conjugado armónico de u.

2.3 Funciones Complejas Acotadas y Puntos de Singularidad

Para que una función compleja sea candidata a función de activación para una red neuronal, es necesario que sea holomorfa y acotada. A continuación se presenta la definición formal de función acotada.

Definición. Sea f una función de un dominio D, y sea $M > 0, M \in \mathbb{R}$. Se dice que f es acotada si $|f(x)| \leq M$ para cualquier punto $x \in D$.

Teorema de Liouville. Para toda función compleja holomorfa f, si f es acotada en todo $\mathbb{C}$, entonces f es una función constante en todo $\mathbb{C}$.

Condiciones más fuertes a las presentadas por el teorema de Liouville, sobre funciones no acotadas, están dadas por el siguiente teorema:

"Pequeño" Teorema de Picard. Si una función compleja f es entera y no constante, entonces el conjunto de valores que puede tomar $f(z)$ es el plano complejo $\mathbb{C}$, o $\mathbb{C}$ menos un punto.

Se observa de estos teoremas que, si se busca una función no constante, esta no puede ser holomorfa y acotada a la vez, lo cual causa conflicto al elegir candidatas para funciones de activación.

Otra propiedad que se busca en estas funciones es que no contengan ningún punto de singularidad en su dominio, esto debido a que los algoritmos de entrenamiento utilizados divergen o se desestabilizan alrededor de estos. Los puntos de singularidad se definen a continuación:

Definición. Si una función f no es holomorfa en un punto z_0, pero sí lo es para cualquier punto de la vecindad de z_0, entonces z_0 es un *punto de singularidad*.

Por ejemplo, el punto $z = 0$ es una singularidad de la función $f(z) = 1/z$. Por otro lado, la función $f(z) = |z|^2$ no tiene puntos de singularidad en su dominio, dado que la función no es holomorfa en ningún punto.

Definición. Si una función compleja $f(z)$ contiene sólo un punto singular en z_0, se dice que la singularidad es *removible* si el límite $\lim_{z\to z_0} f(z)$ existe.

Definición. Si la función $f(z)$ es holomorfa en una vecindad de z_0 y $\lim_{z\to z_0}|f(z)| \to \infty$, entonces la singularidad no es removible y se le llama *aislada*.

Para el caso de singularidades removibles, se puede forzar el valor del límite en el punto de singularidad; mientras que para singularidades aisladas, se puede restringir el dominio de definición para que la función no tome esos valores. A partir de este tratamiento que se da a los puntos de singularidad, y dado que el teorema de Liouville es válido para todo el plano complejo, se hace la siguiente proposición:

Proposición. En un dominio acotado del plano complejo $D \subset \mathbb{C}$, una función de activación compleja y no lineal f necesita ser holomorfa, acotada y no contener ningún punto de singularidad en D [18].

Si se considera una región restringida del plano complejo, es posible encontrar una función que cumpla con estas características. Para esto se utilizan las funciones complejas construidas, las cuales consisten en deshacerse de las propiedades que hacen que una función no constante sea no holomorfa y no acotada al mismo tiempo, al igual que unir más de una función al borde de las singularidades [23].

El tema de funciones complejas construidas se abordará en un capítulo posterior.

2.4 Función de Error

El entrenamiento de una red neuronal consiste en minimizar una función de costo, la cual se encuentra expresada en términos de los pesos sinápticos W, por medio de un algoritmo de optimización. Por lo general, la función de costo consiste de una función del error, por lo que es de gran importancia discutir sobre su naturaleza analítica en el dominio complejo.

La función del error medio cuadrático en el dominio complejo se define por la ecuación [24]:

$$\zeta(k) = \frac{1}{2}\sum_j \left[E_j(k)\cdot\overline{E_j(k)}\right], \qquad \zeta = \frac{1}{N_e}\sum_k \zeta(k) \tag{2.15}$$

Donde $E_j(k) = Y_{j,N}(k;W) - Y_{j,P}(k)$ es el j–ésimo error en el instante k entre la salida de la red neuronal y la salida de la planta. La función de error $\zeta(k)$ se utiliza para

aplicaciones en línea de la red neuronal, mientras que la función de error ζ se utiliza para aplicaciones fuera de línea, para una época de N_e iteraciones.

La función de error (2.15) no cumple con las condiciones de Cauchy–Riemann (2.10), por lo que no es una función holomorfa y no se puede obtener su derivada compleja. A continuación se demuestra esta afirmación, tomando en cuenta que la función (2.15) contiene un número complejo conjugado, basta con demostrar que esta no es holomorfa.

Sea una función de complejo conjugado:

$$f(z) = \bar{z} = x - iy \tag{2.16}$$

Definiendo $u(x, y) = x$ y $v(x, y) = -y$, sus derivadas parciales respecto a x y y quedan de la forma:

$$\frac{\partial}{\partial x}u = 1 \neq \frac{\partial}{\partial y}v = -1, \qquad \frac{\partial}{\partial y}u = \frac{\partial}{\partial x}v = 0 \tag{2.17}$$

Por lo cual la función de complejo conjugado no es holomorfa en ningún punto $z \in \mathbb{C}$ y, por extensión, tampoco lo es la función (2.15).

A continuación, se presenta el cálculo de Wirtinger, el cual es una herramienta de gran utilidad para encontrar derivadas de funciones no holomorfas.

2.5 Cálculo de Wirtinger

El cálculo de Wirtinger se utiliza en problemas de optimización de funciones complejas no holomorfas [25]. Sea una función compleja:

$$w = f(z) = u(x, y) + iv(x, y), \qquad z \in \mathbb{C}, \qquad x, y \in \mathbb{R} \tag{2.18}$$

Para el caso particular donde $v(x, y) = 0$, la función f es no holomorfa generalmente. Luego, (2.15) se reduce al caso real de dos variables $f(z) = u(x, y)$ y el problema de optimización queda definido de la forma:

$$f(z) \to \text{opt} \iff u(x, y) \to \text{opt} \tag{2.19}$$

El cual se cumple con las siguientes condiciones:

$$\frac{\partial}{\partial x}u(x, y) = 0, \qquad \frac{\partial}{\partial y}u(x, y) = 0 \tag{2.20}$$

Dado que $0 + i0 = 0$, la condición (2.19) se puede reescribir en una sola ecuación compleja:

$$a_1 \frac{\partial}{\partial x} u(x,y) + i a_2 \frac{\partial}{\partial y} u(x,y) = 0 \tag{2.21}$$

Donde a_1 y a_2 son constantes reales diferentes a cero. Haciendo $a_1 = a_2 = 1/2$, se define el operador:

$$\frac{\partial}{\partial z} \coloneqq \frac{1}{2} \cdot \frac{\partial}{\partial x} + i \frac{1}{2} \cdot \frac{\partial}{\partial y} \tag{2.22}$$

Definición. Las derivadas parciales de una función $f(z)$ con variable compleja $z = x + iy$, con respecto a z y a $\bar{z}$ se definen:

$$\frac{\partial}{\partial z} f(z) \coloneqq \frac{1}{2}\left(\frac{\partial}{\partial x} - i \frac{\partial}{\partial y}\right) f(z) \tag{2.23}$$

$$\frac{\partial}{\partial \bar{z}} f(z) \coloneqq \frac{1}{2}\left(\frac{\partial}{\partial x} + i \frac{\partial}{\partial y}\right) f(z) \tag{2.24}$$

Usando las definiciones (2.23), (2.24) a la función (2.15), tomando en cuenta $z = E_j(k)$, se obtienen las derivadas parciales respecto a z y $\bar{z}$ de la función $\zeta(k)$:

$$\frac{\partial}{\partial z} z\bar{z} \coloneqq \frac{1}{2}\left(\frac{\partial}{\partial x} - i \frac{\partial}{\partial y}\right)(x^2 + y^2) = \frac{1}{2}(2x - 2iy) = \bar{z} \tag{2.25}$$

$$\frac{\partial}{\partial \bar{z}} z\bar{z} \coloneqq \frac{1}{2}\left(\frac{\partial}{\partial x} + i \frac{\partial}{\partial y}\right)(x^2 + y^2) = \frac{1}{2}(2x + 2iy) = z \tag{2.26}$$

En la Tabla 2.1 se muestran las derivadas de algunas funciones básicas utilizando el cálculo de Wirtinger.

Tabla 2.1 Derivadas de algunas funciones básicas utilizando cálculo de Wirtinger

$f(z)$	cz	$c\bar{z}$	$z\bar{z}$
$D_z f(z)$	c	0	$\bar{z}$
$D_{\bar{z}} f(z)$	0	c	z

Al igual que en la derivada compleja, las reglas de suma, producto y cociente se mantienen para la derivación en el cálculo de Wirtinger. Para funciones holomorfas la derivada de Wirtinger coincide con la derivada compleja y las condiciones de Cauchy–Riemann se cumplen. De igual forma, la derivada de Wirtinger para funciones reales coincide con la derivada real [25].

De forma intuitiva se puede decir que el cálculo de Wirtinger se encuentra en un punto intermedio entre la derivada real de una función real y la derivada compleja de una función compleja.

2.6 Cálculo Vectorial

A continuación se presentan algunos conceptos básicos de cálculo vectorial necesarios para el desarrollo de este trabajo [22].

Definición. Sea $f: \mathbb{R}^n \to \mathbb{R}$ una función escalar diferenciable en todo su dominio, el gradiente de $f(x_1, \cdots, x_n)$ es un vector columna cuyas componentes son las derivadas parciales de f respecto a $x_i,\ i = 1, \cdots, n$. O bien:

$$\boldsymbol{\nabla} f(x_1, \cdots, x_n) \coloneqq \left[\frac{\partial f}{\partial x_1}, \cdots, \frac{\partial f}{\partial x_n}\right]^T \tag{2.27}$$

Definición. Sea $f: \mathbb{R}^n \to \mathbb{R}$ una función escalar diferenciable en todo su dominio, la matriz Hessiana de $f(x_1, \cdots, x_n)$ es una matriz cuadrada y simétrica cuyas componentes son todas las derivadas parciales de segundo orden de f respecto a $x_i,\ i = 1, \cdots, n$. O bien:

$$\mathbf{H} \coloneqq \boldsymbol{\nabla}^2 f(x_1, \cdots, x_n) = \begin{bmatrix} \dfrac{\partial^2 f}{\partial x_1^2} & \dfrac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_1 \partial x_n} \\ \dfrac{\partial^2 f}{\partial x_2 \partial x_1} & \dfrac{\partial^2 f}{\partial x_2^2} & \cdots & \dfrac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial^2 f}{\partial x_n \partial x_1} & \dfrac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_n^2} \end{bmatrix}_{n \times n} \tag{2.28}$$

Definición. Sea $\mathbf{f}: \mathbb{R}^n \to \mathbb{R}^m$ una función vectorial diferenciable en todo su dominio, la matriz Jacobiana de $\mathbf{f}(x_1, \cdots, x_n)$ es la matriz cuyas componentes son las derivadas parciales de $f_i,\ i = 1, \cdots, m$ respecto a $x_j = 1, \cdots, n$. O bien:

$$\mathbf{J} \coloneqq \frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \dfrac{\partial f_1}{\partial x_2} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \dfrac{\partial f_2}{\partial x_1} & \dfrac{\partial f_2}{\partial x_2} & \cdots & \dfrac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial x_1} & \dfrac{\partial f_m}{\partial x_2} & \cdots & \dfrac{\partial f_m}{\partial x_n} \end{bmatrix}_{m \times n} \tag{2.29}$$

Observación. Sea $f: \mathbb{R}^n \to \mathbb{R}$ una función escalar diferenciable en todo su dominio, entonces la matriz Hessiana y la matriz Jacobiana se encuentran relacionadas por medio de la siguiente ecuación:

$$\mathbf{H}\big(f(x_1, \cdots, x_n)\big) = \mathbf{J}\big(\boldsymbol{\nabla} f(x_1, \cdots, x_n)\big) \tag{2.30}$$

3. Redes Neuronales

En este capítulo se aborda el tema de las redes neuronales, con su definición, propiedades, aplicaciones, representación, topología y los diferentes tipos de redes, funciones de activación y la diferencia entre redes neuronales en el dominio de los números reales y los números complejos.

También se introduce la noción de funciones de activación en el dominio complejo y funciones de activación construidas, las cuales son de gran utilidad en el desarrollo de este trabajo.

3.1 Definición, Propiedades y Aplicaciones

En su forma más general, una Red Neuronal Artificial, es un agente diseñado para modelar y emular la forma como el cerebro, el cual puede verse como una Red Neuronal Biológica, desempeña una tarea particular o una función de interés, esto a través de un proceso de aprendizaje.

En [1] se ofrece una definición formal de una red neuronal:

> *Una **Red Neuronal Artificial** es un procesador, masivo y en paralelo, constituido de unidades de procesamiento sencillas, la cual tiene una gran disposición natural de almacenar conocimiento experimental y hacerlo disponible para su uso. Se asemeja al cerebro en dos aspectos:*
>
> - *El conocimiento es adquirido del ambiente por medio de un proceso de aprendizaje.*
> - *Las fuerzas de interconexión entre neuronas se utilizan para guardar el conocimiento.*

El proceso de aprendizaje de una red neuronal se lleva a cabo por medio de un algoritmo de entrenamiento, el cual modifica las fuerzas de interconexión entre neuronas, conocidas como pesos sinápticos, para obtener una respuesta deseada. Luego, el proceso de generalización se refiere a la red neuronal produciendo una respuesta razonable a estímulos no presentes durante la etapa de aprendizaje.

Las redes neuronales cuentan con las siguientes propiedades [1]:

- **No Linealidad.** Una red compuesta de neuronas no lineales interconectadas es en sí no lineal. Es una propiedad importante si la naturaleza del fenómeno al cual se le va a aplicar es inherentemente no lineal.
- **Mapeo Entrada–Salida.** En la etapa de aprendizaje, la red neuronal construye un mapeo de entrada–salida de los ejemplos utilizados y sus correspondientes respuestas.
- **Adaptabilidad.** Las redes neuronales tienen integrada la capacidad de adaptarse a cambios en el ambiente. En particular, una red entrenada para operar en un ambiente muy específico puede ser fácilmente *reentrenada* para tratar con cambios menores en las condiciones del ambiente en las que opera.
- **Información Contextual.** El conocimiento queda representado por el estado de activación de las neuronas, así como la estructura global de la red. Cada neurona queda potencialmente afectada por todas las demás neuronas.
- **Tolerancia a Fallas.** Las redes neuronales tienen la capacidad de cómputo robusto. Por ejemplo, en una implementación de *hardware*, si una neurona o sus conexiones son dañadas, el acceso a ciertos patrones guardados previamente puede verse dañado, pero esto no necesariamente afecta su desempeño global. Una red neuronal exhibe una degradación en su desempeño gradual en vez del fallo total en su funcionamiento.
- **Implementación a Gran Escala.** La naturaleza masiva y en paralelo de una red neuronal la hace adecuada para el uso de tecnología VLSI (*very–large–scale–integration*).
- **Uniformidad en Análisis y Diseño.** Las neuronas representan un elemento común para cualquier red neuronal, esto permite el uso de cualquier conocimiento o algoritmo desarrollado para tareas específicas en cualquier otra aplicación o topología. De igual forma, se pueden construir e integrar redes modulares.
- **Analogía Neurobiológica.** Las redes neuronales se diseñan tomando como analogía al cerebro, el cual es prueba de que el procesamiento en paralelo, a prueba de fallas y adaptable es físicamente posible y muy poderoso computacionalmente.

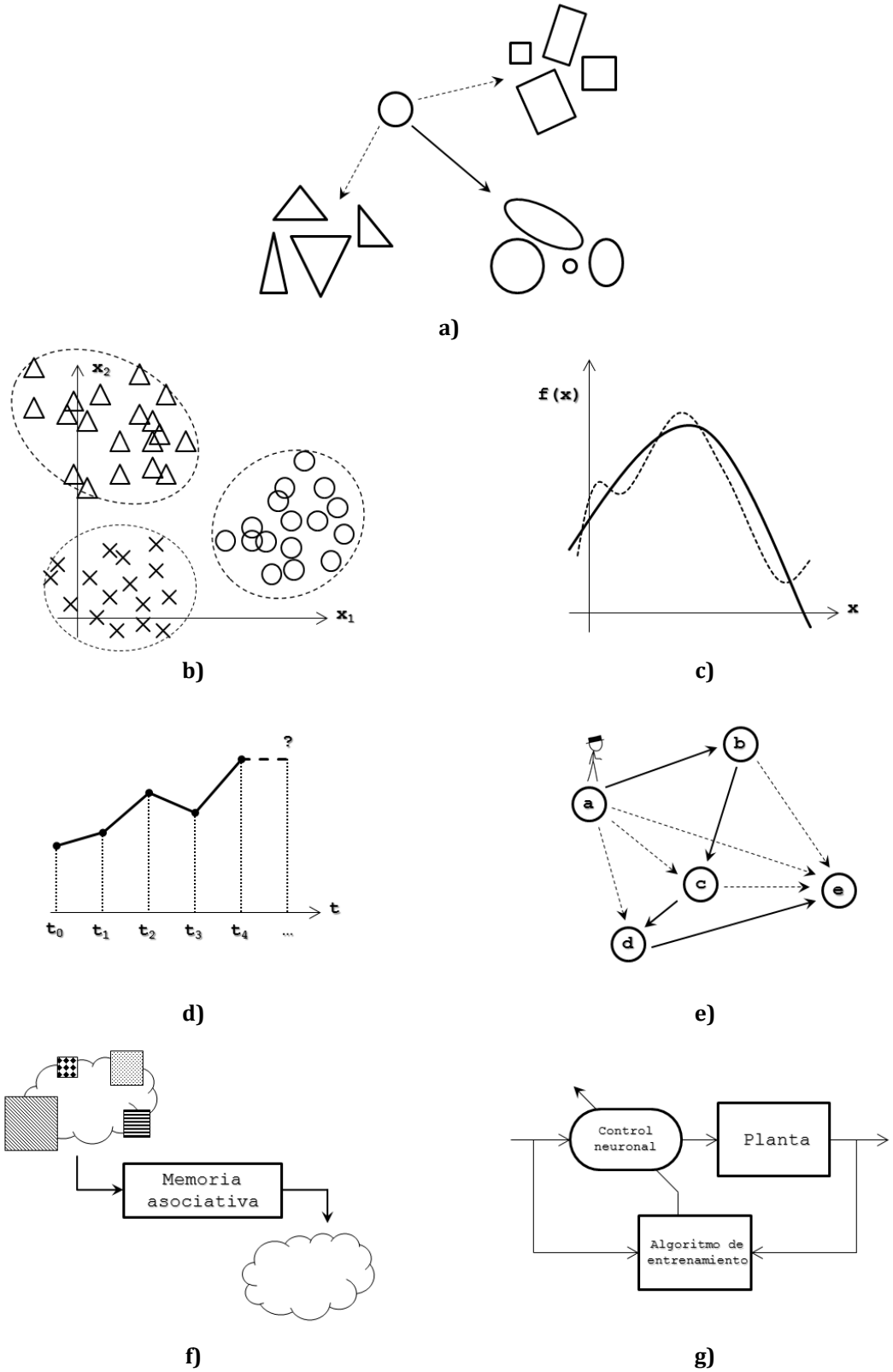


Figura 3.1 Diversas aplicaciones de las redes neuronales, a) clasificador, b) clusterisación, c) aproximación de funciones, d) predictor, e) optimización (ejemplo del *travelling salesman problem*), f) recuperación por contenido y g) sistemas de control neuronal

Las redes neuronales tienen diversos usos (Figura 3.1). Estas pueden ser utilizadas como clasificadoras, en tareas de reconocimiento de patrones, tareas de clusterisación, aproximación de funciones, predictores, problemas de optimización, como recuperadores de información o en un esquema de control neuronal. Esta última aplicación es de gran importancia en el presente trabajo.

Las redes neuronales presentan una alternativa heurística a problemas de alto costo computacional o con dificultad de resolución para métodos analíticos tradicionales. De igual forma la naturaleza de la información que puede manejar una red neuronal es muy amplia, ya que se pueden utilizar para tratar fenómenos físicos, sistemas sociales, sistemas económicos o datos estadísticos, entre otros.

3.2 Modelo de una Neurona

La neurona es una unidad de procesamiento de información, la cual es fundamental para la operación de una red neuronal y consta de tres elementos básicos: las interconexiones caracterizadas por los pesos sinápticos, el punto de adición y la función de activación. Algunos modelos incluyen un término de *bias*, el cual incrementa o decrementa la señal total de entrada a la función de activación.

En términos matemáticos, una neurona n se describe por las siguientes ecuaciones:

$$u_n = \sum_{j=1}^{N} w_{nj} \cdot x_j \tag{3.1}$$

$$y_n = \varphi(u_n + b_n) = \varphi(v_n) \tag{3.2}$$

Donde x_j y w_{nj} son la señal de entrada y el peso sináptico correspondientes a la entrada j; u_n es la combinación lineal de las señales de entrada, b_n es el término de *bias*, y $\varphi(\cdot)$ es la función de activación. Una representación diagramática de la neurona descrita en (3.1), (3.2) se presenta en la Figura 3.2.

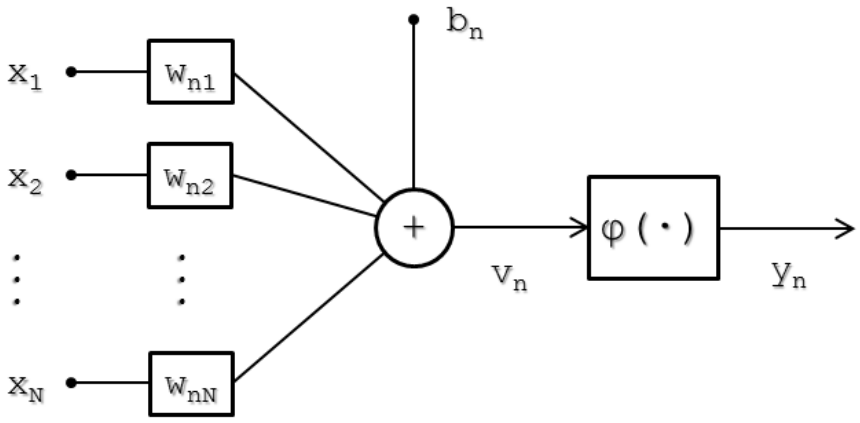


Figura 3.2 Diagrama del modelo de una neurona artificial

3.3 Arquitectura de una Red Neuronal

La arquitectura o topología de una red neuronal se refiere a la estructura de las neuronas que la conforman y sus interconexiones. En una red, las neuronas hacen la función de nodos, los cuales se conectan unidireccionalmente por medio de los pesos sinápticos.

En general, las neuronas se agrupan en unidades estructurales denominadas capas, y el conjunto de una o más capas constituyen a la red neuronal. Estas capas se clasifican de acuerdo a su posición dentro de la red:

- **Capa de entrada.** O capa sensorial, se compone por las neuronas que reciben señales procedentes del entorno.
- **Capa de salida.** Se compone por las neuronas que producen la respuesta total de la red neuronal.
- **Capa oculta.** Se compone por las neuronas que no tienen una conexión directa con el ambiente, las cuales proporcionan a la red neuronal con grados de libertad. Una red neuronal puede no tener capas ocultas, o en el caso de tenerlas, puede tener más de una.

Las redes neuronales pueden clasificarse por el número de capas que la conforman, o bien, por la existencia de lazos de realimentación de señal.

Redes de una capa con conexiones hacia adelante. Este tipo de redes cuentan únicamente con una capa de entrada y una capa de salida. Debido a que en la capa de entrada no se realiza ningún cálculo, esta no se cuenta en el número de capas totales de la red. El flujo de las señales siempre va de la capa de entrada a la capa de salida, o

bien, hacia adelante (*feedforward* en inglés). La Figura 3.3 muestra el diagrama de un ejemplo de red de este tipo.

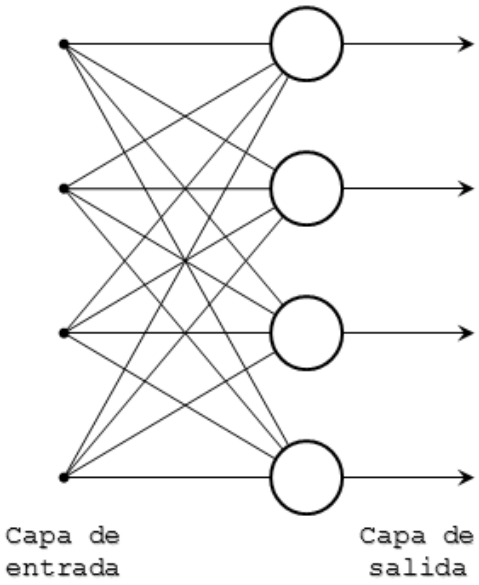


Figura 3.3 Diagrama de una red de una capa con conexiones hacia adelante

Redes multicapa con conexiones hacia adelante. Este tipo de redes cuenta con una o más capas ocultas, además de la capa de entrada y de salida. Las conexiones entre diferentes capas ocultas, y con las capas de entrada y de salida, aumentan la complejidad del procesamiento de las señales internas, lo que aumenta la cantidad de información que puede procesar y guardar la red neuronal. Se dice que una red multicapa es totalmente conectada si cada neurona de cada capa se encuentra conectada a todos los nodos de la capa siguiente. La Figura 3.4 muestra el diagrama de un ejemplo de red de este tipo.

Redes recurrentes. Las redes recurrentes, a diferencia de las redes con conexiones hacia adelante, cuentan con al menos un lazo de realimentación de alguna de sus señales. Por lo general estos lazos de realimentación se llevan a cabo por medio de operadores de retraso (z^{-1}), que resultan en comportamientos dinámicos no lineales, los cuales aumentan la capacidad de aprendizaje y desempeño de la neurona. La Figura 3.5 muestra el diagrama de una red neuronal con lazos de realimentación no propios (la señal de salida de una neurona no se realimenta a sí misma) y sin capas ocultas.

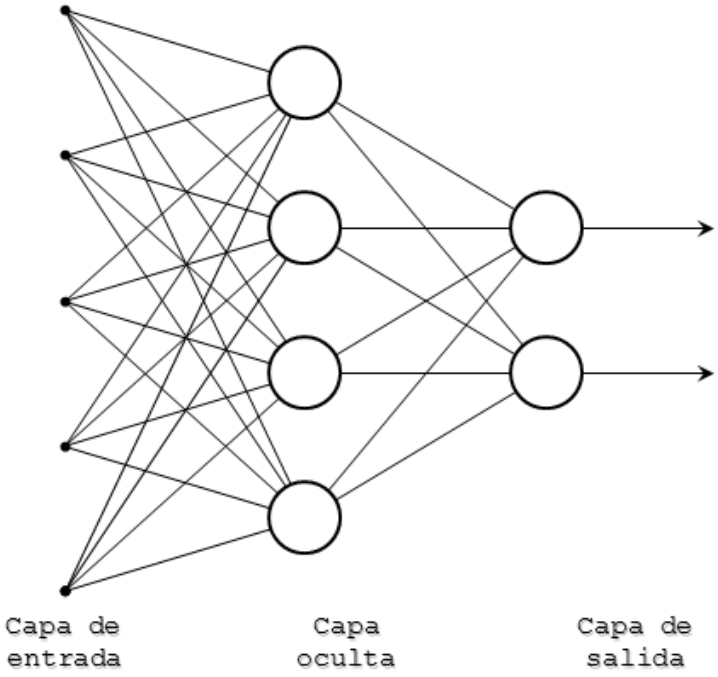


Figura 3.4 Diagrama de una red multicapa totalmente conectada, con conexiones hacia adelante

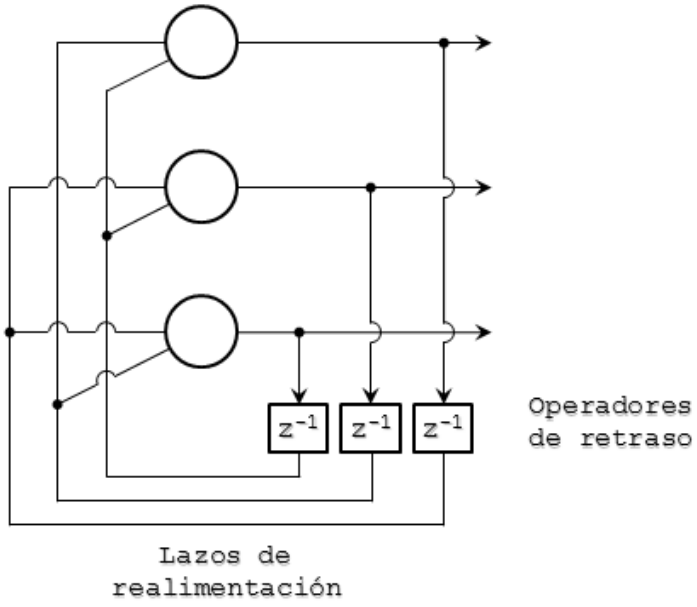


Figura 3.5 Diagrama de una red recurrente de una capa

3.4 Funciones de Activación

La función de activación $\varphi: \mathbb{R} \rightarrow \mathbb{R}$ de una neurona define su salida en términos del campo local inducido $v_n = u_n + b_n$. Usualmente se busca que estas funciones sean diferenciables en todo su dominio y que sean acotadas en su salida en el intervalo $[0,1]$ o $[-1,1]$. Las funciones de activación comúnmente utilizadas son:

- **Función escalón.** O función de Heaviside, definida por la ecuación:

$$\varphi(v_n) = \begin{cases} 1 & \text{si} \quad v_n \geq 0 \\ 0 & \text{si} \quad v_n < 0 \end{cases} \tag{3.3}$$

- **Función lineal a pedazos.** O función lineal con saturación, definida por la ecuación:

$$\varphi(v_n) = \begin{cases} 0 & \text{si} \quad v_n \leq -0.5 \\ v_n + 0.5 & \text{si} \quad -0.5 < v_n < 0.5 \\ 1 & \text{si} \quad v_n \geq 0.5 \end{cases} \tag{3.4}$$

Estas dos funciones tienen puntos en los cuales son no diferenciables y son acotadas en el intervalo de $[0,1]$.

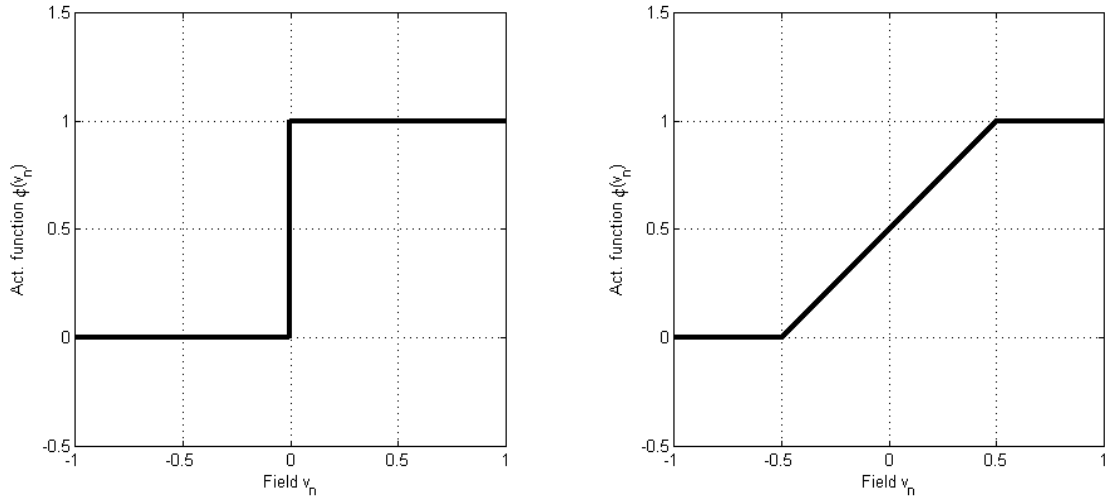


Figura 3.6. Función de activación: a) escalón y b) lineal a pedazos

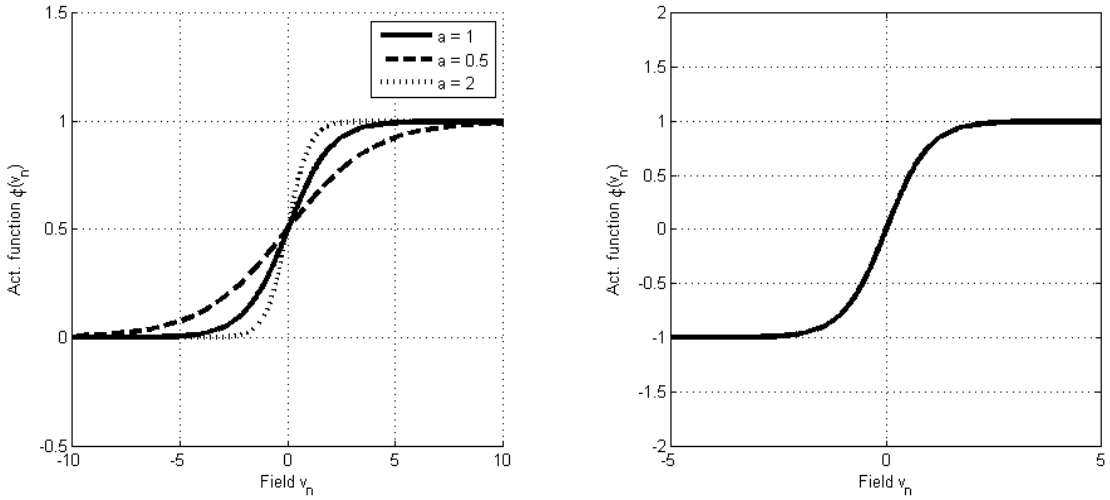
- **Función sigmoidea.** Es la función de activación más utilizada en la construcción de redes neuronales. Al igual que la función lineal a pedazos, esta

cuenta con una región de comportamiento lineal, pero es diferenciable en todo su dominio. Está definida por la ecuación:

$$\varphi(v_n) = \frac{1}{1 + e^{-av_n}} \tag{3.5}$$

- **Función tangente hiperbólica.** Con un comportamiento parecido a la función sigmoidea, pero su salida toma valores en el intervalo de $[-1,1]$. Está definida por la ecuación:

$$\varphi(v_n) = \tanh(v_n) = \frac{1 - e^{-2v_n}}{1 + e^{-2v_n}} \tag{3.6}$$



a) **b)**

Figura 3.7. Función de activación: a) sigmoidea con valores para *a* de 0.5, 1 y 2; y b) tangente hiperbólico

3.5. Redes Neuronales en el Dominio Complejo

Las redes neuronales previamente mencionadas en este trabajo se refieren únicamente al dominio real. A continuación se presenta una breve discusión de las redes neuronales en el dominio complejo.

Una red neuronal compleja toma sus valores de entrada, salida y pesos sinápticos en el dominio de los números complejos [9]. De igual forma, las funciones de activación de sus neuronas son funciones valuadas en el dominio complejo.

Esto presenta ventajas sobre su contraparte en el dominio complejo. Por ejemplo, en [26], se muestra como el problema de aproximar la función lógica XOR es resuelto con una única neurona para el caso complejo, lo cual no es posible para el caso real.

Por lo general, las redes neuronales complejas presentan una mayor riqueza en su aprendizaje, cantidad de información que puede almacenar, capacidad de procesamiento y desempeño en general.

En la Figura 3.8 se muestra el diagrama general de una neurona en el dominio complejo. Aunque por lo general la representación diagramática de una red neuronal compleja es la misma a su contraparte real, por lo general su arquitectura quedará definida por la función de activación.

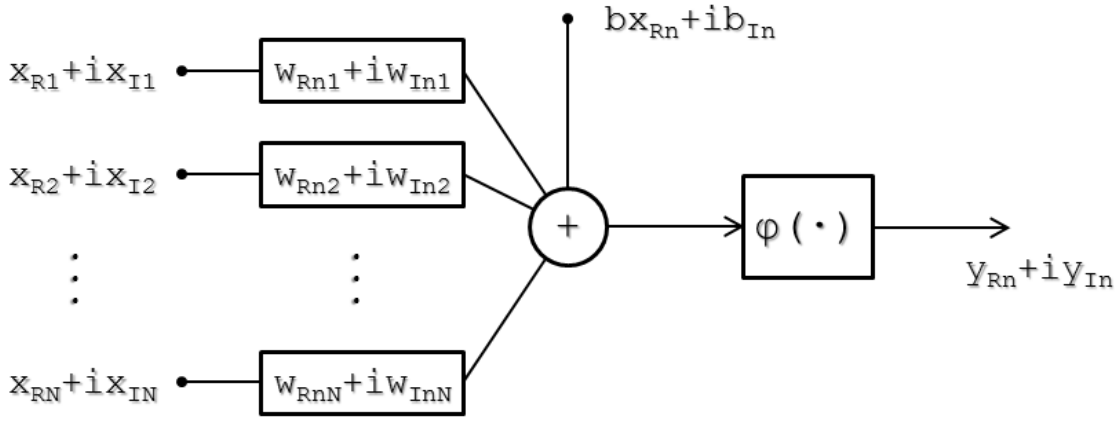


Figura 3.8 Diagrama del modelo de una neurona artificial compleja

En términos matemáticos, una neurona n en el dominio complejo se describe por las siguientes ecuaciones:

$$u_n = \sum_{j=1}^{N} \left(w_{\mathrm{Re},nj} + iw_{\mathrm{Im},nj}\right) \cdot \left(x_{\mathrm{Re},j} + ix_{\mathrm{Im},j}\right) \tag{3.7}$$

$$y_{\mathrm{Re},n} + iy_{\mathrm{Im},n} = \varphi\left(u_n + b_{\mathrm{Re},n} + ib_{\mathrm{Im},n}\right) = \varphi(v_n) \tag{3.8}$$

A continuación se habla sobre las funciones de activación en el dominio complejo.

3.6 Funciones de Activación en el Dominio Complejo

Para el caso de las redes neuronales en el dominio complejo, las condiciones para elegir una función de activación son más fuertes: se busca que estas sean holomorfas, acotadas y sin puntos de singularidad en su dominio [27]. Como se mencionó en el capítulo anterior, no existe función valuada en los números complejos que no sea constante y a la vez sea holomorfa y acotada en todo su dominio.

Por otro lado existe el problema de puntos de singularidad en el dominio de una función. Por ejemplo, la función de la tangente hiperbólica compleja presenta singularidades periódicas sobre la recta de los números reales, como se muestra en las Figuras 3.9 y 3.10.

Ambos problemas se pueden resolver por medio de funciones complejas construidas [23], las cuales buscan eliminar las propiedades que hacen que una función sea no holomorfa y acotada al mismo tiempo, o sustituye las singularidades que esta pueda presentar.

Algunos ejemplos de estas funciones presentadas en [23] están dados por las siguientes ecuaciones:

$$f_1(z) = \varphi(z), \qquad \forall z \in \mathbb{C} \setminus D \tag{3.9}$$

$$f_2(z) = u(\mathrm{Re}(z)) + iv(\mathrm{Im}(z)), \qquad \forall z \in \mathbb{C} \tag{3.10}$$

$$f_3(z) = \varphi(\mathrm{abs}(z)) \cdot \exp\big(i\phi(z)\big), \qquad \forall z \in \mathbb{C} \tag{3.11}$$

Donde $\varphi: \mathbb{C} \to \mathbb{C}$ es un mapeo con valores complejos, $u, v: \mathbb{R} \to \mathbb{R}$ son mapeos con valores reales, $\mathrm{abs}(\cdot)$ es la función de valor de magnitud y $\phi(\cdot)$ es la función de fase de un número complejo. La ecuación (3.9) presenta una función restringida en su dominio, donde $D \subset \mathbb{C}$ es el subconjunto del plano complejo, no necesariamente conexo, que contiene los puntos de singularidad de la función. Las ecuaciones (3.10) y (3.11) presenta dos funciones complejas construidas en forma rectangular y polar.

Es importante mencionar que las funciones de activación definen de forma directa la arquitectura de la red neuronal y la derivación del algoritmo de entrenamiento.

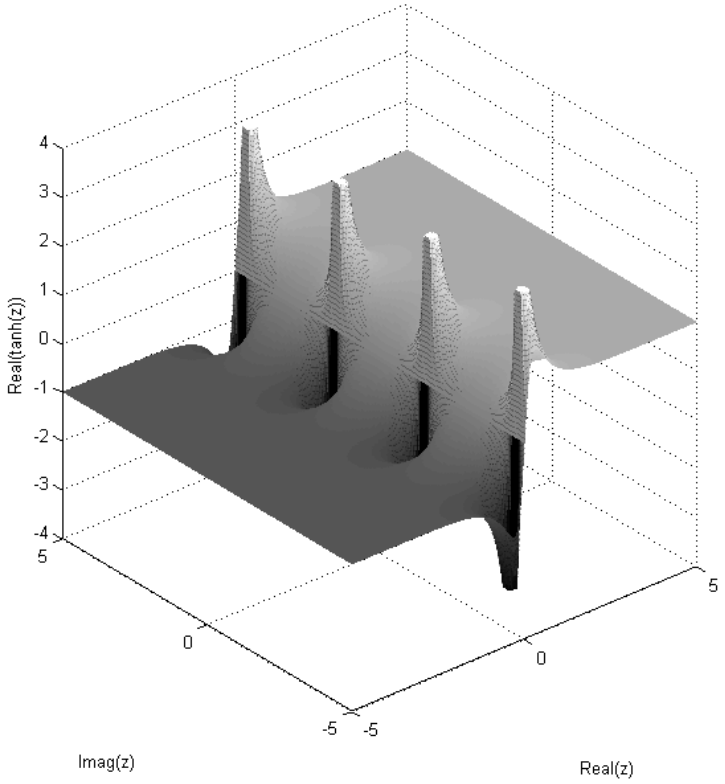


Figura 3.9 Parte real de la función compleja de tangente hiperbólico

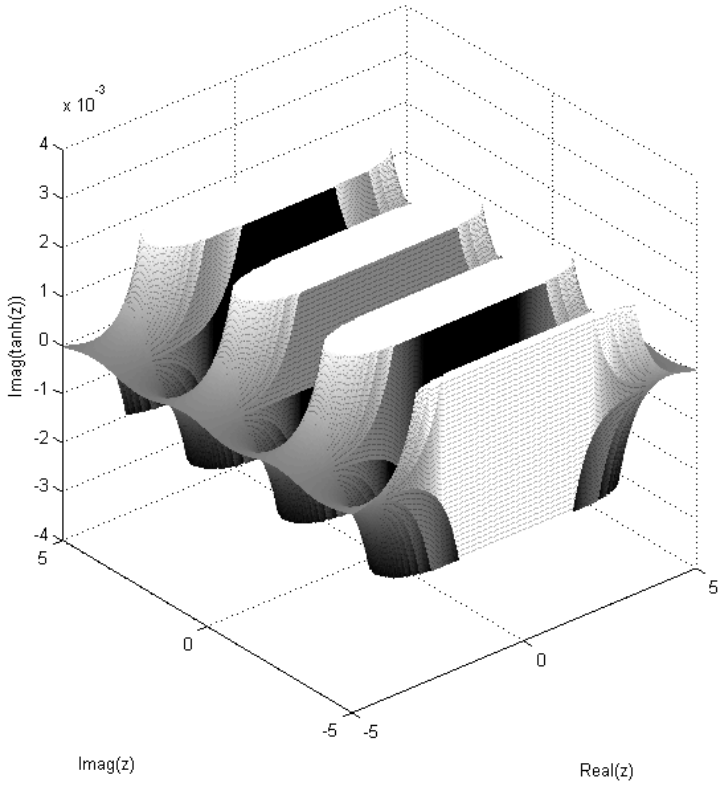


Figura 3.10 Parte imaginaria de la función compleja de tangente hiperbólico

4. Algoritmos de Entrenamiento

En este capítulo se aborda el tema de los algoritmos de entrenamiento, los cuales juegan un papel fundamental para el desarrollo y funcionalidad de las redes neuronales.

Se habla del proceso de aprendizaje y la clasificación de los diferentes algoritmos de acuerdo a su forma de ajustar los pesos sinápticos y su relación con el ambiente de trabajo. Posteriormente se introducen los algoritmos de primer orden, en el dominio complejo, basados en el gradiente descendiente y ciertas reglas diagramáticas que nos ayudan en este trabajo para poder obtener los términos de gradiente de una red neuronal de forma sencilla.

Por último, se desarrolla el algoritmo de entrenamiento en el dominio complejo de Levenberg-Marquardt recursivo, a partir de algoritmos de segundo orden, el cual es tema central de este trabajo.

4.1 El Proceso de Aprendizaje

La propiedad de mayor importancia que tiene una red neuronal es su capacidad de aprender del medio ambiente, y de mejorar su desempeño por medio de este aprendizaje. Una red neuronal aprende sobre su ambiente de trabajo a través de un proceso interactivo de ajustes a sus pesos sinápticos y niveles de *bias* e, idealmente, la red se vuelve más experta sobre su ambiente a cada iteración en su proceso de aprendizaje [1].

En el contexto de las redes neuronales artificiales, se define el aprendizaje como:

> *El proceso por el cual los parámetros libres de una red neuronal se adaptan a través de un proceso de estimulación del ambiente en el cual la red se encuentra integrada.*

Esta definición implica una secuencia de eventos que toman lugar dentro de la red:

- La red es **estimulada** por su ambiente.
- La red experimenta un **cambio** en sus parámetros libres como resultado de la estimulación.
- La red **responde** de una nueva forma al ambiente como resultado del cambio en sus parámetros.

Al conjunto de reglas bien definidas que solucionan el problema de aprendizaje se le denomina **algoritmo de entrenamiento** o de aprendizaje.

4.2 Clasificación del Proceso de Aprendizaje

Existen varios tipos de algoritmos de entrenamiento, cada uno con sus propias ventajas, desventajas y preferencia en aplicaciones. Los algoritmos de entrenamiento se clasifican en una de dos formas:

- Por el modo en el cual ajustan los pesos sinápticos de las neuronas (regla de aprendizaje)
- Por la forma en la cual la red neuronal se relaciona con el ambiente (paradigma de aprendizaje).

En el primer criterio se habla de la regla de aprendizaje que sigue el algoritmo de entrenamiento. A continuación se enumeran algunas reglas de entrenamiento:

- **Aprendizaje Basado en Corrección del Error.** La señal de salida de la neurona n, $y_n(k)$, es comparada con la respuesta objetivo $d_n(k)$ para producir una señal de error $e_n(k)$ definida por la ecuación:

$$e_n(k) = d_n(k) - y_n(k) \tag{4.1}$$

  Esta señal de error actúa como mecanismo de control, con el cual se aplican los ajustes correctivos a los pesos sinápticos de la neurona n para que la salida $y_n(k)$ se acerque al objetivo $d_n(k)$. Este trabajo se enfoca en este tipo de aprendizaje.
- **Aprendizaje Basado en Memoria.** La experiencia pasada se guarda explícitamente en una memoria de ejemplos entrada–salida $\{\mathbf{x_j}, d_j\}_{j=1}^{N}$ ($\mathbf{x_j}$ un vector de entrada). Luego, cuando la respuesta de una entrada no conocida $\mathbf{x_{test}}$ es requerida, el algoritmo responde recuperando y analizando los datos de entrenamiento que se encuentran dentro de una vecindad de $\mathbf{x_{test}}$.

- **Aprendizaje Hebbiano.** El ajuste del peso sináptico entre dos neuronas conectadas entre sí se incrementa selectivamente si ambas neuronas son activadas a la vez, y se decrementa si estas se activan en diferentes tiempos.
- **Aprendizaje Competitivo.** En este tipo de aprendizaje, las neuronas en cada capa de la red neuronal compiten entre sí para elegir una sola que alcance su máximo valor de salida. Esto se lleva a cabo por conexiones recurrentes de activación e inhibición por parte de las neuronas vecinas.
- **Aprendizaje de Boltzmann.** Esta regla de aprendizaje estocástica se basa en los principios de la mecánica estadística y la termodinámica. Las neuronas de la red constituyen una estructura recurrente que operan de forma binaria, siendo los estados de encendido y apagado denotados por los valores de $+1$ y -1. La red se caracteriza por una función de energía dada por la ecuación:

$$E = -\frac{1}{2}\sum_{j,n,j\neq n} w_{nj}x_n x_j \tag{4.2}$$

Donde x_j es el estado de la neurona j y w_{nj} el peso sináptico entre las neuronas n y j. La red opera eligiendo una neurona al azar, en una iteración dada del entrenamiento, y cambiando su estado de x_j a $-x_j$ en una temperatura (virtual) T siguiendo la regla de probabilidad:

$$P(x_n \to -x_j) = \frac{1}{1 + e^{-\Delta E_n \cdot T^{-1}}} \tag{4.3}$$

Aplicando esta regla repetidamente, la red eventualmente llega a un equilibrio térmico y el entrenamiento termina.

Por lo general, el ajuste del peso sináptico w_{nj} de la iteración k para la próxima iteración $k+1$ está determinado por la ecuación de actualización:

$$w_{nj}(k+1) = w_{nj}(k) + \Delta w_{nj}(k) \tag{4.4}$$

Donde Δw_{nj} es el término de corrección del peso w_{nj}, el cual queda definido por el algoritmo de entrenamiento.

En el segundo criterio se habla del paradigma de aprendizaje, que se refiere al modelo del ambiente en el cual opera la red neuronal. Los paradigmas de aprendizaje se enumeran a continuación:

- **Aprendizaje supervisado.** Se realiza por medio de un entrenamiento controlado por un agente externo, el cual determina la respuesta que debe generar la red a partir de una entrada determinada.
- **Aprendizaje no supervisado.** Carece de un agente externo y en su lugar se define una medida independiente de la tarea de la calidad de la representación que la red neuronal debe aprender, y los parámetros libres de la red son optimizados de acuerdo a esta medida.
- **Aprendizaje reforzado.** El aprendizaje de un mapeo entrada–salida se hace a través de la interacción continua con el ambiente para minimizar un índice de desempeño, por medio de un proceso de prueba y error.

4.3 Algoritmos de Entrenamiento Basados en Optimización y Algoritmo de Gradiente Descendiente

Tomando la función del error medio cuadrático (2.15) como función de costo $\mathcal{E}(\mathbf{w})$ en términos del vector de pesos sinápticos $\mathbf{w}$, se busca una solución óptima $\mathbf{w}^*$ tal que satisface la condición:

$$\mathcal{E}(\mathbf{w}^*) \leq \mathcal{E}(\mathbf{w}) \tag{4.5}$$

Esto es, se busca resolver un problema de optimización sin restricciones, donde se minimice la función de costo $\mathcal{E}(\mathbf{w})$ respecto al vector de pesos sinápticos $\mathbf{w}$. La condición necesaria de optimización es:

$$\nabla\mathcal{E}(\mathbf{w}^*) = 0 \tag{4.6}$$

Una clase de algoritmos de optimización sin restricciones diseñado para filtros adaptables se basan en la idea de un descenso local iterativo, en el cual se comienza con un estimado $\mathbf{w}(0)$, se genera una secuencia de vectores de pesos $\mathbf{w}(1), \mathbf{w}(2), \cdots$ tales que la función de costo $\mathcal{E}(\mathbf{w})$ es reducida en cada iteración, o bien:

$$\mathcal{E}\big(\mathbf{w}(k+1)\big) \leq \mathcal{E}\big(\mathbf{w}(k)\big) \tag{4.7}$$

Se busca que el algoritmo eventualmente converja a la solución óptima $\mathbf{w}^*$, lo cual no es seguro, ya que este puede quedar atrapado en un mínimo local, o no existir un mínimo global.

En el método de gradiente descendiente, el ajuste de los pesos se hace en la dirección de mayor inclinación, esto es en dirección opuesta al vector gradiente $\mathbf{g} \coloneqq \nabla\mathcal{E}(\mathbf{w})$. Luego, la ecuación (4.4) se puede reescribir para describir el algoritmo de gradiente descendiente:

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \eta \mathbf{g}(k), \qquad \eta > 0 \tag{4.8}$$

Donde $\Delta\mathbf{w}(k) = -\eta\mathbf{g}(k)$ es el término de corrección de $\mathbf{w}$ y η es el coeficiente de aprendizaje. Se dice que el algoritmo de gradiente descendiente es de primer orden dado que utiliza información de la primera derivada de la función de costo.

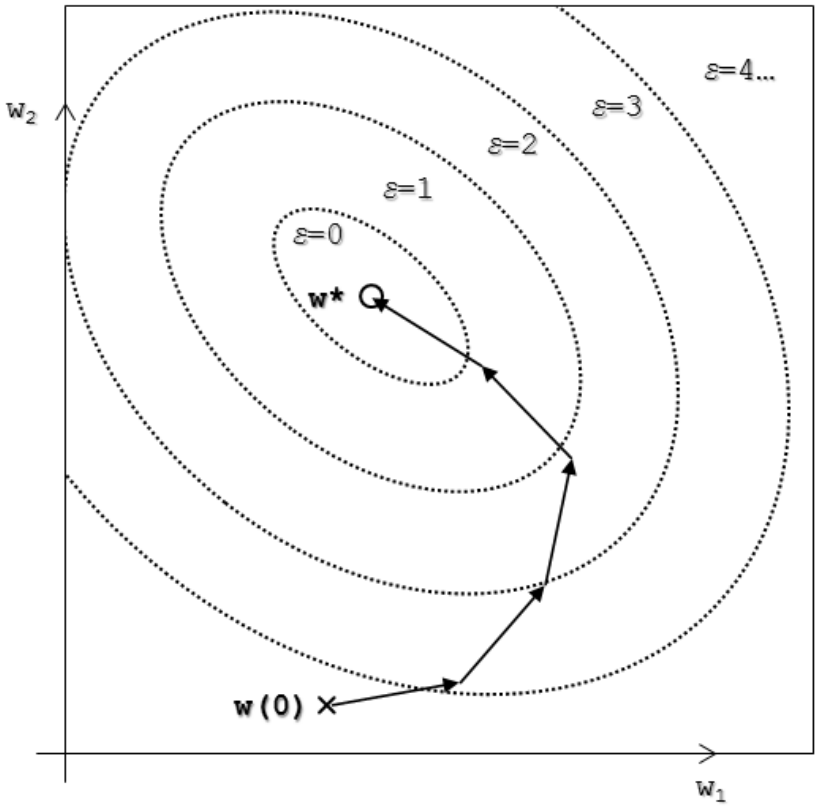


Figura 4.1 Ejemplo del algoritmo de gradiente descendiente en dos dimensiones. La función de costo está representada por los niveles $\mathcal{E} = 0, 1, 2, \ldots$

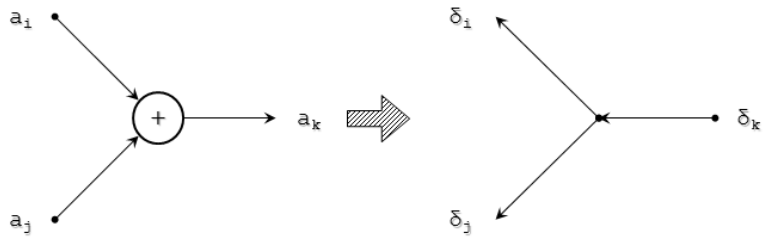
Este algoritmo se aplica a redes neuronales de una sola capa, la generalización para redes multicapa se conoce como algoritmo de *backpropagation*, el cual es discutido con mayor profundidad en [18]. En el caso de redes neuronales multicapa o con una arquitectura más compleja, los términos de gradiente **g** son difíciles de obtener, por lo que se recurre a herramientas y métodos que utilizan la representación diagramática de la red para obtenerlos de forma sencilla.

4.4 Método Diagramático y Red Adjunta

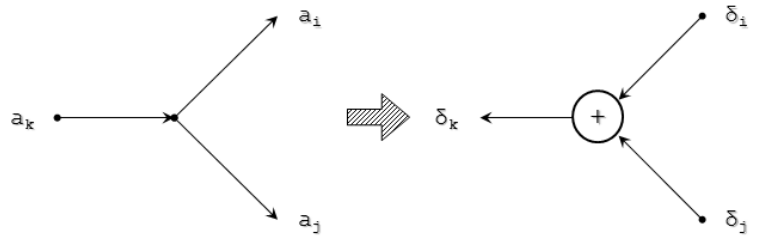
Siguiendo el principio de que existe una relación recíproca entre la propagación hacia delante de las señales de entrada a una red neuronal, y la propagación hacia atrás de sus términos de error [28], se presenta el acercamiento diagramático para derivar los términos de gradiente **g** mediante el uso de herramientas de teoría de gráficas.

Dada la representación diagramática de una red neuronal, se construye una red adjunta invirtiendo la dirección de flujo de las señales y aplicando las siguientes reglas:

- Los puntos de suma de señales se reemplazan por puntos de bifurcación:



- Los puntos de bifurcación de señal se reemplazan por puntos de suma:



- Funciones de una variable se reemplazan por sus derivadas:

- Las transmisiones de señal lineales e invariantes en el tiempo no tienen cambios:

- Funciones de múltiples variables se reemplazan por sus matrices Jacobianas:

- Operadores de retraso se reemplazan por operadores de adelanto:

- Señales de salida de la red se convierten en señales de entrada y viceversa:

Estas reglas no presentan un conjunto mínimo o exhaustivo de reglas diagramáticas. Una vez construida la red adjunta, los términos de gradiente se derivan de forma sencilla, ahorrándose así el proceso de derivación y aplicación de regla de la cadena, los cuales se vuelven difíciles para arquitecturas de redes que son muy complejas [19].

En [18], se utilizan estos métodos para derivar el algoritmo de backpropagation complejo. En este trabajo los términos de gradiente son necesarios en el algoritmo de entrenamiento de Levenberg–Marquardt, el cual se desarrolla más adelante.

4.5 Algoritmos de Entrenamiento Basados en Optimización de Segundo Orden

Método de Newton. El objetivo de este algoritmo se basa en minimizar una aproximación cuadrática de la función de costo $\mathcal{E}(\mathbf{w})$ utilizando su expansión de Taylor de segundo orden alrededor del punto $\mathbf{w}(k)$, dada por:

$$\Delta\mathcal{E}\big(\mathbf{w}(k)\big) \coloneqq \mathcal{E}\big(\mathbf{w}(k+1)\big) - \mathcal{E}\big(\mathbf{w}(k)\big) \approx \mathbf{g}^T(k)\Delta\mathbf{w}(k) + \frac{1}{2}\Delta\mathbf{w}^T(k)\mathbf{H}(k)\Delta\mathbf{w}(k) \tag{4.9}$$

Donde $\mathbf{H}$ es la matriz Hessiana de $\mathcal{E}$, descrita en (2.28). Diferenciando la ecuación (4.9) respecto a $\Delta\mathbf{w}$, el cambio $\Delta\mathcal{E}$ se minimiza cuando:

$$\mathbf{g}(k) + \mathbf{H}(k)\Delta\mathbf{w}(k) = 0 \tag{4.10}$$

Resolviendo (4.10) se tiene:

$$\Delta\mathbf{w}(k) = -\mathbf{H}^{-1}(k)\mathbf{g}(k) \tag{4.11}$$

O bien, se puede reescribir la ecuación (4.4) en términos de (4.11):

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \mathbf{H}^{-1}(k)\mathbf{g}(k) \tag{4.12}$$

En general, el método de Newton converge de manera asintótica sin presentar el comportamiento oscilatorio del método de gradiente descendiente y en menos iteraciones [2]. Sin embargo, se deben tomar en cuenta las siguientes suposiciones:

- La función de costo $\mathcal{E}$ debe ser dos veces diferenciable respecto a $\mathbf{w}$.
- La matriz Hessiana $\mathbf{H}$ debe ser invertible y definida positiva.

La segunda suposición genera problemas dado que no existe una restricción que asegure que la matriz Hessiana sea invertible y positiva definida para cada iteración del algoritmo, por lo que es necesario hacer modificaciones a este método.

Método cuasi–Newton. El método cuasi–Newton es básicamente un método de gradiente de primer orden descrito por la ecuación de actualización de pesos sinápticos:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta \mathbf{s}(k) \tag{4.13}$$

$$\mathbf{s}(k) = -S(k)\,\mathbf{g}(k) \tag{4.14}$$

Donde $\mathbf{s}$ es un vector de dirección definido en términos del gradiente $\mathbf{g}$ y S es una matriz definida positiva que se ajusta en cada iteración, de tal forma que el vector de dirección $\mathbf{s}$ sea aproximado a la dirección del método e Newton de segundo orden (4.12).

Este método utiliza información de segundo orden, i.e. curvatura de la superficie $\mathcal{E}(\mathbf{w})$, sin requerir calcular la matriz Hessiana o su inversa. Para esto, se utilizan dos iteraciones sucesivas en k y $k+1$ del vector gradiente $\mathbf{g}$. Se define el vector $\mathbf{q}$ como:

$$\mathbf{q}(k) = \mathbf{g}(k+1) - \mathbf{g}(k) \tag{4.15}$$

Y de la ecuación (4.4) se obtiene el cambio en el vector de pesos sinápticos:

$$\Delta\mathbf{w}(k) = \mathbf{w}(k+1) - \mathbf{w}(k) \tag{4.16}$$

Luego, la curvatura de la función de error se puede aproximar por medio de la siguiente ecuación:

$$\mathbf{q}(k) \approx \left(\frac{\partial}{\partial \mathbf{w}}\mathbf{g}(k)\right)\Delta\mathbf{w}(k) \tag{4.17}$$

En particular, dado un incremento linealmente independiente en el vector de pesos $\mathbf{w}$, esto es $\Delta\mathbf{w}(0), \Delta\mathbf{w}(1), \cdots, \Delta\mathbf{w}(k-1)$, y su respectivo incremento de gradientes $\mathbf{q}(0), \mathbf{q}(1), \cdots, \mathbf{q}(k-1)$, la matriz Hessiana se aproxima por medio de la siguiente ecuación [2]:

$$\mathbf{H}(k) \approx [\mathbf{q}(0) \quad \mathbf{q}(1) \quad \cdots \quad \mathbf{q}(k-1)][\Delta\mathbf{w}(0) \quad \Delta\mathbf{w}(1) \quad \cdots \quad \Delta\mathbf{w}(k-1)]^{-1} \tag{4.18}$$

Y su inversa por medio de la ecuación:

$$\mathbf{H}^{-1}(k) \approx [\Delta\mathbf{w}(0) \quad \Delta\mathbf{w}(1) \quad \cdots \quad \Delta\mathbf{w}(k-1)][\mathbf{q}(0) \quad \mathbf{q}(1) \quad \cdots \quad \mathbf{q}(k-1)]^{-1} \tag{4.19}$$

Con (4.19) se calcula la matriz S que completa el cálculo de las ecuaciones (4.13), (4.14).

4.6 Algoritmo de Entrenamiento de Levenberg-Marquardt

El método de Levenberg–Marquardt es un algoritmo de optimización que se encuentra en un punto intermedio entre los algoritmos de primer orden como gradiente descendiente y algoritmos de segundo orden como el método de Newton. Este método fue diseñado para minimizar funcionales cuadráticas de funciones no lineales [29], [30].

Algoritmo de Entrenamiento de Levenberg–Marquardt Fuera de Línea. Suponemos que la función de costo está dada por:

$$\mathcal{E}(\mathbf{w}) = \sum_{j=1}^{L} e_j^2(\mathbf{w}) \tag{4.20}$$

Donde e_j es el j–ésimo término del vector de errores $\mathbf{E}$, en función de los pesos sinápticos $\mathbf{w}$. El vector gradiente de (4.20) y su matriz Hessiana están dadas por:

$$\mathbf{g}(k) = \mathbf{J}^T(k)\mathbf{E}(\mathbf{w}) \tag{4.21}$$

$$\mathbf{H}(k) = \mathbf{J}^T(k)\mathbf{J}(k) + S(\mathbf{w}) \tag{4.22}$$

Donde $\mathbf{J}$ es la matriz Jacobiana descrita en (2.29) y la matriz S se calcula por la ecuación:

$$S(\mathbf{w}) = \sum_{j=1}^{L} e_j^2(\mathbf{w}) \cdot \boldsymbol{\nabla}^2 e_j(\mathbf{w}) \tag{4.23}$$

Para el método de Newton, se supone $S(\mathbf{w}) \approx 0$, entonces la actualización de los pesos (4.11) en términos de (4.21) y (4.22) está dada por:

$$\Delta\mathbf{w}(k) = -[\mathbf{J}^T(k)\mathbf{J}(k)]^{-1}\mathbf{J}^T(k)\mathbf{E}(\mathbf{w}) \tag{4.24}$$

La ecuación (4.24) es una variante del método de Newton, mejor conocido como el método de Gauss–Newton.

A (4.24) se le agrega un término de regularización, dependiente de un parámetro ρ, para obtener la actualización de los pesos sinápticos usado en el método de Levenberg–Marquardt:

$$\Delta\mathbf{w}(k) = -[\mathbf{J}^T(k)\mathbf{J}(k) + \rho I]^{-1}\mathbf{J}^T(k)\mathbf{E}(\mathbf{w}) \tag{4.25}$$

Donde I es la matriz identidad. El parámetro ρ varía en cada iteración y depende del comportamiento de la función de costo (4.20):

- Si la función de costo aumenta, entonces ρ es multiplicado por un coeficiente β.
- Si la función de costo disminuye, entonces ρ es dividido por el mismo coeficiente β.

Se observa que para valores grandes de ρ, la ecuación (4.25) se asemeja al método de gradiente descendiente; mientras que para valores pequeños de ρ, se asemeja al método de Newton.

La clave de este algoritmo se encuentra en el cálculo de la matriz Jacobiana, ya que esta se puede obtener por el método diagramático mostrado, lo cual ahorra el cálculo de la matriz Hessiana.

Por último, se asume que el algoritmo de Levenberg–Marquardt ha convergido cuando la función de costo (4.20) o la norma del gradiente (4.21) han sido reducidas dentro de un intervalo admisible [17].

Entrenamiento Recursivo. El entrenamiento fuera de línea se refiere a cuando la red es ajustada respecto a una función de costo definida para un lote de datos ya obtenidos.

En el entrenamiento recursivo (o en línea) la actualización de los pesos sinápticos se realiza en cada iteración del proceso al cual se le aplica la red neuronal, con base en el cálculo instantáneo de la función de costo y utilizando los datos de la red que se obtienen en cada momento [31], [32].

Es de gran importancia para este trabajo el desarrollo del algoritmo de Levenberg–Marquardt de forma recursiva, que es requerido para la aplicación de redes en esquemas de identificación y control de sistemas dinámicos.

En este tipo de entrenamiento se utiliza un vector de errores instantáneos, definido por [33]:

$$\mathbf{E}\big(\mathbf{w}(k)\big) = \mathbf{d}(k) - \mathbf{y}\big(k; \mathbf{w}(k)\big) \tag{4.26}$$

Donde $\mathbf{d}$ es el vector objetivo y $\mathbf{y}$ el vector de salidas de la red neuronal en el instante k. Por ejemplo, el algoritmo de gradiente descendiente recursivo está dado por la siguiente ecuación:

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \lambda(k)\mathbf{g}\big(\mathbf{w}(k)\big), \qquad \lambda(k) > 0 \quad \forall k \tag{4.27}$$

Donde el coeficiente de aprendizaje λ puede variar en cada iteración.

Algoritmo de Error de Predicción Recursivo. El algoritmo de error de predicción recursivo (**EPR**) es un algoritmo de mínimos cuadrados de segundo orden, el cual se utiliza en modelos con parámetros lineales, pero puede extenderse a modelos no lineales por medio de una linealización respecto a los pesos sinápticos instantáneos $\mathbf{w}$ [32].

De esta linealización resulta un método de optimización equivalente al método de Gauss–Newton, con una aproximación de la matriz Hessiana que se hace de forma recursiva para cada iteración, la cual está dada por la siguiente ecuación:

$$R(k) = \alpha(k)R(k-1) + \big(1-\alpha(k)\big)\Big(\mathbf{g}\big(\mathbf{w}(k)\big)\mathbf{g}^T\big(\mathbf{w}(k)\big)\Big) \tag{4.28}$$

Donde $\alpha(k) > 0 \ \forall k$ es un coeficiente de actualización y $R(0) = R_0$ es la condición inicial de R. Cabe mencionar que la matriz R es simétrica positiva definida, por lo que es invertible para cualquier iteración k [33].

Luego, la actualización de pesos, de forma recursiva, del método **EPR** está dada por:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + R^{-1}(k)\mathbf{g}\big(\mathbf{w}(k)\big)\mathbf{E}\big(\mathbf{w}(k)\big) \tag{4.29}$$

Inversión de la Matriz R. Calcular la matriz inversa de R es computacionalmente costoso conforme aumenta la dimensión del vector de pesos sinápticos $\mathbf{w}$. Para esto se utiliza la identidad matricial de Woodbury, también conocida como el lema de inversión de matrices [34]:

$$(A+UCV)^{-1} = A^{-1} - A^{-1}U(C^{-1}+VA^{-1}U)^{-1}VA^{-1} \tag{4.30}$$

Tomando $C = I$, $A = \alpha(k)R(k-1)$, $U = 1-\alpha(k)$, $V = \mathbf{g}\big(\mathbf{w}(k)\big)\mathbf{g}^T\big(\mathbf{w}(k)\big)$, y definiendo una nueva matriz $P \coloneqq R^{-1}$, la matriz inversa de R se puede calcular a partir de (4.28) con ayuda de la identidad (4.30):

$$P(k) = \frac{1}{\alpha(k)}\left[P(k-1) - \frac{P(k-1)\mathbf{g}\big(\mathbf{w}(k)\big)\mathbf{g}^T\big(\mathbf{w}(k)\big)P(k-1)}{\alpha(k) + \mathbf{g}^T\big(\mathbf{w}(k)\big)P(k)\mathbf{g}\big(\mathbf{w}(k)\big)}\right] \tag{4.31}$$

Esta matriz P puede interpretarse como la matriz de covarianza de los pesos sinápticos estimados $\mathbf{w}$ de la red neuronal.

Algoritmo de Entrenamiento de Levenberg–Marquardt Recursivo. Agregando el término de regularización a la ecuación (4.28), se obtiene la siguiente ecuación para el cálculo de la matriz R:

$$R(k) = \alpha(k)R(k-1) + \big(1-\alpha(k)\big)\big(\mathbf{g}\big(\mathbf{w}(k)\big)\mathbf{g}^T\big(\mathbf{w}(k)\big) + \rho I\big) \tag{4.32}$$

Esta ecuación no presenta una forma adecuada para aplicar la identidad de inversión, por lo que el parámetro ρ se introduce en uno de los elementos de la diagonal de la matriz $\mathbf{g}\big(\mathbf{w}(k)\big)\mathbf{g}^T\big(\mathbf{w}(k)\big)$, quedando la ecuación (4.32) expresada de la forma:

$$R(k) = \alpha(k)R(k-1) + \big(1-\alpha(k)\big)\big(\mathbf{g}\big(\mathbf{w}(k)\big)\mathbf{g}^T\big(\mathbf{w}(k)\big) + \rho Z\big) \tag{4.33}$$

Donde Z es una matriz cuadrada de ceros, con un único elemento no cero en su diagonal dado por:

$$z_{ii} = \begin{cases} 1 & \text{si} \quad i = k \bmod(N_{\mathbf{w}}) + 1 \\ 0 & \text{de otra forma} \end{cases} \tag{4.34}$$

Luego, introduciendo nuevas variables, la ecuación (4.33) se reescribe de la siguiente forma:

$$R(k) = \alpha(k)R(k-1) + \big(1-\alpha(k)\big)\Big(\Omega\big(\mathbf{w}(k)\big)\Lambda^{-1}\Omega^T\big(\mathbf{w}(k)\big)\Big) \tag{4.35}$$

Donde Ω y Λ están dados por:

$$\Omega^T\big(\mathbf{w}(k)\big) = \begin{bmatrix} & \mathbf{g}^T\big(\mathbf{w}(k)\big) & \\ 0 & \cdots \quad 1 \quad \cdots & 0 \end{bmatrix} \tag{4.36}$$

$$\Lambda^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & \rho \end{bmatrix} \tag{4.37}$$

El elemento unitario de la matriz Ω^T se encuentra en la i–ésima posición, la cual se calcula de igual forma que en la ecuación (4.34). Aplicando la identidad matricial (4.30) a la ecuación (4.35) se obtienen las ecuaciones:

$$P(k) = \frac{1}{\alpha(k)}\big[P(k-1) - P(k-1)\Omega\big(\mathbf{w}(k)\big)\mathrm{S}^{-1}(k)\Omega^T\big(\mathbf{w}(k)\big)P(k-1)\big] \tag{4.38}$$

$$S(k) = \alpha(k)\Lambda + \Omega^T\big(\mathbf{w}(k)\big)P(k-1)\Omega\big(\mathbf{w}(k)\big) \tag{4.39}$$

Donde $P(0) = P_0$ es la condición inicial de P. Así, la ecuación de actualización de los pesos sinápticos para el algoritmo de entrenamiento de Levenberg–Marquardt recursivo está dado por la siguiente ecuación:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + P(k)\mathbf{g}\big(\mathbf{w}(k)\big)\mathbf{E}\big(\mathbf{w}(k)\big) \tag{4.40}$$

Luego, el algoritmo completo está dado por las ecuaciones (4.36)–(4.40). La principal ventaja que tiene este algoritmo respecto al algoritmo **EPR** es que en este únicamente se tiene que invertir la matriz S, la cual tiene dimensión (2×2), por lo que es computacionalmente más barato [33], [35].

5. Red Neuronal Compleja Recurrente y el Algoritmo de Levenberg–Marquardt Complejo

En este capítulo se introduce la topología de una Red Neuronal Compleja Recurrente con una función de activación compleja construida, al igual que sus términos de gradiente y su algoritmo de entrenamiento, que posteriormente será utilizada para resolver los problemas de identificación y control de sistemas mecánicos no lineales y oscilatorios.

5.1 Red Neuronal Compleja Recurrente con Función de Activación Construida

En [18] se presentaron dos diferentes topologías de Red Neuronal Compleja Recurrente (**CVRNN**) con diferentes funciones de activación, de las cuales la segunda es una función compleja construida como la mostrada en la ecuación (3.10). En [18] se mostró que esta presenta un mejor desempeño para las tareas de identificación y control de sistemas dinámicos, debido a que no presenta singularidades en su dominio; razón por la cual se decide utilizar dicha topología en este trabajo.

Sea una **CVRNN** con descripción matemática dada por las siguientes ecuaciones:

$$X(k+1) = AX(k) + BU(k) \tag{5.1}$$

$$Z(k) = \Gamma[X_{\mathrm{Re}}(k)] + i\Gamma[X_{\mathrm{Im}}(k)] \tag{5.2}$$

$$V(k) = CZ(k) \tag{5.3}$$

$$Y(k) = \Phi[V(k)] \tag{5.4}$$

Esta **CVRNN** se presenta en una forma canónica de Jordan de un sistema dinámico en diferencias, por lo que tiene un mínimo número de parámetros sujetos a

entrenamiento, es apta para una total integración en paralelo, y presenta propiedades de controlabilidad, observabilidad, y condiciones de estabilidad [36].

Esta red neuronal cuenta con una capa de entrada, una capa oculta recurrente con un comportamiento cuasi-lineal, debido a las funciones de activación no lineales; y una capa de salida. Su representación diagramática como la mostrada en la Figura 5.1.

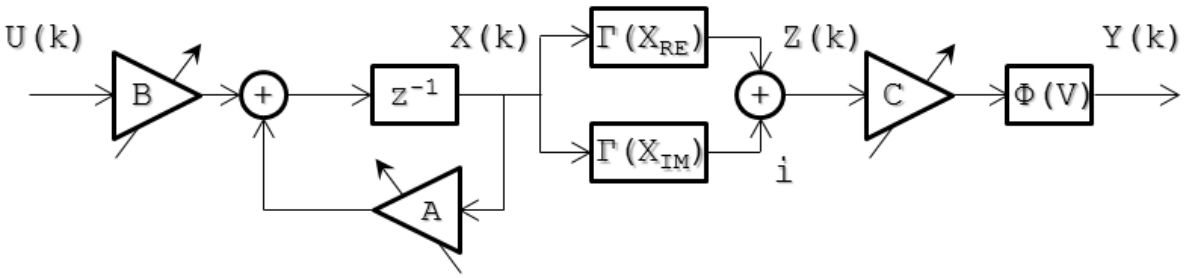


Figura 5.1 Diagrama del modelo de la CVRNN con función de activación construida

Donde $X \in \mathbb{C}^n$ es el vector de estados internos, $Z \in \mathbb{C}^n$ es el vector de estados de la capa oculta, $U \in \mathbb{R}^m$ es el vector de señales de entrada, $Y \in \mathbb{R}^L$ es el vector de señales de salida, y n, m, L son el número de estados, entradas y salidas de la red neuronal respectivamente. $A \in \mathbb{C}^{n\times n}$, $B \in \mathbb{C}^{n\times m}$ y $C \in \mathbb{C}^{L\times n}$ son las matrices de pesos sinápticos de realimentación, entrada y salida respectivamente, y constituyen los parámetros de la red neuronal sujetos a entrenamiento.

La matriz A tiene una forma canónica de Jordan, lo que significa que los valores propios de la red neuronal se encuentran en forma de bloques sobre su diagonal [36]. Los valores propios reales se presentan de forma sencilla, mientras que los valores propios complejos vienen en pares de números complejos conjugados en un solo bloque [37], [38].

Los elementos de las matrices B y C también vienen como valores reales sencillos o como pares de números complejos conjugados, y sus posiciones dentro de las matrices se encuentran relacionados con los elementos de A.

Los bloques A_j que constituyen los elementos diagonales de la matriz A deben cumplir con la siguiente condición de estabilidad:

$$|A_j| < 1, \qquad j = 1,2,\cdots,N_A \tag{5.5}$$

Donde N_A es el número de bloques de Jordan.

Dado que los valores del vector X son complejos, la función de activación $\Gamma: \mathbb{C}^n \to \mathbb{C}^n$ es una función vectorial valuada en el dominio complejo. En cambio, debido a los pares de complejos conjugados y su disposición en la matriz C, el producto $CZ(k)$ produce un vector real, y la función $\Phi: \mathbb{R}^L \to \mathbb{R}^L$ es una función vectorial valuada en el dominio real.

Las funciones de activación elegidas para esta topología están dadas por $\Gamma[\cdot] = \Phi[\cdot] = \tanh(\cdot)$. Luego, la función de activación construida para la capa oculta de la red neuronal está dada por la ecuación:

$$\varphi(z) = \tanh(\mathrm{Re}(z)) + i\tanh(\mathrm{Im}(z)), \qquad z \in \mathbb{C} \tag{5.6}$$

Se observa que esta función de activación no contiene singularidades en su dominio y es similar a la utilizada en [39] y [40].

El algoritmo de entrenamiento de Levenberg–Marquardt en el dominio complejo (**CVLM**), para cualquier vector de pesos W (el cual puede ser A, B o C), está descrito por la siguiente ecuación de actualización:

$$\begin{gathered} W(k+1) = W(k) + P(k)\cdot \nabla Y[W(k)]\cdot E(k) \\ |W_j| < W_0 \quad \forall j = 1, \cdots, N_W \end{gathered} \tag{5.7}$$

Donde W_0 es una región acotada para los valores de W_j y N_W es el número de pesos sinápticos del vector W [37], [38].

Para obtener los términos de gradiente de la salida de la red neuronal Y_N, respecto a un vector de pesos W, de forma rápida y sencilla, se recurre a las reglas diagramáticas para obtener una red adjunta [19]. La Figura 5.2 muestra el diagrama de la red adjunta de la red neuronal representada por el diagrama de la Figura 5.1.

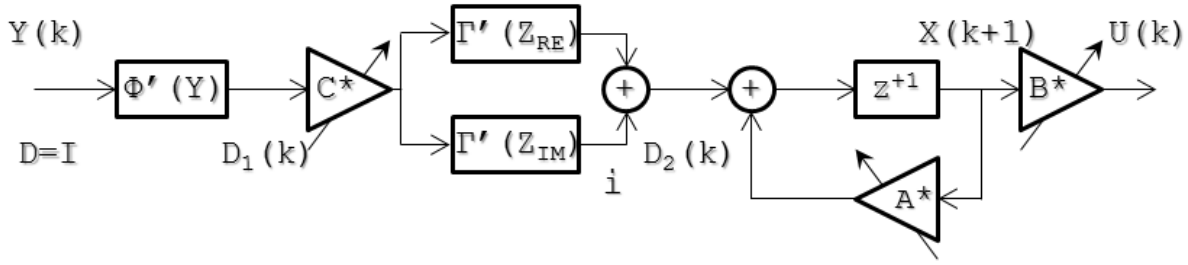


Figura 5.2 Diagrama de la red adjunta de la CVRNN con función de activación construida

Los términos de gradiente ∇Y de la **CVRNN** con función de activación construida (5.6) están descritos por las siguientes ecuaciones:

$$D_1(k) = \Phi'[Y(k)] \cdot D \tag{5.8}$$

$$D_2(k) = (\Gamma'[Z_{\mathrm{Re}}(k)] \cdot \mathrm{Re}(C^*) + i\Gamma'[Z_{\mathrm{Im}}(k)] \cdot \mathrm{Im}(C^*)) \cdot D_1(k) \tag{5.9}$$

$$\boldsymbol{\nabla} Y[C(k)] \coloneqq \boldsymbol{\nabla}_{C(k)} Y(k) = D_1(k) \cdot Z^*(k) \tag{5.10}$$

$$\boldsymbol{\nabla} Y[A(k)] \coloneqq \boldsymbol{\nabla}_{A(k)} Y(k) = D_2(k) \cdot X^*(k) \tag{5.11}$$

$$\boldsymbol{\nabla} Y[B(k)] \coloneqq \boldsymbol{\nabla}_{B(k)} Y(k) = D_2(k) \cdot U^*(k) \tag{5.12}$$

Donde el superíndice $(*)$ denota un vector transpuesto, complejo conjugado, y $D = I$ es una matriz identidad de entrada para la red adjunta [39], [40].

La matriz P se calcula de forma recursiva por medio de la ecuación:

$$P(k) = \alpha^{-1}\left[P(k-1) - P(k-1) \cdot \Omega_{W(k)} \cdot S_{W(k)}^{-1} \cdot \Omega_{W(k)}^* \cdot P(k-1)\right] \tag{5.13}$$

Donde las matrices Ω_W y S_W están dadas por:

$$\Omega_{W(k)}^* = \begin{bmatrix} & \boldsymbol{\nabla} Y^*[W(k)] & \\ 0 & \cdots \quad 1 \quad \cdots & 0 \end{bmatrix} \tag{5.14}$$

$$S_{W(k)} = \alpha\Lambda + \Omega_{W(k)}^* \cdot P(k-1) \cdot \Omega_{W(k)} \tag{5.15}$$

$$\Lambda^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & \rho \end{bmatrix} \tag{5.16}$$

Las matrices P y S_W tienen dimensiones $(N_W \times N_W)$ y (2×2) respectivamente. La matriz Ω_W^* tiene dimensión $(2 \times N_W)$ y su segunda fila consta de $(N_W - 1)$ elementos cero, y un elemento unitario en la posición i, la cual se calcula con la ecuación:

$$i = k \bmod N_W + 1 \tag{5.17}$$

Los parámetros del algoritmo deben seguir las restricciones [36]:

$$\begin{gathered} 10^{-6} \leq \rho \leq 10^{-4}, \qquad 0.97 \leq \alpha \leq 1.00 \\ 10^3 \leq P(0) \leq 10^6 \end{gathered} \tag{5.18}$$

La red descrita por (5.1)–(5.4), con función de activación (5.6), y el algoritmo descrito por la ecuación de actualización de pesos (5.7), términos de gradiente (5.8)–(5.12), y las ecuaciones (5.13)–(5.18) conforman en su totalidad a la Red Neuronal Compleja Recurrente con función de activación construida, con el entrenamiento de Levenberg–Marquardt recursivo en el dominio complejo.

6. Identificación con Redes Neuronales

En este capítulo se explica el problema de identificación dentro del contexto de sistemas dinámicos y se propone un esquema de identificación utilizando redes neuronales recurrentes en el dominio complejo, entrenadas con el algoritmo de Levenberg-Marquardt.

Luego, se presentan dos modelos dinámicos de plantas mecánicas, no lineales y oscilatorias, una de tipo **SISO** y la otra de tipo **MIMO**; los cuales se utilizan como ejemplo para validar el esquema de identificación propuesto.

El esquema de identificación propuesto utiliza una **CVRNN** con la topología con función de activación construida, como la presentada en el capítulo anterior. Luego, se presentan los resultados de las simulaciones realizadas para ambas plantas y se realiza un análisis comparativo entre los algoritmos de entrenamiento en el dominio complejo de Backpropagation (**CVBP**) y Levenberg–Marquardt (**CVLM**).

6.1 El Problema de Identificación

El problema de identificación consiste en encontrar la aproximación a un modelo de una planta desconocida, dadas las señales de entrada y salida. La planta a identificar se puede ver como una caja negra, a la cual se le aplicas señales conocidas en la entrada, y luego se mide su respuesta [41].

Posteriormente se utiliza un algoritmo de identificación ya sea para estimar los parámetros de la planta, su estructura en general o su comportamiento dinámico [3]. La Figura 6.1 muestra un esquema general de identificación de sistemas por el método de corrección del error.

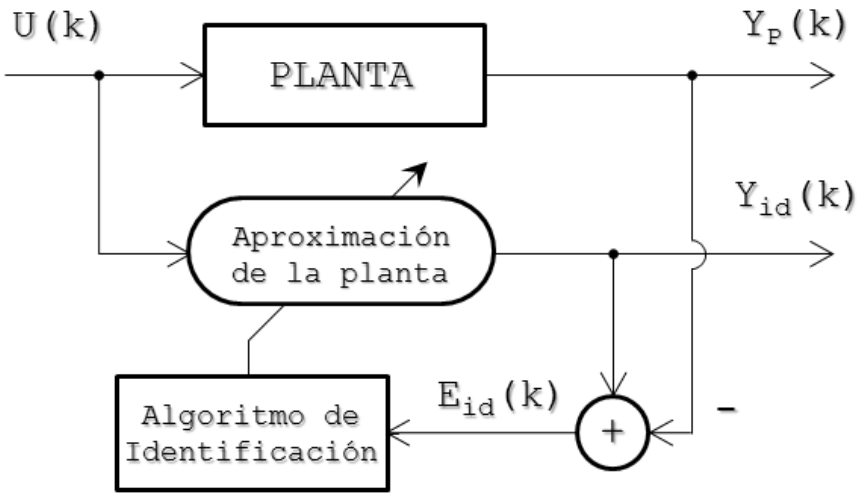


Figura 6.1 Un esquema general de identificación de sistemas con el método de corrección del error

En el contexto de este trabajo, se utiliza una **CVRNN** junto con el algoritmo de entrenamiento **CVLM** como identificador de una planta no lineal, oscilatoria y desconocida. Un esquema del identificador se muestra en la Figura 6.2. Una señal oscilatoria de prueba se alimenta a la planta desconocida y a la red neuronal para producir una respuesta de la planta, la cual será la señal objetivo o deseada, y una respuesta de la neurona respectivamente. Ambas señales de salida se comparan para producir una señal de error de identificación, dado por la ecuación:

$$E_{id}(k) = Y_N(k) - Y_P(k) \tag{6.1}$$

Donde Y_N es la salida de la red neuronal y Y_P es la salida de la planta. Esta señal es alimentada al algoritmo de entrenamiento de la red neuronal, el cual ajusta sus pesos sinápticos. Se espera que, conforme el error de identificación se acerca a cero, la red neuronal converja a un modelo estimado de la planta original.

A continuación se presentan los modelos dos plantas no lineales y oscilatorias, de tipo **SISO** (*single–input, single–output*) y **MIMO** (*multiple–input, multiple–output*), cual está sujeta al esquema de identificación por medio de una **CVRNN**.

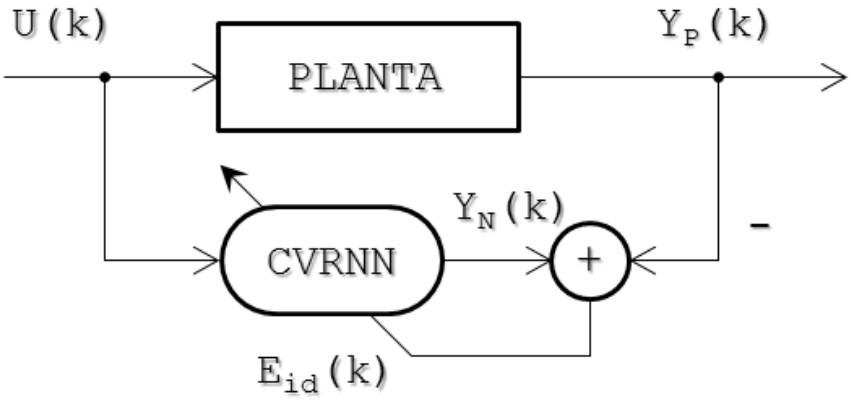


Figura 6.2 Esquema de la identificación de sistemas no lineales por medio de CVRNN

6.2 Brazo Robótico No Lineal de Un Grado de Libertad

La primera planta propuesta consiste en el modelo idealizado de un brazo robótico de un grado de libertad con articulación flexible, la cual genera un comportamiento no lineal. La flexibilidad de este tipo de articulaciones se debe al uso de motores de tipo *harmonic drive*, los cuales contienen un mecanismo de engranajes flexibles que permiten mayor transmisión de par de torsión en un tamaño compacto. Un diagrama de este tipo de articulación se presenta en la Figura 6.3.

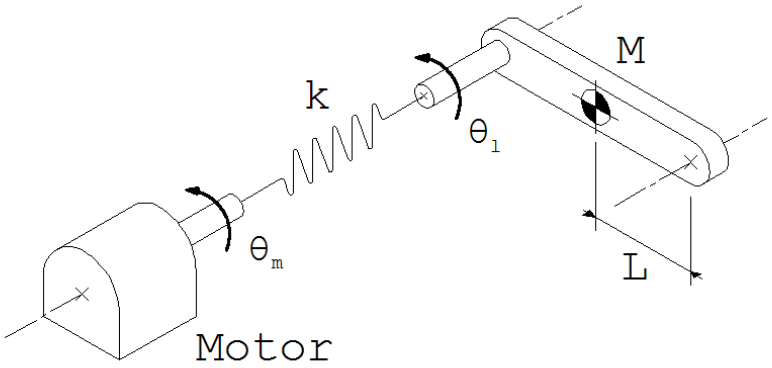


Figura 6.3 Modelo idealizado de un brazo robótico de un grado de libertad con articulación flexible

El modelo idealizado consiste de un actuador conectado a una carga (el eslabón del brazo robótico) por medio de un resorte torsional, el cual representa la flexibilidad de la articulación. Se considera al par de motor y a la posición angular de eslabón como

las señales de entrada $u(t)$ y de salida $y(t)$ respectivamente, por lo cual se tiene un sistema de tipo **SISO**.

Las ecuaciones que describen el movimiento de esta planta son:

$$J_l\ddot{\theta}_l(t) + B_l\dot{\theta}_l(t) + MgL\sin\theta_l(t) + k\big(\theta_l(t) - \theta_m(t)\big) = 0 \tag{6.2}$$

$$J_m\ddot{\theta}_m(t) + B_m\dot{\theta}_m(t) - k\big(\theta_l(t) - \theta_m(t)\big) = u(t) \tag{6.3}$$

Donde J_l, J_m son los momentos de inercia del eslabón y el motor; B_l, B_m son los coeficientes de amortiguamiento del eslabón y el motor; k es el coeficiente de rigidez del resorte torsional, M es la masa del eslabón, L es la longitud entre la flecha del motor y el centro de masa del eslabón; y θ_l, θ_m son las posiciones angulares del eslabón y el rotor del motor respectivamente. Así, obtenemos un sistema oscilatorio descrito por dos ecuaciones diferenciales, no lineales, de segundo orden.

Para poder utilizar redes neuronales discretas para identificar esta planta, las señales continuas de entrada y de salida se discretizan con un tiempo de muestreo (τ).

6.3 Brazo Robótico No Lineal de Dos Grados de Libertad

La segunda planta propuesta consiste en un brazo robótico de dos grados de libertad con articulaciones flexibles. Un diagrama de esta planta se muestra en la Figura 6.4.

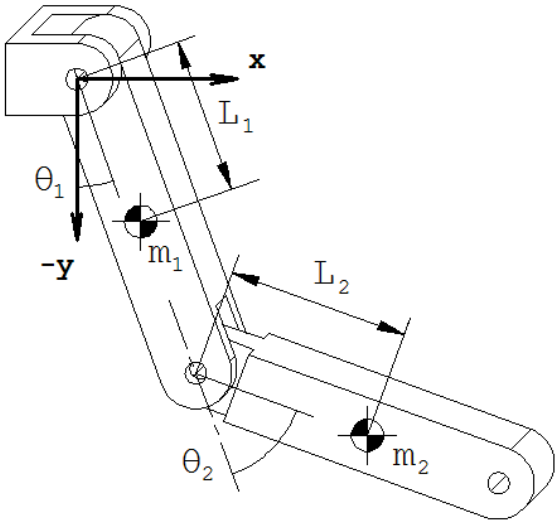


Figura 6.4 Modelo idealizado de un brazo robótico de dos grado de libertad con articulaciones flexibles

Cada articulación consiste de un actuador conectado a una carga por medio de un resorte torsional, el cual representa la flexibilidad de la articulación, mencionado en [42]. Se realizan dos suposiciones para obtener un modelo matemático simplificado de la planta:

- El momento de inercia del rotor de cada articulación es simétrico respecto a su eje de rotación. De esta manera los términos gravitacionales, de velocidad del rotor y del centro de masa de cada eslabón serán independientes de la posición del rotor.
- La energía cinética actuando sobre el rotor de cada eslabón se debe únicamente a su rotación; y su movimiento es estrictamente de rotación respecto a su marco inercial.

La flexibilidad en cada articulación se debe, al igual que en la planta utilizada en el capítulo anterior, a un mecanismo de tipo *harmonic drive*, como el mostrado en la Figura 6.5.

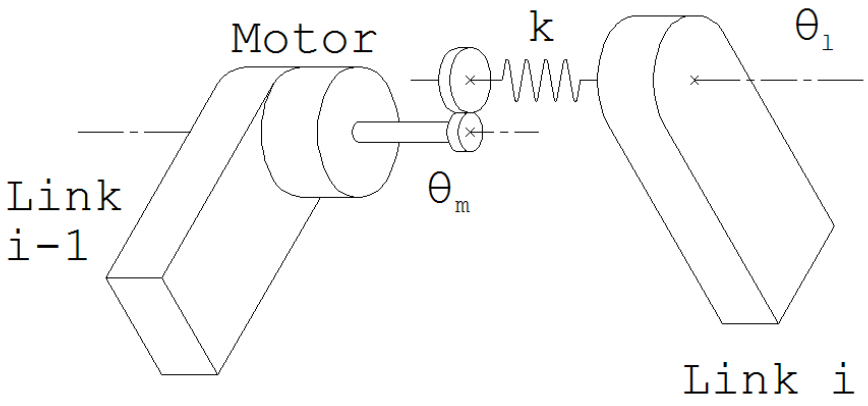


Figura 6.5 Articulación flexible

Se considera al par de los motores de cada articulación y a sus posiciones angulares como las señales de entrada $u(t)$ y de salida $y(t)$ respectivamente, por lo cual se tiene un sistema de tipo **MIMO**. Bajo estas suposiciones, las ecuaciones que describen el movimiento de la planta mecánica, no lineal y oscilatoria de dos grados de libertad están dadas por:

$$D(\theta_L)\ddot{\theta}_L + C\left(\theta_L, \dot{\theta}_L\right)\dot{\theta}_L + G(\theta_L) + K(\theta_L - \theta_M) = 0 \tag{6.4}$$

$$J_M\ddot{\theta}_M - K(\theta_L - \theta_M) = u \tag{6.5}$$

Donde $\theta_L, \theta_M \in \mathbb{R}^n$ son los vectores de posiciones angulares, vistas desde el punto de vista del eslabón y del motor respectivamente, de las articulaciones; $D: \mathbb{R} \to \mathbb{R}^{n\times n}$ es una matriz definida positiva que caracteriza los términos inerciales de los eslabones; $C: \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times n}$ es la matriz que contiene todos los términos de Coriolis y de fuerzas centrífugas que actúan en la planta; $G: \mathbb{R}^n \to \mathbb{R}^n$ es el vector de fuerzas gravitacionales; $K \in \mathbb{R}^{n\times n}$ es una matriz diagonal, definida positiva, que contiene los coeficientes de rigidez de los resortes de cada articulación; $J_M \in \mathbb{R}^{n\times n}$ es una matriz definida positiva que caracteriza los términos inerciales de los motores de cada articulación; y $u \in \mathbb{R}^n$ es el vector de pares de torsión de entrada.

Definiendo el vector extendido $\theta_f \coloneqq [\theta_L^T \quad \theta_M^T]^T$, las ecuaciones (6.4), (6.5) pueden ser reescritas en forma matricial:

$$D_f(\theta_L)\ddot{\theta}_f + C_f\left(\theta_L, \dot{\theta_L}\right)\dot{\theta}_f + G_f(\theta_L) + K_f\theta_f = u_f \tag{6.6}$$

Dónde:

$$\begin{aligned} D_f(\theta_L) &= \begin{bmatrix} D(\theta_L) & 0 \\ 0 & J_M \end{bmatrix}, \qquad C_f\left(\theta_L, \dot{\theta}_L\right) = \begin{bmatrix} C\left(\theta_L, \dot{\theta_L}\right) & 0 \\ 0 & 0 \end{bmatrix} \\ G_f(\theta_L) &= \begin{bmatrix} G(\theta_L) \\ 0 \end{bmatrix}, \qquad K_f = \begin{bmatrix} K & -K \\ -K & K \end{bmatrix}, \qquad u_f = \begin{bmatrix} 0 \\ \mathrm{I} \end{bmatrix} u \end{aligned} \tag{6.7}$$

Esta planta es un sistema dinámico sub–actuado, no lineal, oscilatorio, descrito por cuatro ecuaciones diferenciales ordinarias de segundo orden con múltiples entradas y salidas. Otras propiedades de esta planta se presentan en [42].

Para poder utilizar redes neuronales discretas para identificar esta planta, las señales continuas de entrada y de salida se discretizan con un tiempo de muestreo (τ).

6.4 Simulación y Resultados Para Planta SISO

Las simulaciones fueron realizadas con Simulink de MATLAB en dos etapas [38], [40]:

- Una etapa de identificación, en la cual los pesos sinápticos de la **CVRNN** se ajustan hasta que su salida coincide con la señal de salida oscilatoria de la planta.
- Una etapa de generalización, en la cual los pesos sinápticos de la **CVRNN** entrenada se fijan y se aplica una entrada diferente a la red y a la planta para obtener sus salidas respectivas, las cuales son comparadas para validar el aprendizaje de la red neuronal.

Se realizaron ambas simulaciones para el algoritmo de **CVLM** desarrollado a lo largo de este trabajo, y para el algoritmo de **CVBP** desarrollado en [18].

De ambos se obtiene el Error Medio Cuadrático total (**MSE**), para las etapas de aprendizaje y generalización, con el cual se compara el desempeño de la red neuronal al ser entrenada con ambos algoritmos.

El cálculo del **MSE** se realiza por medio de la siguiente ecuación [18]:

$$\mathbf{MSE} = \frac{1}{N}\sum_{i=1}^{N}[E^*(i)\cdot E(i)] \tag{6.8}$$

Donde N es el número de iteraciones realizadas por la simulación.

Las señales de entrada utilizadas en la etapa de entrenamiento u_L y en la etapa de generalización u_G están dadas por:

$$u_L(t) = \sin\left(\frac{1}{10}t\right) + 0.5\sin\left(\frac{1}{25}t\right) \tag{6.9}$$

$$u_G = 0.5\sin\left(\frac{1}{10}t\right) + 0.8\sin\left(\frac{1}{20}t\right) \tag{6.10}$$

La **CVRNN** utilizada tiene dimensiones $n = 3, m = 1, L = 1$; con condiciones iniciales de los estados internos $X(0) = [0 \quad 0 \quad 0]^T$, condiciones iniciales aleatorias en el intervalo $[-0.5,0.5]$ para los pesos sinápticos, tiempo de simulación de $T = 400$s, tiempo de muestreo $\tau = 0.01$ y la función de activación construida (5.19).

Los parámetros utilizados en la planta de un grado de libertad son: $J_L = J_m = 1$ [kg · m], $B_L = B_m = 0.1$ [kg · m · s^{-1}], $k = 0.1$ [kg · m · s^{-2}],$M = 0.5$ [kg], $L = 0.5$ [m] y $g = 9.81$ [m · s^{-2}].

Para el algoritmo de **CVBP** se utilizó el coeficiente de aprendizaje $\eta = 0.05$ y el coeficiente del término momento $\alpha = 0.005$. Para el algoritmo de **CVLM** se utilizaron los parámetros: $\alpha = 0.9775$, $\rho = 1 \times 10^{-4}$, $P_J(0) = 1 \times 10^6$, $P_B(0) = 1 \times 10^4$ y $P_C(0) = 1 \times 10^5$.

Las simulaciones se realizaron en una computadora portátil HP dv6 con sistema operativo Windows 7 Home Premium de 64 bits, procesador Intel Core i7-3610QM a 2.3 GHz de velocidad, cuatro núcleos, memoria RAM de 8Gb y con el software MATLAB 2013b.

Resultados Para el Algoritmo de Backpropagation Complejo. la Figura 6.6 muestra una comparación entre la salida de la planta y la salida de la red neuronal en la etapa de entrenamiento, mientras que la Figura 6.7 muestra una comparación entre la salida de la planta y la salida de la red neuronal ya entrenada, en la etapa de generalización.

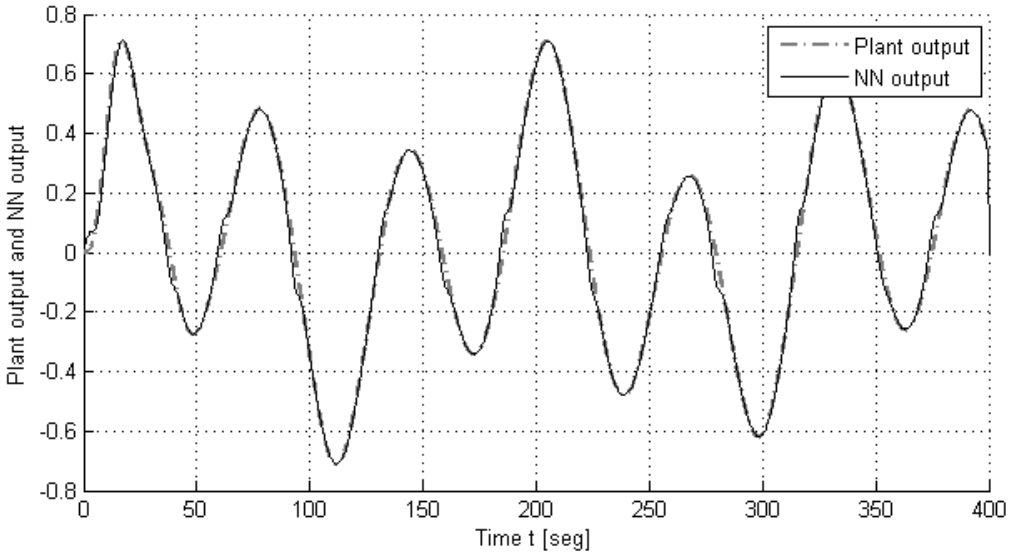


Figura 6.6 Señales de salida de la planta y salida de la red neuronal entrenada con el algoritmo CVBP, para la etapa de aprendizaje

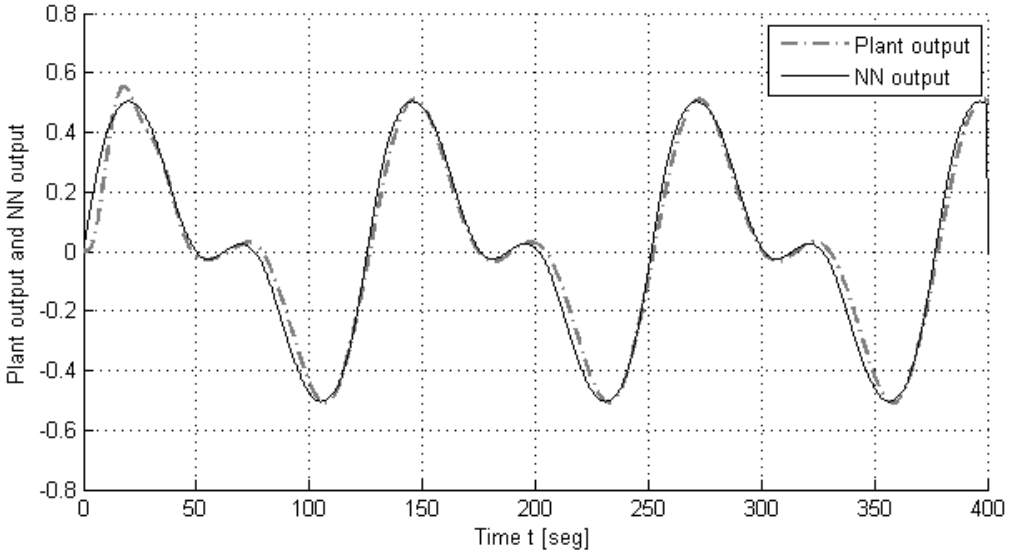


Figura 6.7 Señales de salida de la planta y salida de la red neuronal entrenada con el algoritmo CVBP, para la etapa de generalización

Resultados Para el Algoritmo de Levenberg–Marquardt Complejo. La Figura 6.8 muestra una comparación entre la salida de la planta y la salida de la red neuronal en la etapa de entrenamiento, mientras que la Figura 6.9 muestra una comparación entre la salida de la planta y la salida de la red neuronal ya entrenada, en la etapa de generalización.

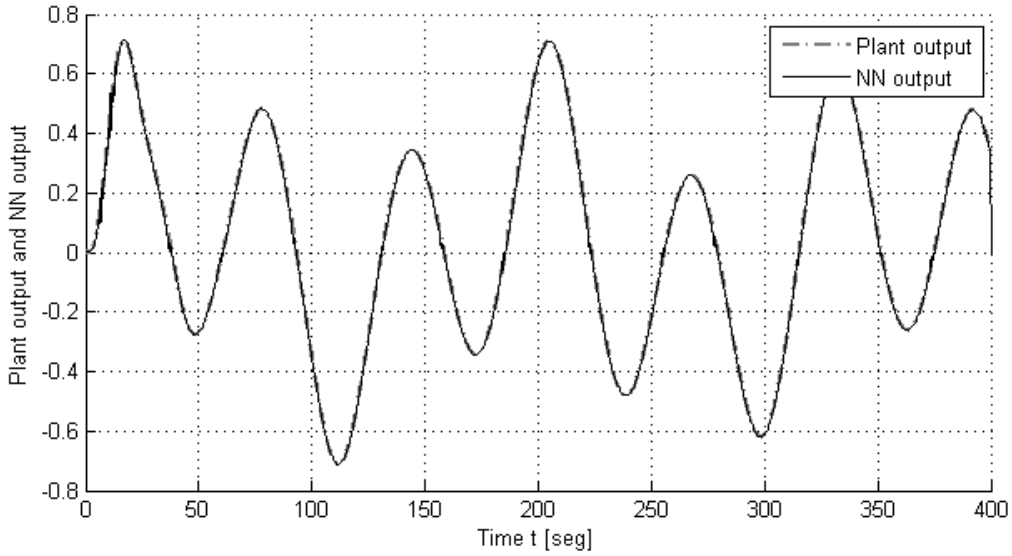


Figura 6.8 Señales de salida de la planta y salida de la red neuronal entrenada con el algoritmo CVLM, para la etapa de aprendizaje

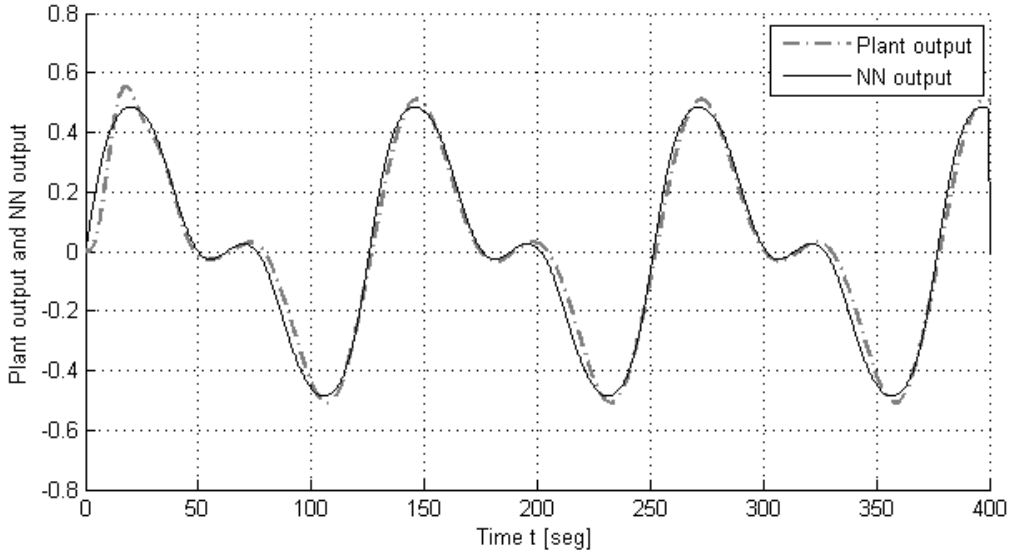


Figura 6.9 Señales de salida de la planta y salida de la red neuronal entrenada con el algoritmo CVLM, para la etapa de generalización

Para ambos algoritmos se observa una convergencia rápida de la salida de la **CVRNN** y la salida de la planta, lo que sugiere que la identificación del sistema se hace de forma casi inmediata. En la etapa de generalización, se observa un buen desempeño de la neurona para seguir la salida de la planta, lo que sugiere que el aprendizaje de la planta para ambos algoritmos tiene un buen desempeño.

La Tabla 6.1 muestra el **MSE** final para 400 segundos de simulación (40,000 iteraciones), con ambos algoritmos de entrenamiento y para ambas etapas. Se observa que el algoritmo de entrenamiento **CVLM** presenta, en general, un mejor desempeño que el **CVBP** en ambas etapas.

Tabla 6.1 MSE final para planta SISO, con 400 segundos de simulación en ambos algoritmos, para la etapa de entrenamiento y generalización

	CVBP	**CVLM**
Entrenamiento	6.19×10^{-4}	0.59×10^{-4}
Generalización	20.52×10^{-4}	17.72×10^{-4}

6.5 Simulación y Resultados Para Planta MIMO

Al igual que para la planta de una entrada y una salida, se realizaron las simulaciones para las etapas de identificación y generalización. Ambas simulaciones se desarrollaron con los algoritmos de **CVLM** y **CVBP**, y el desempeño de la red neuronal entrenada con estos es medido a partir del **MSE** total, calculado con la ecuación (6.8).

Las señales de entrada utilizadas en la etapa de entrenamiento u_L y en la etapa de generalización u_G están dadas por los vectores:

$$u_L(t) = \begin{bmatrix} 0.5\sin\left(\frac{1}{25}t\right) + 0.3\sin\left(\frac{1}{10}t\right) \\ 0.2\sin\left(\frac{1}{15}t\right) + 0.6\sin\left(\frac{1}{20}t\right) \end{bmatrix} \tag{6.11}$$

$$u_G(t) = \begin{bmatrix} 0.6\sin\left(\frac{1}{15}t\right) + 0.4\cos\left(\frac{1}{20}t\right) \\ 0.3\sin\left(\frac{1}{25}t\right) + 0.7\cos\left(\frac{1}{10}t\right) \end{bmatrix} \tag{6.12}$$

La **CVRNN** utilizada tiene dimensiones $n = 3, m = 2, L = 2$; con condiciones iniciales de los estados internos $X(0) = [0 \quad 0 \quad 0]^T$, condiciones iniciales aleatorias en el intervalo $[-0.5,0.5]$ para los pesos sinápticos, tiempo de simulación de $T = 400$s, tiempo de muestreo $\tau = 0.01$ y la función de activación construida (5.19).

Los parámetros utilizados en la planta de dos grados de libertad son: $J_{L1} = J_{L2} = J_{m1} = J_{m2} = 0.5$ [kg · m] , $B_{L1} = B_{L2} = 0.2$ [kg · m · s^{-1}] , $B_{m1} = B_{m2} = 0$, $k_1 = k_2 = 0.1$ [kg · m · s^{-2}], $M_1 = M_2 = 1$ [kg], $L_1 = L_2 = 1$ [m] y $g = 9.81$ [m · s^{-2}].

Para el algoritmo de **CVBP** se utilizó el coeficiente de aprendizaje $\eta = 0.05$ y el coeficiente del término momento $\alpha = 0.005$. Para el algoritmo de **CVLM** se utilizaron los parámetros: $\alpha = 0.9775$, $\rho = 1 \times 10^{-4}$, $P_J(0) = 1 \times 10^6$, $P_B(0) = 1 \times 10^4$ y $P_C(0) = 1 \times 10^5$.

Los datos de la computadora con la cual se realizaron las simulaciones se encuentran reportados en la sección anterior.

Resultados Para el Algoritmo de Backpropagation Complejo. Las Figuras 6.10 y 6.11 muestran una comparación entre la salida de la planta y la salida de la red neuronal, para el primer y segundo grado de libertad de la planta respectivamente, en la etapa de entrenamiento.

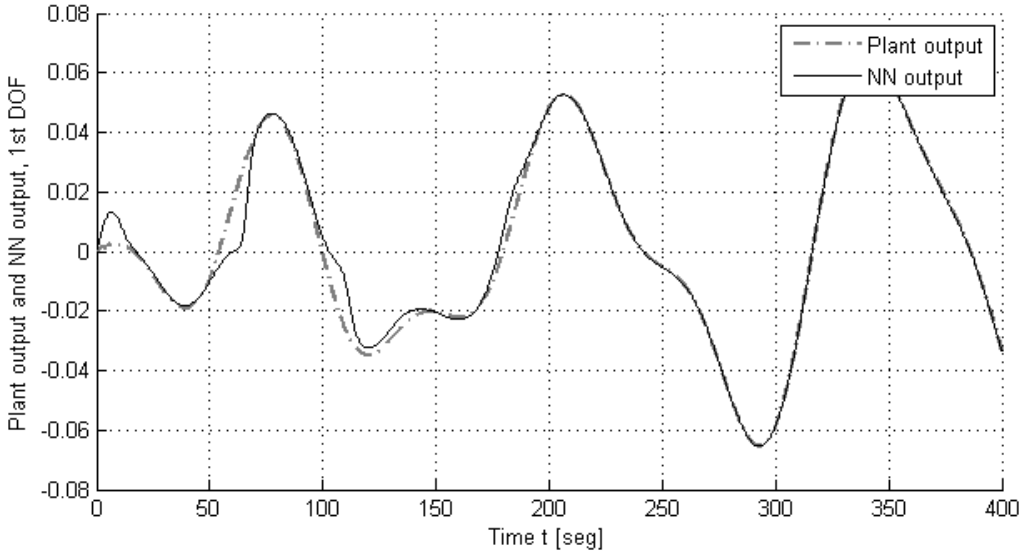


Figura 6.10 Señales del primer grado de libertad de la salida de la planta y salida de la red neuronal entrenada con el algoritmo CVBP, para la etapa de aprendizaje

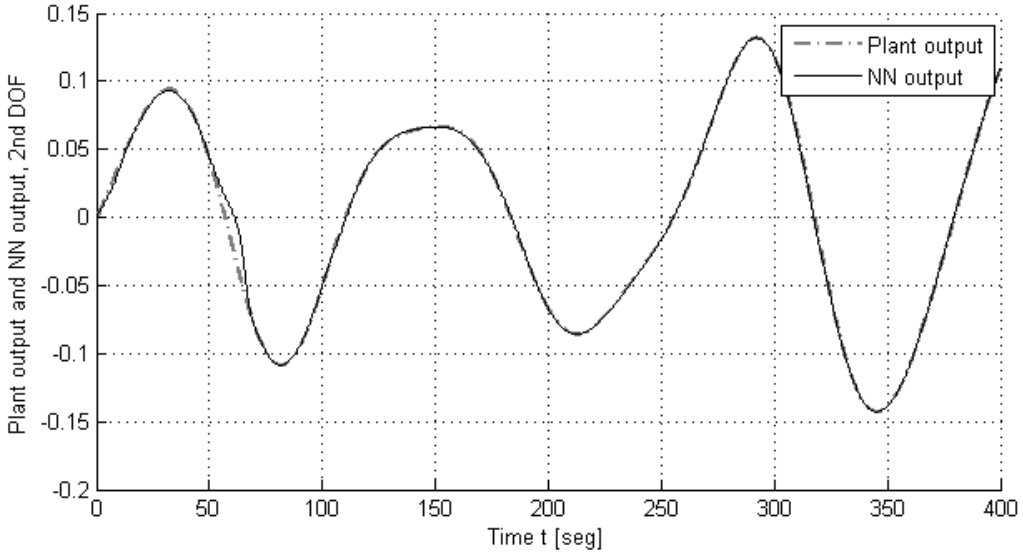


Figura 6.11 Señales del segundo grado de libertad de la salida de la planta y salida de la red neuronal entrenada con el algoritmo CVBP, para la etapa de aprendizaje

Las Figuras 6.12 y 6.13 muestran una comparación entre la salida de la planta y la salida de la red neuronal ya entrenada, para el primer y segundo grado de libertad de la planta, en la etapa de generalización.

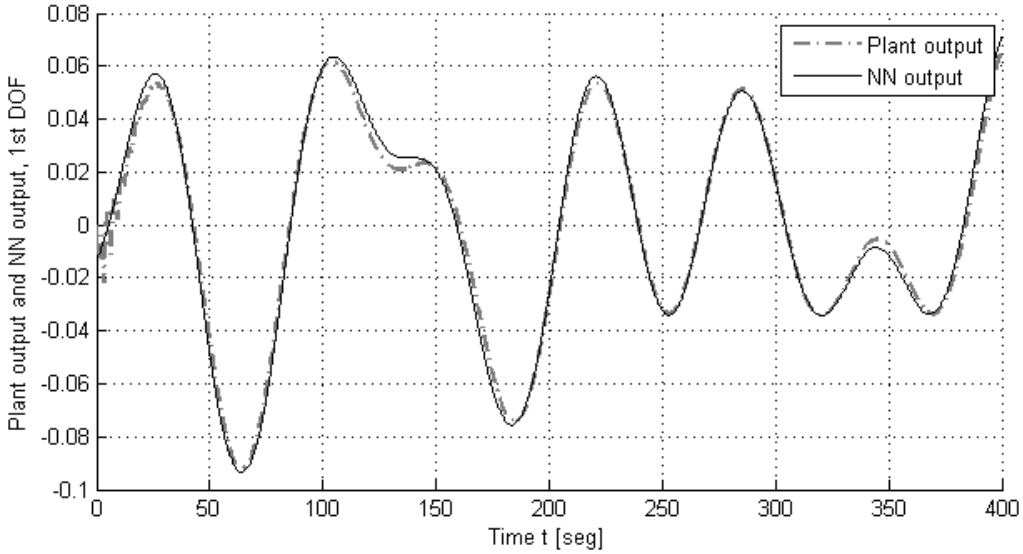


Figura 6.12 Señales del primer grado de libertad de la salida de la planta y salida de la red neuronal entrenada con el algoritmo CVBP, para la etapa de generalización

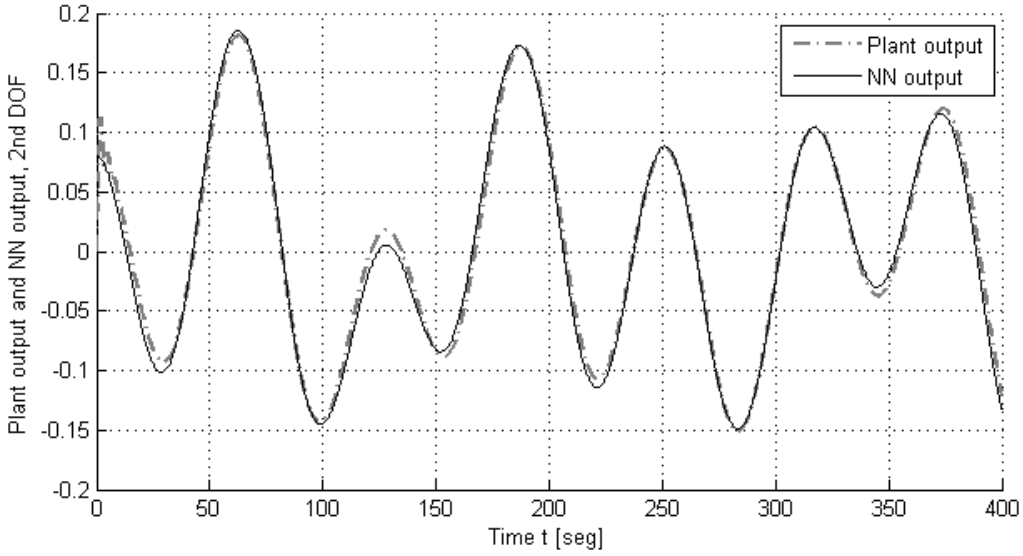


Figura 6.13 Señales del segundo grado de libertad de la salida de la planta y salida de la red neuronal entrenada con el algoritmo CVBP, para la etapa de generalización

Resultados Para el Algoritmo de Levenberg–Marquardt Complejo. Las Figuras 6.14 y 6.15 muestran una comparación entre la salida de la planta y la salida de la red neuronal, para el primer y segundo grado de libertad de la planta respectivamente, en la etapa de entrenamiento.

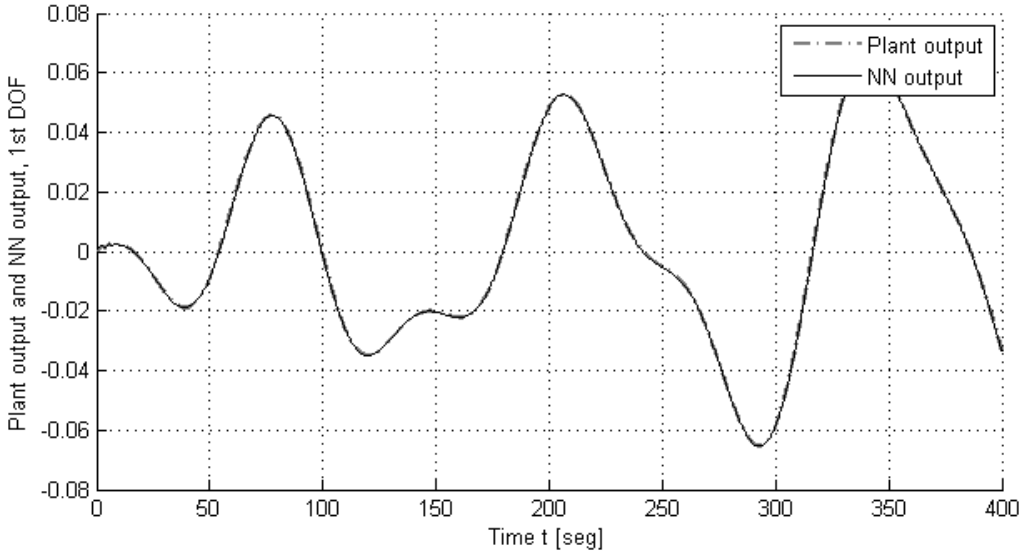


Figura 6.14 Señales del primer grado de libertad de la salida de la planta y salida de la red neuronal entrenada con el algoritmo CVLM, para la etapa de aprendizaje

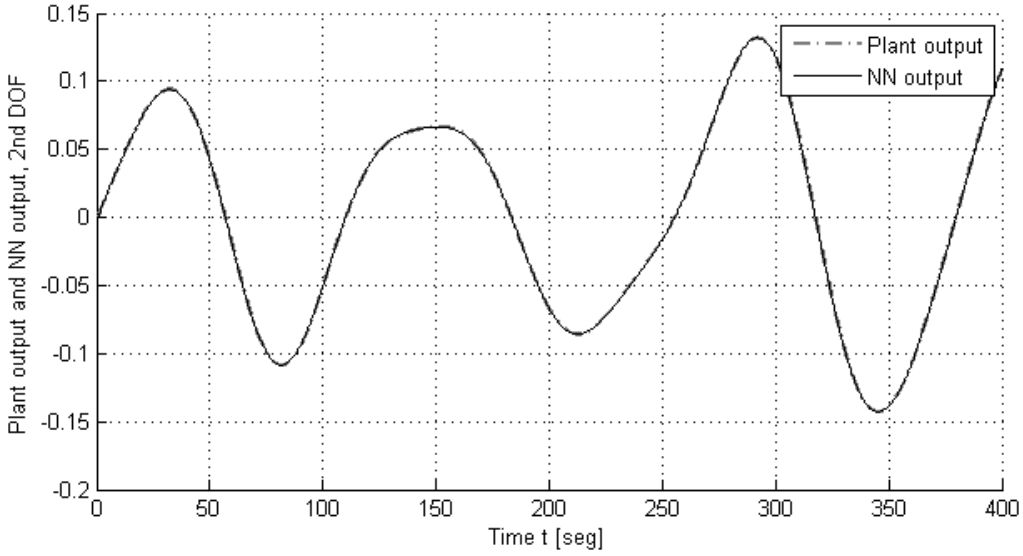


Figura 6.15 Señales del segundo grado de libertad de la salida de la planta y salida de la red neuronal entrenada con el algoritmo CVLM, para la etapa de aprendizaje

Las Figuras 6.16 y 6.17 muestran una comparación entre la salida de la planta y la salida de la red neuronal ya entrenada, para el primer y segundo grado de libertad de la planta, en la etapa de generalización.

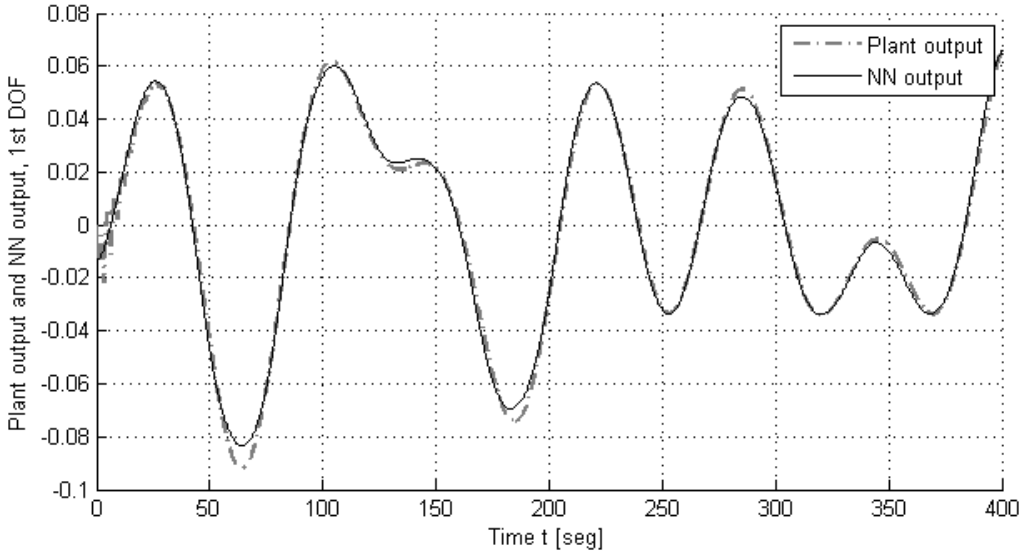


Figura 6.16 Señales del primer grado de libertad de la salida de la planta y salida de la red neuronal entrenada con el algoritmo CVLM, para la etapa de generalización

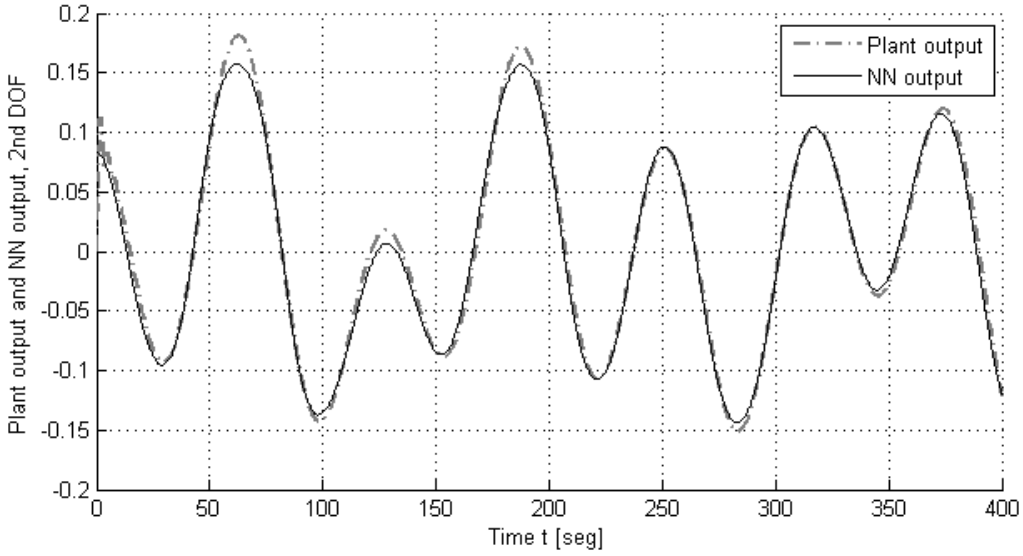


Figura 6.17 Señales del segundo grado de libertad de la salida de la planta y salida de la red neuronal entrenada con el algoritmo CVLM, para la etapa de generalización

Al igual que en el caso **SISO**, en ambos algoritmos se observa una convergencia rápida de la salida de la **CVRNN** y la salida de la planta, lo que sugiere que la identificación del sistema se hace de forma casi inmediata. En la etapa de generalización, se observa un buen desempeño de la neurona para seguir la salida de la planta, lo que sugiere que el aprendizaje de la planta para ambos algoritmos tiene un buen desempeño.

Cabe destacar la facilidad y flexibilidad con la cual la misma red neuronal puede ser adecuada para el problema de identificación de plantas con diferente número de entradas y salidas.

La Tabla 6.2 muestra el **MSE** final para 400 segundos de simulación (40,000 iteraciones), con ambos algoritmos de entrenamiento y para ambas etapas. Se observa que el algoritmo de entrenamiento **CVLM** presenta, al igual que en el caso **SISO**, un mejor desempeño que el **CVBP** en ambas etapas.

Tabla 6.2 MSE final para planta MIMO, con 400 segundos de simulación en ambos algoritmos, para la etapa de entrenamiento y generalización

	CVBP	**CVLM**
Entrenamiento	35.85×10^{-6}	0.03×10^{-6}
Generalización	80.69×10^{-6}	70.01×10^{-6}

7. Control Neuronal Adaptable

En este capítulo se aborda el tema del uso de redes neuronales en el control adaptable, en el contexto de los sistemas dinámicos. Se proponen dos esquemas de control neuronal adaptable directos. Los esquemas de control propuestos utilizan tres **CVRNN** con la topología con función de activación construida, como la presentada en el capítulo anterior.

Posteriormente, se presentan los datos de la simulación y los datos obtenidos de este, usando los algoritmos de **CVBP** y **CVLM** para entrenar a las redes neuronales. Por último, se hace una comparación del desempeño de los controladores para cada uno de los algoritmos.

7.1 Control Adaptable con Redes Neuronales

La adaptación es una característica fundamental de los seres vivos mediante la cual mantienen el equilibrio fisiológico en medio de condiciones ambientales variables. Para proyectar la capacidad de adaptación natural a un sistema de control, se consideran los aspectos adaptables del comportamiento humano o animal para desarrollar un sistema que se comporte análogamente.

Un sistema de control adaptable es un sistema que continua y automáticamente mide las características dinámicas de la planta, las compara con las características dinámicas deseadas y usa la diferencia para variar los parámetros ajustables del sistema de control o generar una señal de control que mantenga el funcionamiento de la planta independientemente de las variaciones ambientales [41].

Todo sistema de control que utiliza redes neuronales es por definición un sistema de control adaptable, dado que esta puede ajustar sus pesos sinápticos en la presencia de perturbaciones externas a la planta. El uso de redes neuronales dentro de los sistemas de control ha aumentado en los últimos años, dado que estas presentan rápida

adaptación, capacidad de aproximación y capacidad de integración masiva y en paralelo [2].

7.2 Esquemas de Control Neuronal Adaptable Directo

En este trabajo se proponen dos esquemas de control neuronal adaptable directos: el primero utilizando un controlador *feedforward* y una realimentación de estado por medio de la identificación del estado; y el segundo incluye un término integral del error para compensar un posible error en estado estacionario [37], [39].

Control Neuronal Adaptable Directo con Controlador Feedforward y Realimentación del Estado. Para el primer sistema de control adaptable utilizando **CVRNN**, se utiliza el esquema de control presentado en la Figura 7.1.

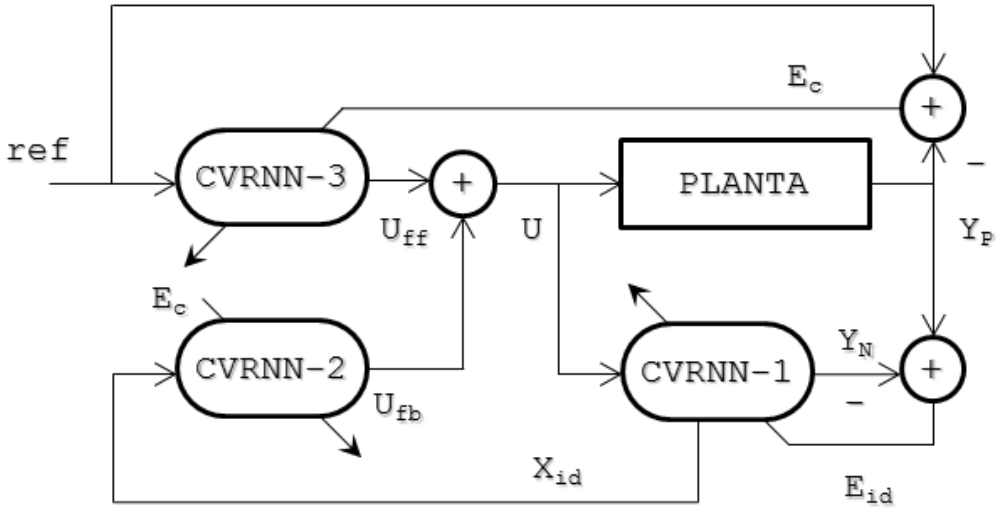


Figura 7.1 Esquema de control neuronal adaptable directo con controlador *feedforward* y realimentación de estado

En este esquema de control se utilizan tres **CVRNN**, cada una con una tarea específica:

- **CVRNN–1.** Se utiliza como un identificador del modelo de la planta. La señal de control U es alimentada a esta red neuronal para producir una salida de identificación Y_N, la cual es comparada con la salida de la planta Y_P para producir el error de identificación E_{id}, el cual es usado para ajustar sus pesos sinápticos. El error de identificación está dado por la ecuación:

$$E_{id}(k) = Y_P(k) - Y_N(k) \tag{7.1}$$

- **CVRNN–2.** Esta red neuronal es alimentada con los estados internos X_{id} de la red neuronal de identificación **CVRNN–1**, para producir una señal de control por realimentación de estado U_{fb}. Esta red es entrenada usando la señal de error de control E_c, la cual se define comparando la señal de referencia R con la señal de salida de la planta, o bien:

$$E_c(k) = R(k) - Y_P(k) \tag{7.2}$$

- **CVRNN–3.** Esta red es alimentada con la señal de referencia R del sistema para producir una señal de control de tipo *feedforward* U_{ff}. Es entrenada utilizando la señal de error de control (7.2) y converge al modelo inverso de la planta en lazo cerrado.

Excepto por las dimensiones de cada una de estas redes neuronales complejas, las tres presentan la misma topología. Luego, la señal de control U que es alimentada a la planta original es la suma de las señales de control por realimentación de estado y *feedforward*, descrita por la siguiente ecuación:

$$U(k) = U_{fb}(k) + U_{ff}(k) \tag{7.3}$$

Función de Transferencia del Control Neuronal Adaptable Directo. Se supone que la planta a la cual se le aplica el esquema de control presenta un modelo discreto y no lineal como el dado por las ecuaciones:

$$X_P(k+1) = F\big(X_P(k), U(k)\big) \tag{7.4}$$

$$Y_P(k) = G\big(X_P(k)\big) \tag{7.5}$$

Donde X_P es el estado interno de la planta, $F(\cdot)$ es una función no lineal del estado y la entrada y $G(\cdot)$ es una función no lineal del estado. Linealizando (7.4) y (7.5) alrededor de un punto de equilibrio, y linealizando las funciones de activación de las **CVRNN–1**, **2** y **3**, se obtienen las siguientes funciones de transferencia en el dominio z:

$$W_P(z) = C_P(zI - J_P)^{-1}B_P \tag{7.6}$$

$$W_{id}(z) = (zI - J_{id})^{-1}B_{id} \tag{7.7}$$

$$Q_1(z) = C_{fb}\big(zI - J_{fb}\big)^{-1}B_{fb} \tag{7.8}$$

$$Q_2(z) = C_{ff}\big(zI - J_{ff}\big)^{-1}B_{ff} \tag{7.9}$$

Donde W_P es la relación entre la salida de la planta Y_P y la señal de control U; W_{id} es la relación entre el estado estimado X_{id} y la señal de control U; Q_1 es la relación entre la señal de control por realimentación de estado U_{fb} y el estado estimado X_{id}; y Q_2 es la relación entre la señal de control feedforward U_{ff} y la señal de referencia del sistema.

Siguiendo el diagrama de bloques del esquema de control de la Figura 7.1, se obtiene la ecuación del sistema en lazo cerrado:

$$Y_P(z) = W_P(z) \cdot [I - Q_1(z)W_{id}(z)]^{-1} \cdot Q_2(z)R(z) \tag{7.10}$$

Luego, la señal de salida del sistema en lazo cerrado queda expresada en términos de las funciones de transferencia de la planta, la red neuronal de identificación y las redes neuronales de control [36].

Control Neuronal Adaptable Directo con Controlador Feedforward, Realimentación del Estado y Término Integral. Para el segundo sistema de control adaptable utilizando **CVRNN**, se utiliza el esquema de control presentado en la Figura 7.2.

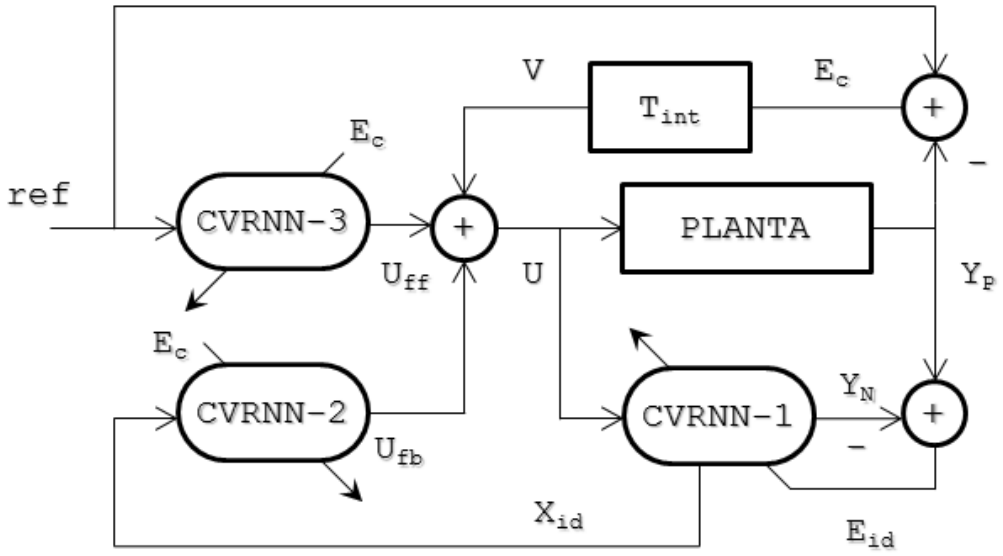


Figura 7.2 Esquema de control neuronal adaptable directo con controlador *feedforward*, realimentación de estado y término integral

En este esquema de control se utilizan tres **CVRNN** con las mismas funciones que el esquema anterior. A la señal de control U se le agrega un término integral V del error de control, con el cual se busca eliminar el error en estado estacionario que pueda

presentar la salida de la planta. Este término integral se calcula por medio de la ecuación:

$$V(k+1) = V(k) + \tau \cdot K_i \cdot E_c(k) \tag{7.11}$$

Donde K_i es la ganancia de acción integral y τ es el tiempo de muestreo. Luego, la señal de control total para este esquema queda de la siguiente forma:

$$U(k) = U_{fb}(k) + U_{ff}(k) + V(k) \tag{7.12}$$

La ecuación del sistema en lazo cerrado para este esquema de control es una modificación de la ecuación (7.10), y está dada por:

$$Y_P(z) = W_P(z) \cdot [I - Q_1(z)W_{id}(z)]^{-1} \cdot [Q_2(z)R(z) + Q_V(z)] \tag{7.13}$$

Donde Q_V es la relación entre la señal de control de acción integral V y la señal del error de control E_C.

7.3 Simulación y Resultados Para Planta SISO

Las simulaciones fueron realizadas con Simulink de MATLAB para cada uno de los esquemas de control presentados.

Se realizaron ambas simulaciones para el algoritmo de **CVLM** desarrollado a lo largo de este trabajo, y para el algoritmo de **CVBP** desarrollado en [18].

De ambos se obtiene el **MSE** con el cual se compara el desempeño de los esquemas de control al ser entrenados con ambos algoritmos. El cálculo del **MSE** se realiza por medio de la ecuación (6.4).

La señal de referencia usada para el sistema **SISO** está dada por la ecuación:

$$R(k) = 0.5 \sin\left(\frac{1}{25}k\right) + 0.3 \sin\left(\frac{1}{15}k\right) \tag{7.14}$$

En ambos esquemas de control, las dimensiones usadas para las redes neuronales **CVRNN–1** y **CVRNN–3** son $n_{1,3} = 3, m_{1,3} = 1, L_{1,3} = 1$; mientras que para la red neuronal **CVRNN–2** son $n_2 = 3, m_2 = 3, L_2 = 1$.

Se usaron condiciones iniciales de los estados internos de $X(0) = [0.1 \quad 0.1 \quad 0.1]^T$, condiciones iniciales aleatorias en el intervalo $[-0.1, 0.1]$ para los pesos sinápticos, tiempo de simulación de $T = 100$s, tiempo de muestreo $\tau = 0.01$ y la función de activación construida (5.19).

Los parámetros utilizados en la planta de un grado de libertad son: $J_L = J_m = 1$ [kg · m], $B_L = B_m = 0.1$ [kg · m · s^{-1}], $k = 0.1$ [kg · m · s^{-2}],$M = 0.5$ [kg], $L = 0.5$ [m] y $g = 9.81$ [m · s^{-2}].

Para el algoritmo de **CVBP** se utilizó el coeficiente de aprendizaje $\eta = 0.1$ y el coeficiente del término momento $\alpha = 0.04$. Para el algoritmo de **CVLM** se utilizaron los parámetros: $\alpha = 0.98$, $\rho = 1 \times 10^{-4}$, $P_J(0) = P_B(0) = P_C(0) = 1 \times 10^5$. Para el esquema de control con acción integral se usó una ganancia integral de $K_i = 0.005$.

Las simulaciones se realizaron en una computadora portátil HP dv6 con sistema operativo Windows 7 Home Premium de 64 bits, procesador Intel Core i7-3610QM a 2.3 GHz de velocidad, cuatro núcleos, memoria RAM de 8Gb y con el software MATLAB 2013b.

Resultados Para el Algoritmo de Backpropagation Complejo. La Figura 7.3 muestra la señal de referencia y la señal de salida de la planta cuando se le aplica el primer esquema de control, utilizando el algoritmo **CVBP** para el entrenamiento de las redes neuronales; mientras que la Figura 7.4 muestra la señal de referencia y la señal de salida de la planta cuando se le aplica el segundo esquema de control, que incluye la acción integral del error.

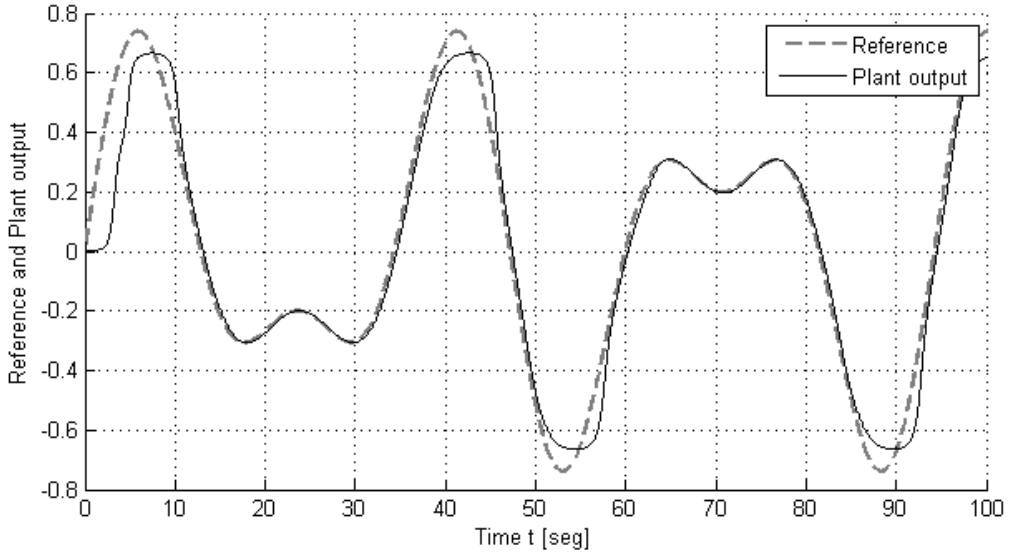


Figura 7.3 Señales de referencia y salida de la planta del primer esquema de control, con las redes neuronales entrenadas con el algoritmo CVBP

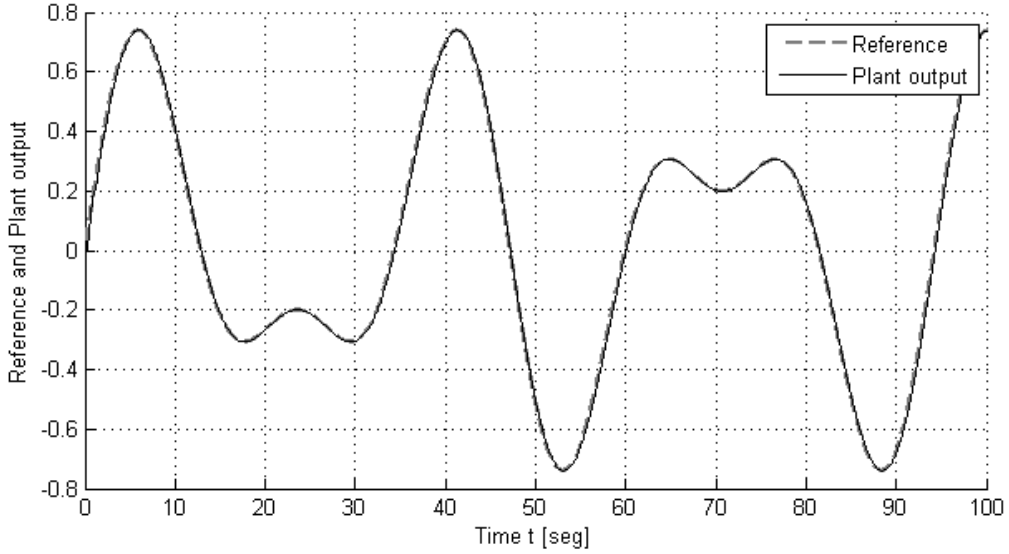


Figura 7.4 Señales de referencia y salida de la planta del segundo esquema de control, con las redes neuronales entrenadas con el algoritmo CVBP

Resultados Para el Algoritmo de Levenberg–Marquardt Complejo. La Figura 7.5 muestra la señal de referencia y la señal de salida de la planta cuando se le aplica el primer esquema de control, utilizando el algoritmo **CVLM** para el entrenamiento de las redes neuronales; mientras que la Figura 7.6 muestra la señal de referencia y la señal de salida de la planta cuando se le aplica el segundo esquema de control, que incluye la acción integral del error.

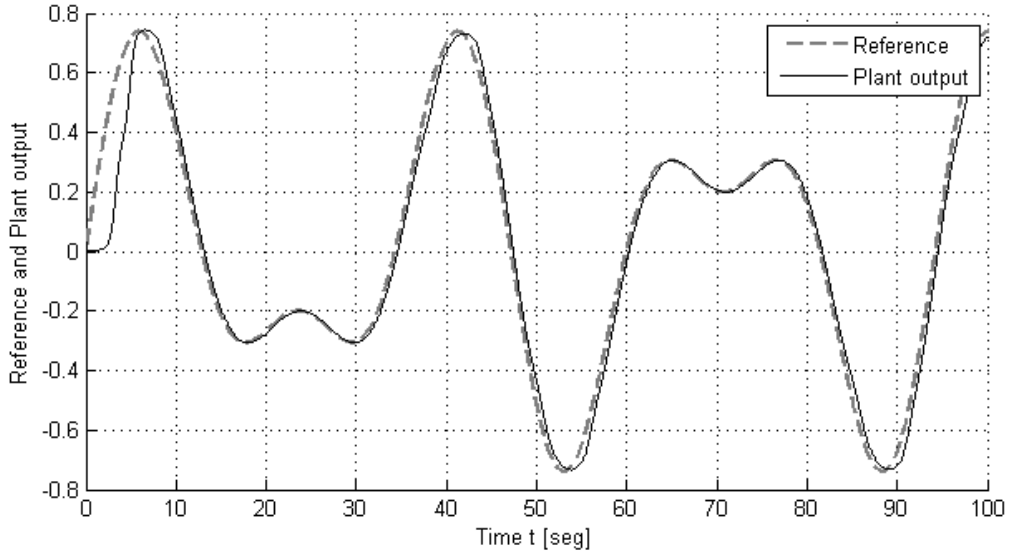


Figura 7.5 Señales de referencia y salida de la planta del primer esquema de control, con las redes neuronales entrenadas con el algoritmo CVLM

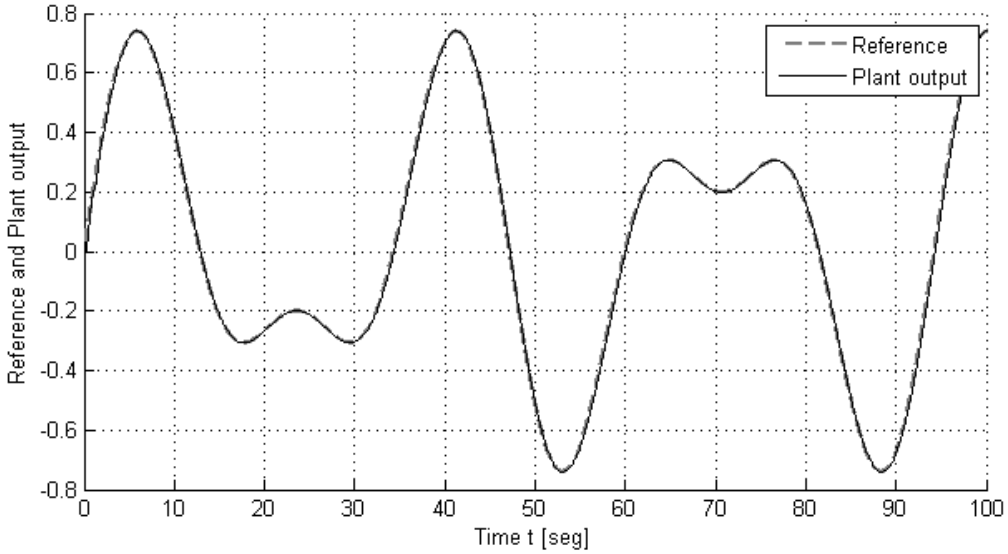


Figura 7.6 Señales de referencia y salida de la planta del segundo esquema de control, con las redes neuronales entrenadas con el algoritmo CVLM

Se observa que el comportamiento de la planta no lineal sigue muy de cerca a la señal de referencia, siendo más favorable el segundo esquema de control, ya que la acción integral del error elimina cualquier error de seguimiento presente. De igual forma se observa un mejor desempeño del esquema de control neuronal adaptable cuando las redes neuronales son entrenadas con el algoritmo de **CVLM**.

La Tabla 7.1 muestra el **MSE** final para 100 segundos de simulación (10,000 iteraciones) con ambos algoritmos de entrenamiento y para ambos esquemas de control. Se observa que el algoritmo de entrenamiento **CVLM** presenta, en general, un mejor desempeño que el **CVBP** en ambas etapas.

Tabla 7.1 MSE final para la planta SISO, con 100 segundos de simulación en ambos algoritmos, y ambos esquemas de control

	CVBP	**CVLM**
1er Esquema de Control	8.62×10^{-3}	5.78×10^{-3}
2do Esquema de Control	3.95×10^{-3}	2.30×10^{-3}

7.4 Simulación y Resultados Para Planta MIMO

Al igual que para la planta de una entrada y una salida, se realizaron las simulaciones para el primer y segundo esquema de control. Se realizaron ambas simulaciones para los algoritmos de **CVLM** y **CVBP**, y el desempeño de los esquemas de control neuronales es medido a partir del **MSE** total, calculado con la ecuación (6.8).

Las señales de referencia usadas están dadas por el vector:

$$R(k) = \begin{bmatrix} 0.25 \\ -0.45 \end{bmatrix} \text{rad} \tag{7.15}$$

En ambos esquemas de control, las dimensiones usadas para las redes neuronales **CVRNN–1** y **CVRNN–3** son $n_{1,3} = 3, m_{1,3} = 2, L_{1,3} = 2$; mientras que para la red neuronal **CVRNN–2** son $n_2 = 3, m_2 = 3, L_3 = 2$. Se usaron condiciones iniciales de los estados internos de $X(0) = [0.1 \quad 0.1 \quad 0.1]^T$, condiciones iniciales aleatorias en el intervalo $[-0.1, 0.1]$ para los pesos sinápticos, tiempo de simulación de $T = 100$s, tiempo de muestreo $\tau = 0.01$ y la función de activación construida (5.19).

Los parámetros utilizados en la planta de dos grados de libertad son: $J_{L1} = J_{L2} = J_{m1} = J_{m2} = 0.5$ [kg · m] , $B_{L1} = B_{L2} = 0.2$ [kg · m · s^{-1}] , $B_{m1} = B_{m2} = 0$, $k_1 = k_2 = 0.1$ [kg · m · s^{-2}], $M_1 = M_2 = 1$ [kg], $L_1 = L_2 = 1$ [m] y $g = 9.81$ [m · s^{-2}].

Para el algoritmo de **CVBP** se utilizó el coeficiente de aprendizaje $\eta = 0.1$ y el coeficiente del término momento $\alpha = 0.04$. Para el algoritmo de **CVLM** se utilizaron los parámetros: $\alpha = 0.98$, $\rho = 1 \times 10^{-4}$, $P_J(0) = P_B(0) = P_C(0) = 1 \times 10^5$. Para el esquema de control con acción integral se usó una ganancia integral de $K_i = 0.005$.

Resultados Para el Primer Esquema de Control. Las Figuras 7.7 y 7.8 muestran las señales de referencia y salidas de la planta, el primer y segundo grado de libertad respectivamente, para el primer esquema de control, cuando las redes neuronales son entrenadas con el algoritmo de **CVBP**.

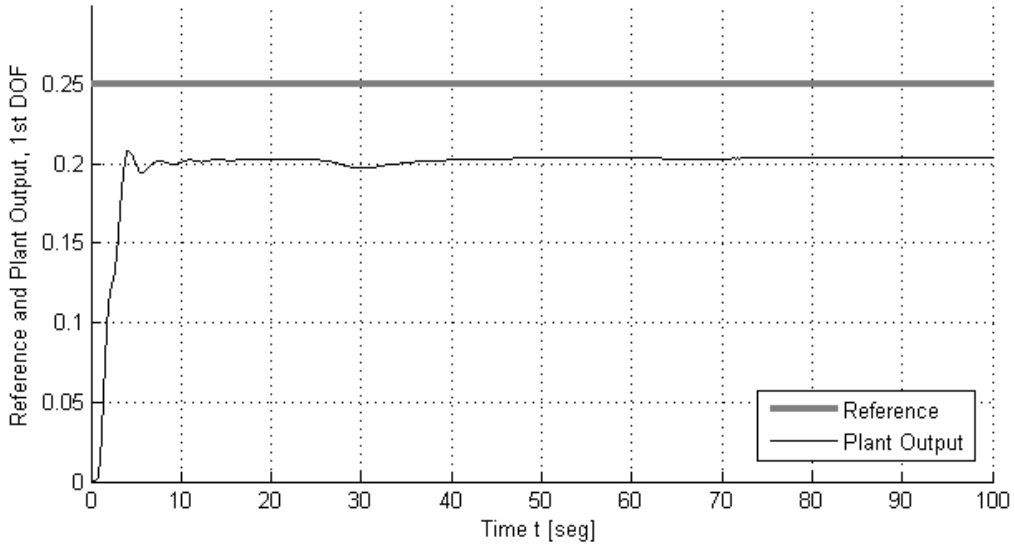


Figura 7.7 Señales del primer grado de libertad de la referencia y salida de la planta, para el primer esquema de control, con las redes neuronales entrenadas con el algoritmo CVBP

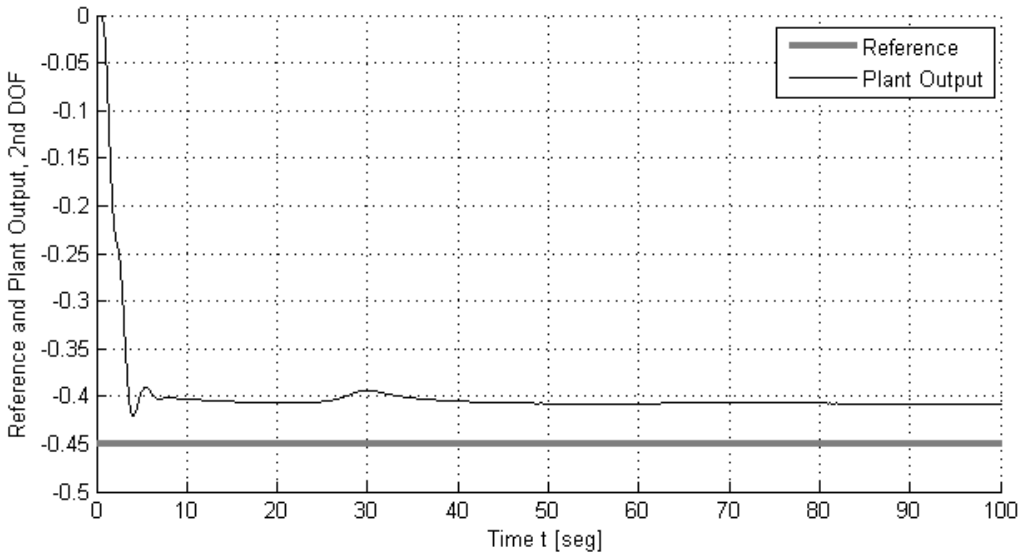


Figura 7.8 Señales del segundo grado de libertad de la referencia y salida de la planta, para el primer esquema de control, con las redes neuronales entrenadas con el algoritmo CVBP

Las Figuras 7.9 y 7.10 muestran las señales de referencia y salidas de la planta, el primer y segundo grado de libertad respectivamente, para el primer esquema de control, cuando las redes neuronales son entrenadas con el algoritmo de **CVLM**.

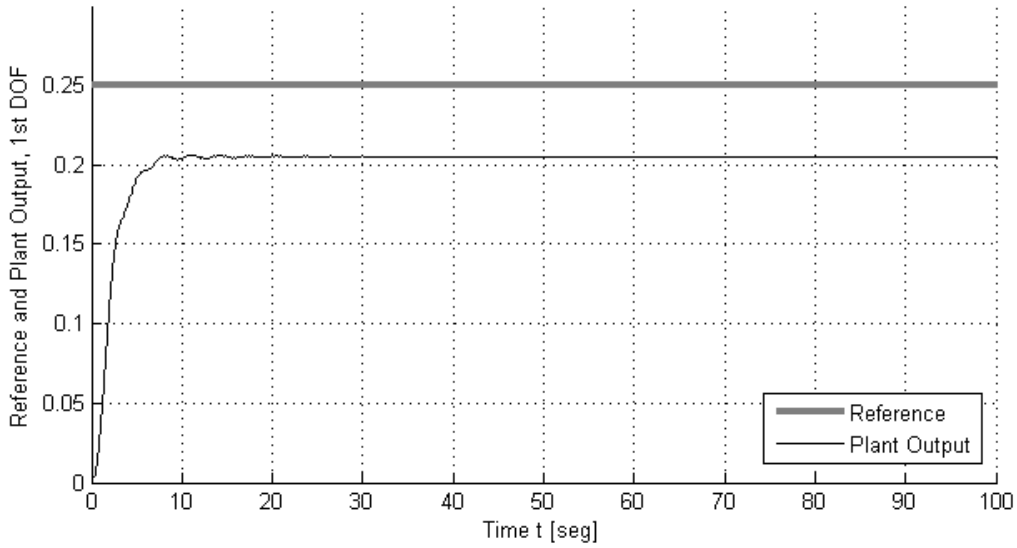


Figura 7.9 Señales del primer grado de libertad de la referencia y salida de la planta, para el primer esquema de control, con las redes neuronales entrenadas con el algoritmo CVLM

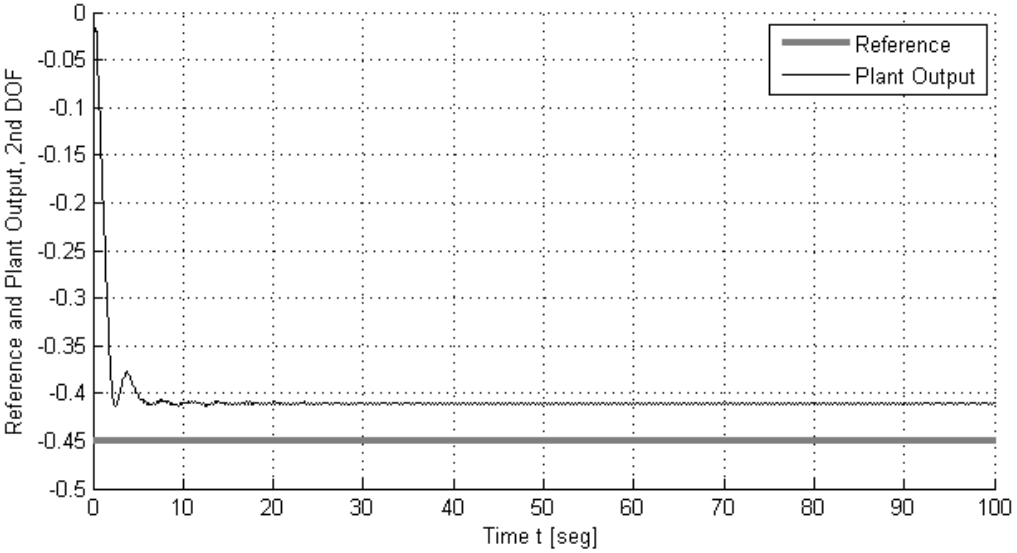


Figura 7.10 Señales del segundo grado de libertad de la referencia y salida de la planta, para el primer esquema de control, con las redes neuronales entrenadas con el algoritmo CVLM

Las Figuras 7.11 y 7.12 muestran una comparación entre las salidas de la planta, durante los primeros 20 segundos de simulación y para cada uno de los grados de libertad, cuando las redes neuronales son entrenadas con ambos algoritmos.

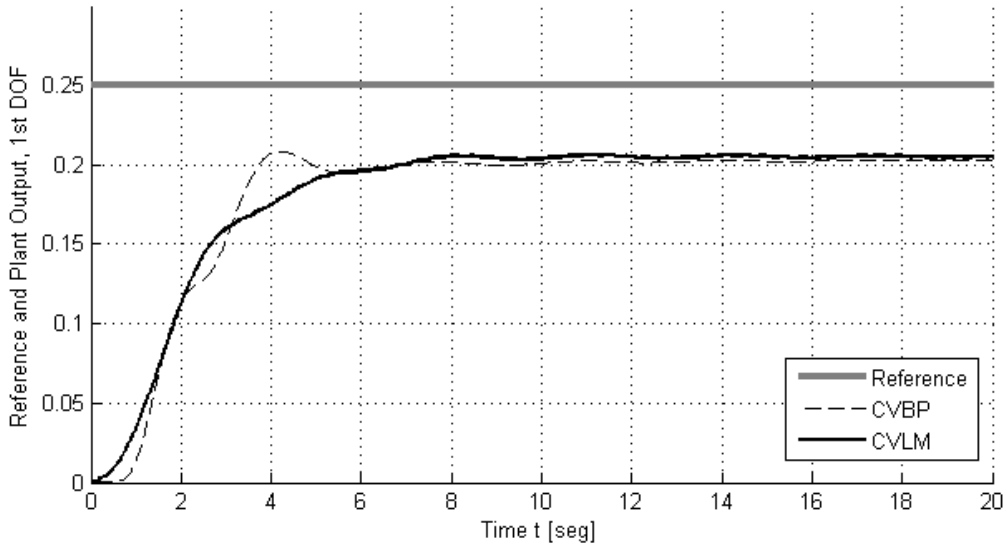


Figura 7.11 Comparación del primer grado de libertad de salida de la planta, para el primer esquema de control, con las redes neuronales entrenadas con ambos algoritmos

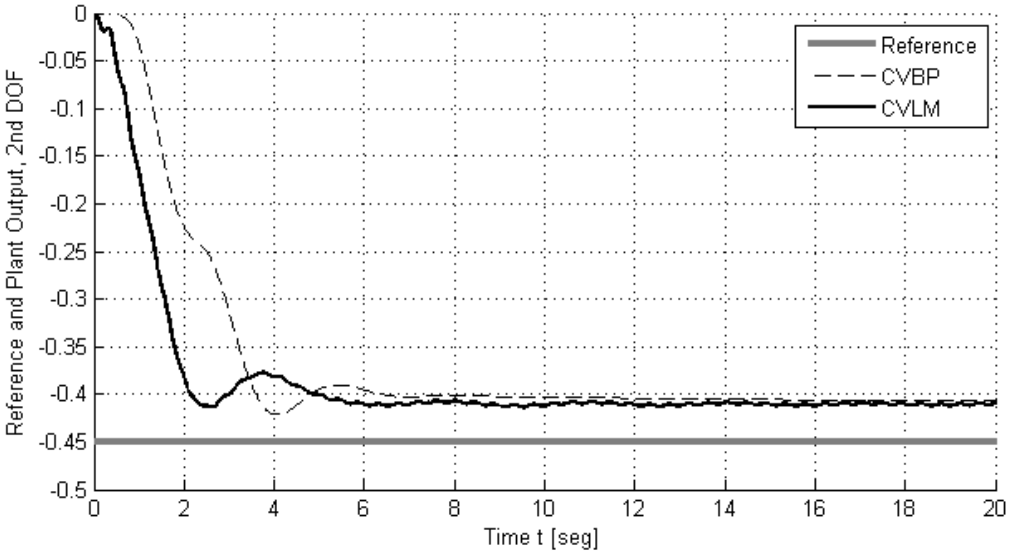


Figura 7.12 Comparación del segundo grado de libertad de salida de la planta, para el primer esquema de control, con las redes neuronales entrenadas con ambos algoritmos

Resultados Para el Segundo Esquema de Control. Las Figuras 7.13 y 7.14 muestran las señales de referencia y salidas de la planta, el primer y segundo grado de libertad respectivamente, para el segundo esquema de control con acción integral, cuando las redes neuronales son entrenadas con el algoritmo de **CVBP**.

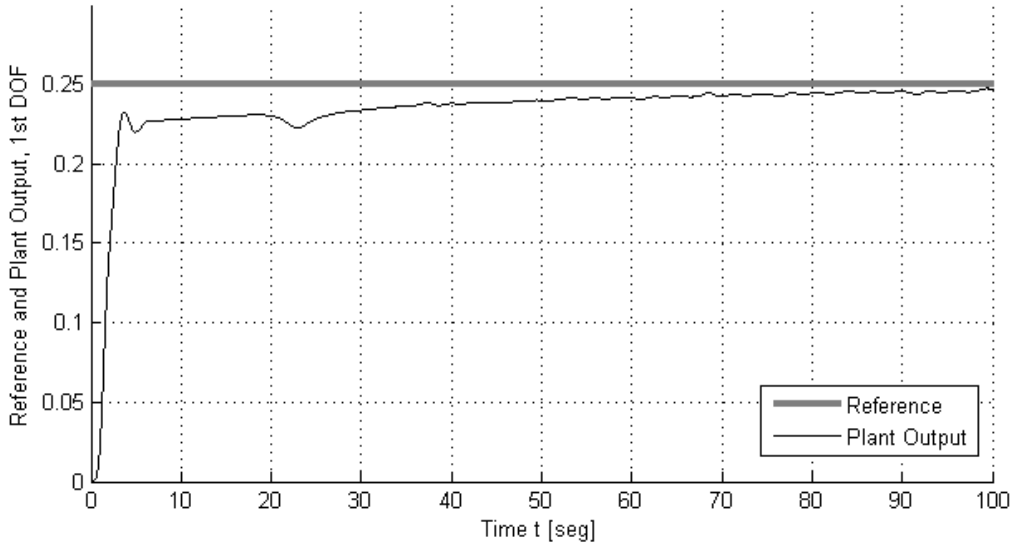


Figura 7.13 Señales del primer grado de libertad de la referencia y salida de la planta, para el segundo esquema de control, con las redes neuronales entrenadas con el algoritmo CVBP

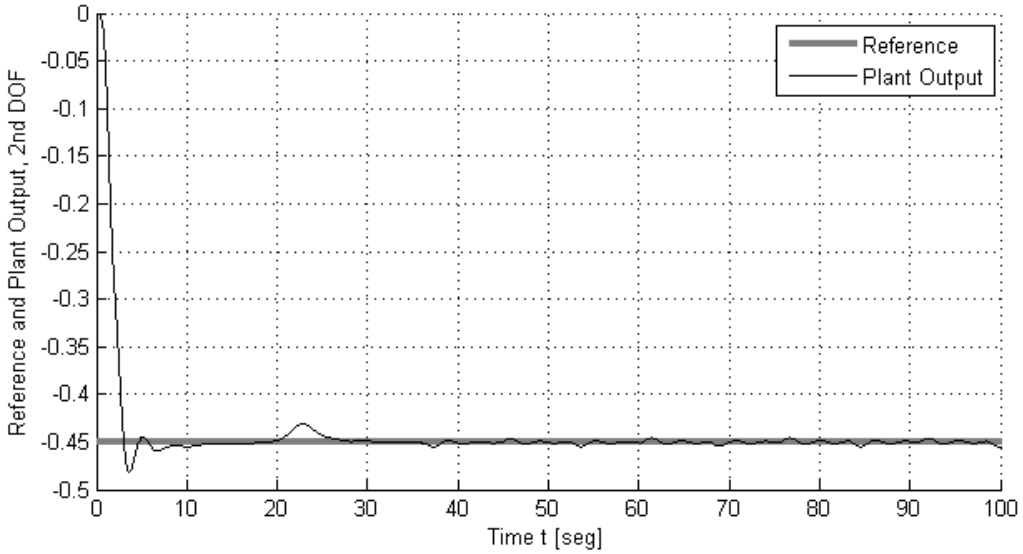


Figura 7.14 Señales del segundo grado de libertad de la referencia y salida de la planta, para el segundo esquema de control, con las redes neuronales entrenadas con el algoritmo CVBP

Las Figuras 7.15 y 7.16 muestran las señales de referencia y salidas de la planta, el primer y segundo grado de libertad respectivamente, para el segundo esquema de control con acción integral, cuando las redes neuronales son entrenadas con el algoritmo de **CVLM**.

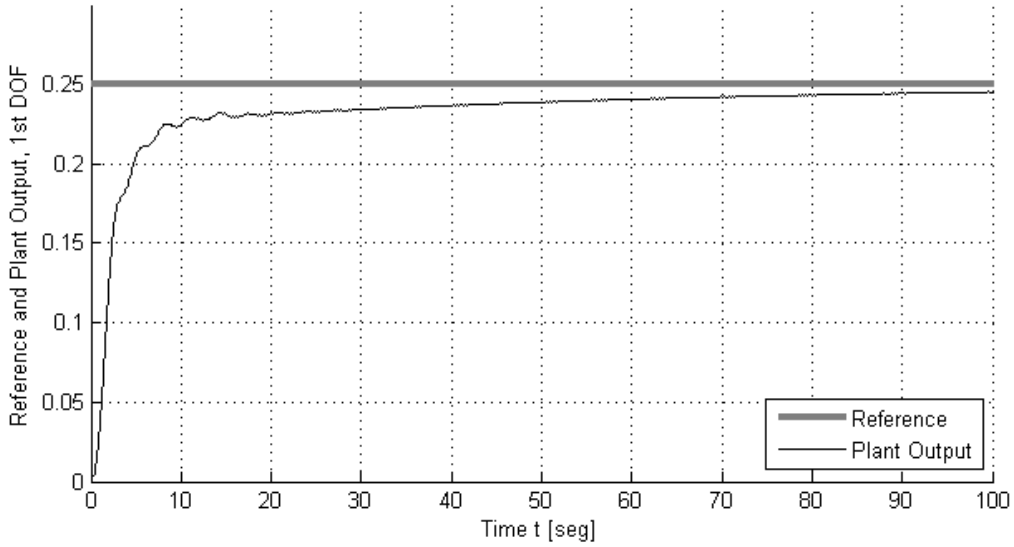


Figura 7.15 Señales del primer grado de libertad de la referencia y salida de la planta, para el segundo esquema de control, con las redes neuronales entrenadas con el algoritmo CVLM

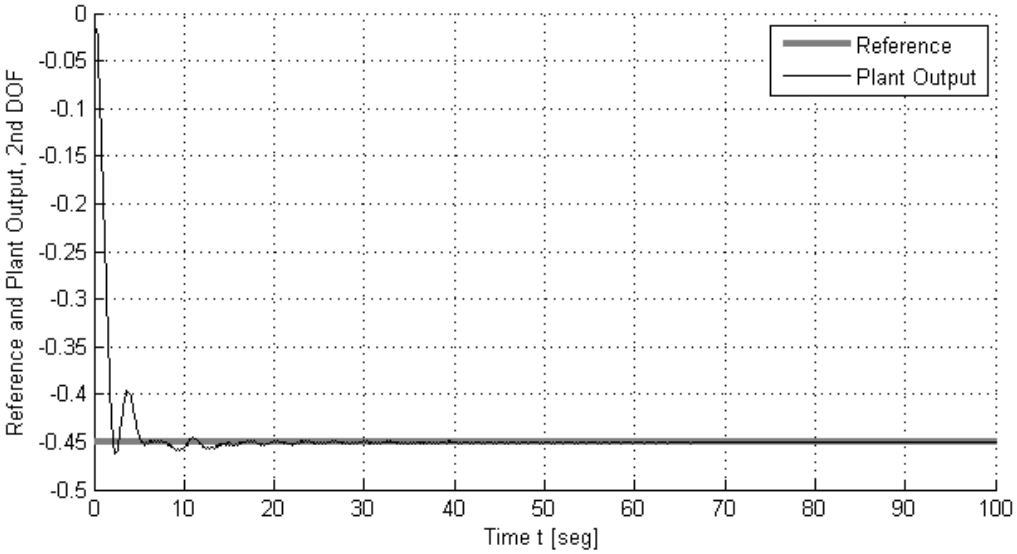


Figura 7.16 Señales del segundo grado de libertad de la referencia y salida de la planta, para el segundo esquema de control, con las redes neuronales entrenadas con el algoritmo CVLM

Las Figuras 7.17 y 7.18 muestran una comparación entre las salidas de la planta, durante los primeros 20 segundos de simulación y para cada uno de los grados de libertad, cuando las redes neuronales son entrenadas con ambos algoritmos.

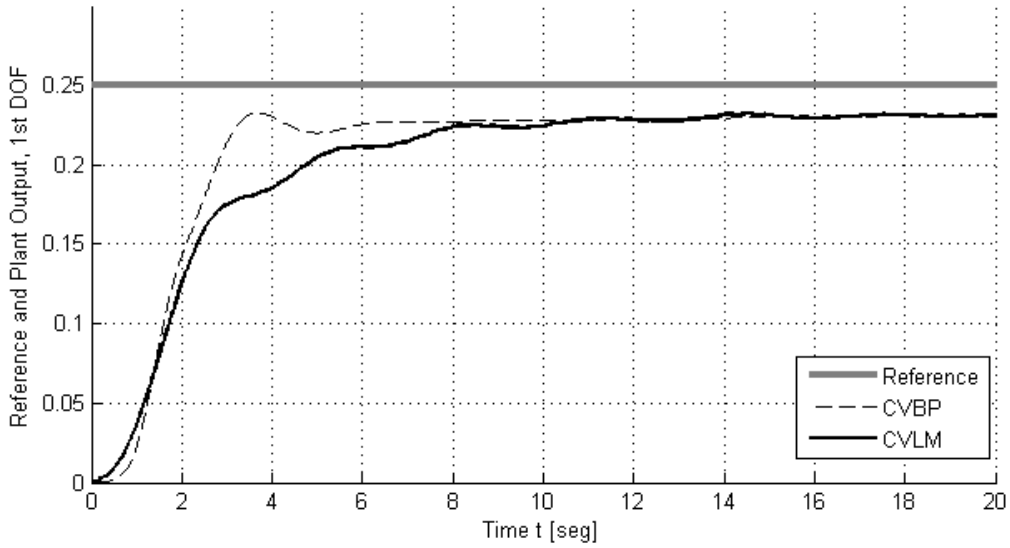


Figura 7.17 Comparación del primer grado de libertad de salida de la planta, para el segundo esquema de control, con las redes neuronales entrenadas con ambos algoritmos

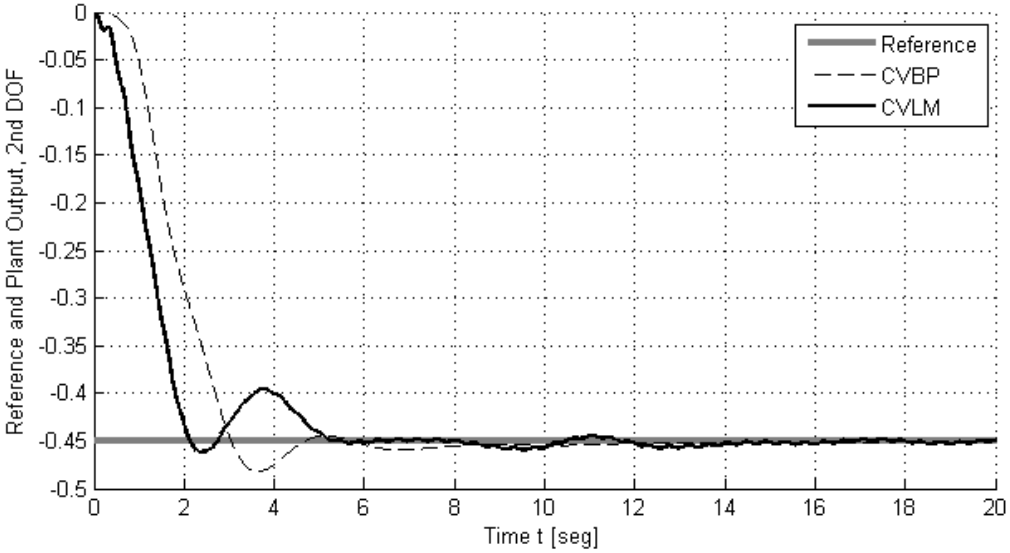


Figura 7.18 Comparación del segundo grado de libertad de salida de la planta, para el segundo esquema de control, con las redes neuronales entrenadas con ambos algoritmos

De las gráficas presentadas para ambas plantas, se puede observar que ambos esquemas de control neuronal adaptable propuestos son efectivos para llevar las señales de salida de la planta a un comportamiento deseado. Esto también es válido para cualquiera de los dos algoritmos de entrenamiento empleados en el ajuste de las **CVRNN**.

El esquema de control sin término integral del error presenta un error de seguimiento cuando la referencia es oscilatoria, o un error en estado estacionario cuando la referencia es fija. Este error se elimina totalmente para el segundo esquema de control donde se agrega la acción integral del error, lo cual valida las suposiciones con las que fue construido el controlador.

Para el caso de la planta **MIMO**, de sus gráficas de salida se observa un sobre–impulso, siendo menor cuando las **CVRNN** son entrenadas con el algoritmo **CVLM**. También se observa que el tiempo de respuesta de las salidas de la planta es más rápido para este algoritmo de entrenamiento.

La Tabla 7.2 muestra el **MSE** final para 100 segundos de simulación (10,000 iteraciones) con ambos algoritmos de entrenamiento y para ambos esquemas de control. De nuevo se observa que el algoritmo de entrenamiento **CVLM** presenta, en general, un mejor desempeño que el **CVBP** en ambas etapas.

Tabla 7.2 MSE final para la planta MIMO, con 100 segundos de simulación en ambos algoritmos, y ambos esquemas de control

	CVBP	**CVLM**
1er Esquema de Control	6.18×10^{-3}	4.67×10^{-3}
2do Esquema de Control	3.44×10^{-3}	1.64×10^{-3}

8. Conclusiones

En este trabajo se presentó una topología de Red Neuronal Recurrente con pesos sinápticos complejos, con una función de activación compleja construida. Luego, para su entrenamiento, se desarrolló el algoritmo de Levenberg–Marquardt recursivo en el dominio de los números complejos, a partir de una modificación del algoritmo de Error de Predicción Recursivo. La introducción del método diagramático con el cual se obtiene la red adjunta mostró una gran facilidad para la obtención de los términos de gradiente necesarios para el algoritmo de Levenberg–Marquardt.

Se propuso un esquema de identificación de sistemas, utilizando una Red Neuronal Compleja Recurrente que aproximara el comportamiento de dos plantas mecánicas, no lineales y oscilatorias, entrenada con el algoritmo desarrollado, dando muy buenos resultados para la etapa de aprendizaje y la etapa de generalización.

Luego, se propusieron dos esquemas de control neuronal adaptable directos, utilizando tres Redes Neuronales Complejas Recurrentes: la primera como identificador, la segunda como controlador por realimentación de estado y la tercera como controlador *feedforward*. Los dos esquemas de control fueron puestos a prueba con las dos plantas mecánicas usadas en la identificación, dando buenos resultados en el desempeño de los controladores. Para el segundo esquema de control, donde un término de acción integral del error fue añadido, se observó la eliminación del error en estado estacionario y error de seguimiento en su totalidad.

Los resultados obtenidos de las simulaciones para el esquema de identificación y los esquemas de control validan la arquitectura de la Red Neuronal Compleja Recurrente propuesta y el algoritmo de aprendizaje complejo de Levenberg–Marquardt recursivo. Cabe mencionar la facilidad de escalamiento del número de parámetros y estados internos de las redes neuronales propuestas, así como su número de entradas y salidas, y como esta no afecta la integración del algoritmo de aprendizaje.

Los resultados numéricos del error medio cuadrático mostraron un mejor desempeño de las redes neuronales al ser entrenadas con este algoritmo, que al ser entrenadas con el algoritmo de *Backpropagation*. Por otro lado, se observó que el algoritmo de

Levenberg–Marquardt tiene mayor costo computacional debido al cálculo recursivo de la matriz de covarianza necesaria para la actualización de los pesos sinápticos de la red neuronal; y presentó mayor sensibilidad a las condiciones iniciales de la red neuronal.

8.1 Trabajo a Futuro

Como trabajo a futuro se busca desarrollar las pruebas de estabilidad, de manera analítica, de las diferentes topologías presentadas en [18] y este trabajo, de Redes Neuronales Complejas Recurrentes al ser entrenadas con los algoritmos de *Backpropagation* y de Levenberg–Marquardt.

También se busca implementar el uso de sistemas de control difuso para la variación de los parámetros de los algoritmos de aprendizaje.

Referencias

[1] S. Haykin. *Neural Networks: A Comprehensive Foundation.* 2nd edition, Ed.: Prentice–Hall, Inc., U.S.A., 1999.

[2] J. R. Hilera González y V. J. Martínez Hernando. *Redes Neuronales Artificiales: Fundamentos, Modelos y Aplicaciones.* 2da edición, Ed,; Alfaomega, Madrid, España, 2010.

[3] K. S. Narendra and K. Parthasarathy. *Identification and Control of Dynamical Systems Using Neural Networks.* IEEE Transactions on Neural Networks, vol. 1, no. 1, pp. 4-27, March 1990.

[4] D. T. Pham and X. Liu. *Neural Networks for Identification, Prediction and Control.* 1st edition, Ed.:Springer–Verlag, U.K., 1995.

[5] I. S. Baruch y J. L. Olivares. *Implementación de un Multi–Modelo Neuronal Jerárquico Para Identificación y Control de Sistemas Mecánicos.* Computación y Sistemas, vol. 9, no. 1, pp. 28-40, 2005.

[6] N. N. Aizenberg, Y. L. Ivaskiv and D. A. Pospelov. *A certain generalization of threshold functions.* Dokrady Akademii Nausk SSSR 196, pp. 1287-1290, Russia 1971.

[7] B. Widrow, J. McCool and M. Ball. *The Complex LMS Algorithm.* Proceedings of the IEEE, vol. 63, pp. 719-720, 1975.

[8] H. Leung and S. Haykin. *The Complex Backpropagation Algorithm.* IEEE Transactions on Signal Processing, vol. 39, no. 9, pp. 2101-2104, 1991.

[9] A. Hirose. *Complex–Valued Neural Networks.* J. Kacprzyk, Studies in Computational Intelligence, 2nd edition, Ed.: Springer–Verlag, vol. 400, Poland, 2012.

[10] L. Ferariu. *Nonlinear System Identification Based on Evolutionary Dynamic Neural Network.* Proceedings of the European Control Conference, article no. 432, Cambridge, U.K., 2003.

[11] A. Minin, Y. Chistyakov, E. Kholodova, H. G. Zimmermann and A. Knoll. *Complex–Valued Open Recurrent Neural Network for Power Transformer Modeling.* International Journal of Applied Mathematics and Informatics, vol. 6, no. 1, pp. 41-48, 2012.

[12] T. Fujiwara, Y. Maeda and H. Ito. *Learning of inverse–kinematics for a robot using high dimensional neural networks.* Proceedings of SICE Annual Conference, pp. 3581-3585, Nagoya, Japan, 2013.

[13] A. Luchetta, S. Manetti and M. C. Piccirilli. Analog System Modeling Based on a Double Modified Complex–Valued Neural Network. The 2013 International Joint Conference on Neural Networks, Dallas, TX., U.S.A., August 2013.

[14] T. Kitajima and T. Yasuno. Output Prediction of Wind Power Generation System Using Complex–Valued Neural Networks. Proceedings of SICE Annual Conference, pp. 3610-3613, Taipei, 2010.

[15] L. Zheng and Z. Wang. *Design of FIR Digital Filters in Complex Domain by Complex–Valued Neural Networks.* Proceedings of the 2010 2nd International Conference on Information Science and Engineering, pp. 4499-4502, Hangzhou, China, 2010.

[16] H. G. Zimmermann and T. Ogawa. *Historical Consistent Complex–Valued Recurrent Neural Network.* ICANN 2011, pp. 185-192, Espoo, Finland, 2011.

[17] S. F. Escalante Magaña. *Identificación, Filtraje y Control de Sistemas No Lineales Usando Redes Neuronales Recurrentes con el Aprendizaje Recursivo de Levenberg–Marquardt.* Tesis de maestría, Departamento de Control Automático del CINVESTAV-IPN, D.F., México, Septiembre 2006.

[18] V. M. Arellano–Quintanilla. *Identificación y Control de Sistemas Dinámicos Utilizando Redes Neuronales Complejas Recurrentes.* Tesis de maestría, Departamento de Control Automático del CINVESTAV-IPN, D.F., México, Febrero 2015.

[19] E. A. Wan and F. Beaufayas. *Diagrammatic Methods for Deriving and Relating Temporal Neural Network Algorithms.* Neural Computation, pp. 182-201, 1996.

[20] R. V. Churchill and J. W. Brown. *Complex Variable and Applications.* 8th edition, Ed.: McGraw–Hill, U.S.A., 2009.

[21] L. V. Ahlfors. *Complex Analysis: An Introduction to the Theory of Analytic Functions of One Complex Variable.* 2nd edition, Ed.: McGraw–Hill Book Company, U.S.A., 1966.

[22] J. E. Marsden and A. J. Tromba. *Vectorial Calculus.* 3rd edition, Ed.: W. H. Freeman and Company, N.Y., U.S.A., 1988.

[23] A, Minin, A. Knoll and H. G. Zimmermann. *Complex–Valued Recurrent Neural Network: From Architecture to Training.* Journal of Signal and Information Processing, pp. 192-197, 2012.

[24] D. H. Brandwood. A Complex Gradient Operator and its Applications in Adaptive Array Theory. IEEE Proceedings on Communications, Radar and Signal Processing, vol. 130, no. 1, pp. 11-16, February 1983.

[25] R. F. H. Fischer. *Precoding and Signal Shaping for Digital Transmission.* 1st edition, Ed.: John Wiley & Sons, Inc., 2002.

[26] I. Aizenberg. *Complex–Valued Neural Networks with Multi–Valued Neurons.* J. Kacprzyk, Studies in Computational Intelligence, 1st edition, Ed.: Springer, 2011.

[27] M. N. Szilagyi and B. Salik. *Neural Networks with Complex Activations and Connection Weights.* Complex Systems 8, vol.2, pp. 115-126, 1994.

[28] K. Matsuoka. *Learning of Neural Networks Using Their Adjoint Systems.* System and Computers, vol. 22, no. 11, pp. 31-41, Japan, 1991.

[29] M. T. Hagan and M. B. Menhaj. *Training Feedforward Networks With the Marquardt Algorithm.* IEEE Transactions on Neural Networks, vol. 5, no. 6, pp. 989.993, November 1994.

[30] R. Fletcher. *Practical Methods of Optimization.* 2nd edition, Ed.: John Wiley & Sons Ltd., U.K., 1987.

[31] P. P. Van Der Smagt. *Minimisation Methods for Training Feedforward Neural Networks.* Neural Networks, vol. 7, no. 1, pp. 1-11, 1994.

[32] S. A. Billings, H. B. Jamaluddin and S. Chen. *A Comparison of the Backpropagation and Recursive Prediction Error Algorithms for Training Neural Networks.* Mechanical Systems and Signal Processing, vol. 5, no. 3, pp. 233-255, May 1991.

[33] V. S. Asirvadam, S. F. McLoose and W. G. Irwin. *Parallel and Separable Recursive Levenberg–Marquardt Training Algorithm.* Proceedings of the 2002 12th IEEE Workshop on Neural Networks for Signal Processing, pp. 129-138, September 2002.

[34] D. J. Tylavsky and G. R. L. Sohie. *Generalization of the Matrix Inversion Lemma.* Proceedings of the IEEE, vol. 74, no. 7, pp. 1050-1052, July 1986.

[35] L. S. Ngia and J. Sjoberg. *Efficient Training of Neural Nets for Nonlinear Adaptive Filtering Using a Recursive Levenberg–Marquardt Algorithm.* IEEE Transactions on Signal Processing, vol. 48, pp. 1915-1927, july 2000.

[36] I. S. Baruch and C. R. Mariaca–Gaspar. *A Levenberg–Marquardt Learning Applied for Recurrent Neural Identification and Control of Wastewater Treatment Bioprocess.* International Journal of Intelligent Systems, no. 24, pp. 1094-1114, 2009.

[37] I. S. Baruch and E. P. Reynaud. *Control of Nonlinear Dynamical Systems Using Levenberg–Marquardt Learning Algorithm for Recurrent Complex–Valued Neural Networks.* Proceedings of the 19th International Conference on Circuits, Systems, Communications and Computers, pp. 398-403, Zakynthos, Greece, 2015.

[38] E. P. Reynaud and I. S. Baruch. *Identification of Dynamical Systems Using Levenberg–Marquardt Learning Algorithm for Recurrent Complex–Valued Neural Networks.* Proceedings of the 19th International Conference on Circuits, Systems, Communications and Computers, pp. 331-336, Zakynthos, Greece, 2015.

[39] I. S. Baruch and V. M. Arellano–Quintana. *Identification and Control of Oscillatory Dynamical Systems Using Recurrent Complex–Valued Neural Networks.* Proceedings of the 18th International Conference on Circuits, Systems, Communications and Computers, pp. 534-539, Santorin, Greece, 2014.

[40] V. M. Arellano–Quintana and I. S. Baruch. *Identification of Dynamical Systems Using Recurrent Complex–Valued Neural Networks.* Proceedings of the 18th International Conference on Circuits, Systems, Communications and Computers, pp. 74-79, Santorin, Greece, 2014.

[41] K. Ogata. *Modern Control Engineering.* 4th edition. Ed.: Pearson Education International, U.S.A., 2002.

[42] S. S. Ge, T. H. Lee and C. J. Harris. *Adaptive Neural Network Control of Robotic Manipulators.* World Scientific Series in Robotics and Intelligent Systems, vol. 19, Ed.: World Scientific Publishing Co. Pte. Ltd., U.K., 1998.

Anexo – Artículos Publicados

Derivadas de este trabajo se realizaron las siguientes publicaciones:

- E. P. Reynaud and I. S. Baruch. *Identification of Dynamical Systems Using Levenberg–Marquardt Learning Algorithm for Recurrent Complex–Valued Neural Networks.* In: V. Mladenov et al., Recent Advances in Systems, Series: Recent Advances in Electrical Engineering Series – 52, Proceedings of the 19th International Conference on Circuits, Systems, Communications and Computers, pp. 331-336, Zakynthos Island, Greece, July 2015. ISSN: 1790-5117, ISBN: 978-1-61804-321-4.
- I. S. Baruch and E. P. Reynaud. *Control of Nonlinear Dynamical Systems Using Levenberg–Marquardt Learning Algorithm for Recurrent Complex–Valued Neural Networks.* In: V. Mladenov et al., Recent Advances in Systems, Series: Recent Advances in Electrical Engineering Series – 52, Proceedings of the 19th International Conference on Circuits, Systems, Communications and Computers, pp. 398-403, Zakynthos Island, Greece, July 2015. ISSN: 1790-5117, ISBN: 978-1-61804-321-4.

Identification of dynamical systems using Levenberg-Marquardt learning algorithm for recurrent complex-valued neural networks

Edmundo P. Reynaud and Ieroham S. Baruch.

***Abstract*—**The use of the Levenberg-Marquardt learning algorithm is scarce in applications with Complex-Valued Artificial Neural Networks, mainly because of the difficulty in obtaining the correction terms for the weights to be trained with an algebraic approach. In this work, a diagrammatic based procedure is proposed for the derivation and implementation of said algorithm for a Complex-Valued Recurrent Neural Network. Furthermore, we get some comparative simulation results between the Complex-Valued Back-Propagation algorithm and our proposed Levenberg-Marquardt algorithm, for the application of a Recurrent Neural Network on a plant identification problem.

***Keywords*—**Complex-valued neural networks, Levenberg-Marquardt learning algorithm, System identification.

I. Introduction

THERE has been a rise in the use of Complex-Valued Artificial Neural Networks (**CVNN**) in the last decade, this is because models of certain physical phenomena have a better approach in the complex-domain than its real-domain counterparts. Examples of these are seen in the fields of electromagnetism, optics, power systems, electronic systems, and digital image processing, to name a few. ([1], [2], [3]).

Although its applications are not as spread in the field of mechanical systems, there are some works (see [3], [7]) that propose the use of **CVNN** for identification and control of mechanical plants that report quite good results.

The extension to the complex-domain of Recurrent Neural Networks enable its use for tasks that can be solved with the real-domain approach, but with better performance and less computational cost. For example, in [4], **CVNN** are used to identify analogic electric circuits; in [5] authors use a **CVNN** as a behavior predictor for a wind-powered generation system; or in [6], a **CVNN** is proposed for the design of FIR digital filters.

In [8] and [9] a method is proposed to derive a Back-Propagation learning algorithm for a **CVNN**, applied to the identification and control of a **MIMO** mechanical plant with nonlinear and oscillatory dynamics. Following that line of work, we use the diagrammatic methods presented in [10] to derive a second order Levenberg-Marquardt learning algorithm for the same proposes. We further make a comparison in performance between the complex-valued Back-Propagation used in [8], [9] and the Levenberg-Marquardt learning algorithms developed through this work.

E. P. Reynaud is a Master in Science student in the Department of Automatic Control, CINVESTAV-IPN, D.F, Mexico. E-mail: eperez@ctrl.cinvestav.mx

I. S. Baruch is with the Department of Automatic Control, CONVESTAV-IPN, D.F, Mexico. E-mail: baruch@ctrl.cinvestav.mx

II. Topology and Levenberg-Marquardt Learning of Real-Valued Recurrent Neural Networks

We consider the Real Valued Recurrent Neural Network (**RVRNN**) topology shown in Fig 1.

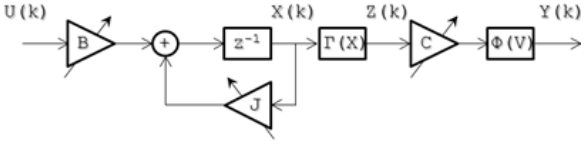


Fig.1 Block-diagram representation of a RVRNN

This network consists of an input layer, a recurrent hidden layer with linear behavior, and an output layer. The description of the network is given by the following equations:

$$X(k+1) = JX(k) + BU(k) \tag{1}$$

$$Z(k) = \Gamma[X(k)] \tag{2}$$

$$V(k) = CZ(k) \tag{3}$$

$$Y(k) = \Phi[V(k)] \tag{4}$$

Where: $X \in \mathbb{R}^n$ is the state vector, $U \in \mathbb{R}^m$ is the input vector, $Y \in \mathbb{R}^L$ is the output vector, $Z \in \mathbb{R}^n$ is the state vector of the hidden layer, $\Gamma[\cdot]$, $\Phi[\cdot]$ are vector-valued activation functions with compatible dimensions; J, B, C are the weight state, weight input and weight output matrices with dimensions $(n \times n)$, $(n \times m)$ and $(L \times n)$ respectively.

The matrix J is a diagonal matrix with blocks J_j as its elements, which must meet the stability condition:

$$|J_j| < 1, \qquad j = 1, 2, \ldots, N \tag{5}$$

This **RVRNN** has a canonical Jordan form, which means that it has the minimum number of parameters subject to training, as well as complete parallel integration. It also presents controllability and observability properties, and

stability conditions. The performance index to be minimized used in the training algorithm is given by the equation:

$$\zeta(k) = \frac{1}{2}\sum_j [E_j(k)]^2, \qquad \zeta = \frac{1}{N_e}\sum_k \zeta(k) \tag{6}$$

$$E(k) = Y_p(k) - Y(k) \tag{7}$$

Where equation (7) describes the identification error. The instantaneous Means Squared Error (**MSE**) $\zeta(k)$ is used in on-line applications, while the total **MSE** denoted by ζ is used for one epoch N_e in off-line applications. The general recursive real-valued Levenberg-Marquardt (**RVLM**) learning algorithm for any weight vector W is described by the following equation (see [11]):

$$W(k+1) = W(k) + P(k)\cdot DY[W(k)]\cdot E(k), \qquad |W_j| < W_0 \tag{8}$$

Where: W_0 is a restricted region for the weight W_j, P can be interpreted as the covariance matrix of the weight's estimation, and $DY[W]$ is the local gradient component of the output of the network with respect to W. Applying certain diagrammatic rules (see [10]) to the **RVRNN** given in Fig.1, we obtain its adjoint topology, which is shown in Fig.2.

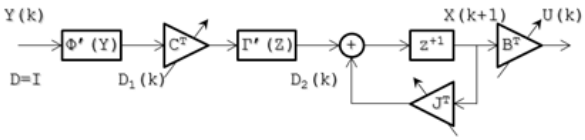


Fig.2 Block-diagram of the adjoint network for the RVRNN

Then gradient terms can be derived directly from the adjoint topology, which are described by the following equations:

$$D_1(k) = \Phi'[Y(k)]\cdot D \tag{9}$$

$$D_2(k) = \Gamma'[Z(k)]\cdot C^T \cdot D_1(k) \tag{10}$$

$$DY[C(k)] := \partial Y(k)/\partial C(k) = D_1(k)\cdot Z^T(k) \tag{11}$$

$$DY[J(k)] := \partial Y(k)/\partial J(k) = D_2(k)\cdot X^T(k) \tag{12}$$

$$DY[B(k)] := \partial Y(k)/\partial B(k) = D_2(k)\cdot U^T(k) \tag{13}$$

Where: $D = I$ is a unitary input for the adjoint topology. The matrix P is computed recursively using the following equation:

$$P(k) = \alpha^{-1}[P(k-1) - P(k-1)\cdot \Omega_{W(k)}\cdot S^{-1}_{W(k)}\cdots \cdot \Omega^T_{W(k)}\cdot P(k-1)] \tag{14}$$

Where the matrices Ω_W and S_W are given by:

$$\Omega^T_{W(k)} = \begin{bmatrix} DY^T[W(k)] \\ 0 \quad \cdots \quad 1 \quad \cdots \quad 0 \end{bmatrix} \tag{15}$$

$$S_{W(k)} = \alpha\Lambda + \Omega^T_{W(k)}\cdot P(k-1)\cdot \Omega_{W(k)} \tag{16}$$

$$\Lambda^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & \rho \end{bmatrix} \tag{17}$$

The matrices P and S_w have dimensions $(N_w \times N_w)$ and (2×2) respectively, where N_w is the number of weights in the vector W. The matrix Ω_w has dimension $(N_w \times 2)$, the second row of Ω^T_w consists of $(N_w - 1)$ zeroes and a unit element in the i-th position computed by:

$$i = k \cdot \mathrm{mod}(N_W) + 1 \tag{18}$$

The parameters for the algorithm should be restricted as follows:

$$10^{-6} \le \rho \le 10^{-4}, \qquad 0.97 \le \alpha \le 1.00$$
$$10^3 \le P(0) \le 10^6 \tag{19}$$

III. Topology and Levenberg-Marquardt Learning of Complex-Valued Recurrent Neural Networks

We consider a **CVRNN** topology with real-valued input $U(k)$ and output $Y(k)$ signals, complex-valued internal state $X(k)$ and hidden state $Z(k)$ vectors, and complex-valued J, B, C weight matrices.

The matrix J is a diagonal matrix with the neuron eigenvalues as its block-elements. Complex eigenvalues come in pairs of complex conjugate numbers, while real eigenvalues come as single values. For the matrices B and C, its elements also come in pairs of complex conjugate numbers or single real numbers, and its positions are related to the positions of the elements of J.

Although the internal state and parameter matrices of the neuron are complex-valued, the complex conjugate pairs produce a real-valued output of the network, which yields a real-valued identification error signal. That way we can use the same performance index as in the **RVRNN** case to be minimized, given in (6).

Because the vector $Z(k)$ is complex-valued, the activation function $\Gamma[\cdot]$ must be also complex-valuated; not so for the activation function $\Phi[\cdot]$ given that the product $CZ(k)$ is real-valued. In this work we consider two types of hyperbolic tangents for the activation function $\Gamma[\cdot]$, one natural and one constructed, which yield two different topologies for the **CVRNN**.

A. CVRNN Topology with First Type Activation Function.

The first activation function considered is described by the following equation:

$$f(z) = \tanh(z),$$
$$z \in \mathbb{C} \setminus \left\{ z : z = 0 \pm \frac{2n-1}{2}\pi i, \quad \forall n \in \mathbb{N} \right\} \tag{20}$$

This activation function has singularities at points $\mathrm{Re}(z) = 0$, $\mathrm{Im}(z) = \{\pm\frac{1}{2}\pi, \pm\frac{3}{2}\pi, \ldots\}$ of the complex domain. Because of this, we restrict the domain of said function near these points of singularity.

The block diagram of the **CVRNN** that uses this activation function is the same as the one shown in Fig.1. The mathematical description of this topology is the same as in equations (1)-(4) but now with some complex-valued variables. The vectors and matrices for the **CVRNN** topology are given as follows: $J \in \mathbb{C}^{n\times n}$ the feedback weight matrix, $B \in \mathbb{C}^{n\times m}$ the input weight matrix, $C \in \mathbb{C}^{L\times n}$ the output weight matrix, $X \in \mathbb{C}^n$ the internal state vector, $Z \in \mathbb{C}^n$ the hidden layer state vector, $U \in \mathbb{R}^m$ network input, $Y \in \mathbb{R}^L$ the network output, $\Gamma[\cdot]$ a complex-valued activation function given by (20), and $\Phi[\cdot]$ a real-valued activation function $f(\cdot) = \tanh(\cdot)$; n, m, L are the number of internal states, inputs and outputs respectively.

The matrix J is a diagonal matrix where its block-elements must meet the stability restriction given in (5). This type of representation allows us to apply diagrammatic rules to derive the adjoint network shown in Fig.3, and use it to derive the gradient terms needed for the learning algorithm.

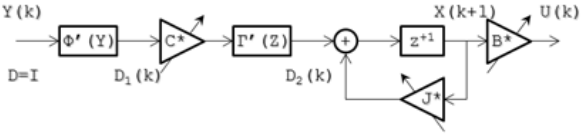


Fig.3 Block-diagram representation of the adjoint network for the CVRNN with first activation function

The complex-valued Levenberg-Marquardt (**CVLM**) algorithm for any weight vector W is described by the following equation:

$$W(k+1) = W(k) + P(k)\cdot DY[W(k)]\cdot E(k), \quad |W_j| < W_0 \tag{21}$$

Where: W_0 is a restricted region for the weight W_j. The gradient terms for the complex-valued network with the first type activation function are described by the following equations:

$$D_1(k) = \Phi'[Y(k)]\cdot D \tag{22}$$

$$D_2(k) = \Gamma'[Z(k)]\cdot C^*\cdot D_1(k) \tag{23}$$

$$DY[C(k)] := \partial Y(k)/\partial C(k) = D_1(k)\cdot Z^*(k) \tag{24}$$

$$DY[J(k)] := \partial Y(k)/\partial J(k) = D_2(k)\cdot X^*(k) \tag{25}$$

$$DY[B(k)] := \partial Y(k)/\partial B(k) = D_2(k)\cdot U^*(k) \tag{26}$$

Where the $(*)$ superscript denotes a complex conjugate and transposed vector, $D = I$ is a real-valued identity matrix input for the adjoint topology. The matrix P is computed recursively using the following equation:

$$P(k) = \alpha^{-1}[P(k-1) - P(k-1)\cdot\Omega_{W(k)}\cdot S^{-1}_{W(k)}\cdots \cdot\Omega^*_{W(k)}\cdot P(k-1)] \tag{27}$$

Where the matrices Ω_W and S_W are given by:

$$\Omega^*_{W(k)} = \begin{bmatrix} DY^*[W(k)] \\ 0 \quad \cdots \quad 1 \quad \cdots \quad 0 \end{bmatrix} \tag{28}$$

$$S_{W(k)} = \alpha\Lambda + \Omega^*_{W(k)}\cdot P(k-1)\cdot\Omega_{W(k)} \tag{29}$$

$$\Lambda^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & \rho \end{bmatrix} \tag{30}$$

Again, the matrices P and S_w have dimensions $(N_w \times N_w)$ and (2×2) respectively, where N_w is the number of weights in the vector W. The matrix Ω_w has dimension $(N_w \times 2)$, the second row of Ω^*_w consists of $(N_w - 1)$ zeroes and a unit element in the i-th position computed by (18). The parameters for the algorithm must keep the same restrictions as in (19).

B. CVRNN Topology with Second Type Activation Function.

The second activation function considered is described by the following equation:

$$f(z) = \tanh(\mathrm{Re}(z)) + i\tanh(\mathrm{Im}(z)), \qquad z \in \mathbb{C} \tag{31}$$

This activation function doesn't have any singularities, so it doesn't need any restrictions in its domain as it did the first activation function. The block diagram of the **CVRNN** that uses this activation function is shown in Fig.4.

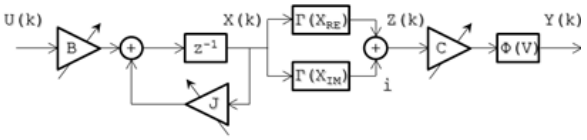


Fig.4 Block-diagram representation for the CVRNN with second activation function

The description of the RVRNN is given by the following equations:

$$X(k+1) = JX(k) + BU(k) \tag{32}$$

$$Z(k) = \Gamma[X_{\mathrm{Re}}(k)] + i\Gamma[X_{\mathrm{Im}}(k)] \tag{33}$$

$$V(k) = CZ(k) \tag{34}$$

$$Y(k) = \Phi[V(k)] \tag{35}$$

The dimensions and domains of each vector and matrix in this **CVRNN** are the same as in the previous network. Applying the complex-valued diagrammatic rules we obtain the adjoint network, shown in Fig.5.

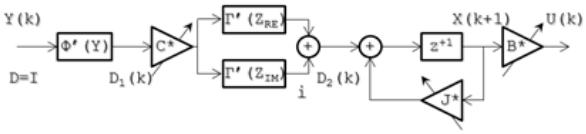


Fig.5 Block-diagram representation of the adjoint network for the CVRNN with second activation function

From this adjoint network, we derive the gradient terms needed for the **CVLM** learning algorithm, which are described by the following equations:

$$D_1(k) = \Phi'[Y(k)] \cdot D \tag{36}$$

$$D_2(k) = (\Gamma'[Z_{Re}(k)] \cdot \mathrm{Re}(C^*) \cdots + \Gamma'[Z_{Im}(k)] \cdot \mathrm{Im}(C^*)) \cdot D_1(k) \tag{37}$$

$$DY[C(k)] := \partial Y(k)/\partial C(k) = D_1(k) \cdot Z^*(k) \tag{38}$$

$$DY[J(k)] := \partial Y(k)/\partial J(k) = D_2(k) \cdot X^*(k) \tag{39}$$

$$DY[B(k)] := \partial Y(k)/\partial B(k) = D_2(k) \cdot U^*(k) \tag{40}$$

Where: $D = I$ is a real-valued identity matrix input for the adjoint topology. Then we apply the **CVLM** equations given by (18), (27)-(30) with parameters restricted by (19).

IV. Complex-Valued Neural Solution for the Nonlinear Identification Problem

We now illustrate the application of the **CVRNN** for nonlinear oscillatory system identification.

A. Description of the Nonlinear System Model.

The plant subject for identification addressed in this work is an idealized nonlinear model of a flexible-joint robot arm, illustrated in Fig.6. The flexibility at the joint is caused by a harmonic drive, which is a type of gear mechanism with high torque transmission, low backlash and compact size.

The robot joint model consists of an actuator connected to a load through a torsional spring, which represents the joint flexibility. We consider the motor torque and the angular position of the link as the input signal $u(t)$ and the output signal $y(t)$ respectively, making this a **SISO** system. The equations that describe the motion of the flexible joint are as follows:

$$J_l\ddot{\theta}_l + B_l\dot{\theta}_l + Mgl\sin\theta_l + k(\theta_l - \theta_m) = 0 \tag{41}$$

$$J_m\ddot{\theta}_m + B_m\dot{\theta}_m - k(\theta_l - \theta_m) = \mathrm{u} \tag{42}$$

Where: J_l, J_m are the link and motor inertial coefficients, B_l, B_m are the link and motor damping coefficients, k is the torsion stiffness coefficient of the harmonic drive gear, M is the mass of the link, L is the length between the shaft and the center of mass of the link, θ_l, θ_m are the angular positions of the link and the rotor of the motor respectively.

The input and output signals are discretized with a sampling period (τ) in order to use a discrete neural network approach for its identification. This is an oscillatory system, described by two nonlinear second order differential equations.

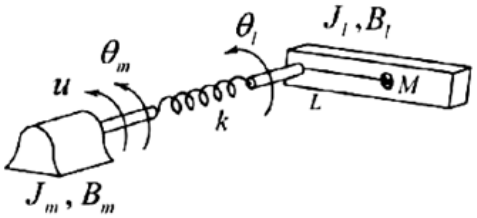


Fig.6 Flexible-joint robotic arm

B. Plant Identification.

For the plant identification, we use the scheme illustrated on Fig.7. Here, the desired output vector for our **CVRNN** is the output of the plant, with which we produce an error signal that is fed to the learning algorithm for the network. The identification objective is to adjust the weight parameters of the **CVRNN** such that its output follows the plant output, minimizing the total **MSE** given by the performance index (6).

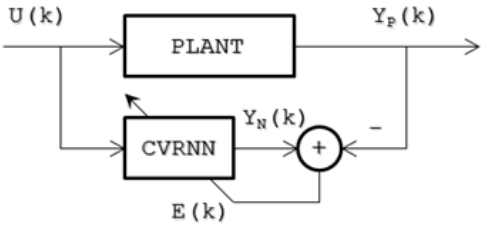


Fig.7 Identification scheme

C. Simulation and Results.

We test the **CVLM** learning algorithm applied to a **CVRNN** with the first activation function, for nonlinear oscillatory plant identification with a simulation using MATLAB.

The simulation has two stages: in the learning stage, weights are adjusted until convergence to a steady value and the **CVRNN** output matches the output of the plant; in the generalization stage, we fix the weight parameters and apply a

different input for the plant and the **CVRNN**, to validate its learning by comparing both outputs. Next, we make a comparison between the **CVBP** and the **CVLM** learning algorithms.

As a comparison measure, we use the total **MSE** for learning and generalization stages. This section describes the simulation settings used and the results obtained. The input signals used on the learning stage $u_L(t)$ and generalization stage $u_G(t)$ are given by:

$$u_L(t) = \sin\left(\tfrac{1}{10}t\right) + 0.5\sin\left(\tfrac{1}{25}t\right) \tag{43}$$

$$u_G(t) = 0.5\sin\left(\tfrac{1}{10}t\right) + 0.8\sin\left(\tfrac{1}{20}t\right) \tag{44}$$

For the **CVRNN** we used dimensions $n=3, m=1, L=1$, with initial conditions of the internal states vector $X(0) = [0 \quad 0 \quad 0]^T$, random initial conditions for each weight parameter in the interval $[-0.5, 0.5]$; simulation time of $T = 500s$ and sampling time $\tau = 0.01$. For the **CVBP** algorithm we used the parameters: $\eta = 0.05$ and $\alpha = 0.005$. For the **CVLM** algorithm we used the parameters: $\alpha = 0.9775$, $\rho = 1\times10^{-4}$, $P_I(0) = 1\times10^{6}$, $P_B(0) = 1\times10^{4}$ and $P_C(0) = 1\times10^{5}$. For both cases, the second activation function is used.

For the **CVBP** algorithm, Fig.8, a) shows the plant and the neural network outputs, and b) the instantaneous **MSE** for the learning stage. Fig.9 a), b) shows the same signals for the generalization stage.

For the **CVLM** algorithm, Fig.10, a) shows the plant and the neural network outputs, and b) the instantaneous **MSE** for the learning stage. Fig.11, a), b) shows the same signals for the generalization stage.

For both algorithms, we observe a fast convergence of the neural network output to the plant output, while the **MSE** shows a decreasing behavior, for the learning stage. For the generalization stage, where the weight parameters are fixed and the input to the plant and neural network is changed, we observe a good performance of the output of the network and the **MSE** in general.

Table I shows the final **MSE** of the simulations for the **CVBP** and **CVLM** learning algorithms, for both learning and generalization stages.

Table I. Final MSE of both learning algorithms for the learning and generalization stages.

	CVBP	**CVLM**
Learning	6.11×10^{-4}	0.18×10^{-4}
Generalization	25.44×10^{-4}	20.84×10^{-4}

We observed from the total **MSE** for both stages that the **CVLM** learning algorithm has a better performance compared to the **CVBP** learning algorithm.

We also observe that the **CVLM** learning algorithm tends to be more sensible to the initial conditions of its weight parameters. Nevertheless, this sensibility doesn't affect the performance and convergence of the learning stage for the **CVRNN**.

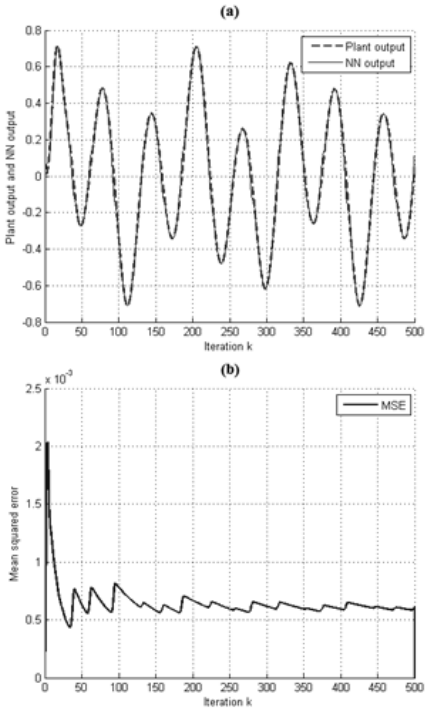


Fig.8 CVBP learning stage, a) Plant output and NN output signals, b) Instantaneous MSE

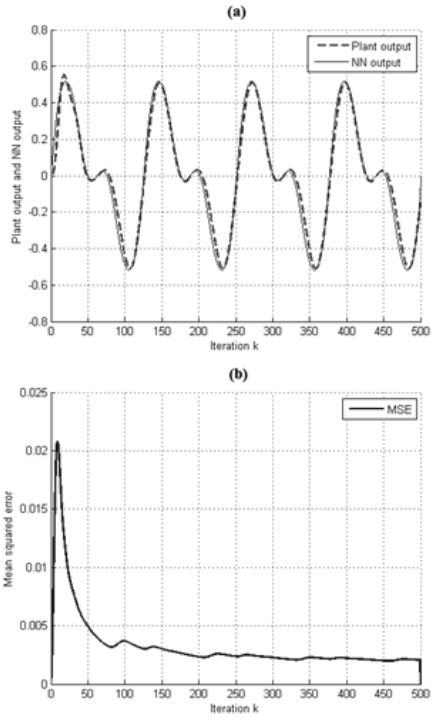


Fig.9 CVBP generalization stage, b) Plant output and NN output signal, b) Instantaneous MSE

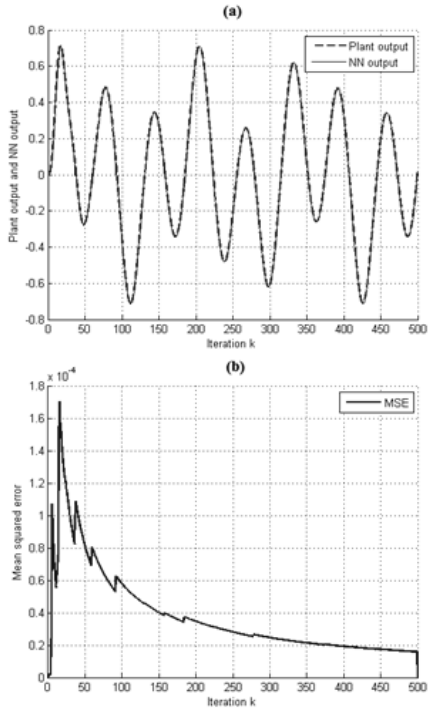


Fig.10 CVLM learning stage, a) Plant output and NN output signals, b) Instantaneous MSE

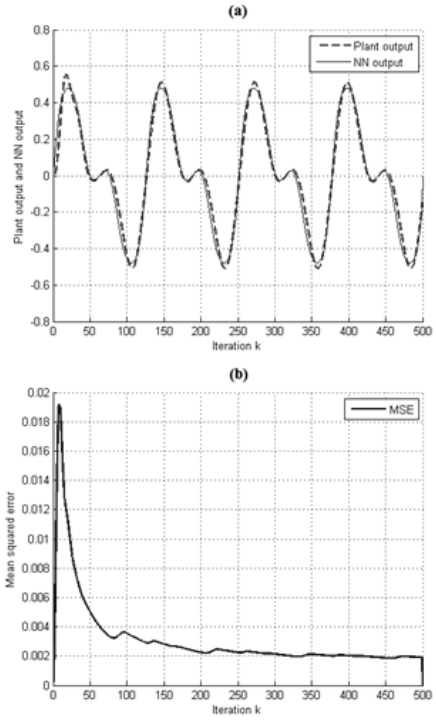


Fig.11 CVLM generalization stage, (b) Plant output and NN output signal, (b) Instantaneous MSE

V. Conclusions

In the present article we proposed a Complex-Valued Levenberg-Marquardt learning algorithm for a Recurrent Neural Network, using a diagrammatic approach to derive the gradient terms and recursive calculations of the weight's covariance matrix, needed for the full implementation of said algorithm. We then applied the Recurrent Neural Network in the complex domain for identification of a dynamic, nonlinear, and oscillatory mechanical plant. The comparative results obtained between the Levenberg-Marquardt and Back-Propagation learning algorithms show a better performance of the neural network for the former algorithm, both in the learning and generalization stages.

Acknowledgment

Master Degree student Edmundo Pérez Reynaud would like to thank CONACyT, Mexico, for the student grant received during his studies at the Department of Automatic Control, CINVESTAV-IPN, Mexico (grant number: 364197, holder number: 295620).

References

[1] A. Hirose. "Complex-Valued Neural Networks." 2nd ed., S.i.C. Intelligence, Springer-Verlag, vol. 400, 2012.

[2] A. Minin, Y. Chistyakov, E. Kholodova, H.G. Zimmermann, A. Knoll. "Complex-Valued Open Recurrent Neural Network for Power Transformer Modeling." *International Journal of Applied Mathematics and Informatics*, vol. 6, no. 1, pp. 41-48, 2012.

[3] L. Feraru, "Nonlinear System Identification Based on Evolutionary Dynamic Neural Networks." *Proc. Of European Control Conference*, Cambridge, UK, paper no. 432, 2003.

[4] A. Luchetta, S. Manetti, M. C. Piccirilli. "Analog System Modeling Based on a Double Modified Complex-Valued Neural Networks." *The 2013 International Joint Conference on Neural Networks*, 2013.

[5] T. Kitajima, T. Yasuno. "Output Prediction of Wind Power Generation System Using Complex-Valued Neural Network." *Proc. Of SICE Annual Conference*, 2010.

[6] L. Zheng, Z. Wang. "Design of FIR Digital Filters in Complex Domain by Complex-Valued Neural Networks." *2010 2nd International Conference on Information Science and Engineering*, Hangzhou, China, 2010.

[7] A. Hirose. "Motion Controls Using Complex-Valued Networks with Feedback Loops." *Proc. Of IEEE International Conference on Neural Networks*, vol.1 San Francisco, CA, pp. 156-161, 1993.

[8] V. M. Arellano-Quintana, I. S. Baruch. "Identification of Dynamical Systems Using Recurrent Complex-Valued Neural Networks." In *Proc. Of the 18th International Conference on Circuits, Systems, Communications and Computers*, Santorini, Greece, pp. 74-79, 2014.

[9] I. S. Baruch, V. M. Arellano-Quintana. "Identification and Control of Oscillatory Dynamical Systems Using Recurrent Complex-Valued Neural Networks." In *Proc. Of the 18th International Conference on Circuits, Systems, Communications and Computers*, Santorini, Greece, pp. 534-539, 2014.

[10] E. A. Wan, F. Beaufays. "Diagrammatic Methods for Deriving and Relating Temporal Neural Network Algorithms." *Neural Computation*, pp. 182-201, 1996.

[11] I. S. Baruch, C.R. Mariaca-Gaspar. "A Levenberg-Marquardt Learning Applied for Recurrent Neural Identification and Control of Wastewater Treatment Bioprocess." *International Journal of Intelligent Systems*, no. 24, pp.1094-1114, 2009.

Control of nonlinear dynamical systems using Levenberg-Marquardt learning algorithm for recurrent complex-valued neural networks

Ieroham S. Baruch and Edmundo P. Reynaud

***Abstract*—**The present work follows the obtained results in the development of a Complex-Valued Recursive Levenberg-Marquardt learning algorithm, applied in the training of Complex-Valued Recurrent Neural Networks. These networks are used in two proposed adaptive neural control schemes, applied to a nonlinear, mechanical plant with multiple inputs and outputs to validate the quality of the learning algorithm. Furthermore, we make a comparative test between this and a Complex-Valued Back-Propagation learning algorithm for the same control schemes, which yields good results.

***Keywords*—**Complex-valued neural networks, Adaptive control, Nonlinear control, Levenberg-Marquardt learning algorithm, System identification.

I. Introduction

THE use of Artificial Neural Networks is fundamental in the field of adaptive control, mainly because of its application in a wide range of dynamical systems of different nature, robustness, high-scale integration and popularity. Even so, the use of Neural Networks in the complex domain for control tasks is still low in comparison to its real-valued counterparts.

The use of Complex-Valued Neural Networks is preferred in systems with an oscillatory behavior, or systems where its nature can be better codified in the domain of complex numbers, such as electromagnetic and optical phenomena, electric power systems, electronic and information systems to name a few (see [1], [2], [3]).

Nonetheless their use is not limited to these fields. For example, in [4] the authors use a Complex-Valued Recurrent Neural Network (**CVRNN**) to generate cooperative motion trajectories for a pair of robots. In [5], a High-Dimension Complex-Valued Neural Network is used to control the end effector of a three-dimensional robotic manipulator. In a previous work (see [6]) we used **CVRNN** in the identification of a nonlinear mechanical system. The present article follows the works of [7] and [8], where a Complex-Valued Back-Propagation (**CVBP**) learning algorithm is devised for a **CVRNN**, which is used for the identification and control of a nonlinear oscillatory plant. In the same way, we developed a second order recursive Complex-Valued Levenberg-Marquardt (**CVLM**) learning algorithm in the complex domain, applied to the training of Recurrent Neural Networks used in the identification and control of a nonlinear mechanical plant. Furthermore, we used simulation results to validate our algorithm and to make a comparison with the **CVBP** algorithm proposed in [7].

I. S. Baruch is with the Department of Automatic Control, CONVESTAV-IPN, D.F, Mexico. E-mail: baruch@ctrl.cinvestav.mx

E. P. Reynaud is a Master in Science student in the Department of Automatic Control, CINVESTAV-IPN, D.F, Mexico. E-mail: eperez@ctrl.cinvestav.mx

II. Complex-Valued Recurrent Neural Network Topology and the Recurrent Levenberg-Marquardt Learning Algorithm

Let us consider a **CVRNN** with a constructed activation function given by the following equation:

$$f(z) = \Gamma(\mathrm{Re}(X)) + i\Gamma(\mathrm{Im}(X)), \quad z \in \mathbb{C} \tag{1}$$

This function has no singular points in its domain and is similar to the one used in [7], [8]. The **CVRNN** with this activation function has a diagrammatic representation shown in Fig 1.

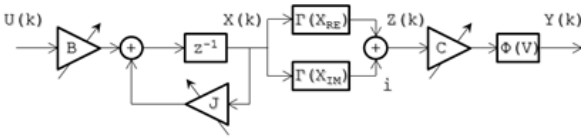


Fig.1 Block-diagram representation of a CVRNN

And its mathematical description is given by the following equations:

$$X(k+1) = JX(k) + BU(k) \tag{2}$$

$$Z(k) = \Gamma[X_{\mathrm{Re}}(k)] + i\Gamma[X_{\mathrm{Im}}(k)] \tag{3}$$

$$V(k) = CZ(k) \tag{4}$$

$$Y(k) = \Phi[V(k)] \tag{5}$$

Where: $X \in \mathbb{C}^n$ is the state vector, $Z \in \mathbb{C}^n$ is the state vector of the hidden layer, $U \in \mathbb{R}^m$ is the input vector, $Y \in \mathbb{R}^L$ is the output vector; n, m, L are the number of internal states, inputs and outputs respectively.

$J \in \mathbb{C}^{n \times n}$, $B \in \mathbb{C}^{n \times m}$ and $C \in \mathbb{C}^{L \times n}$ are the feedback, input and output weight matrices respectively, and they constitute the weight parameters from the neural network subject to

training. The matrix J is a diagonal matrix with the neuron eigenvalues as its block-elements. Complex eigenvalues come in pairs of complex conjugate numbers, while real eigenvalues come as single numbers. For the matrices B and C, its elements also come in pairs of complex conjugate numbers or single real numbers, and its positions are related to the positions of the elements of J.

The diagonal block-elements J_j of the J matrix must meet the stability condition:

$$|J_j| < 1, \qquad j = 1, 2, \ldots, N \tag{6}$$

The map $\Gamma : \mathbb{R}^n \to \mathbb{R}^n$ is a real-valued vector function. Given that the product $CZ(k)$ is a real-valued vector, the map $\Phi : \mathbb{R}^L \to \mathbb{R}^L$ is also a real-valued vector function.

This **CVRNN** has a canonical Jordan form; this means that it has the minimum number of parameters subject to training, as well as complete parallel integration. It also presents properties of dynamical systems such as controllability, observability, and stability conditions.

Although the internal state and parameter matrices of the neuron are complex-valued, the complex conjugate pairs produce a real-valued output of the network, which yields a real-valued identification error signal. The performance index to be minimized used in the training algorithm is given by the equation:

$$\zeta(k) = \frac{1}{2}\sum_j [E_j(k)]^2, \qquad \zeta = \frac{1}{N_e}\sum_k \zeta(k) \tag{7}$$

$$E(k) = Y_p(k) - Y(k) \tag{8}$$

Where equation (8) describes the error between the desired output Y_p and the real output Y of the network. The instantaneous Mean Squared Error (**MSE**) $\zeta(k)$ is used on-line applications, while the total **MSE** denoted by ζ is used for one epoch N_e in off-line applications.

The **CVLM** algorithm for any weight vector W is described by the following update equation (see [6],[10]):

$$W(k+1) = W(k) + P(k)\cdot DY[W(k)]\cdot E(k), \qquad |W_j| < W_0 \tag{9}$$

Where: W_0 is a restricted region for the weight W_j, P can be interpreted as the covariance matrix of the weight's estimation, and $DY[W]$ is the local gradient component of the output of the network with respect to W.

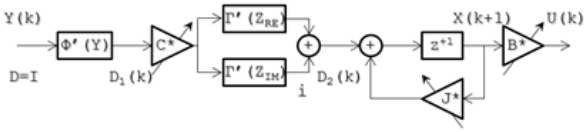


Fig.2 Block-diagram of the adjoint network for the CVRNN

Applying certain diagrammatic rules (see [9]) to the **CVRNN** given in Fig 1, we obtain its adjoint topology, which is shown in Fig 2. Then gradient terms can be derived directly from the adjoint topology, which are described by the following equations:

$$D_1(k) = \Phi'[Y(k)]\cdot D \tag{10}$$

$$D_2(k) = (\Gamma'[Z_{\mathrm{Re}}(k)]\cdot \mathrm{Re}(C^*)\cdots + \Gamma'[Z_{\mathrm{Im}}(k)]\cdot \mathrm{Im}(C^*))\cdot D_1(k) \tag{11}$$

$$DY[C(k)] := \partial Y(k)/\partial C(k) = D_1(k)\cdot Z^*(k) \tag{12}$$

$$DY[J(k)] := \partial Y(k)/\partial J(k) = D_2(k)\cdot X^*(k) \tag{13}$$

$$DY[B(k)] := \partial Y(k)/\partial B(k) = D_2(k)\cdot U^*(k) \tag{14}$$

Where: $D = I$ is a unitary input for the adjoint topology. The matrix P is computed recursively using the following equation:

$$P(k) = \alpha^{-1}[P(k-1) - P(k-1)\cdot \Omega_{W(k)}\cdot S^{-1}_{W(k)}\cdots \cdot \Omega^*_{W(k)}\cdot P(k-1)] \tag{15}$$

Where the matrices Ω_W and S_W are given by:

$$\Omega^*_{W(k)} = \begin{bmatrix} DY^*[W(k)] \\ 0 \quad \cdots \quad 1 \quad \cdots \quad 0 \end{bmatrix} \tag{16}$$

$$S_{W(k)} = \alpha\Lambda + \Omega^*_{W(k)}\cdot P(k-1)\cdot \Omega_{W(k)} \tag{17}$$

$$\Lambda^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & \rho \end{bmatrix} \tag{18}$$

The matrices P and S_w have dimensions $(N_w \times N_w)$ and (2×2) respectively, where N_w is the number of weights in the vector W. The matrix Ω^*_w has dimension $(2\times N_w)$, and its second row consists of $(N_w - 1)$ zeroes and a unit element in the i-th position, which is computed by:

$$i = k\cdot \mathrm{mod}(N_W) + 1 \tag{19}$$

The parameters for the algorithm should be restricted as follows:

$$10^{-6} \le \rho \le 10^{-4}, \qquad 0.97 \le \alpha \le 1.00$$
$$10^3 \le P(0) 10^6 \tag{20}$$

The **CVRNN** topology presented in (1)-(5) with the **CVLM** learning algorithm (9)-(20) can be used in identification (see [6]) and control problems applied to nonlinear, oscillatory dynamical systems.

III. Complex-Valued Adaptive Neural Control

We illustrate the application of **CVRNN** in the design of neural control for a multiple-input multiple-output (**MIMO**) nonlinear plant by proposing a neural adaptive controller scheme with a feedforward term, state feedback term, and an integral term, in two different control schemes.

We further make a comparison between the **CVLM** learning algorithm developed in this work and the **CVBP** learning algorithm presented in [7], [8], using the total **MSE** of the control error as performance index. The plant model is given in continuous time so, in order to use discrete-time **CVRNN** in a control scheme, the input and output signals of the plant are discretized with a sampling time τ.

A. Direct Feedforward Adaptive Neural Control with State Feedback.

For the first adaptive neural controller, we use the control scheme represented in the diagram shown in Fig 3.

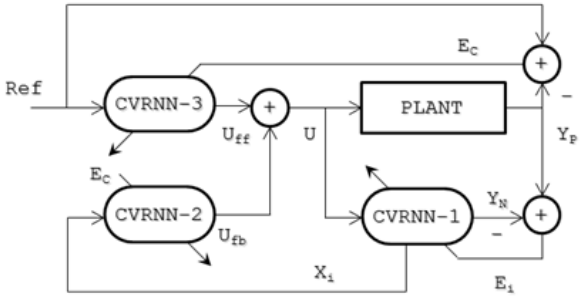


Fig.3 Diagram for the feedforward adaptive neural controller with state feedback

In this control scheme we use three different **CVRNN**, each with a specific task:

- *CVRNN-1*: Is used as a plant model identifier. It has the control signal U that is fed to the plant as its input and produces an identification output Y_N, which is compared to the plant output Y_P to produce the identification error signal used for the training of its own weight parameters. The identification error is described by the following equation:

$$E_i(k) = Y_P(k) - Y_N(k) \tag{21}$$

- *CVRNN-2*: This neural network is fed the internal state vector X_i of the identification network **CVRNN-1** to produce a state feedback control signal U_{fb}. This network is trained with the control error signal E_C, which is produced by comparing the reference signal R to the output of the plant, described by the following equation:

$$E_C(k) = R(k) - Y_P(k) \tag{22}$$

- *CVRNN-3*: This neural network is trained with the control error signal so it converges to the inverse model of the plant. The network is fed the reference signal R to produce a feedforward control signal U_{ff}.

Except for the dimensions of each of the **CVRNN**, the three present the same topology. The control signal is the sum of the feedforward and state feedback control signals, and is described by the following equation:

$$U_1(k) = U_{ff}(k) + \mathrm{U}_{fb}(k) \tag{23}$$

B. Direct Feedforward Adaptive Neural Control with Integral Term and State Feedback

This control scheme is described by the diagram shown in Fig 4.

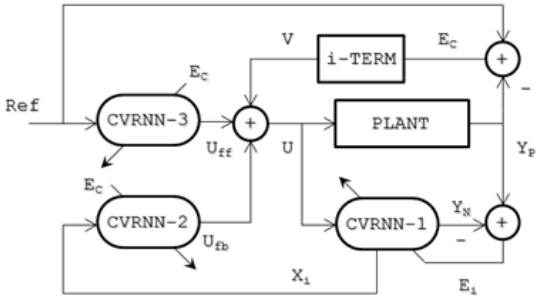


Fig.4 Diagram for the feedforward adaptive neural controller with integral term and state feedback

We used three **CVRNN** for this control scheme: one as a plant model identifier, one as a state feedback controller, and one as a direct feedforward controller. We add an integral term U to the control signal to eliminate the steady-state error present in the plant output. This integral term is given by the following equation:

$$V(k+1) = V(k) + \tau \cdot K_i \cdot E_C(k) \tag{24}$$

Where: K_i is the integral-action gain and τ is the sampling time. The control signal for this scheme is the sum of three terms and is described by the following equation:

$$U_2(k) = U_{ff}(k) + \mathrm{U}_{fb}(k) + \mathrm{V(k)} \tag{25}$$

This control scheme forces the plant to follow the reference signal with zero steady-state error. The stability of the whole system, for both control schemes, is assured by the restriction in the weight parameters of the matrix J of the **CVRNN** and the boundedness of its activation functions.

C. Nonlinear Plant Model Description

The nonlinear plant proposed for this work is a flexible-joint robotic arm with two degrees of freedom (**DOF**) as

shown in Fig 5. Each joint consists of an actuator connected to a load through a torsional spring representing its flexibility. We make the following assumptions to get a simplified mathematical model of the plant:

- The rotor inertia of each joint is symmetric about its rotation axis so that the gravitational terms of the model, the rotor velocity, and the center of mass of each link are independent of the rotor position.
- The kinetic energy acting on the rotor is due to its own rotation, and the motion of the rotor is a pure rotation with respect to the inertial frame.

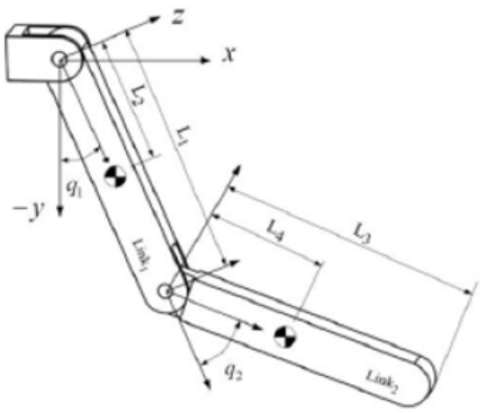


Fig.5 Two-DOF robotic arm with flexible joints

The flexibility at the joint is caused by a harmonic drive, which is a type of gear mechanism with high torque transmission, low backlash and compact size (Fig 6).

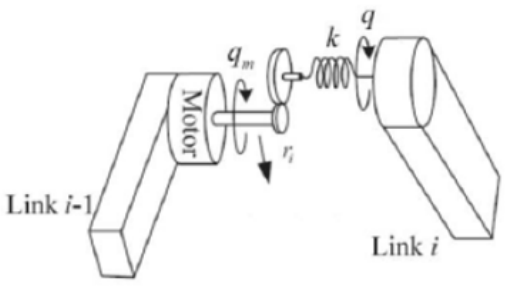


Fig.6 Idealized model of the i-th flexible joint

Under these assumptions, we obtain the equations of motion of our plant, which are given by:

$$D(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) + K(q - q_M) = 0 \qquad (26)$$

$$J_M\ddot{q}_M - K(q - q_M) = u \qquad (27)$$

Where: $q, q_M \in \mathbb{R}^n$ are the links and motor shafts angular displacement respectively; $D(q): \mathbb{R} \to \mathbb{R}^{n\times n}$ is a positive definite inertia matrix, $C(q,\dot{q}): \mathbb{R}^{n\times n} \to \mathbb{R}^n$ is a matrix containing all the Coriolis and centrifugal force terms, $G(q): \mathbb{R}^n \to \mathbb{R}^n$ is the gravitational force vector, $K \in \mathbb{R}^{n\times n}$ is a diagonal, positive definite spring constant matrix of the flexible joints; $J_M \in \mathbb{R}^{n\times n}$ is the inertia matrix of the motors of each joint, and $u \in \mathbb{R}^n$ is the input torque vector.

Defining an extended vector $q_f := \begin{bmatrix} q^T & q_M^T \end{bmatrix}^t$, we rewrite equations (26)-(27) in the following matrix form:

$$D_f(q)\ddot{q}_f + C_f(q,\dot{q})\dot{q}_f + G_f(q) + K_f q_f = u_f \qquad (28)$$

Where:

$$D_f(q) = \begin{bmatrix} D(q) & 0 \\ 0 & J_M \end{bmatrix}, \quad C_f(q,\dot{q}) = \begin{bmatrix} C(q,\dot{q}) & 0 \\ 0 & 0 \end{bmatrix}$$

$$G_f(q) = \begin{bmatrix} G(q) \\ 0 \end{bmatrix}, \quad K_f = \begin{bmatrix} K & -K \\ -K & K \end{bmatrix}, \quad u_f = \begin{bmatrix} 0 \\ I \end{bmatrix} u$$

The plant is a sub-actuated, oscillatory, nonlinear system described by four second order ordinary differential equations. More properties of this model are mentioned in [11].

D. Simulation and Results.

We test the **CVLM** learning algorithm applied to the two control schemes proposed in this work, for a nonlinear plant with a simulation using MATLAB. Next, we make a comparison between the **CVBP** and the **CVLM** learning algorithms. As a comparison measure, we use the total **MSE** of the control error signal. This section describes the simulation settings used and the results obtained.

The reference signals used is given by the following vector:

$$R(k) = \begin{bmatrix} 0.25 \\ -0.45 \end{bmatrix} \text{rad} \qquad (29)$$

For every simulation, a total time of $T = 100s$, a sampling time of $\tau = 0.01$, and activation function $\Gamma[\cdot] = \Phi[\cdot] = \tanh(\cdot)$ were used. For the simulations using the **CVBP** learning algorithm, the learning rate parameter $\eta = 0.1$ and the momentum parameter $\alpha = 0.04$ were used for each one of the neural networks. For the simulations involving the **CVLM** learning algorithm, the parameters $\alpha = 0.98$, $\rho = 1\times10^{-4}$, and $P_J(0) = P_B(0) = P_C(0) = 1\times10^{5}$ were used for each one of the neural networks.

For the first control scheme, the dimensions used for the **CVRNN-1** and **CVRNN-3** were $n_{1,3} = 3, m_{1,3} = 2, L_{1,3} = 2$, for the **CVRNN-2** were $n_2 = 3, m_2 = 3, L_2 = 2$ with initial conditions of the internal state vector $X(0) = \begin{bmatrix} 0.1 & 0.1 & 0.1 \end{bmatrix}^t$, and random initial conditions for each weight parameter in the interval $[-0.1, 0.1]$ for each neural network.

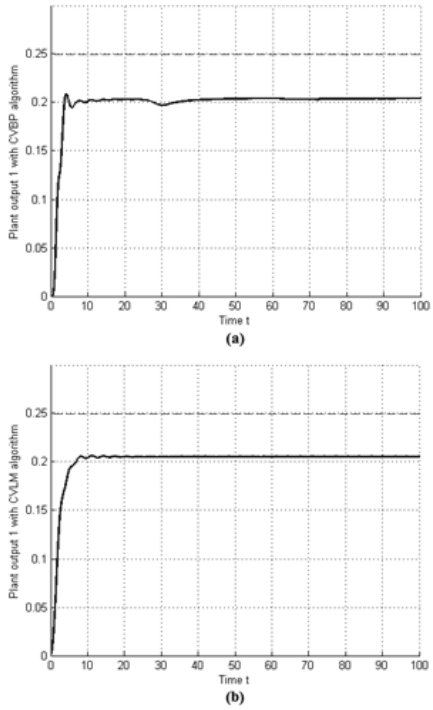


Fig.7 Output 1 of the plant without integral term, for (a) CVBP algorithm and (b) CVLM algorithm.

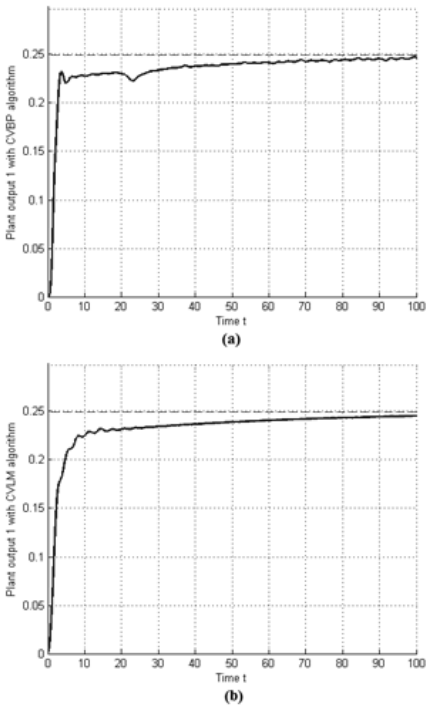


Fig.9 Output 1 of the plant with integral term, for (a) CVBP algorithm and (b) CVLM algorithm.

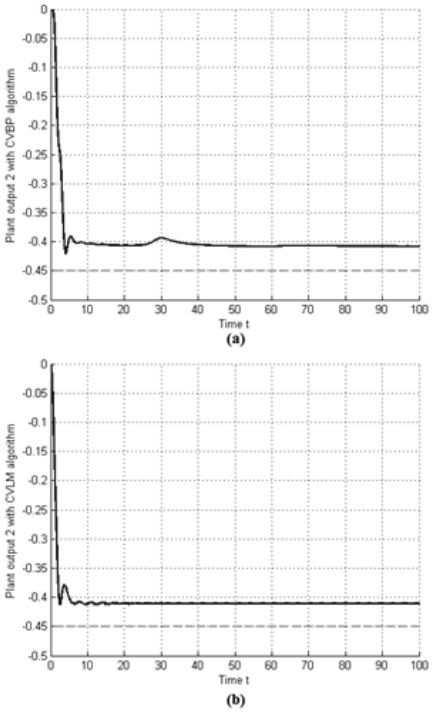


Fig.8 Output 2 of the plant without integral term, for (a) CVBP algorithm and (b) CVLM algorithm.

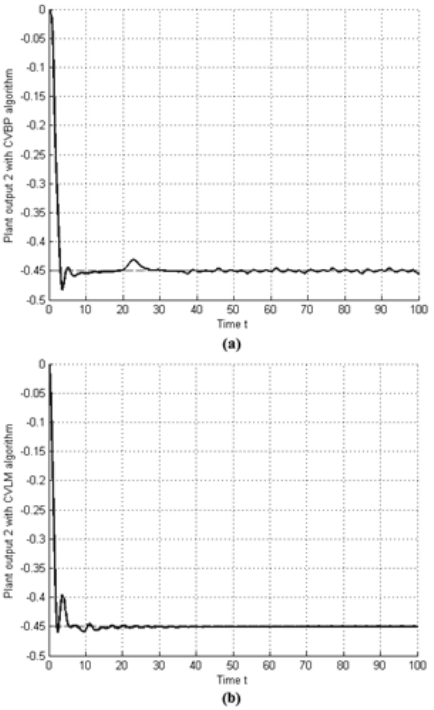


Fig.10 Output 2 of the plant with integral term, for (a) CVBP algorithm and (b) CVLM algorithm.

The outputs of both **DOF** of the plant for this control scheme are shown in Fig7 for the **CVBP** learning algorithm and Fig 8 for the **CVLM** learning algorithm.

For the second control scheme, the dimensions used for the **CVRNN** were the same as the ones used in the first control scheme, with initial conditions of the internal state vector $X(0) = [0.1 \quad 0.1 \quad 0.1]^t$, random initial conditions for each weight parameter in the interval $[-0.1, 0.1]$ and integral-action gain $K_i = 5$. The outputs of both **DOF** of the plant for this control scheme are shown in Fig 9 for the **CVBP** learning algorithm and Fig 10 for the **CVLM** learning algorithm.

From these graphs we observe that the adaptive neural control schemes are effective for taking the plant outputs to the desired positions when using both learning algorithms for the training of the **CVRNN**.

As it was expected, the control scheme with no integral term presents steady-state error, which is eliminated for the control scheme with integral term. An overshoot can be observed in both control schemes, being smaller for the controller trained with the **CVLM** algorithm.

The controllers trained with the **CVLM** learning algorithm have a better time-response than its **CVBP** counterparts. They also present more oscillations throughout the transient state but, unlike the **CVBP** trained controllers, the oscillations attenuate in the steady state.

The final **MSE** for each of the control schemes trained with both learning algorithms are presented on Table I.

Table I. Final MSE for the CVBP and CVLM learning algorithms applied to the three control schemes

	CVBP	**CVLM**
Scheme 1	6.18×10^{-3}	4.67×10^{-3}
Scheme 2	3.44×10^{-3}	1.64×10^{-3}

From the final **MSE** we observe that both control schemes trained with the **CVLM** learning algorithm have a better performance compared to their training with the **CVBP** learning algorithm. Furthermore we observe that the control scheme where the integral term is added has a better performance than when it is not added.

IV. Conclusions

In this article we proposed three adaptive neural control schemes in the complex domain, applied to a nonlinear, sub-actuated mechanical plant.

A second order Recursive Levenberg-Marquardt learning algorithm was obtained to train the Complex-Valued Recurrent Neural Networks that were used in the control schemes, using diagrammatic methods to derive the gradient terms, and calculating in a recursive way the covariance matrix, needed for the update equations of its weight parameters.

The control schemes shown through this article present good behavior, and the comparative results obtained while using the Complex-Valued Back-Propagation algorithm and the Levenberg-Marquardt algorithm show that the later presents a much better performance than the former.

Acknowledgment

Master Degree student Edmundo Pérez Reynaud would like to thank CONACyT, Mexico, for the student grant received during his studies at the Department of Automatic Control, CINVESTAV-IPN, Mexico (grant number: 364197, holder number: 295620,).

References

[1] A. Hirose. "Complex-Valued Neural Networks." 2nd ed., S.i.C. Intelligence, Springer-Verlag, vol. 400, 2012.

[2] A. Minin, Y. Chistyakov, E. Kholodova, H.G. Zimmermann, A. Knoll. "Complex-Valued Open Recurrent Neural Network for Power Transformer Modeling." *International Journal of Applied Mathematics and Informatics*, vol. 6, no. 1, pp. 41-48, 2012.

[3] L. Ferariu. "Nonlinear System Identification Based on Evolutionary Dynamic Neural Networks." *Proc. Of European Control Conference*, Cambridge, UK, paper no. 432, 2003.

[4] H. Suzuki, T. Yasuno, A. Kuwahara, S. Urushihara, E. Yasuno. "Control characteristics of cooperative conveyance system for multiple mobile robots using complex-valued neural networks." *Proceedings of SICE Annual Conference 2010*, Taipei, Taiwan, pp. 3581-3585, 2010.

[5] T. Fujiwara, Y. Maeda, H. Ito. "Learning of inverse-kinematics for robot using high dimensional neural networks." *Proceedings of SICE Annual Conference 2013*, Nagoya, Japan, pp. 2743-2748, 2013.

[6] E. P. Reynaud, I. S. Baruch. "Identification of dynamical systems using Levenberg-Marquardt learning algorithm for recurrent complex-valued neural networks." *19th International Conference on Circuits, Systems, Communications and Computers*, Zakynthos Island, Greece, 2015. (accepted for presentation).

[7] V. M. Arellano-Quintana, I. S. Baruch. "Identification of Dynamical Systems Using Recurrent Complex-Valued Neural Networks." In *Proc. Of the 18th International Conference on Circuits, Systems, Communications and Computers*, Santorini, Greece, pp. 74-79, 2014.

[8] I. S. Baruch, V. M. Arellano-Quintana. "Identification and Control of Oscillatory Dynamical Systems Using Recurrent Complex-Valued Neural Networks." In *Proc. Of the 18th International Conference on Circuits, Systems, Communications and Computers*, Santorini, Greece, pp. 534-539, 2014.

[9] E. A. Wan, F. Beaufays. "Diagrammatic Methods for Deriving and Relating Temporal Neural Network Algorithms." *Neural Computation*, pp. 182-201, 1996.

[10] I. S. Baruch, C.R. Mariaca-Gaspar. "A Levenberg-Marquardt Learning Applied for Recurrent Neural Identification and Control of Wastewater Treatment Bioprocess." *International Journal of Intelligent Systems*, no. 24, pp. 1094-1114, 2009.

[11] S. S. Ge, T. H. Lee, C. J. Harris, "Adaptive Neural Network Control of Robotic Manipulators." C. J. Harris, Ed. UK: World Scientific Publishing Co. Pte. Ltd., 1998, vol. 19.