UNIVERSIDAD AUTÓNOMA DE QUERÉTARO

FACULTAD DE INGENIERÍA

IMPLEMENTACIÓN DE UN ALGORITMO DE CONTROL NEURONAL RECURRENTE EN FPGA Y DSP PARA SISTEMAS DINÁMICOS

TESIS

QUE PARA OBTENER EL GRADO DE:

Maestro en Ciencias con línea terminal en Instrumentación y Control Automático

P R E S E N T A:

Ing. Carlos Alberto Silva Rodríguez

C.U. SANTIAGO DE QUERÉTARO, QRO. FEBRERO 2009

Universidad Autónoma de Querétaro
Facultad de Ingeniería
Maestría en Ciencias (Línea Terminal
Instrumentación y Control Automático)

"Implementación de un algoritmo de control neuronal recurrente en FPGA y DSP para sistemas dinámicos"

TESIS

Que como parte de los requisitos para obtener el grado de:
Maestro en Ciencias

Presenta:
Ing. Carlos Alberto Silva Rodríguez

Dirigido por:
M en C. Alfonso Noriega Ponce

SINODALES

M en C. Alfonso Noriega Ponce
Presidente — Firma

Dr. Rodrigo Castañeda Miranda
Secretario — RUBRICA / Firma

Dr. Yu Tang Xu
Vocal — Firma

Dr. Víctor Manuel Hernández Guzmán
Suplente — Firma

Dr. Roberto Salas Zúñiga
Suplente — Firma

Dr. Gilberto Herrera Ruiz
Director de la Facultad de Ingeniería

Dr. Luis Gerardo Hernández Sandoval
Director de Investigación y Posgrado.

Centro Universitario
Querétaro, Qro.
Febrero de 2009
México

RESUMEN

En el presente trabajo se presenta el desarrollo e implantación de un algoritmo de control inteligente basado en redes neuronales diagonales recurrentes (RNDR). La arquitectura de la RNDR es un modelo simplificado de una red neuronal recurrente completamente conectada en una capa oculta, dicha capa está compuesta por neuronas que se retroalimentan a sí mismas. Dos RNDR se utilizan en el sistema de control: una es la encargada de la identificación llamada neuroidentificador diagonal recurrente (NIDR) y la otra que realiza el papel de controlador, llamada neurocontrolador diagonal recurrente (NCDR). Se desarrolló un algoritmo de retropropagación dinámico (RPD) para entrenar tanto al NIDR como al NCDR y así ajustar sus pesos. Debido a la recurrencia de las RNDR el algoritmo tiene mejores posibilidades que las redes estáticas para controlar sistemas con dinámica difícil. Así mismo se muestran los resultados obtenidos tanto en simulación como en implementación práctica del controlador basado en la RNDR en un FPGA (Field Programmable Gate Array) y DSP (Digital Signal Processing).

Palabras clave: (Redes neuronales, retropropagación, control, identificación, tiempo real, FPGA, DSP).

SUMMARY

This research presents the development and implantation of an intelligent control algorithm, based on diagonal recurrent neural networks (RNDR). The architecture of the RNDR is a simplified model of the fully connected recurrent neural network with one hidden layer, which is comprised of retro-feeding neurons. Two RNDR's are used in a control system, the first one is an identifier called diagonal recurrent neuroidentifier (NIDR) and the second one, called diagonal recurrent neurocontroller (NCDR), performs the role of controller. We developed an algorithm for dynamic backpropagation (DPR) to train both the NIDR as well as the NCDR and to adjust its weights. Also, the results both in simulation and in practical implementation of the controller based on the RNDR in a FPGA (Field Programmable Gate Array) and DSP (Digital Signal Processing) are shown.

Keywords: (Neural networks, backpropagation, control, identification, real-time, FPGA, DSP).

A mis Padres,

Blanca Patricia y Fidel

AGRADECIMIENTOS

Agradezco a mi asesor M.C. Alfonso Noriega Ponce por su incesante apoyo y su excelente dirección, por guiar paso a paso la elaboración de la tesis.

Al Dr. Yu Tang Xu por las cátedras impartidas en el área de control no lineal y sus valiosos comentarios en las implementaciones de algoritmos de control.

Al Dr. Rodrigo Castañeda Miranda por facilitarme la tarjeta Virtex 4 de xilinx.

Al Dr. Víctor Manuel Hernández Guzmán y al Dr. Roberto Salas Zúñiga por la revisión de este documento.

Al Dr. René de Jesús Troncoso por las clases impartidas en electrónica avanzada y las pláticas que tuvimos acerca de las tecnologías DSP y FPGA.

A CONACYT por la beca que me brindaron para la realización de esta tesis.

Finalmente, quiero agradecer a todos los maestros que intervinieron en mi formación académica.

Gracias

ÍNDICE GENERAL

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE CUADROS

1) INTRODUCCIÓN Y OBJETIVOS

Introducción

Los sistemas de control como tal surgen a nivel industrial para resolver diversos problemas, entre los cuales se pueden tomar en cuenta: la protección del medio ambiente, la disminución de las horas hombre-máquina, la prevención de riesgos al personal, mantener la calidad del producto y niveles de producción a un mínimo costo. En los años cincuenta, nacen los sistemas de control, conocidos como control clásico o convencional, actualmente el control convencional domina el mercado del control de procesos industriales y muchas de las empresas siguen teniendo esos sistemas de control en sus procesos de producción.

Estamos viviendo en un mundo en donde la industria está cada vez mas automatizada e integrada, por lo que los nuevos productos industriales deben competir tanto con los que se fabrican en países más industrializados como con aquellos producidos por una mano de obra muy barata. Es por ello que los procesos de fabricación deben tender cada vez más a asegurar una calidad rigurosa y uniforme, al mismo tiempo que una alta productividad. En muchos procesos industriales la función de control es realizada por un operario (ser humano), este operario es el que decide cuando y como manipular las variables de modo tal que se obtenga una cadena productiva continua y eficiente, éste tipo de control utilizado de forma manual presenta muchas fallas, lo que trae como consecuencia que cambien los parámetros de la variable a controlar y a su vez esto implica pérdidas de dinero y tiempo. Esta situación ha originado que los operadores estén verificando el funcionamiento de sus procesos periódicamente, lo cual acarrea la baja calidad en la producción y pérdidas económicas.

Otro de los problemas principales a nivel industrial en los sistemas de producción es la confiabilidad y robustez de los controladores comerciales implementados para diversas variables como por ejemplo sistemas de temperatura, presión, ph, posicionamiento, etc. Donde es muy importante mantener tal variable en un rango adecuado de valores para obtener un producto de calidad y que cumpla con los estándares establecidos, así mismo que se

aproveche el máximo la materia prima minimizando el scrap, muchos de esos controladores adquiridos a empresas extranjeras sus costos son muy elevados y sólo las medianas y grandes empresas deciden implementarlos.

Debido a estos problemas, la industria ha necesitado de la utilización de nuevos y más complejos procesos, que muchas veces el operario no puede controlar debido a la velocidad y exactitud requerida, además las condiciones del espacio donde se lleva a cabo la tarea no son las más adecuadas para el desempeño del ser humano.

Frente a este panorama, surgen los sistemas de control basados en algoritmos inteligentes como son las redes neuronales recurrentes implementadas en FPGA (Arreglo de Compuertas de Campo Programable) y DSP (Procesadores Digitales de Señales) como una solución que va a permitir llevar a la producción a estándares de calidad mucho mejores. Lo que implica el constante aumento de los niveles de producción de la maquinaria instalada, el mejoramiento de la calidad del producto final, la disminución de los costos de producción, y la seguridad tanto para el personal como para los equipos, además de que el costo de este controlador es muy bajo en comparación con los actuales que ofrecen empresas dedicadas al control.

Una de las ventajas de poder implementar algoritmos inteligentes como son las redes neuronales recurrentes en tecnologías de vanguardia como FPGA’s y DSP’s es que permiten mantener el control de las variables de un proceso con un mínimo de conocimiento y muy pocas configuraciones (Pellerin y Thibault, 2005), otra ventaja es el rápido aprendizaje y adaptabilidad que tienen cuando se presentan cambios o perturbaciones directamente en la variable, también el controlador a partir de estos algoritmos de aprendizaje permite auto ajustarse de manera que el operador no tiene que preocuparse por estar cambiando o configurando el controlador si no que éste lo hace automáticamente.

Otro aspecto muy importante para el desarrollo e implementación de este nuevo controlador es que va a romper el paradigma de la metodología clásica tanto de investigación como de la implementación en estas tecnologías, es decir,

se utilizarán herramientas de vanguardia en un lenguaje de alto nivel para la descripción del hardware y se emplearán algoritmos de redes neuronales recurrentes que mejoran los sistemas de control frente a las clásicas redes neuronales. Sobre todo este trabajo de investigación con lleva un estudio muy amplio sobre tecnologías actuales directamente aplicadas a la problemáticas de la industria mexicana, donde el impacto a nivel tecnológico es muy alto y es una solución fiable a corto plazo.

En forma general lo que se pretende en este trabajo de tesis es crear conocimiento en dos vertientes:

a) Crear una base metodología que permita implementar algoritmos de control inteligentes en tecnologías de vanguardia de manera rápida, sencilla y confiable además de que puedan emplearse en innumerables procesos que abarcan desde la solución de sistemas mecatrónicos hasta sistemas de tipo climatológicos muy demandantes en la región.

b) Los resultados que se obtendrán en este trabajo podrán ser aplicados de manera inmediata en procesos industriales que requieran sistemas de control robustos, económicos y accesibles. Además se pretende dar origen al dominio de una tecnología propia que permita contribuir a mejorar los sistemas antes mencionados desde otro enfoque diferente en el área de control.

Objetivos

Los objetivos que se plantearon en este trabajo de investigación fueron los siguientes:

- Con el presente trabajo de investigación se pretende, como primer punto generar una independencia tecnológica en el área relacionada con los controles inteligentes aplicados en el área de control de sistemas dinámicos.

- Como segundo punto se desea avanzar hacia la investigación y construcción de un controlador red neuronal recurrente de tipo industrial que sea competitivo en tecnología con relación a los productos comerciales existentes, pero además que sean generados a bajo costo y que permitan satisfacer las demandas del mercado actual.

- La utilización de componentes digitales económicos y de gran desempeño para optimizar la velocidad de procesamiento de información, en este caso los FPGA y DSP. También desarrollar toda una metodología para la implementación de algoritmos inteligentes que se han venido desarrollando en los últimos años, que permitiría sentar las bases para futuros proyectos de implementación en estas tecnologías utilizando herramientas de descripción y programación de alto nivel.

- Y por último el diseño, implementación y análisis de un sistema con el fin de comprobar su fiabilidad y efectividad.

Esta tesis está organizada en la forma siguiente: En el capítulo 2) se presenta un panorama histórico sobre los trabajos en los cuales motivaron este trabajo de tesis. En el capítulo 3) se recopila los conceptos básicos sobre las redes neuronales y su clasificación. En el capítulo 4) se presentan los conceptos generales sobre las redes neuronales artificiales estáticas y dinámicas haciendo hincapié en sus características de cada una de ellas. En el capítulo 5) Se desarrolla el algoritmo de control recurrente basado en el esquema de control propuesto por Chao Chee y Lee (1995). El capítulo 6) La Implementación embebida del algoritmo de control neuronal recurrente en arquitecturas FPGA y DSP mostrando la metodología y resultados obtenidos. Y por último el capitulo 7) donde se muestran los resultados y conclusiones obtenidos durante el desarrollo, simulación e implementación del algoritmo de control neuronal recurrente.

2) ANTECEDENTES HISTÓRICOS

El campo de la inteligencia artificial se refiere habitualmente de forma más sencilla como redes de neuronas o redes neuronales, las redes neuronales artificiales (denominadas habitualmente como RNA o en inglés como: Artificial Neural Networks ANN) son un paradigma de aprendizaje y procesamiento automático inspirado en la forma en que funciona el sistema nervioso de los animales (Krosse y Smagt, 1996).

Los modelos de las RNA han sido estudiados por varios años con la esperanza de alcanzar resultados similares a los de los humanos en áreas como el reconocimiento de imágenes, voz, predicción, codificación, control y optimización; pero no es sino hasta fechas recientes que se tienen resultados prácticos considerables. Estos modelos se componen por varios elementos que realizan operaciones matemáticas no lineales, operan en paralelo y están arreglados en estructuras que pretenden simular a las redes neuronales de los seres vivos. Los elementos que realizan las operaciones matemáticas (nodos) se conectan entre sí mediante coeficientes de peso que en general son ajustados durante su operación para mejorar su desempeño y de esta forma contar con capacidad de aprender de experiencias, generalizar de casos anteriores a nuevos casos, abstraer características esenciales a partir de entradas que representan información irrelevante, tolerancia a fallos, etc.

Este trabajo titulado redes neuronales diagonales recurrentes para sistemas dinámicos de control (Chao Chee y Lee, 1995), es a consideración propia el más importante dentro de la mayoría de la bibliografía conocida, porque posee las bases tanto matemáticas como metodológicas para poder llevar el esquema de control a la implementación, a su vez que se puede modificar para un tratamiento multidisciplinario del algoritmo. En su trabajo proponen un esquema de control que se basa en 2 redes neuronales recurrentes una de identificación y la otra de control y el algoritmo de entrenamiento se basa en el de retropropagación dinámico para ajuste de pesos. El paradigma propuesto de su

red neuronal diagonal recurrente esta aplicado a los problemas para el control de plantas tanto lineales como no lineales.

Hussin et al., (2004) trabajaron sobre un estudio comparativo de los controladores con red neuronal para sistemas dinámicos no lineales, comentan que los controladores basados en redes neuronales artificiales (RNA) están demostrando un alto potencial en sistemas dinámicos no lineales para controles no convencionales. Hacen la comparativa de estudio de los neurocontroladores basados en diferentes topologías de las RNA, concretamente las redes neuronales retroalimentadas (RNR) usando un algoritmo de adaptación de pesos generalizado y las redes neuronales diagonales recurrentes (RNDR) que usan un algoritmo de retro-propagación dinámico generalizado. De este modo ambos tipos de redes pueden usar la metodología de entrenamiento en línea o fuera de línea. Además también su estudio abarca la controlabilidad de los sistemas usando controladores retroalimentados combinados con identificación de sistemas dinámicos inversos. Los resultados arrojados por las simulaciones son usados para verificar el efecto de cambiar la topología de las RNA y la forma de entrenamiento de las mismas para el rendimiento de sistemas dinámicos no lineales.

El modelo adaptativo de una red neuronal diagonal recurrente trabajado por Yu y Chang (2005), es un desarrollo para el modelado dinámico de sistemas no lineales. Este modelo de adaptación es conseguido gracias al filtro extendido kalman (EKF). Este algoritmo recursivo es propuesto para calcular una matriz jacobiana en el modelo de adaptación con el propósito de que sea simple el desarrollo y que converge más rápido. La eficacia del desarrollo del modelo adaptativo es demostrado a través de una aplicación simulada en un tanque de reactor para sintéticos. Los resultados mostrados dan fe de que el modelo converge a la dinámica del proceso muy rápidamente después de que se aplica una perturbación continua. Este tipo de modelo de control puede ser usado como un modelo adaptativo y predictivo o un modelo de control interno para sistemas con constantes de tiempo variables o controles tolerantes a fallos de sistemas no lineales.

Las redes neuronales recurrentes tienen propiedades interesantes y pueden manejar procesamiento dinámico de información a diferencia de las redes neuronales clásicas retroalimentadas (Maeda y Wakamura, 2005), sin embargo son difíciles de usar porque no hay ningún esquema de aprendizaje conveniente. En su trabajo plantean un esquema para el aprendizaje recursivo de red neuronal recurrente usando un método de perturbación simultáneo, donde explican en detalle el procedimiento y se comenta que no es un aprendizaje clásico de las neuronas si no que se basa en un aprendizaje análogo y oscilatorio. Hacen implementación práctica de de una red neuronal hopfield usando un FPGA y muestran detalles de esta implementación, además muestran resultados gráficos y comentan que la solución a la implementación es viable.

Ren y Chen (2006), propone una red neuronal recurrente para el control de velocidad de un motor de inducción, este control de velocidad consiste de una red neuronal recurrente identificadora (RNRI) y una red neuronal recurrente controladora (RNRC). La RNRI es usada para proveer información adaptativa en tiempo real de la dinámica desconocida del motor. La RNRC es usada para producir un control forzado hacia la velocidad del motor el cual produce una respuesta exacta de la referencia propuesta al sistema. El algoritmo de retroalimentación que es usado para el aprendizaje de la red ajusta automáticamente los pesos de la RNRI y RNRC respectivamente para mantener el rendimiento del sistema. El esquema de control propuesto puede calcular los parámetros de planta rápidamente de forma que la velocidad del motor puede seguir el modelo de referencia con gran exactitud. Las simulaciones en la computadora y los resultados experimentales demuestran que el esquema de control propuesto es un esquema de control de velocidad para el motor muy robusto.

En su trabajo de Khaled et al., (2007), emplearon la metodología de las redes neuronales recurrentes para obtener un control adaptativo aplicado directamente para manejar la velocidad de un motor de CD no lineal. Estas redes fueron usadas para obtener una estimación del modelo del motor el cual poseía una alta fricción, primero se entreno la red fuera de línea para aprender la dinámica inversa del modelo del motor usando una forma modificada del algoritmo

de kalman que es un filtro de desacople extendido. Muestran que la combinación de la estructura de la red neuronal recurrente combinado con el modelo de control inverso es un efectivo control adaptativo para controlar sistemas con motores de CD. Ellos validaron el modelo a través de la experimentación con una computadora personal sobre un motor de CD y los resultados obtenidos confirman que rechazan las perturbaciones y mantienen el rendimiento del sistema dentro del rango de especificaciones planteadas.

Después de presentar algunos de los trabajos de mayor relevancia a consideración propia se llego a la siguiente serie de conclusiones:

a) Todos los artículos a excepción de uno “Reglas de aprendizaje de perturbación simultánea para redes neuronales recurrentes y su implementación en FPGA de Maeda y Wakamura (2005)” no se plantea la posibilidad de llevar a cabo la implementación en tecnologías tipo FPGA y DSP. Lo que permite contribuir de manera relevante en este campo dejando una base de conocimientos muy importante.

b) Se plantean muchos esquemas de aprendizaje para la red neuronal recurrente y mencionan sus ventajas frente a las clásicas redes neuronales retroalimentadas así como sus posibles aplicaciones para diferentes áreas de la ciencia. Esto permite tener un amplio panorama sobre las opciones de implementación más convenientes para la mejora y/o solución de sistemas que se han venido trabajando en los últimos años en la región.

c) La diversidad de aplicaciones hacia donde enfocan los estudios de los algoritmos de red neuronal recurrente (control de posición y velocidad de motores, invernaderos, control de temperatura industrial, etc.) refleja el impacto que tiene el uso de estos algoritmos inteligentes en los sistemas de control.

3) REDES NEURONALES ARTIFICIALES

En este capítulo se describen los fundamentos de las redes neuronales. Comenzamos dando una breve introducción histórica de su nacimiento y las características más importantes que han hecho de las mismas una herramienta fundamental en la resolución de un gran número de problemas. Después presentamos la clasificación de redes neuronales en función de su arquitectura y su mecanismo de aprendizaje, haciendo especial hincapié en los aspectos más relevantes de cada una de ellas.

3.1) Introducción

En 1949 D.O. Hebb propuso una ley de aprendizaje que puede considerarse como el punto de partida de los algoritmos de entrenamiento de las redes neuronales, mientras que la primera estructura de red neuronal fue propuesta por Frank Rosenblatt en 1958. Desde entonces el interés suscitado por las redes neuronales ha crecido enormemente, proponiéndose nuevos modelos que han venido a superar las limitaciones de las redes anteriores, dándoles mayores capacidades.

Las redes neuronales tienen su origen en la observación de los sistemas neuronales de los organismos vivientes, sin embargo, éstas no son un fiel reflejo de dichas redes biológicas. Como características más relevantes de las redes neuronales artificiales citamos las siguientes:

1. La capacidad de aprendizaje. La naturaleza no lineal de las mismas les permite exhibir comportamientos verdaderamente complejos. Esta característica junto con algoritmos que extraen un mayor rendimiento de tales sistemas les confiere esa capacidad.

2. La posibilidad de procesamiento paralelo, aspecto que aprovechado en la implementación lleva a aumentar considerablemente la velocidad de procesamiento.

3. La significativa tolerancia a fallos, dado que un fallo en unas pocas conexiones locales, rara vez contribuye significativamente al funcionamiento global de la red. Esta propiedad las hace convenientes en ciertas aplicaciones, donde, una avería puede constituir un serio peligro.

Estas características, entre otras, hacen de las redes neuronales una herramienta fundamental en varios campos del conocimiento.

Las redes neuronales generalmente han sido implementadas mediante la programación sobre computadoras digitales, sin embargo ya en nuestros días empiezan a proliferar implementaciones en hardware, ya sea puramente mediante montajes electrónicos o incorporando dispositivos ópticos. Tales ingenios, presentan un paso importante hacia la consecución de mayores velocidades de procesamiento, así como un camino hacia la popularización de las mismas, siendo ésta un área de investigación muy activa.

En nuestros días la ingeniería de control se enfrenta a sistemas cada vez más complejos, a requisitos de diseño más exigentes y al desconocimiento de las plantas y sus entornos, revitalizando este hecho la investigación en el campo de las redes neuronales. En cuanto a la clasificación de las redes neuronales podríamos establecer una división de las mismas en función de su arquitectura y forma de procesar las señales en tres clases.

La primera clase de redes se conoce con el nombre de redes neuronales estáticas (RNE). En esta clase podemos encontrar la red Multicapa de Perceptrones, la red de Funciones de Base Radial, Red Neuronal Probabilística (RNP), etc. Todas ellas tienen como característica común el no poseer memoria, es decir, sólo son capaces de transformar un conjunto de entradas en un conjunto de salidas, de tal manera que una vez establecidos todos los parámetros de la red las salidas únicamente dependen de las entradas. En general la relación deseada de entradas y salidas se determina en este caso externamente mediante alguna forma de ajuste de los parámetros del sistema supervisado. Este tipo de redes se

han empleado con éxito en muchos problemas de clasificación, como funciones lógicas, así como en el campo de la aproximación funcional.

Como segunda clase de redes neuronales debemos nombrar las redes neuronales dinámicas (RND). Estas a diferencia de las anteriores permiten establecer una relación entre salidas y entradas y/o salidas y entradas previas. Esto añade cierta memoria a estas redes. Matemáticamente, esta memoria se traduce en la aparición de ecuaciones diferenciales o ecuaciones en diferencia formando parte del modelo de las mismas. Como ejemplos de este tipo de redes encontramos la red de Hopfield, la red de retardos en el tiempo (Time Delay Neural Network), la red de tiempo discreto (Time Discrete Neural Network), etc.

Las redes neuronales dinámicas se han revelado útiles en problemas de modelización de la dinámica directa e inversa de sistemas complejos, tales como robots, cohetes, naves espaciales, etc., así como en la modelización de circuitos secuenciales y en la conversión de texto a voz.

Por último podemos destacar las denominadas redes auto-organizativas, en las cuales los nodos vecinos dentro de la red neuronal compiten en actividad por medio de las interacciones mutuas laterales, y evolucionan adaptativamente hacia detectores específicos de las diferentes señales de entrada. El aprendizaje en este caso se denomina aprendizaje competitivo, no supervisado o auto organizativo. Estas se han aplicado con éxito en problemas de reconocimiento de patrones, control de procesos e incluso en el procesamiento de información semántica.

En las secciones siguientes comentaremos en más detalle cada una de las clases de redes nombradas en los párrafos anteriores.

3.2) Funcionamiento de una neurona biológica

El cerebro consta de un gran número (aproximadamente 10^{11}) de elementos altamente interconectados (aproximadamente 10^4 conexiones por elemento), llamados neuronas. Estas neuronas tienen tres componentes principales, las dendritas, el cuerpo de la célula o soma, y el axón. Las dendritas, son el árbol receptor de la red, son como fibras nerviosas que cargan de señales eléctricas el cuerpo de la célula. El cuerpo de la célula, realiza la suma de esas señales de entrada. El axón es una fibra larga que lleva la señal desde el cuerpo de la célula hacia otras neuronas. El punto de contacto entre un axón de una célula y una dendrita de otra célula es llamado sinapsis, la longitud de la sinapsis es determinada por la complejidad del proceso químico que estabiliza la función de la red neuronal. Un esquema simplificado de la interconexión de dos neuronas biológicas se observa en la Figura 3.1.

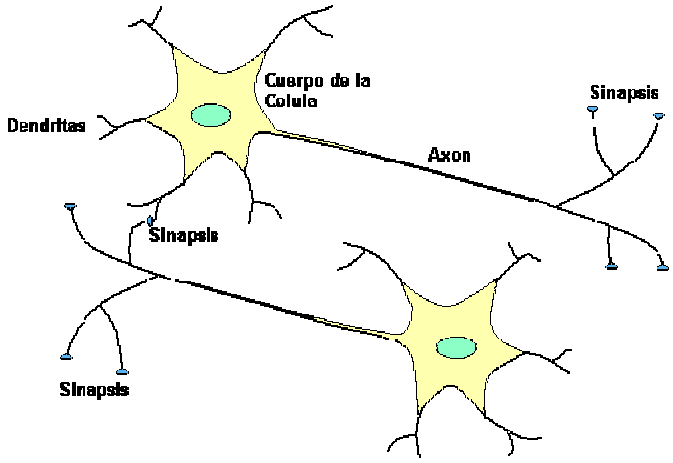


Figura 3.1 Neuronas Biológicas

Todas las neuronas conducen la información de forma similar, esta viaja a lo largo de axones en breves impulsos eléctricos, denominados potenciales de acción; los potenciales de acción que alcanzan una amplitud máxima de unos 100 mV y duran 1 ms, son resultado del desplazamiento a través de la membrana celular de iones de sodio dotados de carga positiva, que pasan desde el fluido extracelular hasta el citoplasma intracelular; la concentración extracelular de sodio supera enormemente la concentración intracelular. La membrana en reposo

mantiene un gradiente de potencial eléctrico de -70mv, el signo negativo se debe a que el citoplasma intracelular está cargado negativamente con respecto al exterior; los iones de sodio no atraviesan con facilidad la membrana en reposo, los estímulos físicos o químicos que reducen el gradiente de potencial, o que despolaricen la membrana, aumentan su permeabilidad al sodio y el flujo de este ion hacia el exterior acentúa la despolarización de la membrana, con lo que la permeabilidad al sodio se incrementa más aún.

Alcanzado un potencial crítico denominado "umbral", la realimentación positiva produce un efecto regenerativo que obliga al potencial de membrana a cambiar de signo. Es decir, el interior de la célula se torna positivo con respecto al exterior, al cabo de 1 ms, la permeabilidad del sodio decae y el potencial de membrana retorna a -70mv, su valor de reposo. Tras cada explosión de actividad iónica, el mecanismo de permeabilidad del sodio se mantiene refractario durante algunos milisegundos; la tasa de generación de potenciales de acción queda así limitada a unos 200 impulsos por segundo, o menos (Ben y Patrick, 1996).

Aunque los axones puedan parecer hilos conductores aislados, no conducen los impulsos eléctricos de igual forma, como hilos eléctricos no serían muy valiosos, pues su resistencia a lo largo del eje es demasiado grande y a resistencia de la membrana demasiado baja; la carga positiva inyectada en el axón durante el potencial de acción queda disipada uno o dos milímetros más adelante, para que la señal recorra varios centímetros es preciso regenerar frecuentemente el potencial de acción a lo largo del camino la necesidad de reforzar repetidamente esta corriente eléctrica limita a unos 100 metros por segundo la velocidad máxima de viaje de los impulsos, tal velocidad es inferior a la millonésima de la velocidad de una señal eléctrica por un hilo de cobre.

Los potenciales de acción, son señales de baja frecuencia conducidas en forma muy lenta, estos no pueden saltar de una célula a otra, la comunicación entre neuronas viene siempre mediada por transmisores químicos que son liberados en las sinapsis. Un ejemplo de comunicación entre neuronas y del proceso químico de la liberación de neurotransmisores se ilustra en la Figura 3.2.

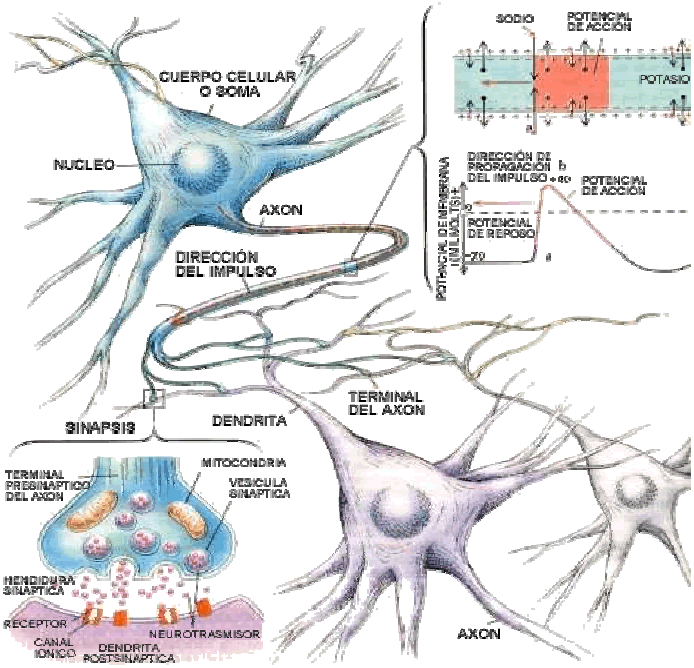


Figura 3.2 Comunicación entre neuronas

3.3) Redes Neuronales Artificiales (RNA)

Resulta irónico pensar que máquinas de cómputo capaces de realizar 100 millones de operaciones en coma flotante por segundo, no sean capaces de entender el significado de las formas visuales o de distinguir entre distintas clases de objetos. Los sistemas de computación secuencial, son exitosos en la resolución de problemas matemáticos o científicos, en la creación, manipulación y mantenimiento de bases de datos, en comunicaciones electrónicas, en el procesamiento de textos, gráficos y auto edición, incluso en funciones de control de electrodomésticos, haciéndolos más eficientes y fáciles de usar, pero definitivamente tienen una gran incapacidad para interpretar el mundo.

Esta dificultad de los sistemas de computo que trabajan bajo la filosofía de los sistemas secuenciales, desarrollados por Von Neuman, ha hecho que un gran número de investigadores centre su atención en el desarrollo de nuevos sistemas de tratamiento de la información, que permitan solucionar problemas cotidianos, tal como lo hace el cerebro humano; este órgano biológico cuenta con varias características deseables para cualquier sistema de procesamiento digital, tales como:

1. Es robusto y tolerante a fallas, diariamente mueren neuronas sin afectar su desempeño.
2. Es flexible, se ajusta a nuevos ambientes por aprendizaje, no hay que programarlo.
3. Puede manejar información difusa, con ruido o inconsistente.
4. Es altamente paralelo
5. Es pequeño, compacto y consume poca energía.

El cerebro humano constituye una computadora muy notable, es capaz de interpretar información imprecisa suministrada por los sentidos a un ritmo increíblemente veloz y lo más impresionante de todo, es que el cerebro aprende sin instrucciones explícitas de ninguna clase, a crear las representaciones internas que hacen posibles estas habilidades.

Basados en la eficiencia de los procesos llevados a cabo por el cerebro, e inspirados en su funcionamiento, varios investigadores han desarrollado desde hace más de 30 años la teoría de las Redes Neuronales Artificiales (RNA), las cuales emulan las redes neuronales biológicas, y que se han utilizado para aprender estrategias de solución basadas en ejemplos de comportamiento típico de patrones; estos sistemas no requieren que la tarea a ejecutar se programe, ellos generalizan y aprenden de la experiencia.

La teoría de las RNA ha brindado una alternativa a la computación clásica, para aquellos problemas, en los cuales los métodos tradicionales no han entregado resultados muy convincentes, o poco convenientes. Las aplicaciones más exitosas de las RNA son:

1. Procesamiento de imágenes y de voz
2. Reconocimiento de patrones
3. Planeamiento
4. Interfaces adaptivas para sistemas Hombre/máquina
5. Predicción
6. Control y optimización
7. Filtrado de señales

Los sistemas de computo tradicional procesan la información en forma secuencial; un computador serial consiste por lo general de un solo procesador que puede manipular instrucciones y datos que se localizan en la memoria, el procesador lee, y ejecuta una a una las instrucciones en la memoria; este sistema serial es secuencial, todo sucede en una sola secuencia determinística de operaciones. Las RNA no ejecutan instrucciones, responden en paralelo a las entradas que se les presenta. El resultado no se almacena en una posición de memoria, este es el estado de la red para el cual se logra equilibrio. El conocimiento de una red neuronal no se almacena en instrucciones, el poder de la red está en su topología y en los valores de las conexiones (pesos) entre neuronas (Lippmann, 1987).

Las RNA son una teoría que aún está en proceso de desarrollo, su verdadera potencialidad no se ha alcanzado todavía; aunque los investigadores han desarrollado potentes algoritmos de aprendizaje de gran valor práctico, las representaciones y procedimientos de que se sirve el cerebro, son aún desconocidas. Tarde o temprano los estudios computacionales del aprendizaje con RNA acabarán por converger a los métodos descubiertos por evolución, cuando eso suceda, muchos datos empíricos concernientes al cerebro comenzarán súbitamente a adquirir sentido y se tornarán factibles muchas aplicaciones desconocidas de las redes neuronales.

3.4) Características de una red neuronal artificial

El modelo de una neurona artificial es una imitación del proceso de una neurona biológica, puede también asemejarse a un sumador hecho con un amplificador operacional tal como se ve en la Figura 3.3.

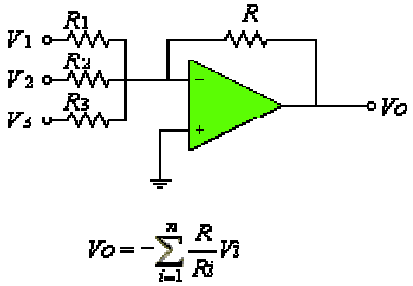


Figura 3.3 Modelo primitivo de la Neurona Artificial

Existen varias formas de nombrar una neurona artificial, es conocida como nodo, neuronodo, celda, unidad o elemento de procesamiento (EP); En la figura 3.4 se observa un PE en forma general y su similitud con una neurona biológica.

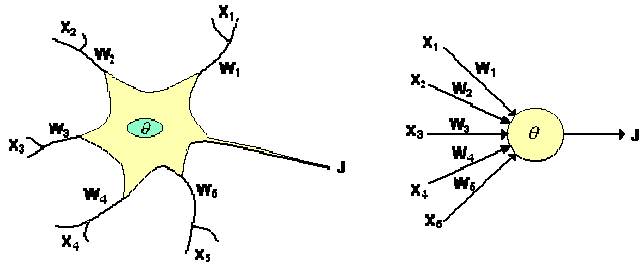


Figura 3.4 Esquema de la neurona biológica a la neurona artificial

De la observación detallada del proceso biológico se han hallado los siguientes análogos con el sistema artificial:

- Las entradas X_i representan las señales que provienen de otras neuronas y que son capturadas por las dendritas.
- Los pesos W_i son la intensidad de la sinapsis que conecta dos neuronas; tanto X_i como W_i son valores reales.
- θ es la función umbral que la neurona debe sobrepasar para activarse; este proceso ocurre biológicamente en el cuerpo de la célula.

Las señales de entrada a una neurona artificial X_1, X_2,.., X_n son variables continuas en lugar de pulsos discretos, como se presentan en una neurona biológica. Cada señal de entrada pasa a través de una ganancia o peso, llamado peso sináptico o fortaleza de la conexión cuya función es análoga a la de la función sináptica de la neurona biológica. Los pesos pueden ser positivos (excitatorios), o negativos (inhibitorios), el nodo sumatorio acumula todas las señales de entradas multiplicadas por los pesos o ponderadas y las pasa a la salida a través de una función umbral o función de transferencia. La entrada neta a cada unidad puede escribirse de la siguiente manera:

$$neta_i = \sum_{i=1}^{n} W_i X_i = \vec{X}\,\vec{W} \qquad (3.1)$$

Una idea clara de este proceso se muestra en la Figura 3.5, en donde puede observarse el recorrido de un conjunto de señales que entran a la red.

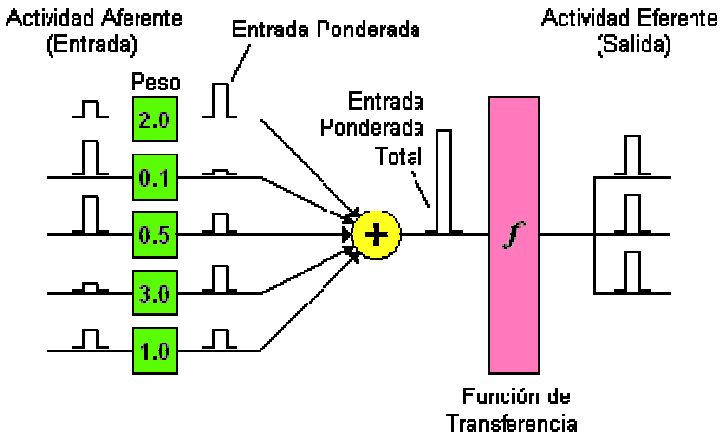


Figura 3.5 Proceso de una red neuronal

Una vez que se ha calculado la activación del nodo, el valor de salida equivale a:

$$x_i = f_i(neta_i) \tag{3.2}$$

Donde f_i representa la función de activación para esa unidad, que corresponde a la función escogida para transformar la entrada $neta_i$ en el valor de salida X_i y que depende de las características específicas de cada red.

3.5) Funciones de Transferencia o de activación

Un modelo más académico que facilita el estudio de una neurona, puede visualizarse en la Figura 3.6.

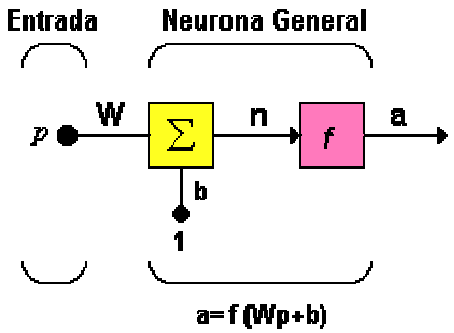


Figura 3.6 Neurona de una sola entrada

Las entradas a la red serán ahora presentadas en el vector **p**, que para el caso de una sola neurona contiene solo un elemento, **W** sigue representando los pesos y la nueva entrada **b** es una ganancia que refuerza la salida del sumador **n**, la cual es la salida neta de la red; la salida total está determinada por la función de transferencia, la cual puede ser una función lineal o no lineal de **n**, y que es escogida dependiendo de las especificaciones del problema que la neurona tenga que resolver; aunque las RNA se inspiren en modelos biológicos no existe ninguna limitación para realizar modificaciones en las funciones de salida, así que se encontrarán modelos artificiales que nada tienen que ver con las características del sistema biológico.

Limitador fuerte (Hardlim): La Figura 3.7, muestra como esta función de transferencia acerca la salida de la red a cero, si el argumento de la función es menor que cero y la lleva a uno si este argumento es mayor que uno. Esta función crea neuronas que clasifican las entradas en dos categorías diferentes, característica que le permite ser empleada en la red tipo Perceptrón.

$$a = \begin{Bmatrix} 1\mathbf{K}\,si\,\mathbf{K}\,n \geq 0 \\ 0\mathbf{K}\,si\,\mathbf{K}\,n < 0 \end{Bmatrix} \tag{3.3}$$

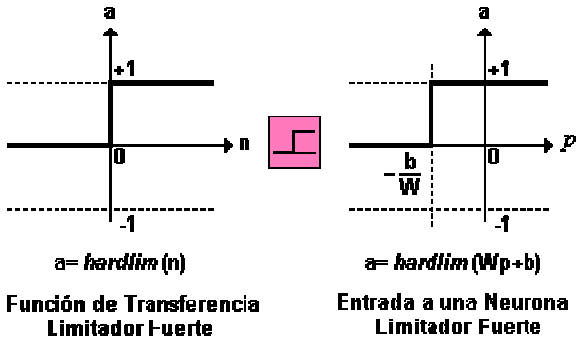


Figura 3.7 Función de transferencia Hardlim

Una modificación de esta función puede verse en la figura 3.8, la que representa la función de transferencia Hardlims que restringe el espacio de salida a valores entre 1 y –1.

$$a = \begin{Bmatrix} 1\mathbf{K}\,si\,\mathbf{K}\,n \geq 0 \\ -1\mathbf{K}\,si\,\mathbf{K}\,n < 0 \end{Bmatrix} \tag{3.4}$$

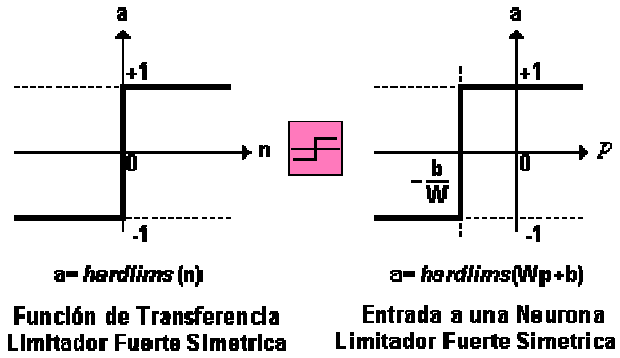


Figura 3.8 Función de transferencia Hardlims

Función de transferencia lineal (purelin): La salida de una función de transferencia lineal es igual a su entrada, véase Figura 3.9.

$$a = n$$

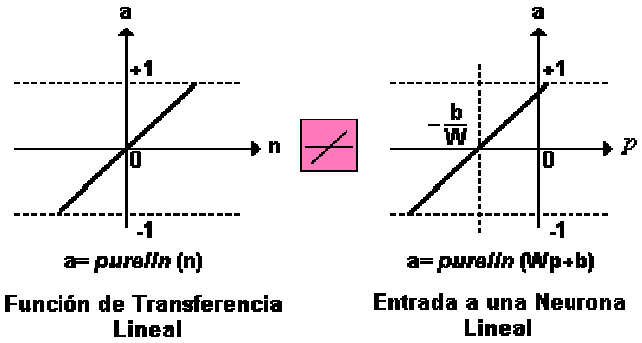


Figura 3.9 Función de transferencia lineal

Función de transferencia sigmoidal (logsig): Esta función toma los valores de entrada, los cuales pueden oscilar entre más y menos infinito, y restringe la salida a valores entre cero y uno, de acuerdo a la expresión.

$$a = \frac{1}{1+e^{-n}} \tag{3.5}$$

Esta función es comúnmente usada en redes multicapa, como la Backpropagation, en parte porque la función logsig es diferenciable Figura 3.10.

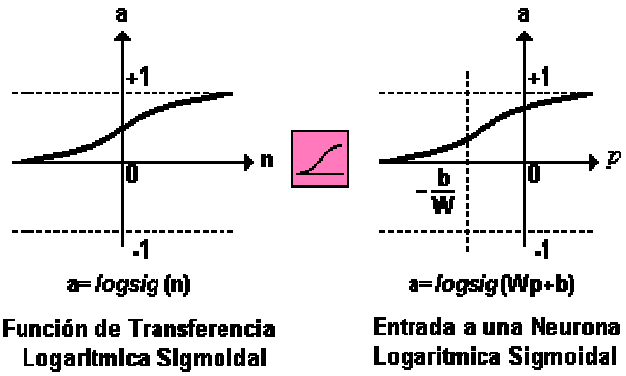


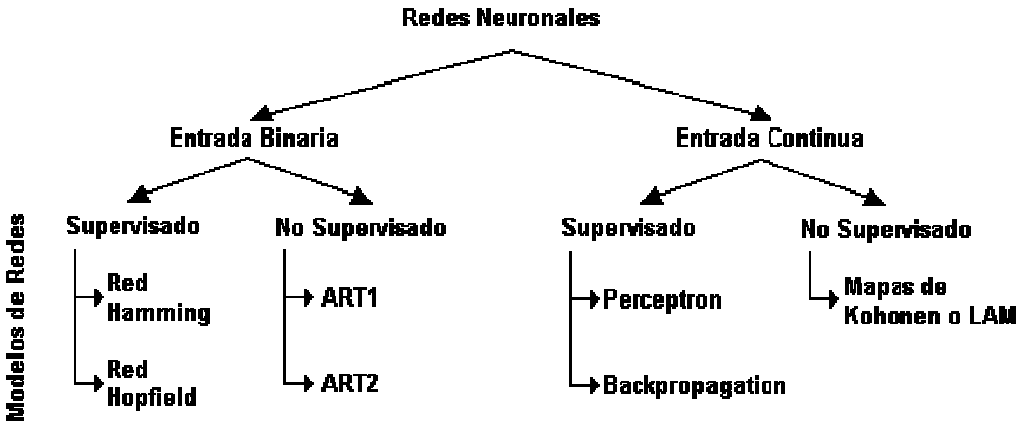
Figura 3.10 Función de transferencia sigmoidal

En el cuadro 3.1 hace una relación de las principales funciones de transferencia empleadas en el entrenamiento de redes neuronales.

Cuadro 3.1 Funciones de Transferencia

Nombre	Relación Entrada / Salida	Icono	Función
Limitador Fuerte	$a = 0\ n < 0$ $a = 1\ n \geq 0$		*hardlim*
Limitador Fuerte Simétrico	$a = -1\ n < 0$ $a = +1\ n \geq 0$		*hardlims*
Lineal Positiva	$a = 0\ n < 0$ $a = n\ 0 \leq n$		*poslin*
Lineal	$a = n$		*purelin*
Lineal Saturado	$a = 0\ n < 0$ $a = n\ 0 \leq n \leq 1$ $a = 1\ n > 1$		*satlin*
Lineal Saturado Simétrico	$a = -1\ n < -1$ $a = n\ -1 \leq n \leq 1$ $a = +1\ n > 1$		*satlins*
Sigmoidal Logarítmico	$a = \frac{1}{1+e^{-n}}$		*logsig*
Tangente Sigmoidal Hiperbólica	$a = \frac{e^{n} - e^{-n}}{e^{n} + e^{-n}}$		*tansig*
Competitiva	$a = 1$ *Neurona con n max* $a = 0$ *El resto de neuronas*	C	*compet*

3.6) Clasificación de las Redes Neuronales Artificiales



- PERCEPTRÓN
  Estructura de la Red
  Regla de Aprendizaje
  Limitación del Perceptrón
  Perceptrón Multicapa
- ADALINE
  Estructura de la Red
  Regla de Aprendizaje
- REDES COMPETITIVAS
  Red de Kohonen
  Red de Hamming
  Estructura de la Red
  Regla de Aprendizaje
  Problemas con las Redes Competitivas
  Mapas de Auto Organización (SOM)
  Learning Vector Quantization (LVQ)
- REDES RECURRENTES
  +Red de hopfield
  Estructura de la Red
  Regla de Aprendizaje
  Identificación de Sistemas Dinámicos No Lineales
  +Redes Multicapa
  Estructura de la Red
  Regla de aprendizaje
  +Red de Elman
  Estructura de la Red
  Entrenamiento de la Red
- APRENDIZAJE ASOCIATIVO
  Estructura de la Red
  Regla de Hebb
  Red Instar
  Red Oustar

En el cuadro 3.2 se muestra un resumen de las principales características de las redes neuronales artificiales, describiendo el modelo supervisado o no supervisado, hibrido, etc., así como también su respectiva regla de aprendizaje, arquitectura y principales aplicaciones hacia donde están orientadas.

Cuadro 3.2 Clasificación de las RNA de acuerdo a su modelo

Paradigma	Regla de aprendizaje	Arquitectura	Algoritmo de aprendizaje	Tareas
Supervisado	Corrección del error	Perceptrón o perceptrón multicapa	Algoritmos de aprendizaje perceptrón, retropropagación del error, ADALINE, MADALINE	Clasificación de patrones, aproximación de funciones, predicción, control, ...
		Elman y Jordan recurrentes	Retropropagación del error	Síntesis de series temporales
	Boltzmann	Recurrente	Algoritmo de aprendizaje Boltzmann	Clasificación de patrones
	Competitivo	Competitivo	LVQ	Categorización intra-clase, compresión de datos
		Red ART	ARTMap	Clasificación de patrones, categorización intra-clase
No supervisado	Corrección del error	Red de Hopfield	Aprendizaje de memoria asociativa	Memoria Asociativa
		Multicapa sin realimentación	Proyección de Sannon	Análisis de datos
	Competitiva	Competitiva	VQ	Categorización, compresión de datos
		SOM	Kohonen SOM	Categorización, análisis de datos
		Redes ART	ART1, ART2	Categorización
Por Refuerzo	Hebbian	Multicapa sin realimentación	Análisis lineal de discriminante	Análisis de datos, clasificación de patrones
		Sin realimentación o competitiva	Análisis de componentes principales	Análisis de datos, compresión de datos
Híbrido	Corrección de error y competitivo	Redes RBF	Algoritmo de aprendizaje RBF	Clasificación de patrones, aproximación de funciones, predicción, control, ...

El mundo de la inteligencia artificial resulta ser una compleja estructura de conocimientos e inclinaciones, el desarrollo y los avances tecnológicos que se han manifestados en los últimos 20 años han marcado una tendencia clara de lo que se espera sea la inteligencia artificial en este siglo que comienza.

Las redes neuronales, un tema extenso y complejo el cual resulta ser una puerta al discernimiento del propio ser humano, así mismo resulta paradójico que la inspiración del modelo biológico no se halla desarrollo al parejo de los sistemas computacionales aplicados a la inteligencia artificial o con enfoques similares para aplicarse mediante modelos.

Las RNA han sido aplicadas a un creciente número de problemas reales de considerable complejidad, por ejemplo reconocimiento de patrones, clasificación de datos, predicciones, etc. Su ventaja más importante esta en solucionar problemas que son demasiado complejos para las técnicas convencionales: problemas que no tienen un algoritmo específico para su solución, o cuyo algoritmo es demasiado complejo para ser encontrado.

En general, las Redes Neuronales Artificiales han sido claramente aceptadas como nuevos sistemas muy eficaces para el tratamiento de la información en muchas disciplinas. Ellos han dado como resultado una variedad de aplicaciones comerciales (tanto en productos como en servicios) de esta tecnología de redes neuronales.

4) REDES NEURONALES ARTIFICIALES ESTÁTICAS Y DINÁMICAS

En este capítulo se presenta una breve revisión de los conceptos que fundamentan la teoría de las redes neuronales estáticas y dinámicas. Se inicia con las redes neuronales estáticas presentando una topología de la red y una regla de aprendizaje en forma general para el ajuste de pesos; de igual manera se presenta en forma breve algunas arquitecturas muy utilizadas en las redes neuronales dinámicas dando énfasis en sus principales características de cada una de ellas.

4.1) Redes Neuronales Artificiales Estáticas

La primera clase de redes se conoce con el nombre de redes neuronales estáticas. En esta clase podemos encontrar la red Multicapa de Perceptrones, la red de Funciones de Base Radial, Red Neuronal Probabilística (RNP), etc. Todas ellas tienen como característica común el no poseer memoria, es decir, sólo son capaces de transformar un conjunto de entradas en un conjunto de salidas, de tal manera que una vez establecidos todos los parámetros de la red las salidas únicamente dependen de las entradas (Krosse y Smagt, 1996).

En general la relación deseada de entradas y salidas se determina en este caso externamente mediante alguna forma de ajuste de los parámetros del sistema supervisado. Este tipo de redes se han empleado con éxito en muchos problemas de clasificación, como funciones lógicas, así como en el campo de la aproximación funcional.

Comencemos nuestra exposición de las redes estáticas dando el modelo de neurona más empleado para éstas. En esencia, responde al modelo de neurona introducido por Rosenblatt al que denominó perceptron (Fausett, 1994). Dicha neurona se ilustra en la Figura 4.1.

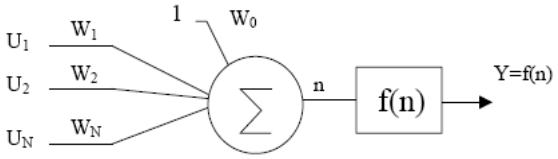


Figura 4.1 Perceptrón

Como se puede ver en la Figura 4.1 las entradas se multiplican por una serie de pesos, siendo la suma de esos productos la variable independiente de una función f(x). Dicha función recibe el nombre de función de activación, en analogía con las neuronas biológicas que inspiraron su creación. Además suele ser una función no lineal. En el modelo propuesto originalmente se tomó la función salto unidad.

$$f(n) = \begin{Bmatrix} 1 \mathbf{K}\, n > 0 \\ 0 \mathbf{K}\, n \le 0 \end{Bmatrix} \tag{4.1}$$

El resultado de la aplicación de esa función a la suma de productos nos dará la salida de la neurona. Se suele adicionar un peso más que no es multiplicado por ninguna entrada y sirve como una entrada de referencia. Este nuevo peso contribuye generalmente a hacer el aprendizaje más rápido. Una neurona de este estilo, por tanto, vendrá representada por las ecuaciones que se muestran a continuación:

$$n = \sum_{i=1}^{N} w_i u_i + w_0 \tag{4.2}$$

$$Y = f(n) \tag{4.3}$$

Donde:

N = número de entradas.	w_0 = entrada de referencia (peso bias).
u_i = entrada i-ésima.	f(n) = función de activación.
w_i = peso i-ésimo.	Y = Salida de la neurona.

A pesar de la simplicidad del perceptrón, éste es capaz de simular diferentes funciones lógicas, aun cuando el número de las mismas está limitado, denominándose a este límite, umbral de funciones lógicas (Laurene, 1994). También se ha empleado en problemas de clasificación de patrones dentro de dos clases en donde la limitación fundamental se encuentra en que la frontera entre las dos clases puede ser únicamente un hiperplano.

Existen varios algoritmos de entrenamiento diseñados para el perceptrón. Entre ellos podemos citar el algoritmo de los mínimos cuadrados (AMC) que tiene la peculiaridad de ser un caso particular del algoritmo backpropagation, algoritmo propuesto por Werbos en 1974.

Tal como hemos observado el perceptrón por si sólo tiene ciertas limitaciones, varios investigadores realizaron experimentos con una capa de perceptrones, al objeto de conseguir aprender ciertas funciones que un único perceptron no permitía aprender. El resultado de tales experimentos fue publicado por Minsky y Papert (1987) en su libro titulado “Perceptrons” donde ponían de manifiesto la incapacidad de una capa de perceptrones para aprender ciertas funciones, incluyendo la función lógica or-exclusiva. Este resultado abrió un período de pesimismo frente a las posibilidades de las redes neuronales estáticas. De hecho tales resultados llevaron al abandono de las redes neuronales durante dos décadas. Sin embargo gracias al esfuerzo de unos pocos investigadores tales como Teuvo Kohonen, Stephen Grossberg y James Anderson las redes neuronales volvieron a coger auge, dado que demostraron que utilizando varias capas de neuronas se podían solucionar problemas que con una capa eran irresolubles tales como el aprendizaje de una función or-exclusiva. A partir de entonces han surgido diferentes topologías de redes neuronales.

La topología más general de red neuronal, es aquella en donde un conjunto de neuronas se agrupan entre sí. Las salidas de unas neuronas hacen la función de entradas de otras, siendo algunas de las salidas escogidas como salidas de la red neuronal. Un cierto subconjunto de neuronas recibirán las entradas a la red. Este tipo de red neuronal recibe el nombre de red neuronal completamente

conexionada, dado que contempla todas las posibilidades de uniones entre neuronas. La mayoría de investigadores prefieren una organización de la red en forma de capas, es decir las neuronas se posicionan en forma de capas unas detrás de otras, tal que las salidas de las neuronas de unas capas sean las entradas de las neuronas de la capa siguiente. A este tipo de redes en donde las neuronas se disponen en capas sin realimentaciones entre ellas se les da el nombre de redes neuronales feedforward (Madan et al., 2003).

En estas redes estructuradas por capas se suele distinguir entre la capa de entrada, o sea la capa de neuronas que recibe las entradas a la red, la capa de salida, es decir la capa que contiene las neuronas cuyas salidas serán las que constituyan las salidas de la red y las capas ocultas que son las capas existentes entre las dos anteriores. En la Figura 4.2 se muestra una red de este tipo constituida por tres capas.

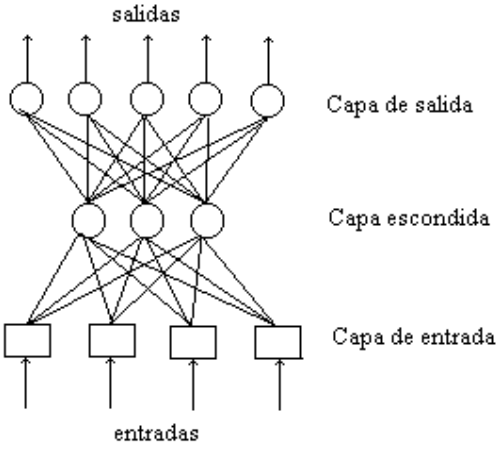


Figura 4.2 Red neuronal de 3 capas

Este es el caso de la Red Multicapa de Perceptrones (RMP), en donde el modelo de neurona adoptado es el perceptron, ilustrado en los párrafos anteriores. Sin embargo en este caso la función de activación deja de ser la función salto unidad para convertirse en la función sigmoide, la tangente hiperbólica o cualquier otra función no lineal que sea continua, suave y monótonamente creciente. La función salto unidad presenta el inconveniente de no ser derivable, circunstancia ésta, que hace más difícil el diseño de algoritmos de entrenamiento basados en el gradiente.

En ocasiones se han utilizado funciones de activación lineales para las neuronas de la capa de salida, motivado por el hecho de facilitar el aprendizaje. En general se han aplicado varios algoritmos de entrenamiento a la RMP, siendo el más usado el “backpropagation”.

4.2) Backpropagation

La regla de aprendizaje del Perceptrón de Rosenblatt y el algoritmo LMS (Least-Mean-Square) de Widrow y Hoff fueron diseñados para entrenar redes de una sola capa. Como se discutió anteriormente, estas redes tienen la desventaja que solo pueden resolver problemas linealmente separables, fue esto lo que llevo al surgimiento de las redes multicapa para sobrepasar esta dificultad en las redes hasta entonces conocidas.

El primer algoritmo de entrenamiento para redes multicapa fue desarrollado por Paul Werbos en 1974, este se desarrolló en un contexto general, para cualquier tipo de redes, siendo las redes neuronales una aplicación especial, razón por la cual el algoritmo no fue aceptado dentro de la comunidad de desarrolladores de redes neuronales (Freeman y Skapura, 1991).

Fue solo hasta mediados de los años 80 cuando el algoritmo Backpropagation o algoritmo de propagación inversa fue redescubierto al mismo tiempo por varios investigadores, David Rumelhart, Geoffrey Hinton y Ronal Williams, David Parker y Yann Le Cun.

El algoritmo se popularizó cuando fue incluido en el libro "Parallel Distributed Processing Group" por los psicólogos David Rumelhart y James McClelland. La publicación de este libro trajo consigo un auge en las investigaciones con redes neuronales, siendo la Backpropagation una de las redes más ampliamente empleadas, aun en nuestros días.

Uno de los grandes avances logrados con la Backpropagation es que esta red aprovecha la naturaleza paralela de las redes neuronales para reducir el

tiempo requerido por un procesador secuencial para determinar la correspondencia entre unos patrones dados. Además el tiempo de desarrollo de cualquier sistema que se esté tratando de analizar se puede reducir como consecuencia de que la red puede aprender el algoritmo correcto sin que alguien tenga que deducir por anticipado el algoritmo en cuestión.

La mayoría de los sistemas actuales de cómputo se han diseñado para llevar a cabo funciones matemáticas y lógicas a una velocidad que resulta asombrosamente alta para el ser humano. Sin embargo la destreza matemática no es lo que se necesita para solucionar problemas de reconocimiento de patrones en entornos ruidosos, característica que incluso dentro de un espacio de entrada relativamente pequeño, puede llegar a consumir mucho tiempo. El problema es la naturaleza secuencial del propio computador; el ciclo tomar–ejecutar de la naturaleza Von Neumann solo permite que la máquina realice una operación a la vez. En la mayoría de los casos, el tiempo que necesita la máquina para llevar a cabo cada instrucción es tan breve (típicamente una millonésima de segundo) que el tiempo necesario para un programa, así sea muy grande, es insignificante para los usuarios. Sin embargo, para aquellas aplicaciones que deban explorar un gran espacio de entrada o que intentan correlacionar todas las permutaciones posibles de un conjunto de patrones muy complejo, el tiempo de computación necesario se hace bastante grande.

Lo que se necesita es un nuevo sistema de procesamiento que sea capaz de examinar todos los patrones en paralelo. Idealmente ese sistema no tendría que ser programado explícitamente, lo que haría es adaptarse a sí mismo para aprender la relación entre un conjunto de patrones dado como ejemplo y ser capaz de aplicar la misma relación a nuevos patrones de entrada. Este sistema debe estar en capacidad de concentrarse en las características de una entrada arbitraria que se asemeje a otros patrones vistos previamente, sin que ninguna señal de ruido lo afecte. Este sistema fue el gran aporte de la red de propagación inversa, Backpropagation.

La Backpropagation es un tipo de red de aprendizaje supervisado, que emplea un ciclo propagación – adaptación de dos fases. Una vez que se ha aplicado un patrón a la entrada de la red como estímulo, este se propaga desde la primera capa a través de las capas superiores de la red, hasta generar una salida. La señal de salida se compara con la salida deseada y se calcula una señal de error para cada una de las salidas (De Jesús y Hagan, 2007).

Las salidas de error se propagan hacia atrás, partiendo de la capa de salida, hacia todas las neuronas de la capa oculta que contribuyen directamente a la salida. Sin embargo las neuronas de la capa oculta solo reciben una fracción de la señal total del error, basándose aproximadamente en la contribución relativa que haya aportado cada neurona a la salida original. Este proceso se repite, capa por capa, hasta que todas las neuronas de la red hayan recibido una señal de error que describa su contribución relativa al error total.

Basándose en la señal de error percibida, se actualizan los pesos de conexión de cada neurona, para hacer que la red converja hacia un estado que permita clasificar correctamente todos los patrones de entrenamiento.

La importancia de este proceso consiste en que, a medida que se entrena la red, las neuronas de las capas intermedias se organizan a sí mismas de tal modo que las distintas neuronas aprenden a reconocer distintas características del espacio total de entrada. Después del entrenamiento, cuando se les presente un patrón arbitrario de entrada que contenga ruido o que esté incompleto, las neuronas de la capa oculta de la red responderán con una salida activa si la nueva entrada contiene un patrón que se asemeje a aquella característica que las neuronas individuales hayan aprendido a reconocer durante su entrenamiento. Y a la inversa, las unidades de las capas ocultas tienen una tendencia a inhibir su salida si el patrón de entrada no contiene la característica para reconocer, para la cual han sido entrenadas.

Varias investigaciones han demostrado que, durante el proceso de entrenamiento, la red Backpropagation tiende a desarrollar relaciones internas

entre neuronas con el fin de organizar los datos de entrenamiento en clases. Esta tendencia se puede extrapolar, para llegar a la hipótesis consistente en que todas las unidades de la capa oculta de una Backpropagation son asociadas de alguna manera a características específicas del patrón de entrada como consecuencia del entrenamiento. Lo que sea o no exactamente la asociación puede no resultar evidente para el observador humano, lo importante es que la red ha encontrado una representación interna que le permite generar las salidas deseadas cuando se le dan las entradas, en el proceso de entrenamiento. Esta misma representación interna se puede aplicar a entradas que la red no haya visto antes, y la red clasificará estas entradas según las características que compartan con los ejemplos de entrenamiento.

4.3) Algoritmo de aprendizaje backpropagation

Consideremos la red representada en forma simplificada en la figura 4.3 y supongamos que se dispone de un conjunto de pares de datos, $f(x)\ x^p, y^p,\ \ p = 1, 2, \mathbf{K}, N$.

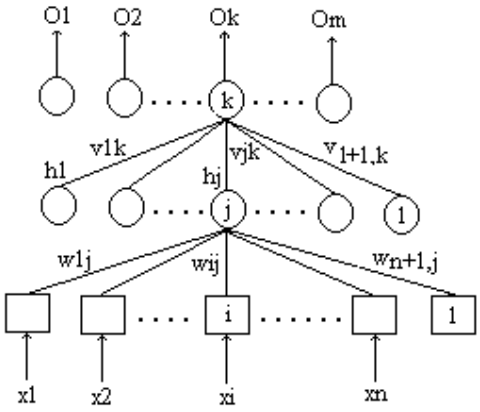


Figura 4.3 Esquema de una red neuronal de 3 capas

Para la neurona ***j*** de la capa escondida, definimos la función de excitación, correspondiente a la muestra ***p*** de los datos:

$$S_j^p = \sum_{i=1}^{n} w_{ij}^p x_i^p + w_{n+1,j}^p \qquad (4.4)$$

Es conveniente aclarar en este punto que los datos de entrada y de salida x_i, y_i deben estar normalizados en el intervalo 0-1 (Man et al., 2006). Los coeficientes w_{ij}, v_{jk}, se denominan "pesos" y a la entrada n+1, cuyo valor es fijado en el valor 1, se le conoce como "bias". Este último elemento permite que la red pueda modelar sistemas en que las salidas pueden tener valores distintos de cero cuando todas las entradas son cero.

La salida de la neurona *j* de la capa escondida se calcula generalmente como una función no-lineal de la excitación, aunque en algunos casos particulares puede ser también una función lineal. Por el momento vamos a definir la función no lineal de salida o función de activación, como la función sigma, tanto para las neuronas de la capa escondida como para las de la capa de salida. Esta función presenta considerables ventajas dada su simplicidad, aunque es posible utilizar otras funciones como la tangente hiperbólica, la función de Gauss, etc.

Entonces la salida de la neurona *j* de la capa escondida, se expresa:

$$h_j^p = f(S_j^p) = \frac{1}{1+\exp(-S_j^p)} \tag{4.5}$$

La excitación de la neurona *k* de la capa de salida, se calcula en forma análoga, mediante la expresión:

$$r_k^p = \sum_{j=1}^{l} v_{jk}^p h_j^p + v_{l+1,k}^p \tag{4.6}$$

Y por último, la salida de la neurona *k* de la capa de salida se expresa como:

$$O_k^p = f(r_k^p) = \frac{1}{1+\exp(-r_k^p)} \tag{4.7}$$

Definimos ahora al error en la salida *k*, para la muestra *p* como:

$$e_k^p = y_k^p - O_k^p \tag{4.8}$$

Donde y_k^p es la muestra *p* de la salida *k*. El criterio a minimizar en la muestra *p* se define:

$$E_p = \frac{1}{2}\sum_{k=1}^{m}(e_k^p)^2 = \frac{1}{2}\sum_{k=1}^{m}(y_k^p - O_k^p)^2 \tag{4.9}$$

O también:

$$E_p = \frac{1}{2}\sum_{k=1}^{m}(y_k^p - f(r_k^p))^2 \tag{4.10}$$

El proceso de entrenamiento de la red consiste entonces en presentar secuencialmente las entradas $x_i^p (i = 1,2,...,n)$, $(p = 1,2,...,m)$, calcular las salidas de la red $O_k^p (k = 1,2,...,m)$, $(p = 1,2,...,m)$, los errores e_k^p y el criterio E_p y aplicar algún procedimiento de minimización de la función E_p con respecto a los coeficientes de peso w_{ij} y v_{jk} de manera que estos se vayan aproximando paulatinamente a los valores que garantizan un error mínimo entre las salidas de la red O_k y los datos de salida y_k.

Este procedimiento se repite tantas veces como sea necesario, es decir, los vectores de datos x^p e y^p se utilizan reiteradamente hasta tanto el error en cada salida y el criterio E_p, para $(p = 1,2,...,m)$, se encuentren por debajo del límite prefijado. Una vez lograda esta condición, se dice que la red está entrenada, lo que significa que ella será capaz de reproducir la función $y = f(x)$ con suficiente exactitud si los datos de entrenamiento han sido bien seleccionados

y suficientes. A cada ciclo de uso de los datos de entrenamiento suele denominársele “época” en el argot de las redes neuronales.

Para minimizar la función E_p con respecto a los coeficientes de peso w_{ij} y v_{jk} vamos a utilizar el método del “descenso más rápido” (steepest descent) que consiste en moverse siempre en la dirección del negativo del gradiente de la función E_p con respecto a los coeficientes w_{ij} y v_{jk}.

Ahora bien después del desarrollo matemático que involucra el cálculo de derivadas parciales con respecto a las neuronas de la capa de salida y la capa de entrada con respecto a la escondida se tiene los siguientes resultados para el ajuste de pesos:

$$v_{jk}^{p} = v_{jk}^{p-1} + h\, d_k^{\,p}\, h_j^{p} \tag{4.11}$$

$$w_{ij}^{p} = w_{ij}^{p-1} + h\, \Delta_j^{\,p}\, x_i^{p} \tag{4.12}$$

$$\Delta_j^{\,p} = \left(\sum_{k=1}^{m} d_k^{\,p}\, v_{jk}^{p} \right) h_j^{p} \,(1 - h_j^{p}) \tag{4.13}$$

$$d_k^{\,p} = (y_k^{p} - O_k^{p})\, O_k^{p}\,(1 - O_k^{p}) \tag{4.14}$$

En las que η es una constante que se denomina coeficiente de aprendizaje.

Es importante destacar que la estrategia de entrenamiento de una red neuronal mediante el algoritmo de retropropagación explicado hasta aquí, tiene algunas particularidades que lo distinguen de un problema convencional de optimización.

Si se dispone de una muestra de *M* pares de valores x^p, y^p de entrada y salida, estos datos serán presentados secuencialmente a la red, calculándose una

actualización de los coeficientes de peso w_{ij}, v_{jk} para cada par de datos, hasta agotarlos todos, es decir, para $p = 1,2,...M$. Es muy frecuente que un solo ciclo o época no sea suficiente para lograr un valor aceptablemente pequeño del criterio E_p para todo *p* y entonces es necesario reciclar los datos, o en otras palabras, comenzar una nueva época. Dependiendo del tamaño de la muestra de entrenamiento, se necesitarán más o menos épocas, pudiendo ser su número, desde unas decenas, hasta cientos o miles.

En las técnicas de gradiente descendiente es conveniente avanzar por la superficie de error con incrementos pequeños de los pesos; esto se debe a que tenemos una información local de la superficie y no se sabe lo lejos o lo cerca que se está del punto mínimo, con incrementos grandes, se corre el riesgo de pasar por encima del punto mínimo, con incrementos pequeños, aunque se tarde más en llegar, se evita que esto ocurra Hagan et al., (2002). El elegir un incremento adecuado influye en la velocidad de convergencia del algoritmo, esta velocidad se controla a través de la rata de aprendizaje *η*, la que por lo general se escoge como un número pequeño, para asegurar que la red encuentre una solución. Un valor pequeño de *η* significa que la red tendrá que hacer un gran número de iteraciones, si se toma un valor muy grande, los cambios en los pesos serán muy grandes, avanzando muy rápidamente por la superficie de error, con el riesgo de saltar el valor mínimo del error y estar oscilando alrededor de él, pero sin poder alcanzarlo.

Es recomendable aumentar el valor de *η* a medida que disminuye el error de la red durante la fase de entrenamiento, para garantizar así una rápida convergencia, teniendo la precaución de no tomar valores demasiado grandes que hagan que la red oscile alejándose demasiado del valor mínimo. Algo importante que debe tenerse en cuenta, es la posibilidad de convergencia hacia alguno de los mínimos locales que pueden existir en la superficie del error del espacio de pesos como se ve en la Figura 4.4.

En el desarrollo matemático que se ha realizado para llegar al algoritmo Backpropagation, no se asegura en ningún momento que el mínimo que se

encuentre sea global, una vez la red se asiente en un mínimo sea local o global cesa el aprendizaje, aunque el error siga siendo alto. En todo caso, si la solución es admisible desde el punto de vista del error, no importa si el mínimo es local o global o si se ha detenido en algún momento previo a alcanzar un verdadero mínimo.

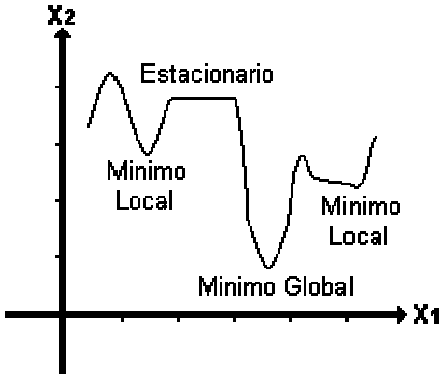


Figura 4.4 Superficie típica de error

El método de la retropropagación basado en el algoritmo del descenso más rápido descrito por Fletcher y Powell en 1963, tiene la virtud de su simplicidad, no obstante, como es conocido, ese algoritmo puede resultar en algunos casos, demasiado lento y además, no siempre garantiza la convergencia a un mínimo absoluto, es decir, puede empantanarse en un mínimo local.

Existen otros métodos, como los llamados del tipo Quasi-Newton de Powell en 1970 que garantizan la convergencia a un mínimo absoluto y además convergen más rápido, siendo característico de ellos que en cada etapa se calcula un tamaño del paso óptimo, para lo que se requiere el cálculo o estimación de la matriz Hessiana, formada por las segundas derivadas del criterio E_p con respecto a w_{ij} y v_{jk}. No obstante, para redes de dimensiones grandes, el número de coeficientes de peso puede ser hasta de varios cientos o miles, lo cual hace que la dimensión del problema de optimización crezca intolerablemente.

Concluyendo, podemos decir que el método de retropropagación convencional presentado aquí, es una alternativa simple y económica que debe ser intentada en primer lugar antes de acudir a otros métodos más sofisticados.

4.4) Redes Neuronales Artificiales Dinámicas

Las redes neuronales dinámicas se caracterizan por tener una dependencia de sucesos ocurridos en instantes pasados. Desde un punto de vista matemático esto se refleja en el hecho de que en las ecuaciones que las definen aparecen ecuaciones diferenciales o ecuaciones en diferencia según sean redes continuas o discretas. Son éstas las que le confieren una cierta memoria, que resulta imprescindible en problemas tales como la identificación de sistemas dinámicos. La principal atracción de este tipo de redes es que tienen memoria interna y por lo tanto son en sí mismas un sistema dinámico.

Como primer intento de incorporar un comportamiento dinámico en las redes neuronales se diseñaron redes neuronales dinámicas constituidas por una combinación entre redes neuronales estáticas y sistemas dinámicos. En la figura 4.5 representamos cuatro configuraciones conectadas en cascada y con realimentaciones que representan las unidades básicas con las cuales se construye cualquier red dinámica de este tipo. Obsérvese que W(z) hace referencia a la matriz de transferencia del sistema dinámico y N es la red neuronal estática o dado que podemos ver las redes neuronales estáticas como operadores no lineales, también se puede considerar como un operador no lineal. Cuando en las configuraciones de la figura 4.5 utilizamos varias redes neuronales estáticas denotamos cada una de ellas mediante un superíndice.

Mediante estas configuraciones, considerando también el operador multiplicación por una constante, podemos construir una gran variedad de redes dinámicas discretas. Hemos de comentar que las redes consideradas son discretas, sin embargo el análisis se puede extender a las redes dinámicas continuas sin más que tomar matrices de transferencia en el dominio s (W(s)) correspondientes a sistemas dinámicos continuos. Lo que se traduce en el dominio temporal en sustituir los operadores retardo por integradores.

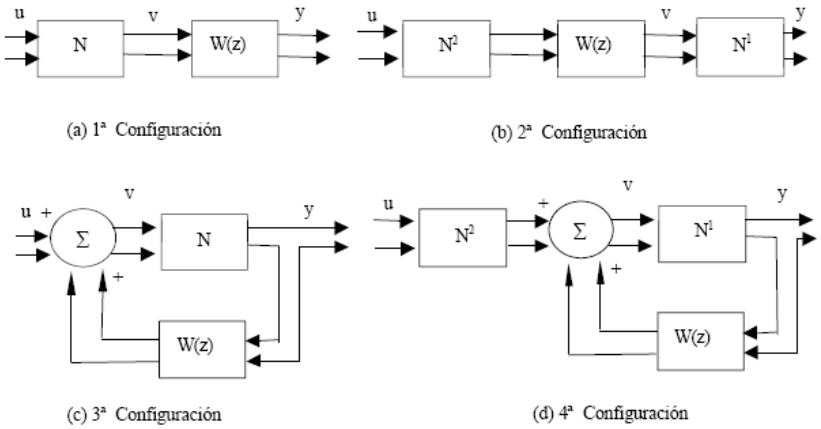


Figura 4.5 Configuraciones básicas de redes neuronales dinámicas.

Dentro de esta clase de redes encontramos como aproximación más sencilla la red de retardos en el tiempo o Time Delay Neural Network (TDNN). En esta red el sistema dinámico se introduce entre las entradas y la red estática, de tal forma que el sistema dinámico actúa retardando las entradas.

Esta red neuronal ha sido aplicada con éxito en problemas de conversión de texto a voz, así como en la predicción de series temporales no lineales. Otra red neuronal que ha suscitado un gran interés es la red de Narendra y Parthasarathy (1990), en donde además de retrasar las entradas, realimentamos las salidas de la red a través de un sistema dinámico. Esta arquitectura corresponde a la combinación de la primera y tercera configuración de la figura 4.5, atribuyéndose el nombre genérico de redes con realimentación de las salidas a las redes que realimentan las salidas con el objeto de introducir un comportamiento dinámico. En teoría este tipo de redes son capaces de modelizar cualquier sistema dinámico cuya relación entre la entrada y salida se pueda expresar como:

$$y(k) = F\left[u(k), u(k-1), \ldots, u(k-n), y(k-1), y(k-2), \ldots, y(k-m)\right] \qquad (4.15)$$

Donde:

y = salidas de la red.

m = número de salidas previas tomadas.

u = entradas a la red.

n = número de entradas previas tomadas.

Otro tipo de redes dinámicas son las denominadas redes con realimentación de estados. Estas consisten en una capa de neuronas interconectadas entre sí como se muestra en la figura 4.6. En general muchos autores denominan a las redes dinámicas donde hay alguna clase de realimentación redes neuronales recurrentes. La introducción de realimentaciones favorece el aprendizaje. Este hecho se basa en que un sistema con realimentación es equivalente a un gran sistema feedforward, posiblemente con un número infinito de unidades. Así es bien conocido que un circuito secuencial es equivalente a un circuito con un número infinito de puertas lógicas colocadas en forma de capas unas detrás de las otras.

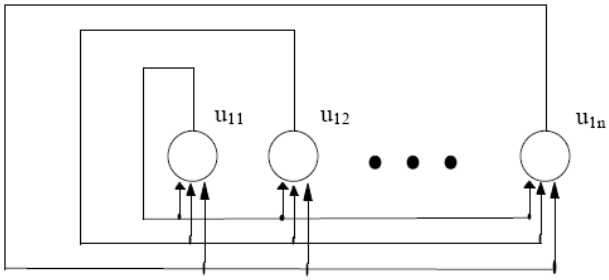


Figura 4.6 Redes con realimentación de estados

Una de las primeras redes propuestas de este tipo es la red de Hopfield. Debido a este hecho, ha sido una de las más estudiadas. Según tomemos el tiempo de forma continua o a intervalos regulares tenemos la red de Hopfield de tiempo continuo o la red de Hopfield de tiempo discreto. Aquí únicamente consideraremos la red de Hopfield de tiempo discreto ilustrando las ideas implícitas a este modelo de red.

Los nodos de esta red vienen caracterizados por las ecuaciones en diferencia:

$$y_i(k) = \sum_{j=1}^{N} w_{ij} u_j(k) + v_i \tag{4.16}$$

$$u_i(k+1) = f(y_i(k)) \tag{4.17}$$

Donde:

u_i = Salida de la neurona j.

v_i = Entrada externa a la neurona i.

N = Número de neuronas en la primera capa.

Hemos de destacar que para la red de Hopfield se han empleado diferentes funciones de activación. En problemas de reconocimiento de patrones la función más comúnmente utilizada ha sido la función salto unidad. Se ha hecho un estudio detallado de sus propiedades de estabilidad y convergencia. Mediante la aplicación del segundo método de Lyapunov, se ha demostrado como condición suficiente para la estabilidad el que la matriz de pesos sea simétrica y definida positiva. Son estos principios los que se utilizan en la denominada memoria de Hopfield asociativa, una de las aplicaciones más frecuentes de la red de Hopfield, en donde se asignan los pesos de tal forma que el sistema posea los puntos de equilibrio adecuados a la aplicación.

Aunque la aplicación más común de las redes de Hopfield es en problemas de memoria asociativa, también se ha empleado en otros problemas tales como la descomposición de señales en un conjunto específico de funciones base. Esta propiedad ha sido utilizada en la identificación de sistemas en el dominio frecuencial. Otra red de características similares a la red neuronal de Hopfield es la red neuronal recurrente de tiempo discreto. Sus nodos vienen descritos por las ecuaciones mostradas a continuación:

$$u_i(k+1) = f\left(\sum_{j=0}^{M+N} w_{ij} v_j(k) \right) \tag{4.18}$$

Donde:

$$u_i(k) = \begin{Bmatrix} 1, i = 0 \\ u_i(k), i = 1,2,...,N \\ v_{i-k}(k), i = N+1,...,N+M \end{Bmatrix} \quad (4.19)$$

Obsérvese que:

N = Número de salidas.

M = Número de entradas externas.

u_i = Salida de la neurona i.

v_{i-N} = Salida i-N.

Esta red en la terminología inglesa se conoce con el nombre de “Discrete Time Recurrent Neural Network” y de forma abreviada DTRNN. Ha sido aplicada con éxito en el aprendizaje de autómatas deterministas finitos. En cuanto a los algoritmos de aprendizaje, estos son más complejos que los algoritmos empleados en el caso de las redes neuronales estáticas.

En general existen dos aproximaciones a la hora de elaborar nuevos algoritmos de aprendizaje para redes dinámicas. Por un lado, existen algoritmos que tratan de desdoblar a lo largo del tiempo la red neuronal de tal forma que podamos aplicar las ideas vistas para las redes estáticas, así uno de tales algoritmos es el algoritmo de aprendizaje “Backpropagation Through Time” propuesto por Werbos, donde se propaga el error en vez de a través de las capas como en el caso de la RMP a través del tiempo. Como una simplificación de este algoritmo surge el algoritmo “Truncated Back-propagation Through Time”, donde se toman únicamente un cierto número de instantes a la hora de calcular el gradiente.

Por otra parte existen otros algoritmos de aprendizaje basados en calcular de forma recursiva el gradiente, uno de ellos es el “Real Time Recurrent Learning” (RTRL).

En el contexto de las redes recurrentes existen redes dinámicas por naturaleza como lo son la red de Hopfield, la red de Jordan y la red de Elman que siendo de naturaleza estática como lo son las redes multicapa logran el comportamiento dinámico realimentando sus entradas con muestras anteriores de las salidas, el comportamiento dinámico de dichas redes hace que sean una poderosa herramienta para simular e identificar sistemas dinámicos no lineales Medsker y Jain (2001).

Las redes recurrentes son aquellas que poseen neuronas recurrentes, es decir, aquellas que disponen de conexiones con ellas mismas, con neuronas de capas anteriores y de la misma capa, además de las conexiones hacia delante.

En general, las redes recurrentes son mucho más potentes, en todos los aspectos, que las estáticas, dado que añaden un carácter temporal a las redes, introduciendo el concepto de memoria en estas. Es decir, las redes recurrentes son capaces de “recordar” instantes anteriores de tiempo y, por lo tanto, resultan idóneas para el análisis de secuencias temporales, y en concreto para:

- Modelización neurobiológica
- Tareas lingüísticas
- Reconocimiento de palabras y fonemas
- Control de procesos dinámicos

Sin embargo, la potencia que ofrecen estas redes viene eclipsada por el gran incremento de parámetros para el entrenamiento que provoca la recurrencia. En general, el análisis y el entrenamiento de estas redes resulta mucho más complejo que las redes estáticas, en las que la red actuaba como un simple “mapeador”. Es por ello por lo que este tipo de arquitecturas es menos utilizado que las estáticas, incluso a la hora de analizar secuencias temporales.

Dentro de la familia de las redes recurrentes se disponen de diversos tipos de arquitecturas. Se analizarán 3 de las más comunes: La red *Hopfield*, la red *Elman* y la red *Jordan*.

Red Hopfield:

La red Hopfield fue desarrollada por el físico de mismo nombre en 1982. Este tipo de red se engloba dentro de las redes totalmente recurrentes y se caracteriza porque cada una de las neuronas de la red tiene conexión con todas las demás.

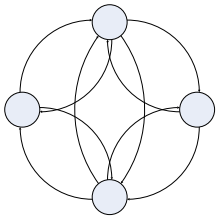


Figura 4.7 Red Hopfield conexionado totalmente recurrente

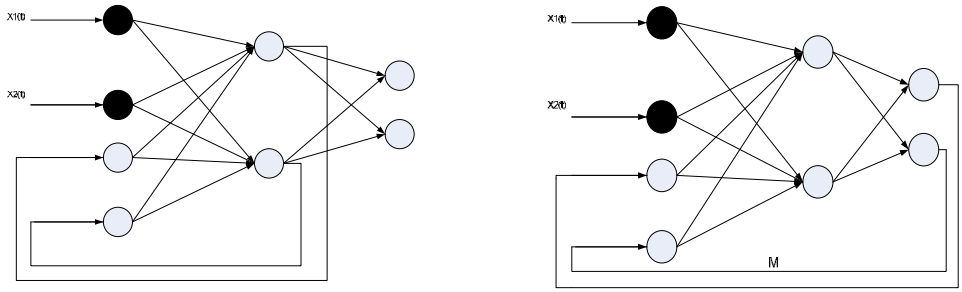
El funcionamiento de la red es el de una memoria asociativa, permitiendo el almacenamiento y recuperación de patrones incluso cuando estos están incompletos. Los patrones que procesa son estáticos, aunque su procesamiento es temporal, es decir, la red requiere de un cierto tiempo de evolución hasta ofrecer la salida. La filosofía de la red Hopfield es diferente al de resto de redes. Su arquitectura carece de entradas o salidas en forma de neuronas. En el caso de estas redes, se habla del estado de la red. Es decir, se inicializan las neuronas a un cierto valor y la red evoluciona hacia un estado estable, que corresponderá con la salida asociada a la entrada.

Este tipo de arquitectura se utiliza mucho para optimización y reconstrucción de patrones a partir de datos incompletos.

Redes Elman y Jordan:

Las redes Elman y Jordan, definidas por los investigadores de mismo nombre, son redes parcialmente recurrentes, es decir, son redes con conexionado feed-fordward, a las que se le han añadido algunas conexiones hacia atrás.

Tanto en la red Elman como en la Jordan, la arquitectura básica de la red es básicamente una red feed-fordward, y, en concreto, una Perceptron Multicapa, dado que las activaciones de todas las neuronas son saturantes. Sin embargo, de todas las entradas de las que se dispone en la capa de entrada algunas son utilizadas para recoger la información de la que disponían otras neuronas en el instante anterior. Estas neuronas se denominan entradas de contexto, dado que hacer referencia al estado anterior de la red.



Elman Jordan

Figura 4.8 Redes Elamn y Jordan Respectivamente

La diferencia principal entre los dos tipos de arquitectura está en la información que se transmite a las entradas de contexto. En las redes Elman, se realimentan las salidas de las neuronas de la última capa oculta, de modo que, en cierto sentido, la red dispone de información acerca de la entrada del instante anterior.

En las redes Jordan, sin embargo, existe una doble recurrencia. Por una parte se realimentan las salidas del instante anterior ponderadas con un parámetro fijo μ, y además, cada neurona de contexto recibe una copia de su estado anterior. El parámetro μ determina el horizonte de la memoria de la red, es decir, determina la ventana de tiempo que recuerda la red los datos de salida.

El aprendizaje de estos dos tipos de redes se basa en el algoritmo de retropropagación, dado que a la hora de entrenar se desacoplan los bucles. De este modo, inicialmente se calculan las salidas y se hallan los datos para pasar a

las neuronas de contexto. En el siguiente instante, se consideran estos datos como entradas a la red, y se aplica de nuevo el algoritmo, y así sucesivamente.

4.5) ¿Porque usar redes neuronales en los sistemas de control?

Las redes neuronales artificiales constituyen una excelente herramienta para el aprendizaje de relaciones complejas a partir de un conjunto de ejemplos. Gracias a esto se ha generado un gran interés en la utilización de modelos de redes neuronales para la identificación de sistemas dinámicos con no linealidades desconocidas (Gómez, 1999). Además debido a sus capacidades de aproximación así como a sus inherentes características de adaptabilidad, las redes neuronales artificiales presentan una importante alternativa en el modelado de sistemas no lineales. Otra de sus ventajas es la posibilidad de implementación en paralelo y su respuesta relativamente rápida que constituye un incentivo para la investigación futura utilizando este enfoque en problemas que involucren sistemas dinámicos no lineales.

Por último decir que la identificación de sistemas complejos y no lineales ocupa un lugar importante en las arquitecturas de sistemas de control, como por ejemplo el control inverso, control adaptativo directo e indirecto, etc. Es habitual en esos enfoques usar redes neuronales feedforward con memoria en la entrada o bien redes recurrentes (modelos de Elman o Jordan) entrenadas en línea o fuera de línea para capturar la dinámica del sistema y utilizarla en el lazo de control.

5) ALGORITMO DE CONTROL NEURONAL RECURRENTE

En este capítulo se presenta el desarrollo completo de un esquema de control basado en redes neuronales recurrentes, el cual está compuesto principalmente por 3 estructuras bien definidas: el neuroidentificador diagonal recurrente (NIDR), el neurocontrolador diagonal recurrente (NCDR) y el algoritmo de retropropagación dinámico (RPD) para ambos.

5.1) Introducción

La red neuronal diagonal recurrente (RNDR) desarrollada se basa en el modelo de la red de Elman que típicamente posee dos capas, cada una compuesta de una red tipo Backpropagation, con la adición de una conexión de realimentación desde la salida de la capa oculta hacia la entrada de la misma capa oculta, esta realimentación permite a la red de Elman aprender a reconocer y generar patrones temporales o variantes con el tiempo.

La topología de la red neuronal recurrente que se mostrará a continuación generalmente posee neuronas con función transferencia sigmoidal en su capa oculta y neuronas con función de transferencia tipo lineal en la capa de salida, la ventaja de la configuración de esta red de dos capas con este tipo de funciones de trasferencia, es que puede aproximar cualquier función con la precisión deseada mientras que esta posea un numero finito de discontinuidades, para lo cual la precisión de la aproximación depende de la selección del número adecuado de neuronas en la capa oculta.

Para la red de Elman la capa oculta es la capa recurrente y el retardo en la conexión de realimentación almacena los valores de la iteración previa, los cuales serán usados en la siguiente iteración; dos redes de Elman con los mismos parámetros y entradas idénticas en las mismas iteraciones podrían producir salidas diferentes debido a que pueden presentar diferentes estados de realimentación.

El entrenamiento de la red puede resumirse en los siguientes pasos:

- Presentar a la red, los patrones de entrenamiento y calcular la salida de la red con los pesos iníciales, comparar la salida de la red con los patrones objetivo y generar la secuencia de error.
- Propagar inversamente el error para encontrar el gradiente del error para cada conjunto de pesos y ganancias,
- Actualizar todos los pesos y ganancias con el gradiente encontrado con base en el algoritmo de propagación inversa.

5.2) Modelo de la RNDR para un sistema de control

Considere la figura 5.1 donde para cada tiempo discreto k, $\mathrm{I_i(k)}$ es la i-ésima entrada, $\mathrm{S_j(k)}$ es la suma de las entradas de la j-ésima neurona recurrente, $X_j(\mathrm{k})$ es la salida de la j-ésima neurona recurrente, $O(\mathrm{k})$ es la salida de la red. W^I, W^O, W^D representan los vectores de pesos de la entrada, salida y diagonal respectivamente.

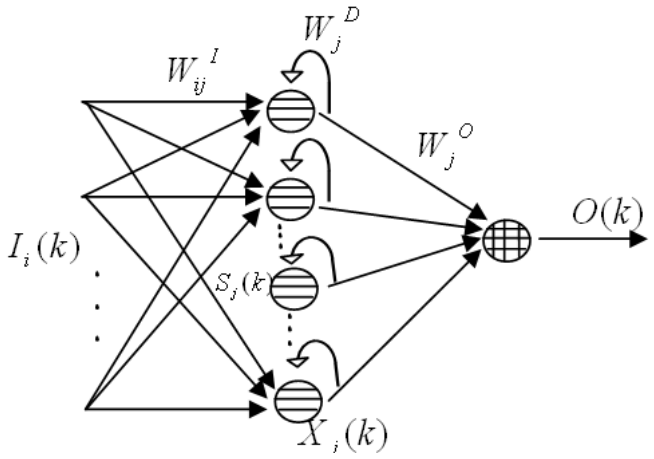


Figura 5.1 Esquema de la red neuronal diagonal recurrente.

Una aproximación para el control y el sistema de identificación usando la RNDR es presentado en este apartado. Una planta desconocida es identificada por un sistema identificador llamado NIDR, el cual provee información acerca de la planta para el controlador llamado NCDR. El neurocontrolador es usado para

manejar la dinámica desconocida del sistema así como minimizar el error de salida entre la planta y la salida del modelo.

Un algoritmo generalizado llamado RPD es desarrollado para entrenar ambos NIDR y NCDR. Para simplificar, la planta se considerará una entrada y una salida. Ambos NIDR y NCDR tienen la misma arquitectura RNDR mostrada en la figura 5.1, las cuales solo tienen una capa escondida con una neurona tipo sigmoide recurrente. El diagrama de bloques del sistema de control basado en la RNDR es mostrado en la figura 5.2. Las entradas al NCDR es la entrada de referencia, la salida previa de la planta, la señal de control previa; y la salida del NCDR es la señal de control hacia la planta. Usando el algoritmo de retropropagación dinámico los pesos del NCDR son ajustados para que el error entre la salida de la planta y la salida deseada del modelo de referencia sea muy pequeño y que este entrenamiento se dé en pocos ciclos. Cuando el NCDR está entrenándose la información sobre la planta es requerida. Debido que la planta es normalmente desconocida, el NIDR es usado para estimar la dinámica de la misma para el NCDR.

La señal actual de control generada del NCDR y la salida previa de la planta son usadas como las entradas al NIDR. El error entre la salida del NIDR y la planta son computadas para cada iteración, y son usadas para ajustar los pesos del NIDR. Entrenando el NIDR y el NCDR alternativamente los pesos de la NCDR pueden ser ajustados más eficientemente.

5.3) Algoritmo de retropropagación dinámico para la RNDR

El modelo matemático para la RNDR de la figura 5.1 es el mostrado a continuación:

$$O(k) = \sum_j W_j^{\,O} X_j(k), \quad X_j(k) = f(S_j(k)) \tag{5.1}$$

$$S_j(k) = W_j^{\,D} X_j(k-1) + \sum_j W_{ij}^{\,I} I_i(k) \tag{5.2}$$

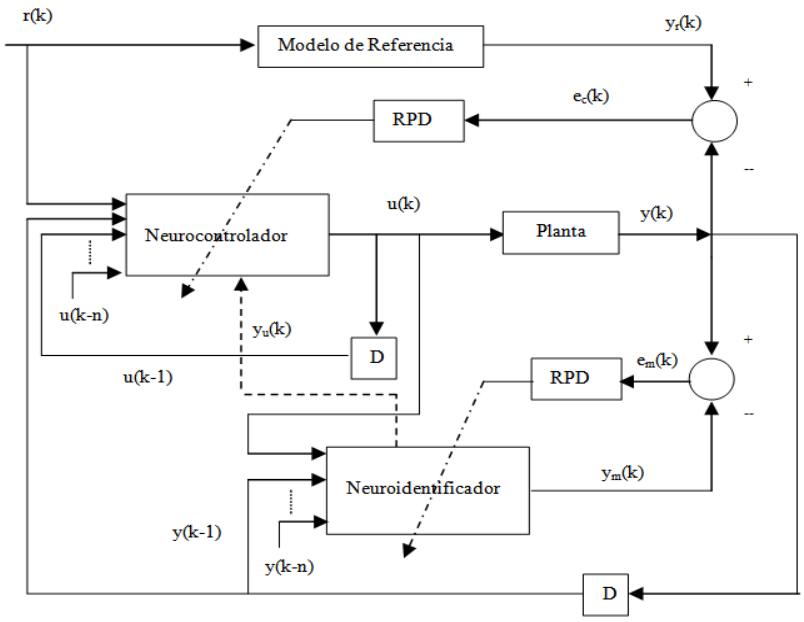


Figura 5.2 Diagrama de bloques del sistema de control basado en la RNDR.

Donde para cada tiempo discreto k, $\mathrm{I_i(k)}$ es la i-ésima entrada a la RNDR, $\mathrm{S_j(k)}$ es la suma de las entradas de la j-ésima neurona recurrente, $\mathrm{X_j(k)}$ es la salida de la j-ésima neurona recurrente, $O(k)$ es la salida de la RNDR. Aquí $f(\cdot)$ es la función sigmoide y W^I, W^D, W^O son entrada, recurrente y salida de los pesos de vectores respectivamente, en $\mathrm{R^{ni}}$, $\mathrm{R^{nd}}$ y $\mathrm{R^{no}}$.

Decir que $\mathrm{y_r(k)}$ y $\mathrm{y(k)}$ son las respuestas deseadas y verdaderas de la planta, el error está en función de los ciclos de entrenamiento para el NCDR y se puede definir como:

$$E_c = \frac{1}{2}(y_r(k) - y(k))^2 \tag{5.3}$$

En general, la respuesta de la planta es un mapeo $\mathrm{G(\cdot)}$ no lineal de la entrada $\mathrm{u(k)}$, ejemplo $\mathrm{y(k) = G(u(i), i \le k)}$. Aquí la entrada de la planta $\mathrm{u(k)}$ es la salida del NCDR, por ejemplo $\mathrm{u(k) = O(k)}$ en (5.1). Por otro lado, en el caso del NIDR, la entrada de la planta $\mathrm{u(k)}$ es la entrada al NIDR.

El error de la ecuación (5.3) es de la misma manera modificado para el NIDR remplazando $\mathrm{y_r(k)}$ y $\mathrm{y(k)}$ por $\mathrm{y(k)}$ y $\mathrm{y_m(k)}$ respectivamente, donde $\mathrm{y_m(k)}$ es la salida del NIDR.

$$E_m = \frac{1}{2}(y(k) - y_m(k))^2 \tag{5.4}$$

Donde $\mathrm{y_m(k) = O(k)}$ de (5.1).

El gradiente del error en (5.3) respecto a un vector de pesos $\mathrm{W} \epsilon \mathrm{R^n}$ arbitrario es representado por:

$$\frac{\partial E_c}{\partial W} = -e_c(k)\frac{\partial y(k)}{\partial W} = -e_c(k)y_u(k)\frac{\partial u(k)}{\partial W} = -e_c(k)y_u(k)\frac{\partial O(k)}{\partial W} \tag{5.5}$$

Donde $\mathrm{e_c(k) = y_r(k) - y(k)}$ es el error entre el valor deseado de la salida del modelo de referencia y la salida de la planta, y el factor $y_u(k) = \partial y(k)/\partial u(k)$ el cual representa la sensibilidad de la planta con respecto a esa entrada.

Ya que la planta es normalmente desconocida, la sensibilidad necesita ser estimada para el NCDR. Sin embargo, en el caso del NIDR, el gradiente del error en (5.4) está dado por:

$$\frac{\partial E_m}{\partial W} = -e_m(k)\frac{\partial y_m(k)}{\partial W} = -e_m(k)\frac{\partial O(k)}{\partial W} \tag{5.6}$$

Donde $\mathrm{e_m(k) = y(k) - y_m(k)}$ es el error entre la planta y la respuesta del NIDR.

La salida del gradiente $\partial O(k)/\partial W$ es común en (5.5) y (5.6) y necesita ser programado para ambos NCDR y NIDR. Esa programación se resume en el siguiente párrafo:

Dado la RNDR mostrado en la figura 5.1 y descrito por (5.1) y (5.2), la salida del gradiente con respecto a salida, recurrente, y entrada de pesos, respectivamente esta dado por:

$$\frac{\partial O(k)}{\partial W_j^{\,O}} = X_j(k) \tag{5.7a}$$

$$\frac{\partial O(k)}{\partial W_j^{\,D}} = W_j^{\,O} P_j(k) \tag{5.7b}$$

$$\frac{\partial O(k)}{\partial W_{ij}^{\,I}} = W_j^{\,O} Q_{ij}(k) \tag{5.7c}$$

Donde:

$P_j(k) = \partial X_j(k) / \partial W_j^{\,D}$ y $Q_{ij}(k) = \partial X_j(k) / \partial W_{ij}^{\,I}$

Satisfacen:

$$P_j(k) = f'(S_j)(X_j(k-1) + W_j^{\,D} P_j(k-1)), \quad P_j(0) = 0 \tag{5.8a}$$

$$Q_{ij}(k) = f'(S_j)(I_i(k) + W_j^{\,D} Q_{ij}(k-1)), \quad Q_{ij}(0) = 0 \tag{5.8b}$$

De (5.1), el gradiente con respecto a la salida de los pesos es encontrado como:

$$\frac{\partial O(k)}{\partial W_j^{\,O}} = X_j \tag{A.1}$$

De la misma manera de (5.1), el gradiente con respecto a los pesos recurrentes es:

$$\frac{\partial O(k)}{\partial W_j^{\,D}} = W_j^{\,O} \frac{\partial X_j(k)}{\partial W_j^{\,D}} = W_j^{\,O} P_j(k) \tag{A.2}$$

De (5.1) y (5.2),

$$\frac{\partial X_j(k)}{\partial W_j^{\,D}} = \frac{\partial X_j(k)}{\partial S_j(k)} \frac{\partial S_j(k)}{\partial W_j^{\,D}} = f'(S_j(k)) \frac{\partial S_j(k)}{\partial W_j^{\,D}} \tag{A.3}$$

Y

$$\frac{\partial S_j(k)}{\partial W_j^{\,D}} = X_j(k-1) + W_j^{\,D} \frac{\partial X_j(k-1)}{\partial W_j^{\,D}} \tag{A.4}$$

Que resulta en (5.8a).

El procedimiento para obtener el gradiente con respecto a los pesos de entrada es similar al de arriba, y las correspondientes ecuaciones (5.7c) y (5.8b).

Las ecuaciones (5.8a) y (5.8b) son ecuaciones con dinámicas recursivas no lineales para el estado de los gradientes $\partial X_j(k) / \partial W$ y pueden ser resueltas dando condiciones iníciales. Para las redes neuronales retroalimentadas (RNR), los pesos recurrentes W_j^D son cero y las ecuaciones se expresan algebraicamente.

5.4) Retropropagación dinámica para el NIDR

De (5.6) el gradiente negativo del error con respecto al vector de pesos en R^{n} es:

$$-\frac{\partial E_m}{\partial W} = e_m(k) \frac{\partial O(k)}{\partial W} \tag{5.9}$$

Donde la salida del gradiente está dada por (5.7) y (5.8), y W representa W^O, W^D, W^I en R^{no}, R^{nd} y R^{ni} respectivamente.

Los pesos pueden ahora ser ajustados siguiendo el método del gradiente, por ejemplo las reglas de actualización de los pesos empiezan de la siguiente manera:

$$W(n+1) = W(n) + h\left(-\frac{\partial E_m}{\partial W}\right) \tag{5.10}$$

Donde η es el parámetro de aprendizaje. Las ecuaciones (5.7)-(5.10) definen el algoritmo de retropropagación dinámico para el NIDR.

5.5) Retropropagación dinámica para el NCDR

En el caso del NCDR, de (5.5), el gradiente negativo del error con respecto al vector de pesos en R^n es:

$$-\frac{\partial E_c}{\partial W} = e_c(k)\, y_u(k)\, \frac{\partial O(k)}{\partial W} \tag{5.11}$$

Ya que la planta es normalmente desconocida, el término de sensibilidad $y_u(k)$ es desconocido. Este valor desconocido puede ser estimado usando el NIDR. Cuando el NIDR es entrenado, el comportamiento de la dinámica del NIDR está cerca de la planta desconocida $y(k) \approx y_m(k)$, donde $y_m(k)$ es la salida de la NIDR. Una vez hecho el proceso de entrenamiento, nosotros asumimos que la sensibilidad puede ser aproximada como:

$$y_u(k) = \frac{\partial y(k)}{\partial u(k)} \approx \frac{\partial y_m(k)}{\partial u(k)} \tag{5.12}$$

Donde $u(k)$ es la entrada del NIDR.

Aplicando la misma regla de (5.12) y notando que $y_m(k) = O(k)$ de (5.1),

$$\frac{\partial y_m(k)}{\partial u(k)} = \frac{\partial O(k)}{\partial u(k)} = \sum_j \frac{\partial O(k)}{\partial X_j(k)} \frac{\partial X_j(k)}{\partial u(k)} = \sum_j W_j^O \frac{\partial X_j(k)}{\partial u(k)} \tag{5.13}$$

También de (5.1),

$$\frac{\partial X_j(k)}{\partial u(k)} = f'(S_j(k)) \frac{\partial S_j(k)}{\partial u(k)} \tag{5.14}$$

Ya que las entradas del NIDR son $u(k)$ y $y(k-1)$ figura 5.2, entonces (5.2) favorece:

$$S_j(k) = W_j^D X_j(k-1) + W_{1j}^I u(k) + W_{2j}^I y(k-1) + W_{3j}^I b_I \tag{5.15}$$

Donde b_1 es la bias del NIDR.

Por lo tanto,

$$\frac{\partial S_j(k)}{\partial u(k)} = W_{1j}^I \tag{5.16}$$

De (5.13), (5.14) y (5.16),

$$y_u(k) = \frac{\partial y_m(k)}{\partial u(k)} = \sum_j W_j^O f'(S_j(k)) W_{1j}^I \tag{5.17}$$

Donde las variables y los pesos son encontrados en el NIDR.

Usando el negativo del gradiente en (5.11), los pesos para el NCDR ahora pueden ser ajustados usando las reglas de actualización similares a (5.10). Las ecuaciones (5.7), (5.8), (5.10), (5.11), y (5.17) definen el algoritmo de retropropagación dinámico de NCDR.

6) IMPLEMENTACIÓN DE LA RED NEURONAL RECURRENTE

En este capítulo se desarrolla la metodología de implementación del algoritmo de la red neuronal diagonal recurrente (RNDR) en el FPGA y DSP, de igual manera se hará una pequeña reseña sobre dichas tecnologías tomando en cuenta sus características y ventajas de cada una de ellas.

6.1) ¿Porque usar FPGA y/o DSP para sistemas de control con redes neuronales recurrentes?

Como bien se sabe el modelado matemático resultante de una red neuronal recurrente son una serie de ecuaciones expresadas en términos de entradas y valores anteriores de la salida, y dentro del procesamiento se requiere calcular algunas funciones no lineales de activación generalmente de índole exponencial. El número de ecuaciones de una red neuronal recurrente depende de la cantidad de neuronas en las diferentes capas de entrada, oculta y de salida que se desea implementar y así mismo se requiere de velocidad de aprendizaje y exactitud; por todas estas razones los FPGA y DSP son idóneos para llevar a cabo la implementación en tiempo real.

En estos sistemas de redes neuronales recurrentes el factor clave para poder llevar a cabo la implementación es la necesidad de tener en hardware sumadores y multiplicadores que en conjunto constituyen una arquitectura MAC (multiplicador-acumulador) y una velocidad de cálculo de procesamiento rápida (Omondi y Rajapakse, 2006).

La velocidad de cálculo para los sistemas de control basados en redes neuronales es un factor muy importante, es por ello que las tecnología FPGA y DSP ofrecen no sólo la velocidad de cálculo sino sistemas MAC desarrollados en hardware lo que los hace una herramienta idónea en el procesamiento digital, cosa que otras tecnologías no ofrecen como son algunos tipos de microcontroladores de uso común.

6.2) El FPGA y su metodología de implementación

Un FPGA (arreglos de compuertas programables en campo) es un circuito integrado que contiene componentes lógicos programables e interconexiones programables entre ellos. La raíz histórica de los FPGA son los dispositivos de lógica programable compleja (CPLD) de mediados de los ochenta. Los componentes lógicos programables pueden ser programados para duplicar la funcionalidad de compuertas lógicas básicas o funciones combinacionales más complejas tales como decodificadores o funciones matemáticas. En muchos FPGA estos componentes lógicos programables (o bloques lógicos, según el lenguaje comúnmente usado) también incluyen elementos de memoria, los cuales pueden ser simples flip-flops o bloques de memoria más complejos.

En un intento de reducir la complejidad y el tiempo de desarrollo en fases de prototipaje rápido, y para validar un diseño en HDL (Hardware Description Language), existen varias propuestas y niveles de abstracción del diseño (Sharp, 2005). Entre otras, National Instruments LabVIEW FPGA propone un acercamiento de programación gráfica de alto nivel, Agility DK Design Suite (Agility, 2008) es un ambiente de diseño ESL (Electronic System Level) completo para lenguaje C ANSI que provee el diseño de alto nivel usando el lenguaje C basadas en FPGA. System Generator y AccelDSP (Xilinx, 2008) herramientas industriales de alto nivel para diseñar sistemas DSP de alto rendimiento para FPGA’s. Estas herramientas suministran los conceptos abstractos que permiten desarrollar sistemas paralelos con los FPGAs más avanzados de la industria, proveyendo el modelado del sistema y código de generación automática a partir de Simulink de MATLAB.

Ventajas de un FPGA:

- Circuitos eléctricos de propósito específico
- Procesamiento paralelo
- Buses personalizados
- Procesamiento en tiempo real
- Mayor capacidad de integración, lo que permite realizar sistemas digitales más complejos en un único circuito integrado.

- Arquitectura flexible que se adapta mejor a cierto tipo de aplicaciones complejas, como el procesamiento de señal.

Aplicaciones FPGA:

- Procesamiento de imágenes
- Redes neuronales para reconocimiento de imágenes
- Compresión de video
- Sistemas de visión
- Redes inalámbricas
- Aplicaciones en comunicaciones
- Automatización y control

Para la realización de esta implementación se utilizo una tarjeta de desarrollo ML402 de Xilinx cuyo FPGA es un Virtex-4, véase figura 6.1.

Figura 6.1 Tarjeta FPGA Virtex 4 de Xilinx

Y para la descripción del esquema de control de la figura 5.2 se utilizo la herramienta de diseño HandelC de Agility DK Design Suite 5.0 (Agility, 2008):

El software de diseño de DK de Agility es un ambiente de diseño (ESL) Electronic System Level completo para lenguaje C ANSI cuyo nombre clave es Handel-C. Provee los beneficios de un sistema integral completo, el diseño de alto nivel usando el lenguaje C basadas en FPGA y en SoC. Esto incluye codiseño del sistema y coverificación, así mismo la capacidad de conversión de C a RTL (Register Transfer Level) y de C a la síntesis de FPGA, (ver apéndice A).

Los principales factores que deben considerarse al aplicar RNDR en hardware reconfigurable son la velocidad de cómputo, el uso de los recursos de hardware, el consumo de energía, etc (Marwedel, 2006). Para una aplicación particular, los requisitos específicos sobre estos factores dan la pauta de las necesidades que hay que satisfacer y la ejecución final es normalmente un compromiso entre todos estos factores.

Por lo tanto, un efectivo y eficiente entorno de prototipado rápido que permita la experimentación y verificación de las distintas configuraciones del algoritmo, así como la arquitectura y los sistemas de aplicación sería muy útil. Las herramientas que se utilizaron para lograr implementar la “RNDR en un chip” se muestran en el diagrama de flujo de la figura 6.2, entre ellas tenemos: la tarjeta FPGA ML402, DK Design Suite de Celoxica 5.0, Visual C++ 6.0 de Microsoft, Matlab 7.0 de Mathworks, Synplify Pro 9.4 y una tarjeta de propósito especifico DUPG (Dispositivo Usb de propósito general) diseñada para configurar y validar el esquema de control de la figura 5.2, a continuación se describen brevemente cada una de ellas y el papel que tuvieron en el desarrollo de la implementación.

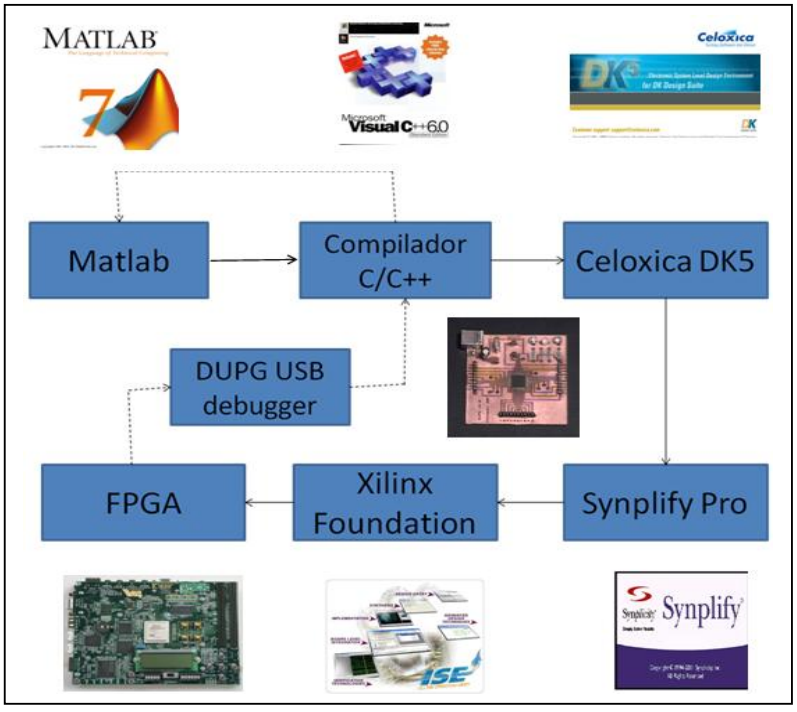


Figura 6.2 Metodología de Implementación para la RNDR

El núcleo de la ML402 es un FPGA XC4VSX35 de Xilinx con 15,360 Slices y algunas otras características en el chip como DSP48 y RAM distribuida. Agility DK Design Suite es un entorno integrado para la implementación en FPGA usando el lenguaje de programación Handel-C, posee una herramienta completa en la cual incluye el compilador, debugger, simulación y un optimizador. Matlab provee una excelente plataforma para el modelado de plantas y sistemas de control, los algoritmos de la RNDR y en general todo el esquema de control fue validado y simulado en esa plataforma.

Handel-C es un lenguaje de implementación de alto nivel para FPGA con sintaxis ANSI C y características particulares al hardware de desarrollo, también posee ejecución en paralelo, canales de comunicación, definiciones de interfaces, etc. Comparado con otros lenguajes de descripción de Hardware como VHDL o Verlog, Handel-C es más conveniente para el rápido desarrollo e implementación de algoritmos de control.

Visual C++, fue utilizado con el propósito de desglosar de una manera más tangible el código de matlab, puesto que la descripción de hardware para el FPGA se desarrollo completamente en lenguaje Handel-C, de esta manera la portabilidad entre ambos lenguajes es más factible y facilita el camino de descripción entre ambos al basarse en la propia sintaxis de ANSI C, por otro lado Visual C++ se utilizo también como interfaz de la tarjeta DUPG, para debuggear, configurar y validar el algoritmo de control implementado en el FPGA.

Synplify Pro, una excelente herramienta y de las mejores de sintetizado para FPGA’s del mercado, realizó una máxima optimización del esquema de control en el FPGA virtex 4 así como tiempos de retardos mucho menores que otras plataformas, de igual manera Xilinx Foundation es una plataforma de desarrollo que consiste en un conjunto integrado de herramientas software y hardware para crear, simular e implementar diseños digitales en FPGA.

La tarjeta DUPG (figura 6.3) es utilizada en conjunto con una interfaz de PC elaborada en visual C++ para configurar el esquema de control y depurarlo, su

núcleo se basa en un microcontrolador pic18f4550 de la empresa microchip, el cual posee un conjunto de periféricos para comunicación SPI, UART, USB, I^2C, etc., que lo hacen idóneo para la validación algoritmo implementado.

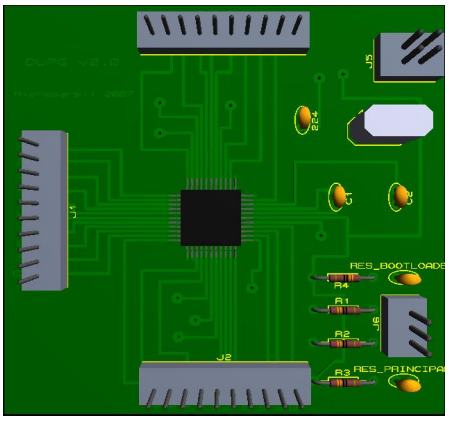


Figura 6.3a DUPG en Diseño 3D

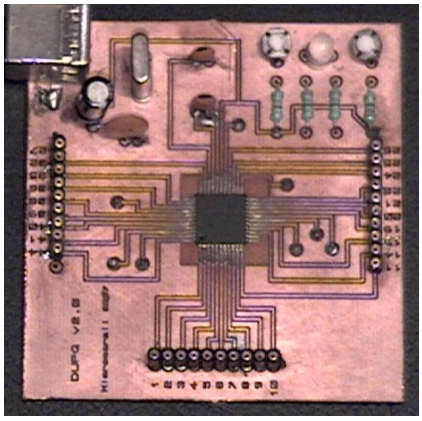


Figura 6.3b DUPG terminada

El esquema general de implementación embebida de la RNDR en un chip se muestra en la figura 6.4, ahí se puede observar que básicamente todo el esquema de control basado en la red neuronal recurrente se encuentra embebido en la tarjeta ML402, donde por medio de convertidores A/D y D/A de 12 bits de la empresa microchip (MCP3208 y MCP4922) se puede medir y controlar una planta analógica que en este caso fue un circuito eléctrico de muestra experimental. De igual forma se observa una tarjeta DUPG que se comunica con la ML402 por medio del puerto de comunicación SPI y sirve de interfaz hacia la PC por medio del puerto de comunicación USB.

En las siguientes líneas describiremos brevemente los principales bloques que se realizaron para llevarlo a cabo la implementación del algoritmo en el FPGA siguiendo el esquema metodológico de la figura 6.2, cabe mencionar que esta metodología no solo sirve para implementar algoritmos de control, sino que también cualquier sistema digital como filtros, interfaces, sistemas de adquisición de datos o algún procesamiento digital de señales, todo ello en tiempos relativamente cortos comparados con las metodologías que generalmente se imparte en los cursos de sistemas digitales con implementaciones en FPGA.

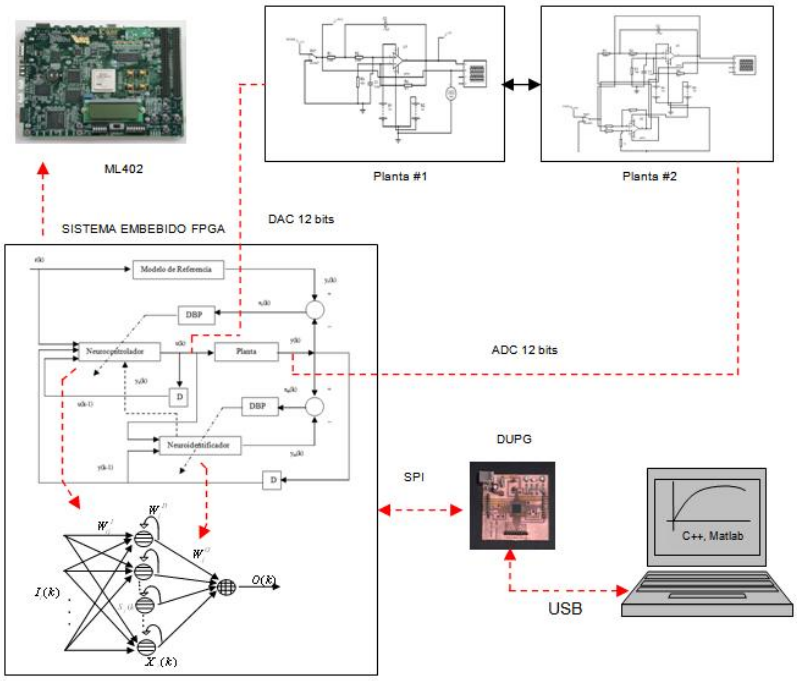


Figura 6.4 Esquema global para el sistema de control embebido en FPGA.

Paso 1: Programación en MATLAB

Todo el sistema de control basado en la RNDR fue programado, simulado, validado y optimizado en el lenguaje de programación matlab desglosando cada uno de los siguientes bloques:

- Modelo del neuroidentificador
- Modelo del neurocontrolador
- Entrenamiento de retropropagación dinámico para el neuroidentificador
- Entrenamiento de retropropagación dinámico para el neurocontrolador

Paso 2: Programación en Visual C++

Todo el código programado en matlab fue portado a visual c++ y verificado numéricamente para obtener un código más compatible a lo que es el lenguaje de descripción de hardware Handel-C, de esta manera la implementación se hace más rápida y confiable. Comentar también que el uso de la función de activación

sigmoide basada en la función exponencial que se empleo en el algoritmo de control fue desarrollada y programada hasta el doceavo término siguiendo la siguiente estructura de Horner mostrada a continuación:

$$\exp(x) = ((((((((((a_{12}x)+a_{11})x+a_{10})x+a_9)x+a_8)x+a_7)x+a_6)x+a_5)x+a_4)x+a_3)x+a_2)x+a_1)x+a_0$$

Teniendo así 24 cálculos de operaciones: 12 multiplicaciones y 12 sumas.

Paso 3: Descripción del algoritmo en Handel-C

Aunque una implementación en FPGA es altamente dependiente de la aplicación, existen algunos componentes básicos comunes. La primera, es una eficiente librería de punto flotante (disponible en la suite de diseño de DK[5] de Agility) que se utilizó para la descripción del esquema de control, esta librería está basada en el estándar 754 de la IEEE (Institute of Electrical and Electronics Engineers), segundo se implementaron subprocesos en forma paralela como lo son las ecuaciones que rigen la dinámica de aprendizaje del neuroidentificador y del neurocontrolador así como los bloques de interfaz SPI (Serial Peripheral Interface) con la tarjeta DUPG y el manejo de los convertidores digital-analógico y analógico-digital de 12 bits. La compilación nos genera los archivos VHDL que posteriormente a través de synplify pro hará la síntesis, de esta manera se han obtenido mejores resultados en el performance del FPGA Virtex-4 (ver apéndice C).

Paso 4: Sintetizado en Synplify Pro

Para realizar la síntesis fue utilizado Synplify Pro, ya que con esta herramienta logramos obtener mejor aprovechamiento del espacio del FPGA. La salida de Synplify Pro, un archivo de tipo EDIF (Electronic Design Interchange Format), que puede ser usado con ISE (Integrated Software Environment) de Xilinx para obtener el archivo de configuración necesario para programar el FPGA.

Paso 5: Implementación del diseño en ISE Xilinx Foundation

El primer paso es la traducción del fichero de diseño EDIF en un formato adecuado NGD (Native Generic Database), seguido por un proceso de mapeo del diseño al dispositivo destino especifico. Luego nos encontramos con la colocación y ruteado. Después viene la generación de información de temporización para que la use el simulador temporizado. El paso final es la generación del bitstream. Podemos seguir el estado de cada paso en la ventana de flujo de síntesis ver figura 6.5.

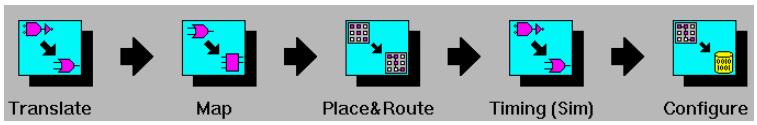


Figura 6.5 Flujo de síntesis.

Los recursos del FPGA utilizados para el esquema RNDR con aritmética de punto flotante según el estándar IEEE 754 se muestran en el cuadro 6.1.

Cuadro 6.1 Recursos utilizados en el FPGA

Slice	LUT	FF	DSP48	System Clock
9,727(63%)	12,721(41%)	9,514(30%)	4	100 Mhz

Paso 6: Validación del esquema de control

De acuerdo a la figura 6.4, la forma en que se valido el esquema de control antes de ponerlo en marcha es de la siguiente manera:

Proponemos una planta cuyo modelo cambia después de alcanzar el estado estacionario, eso es tener básicamente dos modelos de plantas diferentes en sus respuestas, después simulamos todo el esquema de control figura 5.2 en matlab y obtenemos los resultados en forma numérica y gráfica tanto de la

respuesta de la planta como de la señal de control. Ahora bien se implementa todo el algoritmo desarrollado en matlab en el FPGA siguiendo la metodología de la figura 6.2 y guardamos internamente en la memoria del FPGA 600 muestras de la respuesta de la planta y la señal de control, después de ello las extraemos con la tarjeta DUPG y las comparamos numéricamente y gráficamente con las que se simularon en matlab, si las respuestas son iguales quiere decir que la implementación del esquema de control basado en la RNDR trabaja adecuadamente en el chip, lo siguiente a proceder sería simplemente acondicionar nuestra planta a controlar (los convertidores A/D y D/A) y poner la aplicación en marcha teniendo la seguridad de que el esquema de control y en general la RNDR trabajan adecuadamente.

6.3) El DSP y su metodología de implementación

Un DSP (Procesador Digital de Señales) es un sistema basado en un procesador o microprocesador que posee un juego de instrucciones, un hardware y un software optimizados para aplicaciones que requieran operaciones numéricas a muy alta velocidad. Debido a esto es especialmente útil para el procesado y representación de señales analógicas en tiempo real: en un sistema que trabaje de esta forma (tiempo real) se reciben muestras, normalmente provenientes de un conversor analógico/digital (ADC). Se ha dicho que puede trabajar con señales analógicas, pero es un sistema digital, por lo tanto necesitará un conversor analógico/digital a su entrada y digital/analógico en la salida.

Como todo sistema basado en procesador programable necesita una memoria donde almacenar los datos con los que trabajará y el programa que ejecuta. Si se tiene en cuenta que un DSP puede trabajar con varios datos en paralelo y un diseño e instrucciones específicas para el procesado digital, se puede dar una idea de su enorme potencia para este tipo de aplicaciones. Estas características constituyen la principal diferencia de un DSP y otros tipos de procesadores.

Controlador Digital de Señal dsPIC30F de 16 bits con la potencia de un DSP

¿Qué es un controlador digital de señal (DSC)?

Un controlador digital de señal (DSC) es un controlador embedded single-chip que integra de manera compacta las capacidades de control de un microcontrolador (MCU) con las capacidades de computación y rendimiento de un procesador digital de señal (DSP).

El controlador digital de señal dsPIC30F de Microchip ofrece todo lo que se puede esperar de un poderoso MCU de 16-bit: gestión de interrupciones rápida, flexible y sofisticada; un amplio array de periféricos analógicos y digitales; gestión del consumo; opciones de reloj flexibles; power-on-reset; Brown-out; watchdog; seguridad en código, emulación en tiempo real a plena velocidad; y soluciones de depuración en circuito a plena velocidad.

Añadiendo con destreza la capacidad de un DSP a un poderoso microcontrolador de 16-bit, el controlador digital de señal dsPIC30F de Microchip consigue lo mejor de ambos mundos y marca el comienzo de una nueva era en el control embedded.

Potente MCU de 16-bit

El dsPIC30F ejecuta la mayor parte de sus instrucciones en un solo ciclo (33ns a 30 MIPS). Combinando este alto rendimiento con auténticas capacidades de DSP multiplicación de 16-bit en un único ciclo, se consigue tener el MCU de 16-bit más poderoso del momento. La mayoría de los usuarios de MCUs que buscan añadir características de DSP a su sistema se encuentran que añadir un chip DSP al ya existente sistema basado en microcontrolador puede ser realmente costoso y complicado. El DSC dsPIC30F está diseñado para parecer y sentir como un MCU. Añadir funcionalidad DSP en el familiar entorno basado en microcontrolador puede ser ya llevado a cabo con facilidad. (ver apéndice B).

La arquitectura dsPIC30F fue desarrollada en colaboración con el equipo de desarrollo del compilador C30. El resultado es un código en C de alta eficiencia si se compara a cualquier MCU o DSP de 16-bit. Las referencias en código C muestran que los MCUs de 16-bit de la competencia requieren un 70% más de espacio de código de programa para el mismo programa de aplicación escrito en C.

Para el desarrollo de este trabajo se decidió utilizar los DSP de Microchip por las siguientes razones:

a) El costo que ofrecen por su gran desempeño y las aplicaciones hacia donde van enfocadas hacen que sean muy accesibles para el desarrollo.
b) El compilador que ofrece microchip C30 es gratuito.
c) Existe buena información acerca de su uso para sistemas de control.
d) Son fáciles de adquirir y su tipo de encapsulado hace que su desarrollo en PCB se accesible.

Ventajas de un DSP ó DSC:

- Eficiencia en el desempeño comparada con otros tipos de procesadores, debido a su condición de uso particular.
- Minimización en costo y riesgo para el desarrollo de aplicaciones, consecuencia de una tecnología con extensa experiencia en sus bases de diseño.
- Soporte para ejecutar aplicaciones con gran cantidad de operaciones de multiplicación y acumulación (MAC) por segundo.
- Posibilidad para ejecutar manipulación aritmética compleja y de punto flotante.
- Entornos de programación más amigables (C/C++), para desarrollo de aplicaciones en procesamiento de señales.
- Disponibilidad de registros de algoritmos desarrollados a nivel bibliotecas.

Aplicaciones DSP:

- Wireless LAN
- Reconocimiento de voz
- Manejo de imágenes digitales
- Reproductores digitales de audio
- Teléfonos celulares
- Módems inalámbricos
- Cámaras digitales
- Control de motores
- Automatización
- Etc.

El diseño de la tarjeta está basado principalmente en un dspic30f6014A véase figura 6.6, la interface de programación es ICSP (In Circuit Serial Programming) que permitirá reprogramarlo en línea a través de un programador USB diseñado para este fin véase figura 6.7, 6.8.

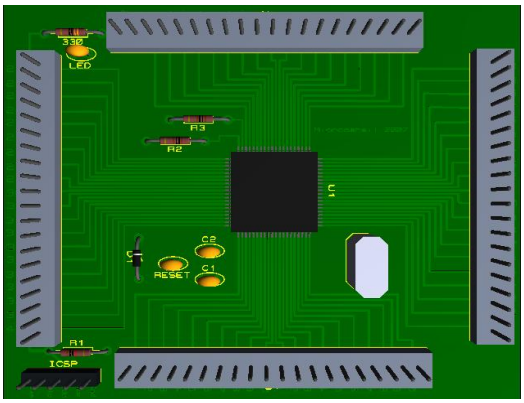


Figura 6.6a DSC en Diseño 3D

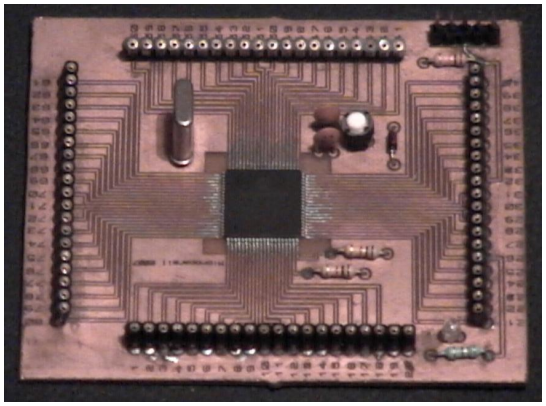


Figura 6.6b DSC terminada

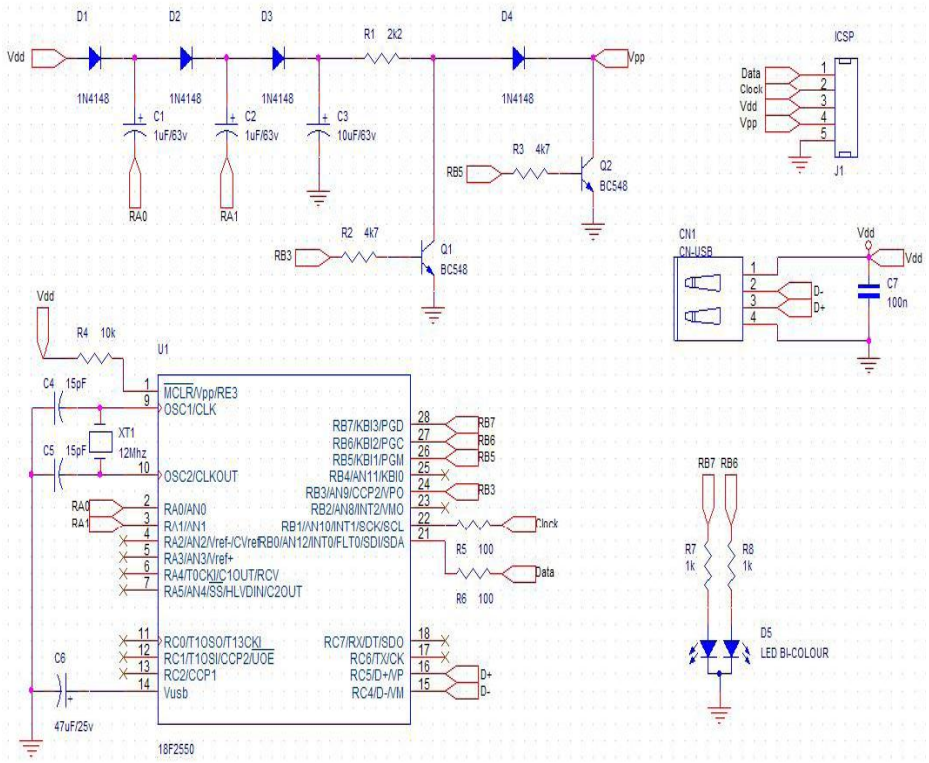


Figura 6.7 Esquemático del programador GTP USB

Figura 6.8 Programador USB para el dsPIC30F6014A

A continuación se presentará la metodología que se llevo a cabo para la implementación del esquema de control en el DSC el cual se ilustra en la figura 6.9. Algo particular que se puede apreciar es la similitud con la metodología del FPGA que se explicó en el apartado 6.2). Sin embargo difieren mucho en cuanto a las herramientas y en la implementación por ser arquitecturas completamente diferentes.

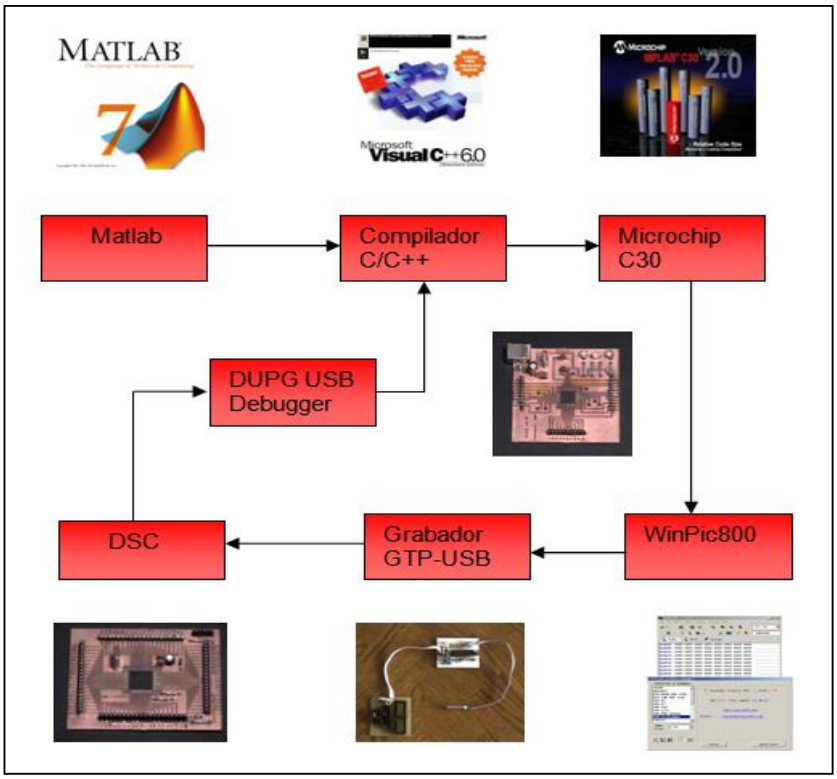


Figura 6.9 Metodología de Implementación para el DSC

Paso 1: Programación en MATLAB

La programación del esquema de control basado en RNDR fue el mismo que para la implementación en el FPGA de la sección 6.2), es decir fueron las mismas ecuaciones del NIDR, NCDR y el algoritmo de RPD para ambos.

Paso 2: Programación en Visual C++

De igual manera la programación en visual c++ fue casi idéntica a la que se desarrollo para la implementación en el FPGA de la sección 6.2), salvo por un aspecto el cual es que la función exponencial no se implemento hasta el doceavo termino en la estructura de Horner, si no que se utilizo la librería math.h que ya posee la función exponencial en punto flotante en el compilador, se hizo de esta manera porque el compilador de Microchip C30 que definiremos en el siguiente

paso también posee esa librería de punto flotante. La demás programación fue exactamente igual.

Paso 3: Programación en el compilador C30 de Microchip

Es un compilador de lenguaje ANSI C para microcontroladores de arquitectura DSP de 16 bits, que permite una rápida implementación con buenos niveles de optimización. Dentro de las características del compilador es la facilidad de utilizar puntos flotantes de 32 bits que corresponden a la especificación 754 IEEE de tal manera que la programación orientada a algoritmos complejos se hace relativamente menos trabajosa a la hora de la implementación. La librería matemática donde se encuentra la función exponencial sigue el estándar IEEE 754 coma flotante y doble precisión y al basarse los compiladores Visual C++ y C30 en ANSI C, la portabilidad entre ambos es muy rápida permitiendo acelerar el proceso de diseño (ver apéndice D). En la figura 6.10 se aprecia el tamaño de programa de memoria que ocupo el esquema de control dentro del DSC.

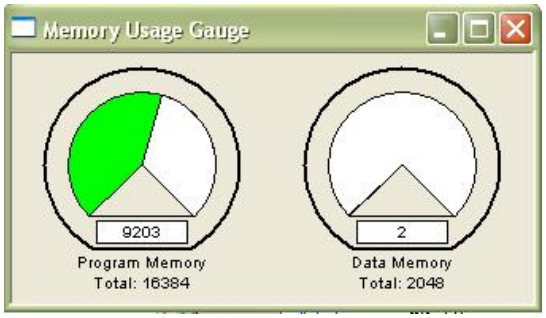


Figura 6.10 Cantidad de Bytes de memoria ocupado en el DSC

Paso 4: WinPic800

Una vez que se haya programado y compilado todo el código en C30, se nos proporcionará un archivo hexadecimal necesario para poderse grabar dentro de la memoria del DSC; ahora bien para poder grabar ese archivo hexadecimal necesitamos el software WinPic800 junto con el programador Gtp Usb anteriormente visto; en dicho programa se encuentran opciones de detección del

programador así como también del DSC a programar, una vez hecho lo anterior se procede a cargar el archivo hexadecimal y a grabarlo, al final el propio software indicara si la grabación fue exitosa o si hubo algún error en alguna parte del archivo. Para asegurarse de que el firmware es el correcto se puede hacer una verificación interna de la memoria del DSC contra el archivo hexadecimal original.

Paso 5: Grabador GTP USB

El GTP USB es un grabador ICSP para DSC, PIC's y memorias que hace uso del puerto USB. Permite grabar todos los PICs de la serie F Y DSC que soporten el modo de grabación ICSP (In-Circuit Serial Programming) y que estén incluidos en el WinPIC800. Al ser un grabador ICSP no tiene placa de zócalos, esto se ha hecho con la intención de que el diseño fuera compacto. Para grabar un DSC se deben conectar cada una de las líneas de esa salida ICSP a los pines correspondientes en el DSC a grabar. En la figura 6.7 se muestra el esquemático del programador, la situación de componentes en la placa y en la figura 6.8 unas imágenes del programador terminado.

Ventajas de usar el programador GTP USB:
1.- Económico de construir
2.- Rápida programación gracias al puerto Usb 2.0
3.- Pequeño y versátil.
4.- No necesita alimentación externa.
5.- Programa de la mayoría de DSC, Pic's de la serie 16F,18F y memorias.
6.- Compatible con todos los sistemas operativos Windows.

Paso 6: Validación del esquema de control

La validación del esquema de control en el DSC sigue exactamente la misma metodología de validación vista en el paso 6 de la implementación en el FPGA.

De acuerdo a la figura 6.11, la forma en que se valido el esquema de control antes de ponerlo en marcha es de la siguiente manera:

Proponemos una planta cuyo modelo cambia después de alcanzar el estado estacionario, eso es tener básicamente dos modelos de plantas diferentes en sus respuestas, después simulamos todo el esquema de control figura 5.2 en matlab y obtenemos los resultados en forma numérica y gráfica tanto de la respuesta de la planta como de la señal de control. Ahora bien se implementa todo el algoritmo desarrollado en matlab en el DSC siguiendo la metodología de la figura 6.9 y guardamos internamente en la memoria del DSC 600 muestras de la respuesta de la planta y la señal de control, después de ello las extraemos con la tarjeta DUPG y las comparamos numéricamente y gráficamente con las que se simularon en matlab, si las respuestas son iguales quiere decir que la implementación del esquema de control basado en la RNDR trabaja adecuadamente en el chip, lo siguiente a proceder sería simplemente acondicionar nuestra planta a controlar (los convertidores A/D y D/A) y poner la aplicación en marcha teniendo la seguridad de que el esquema de control y en general la RNDR trabajan adecuadamente.

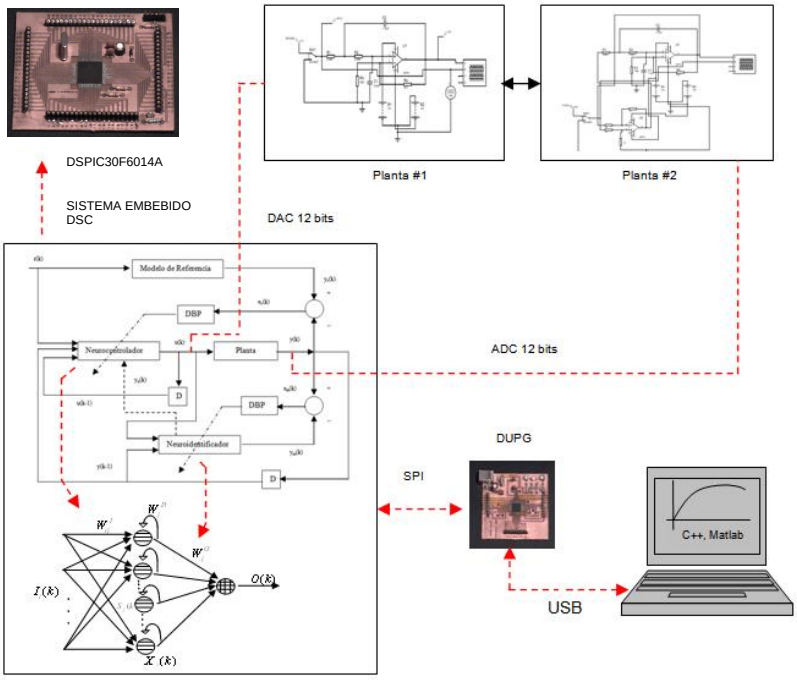


Figura 6.11 Esquema global para el sistema de control embebido en DSC.

7) RESULTADOS Y CONCLUSIONES

RESULTADOS

Aunque el algoritmo descrito en el capítulo 5) ha sido puesto a prueba en muchos ejemplos, aquí mostraremos los resultados obtenidos en simulación e implementación de un motor de CD y un circuito eléctrico cuyos modelos cambian después de alcanzar el estado estacionario.

Antes de mostrar los resultados se presenta las ecuaciones del NCDR y del NIDR del algoritmo descrito en el capítulo 5) las cuales se utilizaron de manera general en las simulaciones e implementación en el FPGA y DSC.

Parte del NCDR

```
j=1;
   Wc11_I=Wc11_I_1+Eta_C*(ec*Yu*Wc1_O*Qc11);
   Wc21_I=Wc21_I_1+Eta_C*(ec*Yu*Wc1_O*Qc21);
   Wc31_I=Wc31_I_1+Eta_C*(ec*Yu*Wc1_O*Qc31);
   Qc11=dFSc1*(r+Wc1_D*Qc11_1);
   Qc21=dFSc1*(U_1+Wc1_D*Qc21_1);
   Qc31=dFSc1*(Y_1+Wc1_D*Qc31_1);
j=2;
   Wc12_I=Wc12_I_1+Eta_C*(ec*Yu*Wc2_O*Qc12);
   Wc22_I=Wc22_I_1+Eta_C*(ec*Yu*Wc2_O*Qc22);
   Wc32_I=Wc32_I_1+Eta_C*(ec*Yu*Wc2_O*Qc32);
   Qc12=dFSc2*(r+Wc2_D*Qc12_1);
   Qc22=dFSc2*(U_1+Wc2_D*Qc22_1);
   Qc32=dFSc2*(Y_1+Wc2_D*Qc32_1);
j=3;
   Wc13_I=Wc13_I_1+Eta_C*(ec*Yu*Wc3_O*Qc13);
   Wc23_I=Wc23_I_1+Eta_C*(ec*Yu*Wc3_O*Qc23);
   Wc33_I=Wc33_I_1+Eta_C*(ec*Yu*Wc3_O*Qc33);
   Qc13=dFSc3*(r+Wc3_D*Qc13_1);
   Qc23=dFSc3*(U_1+Wc3_D*Qc23_1);
   Qc33=dFSc3*(Y_1+Wc3_D*Qc33_1);

j=1;
   Wc1_D=Wc1_D_1+Eta_C*(ec*Yu*Wc1_O*Pc1);
   Pc1=dFSc1*(Xc1_1+Wc1_D*Pc1_1);
j=2;
   Wc2_D=Wc2_D_1+Eta_C*(ec*Yu*Wc2_O*Pc2);
   Pc2=dFSc2*(Xc2_1+Wc2_D*Pc2_1);
j=3;
   Wc3_D=Wc3_D_1+Eta_C*(ec*Yu*Wc3_O*Pc3);
   Pc3=dFSc3*(Xc3_1+Wc3_D*Pc3_1);

j=1
   Sc1=Wc1_D*Xc1_1+(Wc11_I*r)+(Wc21_I*U_1)+(Wc31_I*Y_1);
   Xc1=1/(1+exp(-Sc1));
   dFSc1=exp(-Sc1)/((exp(-Sc1)+1)^2);
j=2
   Sc2=Wc2_D*Xc2_1+(Wc12_I*r)+(Wc22_I*U_1)+(Wc32_I*Y_1);
   Xc2=1/(1+exp(-Sc2));
   dFSc2=exp(-Sc2)/((exp(-Sc2)+1)^2);
j=3
   Sc3=Wc3_D*Xc3_1+(Wc13_I*r)+(Wc23_I*U_1)+(Wc33_I*Y_1);
```

```
   Xc3=1/(1+exp(-Sc3));
   dFSc3=exp(-Sc3)/((exp(-Sc3)+1)^2);


j=1;
   Wc1_O=Wc1_O_1+Eta_C*(ec*Yu*Xc1);
j=2;
   Wc2_O=Wc2_O_1+Eta_C*(ec*Yu*Xc2);
j=3;
   Wc3_O=Wc3_O_1+Eta_C*(ec*Yu*Xc3);

U=Wc1_O*Xc1+Wc2_O*Xc2+Wc3_O*Xc3;
```

Parte del NIDR

```
j=1;
   Wi11_I=Wi11_I_1+Eta_I*(em*Wi1_O*Qi11);
   Wi21_I=Wi21_I_1+Eta_I*(em*Wi1_O*Qi21);
   Wi31_I=Wi31_I_1+Eta_I*(em*Wi1_O*Qi31);
   Qi11=dFSi1*(U+Wi1_D*Qi11_1);
   Qi21=dFSi1*(Y_1+Wi1_D*Qi21_1);
   Qi31=dFSi1*(bias+Wi1_D*Qi31_1);
j=2;
   Wi12_I=Wi12_I_1+Eta_I*(em*Wi2_O*Qi12);
   Wi22_I=Wi22_I_1+Eta_I*(em*Wi2_O*Qi22);
   Wi32_I=Wi32_I_1+Eta_I*(em*Wi2_O*Qi32);
   Qi12=dFSi2*(U+Wi2_D*Qi12_1);
   Qi22=dFSi2*(Y_1+Wi2_D*Qi22_1);
   Qi32=dFSi2*(bias+Wi2_D*Qi32_1);
j=3;
   Wi13_I=Wi13_I_1+Eta_I*(em*Wi3_O*Qi13);
   Wi23_I=Wi23_I_1+Eta_I*(em*Wi3_O*Qi23);
   Wi33_I=Wi33_I_1+Eta_I*(em*Wi3_O*Qi33);
   Qi13=dFSi3*(U+Wi3_D*Qi13_1);
   Qi23=dFSi3*(Y_1+Wi3_D*Qi23_1);
   Qi33=dFSi3*(bias+Wi3_D*Qi33_1);

j=1;
   Wi1_D=Wi1_D_1+Eta_I*(em*Wi1_O*Pi1);
   Pi1=dFSi1*(Xi1_1+Wi1_D*Pi1_1);
j=2;
   Wi2_D=Wi2_D_1+Eta_I*(em*Wi2_O*Pi2);
   Pi2=dFSi2*(Xi2_1+Wi2_D*Pi2_1);
j=3;
   Wi3_D=Wi3_D_1+Eta_I*(em*Wi3_O*Pi3);
   Pi3=dFSi3*(Xi3_1+Wi3_D*Pi3_1);

j=1
   Si1=Wi1_D*Xi1_1+(Wi11_I*U)+(Wi21_I*Y_1)+(Wi31_I*bias);
   Xi1=1/(1+exp(-Si1));
   dFSi1=exp(-Si1)/((exp(-Si1)+1)^2);
j=2
   Si2=Wi2_D*Xi2_1+(Wi12_I*U)+(Wi22_I*Y_1)+(Wi32_I*bias);
   Xi2=1/(1+exp(-Si2));
   dFSi2=exp(-Si2)/((exp(-Si2)+1)^2);
j=3
   Si3=Wi3_D*Xi3_1+(Wi13_I*U)+(Wi23_I*Y_1)+(Wi33_I*bias);
   Xi3=1/(1+exp(-Si3));
   dFSi3=exp(-Si3)/((exp(-Si3)+1)^2);

j=1;
   Wi1_O=Wi1_O_1+Eta_I*(em*Xi1);
j=2;
   Wi2_O=Wi2_O_1+Eta_I*(em*Xi2);
j=3;
   Wi3_O=Wi3_O_1+Eta_I*(em*Xi3);

Yu=(Wi1_O*dFSi1*Wi11_I)+(Wi2_O*dFSi2*Wi12_I)+(Wi3_O*dFSi3*Wi13_I)+0.15;
Ym=Wi1_O*Xi1+Wi2_O*Xi2+Wi3_O*Xi3
```

Un caso simulado corresponde al siguiente modelo de un motor de CD cuya función de transferencia del voltaje de armadura como entrada y la posición angular como salida es:

$$\frac{q(s)}{V_a(s)} = \frac{22}{s(s+11)} \tag{7.1}$$

En la figura 7.1 se muestra la respuesta de la ecuación (7.1) en lazo cerrado ante una entrada escalón.

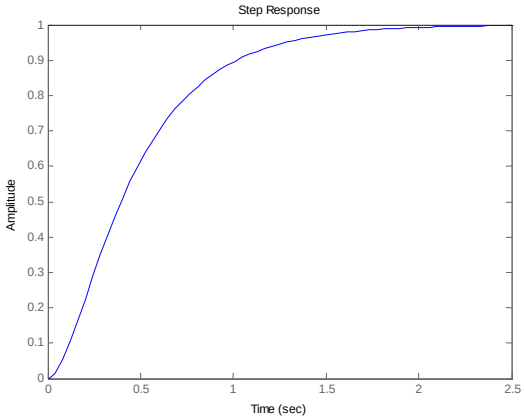


Figura 7.1 Respuesta del motor CD ec. (7.1).

Discretizando la planta anterior con un periodo de muestreo de T = 0.1 se obtiene:

$$Y(k) = [-0.3329 * Y(k-2)] + [1.333 * Y(k-1)] + [0.05472 * U(k-2)] + [0.0787 * U(k-1)] \tag{7.2}$$

La elección del periodo de muestreo se escoge prácticamente como $T \approx \frac{1}{10} t_r$ donde t_r es el tiempo de subida de la planta, para este caso $T \approx \frac{1}{10}(1) = 0.1s$

Después de que se alcanza la referencia se hace un cambio en la constante de tiempo del motor (el motor tiene carga) obteniendo un nuevo modelo como el siguiente:

$$\frac{q(s)}{V_a(s)} = \frac{25}{s(s+10)} \tag{7.3}$$

En la figura 7.2 se muestra la respuesta del modelo de la ecuación (7.3) en lazo cerrado ante una entrada escalón.

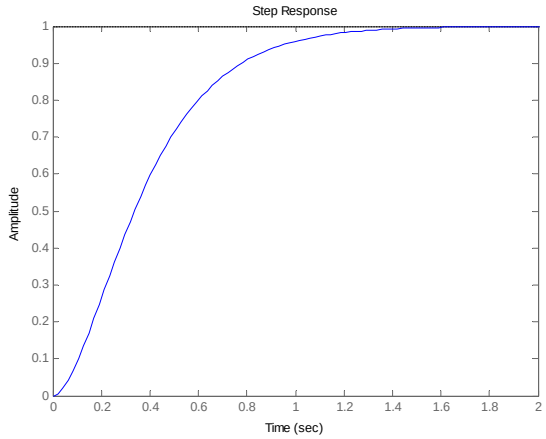


Figura 7.2 Respuesta del motor CD ec. (7.3).

Discretizando la planta ecuación (7.3) con T=0.1 se obtiene lo siguiente:

$$Y(k)=[-0.3679*Y(k-2)]+[1.368*Y(k-1)]+[0.0660*U(k-2)]+[0.0919*U(k-1)] \quad (7.4)$$

Como condiciones iníciales en la programación de la red neuronal recurrente tenemos:

$$\eta_i = 0.15,\ \eta_c = 0.15$$

Donde η_i, η_c son las constantes de aprendizaje del neuroidentificador y neurocontrolador.

Los esquemas NCDR y NIDR se muestran en la figura 7.3 respectivamente.

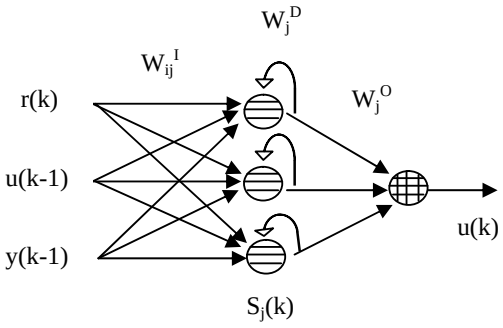


Figura 7.3a Esquema del NCDR

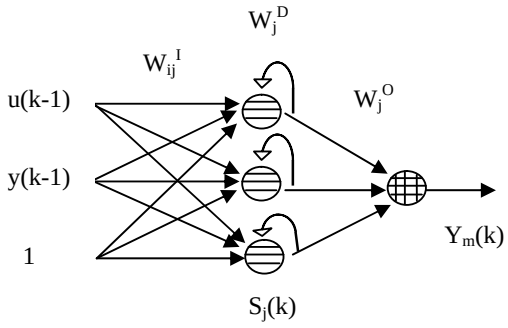


Figura 7.3b Esquema del NIDR

Simulando el algoritmo de control basado en la RNDR al modelo del motor CD cambiante en el tiempo, los resultados se ilustran en la figura 7.4. Como se puede observar el cambio de modelo no hace ningún efecto sobre la posición aún cuando el motor tiene carga.

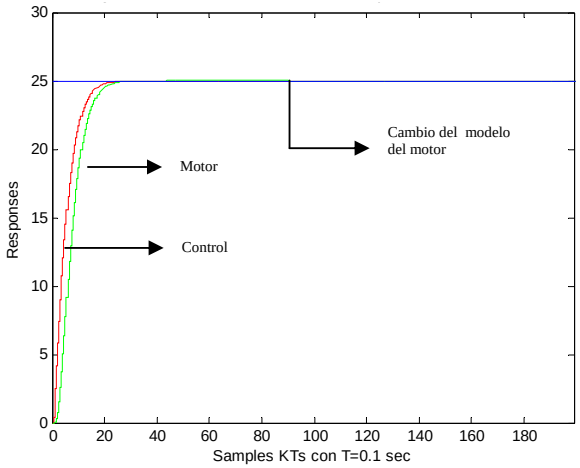


Figura 7.4 Respuesta del motor (posición) aplicando la RNDR

Ahora se presenta un modelo de un motor de CD cuya función de transferencial es la de la ecuación (7.5), siendo la entrada el voltaje de armadura (V_a) en Volts y como salida la velocidad angular (W) en rad/seg.

$$\frac{W(s)}{V_a(s)} = \frac{20.157}{s^2 + 4.244s + 14.342} \tag{7.5}$$

En la figura 7.5 se muestra la respuesta del motor (con carga en la flecha) con una entrada tipo escalón de 25 volts de amplitud; de igual forma se obtendrá otro nuevo modelo pero del motor pero ahora sin carga cuya función de transferencia se aprecia (7.6) y su respuesta ante una entrada tipo escalón de 25 volts en la figura 7.6.

$$\frac{W(s)}{V_a(s)} = \frac{20.16}{s^2 + 4s + 20} \tag{7.6}$$

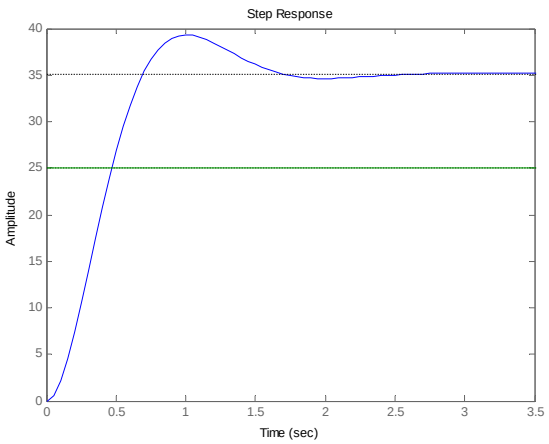


Figura 7.5 Respuesta del motor ec. (7.5).

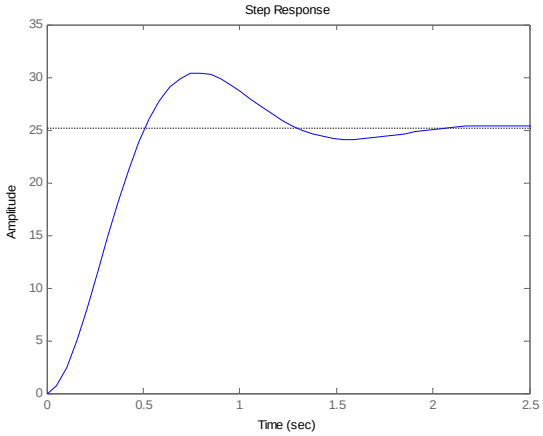


Figura 7.6 Respuesta del motor ec. (7.6).

En la figura 7.7 se aprecian 2 aspectos importantes cuando se aplica el esquema RNDR:

1) Que pese a tener una ganancia de k= 1.4 en estado estacionario y un estado transitorio no deseado el modelo (7.5), es corregido a través del esquema de control basado en la RNDR.

2) Después de que el modelo (7.5) se encuentra en referencia 25 volts se hace el cambio al modelo (7.6), como se aprecia ahí un decremento en la velocidad del motor pero es corregido inmediatamente por la acción del control y puesto nuevamente en referencia.

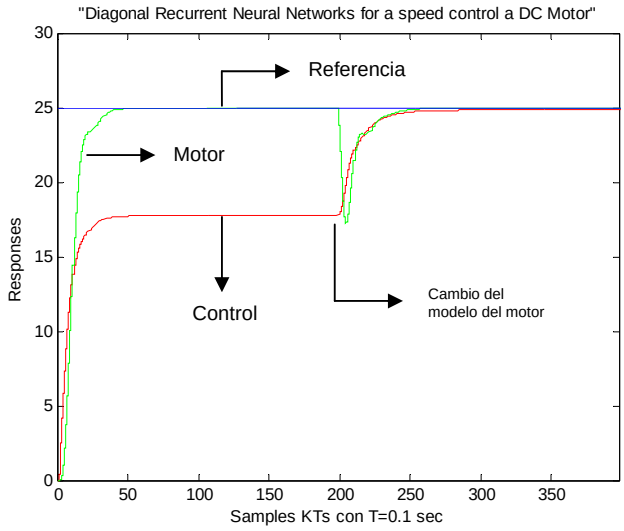


Figura 7.7 Respuesta del motor (velocidad) aplicando la RNDR

Los resultados siguientes muestran el desempeño del algoritmo de control tomando como modelo de seguimiento Y_r de primer y segundo orden respectivamente, como se aprecia en las figuras 7.8 y 7.9 la planta sigue el modelo dado con un retardo, esto debido al aprendizaje de la red neuronal para el NIDR que captura la dinámica del motor de CD.

$$\frac{W(z)}{V_a(z)} = \frac{0.2}{z - 0.8} \tag{7.7}$$

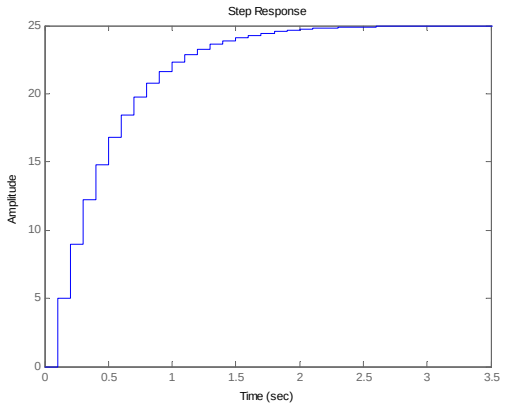


Figura 7.8a Respuesta del modelo ec. (7.7)

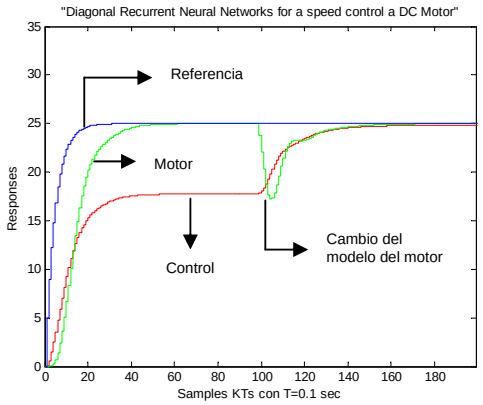


Figura 7.8b Respuesta del motor en la RNDR para el modelo de 1er orden

$$\frac{W(z)}{V_a(z)} = \frac{0.01875z + 0.01763}{z^2 - 1.794z + 0.8303} \tag{7.8}$$

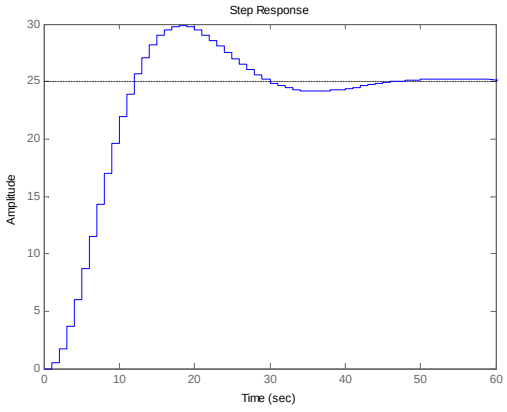


Figura 7.9a Respuesta del modelo ec. (7.8)

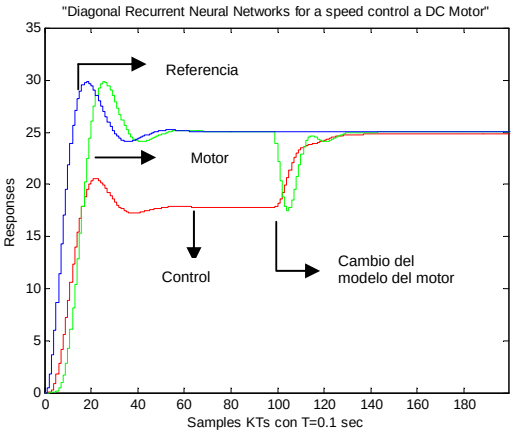


Figura 7.9b Respuesta del motor en la RNDR para el modelo de 2do orden

Aquí se presenta la simulación, validación e implementación de una planta la cual es un circuito eléctrico (figura 7.10) cuyo modelo cambia después de alcanzar el estado estacionario, la simulación se desarrollo programando el esquema de control basado en la RNDR y el modelo del circuito eléctrico en matlab, después de ello se implemento todo lo anterior en el FPGA y también de manera paralela en el DSC, de los cuales se extrajeron las mismas muestras que fueron simuladas en matlab para compararlas entre ambos y así validar el algoritmo neuronal recurrente, una vez validado la implementación fue hecha el acondicionando los convertidores A/D y D/A y varias configuraciones para las 2 arquitecturas a implementar.

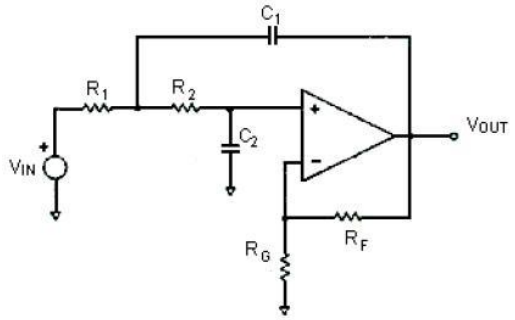


Figura 7.10 Planta a controlar “circuito eléctrico”

La función de transferencia del voltaje de salida respecto al de entrada del circuito anterior es la siguiente:

$$\frac{V_{OUT}(s)}{V_{IN}(s)} = \frac{\frac{K_O}{R_1 R_2 C_1 C_2}}{s^2 + s\left[\frac{1}{R_1 C_1} + \frac{1}{R_2 C_1} + \frac{1-K_O}{R_2 C_1}\right] + \frac{1}{R_1 R_2 C_1 C_2}} \tag{7.9}$$

Donde:

$$K_O = 1 + \frac{R_F}{R_G} \tag{7.10}$$

A continuación se presentan las funciones de transferencia de dos circuitos eléctricos diferentes con sus respectivos valores en sus componentes que nos

ofrecen dinámicas distintas pero teniendo el mismo circuito de la figura 7.10 en común, las respuestas de ambas plantas se muestran en la figura 7.11a y en la figura 7.11b se muestra el resultado del algoritmo de control neuronal aplicado a las dos plantas diferentes ante una entrada tipo escalón de 2 volts. Cabe mencionar que la planta dos se encuentra en lazo cerrado con retroalimentación unitaria.

Planta # 1:

$$\frac{V_{OUT}(s)}{V_{IN}(s)} = \frac{4.041}{s^2 + 1.859s + 4.037}$$ **à**

R_1 = 0.96 MΩ R_2 = 0.96 MΩ
C_1 = 1.12 uF C_2 = 0.24 uF
R_G = 98.7 kΩ R_F = 98.5 Ω

Planta # 2:

$$\frac{V_{OUT}(s)}{V_{IN}(s)} = \frac{0.8659}{s^2 + 1.859s + 1.731}$$ **à**

R_1 = 0.96 MΩ R_2 = 0.96 MΩ
C_1 = 1.12 uF C_2 = 1.12 uF
R_G = 98.7 kΩ R_F = 98.5 Ω

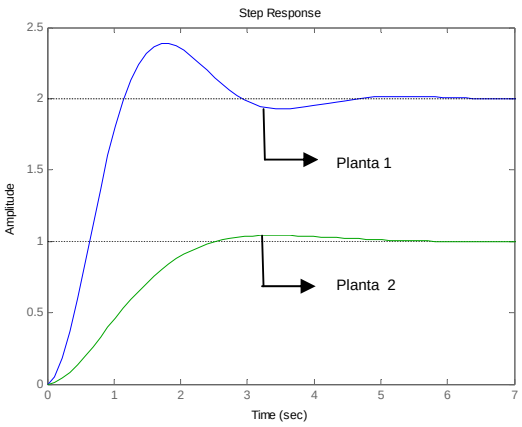


Figura 7.11a Respuesta de los modelos ante una entrada escalón de 2 volts

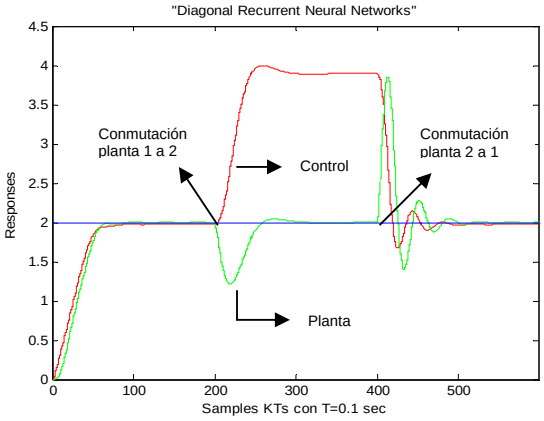


Figura 7.11b Respuesta del circuito eléctrico en la RNDR

En la figura 7.11b se muestra una sola gráfica que representa tanto la simulación como la validación (ya que las dos son exactamente iguales) que se obtuvo para el esquema de la RNDR, la simulación corre en la plataforma de matlab y la validación corre en el FPGA, DSC y la información es extraída por DUPG para comparación con la simulación.

Ahora lo siguiente fue implementar físicamente los sistemas 1 y 2 que en este caso son los circuitos eléctricos y conectarlos a la tarjeta ML402 para la primer validación y después al DSC para la segunda validación y así demostrar que la implementación es todo un éxito como se muestran en las figuras 7.12 a la 7.15, donde se aprecia que las gráficas son casi iguales a la de figura 7.11b, de ésta manera no solo queda validado el esquema de control basado en la RNDR sino que se ha desarrollado toda una plataforma para el desarrollo de algoritmos de control inteligente basado en redes neuronales artificiales, en la cual se pueden llevar a cabo el desarrollo e implementación de una manera rápida y eficiente.

Los resultados obtenidos en tiempo real mostrados en el osciloscopio figuras 7.12 y 7.13 se interpretan de la siguiente manera teniendo como referencia la gráfica de simulación de la figura 7.11b. La planta número uno llega a la referencia de 2 volts en 5 segundos, después de que se alcanza el estado estacionario se conmuta a la planta número dos, que ésta a su vez tarda otros 5 segundos en llegar de nuevo a la referencia mientras que la señal de control llega hasta 4.2 volts para mantener la salida de la planta en referencia, después de ello se conmuta ahora de la planta dos a la planta uno llegando a la referencia en 11 segundos, mientras que la señal de control mantiene la misma magnitud que la señal de la planta esto debido a que la planta uno tiene ganancia unitaria mientras que la planta dos tiene la mitad.

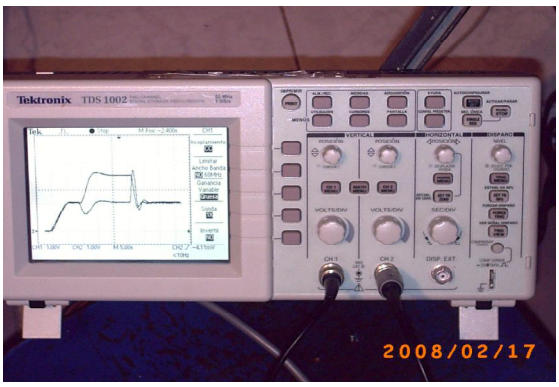


Figura 7.12 Respuestas en tiempo real

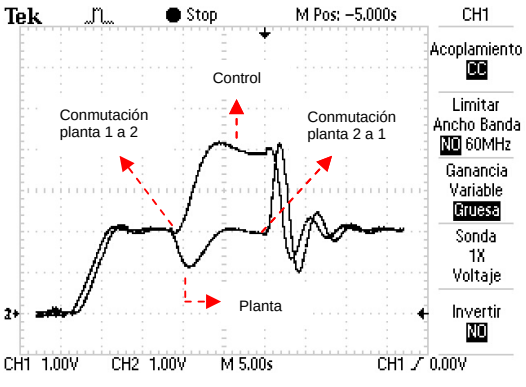


Figura 7.13 Señal de salida del circuito eléctrico Y(t) y del control U(t)

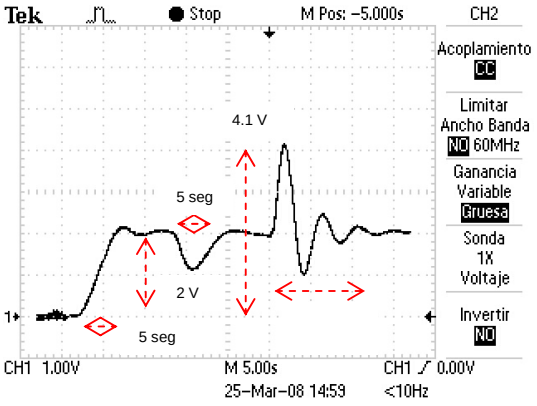


Figura 7.14 Respuesta del sistema $Y(t)$

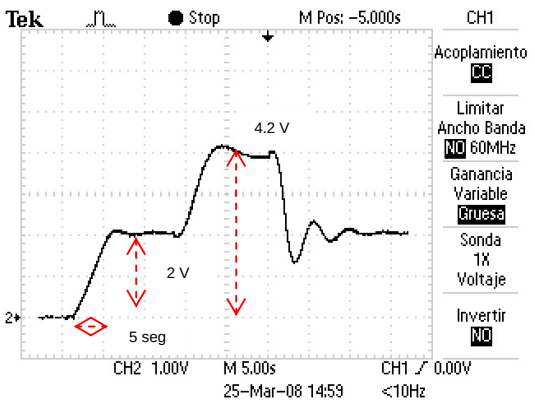


Figura 7.15 Señal de control $U(t)$

CONCLUSIONES

El algoritmo de control utilizado para la red neuronal diagonal recurrente presentada en esta tesis promete ser una alternativa muy interesante para los procesos en el área de control donde las dinámicas de las plantas son difíciles de modelar o estimar con los métodos clásicos. Los tipos de procesos en los que este algoritmo puede ser aplicado son muy amplios y abarcan la mayoría de los casos prácticos. La investigación de este tipo de algoritmo para el área de control permite contribuir de manera más eficiente a la solución de problemas en las áreas de control de ingeniería y de la ciencia exacta.

Los resultados fueron exitosos en todas las pruebas tanto de las simulaciones que se hicieron así como de las reales que se llevaron a cabo a través del FPGA y del DSC. Por otra parte comentar también que el diseño del esquema de control basado en el NIDR, NCDR y su algoritmo de retropropagación y llevado a la práctica tiene muchas ventajas frente a los clásicos algoritmos como el PID, como lo muestran los resultados anteriores donde se hace un cambio radical a la planta de tal manera que el control identifique y se adapte a los cambios y estime nuevos parámetros del controlador para cada instante de muestreo (cosa que los PID no poseen), y es ahí donde el campo de aplicación es muy tangible para el control neuronal recurrente, evitándose lo que se hace en la actualidad con los controles clásicos, que se tienen que ajustar para llegar a los valores que garantizan un control satisfactorio,

además que son muy susceptibles a desajustarse ante cualquier condición no considerada o perturbación.

En este proyecto de tesis, se analizaron las propiedades que ofrece el algoritmo de control neuronal recurrente además el diseño de estos mediante el uso de dos enfoques diferentes. Una conclusión casi obvia con respecto a los algoritmos de control clásicos es la habilidad de adaptarse a las dinámicas complicadas de las plantas y mantener el control en seguimiento y regulación siendo estos dos últimos asignados de manera independiente. Por otra parte, se considero que en este trabajo de investigación se alcanzaron los objetivos que eran esencialmente:

- Crear nuevos enfoques para la construcción e implementación de un control neuronal recurrente de propósito general.
- El diseño, implementación, metodología y análisis de éste a un sistema con el fin de comprobar su fiabilidad y efectividad.

En la etapa de desempeño del algoritmo de control de este proyecto, se mostró que el NIDR ofrece buenos resultados al identificar sistemas lineales cuyos modelos pueden cambiar en un tiempo determinado (una vez alcanzado el estado estacionario). A partir de estos experimentos se concluye que el uso de este algoritmo evita en gran medida el tratar con los algoritmos convencionales. Aunque los ejemplos presentados en este trabajo fueron para sistemas de una entrada-una salida, las ideas pueden ser extendidas para sistemas de control con muchas entradas y muchas salidas.

Al implementar el controlador neuronal recurrente se comparó su respuesta con las simuladas en la plataforma de Matlab, observando que las señales de control correspondientes tienen menos del 1% de error en el objetivo de control y logran la estabilidad de la planta en poco tiempo. Por tanto, de esta forma se concluye que el algoritmo de control neuronal analizado e implementado en este trabajo sugiere ser empleado en cualquier dominio de aplicación donde el objetivo principal del ingeniero de diseño sea el de resolver problemas de control sin la necesidad de emplear algoritmos complejos y por supuesto con un mínimo de recursos computacionales para su implementación.

LITERATURA CITADA

Ben Krosse and Patrick Van der Smagt. 1996. An introduction to Neural Networks, The University of Amsterdam, P. O. Box 1116, D-82230 Wessling GERMANY.

Chao-Chee Ku Lee, K.Y. January 1995. Diagonal recurrent neural networks for dynamic systems control, Neural Networks, IEEE Transactions on, Volume 6, pp.144 – 156.

De Jesús O. and Hagan T. M. 2007. Backpropagation Algorithm for a Broad Class of Dynamic Networks, IEEE Trans. Neural Netw., Vol. 18, No. 1, pp. 14-27.

Freeman A. James and Skapura M. David. 1991. Neural Networks Algorithms, Applications, and Programming Techniques, Addison Wesley publications.

Gómez espinosa Alfonso. 1999. criterios de diseño del controlador neuronal autoajustable y su aplicación a un sistema no lineal, tesis (mtro. en instrumentación y control automático) - Universidad Autónoma de Querétaro, Facultad de ingeniería, México, qro.

Hagan T. Martin, Demuth B. Howard and Beale Mark. January 2002. Neural Networks Design, Publisher: Martin Hagan.

Khaled Nouri, Rached Dhaouadi and Naceur Benhadj Braiek. March 2007. Adaptive control of a nonlinear dc motor drive using recurrent neural networks, Applied Soft Computing, In Press, Corrected Proof.

Laurene V. Fausett. 1994. Fundamentals of Neural Networks, architectures, algorithms and applications, Prentice Hall.

Lippmann R. April 1987. An Introduction to Computing With Neural Nets, IEEE ASSP Magazine, pages. 4-22.

Madan M. Gupta, Liang Yin and Noriyasu homma. 2003. Static and Dinamic neural networks from fundamentals to advance theory, Wiley-Interscience publication.

Maeda, Y. and Wakamura, M. November 2005. Simultaneous perturbation learning rule for recurrent neural networks and its FPGA implementation, Neural Networks, IEEE Transactions on Volume 16, Issue 6, Page(s): 1664 – 1672.

Man Z., Wu H. R., Liu S., and Yu X. November 2006. A new Adaptive Backpropagation Algorithm Based on Lyapunov Stability Theory for Neural Networks, IEEE Trans. Neural Netw., Vol. 17, No. 6, pp. 1580-1591.

Marwedel Peter. 2006. Embedded system Design, Springer.

Medsker R. L. and Jain C. L. 2001. Recurrent Neural Networks Design and Applications, CRC Press.

Minsky L. Marvin, Papert A. Seymour. 1987. Perceptrons - Expanded Edition, The MIT Press.

Narendra S. K. and Parthasarathy K. March 1990. Identification and control of dynamic systems using neural networks, IEEE, vol. I, No 1, pp. 4-27.

Omondi R. Amos and Rajapakse C. Jagath. 2006. FPGA Implementations of Neural Networks, Springer.

Pellerin David and Thibault Scott. 2005. Practical FPGA Programming in C, Prentice Hall.

Ren Tsai-Jiun and Chen Tien-Chi. August 2006. Robust speed-controlled induction motor drive based on recurrent neural network, Electric Power Systems Research, Volume 76, Issue 12, Pages 1064-1074.

Sharp Richard. 2005. Higher-Level Hardware Synthesis, Springer.

Yu L. D. and Chang K. T. March 2005. Adaptation of diagonal recurrent neural network model, Springer London, Volume 14, Number 3, pp. 189-197.

Xilinx, AccelDSP Synthesis Tool, System Generator for DSP, 2008.
http://www.xilinx.com/ise/dsp_design_prod/acceldsp/index.htm
http://www.xilinx.com/ise/optional_prod/system_generator.htm

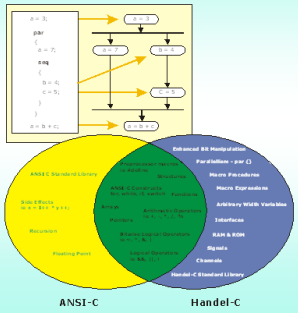
Agility, Agility DK Design Suite, 2008.
http://www.agilityds.com/products/c_based_products/dk_design_suite/default.aspx

APÉNDICE A
Handel-C HDL de alto nivel

Que es Handel-C

- Diseñado por Celoxica, es un lenguaje para implementar algoritmos en hardware directamente de una representación en C.
- Para diseñar sistemas usando Handel-C, se utiliza la herramienta de desarrollo y síntesis digital DK Design Suite.

ANSI C & Handel-C



Fundamentos del lenguaje Handel-C

- Handel-C es un lenguaje tipo C:

--Sintaxis y semántica ANSI-C.
--Extensiones y restricciones para el diseño hardware.

- Diseñado para el diseño hardware síncrono:

--Optimizado para FPGAs.
--Todo lo que se simula, compila en hardware.

- Extensiones a C permiten producir hardware eficiente:

--Par introduce el paralelismo.
--Anchura arbitraria de palabra.
--Sincronización.
--Interfaces hardware.

Procesos Secuenciales y Paralelos en Handel-C

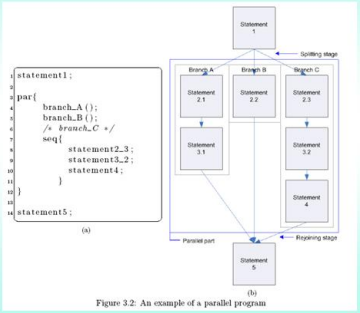


Figure 3.2: An example of a parallel program

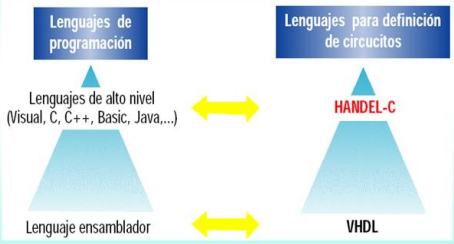
Comparativas de Implementación

The table below summarizes the results of the Handel-C implementation compared to the results in the parallel project.

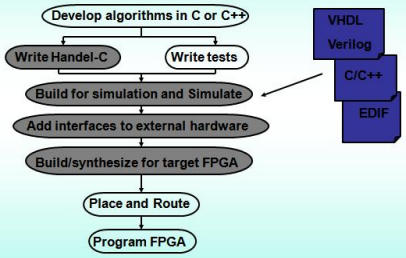
	Handel-C/ DK1	Verilog/Leonardo
Design time	4 man-months	12-16 man-months
Program size	40 pages	200 pages
Compile time	3 minutes	1.5 hours
Size (Virtex 1000-4)	35 % logic, 30 % memory*	NA
Size (Virtex 2000E-8)	17 % logic, 15 % memory*	All logic, all memory**
Speed (Virtex 1000-4)	28 – 33 MHz	NA
Speed (Virtex 2000E-8)	44 MHz	49 MHz

* Distributed memory. No block memory used
** A conscious choice. Used all logic to increase speed. All block memory used.

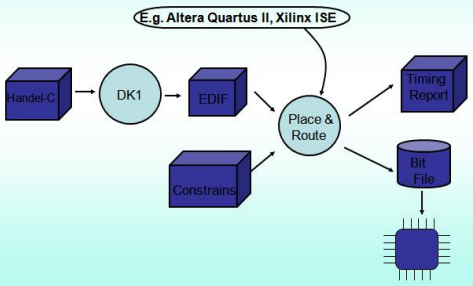
Comparativa Handel-C vs VHDL



Flujo de Diseño



FPGA Place and Route Tools



APÉNDICE B

El dsPIC30F de Microchip

La serie dsPIC30F ocupa una situación muy cercana a los DSP y a los microcontroladores de 32 bits, como puede apreciarse en el gráfico de la figura B.1. Aunque sus prestaciones no alcanzan las máximas de los DSP y microcontroladores de 32 bits, su precio es mucho más ventajoso.

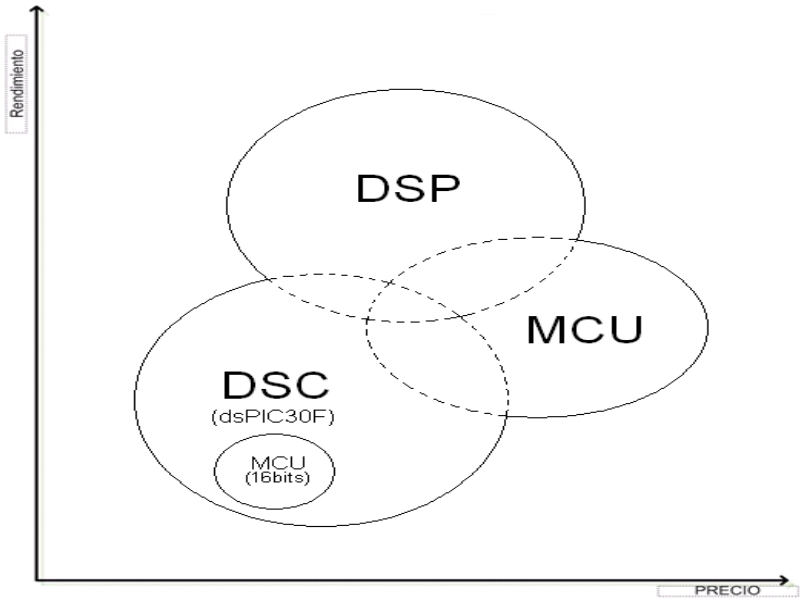


Figura B.1. Relación rendimiento-precio de los diferentes modos de procesamiento digital de la señal

Entre las aportaciones típicas de los DSP que se han implementado en la arquitectura básica de los dsPIC30F cabe destacar:

- Multiplicación MAC 16x16 en un ciclo.
- 2 acumuladores de 40 bits.
- Registro de desplazamiento de 40 bits.
- Acceso simultáneo de dos operandos.
- Bucles con estructura DO y REPEAT.
- Bloque de registros de trabajo.
- Juego flexible de interrupciones.
- Perro guardián
- Emulación en tiempo real.
- Optimizado para programación en lenguaje C.

Con todas estas cualidades, los dsPIC proporcionan un rendimiento muy aceptable en las aplicaciones típicas de los DSP, como puede apreciarse en la tabla de la figura B.2.

Función	Ecuación de la cuenta de ciclos	Condiciones	N^{os} de ciclos	Tiempo de ejecución @30 MIPS
Complejo FFT**	-	N=64	3739	124.6us
Complejo FFT**	-	N=128	8485	282.8us
Complejo FFT**	-	N=256	19055	635.2us
Bloque FIR	53+N(4+M)	N=32, M=32	1205	40.2us
Bloque FIR Lattice	41+N(4+7M)	N=32, M=32	7337	244.6us
Bloque IIR Canonic	36+N(8+7S)	N=32, S=4	1188	39.6us
Bloque IIR Lattice	46+(16+7M)	N=32, M=8	2350	78.3us
Matriz Suma	20+3(C*R)	C=8, R=8	212	7.1us
Matriz Transposición	16+C(6+3(R-1))	C=8, R=8	232	7.7us
Vector de Ptos de Producto	17+3N	N=32	113	3.8us
Vector Máximo	19+7(N-2)	N=32	229	7.6us
Vector Producto	17+4N	N=32	145	4.8us
Vector Potencia	16+2N	N=32	80	2.7us

*C= # columnas, N= #muestras, M= # S= #Secciones R= #Filas
La rutina complejo FFT previene inherentemente el "overflow".
1Ciclo= 33 nanosegundos @ 30MIPS

Figura B.2. Tabla que recoge el tiempo necesario para la realización de funciones típicas del DSP, por los dsPIC30F a 30 MIPS.

Principales Características de los dsPIC:

Rango de funcionamiento:

- DC -30MIPS (30MIPS@ 4.5-5.5V **,** -40' a 85ºC)
- Vdd de 2.5 a 5.5 V
- Temperatura: interna (-40 a 85 º C) y externa (-40 a 125 º C)

CPU de alto rendimiento:

- Núcleo RISC con arquitectura Harvard modificada.
- Juego de instrucciones optimizado para el lenguaje C.
- Bus de datos de 16 bits.
- Bus de instrucciones de 24 bits.
- Repertorio de 84 instrucciones, la mayoría de una palabra de tamaño y ejecutable en un ciclo.
- Banco de 16 registros de propósito general de 16 bits.
- 2 acumuladores de 40 bits, con opciones de redondeo y saturación.
- Modos complejos de direccionamiento indirecto: modular o circular y de inversión de bits ó "bit-reversed".
- Manejo de la pila con software.
- Multiplicador para enteros y fracciones de 16x16.
- División de 32/16 y 16/16.
- Operación de multiplicación y acumulación en un ciclo.
- Registro de desplazamiento de 40 bits.

Controlador de interrupciones:

- Latencia de 5 ciclos.
- Hasta 45 fuentes de interrupción, 5 externas.
- 7 niveles de prioridad, programables.
- 4 excepciones especiales.

Entradas y salidas digitales:

- Hasta 54 patitas programables de E/S digitales.
- 25 mA de consumo por cada patita de E/S.

Memorias:

- Memoria de programa FLASH de hasta 144KB con 100.000 ciclos de borrado/escritura.
- Memoria de datos EEPROM de hasta 4KB con 1.000.000 de ciclos de borrado/escritura.
- Memoria de datos SRAM de hasta 8KB.

Manejo del sistema:

- Flexibles opciones para el reloj de trabajo (Externo, cristal, resonador, RC interno, totalmente integrado PLL, etc.)
- Temporizador programable de “power-up”.
- Temporizador/ estabilizador del oscilador “start-up”.
- Perro guardián con oscilador RC propio.
- Monitor de fallo de reloj.

Control de alimentación:

- Conmutación entre fuentes de reloj en tiempo real.
- Manejo de consumo de los periféricos.
- Detector programable de voltaje bajo.
- Reset programable de “brown-out”.
- Modos de bajo consumo IDLE y SLEEP.

Temporizadores, módulos de captura, comparación y PWM:

- Hasta 5 temporizadores de 16 bits, pudiendo implementar parejas para alcanzar 32 bits y pudiendo trabajar en tiempo real con oscilador externo de 32 KHz.
- Módulo de entrada de 8 canales para la captura por flanco ascendente, descendente o ambos.
- Módulo de salida de comparación hasta 8 canales, en modo simple o doble de 16 bits.
- Modo PWM de 16 bits.

Módulos de comunicación:

- Hasta 2 módulos SPI de 3 líneas.
- Interfaz I/O con CODEC.
- I2C con modo multi-maestro esclavo, con 7 y 10 bits de dirección y con detección y arbitraje de colisiones de bus.
- Hasta 2 módulos UART.
- Módulo de interfaz CODEC que soporta los protocolos I2S y AC97.
- Hasta 2 módulos CAN 2-0B.

Periféricos para control de motores:

- PWM para control de motores de hasta 8 canales con 4 generadores de “duty-cicle”.

Conversores analógico/digital:

- Módulo conversor A/D de 10 bits y 500 Ksps, con 2 o 4 muestras simultáneas y hasta 16 canales de entrada. Conversión posible en el modo SEP
- Módulo conversor A/D de 12 bits y 100 Ksps con 16 canales.

APÉNDICE C

Programa de la RNDR en el FPGA

```
#include "delay.hch"
#include "ADC12.hch"
#include "DAC12.hch"
#include "operaciones.hch"

set clock = external "AE14";
void main()
{
   unsigned 1 x;
   unsigned 12 Analogica_0;
   int 32 res;
   Float tem1,tem2,tem3;
   static Float uno=FloatUnpackFromInt32(0x3F800000);
   Float Y, Y_1, Y_2, Ym, Yu, U, U_1, U_2, r, Yr, Yr_1,
ec, em, Sc1, Sc2, Sc3, dFSc1, dFSc2, dFSc3, dFSi1,
dFSi2,dFSi3,Si1,Si2,Si3,        Xc1, Xc1_1, Xc2, Xc2_1,
Xc3, Xc3_1, Pc1, Pc1_1, Pc2, Pc2_1, Pc3, Pc3_1,
Qc11,Qc11_1,Qc21,Qc21_1,Qc31,Qc31_1,Qc12,
Qc12_1,Qc22,Qc22_1,Qc32,Qc32_1,Qc13,Qc13_1,Qc2
3,Qc23_1,Qc33,Qc33_1,Qi11,Qi11_1,Qi21,Qi21_1,Qi31,
Qi31_1,    Qi12, Qi12_1, Qi22, Qi22_1, Qi32, Qi32_1,
Qi13, Qi13_1, Qi23, Qi23_1, Qi33, Qi33_1, Pi1, Pi1_1,
Pi2,Pi2_1,Pi3,Pi3_1,Xi1, Xi1_1, Xi2, Xi2_1, Xi3, Xi3_1,
Wc1_O_1, Wc2_O_1, Wc3_O_1, Wc11_I_1, Wc21_I_1,
Wc31_I_1,Wc12_I_1,Wc22_I_1,   Wc32_I_1, Wc13_I_1,
Wc23_I_1, Wc33_I_1, Wc1_D_1, Wc2_D_1, Wc3_D_1,
Wi1_O_1,  Wi2_O_1,  Wi3_O_1,  Wi11_I_1,Wi21_I_1,
Wi31_I_1, Wi12_I_1, Wi22_I_1, Wi32_I_1, Wi13_I_1,
Wi23_I_1,Wi33_I_1,Wi1_D_1,          Wi2_D_1,
Wi3_D_1;Wc1_O,Wc2_O,Wc3_O,Wc11_I,Wc21_I,Wc31
_I,Wc12_I,Wc22_I,Wc32_I,Wc13_I,Wc23_I,Wc33_I,Wc1
_D,Wc2_D,Wc3_D,Wi1_O,Wi2_O,    Wi3_O,    Wi11_I,
Wi21_I,Wi31_I,Wi12_I,Wi22_I,Wi32_I,Wi13_I,Wi23_I,Wi
33_I,Wi1_D,Wi2_D,Wi3_D,     Eta_C,Eta_I,     ajuste,
bias=uno;
Eta_C = FloatUnpackFromInt32(0x3E2E147B);  // 0.17
Eta_I = FloatUnpackFromInt32(0x38D1B717);  // 0.0001
ajuste = FloatUnpackFromInt32(0x3E19999A);  // 0.15
dac_init();
adc_init();
while(1)
{
Analogica_0 = read_analog(0); // canal cero
tem1=Entero_a_Float((int 32)(0@Analogica_0));
tem2=multiplica(tem1,FloatUnpackFromInt32(0x40A000
00));  // Y=analogica*5.0/4095.0;
Y=divide(tem2,FloatUnpackFromInt32(0x457FF000));
delay_ms(100);
Yr = FloatUnpackFromInt32(0x40000000); //2.0
r = FloatUnpackFromInt32(0x40000000);  //2.0
//>>>>>>>>>>>>>>>>ec=Yr-Y;
ec=resta(Yr,Y);
//>>>>>>>>>>>>>>>>em=Y-Ym;
em=resta(Y,Ym);
//PARTE DEL NEUROCONTROLADOR
//j=1;
//>>>>>>>>>>>>>>>>Wc11_I=Wc11_I_1+(Eta_C*(ec*Yu
*Wc1_O*Qc11));
tem1 = multiplica(Eta_C,ec);
tem2 = multiplica(tem1,Yu);
tem1 = multiplica(tem2,Wc1_O);
tem2 = multiplica(tem1,Qc11);
Wc11_I = suma(Wc11_I_1,tem2);
//>>>>>>>>>>>>>>>>Wc21_I=Wc21_I_1+(Eta_C*(ec*Yu
*Wc1_O*Qc21));
tem1 = multiplica(Eta_C,ec);
tem2 = multiplica(tem1,Yu);
tem1 = multiplica(tem2,Wc1_O);
tem2 = multiplica(tem1,Qc21);
Wc21_I = suma(Wc21_I_1,tem2);
//>>>>>>>>>>>>>>>>Wc31_I=Wc31_I_1+(Eta_C*(ec*Yu
*Wc1_O*Qc31));
tem1 = multiplica(Eta_C,ec);
tem2 = multiplica(tem1,Yu);
tem1 = multiplica(tem2,Wc1_O);
tem2 = multiplica(tem1,Qc31);
Wc31_I = suma(Wc31_I_1,tem2);
//>>>>>>>>>>>>>>>Qc11=dFSc1*(r+(Wc1_D*Qc11_1));
tem1 = multiplica(Wc1_D,Qc11_1);
tem2 = suma(r,tem1);
Qc11 = multiplica(dFSc1,tem2);
//>>>>>>>>>>>>Qc21=dFSc1*(U_1+(Wc1_D*Qc21_1));
tem1 = multiplica(Wc1_D,Qc21_1);
tem2 = suma(U_1,tem1);
Qc21 = multiplica(dFSc1,tem2);
//>>>>>>>>>>>>Qc31=dFSc1*(Y_1+(Wc1_D*Qc31_1));
tem1 = multiplica(Wc1_D,Qc31_1);
tem2 = suma(Y_1,tem1);
Qc31 = multiplica(dFSc1,tem2);
//j=2;
//>>>>>>>>>>>>>>>>Wc12_I=Wc12_I_1+(Eta_C*(ec*Yu
*Wc2_O*Qc12));
tem1 = multiplica(Eta_C,ec);
tem2 = multiplica(tem1,Yu);
tem1 = multiplica(tem2,Wc2_O);
tem2 = multiplica(tem1,Qc12);
Wc12_I = suma(Wc12_I_1,tem2);
//>>>>>>>>>>>>>>>>Wc22_I=Wc22_I_1+(Eta_C*(ec*Yu
*Wc2_O*Qc22));
tem1 = multiplica(Eta_C,ec);
tem2 = multiplica(tem1,Yu);
tem1 = multiplica(tem2,Wc2_O);
tem2 = multiplica(tem1,Qc22);
Wc22_I = suma(Wc22_I_1,tem2);
//>>>>>>>>>>>>>>>>Wc32_I=Wc32_I_1+(Eta_C*(ec*Yu
*Wc2_O*Qc32));
tem1 = multiplica(Eta_C,ec);
tem2 = multiplica(tem1,Yu);
tem1 = multiplica(tem2,Wc2_O);
tem2 = multiplica(tem1,Qc32);
Wc32_I = suma(Wc32_I_1,tem2);
//>>>>>>>>>>>>>>>>Qc12=dFSc2*(r+(Wc2_D*Qc12_1))
tem1 = multiplica(Wc2_D,Qc12_1);
tem2 = suma(r,tem1);
Qc12 = multiplica(dFSc2,tem2);
//>>>>>>>>>>>>Qc22=dFSc2*(U_1+(Wc2_D*Qc22_1));
tem1 = multiplica(Wc2_D,Qc22_1);
tem2 = suma(U_1,tem1);
Qc22 = multiplica(dFSc2,tem2);
//>>>>>>>>>>>>Qc32=dFSc2*(Y_1+(Wc2_D*Qc32_1));
tem1 = multiplica(Wc2_D,Qc32_1);
tem2 = suma(Y_1,tem1);
Qc32 = multiplica(dFSc2,tem2);
//j=3;
//>>>>>>>>>>>>>>>>Wc13_I=Wc13_I_1+(Eta_C*(ec*Yu
*Wc3_O*Qc13));
tem1 = multiplica(Eta_C,ec);
tem2 = multiplica(tem1,Yu);
tem1 = multiplica(tem2,Wc3_O);
tem2 = multiplica(tem1,Qc13);
Wc13_I = suma(Wc13_I_1,tem2);
//>>>>>>>>>>>>>>>>Wc23_I=Wc23_I_1+(Eta_C*(ec*Yu
*Wc3_O*Qc23));
tem1 = multiplica(Eta_C,ec);
tem2 = multiplica(tem1,Yu);
tem1 = multiplica(tem2,Wc3_O);
tem2 = multiplica(tem1,Qc23);
Wc23_I = suma(Wc23_I_1,tem2);
//>>>>>>>>>>>>>>>>Wc33_I=Wc33_I_1+(Eta_C*(ec*Yu
*Wc3_O*Qc33));
tem1 = multiplica(Eta_C,ec);
tem2 = multiplica(tem1,Yu);
tem1 = multiplica(tem2,Wc3_O);
tem2 = multiplica(tem1,Qc33);
Wc33_I = suma(Wc33_I_1,tem2);
```

```
//>>>>>>>>>>>>>>>Qc13=dFSc3*(r+(Wc3_D*Qc13_1));
tem1 = multiplica(Wc3_D,Qc13_1);
tem2 = suma(r,tem1);
Qc13 = multiplica(dFSc3,tem2);
//>>>>>>>>>>>>>>Qc23=dFSc3*(U_1+(Wc3_D*Qc23_1));
tem1 = multiplica(Wc3_D,Qc23_1);
tem2 = suma(U_1,tem1);
Qc23 = multiplica(dFSc3,tem2);
//>>>>>>>>>>>>>Qc33=dFSc3*(Y_1+(Wc3_D*Qc33_1));
tem1 = multiplica(Wc3_D,Qc33_1);
tem2 = suma(Y_1,tem1);
Qc33 = multiplica(dFSc3,tem2);
//j=1;
//>>>>>>>>>>>>>>>Wc1_D=Wc1_D_1+(Eta_C*(ec*Yu*
Wc1_O*Pc1));
tem1 = multiplica(Eta_C,ec);
tem2 = multiplica(tem1,Yu);
tem1 = multiplica(tem2,Wc1_O);
tem2 = multiplica(tem1,Pc1);
Wc1_D = suma(Wc1_D_1,tem2);
//>>>>>>>>>>>>>Pc1=dFSc1*(Xc1_1+(Wc1_D*Pc1_1));
tem1 = multiplica(Wc1_D,Pc1_1);
tem2 = suma(Xc1_1,tem1);
Pc1 = multiplica(dFSc1,tem2);
//j=2;
//>>>>>>>>>>>>>>>Wc2_D=Wc2_D_1+(Eta_C*(ec*Yu*
Wc2_O*Pc2));
tem1 = multiplica(Eta_C,ec);
tem2 = multiplica(tem1,Yu);
tem1 = multiplica(tem2,Wc2_O);
tem2 = multiplica(tem1,Pc2);
Wc2_D = suma(Wc2_D_1,tem2);
//>>>>>>>>>>>>>Pc2=dFSc2*(Xc2_1+(Wc2_D*Pc2_1));
tem1 = multiplica(Wc2_D,Pc2_1);
tem2 = suma(Xc2_1,tem1);
Pc2 = multiplica(dFSc2,tem2);
//j=3;
//>>>>>>>>>>>>>>>Wc3_D=Wc3_D_1+(Eta_C*(ec*Yu*
Wc3_O*Pc3));
tem1 = multiplica(Eta_C,ec);
tem2 = multiplica(tem1,Yu);
tem1 = multiplica(tem2,Wc3_O);
tem2 = multiplica(tem1,Pc3);
Wc3_D = suma(Wc3_D_1,tem2);
//>>>>>>>>>>>>>Pc3=dFSc3*(Xc3_1+(Wc3_D*Pc3_1));
tem1 = multiplica(Wc3_D,Pc3_1);
tem2 = suma(Xc3_1,tem1);
Pc3 = multiplica(dFSc3,tem2);
//j=1
//>>>>>>>>>>>>>>>Sc1=(Wc1_D*Xc1_1)+(Wc11_I*r)+(
Wc21_I*U_1)+(Wc31_I*Y_1);
tem1 = multiplica(Wc1_D,Xc1_1);
tem2 = multiplica(Wc11_I,r);
tem3 = suma(tem1,tem2);
tem1 = multiplica(Wc21_I,U_1);
tem2 = suma(tem3,tem1);
tem1 = multiplica(Wc31_I,Y_1);
Sc1 = suma(tem2,tem1);
x=mayor(Sc1,FloatUnpackFromInt32(0x40A00000)); // 5
if(x)
    tem2=FloatUnpackFromInt32(0x00000000); // cero
else
{
    //a=expo(-Sc1);
tem1=multiplica(FloatUnpackFromInt32(0xBF800000),S
c1);
tem2 = exp(tem1);
}
//>>>>>>>>>>>>>>>Xc1=1/(1+exp(-Sc1));
tem1 = suma(uno,tem2);
Xc1 = divide(uno,tem1);
//>>>>>>>>>>>>>>>dFSc1=exp(-Sc1)/((exp(-
Sc1)+1)*(exp(-Sc1)+1));
 tem3 = multiplica(tem1,tem1);
dFSc1 = divide(tem2,tem3);
//j=2
//>>>>>>>>>>>>>>>Sc2=(Wc2_D*Xc2_1)+(Wc12_I*r)+(
Wc22_I*U_1)+(Wc32_I*Y_1);
tem1 = multiplica(Wc2_D,Xc2_1);
tem2 = multiplica(Wc12_I,r);
tem3 = suma(tem1,tem2);
tem1 = multiplica(Wc22_I,U_1);
tem2 = suma(tem3,tem1);
tem1 = multiplica(Wc32_I,Y_1);
Sc2 = suma(tem2,tem1);
x=mayor(Sc2,FloatUnpackFromInt32(0x40A00000)); // 5
if(x)
  tem2=FloatUnpackFromInt32(0x00000000); // cero
else
{
//a=expo(-Sc2);
tem1=multiplica(FloatUnpackFromInt32(0xBF800000),S
c2);
tem2 = exp(tem1);
}
//>>>>>>>>>>>>>>>Xc2=1/(1+exp(-Sc2));
tem1 = suma(uno,tem2);
Xc2 = divide(uno,tem1);
//>>>>>>>>>>>>>>>dFSc2=exp(-Sc2)/((exp(-
Sc2)+1)*(exp(-Sc2)+1));
tem3 = multiplica(tem1,tem1);
dFSc2 = divide(tem2,tem3);
//j=3
//>>>>>>>>>>>>>>>Sc3=(Wc3_D*Xc3_1)+(Wc13_I*r)+(
Wc23_I*U_1)+(Wc33_I*Y_1);
tem1 = multiplica(Wc3_D,Xc3_1);
tem2 = multiplica(Wc13_I,r);
tem3 = suma(tem1,tem2);
tem1 = multiplica(Wc23_I,U_1);
tem2 = suma(tem3,tem1);
tem1 = multiplica(Wc33_I,Y_1);
Sc3=suma(tem2,tem1);
x=mayor(Sc3,FloatUnpackFromInt32(0x40A00000)); // 5
if(x)
tem2=FloatUnpackFromInt32(0x00000000); // cero
else
{
//a=expo(-Sc3);
tem1=multiplica(FloatUnpackFromInt32(0xBF800000),S
c3);
tem2 = exp(tem1); // exp(-Sc3)
}
//>>>>>>>>>>>>>>>Xc3=1/(1+exp(-Sc3));
tem1 = suma(uno,tem2);
Xc3 = divide(uno,tem1);
//>>>>>>>>>>>>>>>dFSc3=exp(-Sc3)/((exp(-
Sc3)+1)*(exp(-Sc3)+1));
tem3 = multiplica(tem1,tem1);
dFSc3 = divide(tem2,tem3);
//j=1;
//>>>>>>>>>>Wc1_O=Wc1_O_1+(Eta_C*(ec*Yu*Xc1));
tem1 = multiplica(Eta_C,ec);
tem2 = multiplica(tem1,Yu);
tem1 = multiplica(tem2,Xc1);
Wc1_O = suma(Wc1_O_1,tem1);
//j=2;
//>>>>>>>>>>Wc2_O=Wc2_O_1+(Eta_C*(ec*Yu*Xc2));
tem1 = multiplica(Eta_C,ec);
tem2 = multiplica(tem1,Yu);
tem1 = multiplica(tem2,Xc2);
Wc2_O = suma(Wc2_O_1,tem1);
//j=3;
//>>>>>>>>>>Wc3_O=Wc3_O_1+(Eta_C*(ec*Yu*Xc3));
tem1 = multiplica(Eta_C,ec);
tem2 = multiplica(tem1,Yu);
tem1 = multiplica(tem2,Xc3);
Wc3_O = suma(Wc3_O_1,tem1);
//>>>>>>>>>>>>>>>U=(Wc1_O*Xc1)+(Wc2_O*Xc2)+(
Wc3_O*Xc3);
tem1 = multiplica(Wc1_O,Xc1);
tem2 = multiplica(Wc2_O,Xc2);
tem3 = suma(tem1,tem2);
```

```
tem1 = multiplica(Wc3_O,Xc3);
U = suma(tem3,tem1);
x=mayor(U,FloatUnpackFromInt32(0x40A00000)); // 5
if(x)
U=FloatUnpackFromInt32(0x40A00000); // 5
 x=menor(U,FloatUnpackFromInt32(0x00000000)); // 0
if(x)
U=FloatUnpackFromInt32(0x00000000); // 0
tem1=multiplica(U,FloatUnpackFromInt32(0x457FF000))
;
tem2=divide(tem1,FloatUnpackFromInt32(0x40A00000))
;
res=Float_a_Entero(tem2);
write_dac((unsigned 12)res[11:0],0); // canal 0
//>>>>>>>>>>>PARTE DEL NEUROIDENTIFICADOR
//j=1;
//>>>>>>>Wi11_I=Wi11_I_1+(Eta_I*(em*Wi1_O*Qi11));
tem1 = multiplica(Eta_I,em);
tem2 = multiplica(tem1,Wi1_O);
tem1 = multiplica(tem2,Qi11);
Wi11_I = suma(Wi11_I_1,tem1);
//>>>>>>>Wi21_I=Wi21_I_1+(Eta_I*(em*Wi1_O*Qi21));
tem1 = multiplica(Eta_I,em);
tem2 = multiplica(tem1,Wi1_O);
tem1 = multiplica(tem2,Qi21);
Wi21_I = suma(Wi21_I_1,tem1);
//>>>>>>>Wi31_I=Wi31_I_1+(Eta_I*(em*Wi1_O*Qi31));
tem1 = multiplica(Eta_I,em);
tem2 = multiplica(tem1,Wi1_O);
tem1 = multiplica(tem2,Qi31);
Wi31_I = suma(Wi31_I_1,tem1);
 //>>>>>>>>>>>>>>>Qi11=dFSi1*(U+(Wi1_D*Qi11_1));
tem1 = multiplica(Wi1_D,Qi11_1);
tem2 = suma(U,tem1);
Qi11 = multiplica(dFSi1,tem2);
//>>>>>>>>>>>>>Qi21=dFSi1*(Y_1+(Wi1_D*Qi21_1));
tem1 = multiplica(Wi1_D,Qi21_1);
tem2 = suma(Y_1,tem1);
Qi21 = multiplica(dFSi1,tem2);
//>>>>>>>>>>>>>Qi31=dFSi1*(bias+(Wi1_D*Qi31_1));
tem1 = multiplica(Wi1_D,Qi31_1);
tem2 = suma(bias,tem1);
Qi31 = multiplica(dFSi1,tem2);
//j=2;
//>>>>>>>Wi12_I=Wi12_I_1+(Eta_I*(em*Wi2_O*Qi12));
tem1 = multiplica(Eta_I,em);
tem2 = multiplica(tem1,Wi2_O);
tem1 = multiplica(tem2,Qi12);
Wi12_I = suma(Wi12_I_1,tem1);
//>>>>>>>Wi22_I=Wi22_I_1+(Eta_I*(em*Wi2_O*Qi22));
tem1 = multiplica(Eta_I,em);
tem2 = multiplica(tem1,Wi2_O);
tem1 = multiplica(tem2,Qi22);
Wi22_I = suma(Wi22_I_1,tem1);
//>>>>>>>Wi32_I=Wi32_I_1+(Eta_I*(em*Wi2_O*Qi32));
tem1 = multiplica(Eta_I,em);
tem2 = multiplica(tem1,Wi2_O);
tem1 = multiplica(tem2,Qi32);
Wi32_I = suma(Wi32_I_1,tem1);
 //>>>>>>>>>>>>>>>Qi12=dFSi2*(U+(Wi2_D*Qi12_1));
tem1 = multiplica(Wi2_D,Qi12_1);
tem2 = suma(U,tem1);
Qi12 = multiplica(dFSi2,tem2);
//>>>>>>>>>>>>>Qi22=dFSi2*(Y_1+(Wi2_D*Qi22_1));
tem1 = multiplica(Wi2_D,Qi22_1);
tem2 = suma(Y_1,tem1);
Qi22 = multiplica(dFSi2,tem2);
//>>>>>>>>>>>>>>Qi32=dFSi2*(bias+(Wi2_D*Qi32_1));
tem1 = multiplica(Wi2_D,Qi32_1);
tem2 = suma(bias,tem1);
Qi32 = multiplica(dFSi2,tem2);
//j=3;
//>>>>>>>Wi13_I=Wi13_I_1+(Eta_I*(em*Wi3_O*Qi13));
tem1 = multiplica(Eta_I,em);
tem2 = multiplica(tem1,Wi3_O);
tem1 = multiplica(tem2,Qi13);
Wi13_I = suma(Wi13_I_1,tem1);
//>>>>>>>Wi23_I=Wi23_I_1+(Eta_I*(em*Wi3_O*Qi23));
tem1 = multiplica(Eta_I,em);
tem2 = multiplica(tem1,Wi3_O);
tem1 = multiplica(tem2,Qi23);
Wi23_I = suma(Wi23_I_1,tem1);
//>>>>>>>Wi33_I=Wi33_I_1+(Eta_I*(em*Wi3_O*Qi33));
tem1 = multiplica(Eta_I,em);
tem2 = multiplica(tem1,Wi3_O);
tem1 = multiplica(tem2,Qi33);
Wi33_I = suma(Wi33_I_1,tem1);
 //>>>>>>>>>>>>>>>Qi13=dFSi3*(U+(Wi3_D*Qi13_1));
tem1 = multiplica(Wi3_D,Qi13_1);
tem2 = suma(U,tem1);
Qi13 = multiplica(dFSi3,tem2);
//>>>>>>>>>>>>>>Qi23=dFSi3*(Y_1+(Wi3_D*Qi23_1));
tem1 = multiplica(Wi3_D,Qi23_1);
tem2 = suma(Y_1,tem1);
Qi23 = multiplica(dFSi3,tem2);
//>>>>>>>>>>>>>>Qi33=dFSi3*(bias+(Wi3_D*Qi33_1));
tem1 = multiplica(Wi3_D,Qi33_1);
tem2 = suma(bias,tem1);
Qi33 = multiplica(dFSi3,tem2);
//j=1;
//>>>>>>>>>>Wi1_D=Wi1_D_1+(Eta_I*(em*Wi1_O*Pi1));
tem1 = multiplica(Eta_I,em);
tem2 = multiplica(tem1,Wi1_O);
tem1 = multiplica(tem2,Pi1);
Wi1_D = suma(Wi1_D_1,tem1);
//>>>>>>>>>>>>>>>Pi1=dFSi1*(Xi1_1+(Wi1_D*Pi1_1));
tem1 = multiplica(Wi1_D,Pi1_1);
tem2 = suma(Xi1_1,tem1);
Pi1 = multiplica(dFSi1,tem2);
//j=2;
//>>>>>>>>>>Wi2_D=Wi2_D_1+(Eta_I*(em*Wi2_O*Pi2));
tem1 = multiplica(Eta_I,em);
tem2 = multiplica(tem1,Wi2_O);
tem1 = multiplica(tem2,Pi2);
Wi2_D = suma(Wi2_D_1,tem1);
//>>>>>>>>>>>>>>>Pi2=dFSi2*(Xi2_1+(Wi2_D*Pi2_1));
tem1 = multiplica(Wi2_D,Pi2_1);
 tem2 = suma(Xi2_1,tem1);
Pi2 = multiplica(dFSi2,tem2);
//j=3;
//>>>>>>>>>>Wi3_D=Wi3_D_1+(Eta_I*(em*Wi3_O*Pi3));
tem1 = multiplica(Eta_I,em);
tem2 = multiplica(tem1,Wi3_O);
tem1 = multiplica(tem2,Pi3);
Wi3_D = suma(Wi3_D_1,tem1);
//>>>>>>>>>>>>>>>Pi3=dFSi3*(Xi3_1+(Wi3_D*Pi3_1));
tem1 = multiplica(Wi3_D,Pi3_1);
tem2 = suma(Xi3_1,tem1);
Pi3 = multiplica(dFSi3,tem2);
//j=1
//>>>>>>>>>>>>>>>Si1=(Wi1_D*Xi1_1)+(Wi11_I*U)+(
Wi21_I*Y_1)+(Wi31_I*bias);
tem1 = multiplica(Wi1_D,Xi1_1);
tem2 = multiplica(Wi11_I,U);
tem3 = suma(tem1,tem2);
tem1 = multiplica(Wi21_I,Y_1);
tem2 = suma(tem3,tem1);
tem1 = multiplica(Wi31_I,bias);
Si1 = suma(tem2,tem1);)
x=mayor(Si1,FloatUnpackFromInt32(0x40A00000)); // 5
if(x)
tem2=FloatUnpackFromInt32(0x00000000); // cero
else
{
//a=expo(-Si1);
tem1=multiplica(FloatUnpackFromInt32(0xBF800000),Si
1);
tem2 = exp(tem1);
}
//>>>>>>>>>>>>>>>Xi1=1/(1+exp(-Si1));
tem1 = suma(uno,tem2);
Xi1 = divide(uno,tem1);
```

```
//>>>>>>>>>>>>>>>dFSi1=exp(-Si1)/((exp(-
Si1)+1)*(exp(-Si1)+1));
tem3 = multiplica(tem1,tem1);
dFSi1 = divide(tem2,tem3);
//j=2
//>>>>>>>>>>>>>>>Si2=(Wi2_D*Xi2_1)+(Wi12_I*U)+(
Wi22_I*Y_1)+(Wi32_I*bias);
tem1 = multiplica(Wi2_D,Xi2_1);
tem2 = multiplica(Wi12_I,U);
tem3 = suma(tem1,tem2);
tem1 = multiplica(Wi22_I,Y_1);
tem2 = suma(tem3,tem1);
tem1 = multiplica(Wi32_I,bias);
Si2=suma(tem2,tem1);
x=mayor(Si2,FloatUnpackFromInt32(0x40A00000)); // 5
if(x)
tem2=FloatUnpackFromInt32(0x00000000); // cero
else
{
//a=expo(-Si2);
tem1=multiplica(FloatUnpackFromInt32(0xBF800000),Si
2);
tem2 = exp(tem1);
}
//>>>>>>>>>>>>>>>Xi2=1/(1+exp(-Si2));
tem1 = suma(uno,tem2);
Xi2 = divide(uno,tem1);
//>>>>>>>>>>>>>>>dFSi2=exp(-Si2)/((exp(-
Si2)+1)*(exp(-Si2)+1));
tem3 = multiplica(tem1,tem1);
dFSi2 = divide(tem2,tem3);
//j=3
//>>>>>>>>>>>>>>>Si3=(Wi3_D*Xi3_1)+(Wi13_I*U)+(
Wi23_I*Y_1)+(Wi33_I*bias);
tem1 = multiplica(Wi3_D,Xi3_1);
tem2 = multiplica(Wi13_I,U);
tem3 = suma(tem1,tem2);
tem1 = multiplica(Wi23_I,Y_1);
tem2 = suma(tem3,tem1);
tem1 = multiplica(Wi33_I,bias);
Si3=suma(tem2,tem1);
x=mayor(Si3,FloatUnpackFromInt32(0x40A00000)); // 5
if(x)
tem2=FloatUnpackFromInt32(0x00000000); // cero
else
{
//a=expo(-Si3);
tem1=multiplica(FloatUnpackFromInt32(0xBF800000),Si
3);
tem2 = exp(tem1);
}
//>>>>>>>>>>>>>>>Xi3=1/(1+exp(-Si3));
tem1 = suma(uno,tem2);
Xi3 = divide(uno,tem1);
//>>>>>>>>>>>>>>>dFSi3=exp(-Si3)/((exp(-
Si3)+1)*(exp(-Si3)+1));
tem3 = multiplica(tem1,tem1);
dFSi3 = divide(tem2,tem3);
//j=1;
//>>>>>>>>>>>>>>>Wi1_O=Wi1_O_1+(Eta_I*(em*Xi1));
tem1 = multiplica(Eta_I,em);
tem2 = multiplica(tem1,Xi1);
Wi1_O = suma(Wi1_O_1,tem2);
//j=2;
//>>>>>>>>>>>>>>>Wi2_O=Wi2_O_1+(Eta_I*(em*Xi2));
tem1 = multiplica(Eta_I,em);
tem2 = multiplica(tem1,Xi2);
Wi2_O = suma(Wi2_O_1,tem2);
//j=3;
//>>>>>>>>>>>>>>>Wi3_O=Wi3_O_1+(Eta_I*(em*Xi3));
tem1 = multiplica(Eta_I,em);
tem2 = multiplica(tem1,Xi3);
Wi3_O = suma(Wi3_O_1,tem2);
//>>>>>>>>>>>>>>>Yu=(Wi1_O*dFSi1*Wi11_I)+(Wi2_
O*dFSi2*Wi12_I)+(Wi3_O*dFSi3*Wi13_I)+ajuste;
tem1 = multiplica(Wi1_O,dF
Si1);
tem2 = multiplica(tem1,Wi11_I);
tem1 = multiplica(Wi2_O,dFSi2);
tem3 = multiplica(tem1,Wi12_I);
tem1 = suma(tem2,tem3);
tem2 = multiplica(Wi3_O,dFSi3);
tem3 = multiplica(tem2,Wi13_I);
tem2= suma(tem1,tem3);
Yu=suma(tem2,ajuste);
//>>>>>>>Ym=(Wi1_O*Xi1)+(Wi2_O*Xi2)+(Wi3_O*Xi3);
tem1 = multiplica(Wi1_O,Xi1);
tem2 = multiplica(Wi2_O,Xi2);
tem3 = suma(tem1,tem2);
tem1 = multiplica(Wi3_O,Xi3);
Ym = suma(tem3,tem1);
// ACTUALIZACION DE VARIABLES
U_2=U_1;U_1=U;Y_2=Y_1;Y_1=Y;Yr_1=Yr;
Xc1_1=Xc1;Xc2_1=Xc2;Xc3_1=Xc3;Pc1_1=Pc1;
Pc2_1=Pc2;Pc3_1=Pc3;Wc1_O_1=Wc1_O;
Wc2_O_1=Wc2_O; Wc3_O_1=Wc3_O;
Wc11_I_1=Wc11_I;Wc21_I_1=Wc21_I;Wc31_I_1=Wc31
_I;Wc12_I_1=Wc12_I;Wc22_I_1=Wc22_I;
Wc32_I_1=Wc32_I;Wc13_I_1=Wc13_I;Wc23_I_1=Wc23
_I;Wc33_I_1=Wc33_I;Wc1_D_1=Wc1_D;
Wc2_D_1=Wc2_D;Wc3_D_1=Wc3_D;Qc11_1=Qc11;
Qc21_1=Qc21;Qc31_1=Qc31;Qc12_1=Qc12;
Qc22_1=Qc22;Qc32_1=Qc32;Qc13_1=Qc13;
Qc23_1=Qc23;Qc33_1=Qc33;Xi1_1=Xi1;
Xi2_1=Xi2;Xi3_1=Xi3;Pi1_1=Pi1;Pi2_1=Pi2;Pi3_1=Pi3;
Wi1_O_1=Wi1_O;Wi2_O_1=Wi2_O;Wi3_O_1=Wi3_O;
Wi11_I_1=Wi11_I;Wi21_I_1=Wi21_I;Wi31_I_1=Wi31_I;
Wi12_I_1=Wi12_I;Wi22_I_1=Wi22_I;Wi32_I_1=Wi32_I;
Wi13_I_1=Wi13_I;Wi23_I_1=Wi23_I;Wi33_I_1=Wi33_I;
Wi1_D_1=Wi1_D;Wi2_D_1=Wi2_D;Wi3_D_1=Wi3_D;
Qi11_1=Qi11;Qi21_1=Qi21;Qi31_1=Qi31;
Qi12_1=Qi12;Qi22_1=Qi22;Qi32_1=Qi32;
Qi13_1=Qi13;Qi23_1=Qi23;Qi33_1=Qi33;
}
}
```

APÉNDICE D

Programa de la RNDR en el DSC

```
#include <p30f6014a.h>
#include <stdio.h>
#include <adc10.h>
#include <math.h>

#define FCY 28800000 // MIPS = cristal*16/4 = 7.2*16
Mhz/ 4 = 28.8

#define   US_10       (1*FCY/100000)+1   //289
#define   US_25       (25*FCY/1000000)+1
#define   US_50       (5*FCY/100000)+1
#define   US_100       (1*FCY/10000)+1
#define   US_250       (25*FCY/100000)+1
#define   US_500       (5*FCY/10000)+1
#define   MS_10       (1*FCY/100)+1
#define   MS_25       (25*FCY/1000)+1
#define   MS_50       (5*FCY/100)+1
#define   MS_100       (1*FCY/10)+1
#define   MS_250       (25*FCY/100)+1
#define   MS_500       (5*FCY/10)+1

#include "LCD.h"
#include "dac12.h"

 _FOSC( CSW_FSCM_OFF  & XT_PLL16);
 _FWDT(WDT_OFF);
 _FBORPOR(PBOR_OFF & PWRT_OFF & MCLR_EN);
 _FGS(CODE_PROT_OFF);

void adc10_init()
{
        unsigned int PinConfig, Scanselect,
        Adcon3_reg, Adcon2_reg,Adcon1_reg;
        TRISBbits.TRISB0 = 1;
        TRISBbits.TRISB1 = 1;
        ADCON1bits.ADON = 0;
        ConfigIntADC10(ADC_INT_DISABLE);
        PinConfig           =
        ENABLE_AN0_ANA &
        ENABLE_AN1_ANA ;
        Scanselect          =
        SKIP_SCAN_AN2 &
        SKIP_SCAN_AN3 &
        SKIP_SCAN_AN4 &
        SKIP_SCAN_AN5 &
        SKIP_SCAN_AN6 &
        SKIP_SCAN_AN7 &
        SKIP_SCAN_AN8 ;
        Adcon3_reg          =
        ADC_SAMPLE_TIME_15 &
        ADC_CONV_CLK_SYSTEM &
        ADC_CONV_CLK_15Tcy;
        Adcon2_reg          =
        ADC_VREF_AVDD_AVSS &
        ADC_SCAN_ON &
        ADC_ALT_BUF_OFF &
        ADC_ALT_INPUT_OFF &
        ADC_SAMPLES_PER_INT_16 &
        ADC_CONVERT_CH0;
        Adcon1_reg          =
        ADC_MODULE_ON &
        ADC_IDLE_STOP &
        ADC_FORMAT_INTG &
        ADC_CLK_AUTO &
        ADC_AUTO_SAMPLING_ON &
        ADC_SAMPLE_INDIVIDUAL &
        ADC_SAMP_ON;
        OpenADC10(Adcon1_reg,
        Adcon2_reg,Adcon3_reg,PinConfig,
        Scanselect);
}

unsigned int read_adc(int channel)
{
        unsigned int Channe0 =
        ADC_CH0_POS_SAMPLEA_AN0 &
        ADC_CH0_NEG_SAMPLEA_NVREF;
        unsigned int Channe1 =
        ADC_CH0_POS_SAMPLEA_AN1 &
        ADC_CH0_NEG_SAMPLEA_NVREF;
         if(channel==0)
                SetChanADC10(Channe0);
        if(channel==1)
                SetChanADC10(Channe1);
                ADCON1bits.SAMP = 1;
        while(!ADCON1bits.DONE);
        ADCON1bits.DONE=0;
        return(ReadADC10(channel));
}

int main ()
{
   char sbuf1[16];
   float x=0;
   unsigned int a=0,b=0;
    float  Y=0, Y_1=0, Y_2=0, Ym=0, Yu=0, U=0, U_1=0,
U_2=0,  r=0,  Yr=0,  Yr_1=0,  ec=0,  em=0,  dFSc1=0,
dFSc2=0,dFSc3=0,Sc1=0,Sc2=0,Sc3=0,dFSi1=0,
dFSi2=0,dFSi3=0,Si1=0,Si2=0,Si3=0,bias=1,
Eta_C=0.17,Eta_I=0.00001,ajuste=0.15,Xc1=0,Xc1_1=0
,Xc2=0,Xc2_1=0,Xc3=0,Xc3_1=0,Pc1=0,Pc1_1=0,
Pc2=0,Pc2_1=0,Pc3=0,Pc3_1=0,Wc1_O=1,Wc1_O_1=0
,Wc2_O=1,Wc2_O_1=0,Wc3_O=1,Wc3_O_1=0,
Wc11_I=1,Wc11_I_1=0,Wc21_I=1,Wc21_I_1=0,
Wc31_I=1,Wc31_I_1=0,Wc12_I=1,Wc12_I_1=0,
Wc22_I=1,Wc22_I_1=0,Wc32_I=1,Wc32_I_1=0,
Wc13_I=1,Wc13_I_1=0,Wc23_I=1,Wc23_I_1=0,
Wc33_I=1,Wc33_I_1=0,Wc1_D=1,Wc1_D_1=0,
Wc2_D=1,Wc2_D_1=0,Wc3_D=1,Wc3_D_1=0,
Qc11=0,Qc11_1=0,Qc21=0,Qc21_1=0,
Qc31=0,Qc31_1=0,Qc12=0,Qc12_1=0,
Qc22=0,Qc22_1=0,Qc32=0,Qc32_1=0,
Qc13=0,Qc13_1=0,Qc23=0,Qc23_1=0,
Qc33=0,Qc33_1=0,Qi11=0,Qi11_1=0,Qi21=0,Qi21_1=0,
Qi31=0,Qi31_1=0,Qi12=0,Qi12_1=0,Qi22=0,Qi22_1=0,
Qi32=0,Qi32_1=0,Qi13=0,Qi13_1=0,Qi23=0,Qi23_1=0,
Qi33=0,Qi33_1=0,Pi1=0,Pi1_1=0,Pi2=0,Pi2_1=0,
Pi3=0,Pi3_1=0,Xi1=0,Xi1_1=0,Xi2=0,Xi2_1=0,
Xi3=0,Xi3_1=0,Wi1_O=1,Wi1_O_1=0,  Wi2_O=1,
Wi2_O_1=0,Wi3_O=1,Wi3_O_1=0,Wi11_I=1,
Wi11_I_1=0,Wi21_I=1,Wi21_I_1=0,Wi31_I=1,
Wi31_I_1=0,Wi12_I=1,Wi12_I_1=0,Wi22_I=1,
Wi22_I_1=0,Wi32_I=1,Wi32_I_1=0,Wi13_I=1,
Wi13_I_1=0,Wi23_I=1,Wi23_I_1=0,Wi33_I=1,
Wi33_I_1=0,Wi1_D=1,Wi1_D_1=0,Wi2_D=1,
Wi2_D_1=0,Wi3_D=1,Wi3_D_1=0;

lcd_init();
lcd_clear();
adc10_init();
dac12_init();
while(1)
 {
    a = read_adc(0);
    x=a*5.0/1023.0;
    b = read_adc(1);
    Y=b*5.0/1023.0;
     sprintf(sbuf1, "%1.2f %1.2f %1.2f",Sc1,Sc2,Sc3);
     lcd_puts(LINE1,sbuf1);
     sprintf(sbuf1, "%1.2f %1.2f %1.2f",ec,Wc1_O,Qc31);
     lcd_puts(LINE2,sbuf1);
    __delay32(MS_100);
    Yr=2.0;
```

```
    r=2.0;
    ec=Yr-Y;
    em=Y-Ym;
    //PARTE DEL NEUROCONTROLADOR
    //j=1;

Wc11_I=Wc11_I_1+Eta_C*(ec*Yu*Wc1_O*Qc11);
Wc21_I=Wc21_I_1+Eta_C*(ec*Yu*Wc1_O*Qc21);
Wc31_I=Wc31_I_1+Eta_C*(ec*Yu*Wc1_O*Qc31);
Qc11=dFSc1*(r+Wc1_D*Qc11_1);
Qc21=dFSc1*(U_1+Wc1_D*Qc21_1);
Qc31=dFSc1*(Y_1+Wc1_D*Qc31_1);
//j=2;
Wc12_I=Wc12_I_1+Eta_C*(ec*Yu*Wc2_O*Qc12);
Wc22_I=Wc22_I_1+Eta_C*(ec*Yu*Wc2_O*Qc22);
Wc32_I=Wc32_I_1+Eta_C*(ec*Yu*Wc2_O*Qc32);
Qc12=dFSc2*(r+Wc2_D*Qc12_1);
Qc22=dFSc2*(U_1+Wc2_D*Qc22_1);
Qc32=dFSc2*(Y_1+Wc2_D*Qc32_1);
//j=3;
Wc13_I=Wc13_I_1+Eta_C*(ec*Yu*Wc3_O*Qc13);
Wc23_I=Wc23_I_1+Eta_C*(ec*Yu*Wc3_O*Qc23);
Wc33_I=Wc33_I_1+Eta_C*(ec*Yu*Wc3_O*Qc33);
Qc13=dFSc3*(r+Wc3_D*Qc13_1);
Qc23=dFSc3*(U_1+Wc3_D*Qc23_1);
Qc33=dFSc3*(Y_1+Wc3_D*Qc33_1);
//j=1;
Wc1_D=Wc1_D_1+Eta_C*(ec*Yu*Wc1_O*Pc1);
Pc1=dFSc1*(Xc1_1+Wc1_D*Pc1_1);
//j=2;
Wc2_D=Wc2_D_1+Eta_C*(ec*Yu*Wc2_O*Pc2);
Pc2=dFSc2*(Xc2_1+Wc2_D*Pc2_1);
//j=3;
Wc3_D=Wc3_D_1+Eta_C*(ec*Yu*Wc3_O*Pc3);
Pc3=dFSc3*(Xc3_1+Wc3_D*Pc3_1);
//j=1
Sc1=Wc1_D*Xc1_1+(Wc11_I*r)+(Wc21_I*U_1)+(Wc31_
I*Y_1);
Xc1=1/(1+exp(-Sc1));
dFSc1=exp(-Sc1)/((exp(-Sc1)+1)*(exp(-Sc1)+1));
//j=2
Sc2=Wc2_D*Xc2_1+(Wc12_I*r)+(Wc22_I*U_1)+(Wc32_
I*Y_1);
Xc2=1/(1+exp(-Sc2));
dFSc2=exp(-Sc2)/((exp(-Sc2)+1)*(exp(-Sc2)+1));
//j=3
Sc3=Wc3_D*Xc3_1+(Wc13_I*r)+(Wc23_I*U_1)+(Wc33_
I*Y_1);
Xc3=1/(1+exp(-Sc3));
dFSc3=exp(-Sc3)/((exp(-Sc3)+1)*(exp(-Sc3)+1));
//j=1
Wc1_O=Wc1_O_1+Eta_C*(ec*Yu*Xc1);
//j=2;
Wc2_O=Wc2_O_1+Eta_C*(ec*Yu*Xc2);
//j=3;
Wc3_O=Wc3_O_1+Eta_C*(ec*Yu*Xc3);
U=Wc1_O*Xc1+Wc2_O*Xc2+Wc3_O*Xc3
if(U>5.0)
        U=5.0;
if(U<0)
        U=0;
write_dac12((4095.0*U)/5.0);
//PARTE DEL NEUROIDENTIFICADOR
//j=1;
Wi11_I=Wi11_I_1+Eta_I*(em*Wi1_O*Qi11);
Wi21_I=Wi21_I_1+Eta_I*(em*Wi1_O*Qi21);
Wi31_I=Wi31_I_1+Eta_I*(em*Wi1_O*Qi31);
Qi11=dFSi1*(U+Wi1_D*Qi11_1);
Qi21=dFSi1*(Y_1+Wi1_D*Qi21_1);
Qi31=dFSi1*(bias+Wi1_D*Qi31_1);
//j=2;
Wi12_I=Wi12_I_1+Eta_I*(em*Wi2_O*Qi12);
Wi22_I=Wi22_I_1+Eta_I*(em*Wi2_O*Qi22);
Wi32_I=Wi32_I_1+Eta_I*(em*Wi2_O*Qi32);
Qi12=dFSi2*(U+Wi2_D*Qi12_1);
Qi22=dFSi2*(Y_1+Wi2_D*Qi22_1);
Qi32=dFSi2*(bias+Wi2_D*Qi32_1);
//j=3;
Wi13_I=Wi13_I_1+Eta_I*(em*Wi3_O*Qi13);
Wi23_I=Wi23_I_1+Eta_I*(em*Wi3_O*Qi23);
Wi33_I=Wi33_I_1+Eta_I*(em*Wi3_O*Qi33);
Qi13=dFSi3*(U+Wi3_D*Qi13_1);
Qi23=dFSi3*(Y_1+Wi3_D*Qi23_1);
Qi33=dFSi3*(bias+Wi3_D*Qi33_1);
//j=1;
Wi1_D=Wi1_D_1+Eta_I*(em*Wi1_O*Pi1);
Pi1=dFSi1*(Xi1_1+Wi1_D*Pi1_1);
//j=2;
Wi2_D=Wi2_D_1+Eta_I*(em*Wi2_O*Pi2);
Pi2=dFSi2*(Xi2_1+Wi2_D*Pi2_1);
//j=3;
Wi3_D=Wi3_D_1+Eta_I*(em*Wi3_O*Pi3);
Pi3=dFSi3*(Xi3_1+Wi3_D*Pi3_1);
//j=1
Si1=Wi1_D*Xi1_1+(Wi11_I*U)+(Wi21_I*Y_1)+(Wi31_I*bi
as);
Xi1=1/(1+exp(-Si1));
dFSi1=exp(-Si1)/((exp(-Si1)+1)*(exp(-Si1)+1));
//j=2
Si2=Wi2_D*Xi2_1+(Wi12_I*U)+(Wi22_I*Y_1)+(Wi32_I*bi
as);
Xi2=1/(1+exp(-Si2));
dFSi2=exp(-Si2)/((exp(-Si2)+1)*(exp(-Si2)+1));
//j=3
Si3=Wi3_D*Xi3_1+(Wi13_I*U)+(Wi23_I*Y_1)+(Wi33_I*bi
as);
Xi3=1/(1+exp(-Si3));
dFSi3=exp(-Si3)/((exp(-Si3)+1)*(exp(-Si3)+1));
//j=1;
Wi1_O=Wi1_O_1+Eta_I*(em*Xi1);
//j=2;
Wi2_O=Wi2_O_1+Eta_I*(em*Xi2);
//j=3;
Wi3_O=Wi3_O_1+Eta_I*(em*Xi3);
Yu=(Wi1_O*dFSi1*Wi11_I)+(Wi2_O*dFSi2*Wi12_I)+(Wi
3_O*dFSi3*Wi13_I)+ajuste;
Ym=Wi1_O*Xi1+Wi2_O*Xi2+Wi3_O*Xi3
// ACTUALIZACION DE VARIABLES
U_2=U_1; U_1=U; Y_2=Y_1; Y_1=Y; Yr_1=Yr;
Xc1_1=Xc1;Xc2_1=Xc2;Xc3_1=Xc3;
Pc1_1=Pc1;Pc2_1=Pc2;Pc3_1=Pc3;
Wc1_O_1=Wc1_O;Wc2_O_1=Wc2_O;Wc3_O_1=Wc3_
O;
Wc11_I_1=Wc11_I;Wc21_I_1=Wc21_I;Wc31_I_1=Wc31
_I;
Wc12_I_1=Wc12_I;Wc22_I_1=Wc22_I;Wc32_I_1=Wc32
_I;
Wc13_I_1=Wc13_I;Wc23_I_1=Wc23_I;Wc33_I_1=Wc33
_I;
Wc1_D_1=Wc1_D;Wc2_D_1=Wc2_D;Wc3_D_1=Wc3_
D;
Qc11_1=Qc11;Qc21_1=Qc21;Qc31_1=Qc31;
Qc12_1=Qc12;Qc22_1=Qc22;Qc32_1=Qc32;
Qc13_1=Qc13;Qc23_1=Qc23;Qc33_1=Qc33;
Xi1_1=Xi1;Xi2_1=Xi2;Xi3_1=Xi3;
Pi1_1=Pi1;Pi2_1=Pi2;Pi3_1=Pi3;
Wi1_O_1=Wi1_O;Wi2_O_1=Wi2_O;Wi3_O_1=Wi3_O;
Wi11_I_1=Wi11_I;Wi21_I_1=Wi21_I;Wi31_I_1=Wi31_I;
Wi12_I_1=Wi12_I;Wi22_I_1=Wi22_I;Wi32_I_1=Wi32_I;
Wi13_I_1=Wi13_I;Wi23_I_1=Wi23_I;Wi33_I_1=Wi33_I;
Wi1_D_1=Wi1_D;Wi2_D_1=Wi2_D;Wi3_D_1=Wi3_D;
Qi11_1=Qi11;Qi21_1=Qi21;Qi31_1=Qi31;
Qi12_1=Qi12;Qi22_1=Qi22;Qi32_1=Qi32;
Qi13_1=Qi13;Qi23_1=Qi23;Qi33_1=Qi33;
    }
return 0;
}
```

APÉNDICE E

Constancia de participación:

ELECTRO 2007 (Instituto Tecnológico de Chihuahua)

ITCH - ELECTRO 2007 *Octubre 17-19, Chihuahua, México*

REDES NEURONALES DIAGONALES RECURRENTES PARA SISTEMAS DINÁMICOS DE CONTROL

Silva Rodríguez Carlos Alberto, Noriega Ponce Alfonso
Universidad Autónoma de Querétaro.
Cerro de las campanas s/n Centro Universitario
C.P. 76010 Querétaro, Qro. (442)192-12-00 Ext. 6023
microcarsil@hotmail.com , anoriega@uaq.mx

RESUMEN

Un nuevo algoritmo basado en redes neuronales diagonales recurrentes (DRNN) es presentado. La arquitectura de la DRNN es un modelo modificado de la completa red neuronal recurrente conectada a una capa escondida [1] dicha capa está compuesta por neuronas que se recurren a sí mismas. Dos DRNN's son utilizadas en el sistema de control, una es la encargada de la identificación llamada neuroidentificador diagonal recurrente (DRNI) y la otra es un controlador llamado neurocontrolador diagonal recurrente (DRNC). Una planta es identificada por la DRNI la cual provee información de sensibilidad hacia el DRNC. Un algoritmo dinámico de retropropagación (BP) es desarrollado para entrenar tanto a DRNI como a DRNC. A partir de la recurrencia la DRNN puede capturar el comportamiento dinámico del sistema. El paradigma propuesto de la DRNN esta aplicado a los problemas numéricos y los resultados de simulación para el control de posición de un motor de DC.

1. INTRODUCCIÓN

Los desarrollos en el área de control han sido alimentados por tres necesidades muy importantes: la necesidad de tratar con sistemas cada vez más complejos, la necesidad de cumplir con los requerimientos de diseño cada vez más exigentes y la necesidad de lograr estos requerimientos de diseño con el menor conocimiento de la planta y su ambiente [2], [3]. Los sistemas dinámicos son cada vez más complejos y obligan a los diseñadores e ingenieros a renunciar a los métodos de control convencionales, sin embargo, las técnicas de control adaptativo así como las leyes del control no lineal ofrecen soluciones a estos problemas pero con la contraparte de que incrementan su complejidad en el desarrollo e implantación que se pueda llevar a cabo, en general para aplicaciones en tiempo real se buscan nuevas soluciones que ayuden a disminuir estos problemas de diseño de control para una planta dinámica complicada y que ofrezcan robustez, confiabilidad y sobre todo que se mantengan los requerimientos de diseño[4]. Varios modelos de redes neuronales y esquemas de aprendizaje han sido aplicadas como diseños de controladores durante las últimas tres décadas dando resultados satisfactorios para diferentes problemas dinámicos que se han reportado [4]-[8], la mayoría de las personas usan redes neurales de retroalimentación con algoritmos de entrenamiento retropropagables [4], [9] para solucionar problemas dinámicos, sin embargo la red de retroalimentación es estática. Por otro lado las redes neuronales recurrentes [1]-[2] y [7]-[11] tienen capacidades importantes no encontradas en las retroalimentadas estáticas como es la evolución de la red dinámica y la habilidad de almacenar la información para uso posterior. Por lo tanto la red neuronal recurrente se adecua mejor a sistemas dinámicos que la red neuronal retroalimentada. En la mayoría de las aplicaciones en control la implementación en tiempo real es muy importante, y por tanto el neurocontrolador también tiene que ser diseñado de modo que converja con un número pequeño de ciclos de entrenamiento. Este documento está organizado de la siguiente manera: En la sección II un modelo de DRNN es desarrollado, en la sección III un algoritmo de entrenamiento de retropropagación dinámico es diseñado para entrenar un sistema de control DRNN y en la sección IV la relevancia práctica de las estructuras de control propuestas es ilustrada por la simulación para el control de posición de un motor de CD.

2. MODELO DRNN

Considere la figura 1 donde para cada tiempo discreto k, $I_i(k)$ es la i-ésima entrada, $S_j(k)$ es la suma de las entradas de la j-ésima neurona recurrente, $X_j(k)$ es la salida de la j-ésima neurona recurrente, O(k) es la salida de la red. W^I, W^O, W^D


ELECTRO
2007
VOLUMEN XXIX
ISSN 1405-2172
50 años
Pioneros de la educación tecnológica superior en prónicas
áreas
Técnicas de Procesamiento de Señales
Procesamiento Digital de Señales Neurodifuso
Control Inteligente y Robótica
Sistemas Digitales
Diseño de Sistemas y Dispositivos de Comunicaciones
Implementaciones de Procesamiento Digital en Hardware
Diseño de Control
Aplicaciones de Computación
Automatización de Equipos y Análisis de Sistemas de Potencia
Análisis de Sistemas y Dispositivos de Comunicaciones
Visión por Computadora
Instrumentación
Inteligencia Artificial
Diseño de Sistemas de Potencia
Automatización de Procesos
INSTITUTO TECNOLÓGICO DE CHIHUAHUA
DIMISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN
COMITE ORGANIZADOR ELECTRO 2007
AV. TECNOLÓGICO 2909 CHIHUAHUA, CHIH., MEXICO
TEL. 52 (614) 413 7474 EXT. 112 y 114
EMAIL: electro@itchihuahua.edu.mx
WEB: http://electro.itch.edu.mx/

APÉNDICE F

Constancia de participación:

II Magno Congreso Internacional de Computación CIC-IPN

INSTITUTO POLITÉCNICO NACIONAL

CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN

Otorgan la presente:

Constancia

Carlos Silva

Por su participación como

PONENTE

Con el tema:

Diagonal Recurrent Neural Networks for Speed Control of a DC motor.

EN EL II MAGNO CONGRESO INTERNACIONAL DE COMPUTACIÓN CIC-IPN

Celebrado en México, D.F., del 6 al 8 de Noviembre de 2007.

"La Técnica al Servicio de la Patria"

Dr. Juan Humberto Sossa Azuela

Presidente del Comité Organizador

Diagonal Recurrent Neural Networks for Speed Control of a DC motor

Alfonso Noriega[1], Carlos A. Silva[1] and Jesús Pichardo[2]

[1] Facultad de Ingenieria, Universidad Autónoma de Querétaro, Querétaro, Qro., México
[2] CICATA-IPN, Querétaro, Qro., México
{Alfonso Noriega, anoriega@uaq.mx}

Abstract. A new control algorithm based on diagonal recurrent neural network (DRNN) is presented. The architecture of DRNN is a modified model of the fully connected recurrent neural network with one hidden layer [1], and the hidden layer is comprised of self-recurrent neurons. Two DRNN's are utilized in a control system, one as an identifier called diagonal recurrent neuroidentifier (DRNI) and the other as a controller called diagonal recurrent neurocontroller (DRNC). A controlled plant is identified by the DRNI which then provides the sensitivity information of the plant to the DRNC. A generalized dynamic backpropagation algorithm (DBP) is developed and used to train both DRNC and DRNI. Due the recurrence, the DRNN can capture the dynamic behavior of the system. The proposed DRNN paradigm is applied to numerical problems and the simulation results for speed control of a DC motor are included.

Keywords: Recurrent Neural Networks, Dynamic Back-propagation, DC Motor, Diagonal Recurrent Neural Networks, Adaptive Control.

I Introduction

The development in the control area has been fueled by three major needs: the need to deal with increasingly complex systems, the need to accomplish increasingly demanding design requirements, and the need to attain these requirements with less precise advanced knowledge of the plant and its environment [2], [3]. Increasingly complex dynamical systems with significant uncertainty have forced system designers to turn away from conventional control methods. However, the fundamental shortcomings of current adaptive control techniques, such as nonlinear control laws which are difficult to derive, geometrically increasing complexity with the number of unknown parameters, and the general unsuitability for real time applications have compelled researchers to look for solutions elsewhere [4].

Several neural network models and neural learning schemes were applied to system controller design during the last three decades, and many promising result are reported [4]-[8]. Most people used the feedforward neural network and the backpropagation training algorithm [4], [9] to solve the dynamical problems; however, the feedforward network is a static mapping. On the other hand, recurrent neural networks [1]-[2] and [7]-[11] have important capabilities not found in feedforward network, such as attractor dynamics and the ability to store information

Industrial Informatics
Research in Computing Science 31, 2007, pp. 143-152

APÉNDICE G

Constancia de participación:

4to Congreso Internacional de Ingeniería